La Totale – Introduction

Document créé par Codaline Bourotte, compilant tous les exercices d'informatique que je connais et que je trouve intéressants et pédagogiques. Ce doc vise surtout les profs de prépa qui veulent créer une feuille de TD ou préparer des colles, les agrégatifs, normaliens, chercheurs ou étudiants qui veulent découvrir / s'entraîner, ou encore les préparationnaires qui veulent aller plus loin.

Les exos sont faits pour être indépendants les uns des autres, sauf mention contraire. Ils sont plutôt durs: n'hésitez pas à rajouter des questions intermédiaires si vous voulez les donner à des élèves. L'idée, c'est qu'ils correspondent à peu près à ce qu'on pourrait faire en une bonne colle, même si certains sont plutôt de l'ordre d'une partie d'un sujet d'écrit. J'essaie autant que possible de donner un lien vers l'inspiration bibliographiques de l'exercice quand je l'ai ; ça permet souvent d'avoir un autre point de vue, mais ce n'est pas toujours facile à retrouver.

Ce document est bourré de fautes d'orthographe, et probablement de quelques fautes de logique. À la base, c'est un document très personnel et assez brouillon, mais il peut intéresser plus d'une personne.

N'hésitez pas à m'envoyer d'autres exercices, des corrections, ou des demandes de corrigés pour certains exos. Normalement, je sais tous les faire, et je les rescanne de temps en temps. Vous pouvez me contacter à la-totale@bourotte.com.

Lien vers le githug du document: https://github.com/Cypooos/la-totale. Dans le futur, je bougerai en dehors de github pour des raisons étique. N'hésitez pas à faire des push request avec d'autres exercices !

Certaines inspirations bibliographique utilisées:

- Le livre de Galatée Hemery (mon ancienne prof de prépa !) et Lou Chalmain: https://www.fnac.com/a22502218/Lou-Chalmain-Informatique-MPI-MPI
- Anciens oraux d'Ulm: https://a3nm.net/work/exams/ens/
- Des super exos de spé avec correction: https://anthonylick.com/wp-content/uploads/MPI_exos_xens_corrige.pdf
- Le Requin (certains sont vraiment bien)
- Le PDF des colles donné par Mallory Marin au Lycée du Parc pour la 1ère promo de MP2I/MPI
- Le PDF de la totale de tous les exercices à la Martinière par Quentin Fortier
- Les exercices du TD de Maxime Bridoux à la prépa Fénelon Sainte-Marie
- Des sujets d'oraux et de concours de chaque année
- Des sujets d'agrég
- Des sujets provenant de mes cours au MPRI ou à l'ENS de Lyon
- Mes sujets d'Info D
- Des colles données par José en informatique à Fénelon Sainte-Marie
- Le Putnam

Contents

La Totale – Introduction 1
MP2I: Programmation impérative, représentation machine 10
Cours 10
Complexité 10
MP2I: Programmation fonctionnelle 10
Cours 10
Mémoisation Automatique[1] 11

[1] de Maxime BRIDOUX

Suspension et listes infinies 11
Continuation 12
Binary random acces lists 13
Monades 14
Parsing avec des monades[2] 15
MP2I: Structures de données séquentielles 17
Cours 17
Complexité des opérations de bases 17
Merge de liste 17
Calcul des différences 17
Permutation suivante 17
Tableaux binaire 17
Sous-tableau connexe 18
Le glouton par défault[3] 18
Recherche dans une matrice 18
Tableau cumulatif 18
Multiplication rapide de polynome[4] 19
Structure efficace pour représenter des ensembles 19
Inverser une liste sans toucher aux pointeurs 19
Liste cyclique sans lièvre et torture 20
Tableaux auto-référents 20
MP2I: Inductions & ordres[5] 21
Cours 21
Des ordres 21
Des ensembles inductifs 21
Soustraction entière 21
Ordre sur les mots 21
Tri aléatoire 22
Langage de Dyck 22
Système MIU 22
Sous-ordres indénombrable de $(\mathcal{P}(\mathbb{N}), \subseteq)$ 22
MP2I: Arbres & structures inductives 23
Cours 23
Nombre de sommets 23
Plus de feuilles 23
Contour d'un arbre 23
Arbres d'intervalles 24
Minimum locaux 24
Accès dans un arbre parfait 24
Arbres 2-dimensionnels 25
Reconstruire l'arbre avec des parcours 25
Arbre canonique[6] 26
Préfixe vers Suffixe 26
Arbres généraux 26

[2] MPRI - FUN Exam 2026
[3] Algo 1 ENS Lyon
[4] Algo 1 ENS Lyon
[5] Maxime Bridoux, MP2I
[6] ENS 2023 MPI Info A

Tableaux tri-coloré 27
Permutation triable par pile[7] 27
Initialisation d'un Tas 29
Mots d'arbres 29
Ranger des boîtes 30
Parcours préfixe en $O(1)$ mémoire 30
Parcours infixe en $O(1)$ mémoire 30
Arbre et oracle 31
Indépendant dans les arbres 32
Arbre α-équilibré 32
MP2I: Graphes (introduction) 33
Cours 33
Autour des degrés 33
Base des graphes connexes 34
Le puits et le roi 34
Graphes à unique sortie 34
Reconnaitre les graphes biconnecté en $O(|V| + |E|)$ 34
Algorithme de Johnson[8] 35
Reconnaitre un DFS 35
Conversions de DAG 35
Chemin hamiltonien dans un tournois 36
Degrée minimal[9] 36
Graphes d'Halin[10] sont 3-connexes 37
Graphes d'Halin[11] sont hamiltoniens 37
Nombre cyclomatiques[12] 37
Calcul des triangles 37
Graphes orienté semi-connexes[13] 38
Graphes Cordaux 38
Codage de Prufer[14] 38
MP2I: Logique Propositionelle 39
Cours 39
Equivalence 40
Retirer les constantes 40
Vers le 3-Fnc 40
Push down not 40
Systèmes Complet de Connecteurs[15] 41
Formule pour la bi-partition 41
Formule pour le centre d'un graphe 41
Formules linéaire 41

[7] Oral blanc MPI CCINP

[8] Mines Télécom 2024

[9] CCINP sujet 0

[10] On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

[11] On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

[12] Tiré du TD de la martinière de MPI

[13] Tiré du TD de la martinière de MPI

[14] Tiré du TD de la martinière de MPI

[15] Exercice de Maxime Bridoux

Equivalences avec Sheffer 42
Equivalence avec ite[16] 42
Indéterminé 42
Formules monotones 43
Transformation de Tseitin 43
Formules duales 44
XOR SAT 44
Interpolation de Craig[17] 44
FNC-SAT vers CLIQUE 44
Binary Decision Diagram 45
Horn-SAT 46
Formules Evasives[18] 46
Compacité 47
MP2I: Bases de Données 48
Cours 48
Films et notes[19] 48
Des étudiants et des notes[20] 48
Matières difficiles 49
Ensembles Egaux 49
La limite de SQL 49
Topologie d'une montagne[21] 50
MP2I: Algorithmique 51
Cours 51
Merge de liste 51
Nombre d'inversions dans une liste 51
Meilleur chemin dans une grille 52
Points les plus proches 52
Sous-tableau connexe 52
Le glouton par défault[22] 52
SUBSET-SUM 52
Elements Majoritaire 53
Disco-world[23] 53
Recherche dans une matrice 54
Multiplication rapide de polynome[24] 54
Casser des oeufs 54
Permutations de Langford[25] 55
Tableaux auto-référents 55
MPI: Langages 56
Cours 56
Opérations de langages 56

[16] Exercice de Maxime Bridoux
[17] Un mix d'un exercice de Maxime Bridoux et d'un exercice du cours de logique de l'ENS de Lyon
[18] Sujet d'ENS
[19] Mallory Marin
[20] Mallory Marin
[21] Inspiré du concours Général d'informatique 2022
[22] Algo 1 ENS Lyon
[23] Cours de programmation compétitive ENS Lyon
[24] Algo 1 ENS Lyon
[25] CCINP 2025

Language contenant ε 56
Cloture de langages 56
Suite de mots par récurrence 57
Mot circulaire 57
Hypercubes et mots 57
Mots univers 57
Ensemble inévitables[26] 58
Mots sans cube[27] 58
Facteurs de points fixe de morphismes[28] 58
Langage sans étoile 59
(∗) Egalité pour les résiduels 59
(∗) Lemme d'Higman 59
MPI: Langages réguliers et Automates 59
Cours 59
Berry-Sethi 59
Langages réguliers 59
Non régulier 60
Petites questions 60
Cloture interne par préfixe 60
Factorisation arbitraire de l'étoile 60
Mots ordonnés[29] 61
Automates à sauts[30] 61
Automates avec couts (algo A^*, sujet bof) 61
Langage croisé 61
Sous-langage palindromique 62
Langage régulier à partir de l'automate[31] 62
Langage rotationel 62
Langage entrelacé 63
Mots univers 63
Astucieux langage[32] 63
Régulier à une lettre 64
Permuté 64
Langages surprenants 64
Automates de Büchi[33] 64
Automates partiellement ordonnés[34] 65
Commuteurs de Langages 65
Langage résiduel & langages régulier 65
Commutant de Langage[35] 65
Langage coupé au millieu 66
Langages fins 66

[26] oral info Ulm ENS 2019
[27] Oral de l'ENS Info 2019 et sujet de CCINP d'Informatique 2026
[28] Tiré de l'Info D 2026 de décembre
[29] CCINP 2024?
[30] Oral Mines-Télécom 2024, https://beos.prepas.org/sujet.php~?id=8151
[31] Colle de José
[32] Le fabuleux cours de théorie des langages formels d'Olivier Carton
[33] Mines-Télécom 2024 https://beos.prepas.org/sujet.php~?id=8156
[34] Oral ENS 2022 C2 https://diplome.di.ens.fr/informatique-ens/annales/2022_InfoU-exercices.pdf
[35] Sujet de José

Langages épars[36] 66
Langages rationel infinis 66
Racine de langage[37] 67
Arithmétique de Presburger (+ déduction) 67
Cloture par sur-mot[38] 67
Automates (n, d)-locaux $(*)$ 68
Langages continuables et mots primitifs 68
MPI: Langages Hors-contexte 68
Cours 68
Langage hors-contexte 69
Paires différentes 69
Autant de lettres 69
Forme normale de Chompsky 69
Morphisme de langage hors-contexte 69
Lemme d'itération pour les langages hors-contexte 70
Automates à pile 70
Mélange et grammaires 71
Circular shift 71
Théorème de Chomsky-Schützenberger[39] 72
Image de Parikh de langage[40] 72
MPI: Graphes avancé 73
Cours 73
Graphes à unique sortie 74
Kruskal en recherche locale 74
Graphes k-réguliers 74
Théorème de Hall 74
Maille d'un graphe 75
Largeur de bande 75
Graphes planaires 76
Couverture d'aretes et couplages 76
C-approx du couplages maximum[41] 76
Calcul de triangles[42] 77
Planaire et degré 77
Graphes d'amis sans C_5 78
Graphes d'amis 2 78
Un Graphe dénombrable 78
Graphes d-dégénéré[43] 78
Coeur de k-domination[44] 78
Théorème de vizig[45] 79
FPT: Kernelisation de Feedback vertex set 80

[36] ENS Ulm oral d'info, je ne sais plus de quand
[37] Oral de l'X
[38] Oral ENS Ulm, je ne sais plus quand
[39] Exos de la martinière MPI
[40] Tiré d'une preuve simplifié https://arxiv.org/pdf/2301.00047
[41] TD11 L3 ENS Lyon Algo 1
[42] Tiré de Mallory Marin
[43] InfoFonda 2026 partie I
[44] InfoFonda 2026 partie IV
[45] InfoFonda 2026 partie V

FPT: Kernelisation de Vertex cover 80
FPT: Kernelisation de Edge clique cover 81
MPI: Théorie des jeux 81
Cours 81
Jeu de Chomp 81
Jeu et graphes biparti[46] 82
Jeu de nim généralisé[47] 82
Géographie dans les hypercubes[48] 83
Géographie[49] 83
Exercice 2 : Jeux et automates [50] 83
Jeu de Shannon 84
Parité du nombre de points fixes[51] 84
Jeu du sous-mot[52] 84
MPI: Apprentissage 85
Cours 85
Single-pass (CCINP 2024)[53] 85
Clustering en dimension 1[54] 86
k-centres 86
Algorithme ID3 avec intervalles 87
MPI: Système, Mutex & Sémaphore 87
Cours 87
Lecteurs-Rédacteurs 87
Votes en parralèlles 88
Salle de spectacle 88
Sémaphore pondéré 88
Mutex ré-entrant 89
Pont à voie unique 89
MPI: Proba & Approx 90
Cours 90
Espérence pour tout choisir 90
Le programme de Von Neumann 90
2-approximation du circuit eulérien (TSP) 91
$3/2$-approx de TSP 91
C-approx du couplages maximum[55] 91
Vertex cover 92
Set cover 92
2-approx de Bin-Packing[56] 93
$\log(n)$-approx de MLST 93
Arbres ternaires complets (+Jeux)[57] 94

[46] CCINP Sujet A 2025
[47] Sujet original !
[48] Le putnam de 2025
[49] Marathon de math de Paris-Orsay 2021
[50] Sujet fait main!
[51] Putnam 2023
[52] Sujet original !
[53] https://prepas-mp2i.fr/documents/sujets/2024/CCINP-INFO.pdf
[54] tiré de https://mpi-lamartin.github.io/mpi-info/docs/ia/non_supervise/td_apprentissage
[55] TD11 L3 ENS Lyon Algo 1
[56] CCINP 2025

Unique graphe infini aléatoire 94
$(\Delta + 1)$-approx de Maximum independant set 95
Coloration aléatoire 95
K-Centres 96
Paire de points 96
MPI: Déduction naturelle 96
Cours 96
Arbres à faire 96
Arbres des lois de morgan 97
Arbre des lois de morgan avec quantificateurs 97
Equivalence entre les formules donnant la logique classique 97
Arbres nécésitant de deviner 97
Affaiblissement 97
XOR 97
Equivalence 97
L'opérateur de Sheffer 98
Sans implication 98
Complétude[58] 98
Sans négation 98
Théorie déductive des graphes 99
Formule Duale 99
Logique pertinente[59] 99
Introduction à la logique linéaire multiplicative 100
Correspondance du système à la Hilbert[60] 101
$\neg\neg$-Traduction de Godel-Kolmogorov[61] 102
Logique multiplicative 102
Modèles de Kripke 102
Logique modale[62] 103
Logique linéaire temporelle & automates de Buchis (sujet d'écrit) 104
MPI: Classes de Complexités 106
Cours 106
Max 2-SAT 106
SUBSET-SUM 106
INDEPENDANT-SET et CLIQUE 107
MIN-COLOR-PATH 107
DOMINATING-SET 108
Langages NP-Complets sur $\Sigma = \{a\}$[63] 108
Arbre couvrant minimum en couleurs (MLST) 108
Activation de Processus[64] 109
Intersection d'automates[65] 109
Sheffer 110

[57] Exercice 100% recopié de Florian Bourse : https://www.di.ens.fr/~fbourse/enseignement/Proba.pdf
[58] Le magnifique livre de Mme. Galatée Hemery ! ! ! ! <3
[59] Sujet custom!
[60] Partie 4 du projet de rocq du cours PRFA 2025/2026
[61] Tiré de https://www.lirmm.fr/~retore/LL/nonnon.pdf
[62] Centrale Info 2026
[63] Mallory Marin, doc de colles
[64] copie presque exacte CCINP 2023 sujet A
[65] Algo 1 ENS Lyon TD 8

MPI: Calculabilité 110
Cours 110
Indécidabilité 110
ARRET Bool → Bool[66] 110
Entrée bornée 111
L'indécidable est partout 111
Complexité et décidabilité[67] 111
Existence d'une quine 112
Réels calculables 112
Représentation d'ensembles infini 112

[66] Un mines-télécom bof de 2024

[67] tiré d'un oral d'Ulm 2021 https://a3nm.net/work/exams/ens/exercices_info_ulm_2021.pdf

MP2I: Programmation impérative, représentation machine

Je n'ai pas encore d'exos ici, car c'est beaucoup d'introduction aux bases de l'impératif et au cours sur le fonctionnement de la machine, et les exos y sont rarement intéressants. On pourrait quand même faire l'analyse de la fast inverse square-root ou des bizarreries avec des flottants.

Cours

- Paradigmes: impératif structuré, déclaratif fonctionnel, logique
- Savoir coder en C
- Différence entre compilé et interprété
- Représentation des entiers: signé, non signé, bytes/bits
- Représentation des flottants: signe, mantisse, exposant
- Preuve sur programme. Invariant de boucle. Correction partielle (vrai si termine), correction totale (vrai et termine)
- Complexité : pire cas, cas moyen, coût amorti
- Spécialisation d'une fonction. Commentaires, annotation
- Programmation défensive, assertions
- Jeu de test. Graphe de flot de contreole. Chemin faisable. Couverture des sommets, couverture des arêtes. Test exaustif d'une boucle.
- Pointeurs, allocation, stack, heap, call stack, stack overflow.

Complexité

En complexité il y a 2 écoles: celle qui suppose que les opérations sur les entiers (addition, multiplication, xor) sont en $O(1)$ et celle de ceux qui pensent que c'est en $O(\log n)$. On cherche maintenant à analyser la complexité dans la seconde école, avec des entiers de taille arbitraire. Pour ce faire, on représentera un entier par un `type int' = bool list;` tel que l'entier soit représenté en base 2 **à l'envers** avec (`true` = 1, `false` = 0).

Par exemple, 19 sera représenté par `[true, true, false, false, true];;`

1. Donner la liste qui encode les nombres 0, 1 et 13
2. Proposer le code de `val incr: int' -> int'` qui incrémente un entier de 1. Pourquoi représenter un entier à l'envers ?

On supposera la fonction `val decr: int' -> int'` écrite.

3. Pour deux entiers A, B, quelle est la complexité du code suivant ? Proposer un code plus rapide. Quelle est la complexité dans le pire des cas ?

```
let rec add (a: int') (b:int'): int' =
  match a with
  | [] | [false] -> b
  | _ -> incr (add (decr a) b)
```

4. Justifier que le programme suivant est bien en $O(N)$

```
let rec etrange (n:int') = match n with
  | [] | [false] -> [false]
  | _ -> incr (etrange (decr n))
```

MP2I: Programmation fonctionnelle

Cours

- Récursivité, récursivité croisé. Arbre d'appels.
- Résoudre des équations sur la complexité

- Différence entre structure mutable et non mutable
- Structure de données abstraite = type muni d'opérations
- Polymorphisme (HP)
- Techniques avancées de fonctionnel (HP)

Je préviens (si tu commence par cette section) que les exercices ici (surtout vers la fin) sont particulièrement difficille. C'est des bon exos si un collé est trop fort ou pour une fin de DS, mais pas pour des gens qui débutent.

Mémoisation Automatique[68]

On s'intéresse à écrire une fonction dans un langage de programmation fonctionnel capable de prendre en entrée une fonction f et de renvoyer une version mémoïsée de f.

1. Écrire une fonction `val memo: ('a -> 'b) -> ('a -> 'b)` telle que si f est une fonction non récursive, alors `memo f` renvoie une version mémoïsée de f
2. Écrire une fonction `val memo_rec: ('a -> 'b) -> ('a -> 'b)` telle que si f est une fonction récursive, alors `memo f` renvoie une version mémoïsée de f. On suppose que les appels récursif sont tous de la forme `memo f`. Attention à préserver la structure des appels récursifs.
3. Écrire une fonction `val fibo: int -> int` qui calcule la fonction de Fibonacci avec une complexité en $O(n)$ en utilisant `memo_rec`.

Suspension et listes infinies

On se propose d'implémenter, en OCaml, une bibliothèque de calcul paresseux. Celle-ci expose un type paramétré de *suspensions* `'a susp` et des fonctions de types suivants:

- Une fonction `make: (unit -> 'a) -> 'a susp`
- Une fonction `force: 'a susp -> 'a`

Une suspension (de type `'a susp`) contient une fonction permettant de calculer une valeur de type `'a` quand cela sera nécessaire. Elle peut être construite facilement grâce à la fonction `make f`. Le calcul n'est effectué que lorsque l'utilisateur de la bibliothèque le demande via la fonction `force susp`. Cette dernière fonction vérifie si le calcul a déjà été effectué : si tel est le cas, elle en renvoie le résultat pré-calculé. Sinon, elle lance le calcul de `f ()`, stocke le résultat pour de futurs appels, et elle le renvoie.

2. Donner une implémentation possible de cette bibliothèque en OCaml. *On pourra utiliser des références.*

On définit le type des *listes paresseuses* par

```
type 'a slist_cell =
| SNil
| SCons of 'a * 'a slist
and 'a slist = 'a slist_cell susp
```

3. Comparer la notion de liste paresseuse avec la notion habituelle de liste.
4. Définir une fonction `scons: 'a -> 'a slist -> 'a slist` qui ajoute un élément en tête d'une liste paresseuse. Définir une valeur `snil` représentant la liste paresseuse vide.
5. Écrire deux fonctions `shd: 'a slist -> 'a` et `stl: 'a slist -> 'a slist` qui prennent une liste paresseuse non vide en paramètre, et qui renvoient respectivement son premier élément et la liste paresseuse des autres éléments
6. Écrire une fonction `sappend: 'a slist -> 'a slist -> 'a slist` qui concatène en $O(1)$ deux listes paresseuses.

[68] de Maxime BRIDOUX

7. Écrire une fonction `srev: 'a slist -> 'a slist` qui renverse une liste paresseuse de manière efficace. Quelle est la complexité des accès aux différents éléments de la liste renversée ?
8. Définir en OCaml une liste paresseuse qui contient *tous* les carrés parfaits.

Continuation

On se donne le type tree suivant pour représenter des arbres binaires:

```
type 'a tree = E | N of 'a * tree * tree
```

La hauteur d'un arbre peut être calculée par la fonction height suivante, de type `tree -> int` :

```
let rec height t =
  match t with
  | E -> 0
  | N (_, l, r) -> 1 + max (height l) (height r)
```

1. On regarde le code suivant:

   ```
   let rec left t n = if n = 0 then t else left (N ((), t, E)) (n - 1) in
   let t = left E 1000000
   let h = height t
   ```

   La première ligne est exécutée sans problème mais la seconde provoque l'erreur `Fatal error: exception Stack_overflow`. Expliquer l'erreur

Pour parvenir à calculer la hauteur en toute circonstance, une solution consiste à adopter un style de programmation dit par continuation. Plutôt que de calculer directement la hauteur $h(t)$ d'un arbre t, on va calculer $k(h(t))$ pour une fonction k quelconque en paramètre de la fonction `height`. La hauteur s'en déduira alors en prenant pour k la fonction identité.

Observer ce code:

```
let rec aux1 t k =
  match t with
  | E -> k 0
  | N (_, l, r) ->
    aux1 l ((*1*) fun hl ->
      aux1 r ((*2*) fun hr ->
      k (1 + max hl hr)))
let height1 t = aux1 t ((*3*) fun h -> h)
```

2. Donner le type de `aux` et montrer que `height1 t` calcule la hauteur de `t`
3. On définit la taille d'un arbre $|t|$ par son nombre de noeuds. Donner une version par continuation de `size t k` qui calcule `k t'` pour t' la taille de t et k une fonction
4. Donner une version par continuation de `get_prefixe: 'a tree -> 'a list` qui calcule la liste du parcours préfixe de l'arbre donné en argument. On fera attention à avoir une bonne complexité.

On peut modifier le code par continuation pour qu'il n'utilise plus de fonctions anonymes, mais des valeurs d'un type somme OCaml qui représente les différentes fonctions qui rentrent en jeu. On appelle cela la *défonctionnalisation*. On reprend le code d'exemple pour la fonction `height1`

5. Compléter le code à trou suivant. K1, K2, K3 sont les fonctions anonymes marquées par les commentaires `(*1*)`, `(*2*)` et `(*3*)` respectivement. La composition d'une suite de fonctions anonymes est alors représentée par une liste desdites fonctions qui la composent, aka le type `cont`. `apply k v` simule l'application de la fonction anonyme représentée par `k` avec `v`. La fonction `aux2` est l'analogue de la fonction `aux1`

```
type cont =
  | K1 of tree * cont
  | K2 of int * cont
  | K3
let rec aux2 t k =
  match t with
  | E -> apply ...
  | N (l, r) -> aux2 ...
and apply k v =
  match k with
  | K1 (r, k) -> aux2 ...
  | K2 (h, k) -> apply ...
  | K3 -> ...
let height2 t = aux2 ...
```

6. Montrer que cette version défonctionalisé est "efficace" et permet de simuler des fonctions anonyme dans du code n'en permettant pas de base comme C.
7. Défonctionaliser le code obtenu question 4.

Binary random acces lists

On étudie ici un type inductif de liste qui stocke dans sa structure de type sa longueur, et qui permet d'avoir des opérations en $O(\log n)$ avec n sa longueur.

On ce done le type suivant:

```
type 'a seq =
  | NIL
  | ZERO of ('a * 'a) seq
  | ONE of 'a * ('a * 'a) seq
```

Dans cet exercice, quand on écrit un entier n en binaire, on l'écrit avec le bit de poids fort à la fin (à l'envers par rapport à l'écriture standard). On remarquera alors qu'un tel nombre fini toujours par 1, et peut commencer par 0. Par exemple, 6 s'écrit 011. Dans ce cas, la liste de longueur n est représenté par une suite de `ZERO` et de `ONE` qui correspon à cette écriture.

1. Donner en OCaml des valeurs de type `int seq` représentant respectivement les listes $[1, 2]$, $[4, 2, 3]$ et $[9, 1, 2, 3, 4, 5]$

Petit encart intéressant mais non obligatoire pour comprendre: En OCaml, normalement, un appel récursif doit forcément être sur le même type que l'entrée. Cela peut poser quelques problèmes. Considérer le code suivant:

```
let rec always_true a b =
  if a = b then true
  else always_true 27 27
```

Ici OCaml infère le type `int -> int -> bool`, car il y a un appel récursif de la forme `always_true 27 27`. Mais le type `'a -> 'a -> bool` marche aussi ! Pour préciser à OCaml qu'il doit généraliser un paramètre de type, on doit écrire le type de la fonction avant, en précisant qu'une variable doit marcher "pour tout type"

```
let rec always_true: 'a. 'a -> 'a -> bool = fun a b ->
  if a = b then true
  else always_true 27 27
```

Pour les questions suivantes, on ignorera ce genre de subtilitées (aka on donne le type de la fonction).

2. Écrire une fonction `val cons: 'a. 'a -> 'a seq -> 'a seq` tel que `cons x xs` ajoute `x` au début de la liste `xs`
3. Écrire une fonction `val uncons: 'a. 'a seq -> 'a * 'a seq` tel que `uncons xs` pour `xs` une liste non vide, renvoie le couple du premier élément et du reste de la liste.
4. Quels sont les complexités de `cons` et `uncons` ?
5. Écrire une fonction `val get: 'a. 'a seq -> int -> 'a` tel que `get xs i` renvoie le $(i+1)$-ème élément de `xs`.
6. Écrire une fonction `val set: 'a. 'a seq -> int -> 'a -> 'a seq` tel que `set xs i x` remplace le $(i+1)$-ème élément de `xs` par `x`. On pourra s'aider des autres fonctions. Une complexité de $O(\log^2 n)$ est attendu.

On va essayer de faire `set` en $O(\log n)$. Pour cela, on chercher à définir `val transform: 'a. 'a seq -> int -> ('a -> 'a) -> 'a seq` tel que `transform xs i f` applique `f` au $(i+1)$-ème élément de la liste `xs` et renvoie la nouvelle liste.

7. Coder `transform`, de manière à ce que la complexité soit en $O(\log n + C_f)$ avec C_f la complexité de f
8. Donner une nouvelle version de `set` à l'aide de `transform` qui est en $O(\log n)$
9. En appliquant une technique de défonctionalisation vu dans l'exercice sur les continuations, proposer un type enum `type trans = | ...` et une implémentation de `transform` et `set` de telle sorte à ce que le type soit maintenant `val transform2: 'a. 'a seq -> int -> trans -> 'a seq`. A la fin, on ne devrai plus voir de fonction anonymes.

   Petite rappel de l'intuition sur la défonctionalisation: on simule la stack avec une liste.

Trop cool, on a obtenu un `set` en $O(\log n)$ qui n'utilise pas de fonctions anonymes !

Monades

On considère les modules de signature qui suit:

```
module type Monad = sig
  type 'a m
  val return: 'a -> 'a m
  val bind: 'a m -> ('a -> 'b m) -> 'b m
end
```

On définit une *monade* comme étant un module de signature `Monad` qui respecte 3 égalitées:

- `return` est neutre à gauche: `bind (return x) f = f x`
- `return` est neutre à droite: `bind m return = m`
- La monade est associative: `bind (bind m f) g = bind m (fun x => bind (f x) g)`

Question 1 On se donne le module suivant:

```
module OptionMonad = struct
  type 'a m = | None | Some of 'a;
  let return x = Some x;
  let bind x f = match x with
    | None -> None
    | Some x -> f x
end
```

Montrer que `OptionMonad` est une monade

Question 2 Proposer une implémentation de `ListMonade` avec `type 'a m = 'a list`

Soit M une monade, on définit alors les fonctions suivantes:

```
let map f x = bind x (fun y -> return (f y))
let join x = bind x (fun y -> y)
```

Question 3 Donner le type de `map` et `join`. Pour les monades de la question 1 et 2, à quoi correspond `map` et `join` ? *`join` est parfois appellé `flatten`*

Question 4 Montrer que `map` et `join` satisfait `join (map join x) = join (join x)`.

On admettra que `map` et `join` satisfont aussi
- `join (map return x) = join (return x)`
- `join (map (map f) x) = map f (join x)`

Je ne suis pas sur de cette question et de la question 5

Question 5 Montrer que si un module possède `join` et `map` avec les 3 équations précédente alors on peut le munir d'une structure de monade. On donnera la définition de `bind` et la preuve que la loi d'associativié est respectées (on admettera les 2 autres).

Question 6 On s'intéresse maintenant à la monade suivante:

```
module ContIntMonad = struct
  type 'a m = ('a -> int) -> int
  let return x k = k x
  let bind c f k = c (fun t -> f t k)
end
```

Montrer que c'est une monade.

Question 7 Que retourne le code suivant ?

```
open ContIntMonad
let rec mystere n =
  if n <= 1 then return n
  else bind (mystere (n-2)) (fun v -> return (v+1))
in mystere 45 (fun i -> i)
```

Question 8 Proposer une implémentation récursive naïve de la fonction de Fibonacci (sans se soucier de l'explosion des appels récursifs) en utilisant la monade `ContIntMonad`

Parsing avec des monades[69]

Attention: Ce sujet nécessite de comprendre les `lazy` (voir sujet "Suspensions et files persistantes"), et sera bien plus facile avec la compréhension des monades.

On cherche à définir un parser en OCaml en utilisant des combinateurs que l'on pourra imbriquer les uns aux autres. Un parser sera une fonction de `char list -> ('a * char list) seq`, avec `seq` les séquences (potentiellement infinies) suspendues

```
type 'a seq = | Nil | Cons of 'a * 'a node
and 'a node = 'a next
type 'a parser = char list -> ('a * char list) seq
```

L'idée est qu'un objet de type `'a parser` est une fonction qui prend une liste de caractère et qui renvoie une séquence (potentiellement infinie) de résultats de match. Chaque résultat est modélisé par un couple `(res,reste)` tel que `res` soit le résultat du parsing correct d'un préfixe et `reste` soit la suite de liste de caractère pas encore lu. En particulier, si la séquence est vide, c'est que le parsing a échoué. On dit qu'un caractère est consommé par le parser s'il est lu.

[69] MPRI - FUN Exam 2026

Par exemple, si `parser_int: int parser` parse des nombres, on peut imaginer que `parser_int ['0','4','5','3','v','7']` renvoie la séquence contenant $[(0, 453\text{v}7), (4, 53\text{v}7), (45, 3\text{v}7), (453, \text{v}7)]$. Dans ce cas, les caractères v et 7 ne sont jamais consommé.

Question 1 Définir les fonctions suivantes:

- Une fonction `val fail: 'a parser` qui est le parser qui échoue tout le temps.
- Une fonction `val return: 'a -> 'a parser` tel que `return x` est un parser qui pour toute entrée l renvoie $[(x, l)]$.
- Une fonction `val satisfy: (char -> bool) -> char parser` telle que `satisfy predicate` renvoie un parser qui consomme qu'un caractère c s'il satisfait $\text{predicate}(c)$, ou qui échoue sinon.
- Une fonction `val char: char -> char parser` telle que `char c` renvoie le parser qui ne consomme que le premier caractère seulement s'il est égal à `c`.

Question 2 Une fonction `val bind: 'a parser -> ('a -> 'b parser) -> 'b parser` tel que `bind p f` renvoie un parser qui pour une entrée l donné, applique $p(l) = [(x_1, l_1), ..., (x_n, l_n), ...]$ et renvoie la concaténation des $(f(x_i)(l_i))_{i\in\mathbb{N}}$

Question optionelle Montrer que `parser` avec `bind` et `return` forment une monade.

Question 3 Définir la fonction `let (|||) p1 p2 = ...` telle que `(|||) p1 p2` renvoie le parser qui effecture la concaténation des résultats des deux parser. On rappelle que la syntaxe `(|||)` en OCaml permet de redéfinir la fonction `||` et d'écrire `p1 || p2` à la place de `(|||) p1 p2`.

Question 4 (dur) Définir la fonction `val fix: ('a parser -> 'a parser) -> 'a parser` qui prend en argument une fonction `f` et qui renvoie un parser `p` tel que `p` agit comme `f p`. On fera bien attention à ce que `fix` termine quand `f` termine.

Question 5 Définir la fonction `val map: ('a -> 'b) -> 'a parser -> 'b parser` tel que `map f p` renvoie le parser qui applique une fonction `f` au résultat du parsing de `p`

On définit alors le combinateur suivant:

```
let (<*>) (p1: ('a -> 'b) parser) (p2: 'a parser): 'a parser = bind p1 (fun f -> map f p2)
```

Il prend en argument deux parser `p1` et `p2`, parse l'entrée d'abord avec p_1, puis ensuite avec p_2, et applique la fonction résultat du parsing de p_1 au résultat du parsing de p_2, et ceci pour chaque entrée. Comme pour `(|||)`, on pourra écrire `p1 <*> p2` au lieu de `(<*>) p1 p2`.

Question 6 Quel est le type et que fait la fonction suivante ?

```
let parens p = ((map (fun _ x _ -> x) (char '(') <*> p) <*> char ')')
```

Question 7 Quel est le type et que fais `parse_d` définit si-dessous ?

```
let parse_d = fix (fun parse_d ->
    ((map (fun b -> max (1+b)) (parens parse_d)) <*> parse_d)
  || return 0)
```

Question 8 Définir un combinateur `val repeat: 'a parser -> int -> 'a list parser` tel que `repeat p n` parse `n` séquences concaténées reconnues par `p` et renvoie la liste des séquences

Question 9 Définir un combinateur `val list: 'a parser -> 'a list parser` qui reconnaît une suite arbitraire de concaténation de séquences reconnues par `p`

Question 10 Écrire un parser qui reconnaît exactement le plus grand préfixe de $\{0, 1\}$ et qui renvoie sa longueur. Le parser renverra une séquence d'exactement un élément $[(x, l)]$ avec x la longueur et l le reste de l'entrée. On n'utilisera que les combinateurs précédemment utilisés

Question 11 Écrire un parser qui reconnaît exactement les entrées (et pas leur préfixes) dans le langage $\{a^n b^n c^n : n \in \mathbb{N}\}$ et qui retourne sa longueur.

MP2I: Structures de données séquentielles

Cours

- Définition d'une liste. Liste doublement chainée. D'un tableau. Cout des opérations de création, ajout, supression en temps et mémoire.
- Différence entre struture immuable et mutable.
- Définition d'une pile. Implémentation d'une pile par tableau dynamique.
- Définition d'une file. Implémentation par une liste doublement chainé, et par un tableau dynamique.
- Table de Hachage, tableau associatif.
- Hachage: Collision, densité, structure hachable (HP)
- Sérialisation d'un tableau ou d'une table de hashage (ou d'un arbre).

Complexité des opérations de bases

Pour chacune des structure entre tableau, liste, liste doublement chainée, tableau dynamique, pile, file, que l'on considèrent toutes de longeur n indiquer les complexitées au pire des cas, meilleurs des cas et amortis des opérations suivantes :

1. Lire le premier élément.
2. Lire le dernier élément.
3. Ajouter un élément à la fin.
4. Ajouter un élément au début.
5. Retirer le premier élément.
6. Retirer le dernier élément.
7. Inverser l'ordre.
8. Tester si il y a un 42.
9. Insérer un élément au milleu.

Merge de liste

Soit L_1, L_2 deux listes triées. Proposer un algoritme en OCaml qui calcule la liste trié des éléments contenue dans L_1 et L_2.

Calcul des différences

Proposer un algorithme qui prend un tableau d'entiers T de longueur N et qui en $O(n)$ renvoie $\sum_{0 \leq j < i < n} T[i] - T[j]$

On peut facilement transformer cet exercice en un autre en changeant la formule

Permutation suivante

On représente une permutation σ comme un tableau T de longueur n tel que $\forall i < N, T[i] = \sigma(i)$

Donner le code d'une fonction qui à une permutation de $[\![0; N]\!]$ représenté par T retourne la prochaine permutation de $[\![0; N]\!]$ dans l'ordre lexicographique.

Tableaux binaire

On considère le problème suivant:

Minimum change binary matrix
Entrée: Une matrice $M \in \mathcal{M}_{n,m}(\{0,1\})$
Sortie: Le nombre minimum de coefficients à changer pour que chaque ligne et chaque collone ait un nombre pair de 1.

Donner un algorithme résolvant ce problème en $O(NM)$ avec N, M les dimensions du tableau. Montrer sa correction.

Sous-tableau connexe

Soit A un tableau de n entiers relatifs, on cherche un algorithme en $O(n)$ qui calcule le sous-tableau connexe qui maximise la somme de ses éléments, i.e. le couple (i, j) avec $0 \leq i \leq j \leq |T|$ tel que $\sum_{k \in [\![i;j]\!]} A[k]$ soit maximal.

Le glouton par défault[70]

Étant donné un ensemble $\{x_1, ..., x_n\}$ de n points sur une droite, décrire un algorithme qui détermine le plus petit ensemble d'intervalles fermés de longueur 1 qui contient tous les points donnés.

Prouver la correction de votre algorithme et donner sa complexité.

Recherche dans une matrice

Question 1 Donner le code C d'une fonction `int find(int *arr, int n, int key);` qui prend en argument un tableau `arr` trié de longueur `n` et qui trouve un indice `i` tel que `arr[i] == key`.

On cherche à faire un algorithme efficace aussi dans le cadre de matrice trié. On dit qu'une matrice M de taille $n \times m$ est *linéairement trié* si chaque ligne est trié de haut en bas et que chaque colone est trié de gauche à droite.

Question 2 Donner le code d'un fonction C `bool is_ordered(int **mat, int n, int m)` qui à une matrice `mat` de taille $n \times m$ teste si elle est linéairement trié.

Question 3 Donner le code d'une fonction `is_in(int **mat, int n, int m, int key)` qui teste si `key` est présent dans la matrice `mat` linéairement trié de taille $n \times m$ par diviser pour reigner.

Question 4 Meme question en $O(n + m)$

Question 5 Donner le code d'une fonction C qui en $O(n)$ trouve un minimum local dans une matrice $n \times n$

Tableau cumulatif

Soit T un tableau de n entiers relatifs, on note C_T le tableau de longueur N tel que

$$C_T[i] = \sum_{j=0}^{i} T[j]$$

Question 1 Donner le code de la fonction `int* get_cumulatif(int* T, int n);` qui, à un tableau T de longueur N associe son tableau cumulatif.

Question 2 Montrer par récurrence sur la longueur de T que si $\max C_T > N$ avec N la longueur du tableau alors il existe un élément i avec $T[i] > 1$.

On cherche maintenant $s \leq t \leq N$ tel que $\sum_{i=s}^{t} T[i]$ soit maximale

[70] Algo 1 ENS Lyon

Question 3 Proposer un algorithme qui, étant donné un tableau T, renvoie $i < j$ tel que $T[j] - T[i]$ soit maximal. En déduire un code C qui répond au problème en utilisant un tableau cumulatif.

Question 4 On cherche maintenant à calculer le nombre de couples (i, j) avec $i < j$ tel que $\sum_{i=s}^{t} T[i]$ soit maximale. Proposer un algorithme répondant au problème.

Multiplication rapide de polynome[71]

Ici on considère des polynômes d'entiers $\mathbb{Z}[X]$. Soient $P, Q \in \mathbb{Z}_n[X]$, leur produit $R = PQ \in \mathbb{Z}_{2d}[X]$

On représente un polynome par un tableau d'entiers T, tel que pour $P \in \mathbb{Z}_d[X]$, si on écrit $P = \sum_{i \leq d} a_i X^i$ on a $T[i] = a_i$.

1. Donner une fonction C `int* multiply(int* p, int* q, int n)` qui prend deux polynômes de degré $\leq n$ représenter par deux tableaux de longueur n et qui renvoie le polynome produit.
2. Soit $n = 2m$, pour $P \in \mathbb{Z}_n[X]$, montrer que on peut décomposer $P = P_1 + X^m P_2$ avec $P_1, P_2 \in \mathbb{Z}_m[X]$.

Soient $P, Q \in \mathbb{Z}_n[X]$, on décompose $P = P_1 + X^m P_2$ et $Q = P_1 + X^m P_2$. On définit alors $R_1 = P_1 Q_1$, $R_2 = P_2 Q_2$ et $R_3 = (P_1 + P_2) \times (Q_1 + Q_2)$.

3. Exprimer $P \times Q$ en fonctions de R_1, R_2, R_3.
4. En déduire un algorithme récursif pour calculer le produit de polynome.
5. Quelle est la complexité de cet algorithme ?

Structure efficace pour représenter des ensembles

On cherche à implémenter une structure de données pour représenter un sous-ensemble de $[\![0; N]\!]$ (N sera fixé par la fonction `create`). Pour cela, on chercherait à définir une structure avec 3 opérations:

- `val create: int -> set` telle que `create N` renvoie $\emptyset$ (un sous-ensemble de $[\![0; N]\!]$)
- `val add: set -> int -> unit` telle que `add x i` renvoie $X \cup \{i\}$ (si $0 \leq i \leq N$)
- `val del: set -> int -> unit` telle que `del x i` renvoie $X \setminus \{i\}$ (si $0 \leq i \leq N$)

1. Proposer une implémentation de cette structure en utilisant des listes. Quelles sont les complexités des différentes opérations en mémoire et en espace ?

On cherche à avoir une complexité spatiale aussi petite que possible. Attention, à partir de maintenant, on prendra en compte la taille des entiers. On rappelle qu'un entier N à une taille de mémoire $\log_2(N)$.

2. On considère la représentation qui à $1 \leq a_1 \leq ... \leq a_m \leq n$ associe la liste

$$[a_1, a_2 - a_1, a_3 - a_2, ..., a_m - a_{m-1}]$$

Montrer que cette représentation est en $O(n)$ de mémoire.

3. Implémenter les fonctions `add` et `del` pour cette représentation. Quels sont les complexités ?
4. Proposer une implementation tel que `add` et `del` soient en $O(1)$ en complexité, mais que la structure soit toujours en $O(N)$ d'espace. La fonction `create` peut-être en $O(N)$.
5. Peut-on faire mieux en complexité spatiale que $O(N)$?

Inverser une liste sans toucher aux pointeurs

On considère le type suivant d'une liste en C :

[71] Algo 1 ENS Lyon

```c
struct list_t {
  int v;
  struct list_t *next;
};
typedef struct list_t list_t;
```

1. Donner le code C d'une fonction `list_t* invert(list_t *liste);` qui inverse la liste donné en entrée en $O(n)$ de temps et $O(1)$ d'espace.

On cherche a crée une solution de la meme complexité telle que les pointeurs des différentes listes ne changent pas, et seules les valeurs `v` sont inversé à la fin de l'évaluation. Durant l'évaluation, on a le droit de modifier les pointeurs `*next` tant qu'ils sont remis à leur état initial à la fin.

2. Proposer une solution en $O(n^2)$ de temps et $O(1)$ d'espace.
3. Donner une solutionn en $O(n)$ de temps et $O(1)$ d'espace. On pourra considérer couper la liste en 2 bouts et en inverser qu'un seul en utillisant la méthode usuelle.

Liste cyclique sans lièvre et torture

1. Proposer une structure pour implémenter une liste simplement chainé en C.

Remarquer que le pointeur "next" pourrait être n'importe quel maillon de la liste, y compris un des éléments précédemment vu. On dit qu'une liste est cyclique si un pointeur "revient" sur un des maillons précédent. Dans ce cas, la liste représenté est infinie et, à partir d'un certain rang, devient périodique.

2. Proposer un code C pour définir la liste cyclique qui représente $[1, 2, 3, 2, 3, \ldots]$ en n'utilisant que 3 maillons.
3. Proposer un algorithme pour tester si une liste est cyclique. On fera bien attention à ce que l'algorithme termine.
4. Donner une fonction `liste_t* reverse(liste_t* list);` qui inverse une liste simplement chainé en C. Que ce passe-t'il si on inverse une liste cyclique ?
5. On cherche maintenant à calculer le nombre de maillons d'une liste cyclique. Proposer un algorithme en $O(n)$ de temps et $O(1)$ de mémoire.

Info bonus: Sachez que l'on peut définir de telle listes en OCaml ! Le code suivant définit la liste de la question 2 : `let q2 = 1 :: (let rec l = 2 :: 3 :: l in l)`

Tableaux auto-référents

Pour T un tableau de longueur n, on définit l'histogramme de T noté H_T comme le tableau de longueur n tel que $H_T[i]$ correspond au nombre de i dans T. Formellement, $H_T[i] = |\{j : T[j] = i\}|$

On dit que T est auto-référent si $H_T = T$. Par exemple, $[1, 2, 1, 0]$ est un tableau autoréférent.

Question 1 Donner le code d'une fonction `int* histogramme(int *t, int n);` qui, à un tableau t de longueur n renvoie H_t son histogramme.

Question 2 Donner tous les tableaux auto-référent de longueur 4 et 5

Question 3 Donner le code d'une fonction `bool is_autoref(int *t, int n);` qui teste si un tableau t est auto-référent.

Question 4 Donner le code d'une fonction de recherche exhaustive `int count_autoref(int n);` qui donne le nombre de tableaux auto-référents de longueur n.

Question 5 Montrer que pour tout $n > 6$ il existe un tableau autoréférent de longueur n

Question 6 $(*)$ Montrer qu'il est unique

MP2I: Inductions & ordres[72]

Cours

- Définition d'une relation d'équivalence.
- Définition d'une relation d'ordre, ordre total = bon ordre.
- Minimum, éléments comparables, incomparables, chaine, antichaine.
- Les deux définitions d'un ordre bien fondé, et la démonstration de leur équivalence.
- Préordre, préordre bien fondé = WQO (HP)
- Montrer que $(\mathbb{N}, \leq)$ est bien fondé.
- Ordre lexicographique, ordre produit. Démonstration que si A, B sont bien ordonné, alors $A \times B$ l'est aussi pour l'ordre produit et l'ordre lexicographique
- Qu'est-ce qu'un ensemble inductif ?

Des ordres

Les ordres suivant sont-ils bien fondés ?

1. $\leq$ sur $\mathbb{Z}$
2. $\leq$ sur $\mathbb{R}^+$
3. $\leq$ sur $\mathbb{Q}^+$
4. $\leq_{\text{lex}}$ sur $\mathbb{N}^2$
5. $\subseteq$ sur $\mathcal{P}(\mathbb{N})$
6. $\subseteq$ sur $\mathcal{P}_{\text{finie}}(\mathbb{N})$

Des ensembles inductifs

Proposer une définition inductive de chacun des ensemble suivants

1. $2\mathbb{N}$
2. Les mots sur un alphabet Σ fixé
3. Les puissances de 2
4. $\{(x, y) \in \mathbb{N}^2 : x < y\}$
5. $\{(x, y) \in \mathbb{N}^2 : 2x = y\}$
6. Les listes d'entiers contenant un 42

Soustraction entière

La soustraction entière est la fonction $\text{sub} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ définit par:

$$\begin{cases} \text{sub}(m, 0) = m \\ \text{sub}(0, n) = 0 \\ \text{sub}(m, n) = \text{sub}(m-1, n-1) \end{cases}$$

1. Montrer que sub est bien définie.
2. Montrer que $\forall m, n \in \mathbb{N}, \text{sub}(m, n) \leq m$

Ordre sur les mots

Étant donné un ensemble alphabet Σ, on pose pour $n \in \mathbb{N}^*$ l'ensemble

$$\Sigma^n = \underbrace{\Sigma \times ... \times \Sigma}_{n \text{ fois}}$$

des mots d'exactement n lettres. L'ensemble $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$ est alors l'ensemble de tous les mots sur Σ.

1. Soit un ensemble ordonné $(\Sigma, \leq)$ tel que $\Sigma = \{a, b, c\}$ et $a < b < c$. Donner une définition de l'ordre lexicographique $\leq_{\text{lex}}$ sur Σ^3 puis l'utiliser pour ordonner les mots aba, baa, caa, aaa, abc, bbb.
2. Montrer que l'ordre lexicographique $\leq_{\text{lex}}$ sur Σ^n est une relation d'ordre bien fondée.
3. Est-ce que l'ordre lexicographique $\leq_{\text{lex}}$ est une relation d'ordre bien fondée sur Σ^* ?
4. Montrer que l'ordre $\leq_m$ sur Σ^* définit par

[72] Maxime Bridoux, MP2I

$$u \leq_m v \text{ si et seulement si } |u| < |v| \text{ ou } (|u| = |v| \text{ et } u \leq_{\text{lex}} v)$$

est une relation d'ordre totale sur Σ^*. Est-ce que $\leq_m$ est bien fondé ?
5. En déduire que l'ensemble Σ^* est dénombrable.

Tri aléatoire

On considère l'algorithme de tri suivant:
- Tant qu'il existe $i < j$ tel que $T[i] > T[j]$, on échange $T[i]$ et $T[j]$

1. Montrer que cet algorithme termine toujours
2. Quel est sa complexité dans le pire des cas ?

Langage de Dyck

On se fixe un ensemble fini de lettres $\Sigma = \{a, b\}$. On déifni un *mot* comme étant une suite finie de lettres de Σ. Le mot vide (la suite vide) sera noté ε, et la concaténation de deux mots u, v sera noté par la concaténation uv. On définit par induction l'ensemble A par:
- $\varepsilon \in A$
- Si $u \in A$, alors $aub \in A$
- Si $u, v \in A$, alors $uv \in A$

1. Montrer que tous les mots dans A ont autant de a que de b
2. Monter que l'ensemble de ces règles sont ambigues. Proposer sans justifier une verison non-ambigue (et faite la vérifier par la colleuse avant de passer à la suite).
3. Soit $w_1 ... w_n$ un mot de A, montrer qu'il existe un $i \leq n$ tel que pour tout $j < i$, le mot $w_1 ... w_j$ possède strictement plus de a que de b et que le mot $w_1 ... w_i$ possède autant de a que de b
4. Montrer que votre définition de la question 2 est équivalente à la définition question 1.

Système MIU

On pose $\Sigma = \{M, I, U\}$ et on définit inductivement l'ensemble $S \subseteq \Sigma^*$ par les règles suivantes:
- $MI \in S$
- Si $x \in \Sigma^*$ et $xI \in S$ alors $xIU \in S$
- Si $x \in \Sigma^*$ et $Mx \in S$ alors $Mxx \in S$
- Si $x, y \in \Sigma^*$ et $xIIIy \in S$ alors $xUy \in S$
- Si $x, y \in \Sigma^*$ et $xUUy \in S$ alors $xUy \in S$

Ce système est extrait du livre « Gödel, Escher, Bach » de Douglas Hofstadter, sous la forme du puzzle suivant: est-ce que la chaîne MU appartient au système MIU ?

1. Montrer que $MUIU \in S$
2. Donner l'ensemble des chaines de S que l'on peut dériver en appliquant au plus 3 règles.
3. Démontrer que toute chaine $s \in S$ commence par un M
4. Démontrer que pour toute chaine $s \in S$, son nombre de $|s|_M$ de M est exactement 1
5. Démontrer que pour tout chaine $s \in S$, son nombre de $|s|_I$ de I n'est jamais un multiple de 3. Conclure.

Sous-ordres indénombrable de $(\mathcal{P}(\mathbb{N}), \subseteq)$

1. Est-ce qu'il existe $A \subseteq \mathcal{P}(\mathbb{N})$ indénombrable tel que tous les éléments de A sont incomparable deux à deux pour $\subseteq$?
2. **(∗)** Est-ce qu'il existe $T \subseteq \mathcal{P}(\mathbb{N})$ indénombrable tel que $(T, \subseteq)$ soit une relation d'ordre totale ?
3. Montrer qu'il existe une infinité indénombrable d'ordre totaux sur $\mathbb{N}$ non isomorphes les uns aux autres.

MP2I: Arbres & structures inductives

Cours

- Donner la définition inductive d'un arbre. D'un arbre binaire.
- Qu'est-ce qu'une feuille ? Noeud ? Racine ?
- Donner la défintion de la hauteur d'un arbre.
- Définition d'un arbre complet. Localement complet. Parfait.
- Donner la définition d'un tas min (et d'un tas max)
- Rapeller le principe du tri par tas et sa complexité.
- Donner la définition d'un arbre binaire de recherche.
- Arbre rouge-noir. Ajout dans un arbre rouge-noir.

Questions de programmation du cours:

Pour chacune des fonctions suivantes, donne le code en OCaml et en C. Les signatures sont données en OCaml:

- Coder la fonction `let hauteur (t: 'a tree): int` qui prend un arbre en entrée et qui renvoie sa hauteur.
- Coder la fonction `let count (t: 'a tree): int` qui prend un arbre en entrée et qui renvoie son nombre de sommets.
- Coder la fonction `let prefixe (t: 'a tree): 'a list` qui prend un arbre en entrée et qui renvoie son parcours préfixe. *Le code C renverra un tableau au lieu d'une liste*
- Coder la fonction `let infixe (t: 'a tree): 'a list` qui prend un arbre en entrée et qui renvoie son parcours infixe. *Le code C renverra un tableau au lieu d'une liste*
- Coder la fonction `let suffixe (t: 'a tree): 'a list` qui prend un arbre en entrée et qui renvoie son parcours suffixe. *Le code C renverra un tableau au lieu d'une liste*
- Coder la fonction `let add (t: int tree) (x:int): int tree` qui ajoute à un arbre binaire de recherche `t` un élément `x`.
- Coder la fonction `let remove (t: int tree) (x:int): bool` qui retire à un arbre binaire de recherche `t` un élément `x`, et renvoie s'il existait.
- Coder la fonction `let push (t: tas) (x:int): tas` qui ajoute à un tas min `t` un élément `x`.
- Coder la fonction `let pop (t: tas): int` qui retire d'un tas min `t` son minimum et le renvoie.

Exercices à ajouter

- Comptage de sous-arbres: https://diplome.di.ens.fr/informatique-ens/annales/2017_InfoU-exercices.pdf
- https://en.wikipedia.org/wiki/Merkle_tree (mélange hash et arbres)

Nombre de sommets

Montrer que pour A un arbre binaire, si on note $|A|$ son nombre de noeuds et $h(A)$ sa hauteur, montrer que $|A| \leq 2^{h(A)+1} - 1$

Plus de feuilles

Soit T un arbre tel que chaque sommet qui n'est pas une feuille a un degré au moins 3. Montrer que T a plus de feuilles que de nœuds internes.

Contour d'un arbre

On considère la fonction contour suivante en OCaml:

```
type 'a tree = F | N of 'a tree * 'a * 'a tree;
let rec contour t = match t with
```

```
  | F -> []
  | N (g,x,d) -> [x] @ (contour g) @ [x] @ (contour d) @ [x]
```

1. Quel est le type de la fonction `contour` ?
2. Proposer une implémentation avec un accumulateur qui est terminal récursive de `contour`.
3. Donner le code de `val to_prefixe: 'a list -> 'a list` tel que `to_prefixe (to_contour t)` renvoie le parcours préfixe de `t`.
4. Montrer que pour chaque noeud x de parent p, on a $[x, p]$ et $[p, x]$ qui apparaissent dans le contour.

Arbres d'intervalles

Un arbre d'intervalles est un arbre binaire de recherche dont tous les nœuds contiennent un intervalle de la forme $[a; b]$, dont les clefs sont dans l'ordre lexicographique définie par la relation d'ordre $\preceq$:

$$(a, x) \preceq (b, y) \iff a < b \vee (a = b \wedge x \leq y)$$

On se donne le type suivant en OCaml:

```
type intervalle = int * int;;
type arbre_int = F | N of intervalle * arbre_int * arbre_int;;
```

Question 1 Qu'est-ce qu'un arbre binaire de recherche ? Proposer une structure en C similaire à celle proposée pour représenter un arbre d'intervalles.

Question 2 Donner en C la fonction `int hauteur(arbre* a)` donnant la hauteur d'un arbre d'intervalle.

Question 3 Dessiner puis donnez en OCaml un arbre complet contenant les intervalles $\{[0; 2]; [0; 1]; [1; 3]; [4; 5]; [3; 5]; [3; 3]\}$

Question 4 Donner `val trouver: arbre_int -> intervalle -> intervalle` tel que `trouver a i` retourne un intervalle de l'arbre `a` intersectant `i` en $O(h)$ avec h la hauteur de `a`.

Question 5 Définir les opérations de rotations sur les arbres binaire de recherche. Donnez la fonction `val rotg: arbre_int -> arbre_int` effectuant l'opération de rotation gauche. On supposera écrite la fonction `val rotd: arbre_int -> arbre_int` l'opération de rotation droite.

Question 6 Donner `val ajouter: arbre_int -> intervalle -> arbre_int` tel que `ajouter a i` ajoute à un arbre équilibré `a` l'intervalle `i`. On veillera à ce que `a` reste équilibré.

Minimum locaux

On dit qu'un noeud u est un minimum local d'un arbre binaire A stoquant des entiers tous distincts si u est plus petit que son parent et ses (pottentiels) enfants.

1. Montrer que tout arbre non vide possède des minimums locaux. Proposer un code ocaml pour en calculer un.
2. Donner un algorithme qui compte le nombre de minimum locaux.
3. Soit n le nombre de noeud d'un arbre. Montrer que le nombre de minimum locaux est majoré par $\left\lfloor \frac{2n+1}{3} \right\rfloor$

Accès dans un arbre parfait

On se donne le type C suivant

```
typedef struct Noeud *arb;
struct Noeud {
  int valeur;
```

```
  arb fils_g;
  arb fils_d;
};
```

1. Donnez une définition équivalente de ce type en OCaml. Toujours en OCaml, donnez la fonction `val hauteur: arbre -> int` donnant pour un arbre quelconque sa hauteur.
2. Dessinez un arbre parfait à 7 nœuds. Tout les arbres complets sont-ils parfaits ?
3. Démontrez que tout arbre parfait de hauteur h possède $2^{h+1} - 1$ nœuds.
4. Donnez `bool est_parfait(arb a)` renvoyant vrai si l'arbre `a` est parfait.
5. Donnez `arb arb_trouve(arb a, int k)` renvoyant le `k`-ème élément dans l'ordre préfixe de l'arbre. On suppose ici que `a` est parfait et que $0 \leq k < n$ avec n le nombre de nœuds.
6. Discutez de la complexité de `arb_trouve` et de potentiels moyens de l'améliorer. On pourra chercher un algorithme en $O(\ln n)$

Arbres 2-dimensionnels

On cherche à créer une structure de données similaire à un arbre binaire de recherche pour des couples de points. On pose le type suivant pour des arbres 2-dimensionnels

```
type tree = F | N of tree * int * int * tree
```

On pose sur $\mathbb{N}^2$ la relation d'ordre $(x, y) \leq_2 (x', y') \Leftrightarrow x + y \leq x' + y'$.

1. Montrer que $\leq_2$ est bien une relation d'ordre bien fondé.
2. Donner la fonction `add (a:tree) (x: int * int): tree` qui ajoute à un arbre binaire de recherche `a` l'élément `x` selon la relation d'ordre $\leq_2$
3. On cherche à obtenir tous les éléments dans l'ordre trié selon la première composante. Quelle est la complexité d'un algorithme qui retourne ça avec notre structure, en supposant l'arbre équilibré ? Et si on avait utilisé la relation d'ordre $(x, y) \leq' (x', y) \Leftrightarrow x \leq x'$ à la place ?

Pour être capable de renvoyer tous les élément dans l'ordre trié selon la première composante (ou la deuxième, au choix), on considère des arbres 2 dimensionels: pour savoir si (x, y) sera fils gauche ou droit de la racine (x', y'), on regarde en fonction de la profondeur de (x', y') :

- Si la profondeur est paire, on compare selon la première composante
- Si la profondeur est impaire, on compare selon la seconde.

4. Donner un arbre binaire 2-dimensionnels contenant les couples $[(1, 3), (5, 1), (6, 0), (2, 4), (7, 5), (0, 6), (3, 2)]$ respectant la définition du dessus.
5. Donner les fonctions d'ajout dans un tel arbre 2-dimensionel.
6. En supposant l'arbre équilibré, quelle est la complexité d'un algorithme qui renvoie la liste des points stoqué dans l'arbre trié selon la première composante ?

Reconstruire l'arbre avec des parcours

Pour T un arbre *binaire* contenant des entiers tous distincts, on note $T_{\text{pref}}, T_{\text{inf}}, T_{\text{post}}$ respectivement les listes des parcours préfixe, infixe et postfixe de T.

On cherche ici à reconstruire T à partir de T_{inf} et T_{post}

Question 2 Montrer que le parcours T_{pref} et T_{suff} ne suffisent pas forcément pour reconstruire T

Question 3 Donner un algorithme récursif pour reconstruire T à partir de T_{pref} et T_{inf}

Question 3 Quelle est la complexité de l'algorithme ?

Question 4 Donner le code d'une fonction C `arb_t *get_arb(int n, int *pref, int *post);` qui retourne l'arbre T associé à T_{inf} et T_{post}

Question 5 Montrer que si l'arbre est localement complet (chaque nœud à 2 ou 0 enfants) alors on peut reconstruire T à partir de T_{pref} et T_{post}

Arbre canonique[73]

On définit une structure d'arbre:

```
type tree = F | N of tree * tree;;
```

Un arbre binaire strict (ou localement complet) est dit canonique si pour A et B deux feuilles, on a A moins profond que B ssi A arrive avant B dans un parcours préfixe.

Un arbre canonique peut-être représenté par un tableau qui à chaque hauteur associe son nombre de feuilles.

Question 1 Donner une définition équivalente de ce type en C. Toujours en C, donnez la fonction `void parcours(arbre* arb)` qui affiche (`print`) le parcours préfixe de `arb`.

Question 2 Donner les arbres canoniques des tableaux $[0; 2]$, $[0; 0; 3; 2]$, $[0; 1; 1; 1; 1; 2]$.

Question 3 Démontrer que le tableau d'un arbre canonique de longueur plus grande que 1 doit se terminer avec un nombre pair.

Question 4 Donner une fonction OCaml `val to_array: tree -> int array` qui à un arbre canonique associe son tableau d'entiers le représentant.

Question 5 Donner une fonction OCaml `val canonical: int array -> tree` qui à un tableau associe son arbre canonique.

Préfixe vers Suffixe

On se fixe le type suivant pour les arbres:

```
type 'a arb = | V | N of 'a * 'a arb * 'a arb
```

Question 1 Donner le code d'une fonction OCaml `let prefix: 'a arb -> 'a list` qui renvoie le parcours préfixe d'un arbre.

On cherche à modifier un arbre T en un arbre T' tel que le parcours préfixe de T est le parcours suffixe inversé de T'

Question 2 Donner un exemple d'un arbre T qui possède le même parcours préfixe que le parcours suffixe inversé. Donner l'exemple d'un arbre ou ils diffèrent.

Question 3 Donner le code `val to_suff: 'a arb -> 'a arb` tel que `to_suff t` donne un arbre T' tel que le parcours préfixe de T est le parcours suffixe inversé de T'

Question 4 Montrer la correction de votre algorithme

Arbres généraux

On considère les deux type d'arbres suivant en OCaml:

```
type bintree = | V | N of bintree * int * bintree
type tree = | N of int * tree list
```

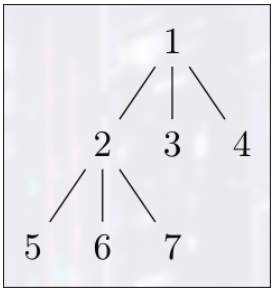
Les arbres `bintree` représente des arbres binaire tandis que `tree` représente des arbres généraux.

1. Proposer une fonction `val to_tree (a:bintree): tree` tel que `to_tree a` convertit un arbre binaire non vide en un `tree` représentant le même arbre.

[73] ENS 2023 MPI Info A

Pour transformer un arbre général en arbre binaire on utilisera la méthode *LCRS (Left child, right sibling)* décrite si dessous. Pour x un noeud, on note E_x la liste des enfants de x. Maintenant, pour x un noeud de parent N (avec donc $E_N[i] = x$) on aura que les deux enfants de x dans l'arbre binaire transformé seront à gauche $E_x[0]$ et à droite $E_N[i+1]$.

2. Donner la transformation de l'arbre suivant (on ordonne les enfant de droite à gauche) :



3. Programmer la fonction `val to_bintree (a:tree): bintree`
4. On note $\Delta(A)$ le nombre d'enfants maximal d'un des noeuds de A pour A un arbre général. Montrer que $h(\text{to_bintree}(A)) \leq \Delta(A) + h(A)$

Tableaux tri-coloré

Soit T un tableau de taille $2N$ dont les valeurs sont dans $\{0, 1, 2\}$. On dit que T est *tricoloré* si

$$\forall 0 \leq i < N, T[i] \neq T[2i+1] \neq T[2i+2] \neq T[i]$$

Dénombrer le nombre de tableau tricoloré de longueur $2^k - 1$

Permutation triable par pile[74]

Dans cet exercice on s'interdit d'utiliser les traits impératifs du langage OCaml (références, tableaux, champs mutables, etc.)

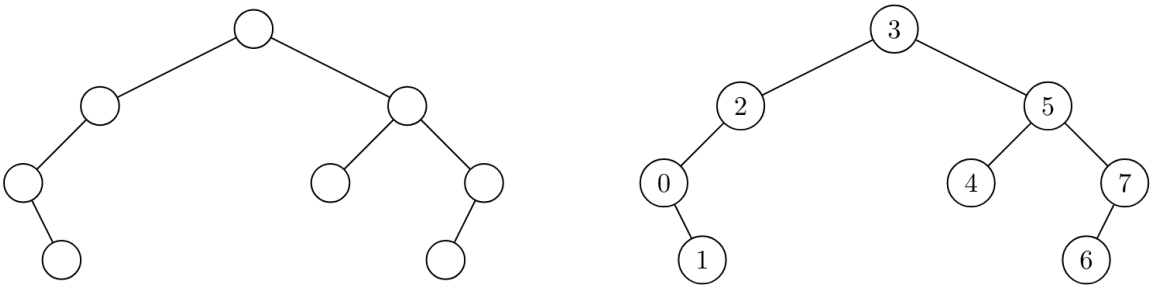
On représente en OCaml une permutation σ de $[\![0; n-1]\!]$ par la liste d'entier $[\sigma(0); \sigma(1); ...; \sigma(n-1)]$. Un arbre binaire étiqueté est soit un arbre vide, soit un nœud formé d'un sous-arbre gauche, d'une étiquette et d'un sous-arbre droit :

```
type arbre =
 | V
 | N of arbre * int * arbre
```

On représente un arbre binaire non étiqueté par un arbre binaire étiqueté en ignorant simplement les étiquettes. On étiquette un arbre binaire non étiqueté à n nœuds par $[\![0; n-1]\!]$ en suivant l'ordre infixe de son parcours en profondeur. La permutation associée à cet arbre est donnée par le parcours en profondeur par ordre préfixe.

Par example, pour l'arbre suivant (gauche), son étiquettage infixe est donné à droite, et la permutation associé est $[3; 2; 0; 1; 5; 4; 7; 6]$.

[74] Oral blanc MPI CCINP



1. Ecrire une fonction parcours `val prefixe : arbre -> int list` qui renvoie la liste des étiquettes d'un arbre dans l'ordre préfixe de son parcours en profondeur. On pourra utiliser l'opérateur `@` et on ne cherchera pas nécessairement à proposer une solution linéaire en la taille de l'arbre.
2. Ecrire une fonction `val etiquette : arbre -> arbre` qui prend en paramètre un arbre dont on ignore les étiquettes et qui renvoie un arbre identique mais étiqueté par les entiers de $[\![0; n-1]\!]$ en suivant l'ordre infixe d'un parcours en profondeur.
Indication : on pourra utiliser une fonction auxiliaire de type `arbre -> int -> arbre * int` qui prend en paramètres un arbre et la prochaine étiquette à mettre et qui renvoie le couple formé par l'arbre étiqueté et la nouvelle prochaine étiquette à mettre.

Une permutation σ de $[\![0; n-1]\!]$ est dite *triable avec une pile* s'il est possible de trier la liste $[\sigma(0), ..., \sigma(n-1)]$ en utillisant uniquement une structure de pile comme espace de stockage interne. On considère l'algorithme suivant, énoncé ici dans un style impératif :

> Initialiser une pile vide
> **Pour chaque** élément en entrée:
> **Tant que** l'élément est plus grand que le sommet de la pile:
> Dépiler le sommet de la pile vers la sortie
> **Fin tant que**
> Empiler l'élement en entrée dans la pile
> **Fin pour chaque**
> Dépiler tous les éléments restants dans la pile vers la sortie.

Par exemple, pour la permutation $[3; 2; 0; 1; 5; 4; 7; 6]$, on empile 3, 2, 0, on dépile 0, on empile 1, on dépile 1, 2, 3, on empile 5, 4, on dépile 4, 5, on empile 7, 6, on dépile 6, 7. On obtient la liste triée $[7; 6; 5; 4; 3; 2; 1; 0]$ en supposant avoir ajouté en sortie les éléments dans une liste. On admet qu'une permutation est triable par pile si et seulement cet algorithme permet de la trier correctement.

4. Dérouler l'exécution de cet algorithme sur la permutation associée à l'arbre exemple de la figure 1 et vérifier qu'elle est bien triable par pile
5. Ecrire une fonction `val trier : int list -> int list` qui implémente cet algorithme dans un style fonctionnel. Par exemple, trier $[3; 2; 0; 1; 5; 4; 7; 6]$ doit s'évaluer en la liste $[7; 6; 5; 4; 3; 2; 1; 0]$. On utilisera directement une liste pour implémenter une pile.
Indication : écrire une fonction auxiliaire de type `int list -> int list -> int list -> int list` qui prend en paramètre une liste d'entrée, une pile et une liste de sortie, et qui, en fonction de la forme de la liste d'entrée et de la pile, applique une étape élémentaire avant de procèder récursivement.
6. Montrer que s'il existe $0 \leq i < j < k \leq n-1$ tels que $\sigma(k) < \sigma(i) < \sigma(j)$, alors σ n'est pas triable par une pile.

7. On se propose de montrer que les permutations de $[\![0; n-1]\!]$ triables par une pile sont en bijection avec les arbres binaires non étiqueté à n nœuds.
   - Montrer que la permutation associée à un arbre binaire est triable par pile. On pourra remarquer le lien entre le parcours préfixe et l'opération empiler d'une part et le parcours infixe et l'opération dépiler d'autre part.
   - Montrer qu'une permutation triable par pile est une permutation associée à un arbre binaire. *Indication : on peut prendre $\sigma(0)$ comme racine, puis procéder récursivement avec les $\sigma(0)-1$ éléments pour construire le fils gauche et avec le reste pour le fils droit.*

Initialisation d'un Tas

On encode un tas max sous la forme d'un tableau d'entiers tel que les 2 fils de `T[i]` sont en position `T[2*i]` et `T[2*i+1]`

Question 1 Donner la fonction `void add_tas(int* T, int n, int k);` qui ajoute un entier `k` au tas `T`, représenté par un tableau de longueur `n`.

Question 2 Quelle est la complexité de cette fonction ? Proposer une meilleure structure que des tableaux pour avoir une complexité en moyenne $O(\log n)$

On cherche maintenant à initialiser un corriger un tableau `T` qui n'est **pas** un tas pour qu'il en devienne un.

Question 3 On suppose que pour tous les indices $j > i$, $T[j] \geq \max(T[2j], T[2j+1])$. Donner un algorithme `void correct(int* T, int n, int i);` qui corrige $T[i]$. Quelle est la complexité de cet algorithme ?

Pour i un noeud, on note $h(i)$ sa hauteur, donnée comme sa distance maximale aux feuilles, aka $h(T) - p(i)$ avec $p(i)$ la profondeur. On considère le code suivant:

```
void to_tas_2(int* T, int n){
  for(int i=n-1; i>=0;i--) {
    correct(T, n, i);
  };
};
```

Question 4 Montrer que

$$\sum_{0 \leq i < N} h(i) = \Theta(n)$$

En déduire que `to_tas_2` est en $\Theta(n)$

Mots d'arbres

On se fixe un ensemble fini de lettres $\Sigma = \{a, b\}$. On définit un *mot* comme étant une suite finie de lettres de Σ. Le mot vide (la suite vide) sera noté ε, et la concaténation de deux mots u, v sera noté par la concaténation uv. Pour w un mot, on note $|w|_a$ le nombre de a et $|w|_b$ le nombre de b. On définit par induction l'ensemble A par:

- $a \in A$
- Si $u, v \in A$, alors $buv \in A$

Question 1 Montrer que $\forall w \in A, |w|_a = 1 + |w|_b$

Question 2 Soit $w_1 ... w_n$ un mot de A, montrer que pour tout $i < n$, on a que $|w_1 ... w_i|_a \leq |w_1 ... w_i|_b$. En déduire que A est non ambiguë.

Question 3 Pour un $w \in A$, on essaye d'associer un arbre binaire T_w tel que T_{buv} soit l'abre contenant T_u et T_v comme enfants. Proposer un algorithme pour effectuer cette transformation.

Question 4 En déduire une manière de stoquer des arbres d'entiers positif sous la forme d'un tableau. Quel est l'avantage par rapport aux représentations que vous connaissez déjà ?

Ranger des boîtes

On cherche à trouver une structure de données efficace pour un magasin de vente de boîte en bois. Chaque boîte a deux paramètre: sa longueur (l) et sa largeur (w), les deux en centimètres.

Question 1 Proposer une définition d'un type `box_t` en C pour représenter une boîte.

On définit un type d'arbre de recherche de boîtes spécial:

```
struct box_arb {
  box_t* box;
  struct box_arb childs[4];
};
typedef struct box_arb box_arb_t;
```

Ou l'idée est que pour E un enfant de P :

- si $E = P.\text{childs}[0]$ alors $E.w < P.w$ et $E.l < P.l$
- si $E = P.\text{childs}[1]$ alors $E.w < P.w$ et $E.l \geq P.l$
- si $E = P.\text{childs}[2]$ alors $E.w \geq P.w$ et $E.l < P.l$
- si $E = P.\text{childs}[3]$ alors $E.w \geq P.w$ et $E.l \geq P.l$

Question 2 Donner un arbre équilibré contenant les boîtes de dimensions $2 \times 8, 4 \times 5, 4 \times 7, 5 \times 8, 7 \times 4$

Question 3 Proposer le code d'une fonction `bool is_box_arb(box_arb_t *arb);` qui teste si un arbre respecte la condition de l'arbre de recherche que l'on souhaite crée.

Question 4 Donner un code C pour effectuer la recherche d'une boîte b dans l'arbre. Quel est sa complexité ?

Question 5 Donner un algorithme pour obtenir la liste trié des boîtes par longueur à partir d'un T. Quelle est sa complexité ?

Parcours préfixe en $O(1)$ mémoire

On considère le type suivant pour un arbre binaire en C :

```
struct tree {
  struct tree *left;
  int v;
  struct tree *right;
};
typedef struct tree tree;
```

Proposer un algorithme en C `void get_prefixe(tree *arbre);` qui prend un arbre et qui `print` ses valeurs dans un parcours préfixe en $O(1)$ mémoire et $O(n)$ de temps. On pourra modifier (et meme détruire) l'arbre. Pour s'assurer que l'on soit bien en $O(1)$ de mémoire, on ne fera pas un programme récursif.

Parcours infixe en $O(1)$ mémoire

On considère le type suivant pour un arbre binaire en C :

```c
struct tree {
  struct tree *left;
  int v;
  struct tree *right;
};
typedef struct tree tree;
```

On note F l'arbre vide et $N(g, v, d)$ l'arbre ayant g comme fils gauche (respectivement d comme fils droit) et v comme valeur.

1. Donner un algorithme qui affiche le parcours infixe de l'arbre. Quelle est sa complexité en temps et mémoire?
2. Pour x un noeud de l'arbre, on dit que y est le précurseur de x si c'est le noeud qui arrive juste avant dans le parcours préfixe. Donner un code `tree* get_precurseur(tree* t);` qui prend un arbre et renvoie un pointeur vers le précurseur de la racine.
3. Justifier que la fonction de la question précédente est en $O(h(t))$ avec $h(t)$ la hauteur de l'arbre t.
4. Soit x un sommet, on suppose qu'il y a une erreur et que son précurseur y est tel que $y.d = x$. Pourquoi est-ce un problème?
5. Donner le code d'une fonction `void correct_problem(tree* t);` qui teste si `t` est un précurseur problématique (aka tel que `t.d==x`) et qui corrige ce problème si tel est le cas en mettant `t.d` à `null`.

On va chercher à faire de $O(1)$ mémoire et $O(n)$ temporel. Pour cela, on considère l'algorithme suivant :

Calcul_Infixe(T):
current $\longleftarrow T$
Tant que current $\neq F$:
    On écrit current $= N(g, v, d)$
    Si $g = F$ **alors:**
        Afficher v
        $x \longleftarrow d$
        CorrectProblem(current)
        current $\longleftarrow x$
    Sinon:
        $x \longleftarrow$ GetPrecurseur(current)
        $x.d \longleftarrow$ current
        current $\longleftarrow g$

6. Montrer que l'algorithme est correct et termine.
7. Montrer que l'arbre à la fin de l'algorithme est le meme que au début.
8. Montrer que chaque arrete n'est visité que au plus 3 fois. En déduire qu'il est en $O(n)$ de temps et $O(1)$ de mémoire.

Arbre et oracle

Soit $A = (S, E)$ un graphe représentant un arbre.

On se fixe une fonction $f : \mathcal{P}(S) \longrightarrow \{\text{true}, \text{false}\}$ qui à tout sous-ensemble de sommets de S qui **forment une composante connexe** renvoie si oui ou non un certain sommet s fixé à l'avance, inconnu pour nous, y appartient.

1. Donner un algorithme qui en $O(|S|)$ appels à f trouve le sommet s.

2. Donner un algorithme qui en $O(\log|S|)$ appels à f trouve le sommet s dans le cas ou l'arbre est un arbre binaire parfait.
3. Donner un algorithme qui en $O(\log|S|)$ appels à f trouve le sommet s dans le cas général.

Indépendant dans les arbres

On se donne en OCaml le type d'arbre suivant stockant des entiers sur chaque noeuds (que on apellera poids) :

```
type abr = F | N of int * abr * abr;;
```

On dit qu'un ensemble de noeuds S d'un arbre est indépendant si aucun n'est enfant direct d'un autre. Autrement dit, pour tout $p \in S$, on a les enfants de p qui ne sont pas dans S.

On dit qu'il est fortement indépendant si pour tout $p \in S$, on a tout le sous arbre de p qui n'est pas dans S.

Question 1 Donner une fonction OCaml `val get_max: abr -> int` qui trouve le poids maximal et le retourne.

Question 2 Montrer que le poids maximal d'un ensemble indépendant est supérieur ou égal au poids maximal d'un ensemble fortement indépendant

Question 3 Proposer un algorithme OCaml qui calcule le poids maximal d'un ensemble fortement indépendant.

Question 4 Proposer un algorithme OCaml qui calcule le poids maximal d'un ensemble indépendant.

Arbre α-équilibré

Pour T un arbre, on note $g(T)$ et $d(T)$ respectivement son fils gauche et droit. On note $|T|$ son nombre de noeud. On dit qu'un arbre binaire de recherche est α-équilibré si pour tout noeud x, on a

$$|g(x)| \geq \alpha|d(x)| \text{ et } |d(x)| \geq \alpha|g(x)|$$

Le type `bintree` représente les arbres usuels. On représente cette donnée par le type suivant en OCaml tel que $\text{EN}(g, x, n, d)$ corresponde à l'arbre T stockant x à la racine avec $|T| = n$, $g = g(T)$ et $d = d(T)$:

```
type 'a bintree = | F | N of 'a bintreee * 'a * 'a bintree
type 'a etree = | EF | EN of 'a etree * 'a * int * 'a etree
```

Question 1 Donner le code d'une fonction `val to_etree: 'a bintree -> 'a etree` qui converti un arbre binaire de recherche en un `etree`. Pour l'instant l'on ne ce soucie pas de la condition d'α-équilibrage

Question 2 Donner le code d'une fonction `val is_balanced: float -> 'a etree -> bool` telle que `is_balanced alpha t` renvoie `true` si t est α-balanced et `false` sinon.

On suppose que on possède une fonction

```
val join: float -> 'a etree -> 'a -> 'a etree -> 'a etree
```

telle que `join alpha t1 x t2` renvoie l'abr α-équilibré contenant les noeuds de t_1, t_2 et x, pour t_1, t_2 deux abr α-équilibré et x plus petit que tous les élements de t_2 et plus grand que tous les éléments de t_1.

Question 3 Donner le code d'une fonction `let split (alpha:float) (t:'a etree) (x:'a etree): 'a etree * 'a etree` telle que pour $\alpha \in]0; 1[$ et t un abr α-équilibré, `split alpha t x`

renvoie un couple de deux abr α-équilibré contenant respectivement tous les éléments plus petits que x et tous les éléments plus grand que x.

Question 4 En supposant que `join` est en $O(\log(|t_1| + |t_2|))$, quel est la complexité de `split` ?

Question 5 (∗) Pour $\alpha \in]\frac{2}{11}; 1 - \frac{1}{\sqrt{2}}[$, donner le code de `join`.

MP2I: Graphes (introduction)

Cours

- Définition d'un graphe $G = (V, E)$ avec $E \subseteq \mathcal{P}_2(V)$ ou $E \subseteq V^2$ symmétrique.
- Graphe orienté avec $E \subseteq V^2$.
- Arretes, sommet, degré, degré sortant/entrant, chemin, cycle, connexité, distance, boucle
- Graphes pondéré sur les sommets, sur les arêtes
- Graphes simples comme sans boucles connexes (HP)
- Multigraphes et Hypergraphes (HP)
- Marche et circuit comme chemin/cycle ou l'on peut repasser par le meme sommet (HP)
- Arbre comme graphes connexes acyclique
- Parcours en largeur, parcours en profondeur
- Algorithme de Dijsktra: poids positifs
- Algorithme de BellmanFord: poids négatif (HP)
- Algorithme de Floyd-Warshall: poids négatif & toutes les paires
- Graphes induits, sous-graphes (HP)
- Coloration, cliques, indépendants (HP)
- Graphes biparti

Questions de Cours

- Montrer que la somme des degré est 2 fois le nombre d'arêtes
- Montrer qu'un graphe est biparti ssi il ne contient pas de cycle de taille impairs
- Montrer qu'un graphe G acyclique est tel que $|A| \leq |V| - 1$
- Montrer qu'un graphe G connexe est tel que $|A| \geq |V| - 1$
- Montrer qu'un graphe G est un arbre ssi il est connexe et $|A| \leq |V| - 1$ ssi il est acyclique et $|A| \geq |V| - 1$
- Donner le pseudo-code de Floyd-Warshall
- Donner le pseudo-code de Dijsktra

Exercices à rajouter

- https://arxiv.org/abs/1904.01210 Implémentation incorrecte de Floyd-Warshall
- Graphes de flot !

Autour des degrés

Soit $G = (S, A)$ un graphe sans boucle. Montrer que

- $2|A| = \sum_{v \in V} \deg(v)$
- Le nombre de sommet de degré impair est pair
- Il existe forcément deux sommets de même degré.
- Soit A un arbre ou chaque arc est orienté d'un noeud vers ses enfant. On note N l'ensemble des noeuds qui ne sont pas des feuilles et f le nombre de feuilles. Montrer que

$$\sum_{x \in N} (\deg^+(x) - 1) = f - 1.$$

  En déduire une relation dans le cadre des arbres binaire.

Base des graphes connexes

Soit G un graphe connexe.

- Montrer que deux chemins de longueur maximale s'intersectent.
- Montrer qu'il existe toujours un sommet v tel que $G - v$ reste connexe.

Le puits et le roi

Soit $G = (V, E)$ un graphe orienté *simple*, c'est à dire que $\forall (u, v) \in E, (v, u) \notin E$. On dit que $p \in V$ est un puits si pour tout $u \in V \setminus \{p\}$, il y a une arête de u dans p (dans ce sens).

Question 1 Montrer qu'un tel puits est unique.

Question 2 On donne G par une matrice d'adjacence de taille $|V| \times |V|$. Proposer un algorithme en C qui calcule le puits s'il existe, ou qui renvoie -1 sinon.

Question 3 Montrer que l'on peut faire un tel algorithme en $O(|V|)$.

On dit qu'un graphe orienté est un *tournoi* si pour tout $u, v \in V$, on a soit $(u, v) \in E$ soit $(v, u) \in E$, mais pas les deux. On dit que dans un tournois, x est le *roi* si tous les sommets sont à distance d'au plus 2 de x.

Question 4 Montrer qu'un tel roi existe toujours, et proposer un algorithme en C qui prend une matrice d'adjacence et qui le calcule.

Graphes à unique sortie

On dit que $G = (V, E)$ un graphe orienté est un *graphe à unique sortie* si $\forall v \in V, \deg_+(v) = 1$. Dans ce cas, on le représentera par un tableau T de longueur $|V|$ tel que $T[i]$ corresponde à l'unique sommet $u \in V$ tel que $(i, u) \in E$.

Question 1 Dessiner le graphe correspondant au tableau $[0, 2, 1, 2, 4, 1, 3]$

Question 2 Proposer un algorithme en C qui prend en argument une matrice d'adjacence représenant un graphe à unique sortie et qui renvoie le tableau T le représentant. La signature de la fonction sera `int* get_table(int **adj, int n);`

Question 3 Montrer que pour tout sommet x, il existe un chemin allant de x à un sommet dans un cycle.

Question 4 Proposer un algorithme en C qui à partir d'un tableau et de 2 élements, décide s'ils sont dans la même composante connexe. On proposera un algorithme en $O(C_x + C_y)$ avec C_v le nombre de sommet que l'on peut atteindre à partir de v

Question 5 Montrer qu'il existe un algorithme en $O(C_x + C_y)$ de temps et $O(1)$ d'espace.

Reconnaitre les graphes biconnecté en $O(|V| + |E|)$

On dit qu'un graphe connexe $G = (V, E)$ est *biconnecté* si quel que soit $x \in V$, si on retire x à G alors le graphe reste connexe. On cherche à obtenir un algorithme en $O(|E| + |V|)$ pour tester si G est biconnecté.

Soit $T = (V_T, E_T)$ l'arbre d'un DFS d'un graphe $G = (V_G, E)$.

Question 1 On note $T(x)$ le sous-arbre de racine $x \in V_G$. Montrer que $x \preceq y \iff x \in T(y)$ est une relation d'ordre.

Question 2 Montrer que pour tout $(x, y) \in E_G$, on a que $x \preceq y$ ou $y \preceq x$.

On dit que $(x, y) \in E_G$ est une *arête de retour* si $(x, y) \notin E_T$ avec $x \preceq y$.

Question 3 Montrer que G est biconnecté ssi pour tout $x \in V_G$:
- Si x est la racine alors il existe une arête de retour de la forme (y, x).
- Sinon, il existe une arête de retour de la forme (y, x') avec $y \preceq x \prec x'$ (strict)

Question 4 En déduire un algorithme en $O(|E| + |V|)$ pour tester si un graphe est biconnecté.

Algorithme de Johnson[75]

Soit $G = (S, A)$ un graphe orienté et $w : S \times S \longrightarrow \mathbb{R}$ une pondération des poids potentiellement négative. On va considérer ici un algorithme de calcul des plus courts chemins intitulé *algorithme de Johnson*

1. Rapeller la complexité de l'algorithme de Dijkstra

Soit $h : S \longrightarrow \mathbb{R}$, on pose $w_h(u, v) = w(u, v) + h(u) - h(v)$.

2. Montrer que tous les plus courts chemins pour w sont les memes que pour w_h et qu'il existe un cycle de poids négatif pour w si et seulement si il en existe un pour w_h
3. On suppose que G ne possède pas de cycle à poids négatifs. Trouver un $h : S \longrightarrow \mathbb{R}$ tel que $\forall a, b \in S, w_h(a, b) \geq 0$. On pourra considerer le fait d'ajouter un sommet r relié à tous les autres sommets par un arc de poid nul.
4. En déduire un algorithme permettant de calculer tous les plus courts chemins entre tous les sommets. Comparer sa complexité avec Floyd-Warshall.

Reconnaitre un DFS

On se donne $G = (V, E)$ un graphe et T un arbre couvrant de G. On cherche à tester en temps linéaire si T est un graphe qui peut être obtenu en faisant un DFS de G.

Pour T un arbre enraciné de G, on note $x \underset{T}{\preceq} y$ si x est un enfant de y

1. Montrer que $\underset{T}{\preceq}$ est une relation d'ordre
2. Montrer que si T est un DFS, alors $\forall (x, y) \in E(G) \setminus E(T), x \underset{T}{\preceq} y \vee y \underset{T}{\preceq} x$
3. Montrer la réciproque de la question précédente
4. En déduire un algorithme en $O(|V|)$ qui teste si un arbre T est un DFS d'un graphe

Conversions de DAG

Un graphe orienté $G = (V, E)$ sans boucle est un DAG (directed acyclic graph) si c'est un graphe acyclique simplement connexe.

On se donne les types suivants en OCaml:

```
type 'a dag = ('a * int list) array;
type 'a tree = N of 'a * tree list;
```

On supposera écrite la fonction `val count: 'a tree -> int` qui compte le nombre de noeud dans un arbre.

Question 1 Crée une fonction `val get_nb: () tree -> int tree` tel qu'un noeud soit étiqueté par un i s'il est le i-ème noeud dans le parcours préfixe.

Question 2 En déduire une fonction qui converti un arbre en un dag.

Dans un DAG, on appelle un sommet x tel que $\deg^-(x) = 0$ *une source* et un un sommet x tel que $\deg^+(x) = 0$ *un puits*.

[75] Mines Télécom 2024

Question 3 Donner une fonction `val get_sources: int dag -> int list` qui renvoie la liste de tous les puits.

On supposera écrite la même fonction pour les puits.

Question 4 Proposer un algorithme `val get_shortest: int dag -> int` qui prend un DAG pondéré par des entiers et qui renvoie le chemin le plus court entre une source et un puits.

Chemin hamiltonien dans un tournois

On dit que $G = (V, E)$ un graphe orienté sans boucle est un *tournois* si pour tout $u, v \in V$ distincts, on a exactement une des deux arêtes (u, v) ou (v, u).

Question 1 Proposer le code d'une fonction C `bool is_tournois(int **graph, int n);` qui teste si un graphe `graph` ayant `n` sommets représenté par matrice d'adjacence est bien un tournois.

Un chemin $s_1, ..., s_n$ avec $n = |V|$ est dit *hamiltonien* s'il passe exactement 1 fois par chaque sommet, avec $\forall i < n, (s_i, s_{i+1}) \in E$.

Question 2 Proposer le code d'une fonction C `bool is_hamiltonien(int **graph, int n, int *chemin);` qui teste si `chemin` est un tableau représentant bien un chemin hamiltonien du graphe `graph` ayant `n` sommets (de 0 à n-1) représenté par matrice d'adjacence.

Question 3 Montrer qu'un tournois possède toujours un chemin hamiltonien.

Question 4 Proposer un algorithme en C `int* get_hamiltonien(int **graph, int n)` qui renvoie un chemin hamiltonien. Quel est sa complexité ?

Question 5 Montrer que le graphe ne contient pas de cycle de longueur 3 ssi il possède un unique chemin hamiltonien.

Degrée minimal[76]

On représente un graphe orienté $G = (S, A)$ avec $n = |S|$ par liste d'adjascence implémenté par des tableaux en C comme suit:

```
struct graph_s {
  int n ;
  int* degree;
  int** voisins;
};
```

On aura `degre` et `voisins` qui sont deux tableaux de taille n. Pour tout i, `degree[i]` représente le degrée sortant du sommet i et on a `voisins[i]` qui est le tableau de taille `degree[i]` indiquant les voisins de `i` (n'incluant pas lui-meme).

1. Donner le code en C d'une fonction qui calcule le degrée sortant maximal du graphe.

Pour $s \in S$, on pose $\mathcal{A}(s)$ l'ensemble des sommets accesible depuis s. Pour $X \subseteq S$ on note $d^*(X) = \max\{d^+(s) : s \in X\}$. Pour $x \in S$ on note $d^*(x) = d^*(\mathcal{A}(x))$. On cherche à calculer $d^*(s)$ pour chaque sommet.

2. Ecrire une fonction en C `bool* accesible(int s, struct graph_s g)` qui renvoie un tableau `t` tel que `t[i]` est `true` ssi `i` est accesible depuis `s`.
3. Ecrire une fonction en C `int* d_star_table(struct graph_s g);` qui calcule le tableau des $d^*(s)$ pour tout s. Quel est sa complexité?

[76] CCINP sujet 0

4. On suppose que G est acyclique. A l'aide d'un tri topologique, proposer un algorithme qui calcule le tableau des $d^*(s)$ en $O(|S| + |A|)$
5. On ne suppose plus le graphe acyclique. Décrire un algorithme permettant de calculer le tableau des $d^*(s)$ en $O(|S| + |A|)$

Graphes d'Halin[77] sont 3-connexes

On dit que $G = (V, E)$ est un graphe d'halin si il peut se décomposer comme $E = C \sqcup T$ (disjoint) avec T un arbre sans noeuds de degré 2 et C un cycle passant par toutes les feuilles.

Un graphe est dit *3-connecté* si quel que soit $u, v \in V$, $G - u - v$ le graphe obtenu en retirant u, v et leur arêtes incidentes est toujours connecté.

1. Montrer que si il existe 3 chemins disjoints entre toute paire de sommets alors le graphe est 3-connecté.
2. En déduire que les graphes de Halin sont 3-connexes

Graphes d'Halin[78] sont hamiltoniens

On dit que $G = (V, E)$ est un graphe d'halin si il peut se décomposer comme $E = C \sqcup T$ (disjoint) avec T un arbre sans noeuds de degré 2 et C un cycle passant par toutes les feuilles.

On dit qu'un graphe G est hamiltonien si il existe un cycle $\langle v_1, ...v_n \rangle$ passant par tous les sommets une fois

1. Montrer que tout arbre sans noeuds de degré 2 peut être obtenue en itérativement transformant une feuille en un sommet de degré k avec $k - 1$ feuilles attaché.
2. Montrer que les graphes d'Halin sont hamiltonien.

Dans les vrais graphes d'Halin, si on retire un sommet alors il reste hamiltonien.

Nombre cyclomatiques[79]

Si $G = (S, A)$ est un graphe à k composantes connexes, on définit alors son *nombre cyclomatique* $c(G) = |A| - |S| + k$

1. Quel est le nombre cyclomatique d'un arbre ?
2. Soit G un graphe acyclique. Donner et prouver son nombre cyclomatique.
3. Montrer que si $c(G) = 0$ alors G possède un sommet de degré inférieur ou égal à 1.
4. Est-ce que la réciproque de la question 2 est vraie ?

Calcul des triangles

Pour $G = (V, E)$ un graphe avec $V = \{1, ..., n\}$, on dit que $\{x, y, z\} \subseteq V$ est un *triangle* si $\{x, y\}, \{y, z\}, \{z, x\} \in E$. On cherche à calculer tous les triangles sans doublons d'un graphe G.

1. Proposer un algorithme en $O(|V|^3)$
2. Soit L_1, L_2 deux listes trié d'entiers de 1 à n, montrer que l'on peut calculer la liste triée des élément appartenant aux deux listes en temps $O(|L_1| + |L_2|)$.
3. Donner un algorithme en $O(|E| \times \Delta)$ pour calculer tous les triangles sans doublons d'un graphe G ou Δ est le degré maximal de G. On supposera que G est donné sous la forme d'une liste d'adjacence.
4. Dans quels cas est-ce que l'algorithme de la question 4 est meilleur que celui de la question 2 ?

[77] On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

[78] On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

[79] Tiré du TD de la martinière de MPI

Graphes orienté semi-connexes[80]

Soit $G = (S, A)$ un graphe orienté. On dit que G est *semi-connexe* si pour toute paire de sommets $s, t \in S$, soit il existe un chemin de s à t soit il existe un chemin de t à s.

1. Donner un exemple d'un graphe simplement connexe non semi-connexe.
2. On suppose que G possède un tri topologique $(s_1, ..., .s_n)$ de ses sommets. Montrer que G est semi-connexe ssi $\forall i \in [\![1; n-1]\!], (s_i, s_{i+1}) \in A$
3. Dans quel cas un graphe n'admet pas de tri topologique ?
4. Donner/Rapeller un algorithme pour calculer un tri topologique.
5. En considérant une décomposition en composante fortement connexe, proposer un algorithme pour tester si un graphe est semi-connexe.

Graphes Cordaux

Un graphe $G = (V, E)$ est dit cordal si tout cycle de longeur ≥ 4 possède une *corde*, c'est à dire une arrete qui relie deux sommets non consécutif du cycle.

On représente G par une matrice d'adjacence `bool adj[n][n]`.

1. Indiquer pour chacun des graphes suivants s'il sont cordaux ou non :
   - K_5, Le graphe complet à 5 éléments
   - C_5 le cycle de longeur 5
   - P_5 le chemin de longeur 5
   - Le carré C_4 avec une des deux diagonales.
2. On dit qu'un sommet v est *simplicial* si l'ensemble de ses voisins forme une clique. Donner une fonctioon `bool is_simplicial(bool **adj, int n, int v);` qui teste si un sommet est simplicial ou non.
3. Montrer que si v est simplicial et si $G - v$ est cordal, alors G est cordal.

On admet le théorème suivant : Tout graphe cordal non vide possède un sommet simplicial.

4. En déduire un algorithme `bool is_chordal(bool **adj, int n);` de reconnaissance des graphes cordaux. On donnera sa complexité.
5. Prouver la correction de l'algorithme.
6. Comparer à la complexité de l'algorithme naif.
7. Un ordre $v_1, ..., v_n$ des sommets est appelé *ordre d'élimination parfait* si, pour tout i, le sommet v_i est simplicial dans le sous-graphe induit par $\{v_i, ..., v_n\}$. Montrer qu'un graphe est cordal si et seulement s'il possède un ordre d'élimination parfait, et en déduire le théorème.

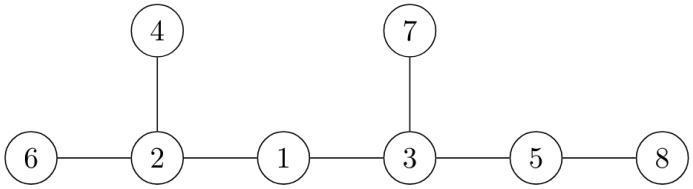
Codage de Prufer[81]

Soit $G = (S, A)$ un abre dont les sommets sont numérotés de 1 à $|S|$, on définit son *codage de prufer* comme la suite de sommets de longueur $|S| - 2$ qui est donné par cet algorithme:

[80] Tiré du TD de la martinière de MPI
[81] Tiré du TD de la martinière de MPI

$L \longleftarrow [\,]$
tant que $|S| > 2$:
    $s \longleftarrow \min\{s \in S \mid \deg(s) = 1\}$
    $s' \longleftarrow$ l'unique $s' \in S$ tel que $\{s, s'\} \in A$
    Ajouter s' en fin de L
    Retirer s et toutes ses arêtes à G
fin tant que
renvoyer L

1. Montrer le fait la ligne $s \longleftarrow \min\{s \in S \mid \deg(S) = 1\}$ ne pose pas de problème et montrer la terminaison de l'algorithme.
2. Donner un codage de prufer de l'arbre suivant:



3. L'opération qui à un arbre sans étiquettes associe un arbre de prufer est-elle injective ? Justifier
4. Donner un algorithme qui prend un codage de prufer et qui renvoie l'arbre dont c'est le codage
5. En déduire le cardinal du nombre d'arbres à n sommets étiqueté de 1 à n

MP2I: Logique Propositionelle

Cours

- Définition des formules propositionnelles par induction avec $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \bot, \top, x \in \mathcal{V}$
- Valuation, valeur de vérité, table de vérité
- Transformer une formule en forme normale conjonctive, en forme normale disjonctive
- Le problème SAT et k-SAT
- L'algorithme de Quine
- L'opérator xor $\oplus$ et l'anneau $((\mathbb{Z}/2\mathbb{Z})^n, \oplus, \wedge)$ (HP)

Questions de Cours

- Rapeller l'algorithme de Quine
- Rapeller la table de vérité de chacun des opérateurs $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
- Pour chacune des formules suivantes, être capable de donner
  - La forme normale conjonctive,
  - La forme normale disjonctive,
  - La table de vérité,
  - Une valuation la satisfaisant si possible,
  - Une valuation la disprouvant si possible.

  Les formules sont: $A \rightarrow (A \wedge B); (B \vee A) \rightarrow B; A \leftrightarrow (\neg A \wedge B), A \rightarrow (C \rightarrow B); \neg(A \leftrightarrow \neg A)$

Questions à rajouter

- BDD & d-DNNF
- d-DNNF are DAG circuit with not only on leafs. d-DNNF are more succint than BDD (hard proof)
- Circuit language
- Weighted model counting

- Weighted model counting in d-DNNF can be done as and -> product, or -> sum
- Every BDD is a d-DNNF: a node A is a "if A then X else X'" so "A and X or not A and X'".
- Links with top down compilation and quine's algorithm: top down is branching on original program

Equivalence

1. Expliciter une bijection φ entre $[\![0; 2^n - 1]\!]$ et $\{\top, \bot\}^n$

On représente la table de vérité d'une formule F à n variables par un tableau de booléens (d'entiers) de longueur 2^n, tel que la i-ème valeur correspond à $\varphi(i)(F)$.

2. Proposer un code qui à une formule associe sa table de vérité.
3. Proposer un code qui teste si deux formules sont équivalentes. Quel est sa complexité ?

Retirer les constantes

On se donne le type suivant en OCaml d'un arbre de syntax d'une formule propositionnelle:

```
type form =
  | Var of string | Top | Bot
  | Not of form
  | And of form * form | Or of form * form | Implies of form * form
```

Ici `Top` représente $\top$, le vrai, et `Bot` représente $\bot$, le faux. Ce sont les deux "constantes".

1. Donner une tautologie et une antilogie n'utilisant pas $\top$ ou $\bot$.
2. Donner le code d'une fonction OCaml `val rem_cst: form -> form` qui renvoie une formule équivalente qui n'utilise aucune constante.

On cherche maintenant à *propager les constantes*, c'est-à-dire à effectuer cette transformation tout en diminuant la taille de la formule.

3. A quoi sont équivalent les formules $A \wedge \top$ et $A \wedge \bot$? Faire de même pour chaque opérateur.
4. En déduire une fonction OCaml `val propage_cst: form -> form` qui renvoie une formule équivalente en propageant les constantes. *On aura le droit de retourner $\top$ ou $\bot$ directement, mais ils ne devront pas apparaitre dans des sous-formules.*

Vers le 3-Fnc

On définit une formule logique en forme normale conjonctive par le type suivant:

```
type lit = Pos of int | Neg of int
type fnc = lit list list
```

Les variables propositionnelles sont indexées et représentées par des entiers.

1. Mettre sous la forme normale conjonctive la formule $\neg(X_1 \vee (\neg X_2 \wedge X_3))$
2. Donner le code OCaml d'une fonction `val new_var: fnc -> int` qui renvoie un nom de variable propositionnel non utilisé.
3. Donner le code OCaml d'une fonction `val to_3: fnc -> fnc` qui prend une formule en forme normale conjonctive et qui renvoie une formule équivalente ou chaque clause possède au plus 3 littéraux. On aura le droit d'introduire de nouvelles variables non utilisée initialement.

Push down not

On se donne le type suivant en OCaml d'un arbre de syntaxe d'une formule propositionnelle:

```
type form =
  | Var of string
```

```
  | Not of form
  | And of form * form | Or of form * form | Implies of form * form
```

On dit qu'une formule est *sans non intérieur* si les seuls non sont autour de variables propositionnelle.

1. Transformer la formule $\neg(A \vee \neg B) \wedge \neg(C \wedge \neg D)$ en une version sans non intérieur
2. Montrer que toute formule est équivalente à une formule sans non intérieur.
3. Donner un programme effectuant cette transformation.

Systèmes Complet de Connecteurs[82]

On dit qu'un système de connecteurs logiques est *complet* si toute formule propositionnelle est logiquement équivalente à une formule propositionnelle écrite uniquement à l'aide de ces connecteurs et de $\top, \bot$.

1. Justifier que l'ensemble $\{\wedge, \vee, \neg\}$ forme un système complet de connecteurs
2. Les ensembles suivants forment-ils un système complet de connecteurs ? Justifier.

$\{\wedge, \neg\}$ $\{\vee, \wedge\}$ $\{\rightarrow, \neg\}$ $\{\rightarrow\}$

On introduit deux nouveaux connecteurs logiques pour écrire des formules propositionnelles:

- Le « OU exclusif » (ou XOR), noté $\oplus$, définit par $\nu(\psi_1 \oplus \psi_2) = V$ si et seulement si $\nu(\psi_1) \neq \nu(\psi_2)$.
- Le « NON-ET » (ou NAND), noté $\uparrow$, définit par $\nu(\psi_1 \uparrow \psi_2) = V$ si et seulement si $\nu(\psi_1) = \nu(\psi_2) = F$.

3. L'ensemble $\{\oplus, \neg\}$ forme t'il un système complet de connecteur ?
4. Montrer que $\{\uparrow\}$ forme un système complet de connecteurs.

Formule pour la bi-partition

Soit G = (V, E) un graphe fini orienté. Pour deux sommets u et v, on définit la variable propositionnelle X_{uv} qui indique s'il y a une arête entre u et v ou non. Pour toute valuation λ, on pose $G_\lambda = (V, E_\lambda)$ le graphe tel que $(u, v) \in E_\lambda$ si et seulement si $\lambda(X_{uv}) = \top$ (est vrai).

Montrer qu'il existe une formule propositionnelle ψ telle que pour tout valuation λ, $\lambda(\psi) = \top$ si et seulement si G_λ est biparti.

Formule pour le centre d'un graphe

Soit G = (V, E) un graphe fini orienté. Pour deux sommets u et v, on définit la variable propositionnelle X_{uv} qui indique s'il y a une arête dirigée de u vers v ou non. On appelle centre un sommet de G tel que tout sommets de G soit à distance au plus 2 du centre. Pour toute valuation λ, on pose $G_\lambda = (V, E_\lambda)$ le graphe tel que $(u, v) \in E_\lambda$ si et seulement si $\lambda(X_{uv}) = \top$.

Montrer qu'il existe une formule propositionnelle ψ telle que pour tout valuation λ, $\lambda(\psi) = \top$ si et seulement si G_λ admet un centre.

Formules linéaire

On dit qu'une formule F de la logique propositionnelle est *linéaire* si chaque variable propositionnelle apparaît au plus une fois.

1. Montrer que si F est une formule propositionnelle linéaire qui n'utilise pas $\bot$, alors il existe une valuation μ telle que $\mu \vDash F$

[82] Exercice de Maxime Bridoux

2. Dans le cas ou F est linéaire, proposer un algorithme polynomial qui compte le nombre de valuations satisfesant F

Equivalences avec Sheffer

On se donne le type suivant en OCaml d'un arbre de syntaxe d'une formule propositionnelle:

```
type form =
  | Var of string
  | Not of form
  | And of form * form | Or of form * form | Implies of form * form
```

On pose l'opérateur de Sheffer $A \uparrow B := \neg(A \wedge B)$.

1. Montrer que $A \to B \equiv A \uparrow (B \uparrow B)$
2. Montrer que n'importe quel formule est équivalente à une formule n'utilisant que l'opérateur de Sheffer $\uparrow$

On pose le type suivant en OCaml:

```
type sheffer =
 | Sheff of sheffer * sheffer
 | SVar of string
```

3. Donner le code OCaml d'une fonction `val to_sheff: form -> sheffer` qui à une formule retourne une formule équivalente n'utilisant que l'opérateur de Sheffer.
4. Est-ce que le résultat est polynomial en la taille de l'entrée ? Si ce n'est pas le cas, proposer, en ajoutant de nouvelles variables non utilisées, une version polynomiale.

Equivalence avec ite[83]

Étant donné trois formules logiques φ, ψ, θ, on définit un opérateur ternaire f par

$$f(\varphi, \psi, \theta) := (\varphi \wedge \psi) \vee (\neg\varphi \wedge \theta)$$

1. Simplifier $f(\top, \psi, \theta)$ et $f(\bot, \psi, \theta)$ en donnant des formules équivalentes.
2. Exprimer en $\varphi \wedge \psi$ et $\varphi \vee \psi$ via une formule ne comportant qu'un seul f.
3. Exprimer $\neg\varphi$ en fonction de $f, \top, \bot$.

Indéterminé

On définit une sémantique non standard sur les opérateurs logique écrits avec $\wedge, \vee, \to$ et $\neg$ pour un ensemble de 3 valeurs: $\top$, $\bot$ et ?.

? représente une valeur que l'on ne connais pas encore.

Pour v une valuation et F une formule, on a:
- $v(F) = \top$ si, quel que soit F' qui est F où l'on a remplacé chaque ? par des valeurs $\top$ ou $\bot$ de manière indépendante, on a $v(F') = \top$
- $v(F) = \bot$ si, quel que soit F' qui est F où l'on a remplacé chaque ? par des valeurs $\top$ ou $\bot$ de manière indépendante, on a $v(F') = \bot$
- $v(F) =$? sinon

Ainsi, par exemple, on a :
- $\top \vee ? \equiv \top$ car $\top \vee \bot \equiv \top \vee \top \equiv \top$
- $? \to ? \equiv ?$ car $\top \to \top \equiv \top$ mais $\top \to \bot \equiv \bot$

1. Proposer des tables de vérité pour $\wedge$, $\to$ et $\neg$

[83] Exercice de Maxime Bridoux

2. Montrer que $p \vee \neg p$ n'est pas une tautologie pour cette sémantique. Est-ce qu'une tautologie existe?
3. Proposer un algorithme qui prend une formule indéterminée et une valuation et qui donne la valeur d'une formule indéterminée. Quelle est sa complexité?

Formules monotones

Une formule est monotone si elle ne contient pas de négation ni d'implications. On considère alors le type suivant en OCaml :

```
type prop_mono =
| Top | Bot
| Var of int
| Or of prop_mono * prop_mono
| And of prop_mono * prop_mono
```

On définit une relation $\preceq$ sur les valeures de vérité V, F par $F \prec V$ que l'ont étent aux valuations par $\mu \preceq \mu'$ si pour toute variable X on a $\mu(X) \preceq \mu'(X)$.

1. Montrer que $\preceq$ est une relation d'ordre.
2. Montrer que si une formule monotone est satisfaite par une valuation v, alors elle est satisfaite par toute valuation w telle que $v \preceq w$.
3. Est-ce que cela marche toujours si on rajoute les implications?
4. En déduire un algorithme en ocaml `val is_sat : prop_mono -> bool` pour tester si une formule monotone est satisfiable.
5. En déduire un algorithme en ocaml `val is_taut : prop_mono -> bool` pour tester si une formule monotone est une tautologie.

Transformation de Tseitin

On souhaite transformer une formule quelconque en une FNC satisfiable si et seulement si la formule de départ est satisfiable, de taille polynomiale.

```
type lit = Pos of int | Neg of int
type cnf = lit list list

type prop =
| Top | Bot
| Var of int
| Not of prop
| Or of prop * prop
| And of prop * prop
| Imp of prop * prop
```

Une clause est représentée par une liste de littéraux, et une FNC par une liste de clauses. Pour chaque sous-formule φ de F, on introduit une nouvelle variable X_φ. On cherche à imposer localement que

$$X_\varphi \leftrightarrow \varphi$$

1. Écrire une fonction `val max_var : prop -> int` qui renvoie le plus grand indice de variable apparaissant dans une formule. S'il n'y a aucune variable, on pourra renvoyer -1.
2. Pour trois variables propositionnelles z, a, b, transformer les formules suivantes en une FNC avec des clauses de taille au plus 3 :

$$z \leftrightarrow a \wedge b \qquad z \leftrightarrow a \vee b \qquad z \leftrightarrow \neg a \qquad z \leftrightarrow (a \to b)$$

3. Proposer un algorithme `val to_cnf : prop -> cnf` qui construit une FNC équisatisfiable à la formule donnée.
4. Montrer que la formule de sortie est de taille polynomiale en la taille de la formule d'entrée.

Formules duales

Étant donné une formule de la logique propositionnelle F, utilisant les symboles $\top, \bot, \neg, \wedge, \vee$ et sur l'ensemble de variables X. On définit F^* la formule *duale* obtenue en remplaçant les $\wedge$ par des $\vee$, les $\vee$ par des $\wedge$ et en inversant les $\top$ et $\bot$.

1. Montrer que si deux formules sont équivalentes, alors leurs duales le sont aussi.
2. En déduire que si une formule est une tautologie, sa formule duale est contradictoire.

XOR SAT

On ajoute à la syntaxe de la logique propositionnelle l'opérateur binaire XOR, noté $\oplus$, ayant pour sémantique $v \vDash \psi_1 \oplus \psi_2$ si et seulement si $v(\psi_1) \neq v(\psi_2)$.

1. Exprimer l'opérateur $\oplus$ en fonction des opérateurs classique
2. Donner un algorithme polynomial qui, étant donnée une formule de la forme $\psi = \bigwedge_{i=1}^{m} \bigoplus_{j=1}^{n} l_i^{(j)}$ (avec $l_i^{(j)}$ des littéraux de la forme $X, \neg X$ pour X une variable) décide si ψ est satifiable

Interpolation de Craig[84]

On note $\mathrm{Var}(\varphi)$ l'ensemble des variables propositionnelles apparaissant dans la formule φ. Étant donné deux formules propositionnelles φ, ψ telles que $\varphi \vDash \psi$, un *interpolant* est une formule propositionnelle θ telle que:

- $\varphi \vDash \theta$
- $\theta \vDash \psi$
- $\mathrm{Var}(\theta) \subseteq \mathrm{Var}(\varphi) \cap \mathrm{Var}(\psi)$

1. Donner un interpolant de $\varphi = (x \vee y) \wedge z$ et $\psi = (t \to x) \vee (t \to y)$
2. Soient φ, ψ deux formules propositionnelles et $X \in \mathrm{Var}(\varphi)$, on note $\varphi[X \leftarrow \psi]$ la formule où l'on a remplacé toutes les variables propositionnelles X par ψ. On pose $\varphi_1 := \varphi[X \leftarrow \psi]$ et $F_2 := \varphi[X \leftarrow \neg\psi]$. Montrer que $\varphi \to \varphi_1 \vee \varphi_2$ est une tautologie.

Soient φ, ψ deux formules propositionnelles telles que $\varphi \vDash \psi$.

3. Montrer que si $\mathrm{Var}(\varphi) \cap \mathrm{Var}(\psi) = \emptyset$, alors soit $\neg\varphi$ soit ψ est une tautologie.
4. Montrer que toutes formules propositionnelles φ, ψ vérifiants $\varphi \vDash \psi$ admettent un interpolant.

FNC-SAT vers CLIQUE

On cherche à montrer que 3-SAT est résoluble en temps polynomial si et seulement si le problème CLIQUE l'est.

Soit $F = \bigwedge_{1 \leq i \leq n} C_i$ une formule en FNC possédant n clauses, que l'on écrit $C_i = l_i^{(1)} \vee \ldots \vee l_i^{(k_i)}$.
On construit un graphe G_F de la manière suivante :

- pour chaque clause C_i et chaque occurrence d'un littéral $l_i^{(j)}$, on crée un sommet (i, j)
- on met une arête entre (i, j) et (i', j') si $i \neq i'$ et si les deux littéraux correspondants ne sont pas contradictoires, c'est-à-dire si $l_i^{(j)}$ n'est pas la négation de $l_{i'}^{(j')}$.

1. Dessiner le graphe correspondant à la formule

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge (\neg x_3 \vee x_1 \vee x_2).$$

[84] Un mix d'un exercice de Maxime Bridoux et d'un exercice du cours de logique de l'ENS de Lyon

Une *clique* $S' \subseteq S$ d'un graphe $G = (S, A)$ est un ensemble de sommets deux à deux reliés par une arête.

2. Montrer que si F est satisfiable par une valuation μ, alors on peut choisir un littéral vrai dans chaque clause de F de sorte que les sommets correspondants forment une clique de taille n dans G_F
3. Montrer que G_F possède une clique de taille n si et seulement si F est satisfiable.
4. En déduire que s'il existe un algorithme polynomial pour le problème CLIQUE, alors FNC-SAT est résoluble en temps polynomial.

On cherche maintenant à montrer la réciproque.

Soit $G = (S, A)$ un graphe non orienté et soit $k \in \mathbb{N}$. On veut décider si G possède une clique S de taille k. Pour cela, pour tout $p \in \{1, ..., k\}$ et tout sommet $s \in S$, on crée une variable propositionnelle $X_{p,s}$, dont l'interprétation est :

$$X_{p,s} = \text{le p-ième sommet choisi est s.}$$

5. Pour chacune des contraintes suivantes, proposer une formule FNC qui dépend seulement du graphe :
   - pour chaque position p, au moins un sommet est choisi
   - pour chaque position p, au plus un sommet est choisi
   - deux positions différentes ne choisissent pas le même sommet
   - deux sommets choisis à deux positions différentes sont toujours reliés par une arête.
6. Montrer que F_G est satisfiable si et seulement si G possède une clique de taille k.
7. Expliquer pourquoi F_G est de taille polynomiale en $|S|$ et k.
9. Conclure que FNC-SAT est résoluble en temps polynomial si et seulement si CLIQUE l'est.

Binary Decision Diagram

Soit $n \in \mathbb{N}^*$. On considère des variables booléennes $X_1, ..., X_n$.

Un *BDD ordonné* est un graphe orienté acyclique dont les feuilles sont étiquetées par `true` ou `false`, et dont les noeuds internes sont étiquetés par une variable. Depuis un noeud étiqueté par X_i, l'arête gauche correspond à $X_i = F$ et l'arête droite à $X_i = V$. On impose que, sur tout chemin de la racine vers une feuille, les variables apparaissent dans le meme ordre $X_1, ..., X_n$. La racine sera toujours 1. On signale que ce n'est pas toujours un arbre : il est possible que deux noeud différents pointent vers le meme fils.

On propose la représentation OCaml suivante :

```
type node =
  | Bot (* feuille étiqueté par false *)
  | Top (* feuille étiqueté par true *)
  | If of int * int * int
  (* If(i,j,k) veux dire "si Xi = V alors aller au noeud j sinon aller au noeud k" *)

type bdd = {
  n : int;
  root : int;
  nodes : node array;
}
```

1. Desiner le BDD associé à la formule $(x \wedge y) \vee (\neg y \vee z)$ pour l'ordre y, x, z.
2. Écrire une fonction `val eval : bool array -> bdd -> bool` qui teste si une valuation `val mu : bool array` satisfait le BDD.

3. Deux BDD B_1 et B_2 ayant le même ordre de variables étant donnés, expliquer comment construire un BDD représentant $B_1 \vee B_2$. On ne donnera pas l'algorithme en OCaml.
4. Montrer qu'il existe des BDD de taille polynomiale reconnaissant des formules de taille exponentielle.
5. Pour $n \geq 1$, on considère la formule suivante :

$$E_n := \bigwedge_{i=1}^{n} (X_i \leftrightarrow X_{i+n})$$

Montrer que toute représentation par BDD ordonné de E_n pour l'ordre $X_1 < ... < X_{2n}$ possède une taille exponentielle en n.
6. Montrer que l'on peut calculer en temps polynomiale en la taille du BDD le nombre de valuations satisfaisantes.

On peut utilliser les BDD pour calculer la probabilité qu'une formule soit vrai si on prend une valuation aléatoire uniformément.

Horn-SAT

On fixe $\mathcal{V} = \{x_1, ..., x_n\}$ un ensemble de variables propositionnelles. Une *clause de Horn* est une clause contenant au plus un littéral positif (qui est directement une variable sans négation).

1. Soit C une clause de Horn, montrer que C est équivalente à une formule de la forme suivante, avec $y_1, ..., y_k \in \mathcal{V}$ et $z \in \{\bot\} \cup \mathcal{V}$:

$$y_1 \wedge ... \wedge y_k \rightarrow z$$

On considère l'algorithme suivant :
- Convertir chaque clause comme vu à la question précédente
- Initialement, toutes les variables sont fausses
- Tant qu'il existe une implication $x_1 \wedge ... \wedge x_k \rightarrow y$ dont toutes les hypothèses sont vraies mais dont la conclusion y est fausse, on rend y vraie
- Si à un moment une implication $x_1 \wedge ... \wedge x_k \rightarrow \bot$ a toutes ses hypothèses vraies, on rejette
- Sinon, quand plus rien ne change, on accepte.

2. Exécuter l'algorithme sur la formule $F = x \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$.
3. Montrer que l'algorithme termine. Quelle est sa complexité?
4. Montrer la correction de l'algorithme.
5. En déduire que tester si une formule de Horn est satisfiable est décidable en temps polynomial.
6. Montrer que l'ensemble des valuations satisfaisant une formule de Horn est stable par intersection.

Formules Evasives[85]

On fixe un ensemble $X = \{x_1, ..., x_n\}$ de variables propositionnelle. Un *diagramme de décision* D sur X est la donnée d'un graphe orienté (V, E), supposé sans cycle, d'un nœud initial v_{init} et d'un ensemble d'états finaux $F = F_1 \sqcup F_0 \subseteq V$ non vide ne possédant aucune arête sortante.

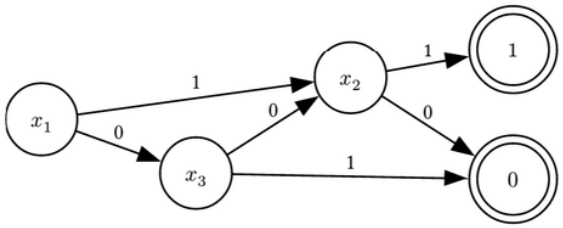
Chaque arrête est étiquetée par un booléen et chaque nœud par une variable de X. Chaque nœud non final possède exactement deux arêtes sortantes, une étiquetée par 1 et une par 0.

Pour toute *valuation* $\mu : X \longrightarrow \{1, 0\}$, un diagramme de décision D associe une valeur $D(\mu) \in \{1, 0\}$ obtenus en parcourant le graphe en commençant à v_{init} et à chaque sommet v, prendre l'arête

[85] Sujet d'ENS

étiqueté par $\mu(v)$, jusqu'à arriver dans F_1 (dans ce cas $D(\mu) = 1$) ou arriver dans F_0 (dans ce cas, $D(\mu) = 0$)

1. On considère $X = \{x_1, ..., x_n\}$ et le graphe de décision D_1 suivant ou l'état initial est x_1, les états finaux ont été entouré (F_1 correspond aux état avec un 1 et respectivement F_0 correspond au états avec un 0):



À quelle valeur est associée la valuation qui envoie (x_1, x_2, x_3, x_4) sur $(1, 1, 0, 1)$? sur $(0, 1, 0, 0)$?

2. Donner une formule de la logique propositionnelle équivalente à la fonction booléenne représentée par D_1.

Si on se donne une formule logique φ, on dit que D *représente* φ si φ est équivalente à la fonction booléenne représentée par D.

3. Donner un diagramme de décision D_2 représentant la fonction $\neg(x_1 \wedge x_2) \vee x_3$

La profondeur d'un diagramme de décision est la plus grande longueur possible d'un chemin orienté à partir du nœud initial v_{init}, en comptant le nombre de nœuds de traversés. On appelle profondeur minimale d'une fonction booléenne φ la plus petite profondeur possible pour un diagramme de décision représentant φ.

4. Décrire la profondeur du diagramme D_2 et celle du diagramme D_1.
5. Quelle est la profondeur minimale de la fonction identifiée en question 1?

Une fonction booléenne φ sur n variables est dite évasive si sa profondeur minimale est de n.

6. Donner un exemple d'une famille infinie de fonctions évasives, et justifier.
7. Pour tout $n \in \mathbb{N}$, on pose $\varphi_n := (x_0 \wedge x_1) \vee (x_1 \wedge x_2) \vee ... \vee (x_{n-1} \wedge x_n)$. Ces fonctions sont-elles évasives ? Justifier.

Compacité

Pour A un ensemble de formules de la logique propositionnelle, on dit qu'une valuation μ *satifiait* A si $\forall F \in A, \mu(F) = \top$

On dit que A est *finement satifiable* si pour tout $E \subseteq A$ fini, E est satifiable.

On pose $(X_n)_n$ une suite de variables propositionnelle.

1. Les ensembles suivants sont-ils satisfiables ?

- $A_1 = \{X_{2n} : n \in \mathbb{N}\} \cup \{\neg X_{2n+1} : n \in \mathbb{N}\}$
- $A_2 = \{X_i \vee \neg X_{i+1} : i \in \mathbb{N}\}$
- $A_3 = \{X_i \wedge \neg X_{i+1} : i \in \mathbb{N}\}$

On cherche à montrer le *théorème de compacité*: A est satisfiable ssi A est finement satifiable.

2. Montrer que si A est satifiable alors il est finement satifiable.
3. ($*$) Dans le cas ou l'ensemble des variables propositonnels de A est dénombrable, démontrer en construisant par récurrence une valuation que si A est finement satifiable alors A est satifiable.

4. Utilliser le théorème de compacité pour montrer qu'un graphe infini dénombrable est N coloriable ssi tout ses sous-graphes fini sont N coloriable.

MP2I: Bases de Données

Cours

Programme:

- Tables, relations, attributs / collonnes, domaine, schéma de table, enregistrements ou lignes, types de données
- Clé primaire, secondaire, étrangère
- Entités et association, association $1-1$, $1-*$, $*-*$. Séparation de $*-*$ en deux $1-*$
- `SQL` : SELECT, WHERE, AS, projections, DISTINCT, LIMIT, OFFSET, ORDER BY, UNION, INTERSECT, EXCEPT, produit cartésien, opérations au programme
- Jointures internes JOIN, JOIN ON, LEFT JOIN ON
- Agrégats: MIN, MAX, SUM, AVG, COUNT
- HAVING

Map d'une requête:

```
SELECT (DISTINCT) champs -- `champs` peut avoir des agrégas
AS names
FROM data
(LEFT|INNER) JOIN jointures ON eq
WHERE condition    -- filtre les données avant regroupement
GROUP BY attributs -- fabrique les groupes
HAVING condition   -- filtre les groupes
ORDER BY attributs -- tri par `attributs` dans l'ordre lexico
LIMIT nombre       -- prendre seulement `nombre` lignes
OFFSET nombre      -- ignorer les premières `nombre` lignes
```

Films et notes[86]

On considère une bases de donn´ees contenant trois tables:

- `Realisateurs(id, nom, prenom, nation, nb films)`
- `Films(id, nom, id_realisateur)`
- `Notes(id, id_film, note)`

1. Pour chaque table, donner les clés primaire et étrangère
2. Écrire une requête SQL qui retourne les noms et prénoms des réalisateurs américains ayant fait plus de 10 films
3. Écrire une requête SQL qui retourne le réalisateur ayant fait le plus de films
4. Écrire une requête SQL qui retourne la table des filmss avec le nom et prénom du réalisateur
5. Écrire une requête qui retourne la note moyenne reçu par un film ayant été réalisé oar nu américain

Des étudiants et des notes[87]

On considère une bases de donn´ees contenant 2 tables:

- `Etudiants (id_etudiant, prenom, nom)`
- `Examen(id_examen, id_etudiant, matiere, note)`

1. Donner pour chaque table les clés primaire et les clef secondaires

[86] Mallory Marin
[87] Mallory Marin

2. Donner un diagramme entité-association correspondant à cette base de données relationnelle
3. Écrire une requête SQL qui retourne l'ensemble des examens pour laquelle la note est plus grande que 15
4. Écrire une requête SQL qui retourne la table contenant pour chaque étudiant son identifiant, nom, prénom, et moyenne générale.
5. Écrire une requête SQL donnant les noms et prénoms des étudiants ayant au moins eu une note supérieure à 15.

Matières difficiles

On considère une base de données contenant 3 tables:

- `Etudiant(id, nom)`
- `Matiere(id, nom, difficulte)`
- `Note(id_etudiant, id_matiere, note)`

Une matière est difficile si `difficulte >= 4`. Un étudiant a validé une matière si sa note est au moins 10.

1. Trouver les étudiants à qui il manque au moins une matière difficile.
2. Trouver les étudiants qui ont validé toutes les matières difficiles.
3. Trouver tous les étudiants qui ont validé toutes les matières difficiles et aucune matière facile (de note ≤ 2).

Ensembles Egaux

On considère une base de données contenant 2 tables:

- `Document(id, titre)`
- `Mot(document, mot)`

On identifie un document à l'ensemble des mots qu'il contient, sans tenir compte des répétitions.

1. Trouver tous les couples de documents distincts (d_1, d_2) tels que tous les mots de d_1 apparaissent dans d_2.
2. Trouver tous les couples de documents distincts (d_1, d_2) tels que d_1 et d_2 ont exactement le même ensemble de mots.
3. Trouver les classes de documents équivalents (i.e. qui ont l'exact même ensemble de mots), représentées par le plus petit identifiant de la classe.

La limite de SQL

On montre ici qu'il existe des requêtes décidables qui ne sont pas exprimables dans le fragment de SQL vu au programme.

On considère pour cela la base de données composé d'une seule table `Arete(x,y)` représenant un graphe orienté avec potentiellement des boucles. On ne considère que des requêtes dans le fragment utilisant seulement les mots-clefs `SELECT`, `DISTINCT`, `FROM`, `JOIN`, `AS`, `ON`, `WHERE` et des conditions dans le WHERE.

Soit $G = (S, A)$ un graphe orienté, un sommet est dit *minimal* s'il n'existe pas de $y \in S$ tel que $(y, x) \in A$, et il est dit maximal s'il n'existe pas $y \in S$ tel que $(x, y) \in A$. On considère ici que le graphe donné contient un sommet initial.

1. Écrire une requête SQL qui donne la table des sommets initiaux et finaux.
2. Écrire une requête SQL qui renvoie les couples (x, y) tel qu'il existe un chemin de longueur exactement 2 entre x et y

3. On considère la requête suivante. Que reconnais-t'elle ? Montrer que l'on peut écrire une requête équivalente sans utiliser `JOIN`.

```
SELECT DISTINCT a.x, c.y
FROM Arete AS a
JOIN Arete AS b ON a.y = b.x
JOIN Arete AS c ON a.x = c.x AND b.y = c.y;
```

4. Expliquer comment transformer une requête utilisant des `JOIN` en une requête utilisant seulement des `SELECT` et `WHERE`

Dans la suite, on se place donc dans ce fragment: les requêtes sont écrites avec plusieurs occurrences de tables dans le `FROM`, puis des conditions d'égalité dans le `WHERE`.

Pour une requête R, on appelle taille en arêtes de R le nombre d'occurrences de la table `Arete` dans la clause `FROM`. Par exemple, la requête suivante a taille en arêtes 2 :

```
SELECT DISTINCT a.x, b.y
FROM Arete AS a, Arete AS b
WHERE a.y = b.x;
```

5. Soit R une requête de taille en arêtes k. Soit T le résultat de la requête sur un graphe $G = (S, A)$.

   Montrer que pour chaque tuple $(x_1, ..., x_n) \in T$, il existe un sous-graphe $G' = (S, A')$ avec $A' \subseteq A$ et $|A'| \leq k$ tel que $(x_1, ..., x_n)$ appartienne encore au résultat de R évaluée sur G'.

On suppose maintenant, par contradiction, qu'il existe une requête R de ce fragment qui renvoie exactement les couples (s, t) tels que:
- s est un sommet initial,
- t est un sommet final,
- il existe un chemin orienté de s vers t.

On note k la taille en arêtes de R.
6. On considère P_n le chemin orienté de longueur n. Montrer que dans P_{k+1}, le sommet 0 est relié au sommet $k + 1$ par un chemin.
7. En appliquant la question 6 à la requête R sur le graphe P_{k+1}, obtenir une contradiction.
8. Montrer que même en ajoutant `ORDER BY`, `LIMIT` ou `OFFSET`, la requête R n'est pas exprimable.

Topologie d'une montagne[88]

On considère une base de donné pour représenter des données topographiques avec l'unique table `Altitude(x, alt)`. `x` est une clef primaire et l'on sait que toutes les valeurs de 1 à N sont remplies, avec une valeur d'altitude `alt` qui n'est jamais `NULL`.

Dans cet exercice, on interdit l'utilisation de GROUP BY, plusieurs fois SELECT dans la même requête, ORDER BY, LIMIT, OFFSET, HAVING.

1. Écrire une requête qui renvoie la somme des altitudes des enregistrement correspondant à une position d'altitude strictement positive.
2. Écrire une requête qui renvoie le plus grand dénivelé entre deux positions consécutives qui sont toutes deux au dessus du niveau de la mer.
3. On dit qu'une position `x` est un *record depuis la gauche* si aucune position strictement plus à gauche n'a d'altitude supérieure ou égale. Écrire une requête qui renvoies les records depuis la gauche.

[88] Inspiré du concours Général d'informatique 2022

4. Pour chaque position x, renvoyer la position $y \leq x$ où l'altitude est maximale parmi les positions $\{1, ..., x\}$. En cas d'égalité, prendre la position la plus à gauche.
5. $(*)$ Écrire une requête qui renvoie le nombre de plateaux, à savoir les zones de une ou plusieurs positions consécutives de même altitude, telles que cette altitude est strictement plus élevée que les altitudes des positions immédiatement avant la position la plus à gauche du plateau, et immédiatement après la position la plus à droite du plateau.

MP2I: Algorithmique

Cours

Technique d'algorithmie :

- Algorithme Glouton
- Diviser pour régner, rencontre au millieu, dichotomie
- Programmation dynamique : Calcul de bas en haut par mémoisatioon, reconstruction de solution.
- Recherche Brute-force, Retour sur trace

Algorithme du Cours :

- Recherche Dichotomique dans un tableau trié
- Tri fusion
- Tri insertion/tri selection (?? c'est au programme meme??)
- Tri par tas
- Tri rapide (????)
- Algorithme de Boyer-Moore (recherche textuelle)
- Algorithme de Rabin-Karp (recherche textuelle)
- Algorithme de Huffman (compression)
- Algorithme de Lempel-Ziv-Welch (compression)
- Parcours en largeur, profondeur dans un graphe
- Kosaraju pour les CFC
- Floyd-Warshall
- Dijsktra

Exercices cité dans le programme :

- Huffman, sélection d'activité, ordonnancement de taches unitaires
- recherche dichotomique dans un tableau trié
- tri fusion
- comptage du nombre d'inversions dans une liste
- calcul des deux points les plus proches dans une ensemble de points
- recherche d'un sous-ensemble d'un ensemble d'entiers dont la somme des éléments est donnée
- problème de la somme d'un sous-ensemble
- ordonnancement de tâches pondérées
- plus longue sous-suite commune
- distance d'édition

Merge de liste

Soit L_1, L_2 deux listes triées. Proposer un algoritme en OCaml qui calcule la liste trié des éléments contenue dans L_1 et L_2.

Nombre d'inversions dans une liste

Soit L une liste de longueur n, on dit que (i, j) pour $0 \leq i < j < n$ est une inversion si $L[i] > L[j]$. On cherche à compter le nombre d'inversion.

En vous inspirant du tri fusion, proposer une approche dichotomique en $O(n \log n)$.

Meilleur chemin dans une grille

Soient $n, m \in \mathbb{N}$. On considère une grille de taille $(n+1) \times (m+1)$. Chaque case (i, j) contient un entier $c_{i,j}$, éventuellement négatif.

On part de la case $(0, 0)$ et on veut atteindre la case (n, m). À chaque étape, on ne peut que aller à droite de (i, j) à $(i+1, j)$ ou en bas de (i, j) à $(i, j+1)$. Le score d'un chemin est la somme des valeurs des cases visitées, y compris la case de départ et la case d'arrivée.

On cherche un chemin de score maximal. On représentera la grille par une matrice en C de taille $(n+1) \times (m+1)$ d'entiers.

1. Trouver une relation de récurrence permettant de calculer le meilleur score pour atteindre chaque case.
2. Donner un algorithme en C de programmation dynamique calculant le score optimal. Quel est sa complexité?
3. Modifier l'algorithme pour reconstruire un chemin optimal.

Points les plus proches

On pose $P = \{p_1, ..., p_n\} \subseteq \mathbb{R}^2$ un ensemble de points du plan. On écrit pour tout $1 \leq i \leq n$ $p_i = (x_i, y_i)$. On cherche deux points p_i, p_j distincts qui minimise

$$\|p_i - p_j\|_2 = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$$

Montrer qu'il existe un algorithme en $O(n \log n)$.

Indication: Trier P selon les $(x_i)_i$, puis couper P en deux pour s'apeller récursivement sur deux sous-ensemble de points. Quel cas manquent-t'ils? Pour ces cas-làs, considérer un tri selon les $(y_i)_i$

Sous-tableau connexe

Soit A un tableau de n entiers relatifs, on cherche un algorithme en $O(n)$ qui calcule le sous-tableau connexe qui maximise la somme de ses éléments, i.e. le couple (i, j) avec $0 \leq i \leq j \leq |T|$ tel que $\sum_{k \in [\![i;j]\!]} A[k]$ soit maximal.

Le glouton par défault[89]

Étant donné un ensemble $\{x_1, ..., x_n\}$ de n points sur une droite, décrire un algorithme qui détermine le plus petit ensemble d'intervalles fermés de longueur 1 qui contient tous les points donnés.

Prouver la correction de votre algorithme et donner sa complexité.

SUBSET-SUM

Soit $\{x_1, ..., x_n\} \subseteq \mathbb{N}$ un sous-ensemble d'entiers et $S \in \mathbb{N}$. On cherche à savoir s'il existe un $I \subseteq [\![1; n]\!]$ tel que

$$\sum_{i \in I} x_i = S$$

Donner un algorithme de programmation dynamique en $O(nS)$ qui répond au problème. On pourra considérer le tableau T de booléens de longeur S tel que $T[k] = \text{true}$ si et seulement si il existe $I \subseteq [\![1; n]\!]$ dont $\sum_{i \in I} x_i = k$.

[89] Algo 1 ENS Lyon

Elements Majoritaire

Soit T un tableau d'entier de longeur N, on dit qu'un entier K est *majoritaire dans* T s'il est présent strictement plus que $\lfloor \frac{N}{2} \rfloor$ fois.

1. Donner un algorithme en C `bool is_majo(int* t, int l);` qui teste si un élément est majoritaire ou non.

On note $\text{head}(P)$ l'élement en tete d'une pile P, et on considère l'algorithme suivant:

$P \longleftarrow$ pile vide
Pour i **allant de** 1 à N:
    si P est vide **alors:**
        Empiler $T[i]$ à P
    sinon si $\text{head}(P) = T[i]$ **alors**:
        Empiler $T[i]$ à P
    sinon:
        Dépiler P
    Fin si
Fin Pour

2. Montrer que à toute étape elle contient que les meme valeurs. Elle peut donc etre implémentée par un couple $(\text{valeur de tete}, \text{compteur})$. Donner un code en C qui execute cet algorithme avec cette optimisation.
3. Montrer que si il y a un élément majoritaire dans T, alors à la fin de l'algorithme P est non vide et $\text{head}(P)$ est majoritaire.
4. En déduire un algorithme linéaire qui teste si un élément majoritaire est dans P

On dit qu'un élément est C-majoritaire s'il est présent strictement plus de $\lfloor \frac{N}{C} \rfloor$ fois.

5. Pour $C \in \mathbb{N}$ est fixé, donner un algorithme linéaire pour calculer les (au plus C) éléments majoritaires de T.
6. Quelle est la complexité en fonction de C et N ?

Disco-world[90]

A DiscoWorld, quand tu as une réduction, elle s'applique à tout... meme aux réductions ! Alice, fane de réductions, veut toutes les acheter, et ce demande quel est la quantité minimale d'argent qu'elle devra dépenser pour obtenir toutes les réductions.

Version 1 On pose $R = \{(c_1, v_1), ..., (c_n, v_n)\}$ un ensemble de réductions, tel que c_i soit le cout de la i-ème réduction et v_i soit la valeure qui sera soustraite à tout les achats suivants (pour un cout pottentiellement négatif). On ne peut pas acheter 2 fois la meme réduction.

Par example, pour $R = \{(5, 4), (10, 4), (5, 2)\}$, Alice n'aura besoin de que 8€ pour tout acheter : elle achete d'abord la réduction 1 pour 5€, puis la réduction 2 pour 6€, puis elle achete la réduction 3 pour... -3€. Elle aura donc au total seulement utilliser $5 + 6 - 3 = 8$€. La réponse est donc 8€ dans ce cas.

Donner un algorithme calculant le prix/gain que Alice devrais dépenser/obtiendra en achetant toutes les réduction. On pourra faire par programmation dynamique.

[90] Cours de programmation compétitive ENS Lyon

Version 2 DiscoWorld se rendant compte qu'il commençait à perdre de l'argent, on décider maintenant que une fois une réduction (c_i, v_i) achetée, elle **diviserai** le prix de toutes les futures réductions par v_i. On ne peut toujours que acheter les réduction qu'en un seul exemplaire.

Donner un algorithme polynomial calculant la quantité d'argent que Alice devrais dépenser pour acheter toutes les réductions.

Indication: Comparer l'achat de la réduction i puis j avec l'achat de j puis i. En déduire qu'il suffit de trier les réductions selon une certaine formule.

Recherche dans une matrice

Question 1 Donner le code C d'une fonction `int find(int *arr, int n, int key);` qui prend en argument un tableau `arr` trié de longueur `n` et qui trouve un indice `i` tel que `arr[i] == key`.

On cherche à faire un algorithme efficace aussi dans le cadre de matrice trié. On dit qu'une matrice M de taille $n \times m$ est *linéairement trié* si chaque ligne est trié de haut en bas et que chaque colone est trié de gauche à droite.

Question 2 Donner le code d'un fonction C `bool is_ordered(int **mat, int n, int m)` qui à une matrice `mat` de taille $n \times m$ teste si elle est linéairement trié.

Question 3 Donner le code d'une fonction `is_in(int **mat, int n, int m, int key)` qui teste si `key` est présent dans la matrice `mat` linéairement trié de taille $n \times m$ par diviser pour reigner.

Question 4 Meme question en $O(n + m)$

Question 5 Donner le code d'une fonction C qui en $O(n)$ trouve un minimum local dans une matrice $n \times n$

Multiplication rapide de polynome[91]

Ici on considère des polynômes d'entiers $\mathbb{Z}[X]$. Soient $P, Q \in \mathbb{Z}_n[X]$, leur produit $R = PQ \in \mathbb{Z}_{2d}[X]$

On représente un polynome par un tableau d'entiers T, tel que pour $P \in \mathbb{Z}_d[X]$, si on écrit $P = \sum_{i \leq d} a_i X^i$ on a $T[i] = a_i$.

1. Donner une fonction C `int* multiply(int* p, int* q, int n)` qui prend deux polynômes de degré $\leq n$ représenter par deux tableaux de longueur n et qui renvoie le polynome produit.
2. Soit $n = 2m$, pour $P \in \mathbb{Z}_n[X]$, montrer que on peut décomposer $P = P_1 + X^m P_2$ avec $P_1, P_2 \in \mathbb{Z}_m[X]$.

Soient $P, Q \in \mathbb{Z}_n[X]$, on décompose $P = P_1 + X^m P_2$ et $Q = P_1 + X^m P_2$. On définit alors $R_1 = P_1 Q_1$, $R_2 = P_2 Q_2$ et $R_3 = (P_1 + P_2) \times (Q_1 + Q_2)$.

3. Exprimer $P \times Q$ en fonctions de R_1, R_2, R_3.
4. En déduire un algorithme récursif pour calculer le produit de polynome.
5. Quelle est la complexité de cet algorithme ?

Casser des oeufs

Soit $K, N \in \mathbb{N}$. On possède K oeufs absolument identique dont on aimerai savoir leur rédistance. On dispose d'un immeuble à N étages : on sait que les oeufs cassent à l'étage N et ne cassent pas au rez-de chaussé. On cherche à connaitre à partir de quelle hauteur les oeufs cassent en un minimum de lancer.

1. On suppose que $K = \lceil \log_2(N) \rceil$. Proposer une solution en $O(\log N)$ lancers.

[91] Algo 1 ENS Lyon

2. On suppose que l'on dispose de $K = 2$ oeufs. Proposer une solution en $o(N)$ lancers (attention, c'est un petit O).

Maintenant on suppose que racheter un oeufs coute X€ et effectuer un lancer coute Y€ euros. Effectuer un lancer après avoir casser l'oeuf coute donc $X + Y$€.

3. Proposer un algorithme de programmation dynamique qui calcule la quantité d'argent minimale nécéssaire pour etre sure que l'on connaisse la hauteur de casse.

Permutations de Langford[92]

Soit $n \in \mathbb{N}^*$. Une *permutation de Langford* de longeur $2n$ est une suite u de longueur $2n$ contenant exactement deux fois chaque entier de 1 à n, telle que, pour tout $k \in [\![1; n]\!]$, les deux occurrences de k soient séparées par exactement k cases.

On représente une telle permuation en C par un tableau `T` de longeur $2n$ d'entiers.

1. Vérifier que $[2, 3, 1, 2, 1, 3]$ est bien une permutation de Langford de longeur 6.
2. Donner une fonction en C qui teste si un tableaux `T` est une permutation de Langford. On s'assurera que l'algorithme tourne en $O(n)$.

On cherche a faire un algorithme de retour sur trace pour trouver le nombre de permutation de langford de longeur n. Pour cela, on représente une suite partielle par un tableau `t` de taille `2n`, initialisé avec des zéros, représentant une valeur pas encore assignées. Notre invarient sera que au i-ème appel récursif,

- Toutes les cases $0, \ldots, i$ ont été remplies (et sont différentes de 0),
- Si une valeur existe dans le tableaux, alors elle existe deux fois.

3. Expliquer pourquoi, pour placer l'entier k à la position $i + 1$, il faut vérifier les cases $i + 1$ et $i + k + 1$.
4. Proposer un algorithme récursif de retour sur trace. Montrer qu'il termine. Quelle est sa complexité?
5. Montrer que si l'algorithme renvoie une solution, alors c'est bien une permutation de Langford.
6. Soit T une permutation de langford. En dénombrant $1 + 2 + \ldots + 2n$ de deux manières différentes, montrer qu'il n'existe pas de permutation de langford de longeur $2n$ si $n \equiv 1[4]$ ou $n \equiv 2[4]$.

Tableaux auto-référents

Pour T un tableau de longueur n, on définit l'histogramme de T noté H_T comme le tableau de longueur n tel que $H_T[i]$ correspond au nombre de i dans T. Formellement, $H_T[i] = |\{j : T[j] = i\}|$

On dit que T est auto-référent si $H_T = T$. Par exemple, $[1, 2, 1, 0]$ est un tableau autoréférent.

Question 1 Donner le code d'une fonction `int* histogramme(int *t, int n);` qui, à un tableau t de longueur n renvoie H_t son histogramme.

Question 2 Donner tous les tableaux auto-référent de longueur 4 et 5

Question 3 Donner le code d'une fonction `bool is_autoref(int *t, int n);` qui teste si un tableau t est auto-référent.

Question 4 Donner le code d'une fonction de recherche exhaustive `int count_autoref(int n);` qui donne le nombre de tableaux auto-référents de longueur n.

Question 5 Montrer que pour tout $n > 6$ il existe un tableau autoréférent de longueur n

[92] CCINP 2025

Question 6 ($*$) Montrer qu'il est unique

MPI: Langages

Cours

- Définir un alphabet, un mot, le mot vide ε, un langage
- Longeur $|u|$ d'un mot, $|u|_a$ nombre d'occurence d'une lettre
- Concaténation, union, intersection de langages, complémentaire, étoile de Kleene
- Préfixe, suffixe, facteur, sous-mot
- Langage clot par opération, classes de langages
- Mirroir, palindrome (HP)
- Le monoide libre $(\Sigma^*, \cdot)$ et morphismes de mots (HP)
- Mots infinis, concaténation de mots infinis, morphismes (HP)

Questions de Cours

- Donner les préfixes, suffixes, facteurs et sous-mots de abbab et abcba.
- **Lemme de levy:** Soient $x, y, x', y' \in \Sigma^*$ tel que $xy = x'y'$. Montrer qu'il existe $z \in \Sigma^*$ tel que soit $x = x'z$ et $y' = zy$, soit $x' = xz$ et $y = zy'$.
- Soient $u, v \in \Sigma^*$ tel que $uv = vu$. Montrer qu'il existe $w \in \Sigma^*$ et $p, q \in \mathbb{N}$ t.q. $u = w^p$ et $v = w^q$

Petites questions

1. Donner le plus petit langage clot par préfixe contenant tous les mots qui peuvent s'écrire de la forme $\{a^n b^n : n \in \mathbb{N}\}$
2. Soient $u, v \in \Sigma^*$ et $a, b \in \mathbb{N}$ tel que $u^a = v^b$. Montrer qu'il existe $w \in \Sigma^*$ et $p, q \in \mathbb{N}$ tel que $u = w^p$ et $v = w^q$
3. Soient $x, y, z \in \Sigma^*$ tel que $xy = yz$ avec $x \neq \varepsilon$. Montrer qu'il existe $k \in \mathbb{N}$ et $u, v \in \Sigma^*$ tels que $x = uv$, $z = vu$ et $y = (uv)^k u$.
4. Un mot est dit *primitif* s'il n'est puissance d'aucun autre mot que lui-même. Montrer que pour tout mot non vide, il existe un unique mot primitif dont il est une puissance.

Exercices à rajouter

- Equations de mots https://diplome.di.ens.fr/informatique-ens/annales/2022_InfoLCR-rapport.pdf

Opérations de langages

Pour les paires de $L_i, L_{i'}$ suivantes sur $\Sigma = \{a, b, c\}$, expliciter les valeurs de $L_i \cap L'_i$, $L_i \cup L_{i'}$, $L_i.L_{i'}$ et L_i^* :

1. $L_1 = \{\text{abb}, \text{ba}, \text{c}\}, L'_1 = \{\text{abb}, \text{cc}, \varepsilon\}$
2. $L_2 = \{a^n : n \in \mathbb{N}^*\}, L'_2 = L_2$
3. $L_3 = \{a^n b^n : n \in \mathbb{N}^*\}, L'_3 = \{c^n : n \in \mathbb{N}\}$
4. $L_3 = \{a^n b^n : n \in \mathbb{N}^*\}, L'_3 = \{a^n : n \in \mathbb{N}\}$
5. $L_4 = \{a^n b^m : n < m\}, L'_4 = \{b^n : n \in \mathbb{N}\}$
6. $L_5 = \{a^n b^m : n < m\}, L'_5 = \{b^n c^m : n < m\}$

Language contenant ε

Soit L un langage, montrer que $\varepsilon \in L \Leftrightarrow L \subseteq L^2$

Cloture de langages

Pour chacun des langages suivant, indiquer le plus petit langage L clot par étoile de kleene, préfixe, suffixe, sous-mot

1. $L = \{a^n : n \in \mathbb{N}\}$

1. $L = \{a^n b^n : n \in \mathbb{N}\}$
1. $L = \{a^n b^n : n \in \mathbb{N}\}$

Suite de mots par récurrence

On définit sur l'alphabet $\Sigma = \{a, b\}$ la suite de mots $(u_n)_{n\geq 0}$ définie par récurrence par $u_0 = a$, $u_1 = b$ et $u_{n+2} = u_{n+1}u_n$

1. Montrer que si $n \geq 2$, alors u_n se termine par ba si n est pair et par ab sinon.
2. Proposer un algorithme qui pour un $n \in \mathbb{N}$ calcule $|u_n|$ la longueur de u_n
3. On définit v_n comme étant u_n où l'on a retiré les deux dernières lettres. Montrer que pour tout $n \in \mathbb{N}_{\geq 2}$, v_n est un palindrome.

Mot circulaire

Étant donnés deux mots $u, v \in \Sigma^n$ et $p < n$, proposer un algorithme qui trouve des indices i, j tels que $u[i...i+p-1]$ et $v[j...j+p-1]$ ont le plus grand nombre de caractères identiques.

Les indices sont à prendre au sens circulaire (i.e. $w[|w| + l] = w[l]$, pour tout $l < |w|$).

Hypercubes et mots

On définit l'hypercube de dimension n comme étant le graphe non orienté $G = (V, E)$ avec $V = \{0, 1\}^n$ et E l'ensemble des couples (u, v) de mots qui ne diffèrent que d'un bit.

1. Montrer que l'hypercube de dimension n possède un cycle hamiltonien.
2. En déduire qu'on peut énumérer les mots de $\{0, 1\}^n$ ne ne modifiant à chaque fois qu'un seul mot (une telle énumération est un *code de Gray*).
3. Proposer un algorithme prenant un n et renvoyant un code de Gray des mots de longueur n
4. Proposer un algorithme (de bonne complexité) qui à un mot associe son suivant dans cette liste.

Mots univers

On dit qu'un mot $w \in \Sigma^*$ est n-univers si tous les mots de Σ^n sont des facteurs de w. On s'intéresse à créer les plus petits mots n-univers.

1. Montrer qu'un mot n-univers sur un alphabet à k lettres à au moins une longueur de $k^n + n - 1$

Soit $G = (V, E)$ un graphe **orienté**, on définit $L(G)$ le *graphe ligne* de G par le graphe orienté (E, E') avec E' l'ensemble des arêtes de la forme $((x, y), (y, z))$ pour $(x, y), (y, z) \in E$.

2. Donner le graphe ligne du cycle à 4 éléments et d'un arbre binaire parfait de hauteur 2.

On construit alors la famille des graphes de Bruijn $(\mathrm{DB}(n))_{n\in\mathbb{N}^*}$ par $\mathrm{DB}(1) = (\{0, 1\}, \{0, 1\}^2)$ et $\mathrm{DB}(n + 1) = L(\mathrm{DB}(n))$.

3. Construire $\mathrm{DB}(2)$
4. Montrer que pour tout $n \in \mathbb{N}^*$, chaque sommet de $\mathrm{DB}(n)$ à autant d'arêtes sortantes que entrantes. Combien de sommets et d'arêtes $\mathrm{DB}(n)$ possède t'il ?
5. Montrer que pour tout graphe orienté fortement connexe tel que pour tout sommet le degré entrant est le même que le degré sortant, il existe un cycle eulérien (un cycle passant par toutes les arêtes du graphe).

   En déduire que pour tout $n \in \mathbb{N}^*$, $\mathrm{DB}(n)$ possède un cycle eulérien.
6. En voyant les sommets de $\mathrm{DB}(n)$ comme des mots dans $\{0, 1\}^{n-1}$, et en étiquetant les arêtes par 0 ou 1, montrer qu'il existe un mot n-univers sur l'alphabet $\{0, 1\}$ de taille $2^n + n - 1$
7. Généraliser la question précédente pour des alphabets plus grand.

Ensemble inévitables[93]

On fixe un alphabet fini $\Sigma = \{a, b\}$. Pour $w \in \Sigma^*$, on écrit $w = w_1...w_n$ où $n := |w|$ est la longueur de w. On dit qu'un mot $w \in \Sigma^*$ évite un mot $s \in \Sigma^*$ si s n'apparaît pas comme facteur de w. On dit que w évite un ensemble de mots $S \subseteq \Sigma^*$ s'il évite chaque mot de S.

1. Donner un mot de longueur au moins 12 qui évite $S = \{aaaa, aaab, aba, baaa, bab, bbbb\}$.
2. Donner un algorithme qui étant donné un ensemble fini de mot S et un w, teste si w évite S.

On dit qu'un ensemble $S \subseteq \Sigma^*$ est inévitable s'il n'existe qu'un nombre fini de mots $w \in \Sigma^*$ qui évitent S. Sinon, on dit que S est évitable.

3. Est-ce que S est évitable ? Et l'ensemble $\{aaa, abb, baa, abab\}$?
4. Montrer que pour tout Σ un alphabet, et pour tout $k \in \mathbb{N}$, il existe un $n \in \mathbb{N}$ tel que pour tout mot $w \in \Sigma^*$ avec $|w| > n$, il existe $p < q$ tel que pour tout $l \in [\![0; k]\!]$, on a $w_{p+l} = w_{q+l}$
5. Montrer que, pour tout ensemble inévitable $S \subseteq \Sigma^*$, il existe un sous-ensemble $S' \subseteq S$ fini tel que S' soit inévitable.

Mots sans cube[94]

On dit qu'un mot w est sans carré s'il n'a pas de facteur de la forme u^2 pour $u \neq \varepsilon$. On dit qu'il est sans cube s'il ne contient pas de u^3 en

1. Donner tous les mots sans carré sur $\Sigma = \{a, b\}$

On définit $\overline{h} : \Sigma \longrightarrow \Sigma^*$ par $\overline{h}(\mathrm{a}) = \mathrm{ab}$ et $\overline{h}(\mathrm{b}) = \mathrm{ba}$ que l'on étend en $h : \Sigma^* \cup \Sigma^{\mathbb{N}} \longrightarrow \Sigma^*$ par $h\big(w_1...w_{|w|}\big) = \overline{h}(w_1)...\overline{h}\big(w_{|w|}\big)$ et $h(w_1...w_n...) = \overline{h}(w_1)...\overline{h}(w_n))...$

2. Montrer que $h^i(a)$ est un préfixe de $h^{i+1}(a)$.
3. Montrer que pour tout i, $h^i(a)$ est sans cube
4. En déduire l'existance d'un mot arbitrairement grand sans cube.

Facteurs de points fixe de morphismes[95]

On dit que $h : \Sigma^* \longrightarrow \Sigma^*$ est un *morphisme* si pour tout $u, v \in \Sigma^*$, $h(uv) = h(u)h(v)$. On dit qu'un morphisme h est une *substitution* si pour tout $\alpha \in \Sigma$, $|h(\alpha)| \geq 1$. Une telle substitution est dite α-*prolongeable* pour $\alpha \in \Sigma$ si α est un préfixe strict de $h(\alpha)$.

1. Soit h une substitution α-prolongeable. Montrer que pour tout $n \in \mathbb{N}$, $h^n(\alpha)$ est un préfixe strict de $h^{n+1}(\alpha)$
2. En déduire l'existence d'un unique mot infini u commençant par α dont tous les préfixes $h^n(\alpha)$ sont préfixes, et tel que $h(u) = u$.

Dans ce cas, on définit $\lim_{n\to\infty} h^n(\alpha)$ l'unique mot infini qui soit point fixe de h commençant par α. Une substitution (pas forcément α-prolongeable) est dite *primitive* s'il existe $k \in \mathbb{N}$ tel que $\forall \alpha, \beta \in \Sigma, |f^k(\alpha)|_\beta > 0$.

Soit f une substition primitive sur un alphabet Σ à au moins 2 lettres.

3. Montrer qu'il existe $k \in \mathbb{N}$ et $\alpha \in \Sigma$ tel que f^k soit α-prolongeable.
3. On définit $F(w)$ l'ensemble des facteurs de w pour $w \in \Sigma^*$. Soient $L = \bigcup_{n\in\mathbb{N}} F\big(f^{nk}(\alpha)\big)$ et $w \in L$, montrer qu'il existe $B \in \mathbb{N}^*$ tel que tout $u \in L$ de longueur $\geq B$ contient w en facteur.

---

[93] oral info Ulm ENS 2019

[94] Oral de l'ENS Info 2019 et sujet de CCINP d'Informatique 2026

[95] Tiré de l'Info D 2026 de décembre

Langage sans étoile

Soit Σ un alphabet, on définit par induction la classe des langages sans étoile $\mathcal{M}$ par:
- Σ^* est sans étoile,
- Pour tout $\alpha \in \Sigma$, $\{\alpha\}$ est sans étoile,
- Pour tout L_1, L_2 deux langages sans étoile, $L_1 \cup L_2$, $L_1 \setminus L_2$ et $L_1 \cdot L_2$ sont sans-étoile.

1. Montrer $\{ab^n : n \in \mathbb{N}\}$ est un langage sans étoile sur $\Sigma = \{a, b\}$.
2. Montrer que pour tout $u \in \Sigma^*$, le langage des mots qui n'ont pas u comme facteur est sans étoile.
3. En déduire que $\{(ab)^n : n \in \mathbb{N}\}$ est un langage sans étoile.

$(*)$ Egalité pour les résiduels

Soit Σ un alphabet. Donner L un langage sur Σ tel que $\forall t \in \Sigma^*, ut \in L \Leftrightarrow vt \in L$ ssi $u = v$

$(*)$ Lemme d'Higman

Soit Σ un alphabet fini à n lettres. On note le fait que u soit un sous-mot de v par $v \succcurlyeq u$.

Montrer que pour toute suite de mot infinie $(v_i)_{i \in \mathbb{N}}$, on a l'existence de $i < j$ tel que $v_i \succcurlyeq v_j$.

MPI: Langages réguliers et Automates

Cours

- Définition d'un automate $(\Sigma, Q, q_{\text{init}}, F, \delta)$ avec δ une fonction ou $(\Sigma, Q, q_{\text{init}}, F, \Delta)$ avec Δ un ensemble.
- Automate déterministe, complet, émondé. Etat finaux, initiaux, accesible, co-accesible.
- La fonction de transition étendue δ^*
- Rapeller la méthode de Glushkov pour obtenir un automate à partir d'une expression régulière.
- Rapeller la méthode de Berry-Sethi pour obtenir un automate à partir d'une expression régulière.
- Rapeller la méthode de Thompson pour obtenir un automate à partir d'une expression régulière. (HP)
- Montrer que la classe des langages régulier est stable par complémentaire, union, intersection.
- Montrer que si $\mathcal{A}$ est un automtate non déterministe alors il existe $\mathcal{A}'$ un automate déterministe qui reconnaît le même langage que $\mathcal{A}$.
- Montrer le lemme de l'étoile.
- Montrer que le langage $\{a^n b^n : n \in \mathbb{N}\}$ n'est pas régulier.
- Montrer le lemme d'Arden (HP).
- L est régulier ssi existe morphisme $\varphi : \Sigma^* \longrightarrow M$ monoide fini et $F \subseteq M$ tq $L = \varphi^{-1}(F)$. Le monoide $M = \{\delta^*(u, \cdot) : u \in \Sigma^*\}$ (HP)

Exos à rajouter

- Oracle hors-contexte aux langages réguliers
- Automates d'arbres

Berry-Sethi

Que vaux les automates obtenu par la méthode de Berry-sethi appliqué aux expressions suivantes:
- $(a \mid b)^*$
- $(a \mid b)^* aaab$
- $(a \mid b)^*(c(ba \mid c))^*$

Langages réguliers

Soit L, L' deux langages réguliers. Montrer que les langages suivants sont réguliers:
- $\left\{w_1 w_1 w_2 w_2 ... w_{|w|} w_{|w|} : w \in L\right\}$

- $\{w_1 w_3 w_5 ....w_k : w \in L$ avec $k = |w|$ si $|w|$ impair et $k = |w| - 1$ sinon$\}$
- $\overline{L} = \{\overline{u} : u \in L\}$ le langages des mirroirs
- $\bigcup_{w \in L} S(w)$ pour $S(w)$ l'ensemble des sous-mots de w
- $L/L' = \{w \in \Sigma^* \mid \exists x \in L', wx \in L\}$
- $u^{-1}L = \{w \in \Sigma^* \mid uw \in L\}$ pour $u \in \Sigma^*$

Non régulier

Montrer que les langages suivant ne sont pas réguliers:
- $\left\{a^{2^n} : n \in \mathbb{N}\right\}$
- $\left\{a^{n^2} : n \in \mathbb{N}\right\}$
- $\{a^n b^m : n \leq m\}$
- $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$
- $\{a^p : p \in \mathbb{P}\}$ pour $\mathbb{P}$ l'ensemble des nombres permiers.
- $\{a^n b^m : n \geq m\}$ (petite subtilité)

Petites questions

- Monter que l'on peut réécrire une expression régulière sans $\emptyset$ si le langage dénoté n'est pas vide.
- Soit L un langage reconnaissable reconnu par $\mathcal{A}$, on pose pour $q, q' \in Q_{\mathcal{A}}$ le langage $L_{q,q'}$ des mots étiquetant un chemin de q à q'. Montrer que $L_{q,q'}$ est régulier.

Cloture interne par préfixe

Soit $w \in \Sigma^*$, on définit $\mathrm{Pref}(w)$ (respectivement $\mathrm{Suff}(w)$) l'ensemble des préfixes (respc. suffixes) de w. Pour L un langage, on pose $\psi(L) = \{w \in \Sigma^* \mid \mathrm{Pref}(w) \subseteq L\}$

1. Montrer que si L est régulier, $\psi(L)$ l'est aussi.
2. Proposer un algorithme pour tester si un automate $\mathcal{A}$ vérfie $L(\mathcal{A}) = \psi(L(\mathcal{A}))$

Factorisation arbitraire de l'étoile

Montrer les 3 lemmes suivants plus forts que le lemme de l'étoile:

Lemme 1 Soit L un langage régulier sur Σ. Alors il existe $N \in \mathbb{N}^*$ tel que pour tout $u, w, v \in \Sigma^*$ tel que $uwv \in L$ et $|w| > N$, il existe $x, y, z \in \Sigma^*$ tels que:
- $w = xyz$,
- $|y| > 0$,
- $\forall i \in \mathbb{N}, uxy^i zv \in L$.

Lemme 2 Soit L un langage régulier sur Σ. Alors il existe $N \in \mathbb{N}^*$ tel que pour tout mot de la forme

$$u_0 w_1 u_1 w_2 u_2 ... w_k u_k \in L$$

avec $k \in \mathbb{N}$ et $|w_j| > N$ pour tout $j \in \{1, ..., k\}$, il existe, pour tout $j \leq k$, une factorisation $w_j = x_j y_j z_j$ telle que $|y_j| > 0$ pour tout $j \leq k$, et pour tous $i_1, ..., i_k \in \mathbb{N}$,

$$u_0 x_1 y_1^{i_1} z_1 u_1 x_2 y_2^{i_2} z_2 u_2 ... x_k y_k^{i_k} z_k u_k \in L.$$

Lemme 3, positions marquées Soit L un langage régulier sur Σ. Alors il existe $N \in \mathbb{N}^*$ tel que pour tout mot $w \in L$ de longueur $n = |w|$ et $I \subseteq [\![1; n]\!]$ avec $|I| > N$ un "marquage" d'au moins N positions, il existe une factorisation $w = xyz$ telle que:
- y contient au moins une position marquée,
- xy contient au plus N positions marquées,
- $\forall i \in \mathbb{N}, xy^i z \in L$.

Mots ordonnés[96]

Soit Σ un alphabet et $\leq$ un ordre sur Σ – pas nécessairement total. On défini l'ordre préfixe $\sqsubseteq$ par $u \subseteq v$ si et seulement si u est un préfixe de v. On définit l'ordre $\preceq$ comme étant l'ordre lexicographique concu à partir de $\leq$.

1. Rapeller la définition de l'ordre lexicographique.
2. Montrer que l'ordre $\sqsubseteq$ est bien fondé.
3. Est-ce que l'ordre $\preceq$ est toujours bien fondé?

Pour L un langage sur Σ, on définit $\min L = \{u \in L \mid \forall v \in L, \neg v \preceq u\}$

4. On pose $\Sigma = \{a, b, c\}$ avec $a \leq b$ et c incomparable à b et c. Donner $\min\{\text{abb}, \text{ba}, \text{ca}, \text{cba}, \text{ac}\}$
5. Montrer que si L est un langage régulier, alors $\min L$ l'est aussi.

Automates à sauts[97]

Soit $\mathcal{A}$ un automate à un seul éta initial q_{init}. On note Q l'ensemble des sommets $R = Q \times \Sigma^* \times Q$ les transitions, F les états finaux.

Soient $u, v, u', v' \in \Sigma^*$, $a \in \Sigma \cup \{\varepsilon\}$ et $(q_1, q_2) \in Q^2$. On pose la relation $(u, q_1, av) \curvearrowright (u', q_2, v')$ valable uniquement si $(q_1, a, q_2) \in R \wedge uv = u'v'$

On a $(u, q_1, av) \curvearrowright^* (u', q_2, v')$ si un nombre fini de $\curvearrowright$ permettent de passer de (u, q_1, av) à (u', q_2, v').

On définit le langage à saut de l'automate comme suit:

$$L_{\ddagger}(A) = \{uv \in \Sigma^* \mid \exists f \in F, (u, q_{\text{init}}, v) \curvearrowright^* (\varepsilon, f, \varepsilon)\}$$

1. Montrer que $L_{\text{ab}} = \{w \in \Sigma^* \mid |w|_a = |w|_b\}$ n'est pas régulier.
2. Montrer que L_{ab} est reconnu par le langage à saut d'un automate.

On note $\text{Perm}(L) = \{u \in \Sigma^* \mid \exists v \in L, u \text{ est une permutation des lettres de } v\}$.

3. Si L est régulier, est-ce que $\text{Perm}(L)$ aussi ?
4. Si L est régulier, est-ce que $\text{Perm}(L)$ est le langage à saut d'un automate ?

Automates avec couts (algo A^*, sujet bof)

On considère un automate $\mathcal{A}$ pas forcément déterministe muni d'une fonction de cout $c : \delta \longrightarrow \mathbb{R}$ sur les transitions. Le cout d'un chemin est la somme des couts de chaque transition. Soit u un mot accepté par l'automate, on définit son cout $c(u)$ comme le cout minimal d'un chemin acceptant u, si c'est bien définit.

Soit $C \in \mathbb{R}$, on pose $L(\mathcal{A})_C = \{u \in L \mid c(u) = C\}$

1. Montrer que $c(u)$ est bien définit dans un automate sans ε-transition. On se place maintenant dans ce contexte.
2. Donner un automate pondéré $\mathcal{A}$ tel que $L(\mathcal{A})_C = \{a^n b^n : n \in \mathbb{N}\}$ pour un certain C
3. Montrer que si $c : \delta \longrightarrow \mathbb{R}^+$ est positive, alors pour tout C, $L(\mathcal{A})_C$ est un langage régulier.
4. Toujours si $c : \delta \longrightarrow \mathbb{R}^+$, proposer un algorithme qui renvoie un des plus petits mots en longueur $u \in L(\mathcal{A})_C$. Pouvez-vous trouver une heuristique admissible et utiliser l'algorithme A^* ?

Langage croisé

Soit Σ un alphabet et L_1, L_2 deux langages, on définit le langage croisé $L_1 \parallel L_2$ par:

[96] CCINP 2024?

[97] Oral Mines-Télécom 2024, https://beos.prepas.org/sujet.php~?id=8151

$$L_1 \parallel L_2 = \left\{xyz : (x,y,z) \in (\Sigma^*)^3 \mid xy \in L_1 \wedge yz \in L_2\right\}$$

1. Montrer que $\{a^n b^n : n \in \mathbb{N}\} \parallel \{b^n a^n : n \in \mathbb{N}\}$ n'est pas régulier.
2. Montrer que si L_1 et L_2 sont régulier alors $L_1 \parallel L_2$ l'est aussi.

Sous-langage palindromique

Soit Σ un alphabet, pour $w = w_1 w_2 ... w_n \in \Sigma^*$, on note $\overline{w} = w_n ... w_2 w_1$ le mot *miroir* de w. Soit L un langage sur Σ, on pose $\overline{L} = \{\overline{w} : w \in L\}$. On pose aussi $\psi(L) = \{w\overline{w} : w \in L\}$

1. Donner un langage régulier non vide L tel que $\psi(L)$ soit régulier.
2. Montrer que si L est régulier, alors $\overline{L}$ aussi.
3. Montrer qu'il existe L régulier, tel que $\psi(L)$ ne le soit pas.
4. $(*)$ Proposer un algorithme qui teste si un automate donné $\mathcal{A}$ est tel que $\psi(L(\mathcal{A}))$ soit régulier.

Langage régulier à partir de l'automate[98]

On se fixe un alphabet Σ dans tout l'exercice. On se donne K, L des langages sur Σ, tels que $\varepsilon \notin K$, et on considère l'équation suivante, d'inconnue le langage X :

$$X = KX \cup L$$

1. Dans le cas $K = \{a\}$, $L = \{b\}$, donner la forme de X. Une preuve formelle n'est pas attendue ici.
2. Généraliser ce résultat pour K et L quelconques vérifiant les hypothèses. Montrer ce résultat, que l'on appelle Lemme d'Arden. Que dire des langages solutions si K et L sont réguliers ? *Indication: pour l'un des sens, on pourra raisonner par récurrence*
3. Comment le résultat précédent se trouve-t'il modifié si K contient le mot vide ? *Indication: que perd-t'on dans la démonstration précédente ?*

On se fixe $n \in \mathbb{N}^*$, $\left(K_{i,j}\right)_{i \leq i,j \leq n}$, des langages ne contenant pas ε, et $L_1, ..., L_n$ des langages sur Σ.
On considère le système d'équation suivant:

$$\begin{cases} X_1 = K_{1,1}X_1 \cup ... \cup K_{1,n}X_n \cup L_1 \\ X_2 = K_{2,1}X_1 \cup ... \cup K_{2,n}X_n \cup L_2 \\ ... \\ X_n = K_{n,1}X_1 \cup ... \cup K_{n,n}X_n \cup L_n \end{cases}$$

4. Montrer que ce système admet une solution. Que dire des langages solutions si les $K_{i,j}$ et $(L_i)_i$ sont réguliers ?
5. En déduire une méthode/algorithme permettant de construire l'expression régulière dénotant le langage reconnu par un automate fini éterministe A.

Langage rotationel

Soit L un langage sur un alphabet Σ. On définit $R(L) = \{xy : yx \in L\}$ le *rotationel* de L.

1. Montrer que $R(R(L)) = R(L)$

Soit $\mathcal{A} = (\Sigma, Q, q_{\text{init}}, \delta, F)$ un automate déterministe complet, pour $q_1, q_2 \in Q$, on pose $L_{q_1,q_2} = \left\{w : \delta^*(q_1, w) = q_f\right\}$ (autrement dit, l'ensemble des mots étiquetant un chemin de q_1 à q_2).

2. Montrer que pour tout $q_1, q_2 \in Q$, on a que L_{q_1,q_2} est régulier.
3. Montrer que si L est un langage régulier, alors $R(L)$ aussi.

[98] Colle de José

Langage entrelacé

Soient Σ un alphabet et $u, v \in \Sigma^*$ avec $|u| = m_u$ et $|v| = m_v$. Pour $w = w_1 ... w_n \in \Sigma^*$ un mot, on dit que c'est un *entrelacement* de u et v s'il existe $\varphi_u : [\![1; m_u]\!] \longrightarrow [\![1; n]\!]$ et $\varphi_v : [\![1; m_v]\!] \longrightarrow [\![1; n]\!]$ strictement croissantes telles que φ_u, φ_v sont à image disjointes avec $u = w_{\varphi_u(1)} ... w_{\varphi_u(m_u)}$ et $v = w_{\varphi_v(1)} ... w_{\varphi_v(m_v)}$. Intuitivement, u, v sont deux sous-mots disjoints de w. On note l'ensemble des mots qui sont des entrelacements de u et v par $u \leftrightarrows v$.

Pour L, L' deux langages on pose

$$L \leftrightarrows L' = \bigcup_{\substack{u \in L \\ v \in L'}} u \leftrightarrows v.$$

1. Montrer que si L, L' sont régulier, alors $L \leftrightarrows L'$ l'est aussi.

On généralise maintenant à des entrelacements de k mots: soient $u_1, ..., u_k \in \Sigma^*$ de longeurs $m_1, ..., m_n$, on dit que $w \in \Sigma^*$ est un entrelacement de $u_1, ..., u_k$ s'il existe $\varphi_1, ..., \varphi_k$ toutes strictement croisssante tel que $\bigsqcup_{i \leq k} \mathrm{Im}(\varphi_i)$ forme une partition de $[\![1; |w|]\!]$ pour tout $i \leq k$, on a

$$\varphi_i : [\![1; m_i]\!] \longrightarrow [\![1; |w|]\!] \text{ tel que } u_i = w_{\varphi_i(1)} w_{\varphi_i(2)} ... w_{\varphi_i(m_i)}$$

On note l'ensemble des entrelacements de $u_1, ..., u_k$ par $\Downarrow (u_1, ..., u_k)$. On définit l'entrelacement de k langages $L_1, ..., L_k$ comme

$$\Downarrow (L_1, ..., L_k) = \bigcup_{u_1 \in L_1} ... \bigcup_{u_k \in L_k} \Downarrow (u_1, ..., u_k)$$

2. Montrer que $\Downarrow (L_1, ..., L_k) = L_1 \leftrightarrows (L_2 \leftrightarrows (... \leftrightarrows (L_{k-1} \leftrightarrows L_k))...)$. En déduire que l'entrelacement de k langages réguliers est toujours un langage régulier.
3. Soit L régulier, est-ce que $\bigcup_{i \in \mathbb{N}} \Downarrow (L^i)$ est régulier ?

Mots univers

On dit qu'un mot $w \in \Sigma^*$ est n-univers si tous les mots de Σ^n sont des facteurs de w. On cherche à créer les plus petits mots n-univers.

1. Pour $\Sigma = \{a, b, c\}$, donner un mot 2-univers.
2. Montrer qu'un mot n-univers sur un alphabet à k lettres à au moins une longueur de $k^n + n - 1$

Soit $n \geq 2$ et Σ un alphabet, on pose l'automate (vu comme un graphe) dont les états sont $Q = \Sigma^{n-1}$ et les transitions de la forme $\alpha w \xrightarrow[\alpha']{} w\alpha'$ pour tout $\alpha, \alpha' \in \Sigma, w, w' \in \Sigma^{n-2}$, avec quelquonques états initial et finaux (ils ne vont pas avoir d'importance).

3. Montrer que tout graphe fortement connexe orienté tel que tout sommet possède autant d'arêtes entrante que d'arêtes sortante possède un chemin eulérien (qui passe par toutes les aretes une et une seule fois)
4. Montrer qu'il existe un mot n-univers de longueur $|\Sigma|^n + n - 1$

Astucieux langage[99]

Soit $\Sigma = \{a, b\}$

1. Montrer que le langage des mots ayant autant de a que de b sur Σ n'est pas régulier
2. Montrer que le langage des mots ayant autant de ab que de ba sur Σ est régulier

[99] Le fabuleux cours de théorie des langages formels d'Olivier Carton

Régulier à une lettre

On dit que $S \subseteq \mathbb{N}$ est *ultimement périodique* si $\exists N, T \in \mathbb{N}, \forall n > N, n \in S \Leftrightarrow n + T \in S$. Pour L un langage régulier sur $\Sigma = \{a\}$, on pose $S(L) \coloneqq \{n \in \mathbb{N} : a^n \in L\}$. Réciproquement, pour $S \subseteq \mathbb{N}$, on pose $L(S) = \{a^n : n \in S\}$

Montrer que L sur $\Sigma = \{a\}$ est régulier si et seulement si $S(L)$ est ultimement périodique.

Permuté

On définit pour L un langage le *permuté* $\sigma(L)$ par

$$\sigma(L) = \{w \in \Sigma^* \mid \exists u \in L, \forall \alpha \in \Sigma, |u|_\alpha = |w|_\alpha\}$$

1. Donner un langage régulier L tel que $\sigma(L)$ soit non régulier.
2. On pose $L_{\text{ab}} = \{a^n b^n : n \in \mathbb{N}\}$. Montrer que si L est un langage tel que $L_{\text{ab}} \subseteq L \subseteq \sigma(L_{\text{ab}})$, alors L n'est pas régulier

Langages surprenants

Soit $\Sigma = \{a, b\}$.

1. Montrer que $L_1 = \{a^k y \mid |y|_a > k\}$ est régulier.
2. Montrer que $L_2 = \{a^k y \mid |y|_a < k\}$ ne l'est pas.
3. Montrer que $L_1 = \{a^k u a^k : k \geq 1, u \in \Sigma^*\}$ est régulier.
4. Montrer que $L_2 = \{a^k b u a^k : k \geq 1, u \in \Sigma^*\}$ ne l'est pas.

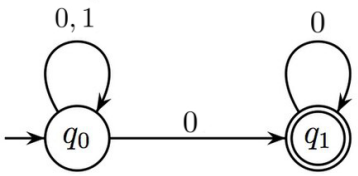
Automates de Büchi[100]

Un automate de Büchi est un automate (Q, Σ, T, I, F) permettant de reconnaître des mots infinis, avec $I, F \subseteq Q$ et $T \subseteq Q \times \Sigma \times Q$.

Soit X un ensemble de mots. On note X^ω l'ensemble des mots infinis de X : $x \in X^\omega$ s'il existe une suite $(x_i)_{i \in \mathbb{N}}$ telle que $x = x_0 x_1 x_2 \ldots$

Un mot infini x sur Σ^ω est reconnu par un automate de Büchi lorsqu'il existe un chemin infini dans cet automate étiqueté par x, commençant par un état initial et passant une infinité de fois par un état final.

1. Quel est le langage reconnu par l'automate de Buchi:



2. Proposer un automate de Buchi recconnaissant $(01)^\omega$.

Soient A, B deux automates de Buchi.

3. Montrer que $L(A) \cup L(B)$ est reconnu par un automate de Buchi.
4. Soit K un langage régulier, Montrer que $K \cdot L(A)$ est reconnu par un automate de Buchi.
5. Montrer que $L(A) \cap L(B)$ est reconnu par un automate de buchi.
6. Montrer que si K est un langage régulier, alors K^ω est reconnu par un automate de buchi.

[100] Mines-Télécom 2024 https://beos.prepas.org/sujet.php~?id=8156

Automates partiellement ordonnés[101]

Un automate (déterministe ou non) $\mathcal{A}$ est dit partiellement ordonné si tous les cycles dans le graphe de $\mathcal{A}$ sont de taille au plus 1.

1. Montrer que l'union et l'intersection de deux langages partiellement ordonné est aussi partiellement ordonné. Est-ce le cas si on se restreint à des langages déterministes ?
2. Montrer que sur $\Sigma = \{a\}$, un langage L est partiellement ordonné si et seulement si il est fini ou il est le complémentaire d'un langage fini.
3. Donner un langage L partiellement ordonné de complémentaire non partiellement ordonné.

Commuteurs de Langages

Soit L_1, L_2 deux langages sur $\Sigma = \{a, b\}$, on définit le langage commutateur de L_1, L_2 par:

$$[\![L_1; L_2]\!] := \{xx'yy' : xy \in L_1 \wedge x'y' \in L_2\}$$

1. Montrer que $[\![L_1; L_2]\!]$ n'est pas régulier pour $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ et $L_2 = \{b^n a^n : n \in \mathbb{N}\}$. *Ind: vous n'etes pas obligé de calculer* $[\![L_1; L_2]\!]$
2. Soit L un langage reconnaissable reconnu par $\mathcal{A}$, on pose pour $q, q' \in Q_{\mathcal{A}}$ le langage $L_{q,q'}$ des mots étiquetant un chemin de q à q'. Montrer que $L_{q,q'}$ est régulier.
3. Montrer que si L_1, L_2 sont régulier, alors $[\![L_1; L_2]\!]$ l'est aussi.

Langage résiduel & langages régulier

Soit Σ un alphabet fini. Pour $u \in \Sigma^*$ et L un langage sur Σ, on définit

$$u^{-1}L = \{w \in \Sigma^* \mid uw \in L\}$$

1. Montrer que si L est régulier, alors pour tout $u \in \Sigma^*$, $u^{-1}L$ est régulier.
2. Montrer que pour tout $u, v \in \Sigma^*$, on a $u^{-1}(v^{-1}L) = (uv)^{-1}L$

Soit $\mathcal{A}$ un automate complet déterministe d'état initial q_{init}. On note $\delta^* : Q \times \Sigma^* \longrightarrow Q$ la fonction de transition étendu, formellement définie par $\delta^*(q, \varepsilon) = q$ et $\delta^*(q, \alpha w) = \delta^*(\delta(q, \alpha), w)$ pour tout $(q, \alpha, w) \in Q \times \Sigma \times \Sigma^*$.

3. Montrer que pour tout $u, v \in \Sigma^*$ et $q \in Q$ on a $\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$
4. Montrer que pour tout $u \in \Sigma^*$ on a $u^{-1}(L(\mathcal{A})) = \{v \in \Sigma^* \mid \delta^*(q_{\text{init}}, uv) \in F\}$
5. En déduire que si L est régulier alors $\{u^{-1}L : u \in \Sigma^*\}$ est fini.
6. Est-ce que la réciproque de la question 5 est vraie ? Justifier.

Commutant de Langage[102]

Soit $u \in \Sigma^*$ un mot, on pose $\mathrm{Comm}(u) = \{v \in \Sigma^* \mid uv = vu\}$

1. Décrire $\mathrm{Comm}(\mathrm{abab})$
2. Soient $u, v \in \Sigma^*$ tels que $uv = vu$. Montrer qu'il existe $w \in \Sigma^*$ et des entiers r, s tels que $u = w^r$ et $v = w^s$
3. En déduire que $\mathrm{Comm}(u)$ est toujours régulier pour $u \in \Sigma^*$.

On souhaite généraliser à un langage. Pour L un langage, on pose

$$\mathrm{Comm}(L) = \bigcap_{u \in L} \mathrm{Comm}(u)$$

Soit L tel que $\mathrm{Comm}(L) \neq \emptyset$

[101] Oral ENS 2022 C2 https://diplome.di.ens.fr/informatique-ens/annales/2022_InfoU-exercices.pdf

[102] Sujet de José

4. Montrer que pour tout $x, y \in \Sigma^*$ tels que $x^k = y^s$ avec $r, s \geq 1$, il existe un mot w tel que x et y soient des puissances de w. *On pourra utiliser la question 2*
5. Montrer qu'il existe un mot w tel que tout mot de L soit une puissance de w
6. En déduire que $\mathrm{Comm}(L)$ est régulier.

Langage coupé au millieu

Soit Σ un alphabet et L un langage, on pose

$$\mathrm{TM}(L) \coloneqq \{w \in \Sigma^* \mid \exists u, v, uwv \in L \wedge |u| = |v| = |w|\}$$
$$\overline{\mathrm{TM}}(L) \coloneqq \{uwv \in \Sigma^* \mid w \in L \wedge |u| = |v| = |w|\}$$

Question 2 Est-ce que $\mathrm{TM}\left(\overline{\mathrm{TM}}(L)\right) = L$? Et est-ce que $\overline{\mathrm{TM}}(\mathrm{TM}(L)) = L$

Question 3 Montrer qu'il existe un langage régulier L tel que $\overline{\mathrm{TM}}(L)$ ne soit pas régulier.

Question 4 Soit L un langage reconnaissable reconnu par $\mathcal{A}$, on pose pour $q, q' \in Q_{\mathcal{A}}$ le langage $L_{q,q'}$ des mots étiquetant un chemin de q à q'. Montrer que $L_{q,q'}$ est régulier.

Question 5 Montrer que si L est régulier, alors $\mathrm{TM}(L)$ aussi.

Langages fins

Soit L un langage, on définit sa fonction de densité $\delta_L : \mathbb{N} \longrightarrow \mathbb{N}$ par $\delta_L(n) = |\Sigma^n \cap L|$. On dit qu'un langage est fin si δ_L est majoré par une constante M.

On cherche à charactériser les langages fins.

1. Montrer que pour tout $k \in \mathbb{N}, u_1, ...u_k, v_1, ..., v_k, w_1, ..., w_k \in \Sigma^*$, le langage suivant est fin:
$$L = \bigcup_{i=1}^{k} L(u_i v_i^* w_i)$$
2. Soit $\mathcal{A}$ un automate émondé reconnaissant un langage fin, soit $q_{\text{init}} \xrightarrow{a_1} .... \xrightarrow{a_n} q_f \in F$ un chemin acceptant acyclique dans $\mathcal{A}$. Montrer qu'il intersecte au plus un cycle.
3. En déduire la réciproque de la question 3, c'est à dire que pour tout langage fin il existe des mots $u_1, ..., w_k$ tel que L à la forme attendue.
4. Proposer un algorithme qui prend en entré un automate et qui indique si $\mathcal{A}$ reconnaît un langage fin. Quel est sa complexité ?

Langages épars[103]

Soit L un langage, on définit sa fonction de densité $\delta_L : \mathbb{N} \longrightarrow \mathbb{N}$ par $\delta_L(n) = |\Sigma^n \cap L|$. On dit qu'un langage est épars si pour tout k, ultimement, on a $\delta_L(n) > n^k$.

1. Proposer une condition nécessaire et suffisante sur les automates pour que le langage reconnu soit épars
2. Donner un algorithme testant si un automate reconnaît un langages épars ou non.

Langages rationel infinis

Un langage est dit infini s'il contient une infinité de mot.

1. Montrer que tout langage rationnel infini est la réunion disjointe de deux langages rationnels infinis.
2. Montrer que tout langage rationnel infini sur un alphabet Σ^* contient un sous-langage non rationnel.

[103] ENS Ulm oral d'info, je ne sais plus de quand

Soit A, B deux langages, on note $A \Subset B$ si $A \subseteq B$ et que $B \setminus A$ contient une infinité de mots.

3. Montrer que si $A \Subset C$ avec A, C deux langages réguliers, alors il existe un langage régulier B tel que $A \Subset B \Subset C$

Racine de langage[104]

Soit L un language régulier sur $\Sigma = \{a, b\}$

1. Montrer que $\sqrt{L} \coloneqq \{u \in \Sigma^* \mid uu \in L\}$ est régulier.
1. Montrer que pour tout $k \in \mathbb{N}$, $\sqrt{L}^k \coloneqq \left\{u \in \Sigma^* \mid u^k \in L\right\}$ est régulier.
2. $(*)$ Montrer que $W(L) \coloneqq \left\{u \in \Sigma^* \mid u^{|u|} \in L\right\}$ est régulier.

Arithmétique de Presburger (+ déduction)

On pose $\Sigma_n = \{0,1\}^n$ l'alphabet des vecteurs de taille n à valeurs dans $\{0, 1\}$. Par exemple,

$$\Sigma_3 = \left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}, ... \begin{pmatrix}1\\1\\1\end{pmatrix} \right\}$$

Un mot sur Σ_n peut être lu comme n nombres binaire (un sur chaque ligne) tel que pour $w \in \Sigma_n^*$ l'on dénote $w[1], w[2], ..., w[n]$. Par exemple, pour $w = \begin{pmatrix}0\\1\\1\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}\begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} \in \Sigma_3$, on aura $w[1] = 0101_2 = 5, w[2] = 1100_2 = 12, w[3] = 1001_2 = 9$

1. Montrer que le language $L = \{w \in \Sigma_2^* \mid w[1] \leq w[2]\}$ est régulier.
2. Est-ce que le langage $L = \{w \in \Sigma_3^* \mid w[0] = w[1] \times w[2]\}$ est régulier ?

On définit l'arithmétique de Presburger comme les formules du premier ordre qui peuvent être construite avec le prédicat R_+ d'arité 3, représentant moralement $R_+(x, y, z) \Leftrightarrow x = y + z$, et le prédicat d'égalité. Pour cela, pour φ une formule avec $\{x_1, ..., x_n\}$ comme variable libre, on montrer par induction sur φ qu'il existe un automate $\mathcal{A}(\varphi)$ sur Σ_n tel que

$$L(\mathcal{A}) = \{w \in \Sigma_n^* \mid \varphi(x_1 = w[1], ..., x_n = w[n]) \text{ est vraie}\}$$

3. Faites le cas des formules "$x = y$" et $R_+(x, y, z)$.
4. Montrer que si on possède un automate $\mathcal{A}(\varphi)$ et un automate $\mathcal{A}(\psi)$ alors on a un automate pour $\varphi \wedge \psi$ et $\neg\varphi$. On fera attention aux alphabets et aux ensemble sde variables libre. On l'admettera pour $\varphi \to \psi$ et $\varphi \vee \psi$.
5. Montrer que si on possède un automate pour $\mathcal{A}(\varphi)$ et que x est une variable libre de φ alors $\mathcal{A}(\exists x.\varphi)$ existe.
6. En déduire un algorithme qui prend en argument une formule du premier ordre de presburger et qui teste si elle est satisfiable.

Cloture par sur-mot[105]

Soit Σ un alphabet avec $|\Sigma| > 1$. On dit que $w \in \Sigma^*$ est un *sous-mot* de $u \in \Sigma^*$ (ou que u est un *sur-mot* de w), noté $w \preceq u$, s'il existe $\varphi : [\![1, ..., |w|]\!] \longrightarrow [\![1, ..., |u|]\!]$ strictement croissante telle que

$$\forall 0 < i \leq |w|, w_i = u_{\varphi(i)}$$

Pour L un language, on note $\hat{L} = \{w \in \Sigma^* \mid \exists u \in L, u \leq w\}$

1. Montrer que si L est régulier, alors $\hat{L}$ aussi.

[104] Oral de l'X

[105] Oral ENS Ulm, je ne sais plus quand

On admet le lemme d'higman suivant pour l'instant:
Lemme d'Higman Soit $(u_n)_{n\in\mathbb{N}} \in (\Sigma^*)^{\mathbb{N}}$ une suite de mot, il existe $i < j$ tel que $u_i \preceq u_j$

2. En addmettant le lemme d'higman, montrer que pour tout language L, il existe un language fini F tel que $\hat{F} = \hat{L}$. En déduire que tout langage clos par sur-mot est régulier.
3. $(*)$ Montrer le lemme d'Higman

Automates (n, d)-locaux $(*)$

On cherche à charactériser les langages réguliers de la forme $\Sigma^* \setminus \Sigma^* I \Sigma^*$ pour I un ensemble de facteurs "interdit" fini.

On dit qu'un automate $\mathcal{A} = (A, Q, I, \delta, F)$ est (n, d)-local avec $d \leq n$ ssi $I = F = Q$ et que pour tout $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$ et $q'_0 \xrightarrow{a_1} q'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q'_n$ (de même étiquettage), on a $q_d = q'_d$. Un automate est dit local s'il est (n, d)-local pour des certains n, d.
Attention à ne pas confondre avec la définition d'automate local du cours, qui se rapproche des automates $(1, 1)$-locaux.

1. Montrer que les langages de la forme $\Sigma^* \setminus \Sigma^* I \Sigma^*$ sont reconnu par des automates locaux.
2. Montrer que si $\mathcal{A}$ est fortement connexe déterministe, alors $\mathcal{A}$ est local ssi il n'existe pas de mot w et deux états distincts q, q' tel que $q \xrightarrow{w} q$ et $q' \xrightarrow{w} q'$.
3. Soit $\mathcal{A}$ un automate (n, d)-local. On pose $I = \left\{ w \in \Sigma^{n+1} \mid \neg\, \exists p, q \in Q, p \xrightarrow{w} q \right\}$. Montrer que $L(\mathcal{A}) = \Sigma^* \setminus \Sigma^* I \Sigma^*$
4. Montrer qu'il existe un algorithme polynomial pour tester si un automate fortement connexe dénote un langage local ou non.

Langages continuables et mots primitifs

On dit qu'un langage L est continuable si $\forall u \in L, \exists v \in \Sigma^*$ tel que $uv \in L$. On dit qu'un mot $w \in \Sigma^*$ est primitif s'il n'existe pas de $p > 1$ et de $x \in \Sigma^*$ tel que $x^p = w$.

1. Montrer qu'il est possible de tester si un mot w est primitif en $O(|w|^{1.5})$. *C'est possible de le faire en $O(|w|)$*
2. Proposer une condition nécessaire et suffisante sur un automate pour que le langage reconnu soit continuable.
3. Montrer que tout langage continuable régulier sur $\Sigma = \{a, b\}$ admet une infinité de mot primitif.
4. $(*)$ Montrer que tout langage continuable régulier sur $\Sigma = \{a, b\}$ admet une infinité de mot non primitif.

MPI: Langages Hors-contexte

Cours

- Grammaire, grammaire hors-contexte, dérivation.
- Grammaires générales, contextuelles (HP)
- Arbre d'analyse, dérivation à gauche, à droite.
- Ambiguité d'une grammaire. Ambiguité du sinon pendant.
- Forme normale de Chompsky (HP)

Question de Cours

- Montrer que les langages réguliers sont hors contexte par induction.
- Montrer que les langages réguliers sont hors contexte par construction de la grammaire à partir de l'automate.
- Montrer que l'intersection d'un langage régulier et d'un langages hors-contexte est hors-contexte.

To Do

- Sujet sur les images de Parikh

Langage hors-contexte

Soit L, L' deux langages hors-contexte. Montrer que les langages suivants sont hors-contexte:

- $\left\{w_1 w_1 w_2 w_2 ... w_{|w|} w_{|w|} : w \in L\right\}$
- $\overline{L} = \{\overline{u} : u \in L\}$ le langages des mirroirs, avec $\overline{u_1 ... u_n} = u_n ... u_1$
- $\bigcup_{w \in L} \text{Pref}(w)$ pour $\text{Pref}(w)$ l'ensemble des préfixes de w
- $\bigcup_{w \in L} S(w)$ pour $S(w)$ l'ensemble des sous-mots de w
- $\{w_1 w_3 w_5 .... w_i : w \in L\}$, une lettre sur deux des mots de L

Paires différentes

On pose $\Sigma = \{a, b\}$. On considère tout d'abord les langages suivants

$$L_a = \{uav : u, v \in \Sigma^* \mid |u| = |v|\}$$
$$L_b = \{ubv : u, v \in \Sigma^* \mid |u| = |v|\}$$
$$L_1 = \{ww' \mid w \neq w' \wedge |w| = |w'|\}$$

1. Montrer que L_a et L_b sont hors-contexte.
2. Montrer que $L_1 = L_a L_b \cup L_b L_a$. En déduire que L_1 est hors-contexte.

On admet que $L' = \{uu : u \in \Sigma^*\}$ n'est pas hors contexte

3. Montrer que les langages hors-contexte ne sont pas stable par complémentaire.

Autant de lettres

Soit $\Sigma = \{a, b\}$, on pose $f : \Sigma^* \to \mathbb{Z}$ tel que $f(w) = |w|_a - |w|_b$. On considère les langages suivants:

$$L = \{w \in \Sigma^* \mid |w|_a = |w|_b\} = f^{-1}(\{0\})$$
$$L_+ \text{ engendré par } S \to SS \mid aSb \mid \varepsilon$$
$$L_- \text{ engendré par } S \to SS \mid bSa \mid \varepsilon$$

On cherche à trouver une grammaire non-ambigue pour L (les mots avec autant de a que de b)

1. Est-ce que la grammaire définissant L_+ est ambigue ? Donner une grammaire pour L (sans preuve)
2. Montrer que L_+ est l'ensemble des mots w tel que $f(w) = 0$ et que pour tout w' préfixe de w on a $f(w') \geq 0$
3. En remarquant qu'un mot de L_+ commence forcément par un a, donner une grammaire non-ambigue pour L_+.
4. Donner une grammaire non-ambigue pour L est hors-contexte

Forme normale de Chompsky

Soit G une grammaire, montrer qu'il existe une grammaire équivalente à G tel que toutes les règles sont soit de la forme $X \to a$ pour a un symbole terminal, soit $X \to AB$ pour A, B non terminaux, soit $S \to \varepsilon$ pour S le symbole initial.

Morphisme de langage hors-contexte

On dit que $\varphi : \Sigma_1^* \longrightarrow \Sigma_2^*$ est un morphisme si pour tout $u, v \in \Sigma_1^*$, $\varphi(uv) = \varphi(u)\varphi(v)$. Montrer que si L est hors-contexte, alors $\varphi(L)$ l'est aussi.

Lemme d'itération pour les langages hors-contexte

On veut montrer que $L_3 = \{a^n b^n c^n : n \in \mathbb{N}\}$ n'est pas hors contexte. Pour cela on montre un lemme analogue au lemme de l'étoile mais pour les langages hors-contexte.

Soit $G = (\Sigma, \Gamma, R, S)$ une grammaire hors-contexte reconnaissant le langage L. On définit $\|R\|_\infty = \max\{|w| : (X \to w) \in R\}$

1. Montrer que si $w \in L$ est un mot tel que $|w| > |\Gamma| \times \|R\|_\infty$, alors il existe $A \in \Gamma$ et $u, v, x, y, z \in \Sigma^*$ avec $xz \neq \varepsilon$ tel que $S \Rightarrow^* uAv$ et $A \Rightarrow^* xAz$ et $A \Rightarrow^* y$.
2. En déduire le lemme *d'itération pour les langages hors-contexte*: pour tout L un langage hors contexte, il existe $N > 0$ tel que pour tout $w \in L$ tel que $|w| > N$ il existe $u, x, y, z, v \in \Sigma^*$ tel que
   - $uxyzv = w$
   - $uxyzv = w$
   - $\forall n \in \mathbb{N}, ux^n yz^n v \in L$
3. En déduire que $\{a^n b^n c^n : n \in \mathbb{N}\}$ n'est pas hors-contexte.
4. Montrer que l'ensemble des langages hors-contexte n'est pas stable par intersection.

Automates à pile

On cherche à donner un modèle de calcul équivalent aux grammaires hors-contexte: les *automates à pile*. Un automate à pile est un tuple$\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ où :
- Σ est l'alphabet d'entrée
- Q est un ensemble fini d'*états* et $q_0 \in Q$ est l'état initial,
- Γ est l'alphabet de pile, et $Z_0 \in \Gamma$ est le symbole initial de pile,
- $F \subseteq Q$ est l'ensemble des états finaux,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma^*)$ est une relation finie de transition.

Une *configuration* est un couple $(q, p) \in Q \times \Gamma^*$. Pour $\alpha \in \Sigma \cup \{\varepsilon\}$ et $\beta \in Z$, on dit qu'il y a *une transition* de la configuration $(q, \beta p)$ vers $(q', p'p)$ si $(q', p') \in \delta(q, \alpha, \beta)$. On note cela

$$(q, \beta p) \xrightarrow{\alpha} (q', p'p).$$

S'il existe une suite de transition $(q, p) \xrightarrow{a_1} \ldots \xrightarrow{a_n} (q', p')$ avec $u = a_1 \ldots a_n$, on note cela $(q, p) \xrightarrow{u} (q', p')$. Remarquer que l'on n'a pas forcément $|u| = n$. On dit qu'un mot u est reconnu par $\mathcal{A}$ si $(q_0, Z_0) \xrightarrow{u} (q_f, \varepsilon)$ avec $q_f \in F$. On note par $L(\mathcal{A})$ le langage des mots reconnu par $\mathcal{A}$ un automate à pile. Ces automates sont connu pour être "acceptant par pile vide" dans la littérature.

1. Montrer que $\{a^n b^n : n \in \mathbb{N}\}$ est un langage reconnu par un automate à pile.
2. Montrer que si un langage est régulier alors il est reconnu par un automate à pile.

On va montrer que tout langage hors-contexte est reconnu par un automate à pile. Soit $G = (\Sigma, \Gamma, R, S)$ une grammaire hors-contexte. On construit un automate à pile $\mathcal{A}_G = (Q, \Sigma, \Gamma', \delta, q, Z_0, F)$ de la manière suivante:
- On pose $\Gamma' = \Gamma \cup \Sigma$, $Q = \{q_0\}$, $F = \{q\}$ et $Z_0 = S$,
- Pour chaque règle $(A \to w) \in R$, on ajoute (q, w) à $\delta(q, \varepsilon, A)$,
- Pour chaque lettre terminale $a \in \Sigma$, on ajoute (q, ε) à $\delta(q, a, a)$.

3. Dessiner l'automate à pile associé à la grammaire $S \to aSbS \mid \varepsilon$.
4. Soient $u \in \Sigma^*$ et $p \in (\Gamma \cup \Sigma)^*$. Montrer que $(q, S) \xrightarrow{u} (q', p')$ si et seulement si $S \Rightarrow^* up$
5. Montrer que $L(\mathcal{A}_G) = L(G)$, et en déduire que tout langage hors-contexte est reconnu par un automate à pile.

On va maintenant montrer la réciproque par la méthode des triplets de Ginsburg. Soit $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q, Z_0, F)$ un automate à pile. Pour tout $q, q' \in Q$ et $\gamma \in \Gamma$, on pose

$$L_{[q,\gamma,q']} = \left\{ u \in \Sigma^* \mid (q,\gamma) \xrightarrow{u} (q',\varepsilon) \right\}$$

6. Montrer que si $(q,\gamma) \xrightarrow{u} (q',\varepsilon)$, alors pour tout $p \in \Gamma^*$ on a $(q,\gamma p) \xrightarrow{u} (q',p)$.
7. Montrer que, pour tout $q, q' \in Q$ et $\gamma \in \Gamma$,

$$L_{[q,\gamma,q']} = \left\{ \alpha \in \Sigma \cup \{\varepsilon\} : (q,\gamma) \xrightarrow{\alpha} (q',\varepsilon) \right\} \cup \bigcup_{\alpha \in \Sigma \cup \{\varepsilon\}} \bigcup_{(q,\gamma) \xrightarrow{\alpha} (q_1, \gamma_1 \dots \gamma_k)} \bigcup_{q_2,\dots,q_{k-1} \in Q} \alpha L_{[q_1,\gamma_1,q_2]} L_{[q_2,\gamma_2,q_3]} \dots L_{[q_{k-1},\gamma_k,q']}$$

8. En déduire qu'il existe une grammaire $G = (\Sigma, \Gamma', R, S)$ avec $\Gamma' = Q \times \Gamma \times Q$ tel que $L(G) = L(\mathcal{A})$

Mélange et grammaires

On fixe $\Sigma = \{a, b, c\}$. Pour deux mots $u, v \in \Sigma^*$, on dit que $w \in \Sigma^*$ est un entrelacement de u et v s'il existe une partition $\{i_1, \dots, i_n\} \cup \{j_1, \dots, j_m\}$ de $[\![1; |w|]\!]$ avec $i_1 < \dots < i_n$ et $j_1 < \dots < j_m$ tel que $u = w_{i_1} \dots w_{i_n}$ et $v = w_{j_1} \dots w_{j_m}$. On note $u ш v$ l'ensemble des entrelacements de u et v.

Pour L, L' deux langages sur Σ, on définit

$$L ш L' = \bigcup_{\substack{u \in L \\ v \in L'}} u ш v$$

1. Donner ab ш ac.
2. Proposer un algorithme qui prend en entrée $u, v, w \in \Sigma^*$ et qui teste si $w \in u ш v$.

On admet que $L_3 = \{a^n (bc)^n a^n : n \in \mathbb{N}\}$ n'est pas un langage hors-contexte.

3. Est-ce que si L, L' sont hors-contexte, alors $L ш L'$ est forcément hors-contexte?
4. Montrer que si L est hors-contexte et R régulier, alors $L ш R$ est hors-contexte.

Circular shift

Soit L un langage. On définit

$$\mathrm{Circ}(L) = \{uv \mid vu \in L\}.$$

Question 1 Montrer que si L est régulier, alors $\mathrm{Circ}(L)$ l'est aussi.

On cherche maintenant à généraliser le résultat pour les langages hors-contexte. Soit $G = (\Sigma, \Gamma, R, S)$ une grammaire hors-contexte telle que $L = L(G)$. Pour $A \in \Gamma$, on pose

$$C_A = \{vu \in \Sigma^* \mid S \Rightarrow^* uAv\}$$

Question 2 Montrer que pour tout $A \in \Gamma$, C_A est hors-contexte.

Question 3 Montrer que pour tout $u, v \in \Sigma^*$ tel que $uv \in L \setminus \{\varepsilon\}$, il existe $A \in \Gamma$ et $x, y \in \Sigma^*$ et $x', y' \in (\Sigma \cup \Gamma)^*$ tel que $S \Rightarrow^* xAy$ et $A \Rightarrow^1 x'y'$ avec $xx' \Rightarrow^* u$ et $yy' \Rightarrow^* v$

Soit $A \to w_1 \dots w_n$ une règle de G, pour tout $0 \leq i \leq n$, on pose $L_A^{i-} = \{u \in \Sigma^* \mid w_1 \dots w_i \Rightarrow^* u\}$ et $L_A^{i+} = \{u \in \Sigma^* \mid w_{i+1} \dots w_n \Rightarrow^* u\}$. On remarque qu'ils sont presque par définition hors conexte.

Question 4 En considérant le langage suivant, montrer que $\mathrm{Circ}(L)$ est hors-contexte:

$$E \cup \bigcup_{(A \to w_1 ... w_n) \in R} \bigcup_{0 \leq i \leq n} L_A^{i-} C_A L_A^{i+}$$

où $E = \{\varepsilon\}$ si $\varepsilon \in L$, et $E = \emptyset$ sinon.

Théorème de Chomsky-Schützenberger[106]

Soit $n \in \mathbb{N}^*$, on définit $\Sigma_n = \{a_1, \overline{a_1}, ..., a_n, \overline{a_n}\}$. Les lettres a_i représentent des parenthèses ouvrantes et les lettres $\overline{a_i}$ représentent des parenthèses fermanantes.

On considère la grammaire $\mathcal{G}_n$ engendrée par les règles

$$S \to a_1 S \overline{a_1} S \mid ... \mid a_n S \overline{a_n} S \mid \varepsilon$$

On pose $D_n = L(\mathcal{G}_n)$ le langage des mots de dyck engendrée par $\mathcal{G}_n$.

1. Donner un arbre de dérivation pour le mot $a_1 a_2 \overline{a_2 a_1} a_3 \overline{a_3}$

On dit que $\varphi : \Sigma_1^* \to \Sigma_2^*$ et un morphisme de mot si $\forall uv \in \Sigma_1^*, \varphi(uv) = \varphi(u)\varphi(v)$

2. Donner une expression régulière dénotant $\varphi(D_1)$ pour φ telle que $\varphi(a_1) = aa$ et $\varphi(\overline{a_1}) = a$
3. Montrer que si L est régulier alors $\varphi(L)$ aussi. Meme question si L est hors-contexte.

On s'intéresse à montrer le théorème de Chomsky-Schützenberger:

> Un langage L est hors-contexte si et seulement il existe un langage régulier K,
> un langage de Dyck D_n et un morphisme alphabétique φ tels que $L = \varphi(D_n \cap K)$.

4. Montrer que l'intersection d'un langage régulier et d'un langage hors-contexte est hors-contexte. En déduire un sens du théorème.

On admet que tout grammaire peut être mise en forme normale de Chompsky, c'est à dire que toutes les règles de la grammaire sont soit $X \to YZ$, soit $X \to \alpha$ ou soit $S \to \varepsilon$ avec $Y, Z \in \Gamma$ et $\alpha \in \Sigma$ et S le symbole initial.

Soit $G = (\Sigma, \Gamma, S, R)$ une grammaire hors-contexte sous forme normale de chompsy. On ordonne les $k := |R|$ règles $r_1, ..., r_k$. On pose $G' = (\Sigma', \Gamma', S, R')$ avec:

$$\Sigma' = \Sigma \cup \{\overline{\alpha} : \alpha \in \Sigma\} \cup \bigcup_{i \in [k]} \left\{a_i, \overline{a_i}, b_i, \overline{b_i}, c_i, \overline{c_i}\right\}$$

Et les règles R' sont:
- $X \to a_i b_i Y \overline{b_i} c_i Z \overline{c_i a_i}$ pour chaque $r_i = X \to YZ$
- $X \to \alpha\overline{\alpha}$ pour toute règle de la forme $X \to \alpha$

5. Donner un morphisme de mot φ tel que $L(G) = \varphi(L(G'))$
6. Proposer un langage régulier K tel que $K \cap D_n = L(G)$. Conclure la preuve du théorème.

Image de Parikh de langage[107]

1. Montrer que tous les langages régulier sont hors-contexte.

On cherche à montrer que la réciproque est vraie sur $\Sigma = \{a\}$. Pour cela, on va étudier l'image de parikh d'un langage.

Soit $\Sigma = \{a_1, ..., a_k\}$ un alphabet fini. Pour un mot $w \in \Sigma^*$, on définit son *image de Parikh* par $\Psi(w) = \left(|w|_{a_1}, ..., |w|_{a_k}\right) \in \mathbb{N}^k$. Pour un langage $L \subseteq \Sigma^*$, on pose $\Psi(L) = \{\Psi(w) : w \in L\} \subseteq \mathbb{N}^k$.

---

[106] Exos de la martinière MPI

[107] Tiré d'une preuve simplifié https://arxiv.org/pdf/2301.00047

Un ensemble $S \subseteq \mathbb{N}^k$ est dit *linéaire* s'il existe $b, p_1, ..., p_m \in \mathbb{N}^k$ tels que

$$S = \{b + \lambda_1 p_1 + ... + \lambda_m p_m : (\lambda_1, ..., \lambda_m) \in \mathbb{N}^m\}.$$

Un ensemble est dit *semi-linéaire* s'il est union finie d'ensembles linéaires.

1. Calculer l'image de Parikh des langages suivants:
   - $L_1 = \{a^n b^n : n \in \mathbb{N}\}$,
   - $L_2 = \{a^n b^m c^{n+m} : n, m \in \mathbb{N}\}$,
   - $L_3 = L(ab^*)$,
2. Montrer que si $A, B \subseteq \mathbb{N}^k$ sont semi-linéaire, alors $A \cup B$ et $A + B$ le sont aussi.
3. Montrer que si L est régulier, alors $\Psi(L)$ est semi-linéaire.

On admet que tout grammaire peut être mise en forme normale de Chompsky, c'est à dire que toutes les règles de la grammaire sont soit $X \to YZ$, soit $X \to \alpha$ ou soit $S \to \varepsilon$ avec $Y, Z \in \Gamma$ et $\alpha \in \Sigma$ et S le symbole initial. Soit $G = (\Sigma, \Gamma, R, S)$ une grammaire hors-contexte sous forme normale de Chompsky.

On dit d'un arbre de dérivation T d'un mot $w \in (\Sigma \cup \Gamma)^*$ qu'il est un *arbre plein* si $w \in \Sigma^*$ et que c'est un *arbre-bloc* un arbre de dérivation si w est de la forme uAv pour $A \in \Gamma$ et $u, v \in \Sigma^*$. Pour un arbre plein T on pose $\Psi(T) = \Psi(w)$ et pour un arbre-bloc T_w de forme $A \Rightarrow^* uAv$, on pose $\Psi(T) = \Psi(uv)$.

4. Donner un exemple d'arbre-bloc pour la grammaire $S \to aSb \mid \varepsilon$. Calculer son image de Parikh.

Soit T un arbre de dérivation. On dit que T est *minimal* s'il ne contient pas strictement un arbre-bloc comme sous-arbre.

5. Montrer qu'un arbre minimal ne peut pas contenir deux occurrences d'un même non-terminal sur une même branche.

TODO: FINISH.

MPI: Graphes avancé

Cours

- Chemin alternant.
- Cliques et graphe biparti. Biparti ssi pas de cycle impair.
- Couplage maximal, maximum, parfait.
- Arbre couvrant, Kruksal
- Tri topologique, recherche de composante fortement connexes, Kosaraju
- Algo couplage max dans biparti
- Sous-graphe, graphe induit (HP)
- Multigraphe, hypergraphe (HP)
- k-coloration (HP)
- Graphes planaire (HP)

Question de Cours

- Montrer qu'un couplage est maximum si et seulement s'il n'existe pas de chemin alternant augmentant pour ce couplage.
- Montrer qu'un graphe est biparti ssi il ne contient pas de cycles impair

Graphes à unique sortie

On dit que $G = (V, E)$ un graphe orienté est un *graphe à unique sortie* si $\forall v \in V, d_+(v) = 1$. Dans ce cas, on le représentera par un tableau T de longueur $|V|$ tel que $T[i]$ corresponde à l'unique sommet $u \in V$ tel que $(i, u) \in E$.

Question 1 Dessiner le graphe correspondant au tableau $[0, 2, 1, 2, 4, 1, 3]$

Question 2 Montrer que le graphe contient autant de cycles que de composantes faiblement connexe.

Question 3 Proposer un algorithme pour calculer le nombre de cycles du graphe.

Question 4 Proposer un algorithme qui teste si le graphe possède un couplage parfait dans sa version non orienté (où l'on ignore les directions).

Kruskal en recherche locale

Soit $G = (V, E)$ un graphe connexe pondéré par $w : E \to \mathbb{R}_+$, on pose $\text{Couv}(G)$ l'ensemble des arbres couvrants. Si $T = (V, E')$ est un arbre couvrant, on note $T[e' \leftarrow e]$ le graphe obtenu en retirant l'arête $e' \in E'$ de T et en y ajoutant l'arête $e \notin E'$.

Question 1 Dans quel cas est-ce que $T[e' \leftarrow e]$ est aussi un arbre couvrant ?

Question 2 Montrer qu'une arête $e \in E$ est choisie par tous les arbres couvrants si et seulement si $G - e$ n'est plus connexe.

Question 3 Montrer que s'il n'existe pas de e, e' tel que $w(T[e' \leftarrow e]) < w(T)$ avec $T[e' \leftarrow e] \in \text{Couv}(G)$, alors T est un arbre couvrant de poids minimal.

On considère alors l'algorithme qui commence sur un arbre couvrant quelquonque T et qui remplace T par $T[e' \leftarrow e]$ tant que c'est possible pour diminuer $w(T)$. C'est ce que on appelle un algorithme de *recherche locale* car il (localement) change l'arbre de manière à atteindre un arbre couvrant de poids minimum.

Question 4 Donner la complexité d'un tel algorithme.

Graphes k-réguliers

On dit que $G = (V, E)$ est un graphe k-régulier si $\forall v \in V, \deg(v) = k$

Question 1 Proposer un algorithme qui donne un couplage maximal dans un graphe 2-régulier.

Question 2 Montrer qu'il existe un graphe k-régulier à $2n$ sommets si $2n \geq k + 1$

Question 3 Montrer qu'il existe un graphe $2k$-régulier à n sommets si $n \geq 2k + 1$

Question 4 Montrer qu'il existe un graphe k régulier si et seulement si $n \geq k + 1$ avec nk pair.

Théorème de Hall

Soit $G = (V, E)$ un graphe, soit $A \subseteq V$, on définit l'ensemble $\mathcal{V}(A)$ des *voisins* de A dans G par

$$\mathcal{V}(A) = \{u \in V \setminus A \mid \exists v \in A, (v, u) \in E\}$$

On dit qu'un graphe biparti $G = (U \sqcup V, E)$ de parties U, V vérifie la *condition des mariages* si pour tout $A \subseteq U$ ou $A \subseteq V$, on a $|A| \leq |\mathcal{V}(A)|$

On cherche à montrer le lemme des mariages: un graphe biparti $G = (U \sqcup V, E)$ de parties U, V admet un couplage parfait si et seulement si G vérifie la condition des mariages.

Question 1 Montrer que G est connexe si et seulement si pour tout $A \subsetneq V$ non vide, $|\mathcal{V}(A)| \geq 1$

Question 2 Montrer que s'il existe $A \subseteq U$ tel que $|A| > |\mathcal{V}(A)|$, alors il n'existe pas de couplage parfait.

Soit G respectant la condition des mariages. Soit H le plus petit ensemble d'arêtes de G tel que $G_H = (V, H)$ respecte toujours la condition des mariages. On note $\deg_H(v) = |\mathcal{V}(\{v\}) \cap H|$

Question 3 Montrer que $|U| = |V|$, et que $\forall v \in V, \deg_H(v) \geq 1$.

Question 4 Montrer que H est un couplage parfait.

Maille d'un graphe

Le *diamètre* de G est la longueur du plus long chemin dans G. On définit la *maille* d'un graphe G comme étant la plus petite longueur d'un cycle de G. On la note, si elle existe, par $\mathcal{M}(G)$. Si la maille existe on dit que le graphe est *maillé*.

Question 1 Soit d le diamètre d'un graphe maillé. Montrer que $\mathcal{M}(G) \leq 2d + 1$

Question 2 Montrer que si G est un graphe biparti maillé, alors $\mathcal{M}(G)$ est pair.

Question 3 Montrer qu'un graphe à n sommets qui contient au plus un cycle contient au plus n arêtes.

Question 4 Montrer que la maille d'un graphe à n sommets et au moins $n + 1$ arêtes est majorée par $\left\lfloor \frac{2n+2}{3} \right\rfloor$

Question 5 Soit $G = (V, E)$ un graphe maillé, on note $m = \mathcal{M}(G)$ et δ le plus petit degré de G. Montrer que

$$|V| \leq \frac{2}{\delta - 2}\left((\delta - 1)^{\frac{m}{2}} - 1\right)$$

Largeur de bande

Soit $G = (V, E)$ un graphe non orienté connexe. Le diamètre de G, noté $d(G)$, est la longueur du plus long chemin dans G. Pour une fonction dite *d'étiquetage* $f : V \to [\![1; |V|]\!]$ bijective, on définit la *largeur de f dans G* comme

$$\psi(f, G) \coloneqq \max\{|f(u) - f(v)| : \{u, v\} \in E\}$$

La *largeur de bande de G*, notée $\psi(G)$, est la plus petite largeur d'une fonction d'étiquetage de G possible

Question 1 Calculer la largeur de bande du graphe de la question 1 (ignorer les poids).

Question 2 Que vaux la largeur de bande d'un cycle à $n \geq 3$ sommets ?

Question 3 Soit $\Delta(G)$ le degré maximal de G, montrer que $\Delta(G) \leq 2\psi(G)$

Question 4 On considère un coloriage des sommets de G tel que chaque arête de G relie deux sommets de couleurs distinctes. Montrer que le nombre minimal de couleurs utilisées pour un tel coloriage est majoré par $\psi(G) + 1$.

Question 6 Montrer que

$$\frac{|S| - 1}{d(G)} \leq \psi(G) \leq |S| - d(G)$$

Graphes planaires

Un graphe $G = (V, E)$ est *planaire* s'il existe $p : V \longrightarrow \mathbb{R}^2$ tel que pour tout $(a, b), (a', b') \in E$, les segments ouverts $]p(a), p(b)[$ et $]p(a'), p(b')[$ ne se croisent pas. Soit $n \in \mathbb{N}$, on dénote par K_n le graphe complet à n sommets. On dénote par $K_{a,b}$ le *graphe biparti complet* contenant $a + b$ sommets, tel que tous les sommets de 1 à a soient relié à tous les sommets de $a + 1$ à b (sans aucune autre aretes).

Une *face* d'un graphe planaire est un cycle délimitant une zone de la représentation (aka une des parties connexe de $\mathbb{R}^2$ auquel on a retiré tous les points dans un segment de G).

Question 1 Montrer que K_4 et $K_{3,2}$ sont planaire

Question 2 Soit $G = (V, E)$ un graphe planaire, on note c le nombre de composante connexes et f le nombre de faces. Montrer que $f + |V| = |E| + c + 1$

Question 3 Montrer que dans tout graphe planaire, $3f \leq 2a$. En déduire que $a \leq 3(n - 2)$ dans un graphe connexe planaire et que donc K_5 n'est pas planaire.

Question 4 Montrer que dans un graphe sans triangle (sans 3 sommets reliés entre eux), $a \leq 2(n - 2)$. En déduire que $K_{3,2}$ n'est pas planaire.

Question 5 Montrer que tout graphe planaire à au moins un noeud de degré ≤ 5.

Question Bonus Montrer que si $G = (V, E)$ est un graphe planaire à plus de 9 sommets, alors $\overline{G} = \left(V, \overline{E}\right)$ n'est pas planaire.

Couverture d'aretes et couplages

Soit $G = (V, E)$ un graphe connexe, on dit que $A \subseteq E$ est une *couverture d'arette* si pour tout $v \in V$, on a un $e \in A$ tel que recouvre v. On dit que A est une couverture d'arêtes *minimale* si A est une couverture d'arête de plus petit cardinal.
Un graphe *étoile* est un graphe tel qu'il existe un sommet u tel que toutes les arêtes sont de la forme $\{u, x\}$.

On cherche à montrer que si M un couplage maximal et A une couverture d'arête minimale, alors

$$|A| + |M| = |V|$$

Question 2 Montrer que s'il existe un couplage parfait M, alors c'est une couverture d'arêtes minimale.

On fixe A une couverture d'arête minimale.

Question 3 Montrer que les graphes étoiles sont exactement les graphes ou il n'existe pas de chemin de longueur strictement plus grande que 2. En déduire que A est une union disjointe de composantes connexe qui sont des graphes étoile.

Question 4 Montrer que $|A| + |M| \geq |V|$

Question 5 Montrer que $|A| + |M| = |V|$. *Indication: On montrer l'inégalité dans l'autre sens. On se fixe un couplage maximal M et on construit une couverture minimale X en prenant M et en ajoutant une arête par sommet non couvert. Borner la taille de X.*

C-approx du couplages maximum[108]

Pour M un couplage, on note $S(M)$ les sommets saturé de M.

[108] TD11 L3 ENS Lyon Algo 1

On considère l'algorithme suivant:

Entrée: G un graphe
$M \leftarrow$ un couplage maximal de G
tant que $\exists \{u', u\}, \{u, v\}, \{v, v'\} \in E$ avec $\{u, v\} \in M$ et $u', v' \notin S(M)$:
$\quad M \longleftarrow (M \setminus \{u, v\{\}\}) \cup \{\{u', u\}, \{v, v'\}\}$
fin tant que
retourner M

1. Quelle est la complexité de l'algorithme ?
2. Montrer que l'algorithme est une C-approx du couplage maximum pour un certain C que l'on déterminera.

On change la boucle pour chercher un chemin alternant P de longueur $2t+1$, et si on en trouve un on effectue $M \longleftarrow (M \setminus P) \cup (P \setminus M)$. L'algorithme donné correspond donc au cas $t = 1$.

3. Donner le facteur d'approximation en fonction de t, et en déduire un *PTAS*, c'est-à-dire que pour tout $\varepsilon > 0$ on a un algorithme polynomial qui renvoie une $(1-\varepsilon)$-approximation. On déterminera exactement la complexité en fonction de ε.

Calcul de triangles[109]

Question 1 Montrer qu'un graphe est biparti si et seulement si il n'admet pas de cycle impair.

Définition Pour $G = (V, E)$ un graphe non orienté avec $V = \{1, ..., n\}$, on dit que $\{x, y, z\} \subseteq V$ est un *triangle* si $\{x, y\}, \{y, z\}, \{z, x\} \in E$. On cherche à calculer tous les triangles sans doublons d'un graphe G.

Question 2 Proposer un algorithme en $O(|V|^3)$

Question 3 Soit L_1, L_2 deux listes trié d'entiers de 1 à n, montrer que l'on peut calculer la liste triée des élément appartenant aux deux listes en temps $O(|L_1| + |L_2|)$.

Question 4 Donner un algorithme en $O(|E| \times \Delta)$ pour calculer tous les triangles sans doublons d'un graphe G ou Δ est le degré maximal de G. On supposera que G est donné sous la forme d'une liste d'adjacence.

Question 5 Dans quels cas est-ce que l'algorithme de la question 4 est meilleur que celui de la question 2 ?

Planaire et degré

Un graphe $G = (V, E)$ est dit k^+-régulier si $\forall v \in V, \deg(v) \geq k$. Un graphe G est dit planaire s'il existe une assignation $\varphi : V \longrightarrow \mathbb{R}^2$ tel que pour toute paire d'arête $(x, y), (x', y') \in E$, les segments $[\varphi(x); \varphi(y)]$ et $[\varphi(x'); \varphi(y')]$ ne se coupent pas.

Soit G connexe planaire. On définie f le nombre de face comme le nombre de composante connexe de $\mathbb{R}^2 \setminus \bigcup_{(x,y)\in E} [\varphi(x); \varphi(y)]$.

1. Montrer le théorème d'Euler: $|E| = |V| + f - 2$
2. En déduire que $|E| < 3|V|$
3. En déduire que si G est planaire alors il n'est pas 6^+-régulier.

[109] Tiré de Mallory Marin

Graphes d'amis sans C_5

Soit $G = (S, A)$ un graphe non orienté. On dit que G est un *graphe d'amis* si pour tout $s_1, s_2 \in S$ avec $s_1 \neq s_2$, il existe un unique s' avec $\{s_1, s'\} \in A$ et $\{s_2, s'\} \in A$. On dit qu'un sommet s^* est un *roi* si pour tout $s \in S \setminus \{s^*\}, \{s, s^*\} \in A$.

On cherche à montrer que un graphe d'amis possède toujours un roi.

1. Montrer que le cycle de longueur 4 n'est pas un graphe d'amis.
2. Montrer que dans un graphe d'amis, toutes les arêtes font parti d'un cycle.
3. Proposer un algorithme pour calculer le roi (s'il en existe un) avec un graphe représenté en matrice d'adjacence.

On admet qu'un graphe d'amis ne possède pas de cycle de longeur 5.

4. Montrer que que si G possède un cycle de longueur ≥ 4 alors G n'est pas un graphe d'amis.
5. Montrer que tout les graphes d'amis possède un roi.

Graphes d'amis 2

Soit $m \geq 3$. Soit $G = (S, A)$ un graphe non orienté tel que pour tout m personnes choisies, il existe un unique $s \in S$ qui est reliées aux m personnes. Quel est le dégrée max du graphe ?

Un Graphe dénombrable

On prend $\varphi : \mathbb{N} \longrightarrow \mathcal{P}_f(\mathbb{N})$ une bijection de $\mathbb{N}$ dans les parties finies de $\mathbb{N}$. Montrer qu'a isomorphisme près, et en retirant les boucles, le graphe $(\mathbb{N}, \{\{x, y\} : y \in \varphi(x)\})$ est unique.

Trouver l'isomorphisme pour φ_1, φ_2 deux bijections.

Graphes d-dégénéré[110]

Un graphe $G = (V_G, E_G)$ est dit d-dégénéré s'il existe un ordre $v_1, ..., v_n$ des sommets de G tel que, pour tout $i \leq n$, le sommet v_i au plus d voisins parmis $\{v_1, ..., v_{i-1}\}$. La dégénéresence d'un graphe, noté $\mathcal{D}(G)$, est le plus petit d tel que G est d-dégénéré.

Question 1 Montrer que si G contient une clique de taille k alors $\mathcal{D}(G) \geq k - 1$

Question 2 Montrer que tout sous-graphe d'un graphe d-dégénérer est aussi d-dégénéré.

Question 3 Montrer que G est d-dégénéré si et seulement si $\forall S \subseteq V_G$, $G[S]$ possède un sommet de degré $\leq d$

Question 4 Donner un algo polynomial qui prend un graphe d-dégénéré et renvoie $(d+1)$-coloration

Coeur de k-domination[111]

Soit $G = (V_G, E_G)$ un graphe. On dit que $D \subseteq V_G$ domine X si $X \subseteq N(D)$ avec $N(A)$ les voisins de A (incluant A). Si $X = V_G$, on dit que D domine X. On dit que D *quasi-domine* X si D domine tout X sauf au plus un point.

Soit $k \geq 2$. Un coeur de k-domination est un X tel que tout dominant de X de taille k domine nécessairement tout le graphe.

Question 1 Soit X tel que $\forall x \in V_G, |N(v) \cap X| < \left\lfloor |X \frac{\downarrow}{k} \right\rfloor$. Montrer que G n'admet pas d'ensemble quasi-dominant de X de taille au plus k.

---

[110] InfoFonda 2026 partie I

[111] InfoFonda 2026 partie IV

Un graphe est dit sans $K_{t,t}$ si le graphe biparti complet à deux parties de t sommets n'est pas présent comme sous-graphe. On considère l'algo suivant:

Entrée: Un graphe G sans $K_{t,t}$, $X \subseteq V_G$.
$Y \leftarrow G$
$S \leftarrow \emptyset$
tant que $\exists v \notin S, |N(v) \cap Y| \geq \left\lfloor |Y^{\downarrow}_{\frac{1}{k}} \right\rfloor$ **faire:**
    $S \leftarrow S \cup \{v\}$
    $Y \leftarrow N(v) \cap Y$
fin tant que
retourner Y, S

Question 2 Montrer que si $|X| \geq 2tk^t$ alors l'algorithme termine en au plus $t-1$ itérations de la boucle.

On suppose que $|X| \geq 2tk^t$, et on se donne S, Y le résultat de l'algorithme

Question 3 Montrer que tout ensemble quasi-dominant de taille au plus k intersecte S. En déduire que, soit $y \in Y$, si D domine $G - y$ alors D domine G

Question 5 En déduire un algorithme polynomial qui prend un $k \geq 2$ et un graphe G sans $K_{t,t}$ qui retourne un coeur de k-domination de taille au plus $2tk^t$.

Théorème de vizig[112]

Soit $G = (V, E)$ un graphe, on note Δ le degré maximal d'un graphe, on dit que $\gamma : E \longrightarrow \{1, ..., k\}$ est une *k-coloration d'aretes* si pour toute paire d'arête e, e' partageant une extrémité, on a $\gamma(e) \neq \gamma(e')$. Une coloration est dite partielle si la fonction l'est.

On cherche à montrer que tout graphe est $(\Delta + 1)$-coloriable en arête.

Question 1 Montrer que tout graphe est $2\Delta - 1$-arête coloriable

Question 2 Montrer qu'un graphe ne peux pas être k-coloré pour $k < \Delta$.

Pour γ une coloration on note $\overline{\gamma}(u)$ l'ensemble des couleurs des arête d'extrémité u qui **ne** sont **pas** prise. Soient a, b deux couleurs, on définit $K(a/b)$ la **chaine de Kempe** comme étant un ensemble d'arettes maximal pour l'inclusion contenant que des arettes de couleur a ou b

Question 3 Montrer que $G[K(a/b)]$ est une union disjointe de cycle de longueur pair ou de chemins.

Question 4 Soit G biaprti, γ une coloration partielle, $(u, v) \in E_G$ avec $a \in \overline{\gamma}(u)$ et $b \in \overline{\gamma}(v)$. Montrer que si u et v ne peuvent pas être dans la même composante connexe de $K(a/b)$

Question 5 Montrer que si G est biparti alors G est $\Delta(G)$-arête-colorable

Question 6 Soit $(u, v) \in E$ une arête non coloré d'une coloration partielle γ et soient a, b deux couleurs telle que $a, b \in \overline{\gamma}(u)$ et $b \in \overline{\gamma}(v)$. Montrer qu'il existe γ' qui colorie les même arêtes telle que

- $\overline{\gamma'}(u) = \overline{\gamma}(u)$,
- $a \in \overline{\gamma'}(v)$ et
- $\overline{\gamma'}(x) = \overline{\gamma}(x)$ pour tout $x \in V \setminus \{u, v\}$ sauf au plus un qui vérifie alors $\left(\overline{\gamma'}(x) \setminus \overline{\gamma}(x)\right) \cup \left(\overline{\gamma}(x) \setminus \overline{\gamma'}(x)\right) = \{a, b\}$

[112] InfoFonda 2026 partie V

Question 6 Soit γ une k-coloration partielle de G dont tous les arêtes non coloré $e_1, ..., e_r$ sont toutess de la forme $e_i = (u, v_i)$. On suppose que $|\overline{\gamma}(u)| \geq r$, $|\overline{\gamma}(v_1) \cap \overline{\gamma}(u)| \geq 1$ et que pour tout $i \geq 2$, on a $|\overline{\gamma}(v_i) \cap \overline{\gamma}(u)| \geq 2$. Montrer que G est k-colorable.

Question 7 En déduire que tout graphe est $(\Delta + 1)$-arête-colorable.

Question 8 Montrer que pour tout $d > 1$ il existe un graphe G non d-arete colorable de degré maximum d.

FPT: Kernelisation de Feedback vertex set

On considère le problème FEEDBACK-VERTEX-SET suivant:

- **Entrée:** Un multigraphe $G = (V_G, E_G)$ et $k \in \mathbb{N}$
- Est-ce qu'il existe $S \subseteq V_G$ avec $|S| \leq k$ tel que pour tout cycle C de G, $C \cap S \neq \emptyset$

1. Montrer que pour $k = 1$ et $k = 2$ le problème est polynomial
2. Soit (G, k) une instance et v un sommet de degré 1. Montrer que (G, k) est une instance positive ssi $(G - v, k)$ l'est aussi.
3. Soit G une instance et v un sommet de degré 2 relié $u, u' \neq v$. Montrer que (G, k) est une instance positive ssi $(G - u + \mathrm{uu}', k)$ l'est aussi
4. Soit G une instance et v un sommet avec une boucle. Montrer que (G, k) est une instance positive ssi $(G - v, k - 1)$ l'est aussi.

Soit G le graph obtenu après exécution des deux règles précédentes autant de fois que possible. On a donc que le degré minimum de G est 3. On note V_{3k} les $3k$ sommets de plus grand degré. Soit S est une solution de FEEDBACK-VERTEX-SET, on suppose par l'absurde que S n'intersecte pas V_{3k}.

5. Montrer que, pour $d = \min_{x \in V_{3k}} \deg(x)$, on a
$$\sum_{v \in X \cup V_{3k}} \deg v \geq 3|X| + 3kd$$
6. Montrer que $G[X \cup V_{3k}] \leq |X| + 3k - 1$
7. En déduire une contradiction.
8. Montrer alors que pour tout $k \leq \log(\log(n))$ il existe un algorithme polynomial pour FEEDBACK-VERTEX-SET. On donnera la complexité.

FPT: Kernelisation de Vertex cover

On considère le problème VERTEX-COVER suivant:

- **Entrée:** Un graphe $G = (V_G, E_G)$ et $k \in \mathbb{N}$
- Est-ce qu'il existe $S \subseteq V_G$ avec $|S| \leq k$ tel que toutes les arêtes sont adjascente à un $s \in S$?

1. Montrer que pour tout k fixé, le problème est polynomial. *On admet qu'il est NP-Complet dans le cadre général*
2. Soit (G, k) une instance et v un sommet de degré 0. Montrer que (G, k) est une instance positive ssi $(G - v, k)$ l'est aussi.
3. Soit (G, k) une instance et v un sommet de degré $\geq k + 1$. Montrer que (G, k) est une instance positive ssi $(G - v, k - 1)$ l'est aussi.
4. Montrer que après exécution des deux règles précédentes autant de fois que possible, on a $|V| \leq k^2 + k$

5. En déduire que VERTEX-COVER possède un algorithme correcte dont la complexité peut s'exprimer sous la forme $O(f(k)n^c)$ pour f, c que l'on donnera.

Avec une décomposition en couronne on peut réduire la taille de G par d'autres règles tel que $|V| \leq 3k$.

Avec de la programmation linéaire et de la relaxation on peut montrer qu'on peut réduire la taille de G tel que $|V| \leq 2k$.

FPT: Kernelisation de Edge clique cover

On considère le problème EDGE-CLIQUE-COVER suivant:

- **Entrée:** Un graphe $G = (V_G, E_G)$ et $k \in \mathbb{N}$
- Est-ce qu'il existe $S_1, ..., S_k \subseteq V_G$ tel que $G[S_i]$ soit une clique pour tout $i \leq k$ et que $E_G = \bigcup_i E(G[S_i])$

1. Montrer que pour $k = 1$ et $k = 2$ le problème est polynomial
2. Soit (G, k) une instance et v un sommet de degré 1. Montrer que (G, k) est une instance positive ssi $(G - v, k - 1)$ l'est aussi
3. Soit $(u, v) \in E$ tel que $N(u) = N(v)$ (avec $N(x)$ le voisinage de x). Montrer que (G, k) est une instance positive ssi $(G - u, k)$ l'est.
4. Montrer que après exécution des deux règles précédentes autant de fois que possible, on a $|V| \leq 2^k$
4. En déduire que pour $k \leq \log_2(|V|)$, le problème EDGE-CLIQUE-COVER est polynomial.

MPI: Théorie des jeux

Cours

- Définition d'un jeu. Jeu d'accesibilité. Stratégie, stratégie gagnante.
- Calcul des attracteurs. Définition des A_i^j dans l'algorithme du calcul des attracteurs.
- Algorithme min-max, élagage α/β, verison avec heuristique.
- Algorithme A^*
- Zermelo (HP)

Petites questions

- Soit un jeu $G = (V, E)$ avec $V = \mathbb{N}$ et telle que les arètes orienté sont toutes de la forme (x, y) avec $y < x$. On suppose de plus que $\forall x \in V, \deg_+ v = 0 \Rightarrow v \in F_1 \cup F_2$. Montrer que pour chaque état du jeu, un des deux joueurs possède toujours une stratégie gagnante.

Exercices à ajouter:

- Devine le nombre de points fixes, Putnam 2023

Jeu de Chomp

On considère une tablette de chocolat rectangulaire de taille $n \times m$. Les cases sont indexées par (i, j) avec $0 \leq i < n$ et $0 \leq j < m$. La case $(0, 0)$ est empoisonnée.

Deux joueurs (Alice et Bob) jouent à tour de rôle, en commençant par Alice. À chaque coup, un joueur choisit une case (i, j) encore présente, puis mange toutes les cases (i', j') telles que $i' \geq i$ et $j' \geq j$. Le joueur qui mange la case empoisonnée perd.

1. Quel joueur possède une stratégie gagnante si $n = 1$?
2. Montrer que le jeu se termine toujours.
3. Montrer que pour toute tablette rectangulaire $n \times m$ avec $n, m \geq 2$, le premier joueur possède une stratégie gagnante.

Jeu et graphes biparti[113]

Soit $G = (S, A)$ un graphe non orienté, connexe. On s'intéresse à un jeu à deux joueurs : R (rouge) et B (bleu). R commence en choisissant un sommet et en le coloriant en rouge. Puis chacun à son tour, le joueur choisit un sommet non colorié voisin de sommet de la couleur de l'adversaire mais non voisin d'un sommet de sa propre couleur, puis colorie ce sommet de sa couleur. Si aucun sommet n'est jouable, le joueur passe son tour.

Quand aucun joueur ne peut jouer, la partie est terminée. — Si le graphe est entièrement colorié, R gagne. — Sinon, B gagne.

1. Donner la définition d'une stratégie.
2. Donner la définition d'une stratégie gagnante.
3. Donner l'ensemble des états finaux sur le graphe P_4, le chemin de longeur 4, de sommets $[\![0; 4]\!]$. On suppose que R commence par colorier le sommet 1.
4. Montrer que s'il existe une partie gagnée par R, alors G est biparti.

On suppose maintenant que G est biparti. On note (S_1, S_2) bipartition des sommets, et l'on suppose que le premier sommet joué par R est dans S_1.

5. Montrer l'invariant suivant : "Tout sommet colorié en rouge est dans S1, et tout sommet colorié en bleu est dans S2."
6. Montrer que s'il reste des sommets non coloriés, alors au moins l'un d'entre eux est coloriable.
7. En déduire qu'un graphe G est biparti si et seulement si toutes les stratégies de R sont gagnantes.
8. En déduire un algorithme déterminant si un graphe connexe non orienté est biparti. Quelle est sa complexité?

Jeu de nim généralisé[114]

Soit $A \subseteq \mathbb{N}^*$ fini avec $1 \in A$. On considère un jeu à deux joueurs où $N > 0$ objets sont disposés sur une table, et chaque joueur doit à tour de rôle retirer $t \in A$ objets de la table (s'il peut). Le joueur qui retire le dernier objet perd.

On appellera (A)lice le joueur qui commence et (B)ob le deuxième joueur.

1. Montrer qu'il existe toujours une stratégie gagnante depuis l'état initial pour l'un ou l'autre des joueurs.
2. Pour les valeurs de A suivantes, pour quels N Alice possède-t'elle une stratégie gagnante ?
   - $A = \mathbb{N}$
   - $A = \{1\}$
   - $A = \{2k + 1 \mid k \in \mathbb{N}\}$
3. Si $\max(A) \leq N$, proposer un algorithme qui décide si Alice possède une stratégie gagnante pour un N donné.
4. Pour $A = \mathbb{P}$ l'ensemble des nombres premiers, pour quelles valeurs de N Alice possède t'elle une stratégie gagnante ? Et pour $A = \{2^i : i \in \mathbb{N}\}$ l'ensemble des puissances de 2 ? Et pour $\{1, ..., p\}$?
5. Montrer qu'il existe un ensemble A tel que si on note S_A l'ensemble des N tel que Alice possède une stratégie gaganante, alors il n'existe pas $N_0, T > 0, \forall n > N_0, n \in S_A \iff n + T \in S_A$.

[113] CCINP Sujet A 2025

[114] Sujet original !

Géographie dans les hypercubes[115]

Soit $\Sigma = \{a, b, c\}$ et $n \in \mathbb{N}^*$. Alice et Bob joue au jeu suivant: à tour de role, iels choisissent un mot dans Σ^n qui n'a pas déjà été choisi avant, et tel que il ne diffère que d'une lettre avec le dernier mot choisi. Alice choisi un mot quelquonque au début. Le premier joueur ne pouvant pas jouer perd. On cherche à établir une statégie gagnante.

On pose $w_a = \underbrace{a...a}_{n \text{ fois}}$

1. Dessiner le graphe des états pour $n = 1$ et $n = 2$. Qui gagne ?
2. Montrer qu'un joueur X possède une stratégie gagnante ssi et seulement il en possède une pour le jeu ou Alice commence par le mot w_a.

On supposera maintenant sans perte de généralité que le premier coup d'Alice est $a...a$. On pose $G_n = (V_n, E_n)$ le graphe définit par $V_n = \Sigma^n$ et $(u, v) \in E$ si u et v ne diffèrent que d'une lettre.

3. Montrer par récurrence sur $n \in \mathbb{N}^*$ que $G_n - w_a$ possède une couplage parfais
4. En déduire que Alice possède une stratégie gagnante.
5. Généraliser pour $\Sigma = \{a, b, c, d\}$ et puis après pour tout alphabet fini non vide Σ

Géographie[116]

Soit $G = (V, E)$ un graphe, on définit le jeu à deux joueurs (A)lice et (B)ob tel que à tour de role, chaque joueur doit marquer un sommet non déjà marqué qui soit un voisin du dernier sommet marqué. Le premier joueur qui ne peut plus marquer de sommet perd. Le premier joueur (Alice) peut choisir ou commencer.

2. On considère C_n le cycle de longueur n. Pour quel $n \in \mathbb{N}$ est-ce que Alice a une stratégie gagnante sur C_n ?
3. Montrer que si le graphe possède un couplage parfait, alors Bob possède une stratégie gagnante. La réciproque est-elle vraie ?
4. Proposer un algorithme dans le cas ou G est un arbre pour savoir qui possède une stratégie gagnante.
5. On considère le cas ou $V \subseteq \mathbb{N}^2$ et $\forall ((x, y), (x', y')) \in E, |x - x'| + |y - y'| = 1$ avec G connexe. A quelle condition sur $|V|$ est-ce que Alice possède une stratégie gagnante ?

Exercice 2 : Jeux et automates [117]

Soit $A \subseteq \mathbb{N}$ un ensemble d'objectifs et $n \in \mathbb{N}$, Alice et Bob jouent à un jeu à 2 joueurs. A tour de role, Alice et Bob, en commençant par Alice, choisissent 0 ou 1. Après n itérations, le jeu s'arrête et on obtient une séquence $(u_1, ..., u_n)$, avec u_i le choix fait au i-ème tour. Alice gagne si

$$\sum_{1 \leq i \leq n} u_i \times 2^{n-i} \in A$$

1. Proposer un pseudo-code qui teste si Alice possède une stratégie gagnante pour un n et un A fini.
2. Montrer que pour $A = 2\mathbb{N}$, Alice possède une stratégie gagnante ssi n est impair.

On pose $A = 3\mathbb{N}$.

3. Proposer un automate $\mathcal{A}$ sur $\Sigma = \{0, 1\}$ qui reconnaît les écritures binaires des nombres congrus à 0 modulo 3.
4. En utilisant cet automate, déterminer pour quels n Alice possède une stratégie gagnante.

---

[115] Le putnam de 2025

[116] Marathon de math de Paris-Orsay 2021

[117] Sujet fait main!

5. Proposer un algorithme qui décide, pour un entier n et un automate $\mathcal{A}$ si Alice possède une stratégie gagnante pour l'objectif des entiers associés aux mots de $L(\mathcal{A})$. Quelle est sa complexité?
6. Montrer que l'on peut faire la question 5 en complexité $O\big(k2^k\big)$ pour k le nombre d'états de l'automate.

Jeu de Shannon

On considère un jeu avec un graphe $G = (V, E)$ non orienté qui peut posséder plusieurs arêtes entre deux sommets. Deux joueurs (A)lice et (B)ob, où Alice commence, choisissent alternativement une arête de G non encore choisie. Si, à un moment de la partie, les arêtes choisies par Bob forment un arbre couvrant de G alors Bob gagne. Sinon, Alice gagne.

Pour $\mathcal{P}$ une partition de V, on note $|\mathcal{P}|$ le nombre de parties et $\|\mathcal{P}\|$ le nombre d'arêtes de G dont les deux extrémités sont dans des ensembles différents de $\mathcal{P}$.

Question 1 Soit T_1, T_2 deux arbres couvrants de G et e_1 une arête de T_1. Montrer qu'il existe e une arête de T_2 tel que $T_1 - e_1 + e_2$ soit un arbre couvrant de G

Question 2 On suppose qu'il existe une partition $\mathcal{P}$ de V telle que $\|P\| < 2(|P| - 1)$. Montrer que Alice possède une stratégie gagnante.

Question 3 Soit T un arbre couvrant de G et e une arête de T. Soient T' et G' obtenus en contractant e dans T et G, c'est-à-dire en supprimant e et identifiant ses extrémités. Montrer que T' est un arbre couvrant de G'.

Théorème On admet le théorème de Tutte: G possède k arbres couvrant disjoints si et seulement si pour tout partition P de V, $\|P\| \geq k(|P| - 1)$

Question 4 Montrer que Bob a une stratégie gagnante si et seulement si G possède deux arbres couvrants disjoints.

Parité du nombre de points fixes[118]

Soit $n \in \mathbb{N}$. Alice et Bob jouent au jeu suivant: Bob choisi "pair" ou impair, puis à tour de role en commençant par Alice, iels choissisent un nombre entre 1 et n qui n'ont pas déjà été choisi. Bob gagne si, une fois que tous les entiers ont été choisi, le cardinal $|\{1 \leq k \leq n \mid k \text{ choisit au } k-\text{ème tour de jeu}\}|$ correspond à la parité qu'il a choisi.

Pour quels n est-ce que Bob à une stratégie gagnante ?

Exemple de partie pour $n = 3$: Bob choisit "impair" et
- Si Alice choisit 1, Bob choisit 2 et alice doit choisir 3. L'ensemble est $\{1, 2, 3\}$ et Bob gagne.
- Si Alice choisit 2, Bob choisit 1 et alice doit choisir 3. L'ensemble est $\{3\}$ et bob gagne
- Si Alice choisit 3, Bob choisit 2 et alice doit choisir 1. L'ensemble est $\{2\}$ donc Bob gagne.

Donc Bob a une stratégie gagnante pour $n = 3$.

Jeu du sous-mot[119]

Soit Σ un alphabet, on considère le jeu à deux joueurs (dénoté (A)lice et (B)ob) du sous-mot suivant: a tour de role, chaque joueur dit un mot $w \in \Sigma^* \setminus \{\varepsilon\}$ qui ne contient aucun des mots déjà dit en sous-mot. Le premier joueur qui ne peux plus dire de mot perd. Par exemple, sur $\Sigma = \{a, b\}$, la partie pourrait être $abb \to bbbb \to aab \to a \to bbb \to bb \to b$ et Alice perd (Alice étant la joueuse qui commence).

[118] Putnam 2023

[119] Sujet original !

On admettera que ce jeu termine toujours (cette propriété viens du lemme d'higman).

Question 1 On admet pour l'instant que Bob possède une stratégie gagnante pour $|\Sigma|$ pair. Montrer que Alice possède une stratégie gagnante pour $|\Sigma|$ impair.

Question 2 Pour $\Sigma = \{a, b\}$, on définit $\overline{a} := b$ et $\overline{b} := a$, puis $\overline{w_1 ... w_n} = \overline{w_1} ... \overline{w_n}$. Montrer que Bob possède une stratégie gagnante pour $\Sigma = \{a, b\}$.

Question 3 En généralisant la question précédente, montrer que Bob possède une stratégie gagnante pour $|\Sigma|$ pair.

Question 4 $(**)$ Montrer que le jeu termine toujours, autrement dit que si $(w_n)_{n\in\mathbb{N}} \in (\Sigma^*)^{\mathbb{N}}$ est une suite infinie de mots, alors il existe $i < j$ tel que w_i soit un sous-mot de w_j.

MPI: Apprentissage

Cours

- Différece entre apprentissage supervisé et non supervisé.
- Algorithme des k-plus proches voisins.
- Arbre de décision, Matrice de confusion.
- Entropie de Shannon, gain, algorithme ID3.
- Algorithme CHA (Classification hiérachique ascendente)
- L'Algorithme des k-moyennes.

Petites questions

1. Donner un algorithme qui ajoute un sommet $\vec{x} \in \mathbb{R}^k$ à un arbre k-dimensionnel. Quel est sa complexité ?
2. Écrire une fonction `int* voisins1D(float x, float* X, int k, int n)` permettant de trouver efficacement les k plus proches voisins de x dans l'ensemble de n données X trié par ordre croissant, où chaque donnée est un réel (en dimension 1). La fonction renvoie un tableau d'entiers contenant les indices des k plus proches voisins de x dans X. Quelle est sa complexité ?
3. Rappeler l'intêret et le fonctionnement de l'algorithme de classification hiérarchique ascendante.

Single-pass (CCINP 2024)[120]

On considère l'algorithme suivant pour catégoriser des données $X = \{\vec{x_1}, ..., \vec{x_n}\} \subseteq \mathbb{R}^d$. On note $\delta(\vec{x}, \vec{y}) = \sum_{i=1}^{d} |\vec{x}_i - \vec{x}_i|$ la distance entre $\vec{x}$ et $\vec{y}$. La distance de $\vec{x}$ à une classe C est la distance de $\vec{x}$ au centre de C. On note $\mathrm{Cl}(\vec{x}, C)$ la classe dont le centre est le plus proche de $\vec{x}$ pour δ.

Soit $\theta \in \mathbb{R}_+$, on considère:

$C \leftarrow \emptyset$
Pour i de 1 à n :
    Si $\forall \vec{c} \in C, \delta(\vec{x}_i, \vec{c}) > \theta$:
        $C \leftarrow C \cup \{\{\vec{x}_i\}\}$ // On crée une nouvelle classe
    Sinon:
        $\overline{C} \leftarrow \mathrm{Cl}(\vec{x}_i, C)$
        $\overline{C'} \leftarrow \overline{C} \cup \{\vec{x}_i\}$
        $C \leftarrow C \setminus \left\{\overline{C}\right\} \cup \left\{\overline{C'}\right\}$
Renvoyer C

[120] https://prepas-mp2i.fr/documents/sujets/2024/CCINP-INFO.pdf

Question 1 Proposer une implémentation en C de la fonction δ. On représentera un $\vec{x} \in \mathbb{R}^d$ par un `float* x`, et d sera traité comme une constante.

Question 2 Montrer que l'ordonnancement des $\vec{x}_1, ..., \vec{x}_n$ importe.

Question 3 Soit $\vec{x} \in X$, on note C_{ini} la classe de $\vec{x}$ au moment ou il a été ajouté, et C^* sa classe à la fin. Montrer que

$$\delta(\vec{x}, C^*) \leq \theta \ln(|C^*| - |C_{\text{ini}}|)$$

On admettera que $\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{k} \leq \ln(k)$

Question 4 Quel sont les avantages de cet algorithme par rapport aux k-moyennes ? Proposer des améliorations à l'algorithme.

Clustering en dimension 1[121]

Définition Soit $K \in \mathbb{N}^*$ et $E = \{x_0, ..., x_{N-1}\}$ un ensemble de réels avec $x_0 \leq ... \leq x_{N-1}$. Pour $X \subseteq E$, on note μ_X son centre et $S(X)$ son score, formellement définit par

$$\mu_X := \frac{1}{|X|} \sum_{x \in X} x \qquad\qquad S(X) := \sum_{x \in X} (x - \mu_X)^2$$

On pose $S(i, j) = S(\{x_i, ..., x_j\})$. On cherche à résoudre l'algorithme des k-moyennes sur E de manière exacte. C'est à dire que l'on cherche une partition $\mathcal{P} = \{X_1, ..., X_K\}$ de $[\![0; N-1]\!]$ qui minimise la somme des scores de chaque X_i

Question 2 Montrer que l'algorithme CHA ne permet pas de résoudre de manière exacte le problème. On pourra prendre $N = 4$ et $K = 2$.

Question 3 On note $I(n, k)$ le score minimale d'une partition de $\{x_0, ..., x_n\}$ en k classes. Que vaux $I(n, 1)$?

Question 4 Montrer que

$$\forall n > 0, \forall k > 1, I(n, k) = \min_{k-1 \leq m \leq n-1} (I(m, k-1) + S(m, n))$$

Question 5 En déduire une fonction `double inertie(double* E, int N, int K)` qui calcule le score minimale possible d'une parition de E en K classes non vides.

Question 6 Montrer que l'on peut calculer l'ensemble des $S(i, j)$ en $O(N^2)$

k-centres

Définition On fixe $V \subseteq \mathbb{R}^d$ un jeu de donnée. Pour $\vec{y} \in \mathbb{R}^d$ on définit $d(\vec{y}, V) = \max_{\vec{x} \in V} \|\vec{y} - \vec{x}\|$. On cherche à choisir k points de V tel qu'ils minimise $\max_{v \in V} d(v, X)$. On regarde donc le problème k-CENTRE suivant:

Entrée: $v_1, ..., v_n \subseteq \mathbb{R}^d$ une liste de points finis **Sortie:** $X := \{x_1, ..., x_k\} \subseteq V$ qui minimise $\max_{v \in V} d(v, X)$

Question 1 Rapeller le fonctionnement de l'algorithme des k-moyennes. Quel sont les différences avec le problème des k-centres ?

Question 2 On cherche à montrer que l'algorithme du cours de la question ne donne pas de bonne approximation du problème des k-centres. Pour cela on pose $k = 2$, $d = 1$ et on pose N points en 0,

[121] tiré de https://mpi-lamartin.github.io/mpi-info/docs/ia/non_supervise/td_apprentissage

N points en 1 et un point en D. Montrer que pour des bonnes valeurs de N et D on peut avoir le ratio $\frac{\text{opt-k-moyennes}}{\text{opt-k-centre}}$ arbitrairement large.

Question 3 Proposer un algorithme glouton pour le problème des k-centres.

Question 4 On note $p_1, ..., p_k$ les k points retrourné par notre algorithme et $p_1^*, ..., p_k^*$ la solution optimale. On partitionne $V = \bigsqcup_{i \in \mathbb{N}} V_i$ avec V_i l'ensemble des points les plus proches de p_i^*. Montrer que

- Si $\forall i, V_i \cap \{p_1, ..., p_n\} \neq \emptyset$ alors on a une 2-approximation
- Sinon, on a aussi une 2-approximation

Algorithme ID3 avec intervalles

On cherche ici à effectuer l'algorithme ID3 pour des attributs non discret (un intervalle des réels par exemple). Pour cela, on considère des attributs $(A_i)_i$ tel que A_i est soit fini, soit $A_i = [\alpha_i, \beta_i]$, et un ensemble de classes C fini. Pour $S \subseteq A_1 \times .... \times A_k \times C$ un ensemble fini d'apprentisage, et pour A_i de la forme $A_i = [\alpha, \beta]$, pour $k \in]\alpha, \beta[$, on définit $S_{<k}, S_{\geq k}$ la coupure à k telle que $S_{<k} = \{\vec{s} \in S \mid \vec{s}_i < k\}$ et $S_{\geq k} = \{\vec{s} \in S \mid \vec{s}_i \geq k\}$

Question 1 On suppose que on a un attribut "taille (en cm)" qui est un naturel. Quel est le problème avec ID3 ?

Question 2 On cherche à trouver une "bonne coupure" : une qui maximise le gain d'information sur la coupure. Proposer une formule pour le gain d'information dans ce contexte similaire.

Question 3 Proposer un algorithme en $O(|S| \log |S| + |C| \times |S|)$ qui trouve le meilleur $k \in [\alpha; \beta]$ pour effectuer une coupure.

Question 4 Quelle est la complexité de l'algorithme ID3 avec notre généralisation en fonction de $|S|$, de $|C|$ et des $(A_i)_i$?

MPI: Système, Mutex & Sémaphore

Cours

- Fil d'exécution, non déterminisme, opération atomique.
- Mutex, Sémaphore.
- L'algorithme de Peterson
- L'algorithme de la boulangerie de Lamport
- Problème du rendez-vous, producteur-consommateur, du diner des philosophes.

Lecteurs-Rédacteurs

On se place dans le cadre d'une base de donné avec 2 types d'utilisateurs: les lecteurs et les rédacteurs. La base de donné a ces deux contraintes:

- Plusieurs lecteurs doivent pouvoir lire la base de données en même temps ;
- Si un rédacteur accède à la base de donné alors personne d'autres ne peut l'utiliser.

On cherchera dans toutes les questions à donner le code des fonctions:

- `void demander_lecture()` et `void demander_redaction()` quand un fil demande accès à la base
- `void terminer_lecture()` et `void terminer_redaction()` quand un fil à terminer d'utiliser la base

Question 1 Proposer une solution avec une variable `nb_lect` protégé par un sémaphore initialisé à 1 et un deuxième sémaphore de sorte à ce sorte que le rédacteur soit mis en attente tant qu'il y a encore des lecteurs. Aurait-on pu utiliser 2 mutex ?

Question 2 Montrer qu'il y une pottentielle famine des rédacteurs avec cette solution.

Question 3 Proposer une solution avec un compteur protégé par un mutex, et deux sémaphores qui résoud ce problème de famine.

Votes en parralèlles

On se proposer d'implémenter un système de vote entre plusieurs fils. Chacun des `k` fils d'exécution executera la fonction `void vote(int valeur)` que l'on se propose d'implémenter. Une fois que tous les fils ont voté, **tous les fils** devront executer la fonction `resultat(int result)` avec `result` la valeur la plus voté par les fils (en cas d'égalité, prendre la valeur la plus petite dans tous les gagnant).

Par exemple, pour $k = 5$, si les 5 fils d'exécution executent `vote(2)`, `vote(4)`, `vote(3)`, `vote(4)`, `vote(3)` en parralèlle, alors on doit s'attendre à ce que `resultat(3)` soit appelé par les 5 fils.

Question 1 Dans le cas ou les votes sont soit 0 soit 1, proposer une implémentation de `void vote(int valeur);` à base d'un mutex, un sémaphore initialisé à 0 et de deux variables entières partagés.

Question 2 On considère le cas ou les votes sont dans $[\![0; p]\!]$. Proposer une solution avec un tableau de mutex de longueur p, un compteur protégé par un mutex et un sémaphore initialisé à 0.

Question 3 Proposer une solution avec un tableau de longueur k, le nombre de fils, n'utilisant qu'un compteur protégé par un mutex et un sémaphore.

Question 4 En s'inspirant de l'algorithme de la boulangerie de Lamport, proposer un algorithme utilisant un tableaux d'entier partagé, mais aucun mutex nrei sémaphore. On suppose le tableau initialisé à 0.

Salle de spectacle

On considère une salle de spectacle qui peut être utilisée par deux groupes de musiciens: les violonistes et les guitaristes. Chaque musicien se prépare au spectacle en un temps aléatoire. Dès que tous les musiciens d'un groupe sont prêts, alors ils devront tous jouer ensemble (tous les violonistes ou bien tous les guitaristes jouent). Une fois leur performance faite, ils retournent en préparation.

On suppose qu'il y a N violonistes et M guitariste.

1. Proposer une modélisation de ce problème où chaque musicien est représenté par un fil d'exécution exécutant soit la fonction `violon` soit la fonction `guitare`, sans ce soucier des problèmes de synchronisation. On représentera le fait que tous les musiciens d'un groupe partent jouer par un **unique** appel à une fonction `void spectacle(bool est_violon);`
   On écrit les fonctions `violon` et `guitare`.
2. Quels peuvent être les problèmes rencontrés ?
3. Proposer une solution utilisant 2 sémaphores et 3 mutexs.
4. Et s'il y avait k groupes de musiciens ?

Sémaphore pondéré

On cherche à implémenter une structure de données qui est une généralisation pondérée des sémaphores. Elle aura trois opérations:

- `init(int capacite)` qui initiallisé et crée notre sémaphore pondéré avec `capacite` ressources
- `prendre(int k)` qui bloque le fil jusqu'à ce que l'on puisse alloué un total de k ressources
- `vendre(int k)` qui libère k ressources

Question 1 On propose la suivante première implémentation de notre sémaphore pondéré à l'aide d'un sémaphore normal `sem` :

```
void init(int k) {init(sem,k);}

void prendre(int k) {
  for (int i = 0;i<k;i++) {P(sem);};
}

void vendre(int k) {
  for (int i = 0;i<k;i++) {V(sem);};
}
```

Expliquer pourquoi cette implémentation peut donner à des situations d'interblocages

Question 2 Proposer une implémentation sans interblocage utilisant un compteur protégé par un mutex. Cette solution peut-elle être source de famine ?

Question 3 Proposer une solution garentissant l'abscence de famine utilisant un sémaphore, et une variable protégé par un mutex.

Question 4 Montrer l'abscence d'interblocage et de famine.

Mutex ré-entrant

On cherche à implémenter un mutex réentrant: c'est un mutex sauf que si la ressource est allouée à un fil k, alors le fils k peut se lock autant de fois qu'il veux. Notamment, cela permet d'executer ce code sans situation de blocage:

```
void exemple() {
  lock(mutex_reentrant);
  lock(mutex_reentrant);
  unlock(mutex_reentrant);
}
```

Question 1 Proposer une implémentation d'un tel mutex réentrant en utilisant seulement un mutex classique, un entier partagé et de l'attente active.

On cherche maintenant à faire la même chose pour des sémaphore: si un fil d'exécution execute `p(semaphore)` et obtiens la ressource, alors il peut appeller `p(semaphore)` autant de fois qu'il veux sans être mis en attente ni faire décroire le compteur interne, et ce jusqu'à ce que le fil appelle `v(semaphore)`.

Question 2 Proposer une fonction qui, si elle est executé par deux fils, peut donner lieu a un interblocage des deux fils avec un sémaphore classique, mais qui ne donne pas lieu à un interblocage avec un sémaphore réentrant.

Question 3 Proposer une implémentation d'un tel sémaphore réentrant avec un tableaux de booléens et un sémaphore classique.

Pont à voie unique

On cherche ici a modéliser un pont à voie unique qui ne peut être que emprunter par des voitures allant dans le même sens. On représentera cela par deux types de fils d'exécution: les fils ascendant et les fils descendant. Chaque fil répète en boucle l'exécution de `request_bridge(b);` puis de `exit_bridge(b);` avec `b` un booléen valant `true` iff le fil est ascendant. On veux s'assurer que pour chaque fil, lors de sa présence dans la section critique (entre `request_bridge(b);` et `(b);`), il n'y a que des fils allant dans la même direction.

Question 1 Proposer une solution utilisant un mutex, un compteur partagé et de l'attente active. Expliquer pourquoi il peut avoir une famine.

Question 2 Donner une solution ayant une abscence de famine avec 2 sémaphores initialisé à 1 et un entier. On pourra s'inspirer de la solution avec de l'attente active.

Question 3 On suppose maintenant qu'un maximum de 5 voitures peuvent être présentent en même temps sur le pont. Modifier le code pour correspondre à ce critère.

Question 4 Et s'il le pont possèdait k voies, tel qu'une seule ne peut être empruntée en même temps ?

MPI: Proba & Approx

Cours

- Algorithme déterministe, de Las Vegas, Monte Carlos.
- Problème de décision, approximation, instance positive, instance négative.
- Fonction de cout, algorithme d'approximation.

Question de Cours

- Qu'est-ce qu'un algorithme de Monte-Carlo et de Las vegas ?
- Comment transformer un problème d'approximation en un problème de décision ?

Petites Questions

- Soit P un problème de minimisation et $\overline{P}$ le problème de décision associé. On suppose que $\overline{P}$
- Quelle est la probabilité que la fonction suivante termine ? `random()` renvoie un flottant aléatoire de manière uniforme dans $[0; 1[$

```
void f() {
  if (random() < 0.5f) {f();f();}
  return;
}
```

A ajouter

- Polya's Urn, Exchangeable random variables
- Chinese restaurant process
- Indian bufffet process
- Moris approximate counting algorithm https://en.wikipedia.org/wiki/Approximate_counting_algorithm (count to n with $\log \log n$ bits)

Espérence pour tout choisir

On choisi uniformément des entiers dans $\{1, \ldots, n\}$ avec remis j'usqu'à ce que tous les entiers ont été choisi. On note X la variable aléatoire du temps d'exécution. Montrer que $\mathbb{E}[X] = n \ln n + o(n \ln n)$

Le programme de Von Neumann

Soit `bool random(float p);` une fonction qui génère le booléen `true` avec probabilité p et le booléen `false` avec probabilité $1 - p$.

1. Soit $0 < p < 1$. Montrer que le code suivant renvoie `true` avec probabilité 1/2 :

```
bool von_neumann(float p) {
  bool a = random(p);
  bool b = random(p);
  if (a!= b) { return a; }
  if (a == b) { return von_neumann(p);};
};
```

2. Que veux l'espérence du temps d'exécution en fonction de p ?

2-approximation du circuit eulérien (TSP)

Soit $G = (V, E)$ un graphe non orienté pondéré par $w : E \to \mathbb{R}_+^*$. On note pour $c := (c_i)_{i \leq n}$ un chemin $w(c) = \sum_{i<n} w((c_i, c_{i+1}))$ son poids. Similairement, on définit $w(T)$ le poids d'un arbre T sur G comme la somme du poids de toutes ses arêtes. Un circuit est un cycle où l'on a le droit de repasser sur des sommets déjà vu.

On cherche un circuit passant par tous les sommets au moins une fois et pouvant repasser sur lui-même tel que le poids de tout le chemin soit minimal. On note w^* le poids du circuit de poids minimal.

Question 1 On suppose que tous les aretes du graphe sont de poids différents. Montrer que le plus petit arbre couvrant est unique.

Question 2 Montrer que si T est arbre couvrant de poids minimal, alors $w(T) < w^*$

Question 3 En déduire via un parcours de cet arbre un circuit repassant sur lui-même c de poids $w(c) < 2w^*$

Question 4 Proposer un algorithme pour calculer un tel arbre. Quel est sa complexité ?

$3/2$-approx de TSP

On considère le problème du voyageur de commerce (TSP) :

- **Entrée** Un graphe $G = (V, E)$ avec $d : V^2 \longrightarrow \mathbb{R}_+$ une distance (respecte les axiomes de la distance).
- **Sortie** Le poids minimum d'un cycle passant par tous les sommets.

On note $w(C)$ pour C un chemin (ou arbre) son poids. On pose T^* un arbre couvrant de G' de poids minimal et I l'ensemble des sommets impair de T^*. On note w^{OPT} le poids optimal d'une instance du problème.

Question 1 Montrer que $w(T^*) < w^{\mathrm{OPT}}$. En déduire une 2-approximation du problème en utilisant un parcours de T^*

Question 2 Montrer que $|I|$ est pair et en déduire que G' restraint à I possède un couplage de poids minimal que l'on notera M.

Question 3 Montrer que si G est un graphe connexe tel que $\forall v \in V, \deg v \in 2\mathbb{N}$, alors G possède un cycle eulérien (passant par toutes les aretes). En déduire que le multigraphe $T^* + M$ possède un cycle eulérien que l'on notera h.

Question 4 Montrer que le problème du voyageur de commerce admet une $\frac{3}{2}$-approximation.

C-approx du couplages maximum[122]

Pour M un couplage, on note $S(M)$ les sommets saturé de M.

On considère l'algorithme suivant:

Entrée: G un graphe
$M \leftarrow$ un couplage maximal de G
tant que $\exists \{u', u\}, \{u, v\}, \{v, v'\} \in E$ avec $\{u, v\} \in M$ et $u', v' \notin S(M)$:
    $M \longleftarrow (M \setminus \{u, v\{\}\}) \cup \{\{u', u\}, \{v, v'\}\}$
fin tant que
retourner M

[122] TD11 L3 ENS Lyon Algo 1

1. Quelle est la complexité de l'algorithme ?
2. Montrer que l'algorithme est une C-approx du couplage maximum pour un certain C que l'on déterminera.

On change la boucle pour chercher un chemin alternant P de longueur $2t+1$, et si on en trouve un on effectue $M \longleftarrow (M \setminus P) \cup (P \setminus M)$. L'algorithme donné correspond donc au cas $t = 1$.

3. Donner le facteur d'approximation en fonction de t, et en déduire un *PTAS*, c'est-à-dire que pour tout $\varepsilon > 0$ on a un algorithme polynomial qui renvoie une $(1-\varepsilon)$-approximation. On déterminera exactement la complexité en fonction de ε.

Vertex cover

Soit $G = (V, E)$ un graphe. On dit que $S \subseteq V$ est *une couverture* si pour toute arête $e \in E$, au moins une des extrémitées de e est dans S. On s'intéresse au problème VERTEX-COVER suivant:

- **Entrée:** $G = (V, E)$ un graphe
- **Sortie:** Une couverture C de taille minimale en cadinal

1. Donner la version problème de décision de VERTEX-COVER.

On considère un algorithme glouton qui, tant qu'il reste des arêtes au graphe, en choisit une et retire les deux sommets du graphe. On note C les arêtes choisies la fin.

2. Montrer que si S est une couverture, alors $|S| \leq 2|C|$
3. En déduire une 2 approximation de VERTEX-COVER

On considère l'algorithme qui, tant que le graphe possède encore des arêtes, prend un sommet de degré non nul au hasard uniformément, le choisi et retire toutes les arêtes relié à ce sommet.

4. Quel est ce type d'algorithme ? Soit $G = (V, E)$ un graphe, on note V_f la variable aléatoire de l'ensemble choisi à la fin de l'exécution de l'algorithme sur G. Montrer que

$$\mathbb{E}\left[|V_f|\right] = \sum_{v \in V} \frac{\deg v}{\deg v + 1}$$

Set cover

On considère le problème SET-COVER suivant:

Entrée: $n, m \in \mathbb{N}, U_1, ..., U_n \subseteq [\![1; m]\!]$
Sortie: Le plus petit I en cardinal tel que $\bigcup_{i \in I} U_i = [\![1; m]\!]$

Dans toute cette colle on supposera que toutes les entrées sont telles que $\bigcup_{i \leq n} U_i = [\![1; m]\!]$

Question 1 Résoudre le problème pour les instances de la forme $n \in \mathbb{N}; m := n+1; U_i = \{i, i+1\}$

Question 2 Donner une $\log(m)$-approximation de SET-COVER. *Ind: On fera un algorithme glouton et on pourra utiliser le fait que* $\frac{1}{2} + ... + \frac{1}{n} \leq \log n$

On considère maintenant que la sortie doit être un I qui maximise le nombre de $x \in [\![1; m]\!]$ tel que x appartienne à un nombre impair d'ensembles de $(U_i)_{i \in I}$

Question 3 Soit $(U_i)_{i \leq n}$ une instance du problème. On choisi avec probabilité $\frac{1}{2}$ chaque U_i. On note N la variable aléatoire du nombre de $x \in [\![1; m]\!]$ appartenant à un nombre impair d'ensembles choisi.

Question 4 Montrer que $\mathbb{E}[N] = \frac{m}{2}$

Question 5 En déduire un algorithme un algorithme de Las Vegas qui est une 2-approximation. Existe t'il une 2-approximation déterministe ?

2-approx de Bin-Packing[123]

On s'intéresse au problème `BIN-PACKING` suivant :

- **Entrée:** Des objets de tailles $t_1, ..., t_n \in]0, 1]$
- **Sortie** Le plus petit nombre de boites $n \in \mathbb{N}$ de capacité 1 nécéssaire pour ranger tout les objets.

On note OPT le nombre minimal de boîtes nécessaires.

On étudie l'algorithme `Next-Fit` suivant :

$N, T \longleftarrow 0$
Pour i allant de 1 à n:
    Si $T + t_i \leq 1$ **alors:**
        $T \longleftarrow T + t_i$
    Sinon:
        $T \longleftarrow t_i$
        $N \longleftarrow N + 1$
Retourner N

1. Appliquer `Next-Fit` à la suite $(0.4, 0.7, 0.3, 0.6, 0.5, 0.5)$. Que vaux N à la fin?
2. Montrer que si `Next-Fit` ouvre une nouvelle boîte, alors la boîte précédente et la nouvelle boîte ont ensemble une charge strictement supérieure à 1.
3. On suppose que `Next-Fit` utilise k boites. Montrer que :

$$\text{OPT} \geq \sum_{i=1}^{n} t_i > \frac{k-1}{2}.$$

4. En déduire que `Next-Fit` est une 2-approximation.
5. Est-ce que `Next-Fit` est meilleur qu'une 2-approximation?

$\log(n)$-approx de MLST

Un *multi-graphe* est un graphe dans lequel on peut avoir plusieurs fois une arête entre 2 sommets. On considère le problème MIN-COLOR-TREE suivant:

- **Entrée:** $G = (S, A)$ un multigraphe et $c : A \longrightarrow \mathbb{N}$ une coloration des arrettes.
- **Sortie:** Un arbre couvrant $T = (S, A')$ de G qui minimise $\text{Card}(\{c(e) : e \in A'\})$

On note n le nombre de sommets et m le nombre d'arêtes.

1. Donner la version problème de décision, et montrer qu'elle est NP.
2. Montrer que si toutes les arêtes sont de couleur différentes alors il existe un algorithme polynomial.

On considère l'algorithme glouton suivant. Il initialise une structure union-find avec une classe par sommet et, a chaque étape, on choisit parmi les couleurs non encore choisies une couleur qui minimise le nombre de composantes après ajout de toutes les arêtes de cette couleur. Une fois que toutes les classes sont regroupées, alors il retourne un arbre couvrant des arêtes des couleurs choisies.

5. Montrer que l'algorithme renvoie bien une solution.

[123] CCINP 2025

On note q le nombre de couleurs de la solution optimale et p_i le nombre de classes de la structure union-find à l'étape i de l'algorithme.

6. Montrer que

$$p_{i+1} - 1 \leq \left(1 - \frac{1}{q}\right)(p_i - 1)$$

7. En déduire une $\log(n)$-approximation.

Arbres ternaires complets (+Jeux)[124]

Un arbre T est dit ternaire complet si chaque noeud interne possède exactement 3 fils et que les feuilles sont toutes à la meme profondeur. On note $\mathcal{F}(T)$ les feuilles de T et $\mathcal{N}(T)$ les noeuds de T (incluant les feuilles).

On dispose d'une valuation $\sigma : \mathcal{F}(T) \longrightarrow \{V, F\}$ qui associe à chaque feuille une valeur booléene. On étend σ à $\mathcal{N}(T)$ en posant $\sigma(u) = b$ pour tout $u \in \mathcal{N}(T)$ ayant deux fils u_1, u_2 tels que $\sigma(u_1) = \sigma(u_2) = b$. On cherche à évaluer la valeur de σ à la racine.

1. Exprimer $|\mathcal{F}(T)|$ et $|\mathcal{N}(T)|$ en fonction de la hauteur de l'arbre T.

On considère l'algorithme probabiliste récursif qui consiste à tirer aléatoirement deux enfants u_1 et u_2, et n'évaluer σ sur u_3 l'enfant non choisi seulement si $\sigma(u_1) \neq \sigma(u_2)$.

2. Quel type d'algorithme s'agit t'il? Montrer que l'éspérence du nombre de feuilles que l'algorithme visite et inférieure à $\left(\frac{8}{3}\right)^h$ pour h la hauteur de T.
3. En déduire un algorithme probabiliste qui, en espérance, est de complexité $O(|\mathcal{N}(T)|^{0.9})$ (donc sous-linéaire). *On indique que* $\log_3(8) \approx 1,893$

On cherche maintenant à montrer qu'un algorithme déterministe correct doit forcément inspecter toutes les feuilles. On pose $n = |\mathcal{F}(T)|$. Soit $F \subseteq \mathcal{F}(T)$, on dit que F *détermine* un noeud $u \in \mathcal{N}(T)$ soit si $u \in F$ ou s'il a deux fils u_1 et u_2 déterminés par F tels que $\sigma(u_1) = \sigma(u_2)$

4. Montrer que pour tout $u \in \mathcal{N}(T)$ non déterminé par F, on peut modifier les valeurs de σ sur $\mathcal{F} \setminus F$ de manière à changer la valeur de $\sigma(u)$.
5. On considère le jeu à deux joueurs suivant : à chacun des $n-1$ tours, Alice choisi une feuille $x \in \mathcal{F}(T)$ non déjà choisit et Bob décide si $\sigma(x) = V$ ou $\sigma(x) = F$. Alice gagne si l'ensemble des $n-1$ feuilles détermine la racine, et Bob gagne dans le cas contraire. Montrer que Bob dispose d'une stratégie gagnante.
6. En déduire qu'un algorithme déterministe ne peut pas faire mieux que du $O(|\mathcal{N}(T)|)$.

Unique graphe infini aléatoire

On dit que deux graphe $G = (V, E)$ et $G' = (V', E')$ sont isomorphe s'il existe $\varphi : V \longrightarrow V'$ bijective tel que $(u, v) \in E \Leftrightarrow (\varphi(u), \varphi(v)) \in E'$

On cherche à montré que les graphes aléatoire infinis dénombrable sont isomorphes avec probabilité 1. On considère pour cela les variables aléatoire de bernoulli $\left(X_{u,v}\right)_{u,v \in \mathbb{N}}$ indépendantes de probabilité p qui représente s'il existe une arête entre u et v. On note G_p cette famille de variables aléatoire.

Question 1 Quelle est la probabilité que le graphe possède le chemin $(0, 1, 2, ..., k)$?

[124] Exercice 100% recopié de Florian Bourse : https://www.di.ens.fr/~fbourse/enseignement/Proba.pdf

On dit qu'un graphe infini possède *la propriété de Rado* si pour tout $U, V \subseteq \mathbb{N}$ disjoint fini, il existe $x \in \mathbb{N}$ tel que x soit connecté à tout U et a aucun V.

Question 2 Soit $U, V \subseteq \mathbb{N}$ fini, montrer que la probabilité qu'un $x \in \mathbb{N} \setminus (U \cup V)$ satisfaite la propriété de rado est non nul. En déduire l'existence d'un tel x avec probabilité 1.

Question 3 Montrer que si G et G' possède la propriété de Rado, alors ils sont isomorphes.

Question 4 Généraliser la question 3 pour montrer que la probabilité que G_p soit isomorphe a $G_{p'}$ est 1 si $0 < p, p' < 1$.

$(\Delta + 1)$-approx de Maximum independant set

Question de Cours Rapeller la définition d'un algorithme de monte-carlo et d'un algorithme de las-vegas.

Question 1 Montrer que si L est un langage hors-contexte alors $\overline{L} = \{\overline{w} : w \in L\}$ (l'ensemble des mirroirs de L) l'est aussi.

On considère le problème MIS suivant:

Entrée: Un graphe $G = (V, E)$ non orienté **Sortie:** Un $I \subseteq V$ de cardinal maximal tel que $E \cap I^2 = \emptyset$

Pour $G = (V, E)$ un graphe, on note $\Delta(G) = \max_{v \in V} \deg(v)$.

Question 2 Donner la version problème de décision de MIS

Question 3 Donner un $(\Delta(G) + 1)$-approximation de MIS qui tourne en $O(|S|^2)$

On considère l'algorithme qui, tant que le graphe possède encore des arêtes, prend un sommet de degré non nul au hasard uniformément, le choisi et retire toutes les arêtes relié à ce sommet.

Question 4 Quel est ce type d'algorithme ? Soit $G = (V, E)$ un graphe, on note V_f la variable aléatoire de l'ensemble choisi à la fin de l'exécution de l'algorithme sur G. Montrer que

$$\mathbb{E}\big[|V_f|\big] = \sum_{v \in V} \frac{\deg v}{\deg v + 1}$$

Coloration aléatoire

Question de Cours Montrer que les langages réguliers sont hors-contexte

Soit $k \in \mathbb{N}$ fixé. On regarde le problème du k-PATH:

Entrée: Un graphe $G = (V, E)$ non orienté **Sortie:** Est-ce qu'il existe un chemin de longueur k ?

On définit une k-coloration comme une fonction $\mu : V \longrightarrow \{1, .., k\}$. On dit qu'un chemin est arc-en-ciel si tous les sommets sont de couleur différente.

Question 1 Quelle est la probabilité qu'un chemin de longueur k soit arc-en-ciel ?

Question 2 Donner un algorithme de programmation dynamique qui, soit G et une k-coloration, vérifie si G possède un chemin arc-en-ciel en $O\big(2^k \times |E|\big)$.

Question 3 En déduire un algorithme à erreur unilateralle de même probabilité que celle obtenu question 1. Quel type d'algorithme est-ce ?

Question 4 En déduire un algorithme à erreur unilateralle de probabilité $\frac{1}{2}$. Quel est son temps d'exécution ?

K-Centres

On fixe $V \subseteq \mathbb{R}^d$ un jeu de donnée. Pour $\vec{y} \in \mathbb{R}^d$ on définit $d(\vec{y}, V) = \max_{\vec{x} \in V} \|\vec{y} - \vec{x}\|$. On cherche à choisir k points de V tel qu'ils minimise $\max_{v \in V} d(v, X)$. On regarde donc le k-CENTRES suivant:

- **Entrée:** $v_1, ..., v_n \subseteq \mathbb{R}^d$ une liste de points finis
- **Sortie** : un $X \coloneqq \{x_1, ..., x_k\} \subseteq V$ qui minimise $\max_{v \in V} d(v, X)$

Question 1 On cherche à montrer que l'algorithme du cours de la question ne donne pas de bonne approximation du problème des k-centres. Pour cela on pose $k = 2$, $d = 1$ et on pose N points en 0, N points en 1 et un point en D. Montrer que pour des bonnes valeurs de N et D on peut avoir le ratio $\frac{\text{opt-k-moyennes}}{\text{opt-k-centre}}$ arbitrairement large.

Question 2 Proposer un algorithme glouton pour le problème des k-centres.

Question 3 On note $p_1, ..., p_k$ les k points retrourné par notre algorithme et $p_1^*, ..., p_k^*$ la solution optimale. On partitionne $V = \bigsqcup_{i \in \mathbb{N}} V_i$ avec V_i l'ensemble des points les plus proches de p_i^*. Montrer que

- Si $\forall i, V_i \cap \{p_1, ..., p_n\} \neq \emptyset$ alors on a une 2-approximation
- Sinon, on a aussi une 2-approximation

Paire de points

Pour tout $x, y \in \mathbb{R}$ deux points distincts inconnus, on te donne avec probabilité $\frac{1}{2}$ le point x et sinon le point y. Tu peux décider de garder ou d'échanger de points avec une certaine probabilité qui dépend seulement du point que l'on te donne. Avec quelle probabilité dois-tu échanger de manière à ce que quel que soit la paire $(x, y) \in \mathbb{R}^2$ tu prend le plus grand point avec probabilité strictement plus grande que $\frac{1}{2}$?

MPI: Déduction naturelle

Cours

- Arbre d'inférence, contexte Γ, séquent prouvable, variables libres, liées.
- Règle d'introduction et d'élimination de chaque opérateur $\wedge, \vee, \neg, \to, \forall, \exists$
- Syllogisme de barbara, modus ponens.
- Etre capable de montrer la correction des règles
- Logique minimale, intuitionniste (+ $\bot_e$), classique (HP)

Exos à ajouter :

- LTS and bisumulation: formula with $a.\varphi, \top, \wedge$ characterise bisumulation Henessy-Milner Modal Logic, image finite iff ....

Arbres à faire

Montrer les séquent suivant en logique minimale:

- $(p \wedge (p \to q)) \vdash q$
- $A \wedge B \to C \vdash A \to B \to C$ (la currification)
- $A \to B \to C \vdash A \wedge B \to C$ (la co-currification)
- $p \to q, q \to r \vdash p \to r$ (le syllogisme de barbara)
- $\vdash p \wedge q \to q \wedge p$ (la commutativité du $\wedge$)
- $\vdash p \vee q \to q \vee p$ (la commutatitivté du $\vee$)
- $\vdash A \to B \vdash \neg B \to \neg A$ (le raisonnement par contraposé)

- $\vdash \neg\neg\neg A \to \neg A$ (même si $\neg\neg A \to A$ n'est pas démontrable, ce séquent l'est !)
- $p \to q \vdash \neg(p \wedge \neg q)$ (l'équivalence de l'implication)

Arbres des lois de morgan

Montre les séquents suivant en logique minimale:
- $\neg(A \vee B) \vdash \neg A \wedge \neg B$
- $\neg A \wedge \neg B \vdash \neg(A \vee B)$
- $\neg A \vee \neg B \vdash \neg(A \wedge B)$

Montrer le séquent $\neg(A \wedge B) \vdash \neg A \vee \neg B$ en logique classique

Arbre des lois de morgan avec quantificateurs

Montrer les séquents suivants en logique intuitioniste:
- $\forall x, \neg A \vdash \neg\exists x, A$ où $x \in \mathrm{VL}(A)$
- $\exists x, \neg A \vdash \neg\forall x, A$ où $x \in \mathrm{VL}(A)$
- $\neg\exists x, A \vdash \forall x, A$ où $x \in \mathrm{VL}(A)$

Montrer en logique classique (ce séquent n'est pas démontrable en logique intuisioniste !) :
- $\neg\forall x, A \vdash \exists x, A$ où $x \in \mathrm{VL}(A)$

Equivalence entre les formules donnant la logique classique

Montrer que en logique intuitioniste, nous avons les jugements suivants:
- $A \vee \neg A \vdash ((A \to B) \to A) \to A$ (Tiers-exclu vers Peirce)
- $((A \to B) \to A) \to A \vdash A \vee \neg A$ (Peirce vers Tiers-exclu)
- $A \vee \neg A \vdash \neg\neg A \to A$ (Tiers-exclu vers élimination du double non)
- $\neg\neg A \to A \vdash A \vee \neg A$ (élimination du double non vers Tiers-exclu)

Arbres nécésitant de deviner

On définit $a \leftrightarrow b$ par $a \to b \wedge b \to a$.

Montrer que les séquents suivant sont prouvables en utilisant la logique intuitioniste:
- $\vdash \neg\neg(A \vee \neg A)$
- $\vdash \neg(A \leftrightarrow \neg A)$

Affaiblissement

Montrer que si le séquent $\Gamma \vdash A$ est démontrable alors $\Gamma, \Delta \vdash A$ l'est aussi.

XOR

On rajoute au langage des formules de logique propositionnelles l'opérateur $\oplus$ (prononcé XOR) d'arité 2, tel que pour η une valuation, on a $\eta \vDash \varphi \oplus \psi$ si et seulement si $\eta \vDash \varphi$ et $\eta \nvDash \psi$ ou bien $\eta \nvDash \varphi$ et $\eta \vDash \psi$.

1. Proposer 2 régles d'introduction et 1 règle d'élimination pour l'opérateur $\oplus$.
2. Montrer la correction de la règle, et utiliser la pour montrer la commutativité du $\oplus$.

Equivalence

On rajoute au langage des formules de logique propositionnelles l'opérateur d'équivalence $\leftrightarrow$ d'arité 2 avec les règles suivantes:

$$\frac{\Gamma, A \vdash B \quad \Gamma, B \vdash A}{\Gamma \vdash A \leftrightarrow B} \leftrightarrow_i \qquad \frac{\Gamma \vdash A \leftrightarrow B}{\Gamma \vdash A \to B} \leftrightarrow_l \qquad \frac{\Gamma \vdash A \leftrightarrow B}{\Gamma \vdash B \to A} \leftrightarrow_r$$

Soit μ une valuation, proposer une définition de $\mu \vDash A \leftrightarrow B$. Montrer la correction de la règle, et utiliser la pour montrer la commutativité de l'équivalence.

L'opérateur de Sheffer

On rajoute au langage des formules de logique propositionnelles *l'opérateur de Sheffer* $A \uparrow B$ tel que $A \uparrow B \equiv \neg(A \wedge B)$.

1. Montrer que $A \to B \equiv A \uparrow (B \uparrow B)$
2. Proposer une règle d'introduction et une règle d'élimination pour l'opérateur de Sheffer et montrer qu'elles sont correcte.
3. Montrer que pour chaque formule F il existe une formule F^* équivalente n'utilisant que l'opérateur de sheffer et $\top$ ou $\bot$.
4. Montrer que si $\vdash F$ est prouvable alors $\vdash F^*$ l'est aussi.

Sans implication

Soit F une formule propositionnelle, on définit par induction $\psi(F)$ la *transformé sans implications de* F par

$$\begin{aligned} \psi(A \vee B) &:= \psi(A) \vee \psi(B) & \psi(X) &:= X \quad \text{pour } X \text{ une variable} \\ \psi(A \wedge B) &:= \psi(A) \wedge \psi(B) & \psi(\neg A) &:= \neg\psi(A) \\ \psi(A \to B) &:= \neg\psi(A) \vee \psi(B)) & \psi(\top) &:= \top, \psi(\bot) := \bot \end{aligned}$$

Montrer que $\vdash F$ est prouvable si et seulement si $\vdash \psi(F)$ est prouvable

Complétude[125]

On cherche à montrer que si F une formule propositionnelle est vraie pour tout modèle, alors $\vdash F$ est démontrable en logique classique.

Pour μ une valuation et φ une formule on pose

$$|\varphi|_\mu = \begin{cases} \varphi & \text{si } \mu \vDash \varphi \\ \neg\varphi & \text{sinon} \end{cases}$$

1. Soit F une formule et $\{X_1, ..., X_n\}$ les variables propositionnelles de F. Montrer que le séquent suivant est démontrable en logique classique pour toute valuation μ :
$$|X_1|_\mu, ..., |X_n|_\mu \vdash |A|_\mu$$
2. Montrer que en logique classique, si $\Gamma, X \vdash \varphi$ et $\Gamma, \neg X \vdash \varphi$ sont prouvable, alors $\Gamma \vdash \varphi$ l'est aussi.
3. Montrer que si A est une tautologie, alors pour toute valuation μ, on a $|X_1|_\mu, ..., |X_n|_\mu \vdash A$ démontrable.
4. Conclure.

Sans négation

Soit F une formule propositionnelle, on définit par induction $\psi(F)$ la *transformé sans négation de* F par

$$\begin{aligned} \psi(A \vee B) &= \psi(A) \vee \psi(B) & \psi(X) &= X \quad \text{pour } X \text{ une variable} \\ \psi(A \wedge B) &= \psi(A) \wedge \psi(B) & \psi(\neg A) &= \psi(A) \to \bot \\ \psi(A \to B) &= \psi(A) \to \psi(B) & \psi(\top) &= \top, \psi(\bot) = \bot \end{aligned}$$

[125] Le magnifique livre de Mme. Galatée Hemery ! ! ! ! <3

Montrer que $\vdash F$ est prouvable si et seulement si $\vdash \psi(F)$ est prouvable

Théorie déductive des graphes

On se donne le langage de la logique du premier ordre auquel on a rajouté une relation R d'arité 2. En plus des règles concernant la déduction naturelle, on rajoute les règle suivantes, indiquant que R est symétrique et n'est pas reflexif:

$$\frac{\Gamma \vdash R(x,y)}{\Gamma \vdash R(y,x)}\text{sym} \qquad\qquad \frac{}{\Gamma \vdash \neg R(x,x)}\ \text{nr}$$

On peut interpréter alors une formule F de cette théorie comme une formule sur des graphes non orienté sans boucle: soit $G = (V, E)$ un graphe, on a $(x, y) \in E$ si et seulement si $R(x, y)$. Par exemple, la formule "$\forall x, R(s, x)$" est la formule qui indique que s est connecté à tous les sommets.

1. Montrer que le séquent $\vdash \neg(\forall x, \forall y, R(x, y))$ est démontrable.
2. Montrer à l'aide d'un arbre de preuve que dans un graphe, si un sommet est relié à tous les autres, alors chaque sommet admet au moins un voisin.

Formule Duale

Pour F une formule de la logique propositionnelle, on définit par induction $F^\perp$ la *formule duale de* F par:

$$(A \vee B)^\perp := A^\perp \wedge B^\perp \qquad X^\perp := \neg X \quad \text{pour } X \text{ une variable}$$
$$(A \wedge B)^\perp := A^\perp \vee B^\perp \qquad (\neg A)^\perp := \neg A^\perp$$
$$(A \to B)^\perp := \neg A^\perp \vee B^\perp$$

1. Montrer que si μ est une valuation, alors $\mu \vDash \neg F$ si et seulement si $\mu \vDash F^\perp$
2. Montrer par induction sur les formules F de la logique propositionnelles que pour tout Γ, si $\Gamma \vdash \neg F$ alors il existe une preuve que $\Gamma \vdash F^\perp$ (en logique classique). On appellera cela la *loi de morgan généralisé.*

Logique pertinente[126]

La logique pertinente est une logique déductive telle que si $\Gamma \vdash F$ est prouvable, alors chaque hypothèse de Γ doit etre utillisé au moins une fois dans la preuve de F. Ainsi, $A \vdash A \wedge A$ sera prouvable mais $A, B \vdash A$ ne sera pas prouvable car B n'est pas utillisé.

On considère les formules propositionnelles construites à partir de variables propositionnelles, des connecteurs $\wedge$, $\vee$ et de $\to$. Les contextes Γ, Δ sont des ensembles finis de formules. On note Γ, A pour dénoter le contexte $\Gamma \cup \{A\}$.

On définit la relation de prouvabilité $\Gamma \vdash F$ par les règles de déduction naturelle suivantes :

$$\frac{}{A \vdash A}\ \text{Ax} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\to_i \qquad \frac{\Gamma \vdash A \to B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B}\to_e$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\vee_i^g \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\vee_i^d \qquad \frac{\Gamma \vdash A \vee B \quad \Delta, A \vdash C \quad \Delta, B \vdash C}{\Gamma \cup \Delta \vdash C}\vee_e$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma \cup \Delta \vdash A \wedge B}\wedge_i \qquad \frac{\Gamma \vdash A \wedge B \quad \Delta, A, B \vdash C}{\Gamma \cup \Delta \vdash C}\wedge_e$$

1. Donner un arbre de preuve en logique pertinente pour les deux jugements suivants:

[126] Sujet custom!

$$\vdash A \to B \to (A \wedge B) \qquad \vdash ((A \vee B) \to C) \to (A \to C) \wedge (B \to C)$$

2. Montrer que si la règle $\wedge_e$ était remplacée par les deux règles suivantes, alors $\vdash A \to (B \to A)$ serai prouvable :

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_e^g \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_e^d$$

Pour F une formule, on définit $[\![F]\!]$ une collection d'ensembles de variables propositionelles de F par induction :

$$\begin{aligned}
[\![X]\!] &= \{\{X\}\} && \text{pour } X \text{ une variable} \\
[\![A \wedge B]\!] &= \{r \cup s \mid r \in [\![A]\!] \wedge s \in [\![B]\!]\} \\
[\![A \vee B]\!] &= \{r_1 \cup ... \cup r_k \mid k \geq 1 \wedge \forall i, r_i \in [\![A]\!] \cup [\![B]\!]\} \\
[\![A \to B]\!] &= \{r : \forall s \in [\![A]\!], r \cup s \in [\![B]\!]\}
\end{aligned}$$

Pour un contexte $\Gamma = \{A_1, ..., A_n\}$, on pose $[\![\Gamma]\!] = \{r_1 \cup ... \cup r_n \mid \forall i, r_i \in [\![A_i]\!]\}$. On posera $[\![\Gamma]\!] = \{\emptyset\}$ pour $\Gamma = \emptyset$ le contexte vide. On dit que le jugement $\Gamma \vdash A$ est *valide* si $[\![\Gamma]\!] \subseteq [\![A]\!]$

3. Donner les valeurs des ensembles suivants pour A, B deux variables propositionelles:

$$[\![A \wedge B]\!], \quad [\![A \vee B]\!], \quad [\![A \to B]\!].$$

4. Montrer que si $t_1, .., t_n \in [\![F]\!]$, alors $t_1 \cup ... \cup t_n \in [\![F]\!]$.
5. Montrer par induction que si $\Gamma \vdash F$ est prouvable alors $\Gamma \vdash F$ est valide.
6. En déduire que $A \to (B \to A)$ n'est pas prouvable en logique pertinente.

Introduction à la logique linéaire multiplicative

On définit par induction les formules de la logique linéaire par le fait que:
- X et $\overline{X}$ sont des formules pour X une variable propositionnelle
- $\varphi \otimes \psi, \varphi \parr \psi$ sont des formules de la logique linéiare pour φ, ψ deux formules de la logique linéaire

L'on note $\mathcal{L}$ l'ensemble des formules de la logique linéaire multiplicative. L'on muni ces formules des règles de déduction suivantes:

$$\frac{}{\vdash A, \overline{A}}\ \text{ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B} \parr$$

Attention, dans ces définition Γ n'est pas un ensemble mais une liste d'hypothèses (que l'ont peut réordonnée). La multiplicité du nombre d'occurence des formules compte, et la séparation Γ, Δ dans $\otimes$ doit être une partition des hypothèses.

1. Montrer que le séquent $\vdash (A \otimes A) \parr \left(\overline{A} \parr \overline{A}\right)$ est prouvable en logique linéaire multiplicative.

Soit $F \in \mathcal{L}$, on définit par induction le dual de F, noté $F^\perp$ par $(X \otimes Y)^\perp = X^\perp \parr Y^\perp$, $(X \parr Y)^\perp = X^\perp \otimes Y^\perp$, $X^\perp = \overline{X}$ et $\overline{X}^\perp = X$

2. Montrer que $\vdash F, F^\perp$ pour $F \in \mathcal{L}$.

3. $(*)$ Montrer que si $\vdash \Gamma, F$ et $\vdash \Delta, F^\perp$, alors $\vdash \Gamma, \Delta$

Contexte On dit qu'une règle déductive est *admissible* si en suppossant qu'il existe une preuve de toute ses prémisse, il existe une preuve de la conclusion. Une règle est admissible si elle est "inutile" au système mais est plus puissante qu'une règle dérivable (qui est elle juste qu'une macro). On a donc montré que la règle cut est admissible:

$$\frac{\vdash \Gamma, F \quad \vdash \Delta, F^\perp}{\vdash \Gamma, \Delta}\ \text{cut}$$

Et avec ça, on a montré le théorème bien compliqué que s'il existe une preuve utilisant la règle cut, il existe une sans.

Correspondance du système à la Hilbert[127]

Si F est une formule, on dit que G est une sous-formule de F (noté $G \prec F$) si on peut remplacer des variable propositionnelle de F en d'autres formules pour obtenir G. Par exemple, la formule $(X \wedge Y) \to (X \wedge Y) \wedge (X \to Y)$ est une sous-formule de $A \to A \wedge B$ obtenu en remplaçant A par $X \wedge Y$ et B par $X \to Y$. On ne peut pas remplacer la même variable par deux formules différente.

Pour R un ensemble de formules, on considère le *système logique à la hilbert sur R* (on le note avec $\rhd_H$) possédant les 2 règles suivantes:

$$\frac{}{\rhd_R F}\ \text{Ax} \qquad \text{pour } F \prec G, G \in R$$

$$\frac{\rhd_R A \qquad \rhd_R A \to B}{\rhd_R B} \to_e$$

Remarquer que comme R ne change jamais dans les règles, $\rhd_R$ en une sorte de système ou les axiomes sont les formules R avec uniquement la règle $\underset{e}{\to}$

On fixe $H = \{A \to B \to A, (A \to B \to C) \to (A \to B) \to (A \to C)\}$. On souhaite montrer que $\rhd_H X$ si et seulement si X est prouvable en logique minimale seulement avec les règles $\underset{i}{\to}$, $\underset{e}{\to}$, Ax.

1. Montrer le séquent $\rhd_H (A \to B) \to (A \to A)$ est prouvable.
2. Montrer que si $\rhd_X F$ est prouvable et que on a $F' \prec F$ et $X \subseteq X'$, alors $\rhd_{X'} F'$ l'est aussi. En déduire que $\rhd_H A \to A$ est prouvable avec une bonne instantiation de B dans la question précédente.

On fixe Γ un ensemble de formules propositionnelle et $\Delta = \Gamma \cup H$

3. Montrer que si $\rhd_\Delta F$ est prouvable, alors $\Gamma \vdash F$ est prouvable en logique minmale seulement avec les règles $\underset{i}{\to}$, $\underset{e}{\to}$, Ax
4. Montrer les 2 faits suivants:
   - Si $\rhd_\Delta B$ est prouvable alors $\rhd_\Delta A \to B$ est prouvable
   - Si $\rhd_\Delta A \to B \to C$ et $\rhd_\Delta A \to B$ sont prouvable alors $\rhd_\Delta A \to C$ est prouvable
5. Montrer que si $\rhd_{\Delta \cup \{F\}} G$ est prouvable alors $\rhd_\Delta F \to G$ l'est aussi.
6. En déduire la réciproque de la question 3

On peut s'amuser à ajouter les autres opérateurs !

7. Proposer une formule $F^i_\wedge$ de la forme $[\ldots] \to A \wedge B$ et deux formules $F^{e,g}_\wedge$ et $F^{e,d}_\wedge$ de la forme $A \wedge B \to [\ldots]$ de telle sorte a ce que $\vdash \varphi$ soit démontrable en logique minimale avec les règles Ax, $\underset{i}{\to}$, $\underset{e}{\to}$, $\wedge_i$, $\wedge^g_e$, $\wedge^d_e$ si et seulement si $\rhd_{H \cup \left\{F^i_\wedge, F^{e,g}_\wedge, F^{e,d}_\wedge\right\}} \varphi$ est prouvable.
8. Meme question pour le $\vee$ et le $\neg$.

[127] Partie 4 du projet de rocq du cours PRFA 2025/2026

[128] Tiré de https://www.lirmm.fr/~retore/LL/nonnon.pdf

$\neg\neg$-Traduction de Godel-Kolmogorov[128]

On note $\mathcal{L}$ l'ensemble des formules de logique propositionnelle. Pour $F \in \mathcal{L}$, on définit inductivement la non-non traduction de F note $F^{\neg\neg}$ par:

$$(A \wedge B)^{\neg\neg} := A^{\neg\neg} \wedge B^{\neg\neg} \qquad X^{\neg\neg} := \neg\neg X \quad \text{pour } X \text{ une variable}$$
$$(A \to B)^{\neg\neg} := A^{\neg\neg} \to B^{\neg\neg} \qquad (\neg A)^{\neg\neg} := \neg A^{\neg\neg}$$
$$(A \vee B)^{\neg\neg} := \neg\neg(A^{\neg\neg} \vee B^{\neg\neg})$$

Pour $\Gamma = \{\Gamma_1, ..., \Gamma_n\}$ on note $\Gamma^{\neg\neg} := \{\Gamma_1^{\neg\neg}, ..., \Gamma_n^{\neg\neg}\}$. On note $\Gamma \vdash_i F$ s'il existe une preuve de $\Gamma \vdash F$ en logique intuitioniste (sans utiliser la règle raa du raisonnement par l'absurde) et $\Gamma \vdash_c F$ s'il existe une preuve de $\Gamma \vdash F$ en logique classique.

1. Montrer que $\neg\neg F^{\neg\neg} \vdash_i F^{\neg\neg}$.
2. Montrer que $\Gamma \vdash_c F$ ssi $\Gamma^{\neg\neg} \vdash_i F^{\neg\neg}$.

Logique multiplicative

On définit la logique déductive multiplicative sur les formules constitué seulements de $\wedge$ et $\to$ par celle respectant les règles suivantes. Attention, pour l'axiome, on n'a pas de Γ :

$$\frac{\Gamma, A \vdash' B}{\Gamma \vdash' A \to B} \to_i \qquad \frac{\Gamma \vdash' A \to B \qquad \Delta \vdash' A}{\Gamma \cup \Delta \vdash' B} \to_e \qquad \frac{}{F \vdash' F}\ \text{Ax}$$

$$\frac{\Gamma \vdash' A \qquad \Delta \vdash' B}{\Gamma \cup \Delta \vdash' A \wedge B} \wedge_i \qquad \frac{\Gamma \vdash' A \wedge B}{\Gamma \vdash' A} \wedge_e^g \qquad \frac{\Gamma \vdash' A \wedge B}{\Gamma \vdash' B} \wedge_e^d$$

1. Montrer que le séquent $\vdash F$ est prouvable en logique minimale ssi $\vdash' F$ est prouvable en logique multiplicative.
2. Proposer des règles pour l'introduction et l'élimination du $\vee$ pour la logique multiplicative et traiter leur cas pour la question 1.

Modèles de Kripke

On pose $\mathcal{V}$ un ensemble dénombrable de variables propositionnelles. Soit F une formule sur $\mathcal{V}$, on appelle un **Modèle de Kripke** un arbre $T = (W, E)$ muni d'une fonction $\rho : W \longrightarrow \mathcal{P}(\mathcal{V})$ tel que pour tout enfant e d'un parent p, on a $\rho(p) \subseteq \rho(e)$. On note $\mathcal{E}(v)$ l'ensemble des descendants accessibles depuis v. On aura toujours $v \in \mathcal{E}(v)$.

On définit par induction le fait qu'une formule F est vérifié par un noeud $v \in V$ d'un modèle de Kripke $\mathcal{M}$ (noté $\mathcal{M}, v \vDash F$) par:

- $\mathcal{M}, v \vDash X$ si $X \in \rho(v)$
- $\mathcal{M}, v \vDash A \wedge B$ si $\mathcal{M}, v \vDash A$ et $\mathcal{M}, v \vDash B$
- $\mathcal{M}, v \vDash A \vee B$ si $\mathcal{M}, v \vDash A$ ou $\mathcal{M}, v \vDash B$
- $\mathcal{M}, v \vDash A \to B$ si pour tout $e \in \mathcal{E}(v)$, on a que $\mathcal{M}, e \vDash A$ implique $\mathcal{M}, e \vDash B$
- $\mathcal{M}, v \vDash \neg A$ si pour tout $e \in \mathcal{E}(v)$, on a pas $\mathcal{M}, e \vDash A$

On note $\mathcal{M} \vDash F$ pour dit que pour tout $v \in W$ on a $\mathcal{M}, v \vDash F$, et l'on notera $\vDash F$ si pour tout $\mathcal{M}$ on a $\mathcal{M} \vDash F$.

1. Donner un modèle de Kripke $\mathcal{M}$ tel que $\mathcal{M} \nvDash X \vee \neg X$
2. - Montrer le lemme de persistance: si $\mathcal{M}, v \vDash F$ et $w \in \mathcal{E}(v)$, alors $\mathcal{M}, w \vDash F$.
   - Montrer par induction que si $\Gamma \vdash F$ est prouvable en logique intuitioniste, alors pour tout modèle de Kripke tel que pour tout $A \in \Gamma$ on a $\mathcal{M} \vDash A$, on aura nécessairement $\mathcal{M} \vDash F$
3. En déduire que l'on ne peut pas démontrer ni $A \vee \neg A$ ni $\neg\neg A \to A$ en logique intuistioniste.

Logiques intermédiaires Pour F une formule de variables propositionnelles $X_1, ..., X_n$ et pour $F_1, ..., F_n$ des formules propositionnelles, on note $F' := F[X_1 \leftarrow F_1, ..., X_n \leftarrow F_n]$ la formule où, pour tout i, l'on a remplacé X_i par F_i. Dans ce cas on dit que F' est une instanciation de F. Par exemple, $(A \to B) \vee \neg(A \to B)$ est une instanciation de $F = A \vee \neg A$

Pour F une formule on note par $\vdash_F$ le système de la logique intuitioniste auquel on rajoute la règle

$$\frac{}{\Gamma \vdash_F F'} \text{Ax}_{F'} \text{ pour } F' \text{ une instanciation de } F$$

4. Montrer que $\vdash_{A \vee \neg A} F$ est prouvable ssi $\vdash F$ est prouvable en logique classique.
5. Montrer que s'il existe un modèle $\mathcal{M}$ tel que pour toute instanciation F' de $F = A \vee \neg A$, on a $\mathcal{M} \vDash F'$, alors on peut transformer $\mathcal{M}$ en un modèle $\mathcal{M}'$ reconnaissant les memes formules en un arbre de hauteur 0.

Soient $X_0, ..., X_n, ...$ une suite infinie dénombrable de variables propositionnelles. On définit

$$P_0 := X_0 \vee \neg X_0$$
$$\forall n \in \mathbb{N}, \quad P_{n+1} := X_{n+1} \vee (X_{n+1} \to P_n)$$

6. Montrer que $\vdash_{P_n} P_{n+1}$
7. Montrer qu'il nexiste pas de modèle $\mathcal{M}$ de profondeur de moins que n tels que $\mathcal{M} \vDash P'$ pour toute instanciation de P' de P_{n+1}
8. Montrer qu'il existe un moddèle de profondeur n tels que $\mathcal{M} \vDash P_n$
9. Pour F une formule, on pose $\mathcal{L}_F := \{\varphi \mid \vdash_F \varphi \text{ est prouvable}\}$. Montrer qu'il existe une suite infinie de formules $(\varphi_i)_{i \in \mathbb{N}}$ telles que

$$... \subsetneq \mathcal{L}_{\varphi_3} \subsetneq \mathcal{L}_{\varphi_2} \subsetneq \mathcal{L}_{\varphi_1} \subsetneq \mathcal{L}_{A \vee \neg A}$$

Fun fact: avec de la théorie plus poussé, on a que il y a un continuium $\aleph_1$ de logiques.

Logique modale[129]

On cherche ici à représenter des formules déontiques, c'est-à-dire des formules représentant plusieurs situations analogues qui se déroule en même temps et qui sont liées. Soit $\mathcal{V}$ un ensemble de variables propositionnelle, on définit par induction l'ensemble des formules déontique $\mathcal{D}$ comme:

- Si $X \in \mathcal{V}$ alors $X \in \mathcal{D}$,
- Pour tout $\varphi, \psi \in \mathcal{D}$, les formules $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \to \psi)$ et $\neg\varphi$ sont des formules déontique,
- Si $\varphi \in \mathcal{D}$ alors $\Box\varphi \in \mathcal{D}$ et $\Diamond \varphi \in \mathcal{D}$.

La formule $\Box\varphi$ se lit "partout φ" et la formule $\Diamond \varphi$ se lit "quelquepart φ"

1. Donner la formule représentant "si partout $A \to B$, et si quelquepart A, alors quelquepart B".

On cherche maintenant à attribuer une valeur de vérité à de telle formules. Soit F une formule de variables propositionnelles V, on définit un **modèle de Kripke de F** comme un couple $(\mu_1, ..., \mu_n)$ tel que pour tout $i \leq n$ on a $\mu_i : V \longrightarrow \{\top, \bot\}$ une valuation. Soit $\mathcal{M} = (\mu_1, ..., \mu_n)$ un modèle de Kripke, on définit $\mathcal{M} \vDash_i \varphi$ par induction:

[129] Centrale Info 2026

$$
\begin{aligned}
&\mathcal{M} \vDash_i A && \text{si } A \in V \text{ et } \mu_i(A) = \top \\
&\mathcal{M} \vDash_i \varphi \wedge \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ et } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \varphi \vee \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ ou } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \varphi \to \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ implique } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \neg\varphi && \text{si } \mathcal{M} \vDash_i \varphi \text{ est faux} \\
&\mathcal{M} \vDash_i \Diamond \varphi && \text{si } \exists j, \mathcal{M} \vDash_j \varphi \\
&\mathcal{M} \vDash_i \Box\varphi && \text{si } \forall j, \mathcal{M} \vDash_j \varphi
\end{aligned}
$$

Dans ce cas on dit que $\mathcal{M}$ *satisfait* φ au i-ème monde. Si $\mathcal{M} = (\mu_1, ..., \mu_n)$ et que pour tout i, $\mathcal{M} \underset{i}{\vDash} \varphi$ on dira que $\mathcal{M}$ satisfait φ. Si pour tout modèle $\mathcal{M}$ de Kripke on a $\mathcal{M} \vDash \varphi$, on dit que φ est une *tautologie*, et on note cela $\vDash \varphi$.

2. Pour chacune des formules φ suivante, indiquer s'il existe un modèle la satisfaisant, s'il existe un modèle satisfaisant $\neg\varphi$ et si elles sont des tautologies:

$\varphi_1 = "A \wedge \Diamond \neg A"$ $\varphi_2 = "\Box(A \wedge \Diamond \neg A)"$ $\varphi_3 = "\Box(\Box A \to \Box A)"$

3. Montrer que $\mathcal{M} \vDash_i \Diamond P$ est vrai si et seulement $\mathcal{M} \vDash_i \neg(\Box\neg P)$ est vrai.
4. En déduire un algorithme qui transforme une fomule déontique en une formule équivalente n'utilisant pas $\Diamond$.

On étudie maintenant que les formules déontiques sans $\Diamond$. On rajoute aux règles de la déduction naturelle de la logique classique les 3 règles suivantes:

$$\frac{\vdash \varphi}{\vdash \Box\varphi} D_1 \qquad \frac{\Gamma \vdash \Box(\varphi \to \psi)}{\Gamma \vdash \Box\varphi \to \Box\psi} D_2 \qquad \frac{\Gamma \vdash \Box\varphi}{\Gamma \vdash \neg\Box\neg\varphi} D_3$$

5. Montrer que $\vdash \neg(\Box A \wedge \Box\neg A)$
6. Montrer que $\vdash \Box(A \wedge B) \leftrightarrow (\Box A \wedge \Box B)$
7. Montrer que le système est correct, c'est-à-dire que si $\vdash \varphi$ est démontrable pour φ une formule sans $\Diamond$ alors $\vDash \varphi$ est vrai.

Logique linéaire temporelle & automates de Buchis (sujet d'écrit)

On cherche ici à représenter des formules qui utilise une certaine notion de temps qui passe. Soit $\mathcal{V}$ un ensemble fini de variables propositionnelle, on définit par induction l'ensemble $\mathcal{L}$ des formules temporelles linéaire comme:

- Si $X \in \mathcal{V}$ alors $X \in \mathcal{L}$,
- Pour tout $\varphi, \psi \in \mathcal{L}$, les formules $(\varphi \vee \psi), (\varphi \wedge \psi), (\varphi \to \psi)$ et $\neg\varphi$ appartiennent à $\mathcal{L}$,
- Si $\varphi \in \mathcal{L}$ alors $\Box\varphi \in \mathcal{L}$, $\bigcirc \varphi \in \mathcal{L}$ et $\Diamond \varphi \in \mathcal{L}$.

La formule $\Box\varphi$ se lit "pour toujours φ", la formule $\Diamond \varphi$ se lit "au bout d'un moment φ" et la formule $\bigcirc \varphi$ se lit "demain φ".

1. Donner la formule représentant "si au bout d'un moment on aura toujours φ, alors pour toujours au bout d'un moment φ".

On cherche maintenant à attribuer une valeur de vérité à de telle formules. Soit F une formule de variables propositionnelles $\mathcal{V}$, on définit un modèle $\mathcal{M}$ de logique linéaire temporel comme une suite infinite de valuation $\mathcal{M} = (\mu_i)_{i \in \mathbb{N}}$ tel que pour tout $i \in \mathbb{N}$ la valuation μ_i soit une valuation de $\mathcal{V}$. L'idée est que la valuation μ_i représente ce qui est vrai au jour i. On définit alors $\mathcal{M} \vDash_i \varphi$ par induction:

$$
\begin{aligned}
&\mathcal{M} \vDash_i A && \text{si } A \in V \text{ et } \mu_i(A) = \top \\
&\mathcal{M} \vDash_i \varphi \wedge \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ et } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \varphi \vee \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ ou } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \varphi \to \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ implique } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \neg\varphi && \text{si } \mathcal{M} \vDash_i \varphi \text{ est faux} \\
&\mathcal{M} \vDash_i \bigcirc \varphi && \text{si } \mathcal{M} \vDash_{i+1} \varphi \\
&\mathcal{M} \vDash_i \Box\varphi && \text{si } \forall j \geq i, \mathcal{M} \vDash_j \varphi \\
&\mathcal{M} \vDash_i \Diamond \varphi && \text{si } \exists j \geq i, \mathcal{M} \vDash_j \varphi
\end{aligned}
$$

Dans ce cas on dit que $\mathcal{M}$ *satisfait* φ au i-ème jour. Si $\mathcal{M} \underset{0}{\vDash} \varphi$ on dira que $\mathcal{M}$ satisfait φ. Si pour tout modèle $\mathcal{M}$ on a $\mathcal{M} \vDash \varphi$, on dit que φ est une *tautologie*, et on note cela $\vDash \varphi$. La précédence de $\bigcirc$, $\Box, \Diamond$ est la même que $\neg$, càd $\Box A \wedge B = (\Box A) \wedge B$ et $\Box A \to B = (\Box A) \to B$

2. Pour chacune des formules φ suivante, indiquer s'il existe un modèle la satisfaisant, s'il existe un modèle satisfaisant $\neg\varphi$ et si elles sont des tautologies:

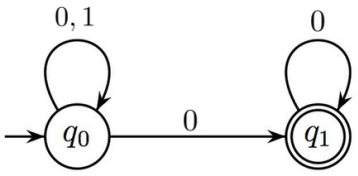
$\varphi_1 = A \wedge \Diamond \neg A$ $\qquad$ $\varphi_2 = \Box \Diamond \varphi \to \Diamond \bigcirc \varphi$ $\qquad$ $\varphi_3 = \Box(\varphi \to \psi) \wedge \neg \Diamond \psi$

3. Montrer que $\vDash \Diamond \Box A \to \Box \Diamond A$. Qu'est-ce que ça veux dire ?

Un **automate de Büchi** est un automate (Q, Σ, T, I, F) complet permettant de reconnaître des mots infinis, avec $I, F \subseteq Q$ et $T \subseteq Q \times \Sigma \times Q$. Soit X un ensemble de mots. On note X^ω l'ensemble des mots infinis de X défini formellement par $x \in X^\omega$ s'il existe une suite $(x_i)_{i \in \mathbb{N}}$ telle que $x = x_0 x_1 x_2 \ldots$

Un mot infini x sur Σ^ω est reconnu par un automate de Büchi lorsqu'il existe un chemin infini dans cet automate étiqueté par x, commençant par un état initial et passant une infinité de fois par un état final.

4. Quel est le langage reconnu par l'automate de Buchi:



5. Proposer un automate de Buchi recconnaissant $(01)^\omega$.

Comme $\mathcal{V}$ est fini, une valuation peut être représenté par $\mu \in \{\text{true}, \text{false}\}^{\mathcal{V}}$ qui est un ensemble fini. On pose $\Sigma_{\mathcal{V}} = \{\text{true}, \text{false}\}^{\mathcal{V}}$. Un modèle peut donc être assimilé à un mot infini $\mathcal{M} \in \Sigma_{\mathcal{V}}^\omega$. Pour φ une formule de logique linéaire temporelle, on note $L(\varphi) = \{\mathcal{M} \in \Sigma_{\mathcal{V}}^\omega \mid \mathcal{M} \vDash \varphi\}$ le langage reconnu par la formule.

6. Donner $L(\Box(x \to \bigcirc \neg x) \wedge \Box(\neg x \to \bigcirc x))$
7. Montrer que si $L(\varphi)$ et $L(\psi)$ sont reconnu par un automate de Buchi, alors $L(\varphi \vee \psi)$ est reconnu par un automate de buchi.
8. Soit $\mathcal{A} = (Q, \Sigma, T, I, F)$ un automate de Buchi sur Σ fini. Pour chaque état $q \in Q$, on pose X_q une variable propositionnelle représentant le fait que l'automate soit à l'état q et pour chaque $\alpha \in \Sigma$, on pose A_α la variable représentant le fait que l'on soit actuellement en train de lire un α. Donner des formules pour:

- Le fait que à tout moment, on ne soit que dans un et un seul état.
- Le fait que à tout moment, on ne lise qu'une et une seule lettre
- Le fait que on passe de X_q à $X_{q'}$ vrais seulement si A_α est vrai pour $(q, \alpha, q') \in T$
- Le fait que l'on passe une infinité de fois par $\{X_q : q \in F\}$

En déduire que si L est reconnu par automate de Buchi alors il existe une formule φ tel que $L(\varphi) = L$

On va essayer de montrer que L est reconnu par un automate de Buchi si et seulement si il existe φ tel que $L = L(\varphi)$, et donc que de tester si une formule est vrai ou fausse est décidable. On a fait le sens simple, mais la preuve du sens retour est très délicate, principalement dans le cas de la négation.

Soient φ, ψ deux formules de langage reconnu par $\mathcal{A} = (Q, \Sigma, T, I, F)$ et $\mathcal{A}' = (Q', \Sigma, T', I', F')$.

9. Montrer que $L(\Box\varphi)$ et $L(\bigcirc \varphi)$ sont reconnus par des automates de Buchi.
10. Pourquoi l'automate produit classique de l'intersection ne marche pas pour reconnaître $L(\varphi \wedge \psi)$?

En considérant une extension de l'automate produit $\mathcal{A}_\wedge = (Q_\wedge, \Sigma, T_\wedge, I_\wedge, F_\wedge)$ avec un bit d'information bonus $b \in \{0, 1\}$: on pose $Q_\wedge = Q \times Q' \times \{0, 1\}$ et $F_\wedge = F \times Q' \times \{0\} \cup Q \times F' \times \{0\}$, tel que à chaque rencontre avec un $F_\wedge$ on alterne le bit d'information $b \leftarrow 1 - b$.

10. Donner une définition formelle de $T_\wedge$. Montrer que $L(\varphi \wedge \psi)$ est reconnu par $\mathcal{A}_\wedge$.

MPI: Classes de Complexités

Cours

- Problème de décision, d'optimisation, de maximisation/minimisation. Comment transformer un problème d'optimisation en un problème de décision.
- Taille d'une instance.
- Définition d'une classe de complexité. Définition de P, NP, NP-Dur, NP-Complet
- Réduction polynomiale
- k-SAT, SAT, certificat

Question de Cours

- Comment transformer un probleme de maximisation en un problème de décision ?
- Montrer que 3-SAT est NP-Complet

Max 2-SAT

On consdière le problème d'optimisation suivant:

Entrée: φ une formule en FNC **Sortie:** Une valuation φ qui maximise le nombre de clauses satisfaites

1. Donner la version problème de décision
2. Montrer que si k est le nombre de clauses le problème deviens polynomial
3. Montrer que la version décisionnel du problème est NP-Complet

SUBSET-SUM

On considère le problème SUBSET-SUM suivant:

- **Entrée** $n \in \mathbb{N}$ et $a_1, a_2, ..., a_n, S \in \mathbb{N}$
- **Sortie:** Est-ce qu'il existe $I \subseteq [\![1; n]\!]$ tel que $\sum_{i \in I} a_i = S$

1. Pour les suites suivante de $(a_n)_n$ et les valeurs de S suivantes, indiquer si le problème est satisfiable ou non:
   - $(a_n)_n = (1, 2, 4, 8, 16), S = 7$
   - $(a_n)_n = (31, 24, 2, 43, 12, 12, 18), S = 29$
   - $(a_n)_n = (1001, 1010, 101, 100, 11), S = 1111$
2. Montrer que le problème SOMME est dans la classe NP
3. Donner un algorithme qui en $O(nS)$ résoud le problème

On cherche à prouver que le problème précédent est NP-Complet. Pour cela on considère une généralisation du problème sur des k-uplets intitulé SUBSET-SUM-VECT:
- **Entrée** $k, n \in \mathbb{N}$ et $a_1, a_2, ..., a_n, S \in \mathbb{N}^k$
- **Sortie:** Est-ce qu'il existe $I \subseteq [\![1; n]\!]$ tel que $\sum_{i \in I} a_i = S$

4. Montrer que on peut réduire le problème **SOMME-VECT** au problème **SOMME**
5. Montrer que **SOMME-VECT** est NP-Complet à partir de 3-SAT.
6. Est-ce que avec la question 3 on a démontré que P=NP ? Justifier.

INDEPENDANT-SET et CLIQUE

On considère les deux problèmes CLIQUE-MAX et INDEPENDANT-SET suivant:

CLIQUE:
- **Entrée:** Un graphe $G = (V, E)$ non orienté
- **Sortie:** Le plus grand $S \subseteq V$ en cardinal tel que $V^2 \subseteq E$

INDEPENDANT-SET:
- **Entrée:** Un graphe $G = (V, E)$ non orienté et un $k \in \mathbb{N}$
- **Sortie:** Est-ce qu'il existe $S \subseteq V$ tel que $S^2 \cap E = \emptyset$ avec $|S| \geq k$?

1. Donner la version problème de décision de CLIQUE. Montrer que elle est NP.
2. Donner une réduction polynomiale de INDEPENDANT-SET dans la version porblème de décision de CLIQUE.
3. En réduisant depuis 3-SAT, et en considérant un graphe avec $3k$ noeuds avec k le nombre de clauses, montrer que CLIQUE est NP-Complet.

MIN-COLOR-PATH

On considère le problème MIN-COLOR-PATH suivant:
- **Entrée:** $G = (V, E)$ un graphe, $s, t \in V$ deux sommets et $c : V \longrightarrow \mathbb{N}$ une coloration
- **Sortie:** Un chemin $s = x_1, .., x_n$ de s à t qui minimise $\text{Card}(\{c(x_1), ..., c(x_n)\})$

1. Donner la version problème de décision, et montrer qu'elle est NP.
2. Donner un algorithme pour résoudre le problème si G est un cycle.
3. Montrer que le problème de décision associé à MIN-COLOR-PATH est NP-Complet.

On suppose maintenant que l'on rajoute la condition $|\text{Im}(c)| \leq k$ avec $k \in \mathbb{N}$ fixé.

4. Montrer maintenant que le problème est P.

On considère l'algorithme qui prend des couleurs uniformément au hasard dans $\text{Im}(c)$ jusqu'à ce qu'il existe un chemin de s à t

5. Quel est ce type d'algorithme ? Proposer des exemples arbitrairement grand ou cet algorithme renvoie une solution d'éspérence en $O(n)$ au lieu de $O(1)$.

DOMINATING-SET

Soit $G = (V, E)$, on dit que $S \subseteq V$ est un *ensemble dominant* si tous les sommets de V sont soit dans S, soit voisin d'un sommet de S. On note $\gamma(G)$ le cardinal minimum d'un ensemble dominant de G.

On considère le problème DOMINATING-SET suivant:
- **Entrée:** $G = (V, E)$ un graphe
- **Sortie:** $\gamma(G)$

1. Donner la version problème de décision de DOMINATING-SET. Montrer qu'elle est NP.
2. Montrer que $\gamma(G) \leq |V|/2$
3. Montrer que la version problème de décision de DOMINATING-SET est NP-Complet.
4. Montrer que si l'on restrain l'entrée du problème à des arbres, alors le problème est dans P.
5. Pour G un graphe, on note Δ_G son degré maximal. Donner une Δ_G-approximation de DOMINATING-SET.
6. Donner similairement une $\log(|V|)$-approximation. Est-ce qu'une des approximations fait les deux ?

Langages NP-Complets sur $\Sigma = \{a\}$[130]

Un langage L est dit unaire si c'ets un langage sur un alphabet à une lettre, donc si $L \subseteq \{a\}^*$. L est dit *NP-Difficile* si le problème de tester si $x \in L$ est NP-Difficile. Dans ce problème, la taille de l'entrée est $|x|$.

1. On note SAT^+ l'ensemble des entrée positives au problème SAT et S. Montrer que L est NP-Difficile ssi il existe une fonction f de complexité polynomiale telle que

$$\forall \varphi, \varphi \in \mathrm{SAT}^+ \iff f(\varphi) \in L$$

On cherche à montrer le théorème suivant

Théorème de Berman: S'il existe un langage unaire NP-Dur alors $\mathrm{P} = \mathrm{NP}$

Soit $\varphi(x_1, x_2, ..., x_n)$ une instance de SAT. On suppose que L est un langage unaire NP-Difficile.
2. On note $\varphi_\top(x_2, ..., x_n) := \varphi(\top, x_1, ..., x_n)$ et $\varphi_\bot(x_2, ..., x_n) := \varphi(\bot, x_1, ..., x_n)$. Montrer que

$$\varphi \in \mathrm{SAT}^+ \iff \varphi_\top \in \mathrm{SAT}^+ \vee \varphi_\bot \in \mathrm{SAT}^+$$

On note $|\varphi|$ la taille d'une formule φ, et on suppose que $\forall i, |\top|, |\bot| < |x_i|$

3. Soit φ une formule à n variables. Donner un algorithme qui, pour tout $i \leq n$, construit un ensemble de formules Φ_i tel que toute formule de Φ_i contient $n - i$ variables et $\varphi \in \mathrm{SAT}^+ \iff \exists \varphi \in \Phi_i, \varphi \in \mathrm{SAT}^+$
4. Montrer qu'il existe un polynome p tel que $\sum_{\varphi \in \Phi_i} |\varphi| \leq p(|\varphi|)$
5. En déduire que $\mathrm{P} = \mathrm{NP}$, et donc le théorème de Berman.

Arbre couvrant minimum en couleurs (MLST)

Un *multi-graphe* est un graphe dans lequel on peut avoir plusieurs fois une arête entre 2 sommets.
On considère le problème MIN-COLOR-TREE suivant:
- **Entrée:** $G = (S, A)$ un multigraphe et $c : A \longrightarrow \mathbb{N}$ une coloration des arrettes.
- **Sortie:** Un arbre couvrant $T = (S, A')$ de G qui minimise $\mathrm{Card}(\{c(e) : e \in A'\})$

On note n le nombre de sommets et m le nombre d'arêtes.

[130] Mallory Marin, doc de colles

1. Donner la version problème de décision, et montrer qu'elle est NP.
2. Montrer que si toutes les arêtes sont de couleur différentes alors il existe un algorithme polynomial.
3. Soit $k \in \mathbb{N}$ fixé. Montrer que si le nombre de couleurs distinctes de c est au plus k, il existe un algorithme en $O\big(2^k(n+m)\big)$ résolvant le problème.

On considère le problème SET-COVER suivant :
- **Entrée:** $n, m, k \in \mathbb{N}$ et des ensembles $U_1, ..., U_m \subseteq [\![1; n]\!]$
- **Sortie:** Est-ce qu'il existe $I \subseteq [\![1; m]\!]$ avec $|I| \leq k$ tel que $\bigcup_{i \in I} U_i = [\![1; n]\!]$?

4. On admet que le problème SET-COVER est NP-dur. Montrer que le problème de décision associé à MIN-COLOR-TREE est NP-Dur.

Activation de Processus[131]

Soit un système temps réel à n processus asynchrones $i \in [\![1; n]\!]$ et m ressources $\left(r_j\right)_{j \leq m}$. Quand un processus i est actif, il bloque un certain nombre de ressources listées dans un ensemble P_i et une ressource ne peut être utilisée que par un seul processus. On cherche à activer simultanément le plus de processus possible. Le problème de décision `ACTIVATION` correspondant ajoute un entier k aux entrées et cherche à répondre à la question : "Est-il possible d'activer au moins k processus en même temps ?""

1. Soit $n = 4$ et $m = 5$. On suppose que $P_1 = \{r_1, r_2\}$, $P_2 = \{r_1, r_3\}$, $P_3 = \{r_2, r_4, r_5\}$ et $P_4 = \{r_1, r_2, r_4\}$. Est-il possible d'activer 2 processus en même temps ? Même question avec 3 processus.
2. Dans le cas où chaque processus n'utilise qu'une seule ressource, proposer un algorithme résolvant le problème `ACTIVATION`. Évaluer la complexité de votre algorithme.

On souhaite montrer que `ACTIVATION` est NP-complet

3. Donner un certificat pour ce problème.
4. Ecrire en pseudo code un algorithme de vérification polynomial. On supposera disposer de trois primitives, toutes trois de complexité polynomiale :
   - `appartient(c,i)` qui renvoie Vrai si le processus `i` est dans l'ensemble d'entiers `c`.
   - `intersecte(`P_i`,R)` qui renvoie `Vrai` si le processus `i` utilise une ressource incluse dans un ensemble de ressources `R`.
   - `ajoute(`P_i`,R)` qui ajoute les ressources P_i dans l'ensemble `R` et renvoie ce nouvel ensemble.

En théorie des graphes, le problème `STABLE` se pose la question de l'existence dans un graphe non orienté $G = (S, A)$ d'un ensemble d'au moins k sommets ne contenant aucune paire de sommets voisins. En d'autres termes, existe-t-il $S' \subseteq S, |S'| \geq k$ tel que $\forall s, p \in S', (s, p) \notin A$?

5. En utilisant le fait que `STABLE` est NP-complet, montrer par réduction que le problème `ACTIVATION` est également NP-complet.

Intersection d'automates[132]

On considère le problème NONEMPTY-REGULAR-INTERSECTION suivant:

[131] copie presque exacte CCINP 2023 sujet A
[132] Algo 1 ENS Lyon TD 8

Entrée: $A_1, ..., A_n$ une liste d'automate sur $\Sigma = \{a, b\}$
Sortie Est-ce que

$$\bigcap_{i \leq n} L(A_i) \neq \emptyset$$

1. Montrer que NONEMPTY-REGULAR-INTERSECTION est NP
2. Donner une instance du problème tel que le plus petit mot dans l'intersection est de longueur exponentielle en la taille de l'entrée.
3. Montrer que si $\Sigma = \{a\}$ alors le problème est polynomial.
4. Montrer que NONEMPTY-REGULAR-INTERSECTION est NP-Complet à partir de 3-SAT

Sheffer

On définit l'opréateur de sheffer $X \uparrow Y := \neg(X \wedge Y)$ et l'opérateur $X \oplus Y := (X \vee Y) \wedge (\neg X \vee \neg Y)$. Indiquer si les problèmes suivant sont Np-Complet ou Polynomial:

- **Entrée:** Une formule φ avec que des $\oplus$ **Sortie:** Si φ est satisfiable.
- **Entrée:** Une formule φ avec que des $\wedge$ et $\neg$ **Sortie:** Si φ est satisfiable.
- **Entrée:** Une formule φ avec que des $\uparrow$ **Sortie:** Si φ est satisfiable.

Plus dur, essayer de trouver une réduction de SAT (pas 3-SAT) aux formules avec $\uparrow$.

MPI: Calculabilité

Cours

- Problème décidable, indécidable
- Le problème de l'arret est indécidable
- Réduction calculable
- Problèmes semi-décidable, co-semi-décidable (HP).
- Fonction calculable (HP)

Question de Cours

- Rapeller le problème de l'arret. Montrer qu'il est indécidable.
- Donner un exemple de problème semi-décidable et montrer qu'il est semi-décidable. (HP)
- Donner un exemple de problème qui n'est pas semi-décidable et le prouver. (HP)

Indécidabilité

Montrer que les problèmes sont indécidables:

- **ε-Arret: Entrée:** Un programme P, **Sortie:** Est-ce que $P(\varepsilon)$ termine ?
- **42-Arret: Entrée:** Un programme P, **Sortie:** Est-ce que $P(42)$ termine ?
- **$\neg$-Arret: Entrée:** Un programme P et une entrée w, **Sortie:** Est-ce que $P(w)$ ne termine pas ?
- **Equivalence: Entrée:** Deux programmes P, Q prenant en entrée des naturels, **Sortie:** Est-ce que pour tout $n \in \mathbb{N}$, $P(n)$ termine ssi $Q(n)$ termine ?
- **Arret-$\exists$: Entrée:** Un programme P prenant en entrée un naturel, **Sortie:** Est-ce qu'il existe un n tel que $P(n)$ termine ?
- **Arret-$\forall$: Entrée:** Un programme P prenant en entrée un naturel, **Sortie:** Est-ce que pour tout $n \in \mathbb{N}$ on a $P(n)$ qui termine ?

ARRET $\text{Bool} \to \text{Bool}$[133]

On considère la grammaire G suivante (en gras les symboles non terminaux) :

[133] Un mines-télécom bof de 2024

$$
\begin{aligned}
\boldsymbol{S} &\longrightarrow \text{bool f (bool x[]) \{ } \boldsymbol{I} \text{ return false ;\}} \\
\boldsymbol{E} &\longrightarrow \text{true} \mid \text{false} \mid \boldsymbol{V} \mid \boldsymbol{E} == \boldsymbol{E} \\
\boldsymbol{V} &\longrightarrow \text{x[i]} \qquad\qquad\qquad\qquad\qquad\qquad \forall i \in \mathbb{N} \\
\boldsymbol{I} &\longrightarrow \text{return } \boldsymbol{E} \ ; \mid \boldsymbol{B} \mid \boldsymbol{C} \\
\boldsymbol{B} &\longrightarrow \text{while } (\boldsymbol{E}) \ \{\boldsymbol{I}\} \\
\boldsymbol{C} &\longrightarrow \text{if } (\boldsymbol{E}) \ \{\boldsymbol{I}\} \text{ else } \{\boldsymbol{I}\}
\end{aligned}
$$

On considère $L_{\text{bool}} = L(G)$ et $L_{\text{bool bool}}$ le langage engendrée par G sans la dérivation B.

1. Est-ce que le problème suivant est décidable ?
   - **Entrée:**$\kappa \in L_{\text{bool bool}}$,
   - **Sortie:** Est-ce que $\kappa(x)$ termine (avec ou sans erreur) pour tout x ?
2. Soit η obtenue à partir de $\boldsymbol{E}$. Montrer qu'il existe une formule ψ_η tel que pour tout ν une valuation, $\psi_\eta(\nu)$ correspond à la valuer de sortie de η pour $x[i] = \nu(X_i)$
3. Donner une formule équivalente à ce code

```
if (x[0]) {
  if (x[0] == x[1]) {return false;}
  else {return true;}
} else {
  return {x[1] == x[2];}
}
```

4. Est-ce que le problème suivant est décidable ?
   - **Entrée:**$\kappa \in L_{\text{bool}}$,
   - **Sortie:** Est-ce que $\kappa(x)$ termine pour tout x ?
5. Est-ce que le problème suivant est décidable ?
   - **Entrée:**$\kappa \in L_{\text{bool}}$,
   - **Sortie:** Est-ce qu'il existe x tel que $\kappa(x)$ termine ?

Entrée bornée

Montrer que le problème suivant est indécidable:

- **Entrée:** Une fonction calculable $P : \mathbb{N} \to \mathbb{N}$ qui termine toujours.
- **Sortie:** Est-ce qu'il existe un n tel que $P(n) = 0$?

L'indécidable est partout

Soit S un ensemble de string, on considère le problème P_S suivant:

- **Entrée:** s une chaine de caractères
- **Sortie:** Est-ce que $n \in S$?

1. Montrer qu'il existe des S tel que P_S est indécidable.
2. Montrer que si C est infini, alors il existe $K \subseteq C$ tel que P_K est indécidable. Est-ce vrai si C est fini ?

Complexité et décidabilité[134]

Montrer que pour toute fonction calculable g : String $\to \mathbb{N}$, il existe une fonction calculable f : String $\to$ char si complexe que pour tout programme P qui calcule f, il existe une entrée $u \in$ String sur laquelle $P(u)$ prend plus de $g(u)$ étapes de calcul.

[134] tiré d'un oral d'Ulm 2021 https://a3nm.net/work/exams/ens/exercices_info_ulm_2021.pdf

Existence d'une quine

Donner un code OCaml non vide $w \in \text{ASCII}^*$ tel que l'exécution de ce code affiche le `string` w.

Réels calculables

1. Rappeler la représentation binaire des flottants en machine.

Les flottants machine ne peuvent pas représenter tout les nombres décimaux qui existe, et encore moins tout les réel. On introduit ici une autre manière de représenter plus de réels via des fonctions.

Un réel $x \in [0, 1[$ est dit *calculable au sens de Cauchy* s'il existe un programme $P : \mathbb{N} \to \mathbb{Q}$ tel que, pour tout $n \in \mathbb{N}$,

$$|P(n) - x| \leq 2^{-n}.$$

On dit alors que P est un *nom de Cauchy* de x. On note $\mathbb{K}$ l'ensemble des réels calculables.

2. Montrer que tout rationnel $q \in [0, 1[$ est calculable.
3. Montrer que si $x, y \in \mathbb{K}$, alors $x + y$ (quand $x + y < 1$) et xy sont calculables.
   Ind: On rappelle que $|ab - a'b'| \leq |ab - ab'| + |ab' - a'b'|$.
4. Montrer que le probleme de tester si deux nombres réels sont égaux est indécidable avec cette représentation. On pourra admettre que le problème suivant est indécidble:
   - **Entrée:** Un programme $A : \mathbb{N} \to \text{bool}$ qui termine toujours
   - **Sortie:** Est-ce qu'il existe un n tel que $A(n) = \text{true}$?

On dit qu'un problème de décision est *semi-décidable* s'il existe un algorithme A tel que :
- A répond VRAI sur toutes les instances positives
- A répond FAUX ou ne termine pas sur toutes les instances négatives.

5. Montrer que le problème suivant est semi-décidable :
   - **Entrée:** $x, y \in [0; 1[$ représenté par leur nom de Cauchy.
   - **Sortie** Est-ce que $x < y$?
6. Montrer qu'il existe des réels qui ne sont pas calculable.

On étudie maintenant les réels calculable sur tout $\mathbb{R}$ et pas que $[0; 1[$. On garde la meme définition, mais pour $x \in \mathbb{R}$.

7. Montrer que $\mathbb{K}$ est un $\mathbb{Q}$-espace vectoriel.
8. $(*)$ Est-ce qu'il existe un algorithme $A : \mathbb{N} \longrightarrow \mathbb{K}$ tel que $(A(n))_{n\in\mathbb{N}}$ forme une base de $\mathbb{K}$ en tant que $\mathbb{Q}$-espace vectoriel?

Représentation d'ensembles infini

On se propose ici de créer une structure de données permettant de représenter un ensemble infini sous la forme d'une fonction de `'a -> bool` déterministe et qui **termine toujours**. On définit donc le type suivant en OCaml:

```
type 'a set = 'a -> bool;;
```

1. Donner un ensemble $A \subseteq \mathbb{N}$ que l'on ne pourra pas représenter avec notre structure. Le nombre d'ensembles non représentable est-t'il fini ? Dénombrable ? Indénombrable ?
2. Montrer que la fonction `val is_empty: int set -> bool` qui à un ensemble donné indique s'il est vide ou non n'est pas calculable.
3. Soit `val P: int set set`, montrer qu'il existe un $N \in \mathbb{N}$ tel que pour tout `val x: int set`, dans le calcul de `P x`, `P` n'évalue `x` que sur entrées inférieure à N. Est-ce vrai avec `val P: (int -> int) set` ?

4. (∗) Montrer que la fonction `val is_empty: (int set set) -> int set` qui à un `int set set` donné indique s'il est vide ou non est calculable.