

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 préparateurs 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. J'essaie autant que possible de donner un lien vers la source 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 rescane de temps en temps. Vous pouvez me contacter, au choix, à la-totale@bourotte.com, colles@bourotte.com ou prepa@bourotte.com.

Lien vers ce même document : <https://typst.app/project/RIXQenh6QVwpDr2IYqAQLq>

Certaines sources 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

Actuellement, je n'ai pas fini de transférer au moins 40% des exos que je veux transférer.

Contents

La Totale – Introduction	1
MP2I: Programmation impérative, représentation machine	8
Cours	8
Complexité (oui du OCaml ici écoute hein)	8
MP2I: Programmation fonctionnelle	8
Cours	8
Mémoïsation Automatique ¹	9
Suspension et listes infinies	9

¹de Maxime BRIDOUX

Continuation	10
Binary random acces lists	11
Monades	12
Parsing avec des monades ²	13
MP2I: Structures de données séquentielles	16
Cours	16
Merge de liste	16
Calcul des différences	16
Permutation suivante	16
Tableaux binaire	16
Sous-tableau connexe	17
Structure efficace pour représenter des ensembles	17
Liste cyclique sans lièvre et torture	17
Recherche dans une matrice	17
Tableaux auto-référents	18
Tableau cumulatif	18
Multiplication rapide de polynome ³	19
Le glouton par défaut ⁴	19
MP2I: Inductions & ordres⁵	20
Cours	20
Des ordres	20
Des ensembles inductifs	20
Soustraction entière	20
Ordre sur les mots	20
Tri aléatoire	21
Langage de Dyck	21
Système MIU	21
Sous-ordres indénombrable de $(\mathcal{P}(\mathbb{N}), \subseteq)$	21
MP2I: Arbres & structures inductives	22
Cours	22
Questions de Prog	22
Nombre de sommets	22
Plus de feuilles	22
Tableaux tri-coloré	22
Arbre et oracle	22
Accès dans un arbres parfait	22
Arbres 2-dimensionels	23
Arbres généraux	23
Mots d'arbres	24
Initialisation d'un Tas	24
Arbre canonique (ENS 2023 MPI Info A)	25
Arbres d'intervalles	25
Reconstruire l'arbre avec des parcours	26
Ranger des boites	26
Préfixe vers Suffixe	27

²MPRI - FUN Exam 2026

³Algo 1 ENS Lyon

⁴Algo 1 ENS Lyon

⁵Maxime Bridoux, MP2I

Indépendant dans les arbres	27
Arbre α -équilibré	27
Comptage de sous-arbres ⁶	28
MP2I: Graphes (introduction)	29
Cours	29
Petites questions	29
Autour des degrés	29
Longs Chemins	29
Chemin hamiltonien dans un tournoi	29
Le puits	29
Reconnaitre les graphes biconnecté en $O(V + E)$	30
Reconnaitre un DFS	30
Graphes d'Halin ⁷ sont 3-connexes	30
Graphes d'Halin ⁸ sont hamiltoniens	30
Calcul des triangles	31
Implémentations incorrectes de Floyd-Warshall	31
MP2I: Logique Propositionnelle	32
Cours	32
Formules linéaire	32
Formules duales	32
Formule pour la bi-partition	32
Formule pour le centre d'un graphe	32
XOR SAT	32
Vers le 3-Fnc	33
Push down not	33
Equivalences	33
Retirer les constantes	33
Compacité	34
Indéterminé	34
Lemme d'interpolation de Craig	35
Equivalence	35
MPI: Langages	36
Petites questions	36
Suite de mots par récurrence	36
Mot circulaire	36
Mots univers	36
Egalité pour les résiduels	37
Hypercubes et mots	37
Equations de mots (oral info LCR ENS 2022)	37
Ensemble inévitables (oral info Ulm ENS 2019)	37
Points fixe de morphismes ⁹	37
Mots sans carré ¹⁰	38

⁶https://diplome.di.ens.fr/informatique-ens/Annales/2017_InfoU-exercices.pdf

⁷On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

⁸On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

⁹Tiré de l'Info D 2026 de décembre

¹⁰Oral de l'ENS Info 2019 et sujet de CCINP d'Informatique 2026

Mots sans cube	38
(*) Egalité pour les résiduels	38
(***) Lemme d'Higman	38
MPI : Langages réguliers et Automates	39
Questions de Cours	39
Berry-Sethi	39
Langages réguliers	39
Non régulier	39
Petites questions	39
Préfixes clos dans L	39
Langage croisé	39
Sous-langage palindromique	40
Langage régulier à partir de l'automate ¹¹	40
Langage rotationnel	40
Mots univers	41
Astucieux langage ¹²	41
Régulier à une lettre	41
Permuté	41
Langages surprenants	41
Automates partiellement ordonnés ¹³	41
Automates avec couts (sujet bof + algo A^*)	42
Commutateurs de Langages	42
Commutant de Langage ¹⁴	42
Langage coupé au milieu	42
Langages fins	43
Langages épars ¹⁵	43
Langages rationnel infinis	43
Racine de langage ¹⁶	43
Cloture par sur-mot ¹⁷	44
Automates (n, d) -locaux (*)	44
Langages continuables et mots primitifs	44
MPI: Langages Hors-contexte	45
Cours	45
Petites questions	45
Langage hors-contexte	45
Paires différentes	45
Autant de lettres	45
MPI: Graphes avancé	46
Cours	46
Question de Cours	46
Petites questions	46
Graphes à unique sortie	46

¹¹Colle de José

¹²Le fabuleux cours de théorie des langages formels d'Olivier Carton

¹³Oral ENS 2022 C2 https://diplome.di.ens.fr/informatique-ens/annales/2022_InfoU-exercices.pdf

¹⁴Sujet de José

¹⁵ENS Ulm oral d'info, je ne sais plus de quand

¹⁶Oral de l'X

¹⁷ENS Ulm de je ne sais plus quand

2-approximation du circuit eulérien (TSP)	46
3/2-approx de TSP	47
($\Delta + 1$)-approx de Maximum independant set	47
Kruskal en recherche locale	47
Graphes k -réguliers	48
Théorème de Hall	48
Maille d'un graphe	48
Largeur de bande	49
Graphes planaires	49
Couverture d'aretes et couplages	50
C -approx du couplages maximum ¹⁸	50
Calcul de triangles ¹⁹	50
Graphes d'amis	51
Graphes d'amis 2	51
Planaire et degré	51
Un Graphe dénombrable	51
Graphes d -dégénéré ²⁰	51
Coeur de k -domination ²¹	52
Théorème de vizig ²²	52
FPT: Kernelisation de Feedback vertex set	53
FPT: Kernelisation de Vertex cover	53
FPT: Kernelisation de Edge clique cover	54
MPI: Théorie des jeux	55
Cours	55
Petites questions	55
Jeu de nim généralisé	55
Géographie dans les hypercubes ²³	55
Géographie ²⁴	56
Jeu de Shannon	56
Jeu du sous-mot	56
MPI: Apprentissage	58
Questions de Cours	58
Petites questions	58
Single-pass (CCINP 2024) ²⁵	58
Clustering en dimension 1 ²⁶	58
k -centres	59
Algorithme ID3 avec intervalles	59
MPI: Système, Mutex & Sémaphore	61
Lecteurs-Rédacteurs	61
Votes en parrallèles	61

¹⁸TD11 L3 ENS Lyon Algo 1

¹⁹Tiré de Mallory Marin

²⁰InfoFonda 2026 partie I

²¹InfoFonda 2026 partie IV

²²InfoFonda 2026 partie V

²³Le putnam de 2025

²⁴Marathon de math de Paris-Orsay 2021

²⁵<https://prepas-mp2i.fr/documents/sujets/2024/CCINP-INFO.pdf>

²⁶tiré de https://mpi-lamartin.github.io/mpi-info/docs/ia/non_supervise/td_apprentissage

Salle de spectacle	61
Sémaphore pondéré	62
Mutex ré-entrant	62
Pont à voie unique	63
MPI: Algo avancé	64
Cours	64
Question de Cours	64
Petites questions	64
C-approx du couplages maximum ²⁷	64
Set cover	64
Unique graphe infini aléatoire	65
Coloration aléatoire	65
CComplexité en log* de Union Find ²⁸	66
Complexité en α de Union-find ²⁹	66
MPI: Déduction naturelle	67
Cours	67
Arbres simples en prop	67
Arbres des lois de morgan	67
Arbre des lois de morgan avec quantificateurs	67
Equivalence entre les formules donnant la logique classique	67
Arbres nécessitant de deviner	67
Affaiblissement	68
XOR	68
Equivalence	68
L'opérateur de Sheffer	68
Sans implication	68
Complétude ³⁰	68
Sans négation	69
Théorie déductive des graphes	69
Formule Duale	69
Introduction à la logique linéaire multiplicative	69
Correspondance du système à la Hilbert ³¹	70
$\neg\neg$ -Traduction de Godel-Kolmogorov ³²	71
Logique multiplicative	71
MPI: Classes de Complexités	73
Cours	73
Question de Cours	73
SAT-LIKE ³³	73
Liste de potentielle réductions classique	73
Max 2-SAT	73
Langages NP-Complets sur $\Sigma = \{a\}$ ³⁴	73

²⁷TD11 L3 ENS Lyon Algo 1

²⁸Colle de José

²⁹Algo 1, TD 4 ENS Lyon

³⁰Le magnifique livre de Mme. Galatée Hemery !!!! <3

³¹Partie 4 du projet de rocq du cours PRFA 2025/2026

³²Tiré de <https://www.lirmm.fr/~retore/LL/nonnon.pdf>

³³Algo 1 ENS Lyon

³⁴Mallory Marin, doc de colles

Intersection d'automates ³⁵	74
Sheffer	74
MPI: Calculabilité	75
Question de Cours	75
Indécidabilité	75
Entrée bornée	75
L'indécidable est partout	75
Complexité et décidabilité ³⁶	75
Existence d'une quine	75
Représentation d'ensembles infini	75

³⁵ Algo 1 ENS Lyon TD 8

³⁶ 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 à des bases d'impératif et du 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 bizzarerie avec des flottants.

Cours

- Paradigmes: impératif structuré, déclaratifs 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, cout amorti
- Spécialisation d'une fonction. Commentaires, annotation
- Programmation défensive, assertions
- Jeu de test. Graphe de flot de controle. Chemin faisable. Couverture des sommets, couverture des arretes. Test exhaustif d'une boucle.
- Pointeurs, allocation, stack, heap, call stack, stack overflow.

Complexité (oui du OCaml ici écoute hein)

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. Quel 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 equations sur la complexité

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

Je préviens (si tu commence la lecture par cette section) que les exercices ici (surtout vers la fin) sont particulièrement difficile. 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 Automatique³⁷

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 mémoisée 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 mémoisée 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 mémoisée 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. Ecrire 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. Ecrire une fonction `sappend : 'a slist -> 'a slist -> 'a slist` qui concatène en $O(1)$ deux listes paresseuses.

³⁷de Maxime BRIDOUX

7. Ecrire une fonction `srev` : `'a list -> 'a list` 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 *tout* 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 noeud. 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 reprends le code d'exemple pour la fonction `height1`

5. Compléter le code à trou suivant. `K1`, `K2`, `K3` sont les fonctions anonymes marquées des commentaires `(*1*)`, `(*2*)` et `(*3*)` respectivement. La composition d'une suite de fonctions anonymes est alors représenté par une liste de ses dites fonctions qui la compose, aka le type `cont`. `apply k v` simule l'application de la fonction anonyme représenté 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 access 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 que 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 problème. Considerer 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 subtilités (aka on donne le type de la fonction).

2. Ecrire une fonction `val cons : 'a. 'a -> 'a seq -> 'a seq` tel que `cons x xs` ajoute `x` au début de la liste `xs`.
3. Ecrire 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. Ecrire une fonction `val get : 'a. 'a seq -> int -> 'a` tel que `get xs i` renvoie le $(i + 1)$ -ème élément de `xs`.
6. Ecrire 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 cherche 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 défonctionnalisation 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 devrait plus voir de fonction anonymes.

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

Trop cool, on a obtenu un `set` en $O(\log n)$ qui n'utilise pas de fonction 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 égalités :

- `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 `ListMonad` avec `type 'a m = 'a list`

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

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

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

Question 4 Montrer que map et join satisfont les 3 équations suivantes :

- $\text{join} (\text{map} \text{ join } x) = \text{join} (\text{join } x)$
- $\text{join} (\text{map} \text{ return } x) = \text{join} (\text{return } x)$
- $\text{join} (\text{map} (\text{map } f)) x = \text{map } f (\text{join } x)$

TODO: Check???!?

Question 4 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 les 3 lois sont respectées.

Question 3 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 4 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 5 Proposer une implémentation récursive naive de la fonction de fibonacci (sans ce soucier de l'explosion des appels récursif) en utilisant la monade ContIntMonad

Parsing avec des monades³⁸

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

On cherche à définir un parser vers en OCaml en utilisant des combinateurs que l'on pourra imbriquer les uns aux autres. Un parser sera une fonction de $\text{char list} \rightarrow ('a * \text{char list}) \text{ seq}$, avec seq les séquences (pottentiellement infinis) 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 que un objet de type 'a parser est une fonction qui prend une liste de caractère et qui renvoie une séquence (pottentiellement infini) de résultats de matche. 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.

³⁸MPRI - 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 $[(0, 453v7), (4, 53v7), (45, 3v7), (453, v7)]$. Dans ce cas, les caractères `v` et `7` ne sont jamais consommés.

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 `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 $p(l) = [x_1, l_1, \dots, x_n, l_n, \dots]$ 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 effectue 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 `p1`, puis ensuite avec `p2`, et applique la fonction résultat du parsing de `p1` au résultat du parsing de `p2`, 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 fait `parse_d` défini 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 Ecrire 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 combinateur précédemment utilisé

Question 11 Ecrire 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 chaînée. D'un tableau. Cout des opérations de création, ajout, suppression en temps et mémoire.
- Différence entre structure immuable et mutable.
- Définition d'une pile. Implémentation d'une pile par tableau dynamique.
- Définition d'une file. Implémentation par une liste doublement chaînée, 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 en un String.

Algorithmes sur les textes:

- Expliquer l'algorithme de Lempel Ziv Welch sur un exemple
- Expliquer l'algorithme de Huffman sur un exemple
- Expliquer l'algorithme de Boyer-Moore sur un exemple
- Expliquer l'algorithme de Rabin-Karp sur un exemple

Techniques de Programmations:

- Recherche exhaustive, algorithme de retour sur trace.

Merge de liste

Soit L_1, L_2 deux listes triées. Proposer un algorithme en ocaml qui calcule la liste triée 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 $\llbracket 0; N \rrbracket$ représenté par T retourne la prochaine permutation de $\llbracket 0; N \rrbracket$ 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 colonne 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.

Structure efficace pour représenter des ensembles

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

- `val create : int -> set` telle que `create N` renvoie \emptyset (un sous-ensemble de $\llbracket 0; N \rrbracket$)
- `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. Quels 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 que un entier N a une taille de mémoire $\log_2(N)$.

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

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

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 implémentation 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 $O(N)$?

Liste cyclique sans lièvre et torture

1. Proposer une structure pour implémenter une liste simplement chaînée 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 représentée est infinie et, à partir d'un certain rang, devient périodique.

2. Proposer un code C pour définir la liste cyclique qui représente $[1, 2, 3, 2, 3, \dots]$ 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 chaînée 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 telles listes en OCaml! Le code suivant définit la liste de la question 2 : `let q2 = 1 :: (let rec l = 2 :: 3 :: l in l)`

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 colonne est trié de gauche à droite.

Question 2 Donner le code d'une 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 régner.

Question 4 Question bonus (demander moi une fois avoir fait la question précédente)

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

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 a un tableau t de longueur n renvoie H_t son histogramme.

Question 2 Donner tout 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 test 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 (Si la question 5 a été répondu ou à la fin de la colle)

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 a 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

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 polynome³⁹

Ici on considère des polynomes 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 prends deux polynomes de degré $\leq n$ représentés 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 $Q = Q_1 + X^m Q_2$. 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. Quel est la complexité de cet algorithme?

Le glouton par défaut⁴⁰

Étant donné un ensemble $\{x_1, \dots, x_n\}$ 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.

⁴⁰ Algo 1 ENS Lyon

MP2I: Inductions & ordres⁴¹

Cours

- Définition d'une relation d'équivalence.
- Définition d'une relation d'ordre. Ordre total.
- Minimum, éléments comparables, incomparables.
- Les deux définitions d'un ordre bien fondé, et la démonstration de leur équivalence.
- 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_{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 $\text{sub} : \mathbb{N}^2 \rightarrow \mathbb{N}$ 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

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

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

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

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

⁴¹Maxime Bridoux, MP2I

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 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 . 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 tout les mots dans A ont autant de a que de b
2. Montrer que l'ensemble de ces règles sont ambiguës. Proposer sans justifier une version non-ambiguë (et faite la vérifier par la colleuse avant de passer à la suite).
3. Soit $w_1 \dots w_n$ un mot de A , montrer qu'il existe un $i \leq n$ tel que pour tout $j < i$, le mot $w_1 \dots w_j$ possède strictement plus de a que de b et que le mot $w_1 \dots w_i$ 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 chaînes de S que l'on peut dériver en appliquant au plus 3 règles.
3. Démontrer que toute chaîne $s \in S$ commence par un M
4. Démontrer que pour toute chaîne $s \in S$, son nombre de $|s|_M$ de M est exactement 1
5. Démontrer que pour toute chaîne $s \in S$, 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 tout 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?

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 définition 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 Prog

- Coder en ocaml la fonction `add (t: int tree) (x:int) : int tree` qui ajoute à un arbre binaire de recherche `t` un élément `x`.
- Donner le type en C d'un arbre binaire stoquant des entiers et la fonction `bool find(arb *arbre, int k)`; qui teste si `k` est dans `arbre`

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 noeuds internes.

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 $2^k - 1$

Arbre et oracle

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

On se fixe une fonction $f : \mathcal{P}(S) \rightarrow \{\text{true}, \text{false}\}$ 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 où 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.

Accès dans un arbres 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-dimensionels

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-dimensionels

```

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 tout 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 tout les éléments 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-dimensionels 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 stockés dans l'arbre trié selon la première composante?

Arbres généraux

On considère les deux types d'arbres suivants en OCaml:

```

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

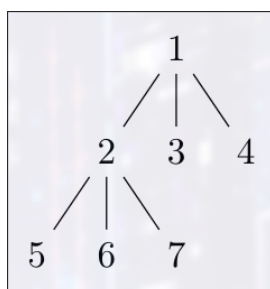
```

Les arbres `bintree` représentent des arbres binaires 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.

Pour transformer un arbre général en arbre binaire on utilisera la méthode *LCRS* (*Left child, right sibling*) décrite ci-dessous. Pour x un nœud, on note E_x la liste des enfants de x . Maintenant, pour x un nœud 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 $h(\text{to_bintree}(A)) \leq \Delta(A) + h(A)$

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 $w_1 \dots w_n$ un mot de A , montrer que pour tout $i < n$, on a que $|w_1 \dots w_i|_a \leq |w_1 \dots w_i|_b$. 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'arbre contenant T_u et T_v comme enfants. Proposer un algorithme pour effectuer cette transformation.

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

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 tableau T qui n'est **pas** un tas pour qu'il en devienne un.

Question 3 On suppose que pour tout 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 $T[i]$. Quel est la complexité de cet algorithme ?

Pour i un noeud, on note $h(i)$ sa hauteur, donné 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 $\Theta(n)$

Arbre canonique (ENS 2023 MPI Info A)

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 canonique 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.

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 ce 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 proposé pour représenter un arbre d'intervalles.

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

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 Définissez 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 Donnez `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.

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 Quel 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 neoud à 2 ou 0 enfants) alors on peut reconstruire T à partir de T_{pref} et T_{post}

Ranger des boites

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

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

On défini un type d'arbre de recherche de boites 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 boites 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 boite b dans l'arbre. Quel est sa complexité?

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

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 où ils diffèrent.

Question 3 Donner le code `let 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

Indépendant dans les arbres

On se donne en OCaml le type d'arbre suivant stockant des entiers sur chaque noeuds (que on appellera 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 que 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 `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 bintree * '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 se soucie pas de la condition d' α -équilibre

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 tout les éléments de t_2 et plus grand que tout 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 tout les éléments plus petits que x et tout 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`.

Comptage de sous-arbres⁴²

On considère dans ce sujet des arbres enracinés binaires (chaque nœud a au plus 2 enfants). Étant donné un arbre T , tout sous-ensemble S des arêtes de T définit une forêt, notée $T[S]$, obtenue en conservant les arêtes de T présentes dans S et en retirant les autres (on retire également les nœuds qui n'ont plus d'arêtes incidentes).

Question 1 Proposer un algorithme qui étant donné un arbre T compte le nombre de forêt différentes à isomorphisme près. On veillera à ce que

⁴²https://diplome.di.ens.fr/informatique-ens/Annales/2017_InfoU-exercices.pdf

MP2I: Graphes (introduction)

Cours

TODO

Petites questions

- Montrer que si A est un arbre, alors il existe toujours une feuille.

Autour des degrés

Soit $G = (S, A)$ un graphe. 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 avec N l'ensemble des noeuds qui ne sont pas des feuilles et f le nombre de feuilles. Montrer que $\sum_{x \in N} \deg x = f$. En déduire une relation dans le cadre des arbres binaire.

Longs Chemins

Soit G un graphe connexe. Montrer que deux chemins de longueur maximal s'intersectent.

Chemin hamiltonien dans un tournois

On dit que $G = (V, E)$ un graphe orienté est un tournois si pour tout $(u, v) \in V^2$, 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'adjascence est bien un tournois.

Un chemin s_1, \dots, 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'adjascence.

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 Un tel chemin hamiltonien est-il unique?

Le puits

Dans un graphe orienté $G = (V, E)$, on dit que $v \in V$ est un puits si pour tout $u \in V$, il y a une arête de u dans v (dans ce sens).

Question 1 Montrer que un tel puits est unique.

Question 2 On donne G par une matrice d'adjascence 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 tournoi, x est le roi si tout 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'adjascence et qui le calcule.

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

On dit qu'un graphe connexe $G = (V, E)$ est *biconnecté* si quelquesoit $x \in V$, 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 *arrete 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 arrete de retour de la forme (y, x) .
- Sinon, il existe une arrete 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.

Reconnaitre un DFS

On ce 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 etre obtenu en faisant un DFS de G .

Pour T un arbre enraciné de G , on note $x \preceq_T y$ si x est un enfant de y

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

Graphes d'Halin⁴³ 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 quelquesoit $u, v \in V$, $G - u - v$ le graphe obtenu en retirant u, v et leur arretes 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'Halin⁴⁴ 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 $\langle v_1, \dots, v_n \rangle$ passant par tout les sommets une fois

⁴³On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

⁴⁴On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

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

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

Calcul des triangles

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

1. Proposer un algorithme en $O(|V|^3)$
2. Soit L_1, L_2 deux listes triées d'entiers de 1 à n , montrer que l'on peut calculer la liste triée des éléments appartenant aux deux listes en temps $O(|L_1| + |L_2|)$.
3. Donner un algorithme en $O(|E| \times \Delta)$ pour calculer tout 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?

Implémentations incorrectes de Floyd-Warshall

Très pédagogique je pense, et assez sympa. TODO: <https://arxiv.org/abs/1904.01210>

MP2I: Logique Propositionnelle

Cours

- Définition des formules logiques par induction
- Valuation, valeur de vérité, table de vérité
- Forme normale conjonctive
- Forme normale disjonctive
- l'algorithme de quine

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 \perp , alors il existe une valuation μ telle que $\mu \models F$
2. Dans le cas où F est linéaire, proposer un algorithme polynomial qui compte le nombre de valuations satisfaisant F

Formules duales

Étant donné une formule de la logique propositionnelle F , utilisant les symboles $\top, \perp, \neg, \wedge, \vee$ 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 \perp .

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.

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 toute 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 sommet 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 toute valuation λ , $\lambda(\psi) = \top$ si et seulement si G_λ admet un centre.

XOR SAT

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

1. Exprimer l'opérateur \oplus en fonction des opérateurs classiques
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 satisfiable

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 où chaque clause possède au plus 3 littéraux. On aura le droit d'introduire de nouvelles variables non utilisées initialement.

Push down not

On ce 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.

Equivalences

On ce 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 $A \rightarrow B \equiv A \uparrow (B \uparrow B)$
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.

Retirer les constantes

On ce donne le type suivant en OCaml d'un arbre de syntaxe 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 \perp , le faux. Ce sont les deux “constantes”.

1. Donner une tautologie et une antilogie n'utilisant pas \top ou \perp .
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 $A \wedge \perp$? 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 \perp directement, mais ils ne devront pas apparaître dans des sous-formules.*

Compacité

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

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

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 satisfiable.

2. Montrer que si A est satisfiable alors il est finement satisfiable.
3. Dans le cas où l'ensemble des variables propositionnelles de A est dénombrable, démontrer en construisant par récurrence une valuation que si A est finement satisfiable alors A est satisfiable.
4. Utiliser le théorème de compacité pour montrer que un graphe infini dénombrable est N coloriable ssi tout ses sous-graphes fini sont N coloriable.

Indéterminé

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

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

Pour v une valuation et F une formule, on a:

- $v(F) = \top$ si quelque soit F' qui est F ou on a remplacé les $?$ par des valeurs \top ou \perp , on a $v(F') = \top$
- $v(F) = \perp$ si quelque soit F' qui est F ou on a remplacé les $?$ par des valeurs \top ou \perp , on a $v(F') = \perp$
- $v(F) = ?$ sinon

Ainsi, par exemple, on a $\top \vee ? \equiv \top$ et $\top \wedge ? \equiv ?$

1. Proposer des tables de vérité pour \wedge, \rightarrow et \neg
2. Montrer que $p \vee \neg p$ n'est pas une tautologie pour cette sémantique. Donner une tautologie.

3. Proposer un algorithme qui teste si une formule avec indéterminé est une tautologie.

Lemme d'interpolation de Craig

Soit F et B deux formules de la logique propositionnelle et X une variable propositionnelle, on note $F[X \leftarrow B]$ la formule où l'on a remplacé la variable propositionnelle X par B .

Par exemple, $(X \wedge Y \rightarrow X)[X \leftarrow \neg Y] = \neg Y \wedge Y \rightarrow \neg Y$

1. On pose $F_1 := F[X \leftarrow B]$ et $F_2 := F[X \leftarrow \neg B]$. Montrer que $F \rightarrow F_1 \vee F_2$ est une tautologie.

Soit F, G deux formules de la logique propositionnelle tel que $F \rightarrow G$ est une tautologie.

2. Montrer que si F n'a aucune variable propositionnelle en commun avec G , alors soit $\neg F$ soit F est une tautologie

3. Montrer que s'il existe au moins une variable propositionnelle en commun entre F et G , il existe une formule H tel que $F \rightarrow H$ et $H \rightarrow G$ sont des tautologies, et tel que toute variable propositionnelle de H apparait dans F et G

Equivalence

1. Expliciter une bijection φ entre $\llbracket 0; 2^n - 1 \rrbracket$ et $\{\top, \perp\}^n$

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 ?

MPI: Langages

- Définir un alphabet. Un mot. Un langage.
- Concaténation, union, intersection de langages
- Préfixe, suffixe, facteur, sous-mot

Petites questions

1. Lemme de Levy : Si $xy = x'y'$, alors il existe z tel que soit $x = x'z$ et $y' = zy$, soit $x' = xz$ et $y = zy'$.
2. Soient $u, v \in \Sigma^*$ tel que $uv = vu$. Montrer qu'il existe $w \in \Sigma^*$ et $p, q \in \mathbb{N}$ tel que $u = w^p$ et $v = w^q$.
3. 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 $v = w^q$.
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.

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 ou 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 $u[i \dots i + p - 1]$ et $v[j \dots j + p - 1]$ 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 $l < |w|$).

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 a 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 $(DB(n))_{n \in \mathbb{N}^*}$ par $DB(1) = (\{0, 1\}, \{0, 1\}^2)$ et $DB(n+1) = L(DB(n))$.

3. Construire $DB(2)$
4. Montrer que pour tout $n \in \mathbb{N}^*$, chaque sommet de $DB(n)$ a autant d'arêtes sortantes que entrantes. Combien de sommets et d'arêtes $DB(n)$ 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 $n \in \mathbb{N}^*$, $DB(n)$ possède un cycle eulérien.

- En voyant les sommets de $DB(n)$ 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 $2^n + n - 1$
- Généraliser la question précédente pour des alphabets plus grand.

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 $u = v$

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.

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

Equations de mots (oral info LCR ENS 2022)

Soit Σ un alphabet et x une *variable*. Une équation de mot est une équation de la forme

$$A_0 x A_1 \dots A_{n-1} x A_n = B_0 x B_1 \dots B_{m-1} x B_m \quad (1)$$

pour $A_0, A_1, \dots, A_n, B_0, B_1, \dots, B_m \in \Sigma^*$. Une solution $w \in \Sigma^*$ est un mot qui vérifie (1)

- Donner toutes les solutions de l'équation $abx = xba$ sur $\Sigma = \{a, b\}$

TODO: continuer l'exos

Ensemble inévitables (oral info Ulm ENS 2019)

On fixe un alphabet fini $\Sigma = \{a, b\}$. Pour $w \in \Sigma^*$, on écrit $w = w_1 \dots w_n$ 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 .

- Donner un mot de longueur au moins 12 qui évite $S = \{aaaa, aaab, aba, baaa, bab, bbbb\}$.
- 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.

- Est-ce que S est évitable ? et l'ensemble $\{aaa, abb, baa, abab\}$?
- 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 $l \in \llbracket 0; k \rrbracket$, on ai $w_{p+l} = w_{q+l}$
- Montrer que, pour tout ensemble inévitable $S \subseteq \Sigma^*$, il existe un sous-ensemble $S' \subseteq S$ fini tel que S' soit inévitable.

Points fixe de morphismes⁴⁵

On dit que $h : \Sigma^* \rightarrow \Sigma^*$ est un morphisme si $h(\varepsilon) = h(\varepsilon)$ et que pour tout $u, v \in \Sigma^*$ $h(uv) = h(u)h(v)$. On dit que h est positif si pour tout $\alpha \in \Sigma$, $|h(\alpha)| \geq 1$.

Soit h un morphisme positif et $\alpha \in \Sigma$ tel que α est un préfixe strict de $h(\alpha)$.

- Montrer que pour tout n , $h^{n(\alpha)}$ est un préfixe strict de $h^{n+1}(\alpha)$
- Montrer que qu'il existe un unique mot infini point fixe de h

⁴⁵Tiré de l'Info D 2026 de décembre

3. Soit u, v deux

Mots sans carré⁴⁶

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 tout les mots sans carré sur $\Sigma = \{a, b\}$

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

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 carré

4. En déduire l'existence d'un mot infini sans carré.

Mots sans cube

(***) 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 dit que u est sous-mot de v s'il existe $\varphi : \llbracket 0, |u| \rrbracket \rightarrow \llbracket 0, |v| \rrbracket$ strictement croissante tel que pour tout $i \in \llbracket 0, |u| \rrbracket$ on ai $u_{\varphi(i)} = v_i$. On note cela $v \succ u$.

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

⁴⁶Oral de l'ENS Info 2019 et sujet de CCINP d'Informatique 2026

MPI : Langages réguliers et Automates

Questions de Cours

- Rapeller la méthode pour obtenir un automate à partir d'une expression régulière.
- Montrer que l'intersection de deux langages réguliers est régulier.
- Montrer que l'union de deux langages réguliers est régulier.
- Montrer que le complémentaire d'un langage régulier est régulier.
- Montrer que si \mathcal{A} est un automate 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).

Berry-Sethi

Que vaut 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 :

- $\{w_1 w_1 w_2 w_2 \dots w_{|w|} w_{|w|} : w \in L\}$
- $\{w_1 w_3 w_5 \dots w_i : w \in L\}$, Une lettre sur deux des mots de L
- $\bar{L} = \{\bar{u} : u \in L\}$ le langage des miroirs, avec $\overline{u_1 \dots u_n} = u_n \dots u_1$
- $\bigcup_{w \in L} S(w)$ pour $S(w)$ l'ensemble des sous-mots de w
- $L \setminus L' = \{w \mid \exists x \in L', wx \in L\}$

Non régulier

Montrer que les langages suivant ne sont pas réguliers :

- $\{a^{2^n} : n \in \mathbb{N}\}$
- $\{a^{n^2} : n \in \mathbb{N}\}$
- $\{a^n b^m : n \leq m\}$
- $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$
- $\{a^p : p \in \mathbb{P}\}$ pour \mathbb{P} l'ensemble des nombres premiers.
- $\{a^n b^m : n \geq m\}$ (petite subtilité)

Petites questions

- Montrer 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 étiquettant un chemin de q à q' . Montrer que $L_{q, q'}$ est régulier.

Préfixes clos dans L

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

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

Langage croisé

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

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

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

1. Donner un langage régulier non vide L tel que $\psi(L)$ soit régulier.
2. Montrer que si L est régulier, alors \bar{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'automate⁴⁷

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

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

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 rotationnel

Soit L un langage sur un alphabet Σ . On défini $R(L) = \{xy : yx \in L\}$ le *rotationnel* 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 $L_{q_1, q_2} = \{w : \delta^*(q_1, w) = q_2\}$ (autrement dit, l'ensemble des mots étiquettant 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.

⁴⁷Colle de José

Mots univers

On dit qu'un mot $w \in \Sigma^*$ est n -univers si tout 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 quelconques 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 arêtes une et une seule fois)
4. Montrer qu'il existe un mot n -univers de longueur $|\Sigma|^n + n - 1$

Astucieux langage⁴⁸

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

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 $S(L) := \{n \in \mathbb{N} : a^n \in L\}$. 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 $L_{ab} = \{a^n b^n : n \in \mathbb{N}\}$. Montrer que si L est un langage tel que $L_{ab} \subseteq L \subseteq \sigma(L_{ab})$, 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 partiellement ordonnés⁴⁹

Un automate (déterministe ou non) \mathcal{A} est dit partiellement ordonné si tout 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?

⁴⁸Le fabuleux cours de théorie des langages formels d'Olivier Carton

⁴⁹Oral ENS 2022 C2 https://diplome.di.ens.fr/informatique-ens/Annales/2022_InfoU-exercices.pdf

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.

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

On considère un automate \mathcal{A} pas forcément déterministe munit d'une fonction de cout $c : \delta \rightarrow \mathbb{R}$ 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 défini.

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 défini 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 $c : \delta \rightarrow \mathbb{R}^+$ est positive, alors pour tout C , $L(\mathcal{A})_C$ est un langage régulier.
4. Toujours si $c : \delta \rightarrow \mathbb{R}^+$, 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^* ?

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 :

$$\llbracket L_1; L_2 \rrbracket := \{xx'yy' : xy \in L_1 \wedge x'y' \in L_2\}$$

1. Montrer que $\llbracket L_1; L_2 \rrbracket$ 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'êtes pas obligé de calculer $\llbracket L_1; L_2 \rrbracket$
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 étiquettant un chemin de q à q' . Montrer que $L_{q,q'}$ est régulier.
3. Montrer que si L_1, L_2 sont réguliers, alors $\llbracket L_1; L_2 \rrbracket$ l'est aussi.

Commutant de Langage⁵⁰

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

1. Décrire $\text{Comm}(abab)$
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 $\text{Comm}(u)$ est toujours régulier pour $u \in \Sigma^*$.

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

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

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

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 $\text{Comm}(L)$ est régulier.

Langage coupé au milieu

Soit Σ un alphabet et L un langage, on pose

⁵⁰Sujet de José

$$\text{TM}(L) := \{w \in \Sigma^* \mid \exists u, v, uvw \in L \wedge |u| = |v| = |w|\}$$

$$\overline{\text{TM}}(L) := \{uvw \in \Sigma^* \mid w \in L \wedge |u| = |v| = |w|\}$$

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

Question 3 Montrer qu'il existe un langage régulier L tel que $\overline{\text{TM}}(L)$ 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 étiquettant un chemin de q à q' . Montrer que $L_{q,q'}$ est régulier.

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

Langages fins

Soit L un langage, on définit sa fonction de densité $\delta_L : \mathbb{N} \rightarrow \mathbb{N}$ 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 caractériser les langages fins.

1. Montrer que pour tout $k \in \mathbb{N}, u_1, \dots, u_k, v_1, \dots, v_k, w_1, \dots, w_k \in \Sigma^*$, 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 $q_{\text{init}} \xrightarrow{a_1} \dots \xrightarrow{a_n} q_f \in F$ 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, \dots, w_k tel que L à la forme attendue.

4. Proposer un algorithme qui prend en entrée un automate et qui indique si \mathcal{A} reconnaît un langage fin. Quel est sa complexité?

Langages épars⁵¹

Soit L un langage, on définit sa fonction de densité $\delta_L : \mathbb{N} \rightarrow \mathbb{N}$ 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 langage épars ou non.

Langages rationnel 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.

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 langage⁵²

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

⁵¹ENS Ulm oral d'info, je ne sais plus de quand

⁵²Oral de l'X

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

Cloture par sur-mot⁵³

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 $\varphi : \llbracket 1, \dots, |w| \rrbracket \rightarrow \llbracket 1, \dots, |u| \rrbracket$ strictement croissante telle que

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

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

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

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 admettant le lemme d'Higman, montrer que pour tout langage L , il existe un langage 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 caractériser 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 étiquetage), 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 reconnus 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 continuable 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.

⁵³ENS Ulm de je ne sais plus quand

MPI: Langages Hors-contexte

Cours

- Grammaire, grammaire hors-contexte
- Grammaires générales, contextuelles (HP)
- Arbre de dérivation
- Dérivation à gauche, à droite
-

Petites questions

Langage hors-contexte

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

- $\{w_1w_1w_2w_2\dots w_{|w|}w_{|w|} : w \in L\}$
- $\overline{L} = \{\overline{u} : u \in L\}$ le langage des miroirs, avec $\overline{u_1\dots u_n} = u_n\dots u_1$
- $\bigcup_{w \in L} S(w)$ pour $S(w)$ l'ensemble des sous-mots de w
- $\{w_1w_3w_5\dots w_i : w \in L\}$, 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 $L_1 = L_aL_b \cup L_bL_a$. 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 $f : \Sigma^* \rightarrow \mathbb{Z}$ tel que $f(w) = |w|_a - |w|_b$. On considère les langages suivants:

$$L = \{w \in \Sigma^* \mid |w|_a = |w|_b\} = f^{-1}(\{0\})$$

$$L_+ \text{ engendré par } S \rightarrow SS \mid aSb \mid \varepsilon$$

$$L_- \text{ engendré par } S \rightarrow SS \mid bSa \mid \varepsilon$$

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 ambiguë? 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

MPI: Graphes avancé

Cours

- Chemin alternant.
- Clique et graphe bi-parti. Biparti ssi pas de cycle impair.
- Couplage maximal, maximum, parfait.
- Algo couplage max dans biparti
- Arbre couvrant, Kruksal
- Tri topologique, Kosaraju
- approximation, 2-approx du TSP
- Sous-graphe, graphe induit (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.

Petites questions

- Soit $G = (V, E)$ un graphe connexe, montrer qu'il existe un sommet $v \in V$ tel que $G - v$ soit connexe.

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 contiens 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).

2-approximation du circuit eulérien (TSP)

Soit $G = (V, E)$ un graphe non orienté pondéré par $w : E \rightarrow \mathbb{R}_+$. 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 défini $w(T)$ le poid d'un arbre T sur G comme la somme du poid de toutes ses arretes. Un circuit est un cycle ou l'on a le droit de repasser sur des sommets déjà vu.

On cherche un circuit passant par tout les sommets au moins une fois et pouvant repasser sur lui-meme tel que le poid de tout le chemin soit minimal. On note w^* le poid du circuit de poid minimal.

Question 1 On suppose que tout 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 poid minimal, alors $w(T) < w^*$

Question 3 En déduire via un parcours de cet arbre un circuit repassant sur lui-meme 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 $d : V^2 \rightarrow \mathbb{R}_+$ une 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 impairs de T^* . On note w^{OPT} le poids optimal d'une instance du problème.

Question 1 Montrer que $w(T^*) < w^{\text{OPT}}$. 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' restreint 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 arêtes). En déduire que $T^* + M$ 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.

$(\Delta + 1)$ -approx de Maximum independent 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 $\bar{L} = \{\bar{w} : w \in L\}$ (l'ensemble des miroirs 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 choisit et retire toutes les arêtes reliées 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

$$\mathbb{E}[|V_f|] = \sum_{v \in V} \frac{\deg v}{\deg v + 1}$$

Kruskal en recherche locale

Définition Soit $G = (V, E)$ un graphe connexe pondéré par $w : E \rightarrow \mathbb{R}_+$, 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 quelconque 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 $\lfloor \frac{2n+2}{3} \rfloor$

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

$$|V| \leq \frac{2}{\delta - 2} ((\delta - 1)^{\frac{m}{2}} - 1)$$

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* $f : V \rightarrow \llbracket 1; |V| \rrbracket$ bijective, on définit la *largeur de f dans G* comme

$$\psi(f, G) := \max\{|f(u) - f(v)| : \{u, v\} \in E\}$$

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 vaut 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

Def Un graphe $G = (V, E)$ est *planaire* s'il existe $p : V \rightarrow \mathbb{R}^2$ 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 tout les sommets de 1 à a soient reliés à tout les sommets de $a + 1$ à b (sans aucune autre arête).

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 tout 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 possible: Montrer que si $G = (V, E)$ est un graphe planaire à plus de 9 sommets, alors $\overline{G} = (V, \overline{E})$ n'est pas planaire.

Couverture d'arêtes et couplages

Soit $G = (V, E)$ un graphe connexe, on dit que $A \subseteq E$ est une *couverture d'arête* 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 où il n'existe pas de chemin de longueur strictement plus grande que 2. En déduire que A est une union disjointe de composantes connexes qui sont des graphes étoile.

Question 4 Montrer que $|A| + |M| \geq |V|$

Question 5 Montrer que $|A| + |M| = |V|$. *Indication: On montre 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 maximum⁵⁴

Pour M un couplage, on note $S(M)$ les sommets saturés 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 $u', v' \notin S(M)$:
 $M \leftarrow (M \setminus \{u, v\}) \cup \{\{u', u\}, \{v, v'\}\}$
fin tant que
retourner M

1. Quel 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 $M \leftarrow (M \setminus P) \cup (P \setminus M)$. 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 triangles⁵⁵

Question 1 Montrer qu'un graphe est biparti si et seulement si il n'admet pas de cycle impair.

⁵⁴TD11 L3 ENS Lyon Algo 1

⁵⁵Tiré de Mallory Marin

Définition Pour $G = (V, E)$ un graphe non orienté avec $V = \{1, \dots, n\}$, on dit que $\{x, y, z\} \subseteq V$ est un *triangle* si $\{x, y\}, \{y, z\}, \{z, x\} \in E$. On cherche à calculer tout 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 triées d'entiers de 1 à n , montrer que l'on peut calculer la liste triée des éléments appartenant aux deux listes en temps $O(|L_1| + |L_2|)$.

Question 4 Donner un algorithme en $O(|E| \times \Delta)$ pour calculer tout 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?

Graphes d'amis

Soit $G = (S, A)$ un graphe non orienté tel que pour toute paire $(s_1, s_2) \in S^2$, il existe un **unique** $s' \in S$ tel que (s_1, s') et $(s', s_2) \in E$. Montrer qu'il existe un sommet relié à tout les autres.

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 degré max du graphe ?

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 $\varphi : V \rightarrow \mathbb{R}^2$ 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 définit 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.

Un Graphe dénombrable

On prend $\varphi : \mathbb{N} \rightarrow \mathcal{P}_f(\mathbb{N})$ une bijection de \mathbb{N} dans les parties finies de \mathbb{N} . Montrer qu'à 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 -dégénéré⁵⁶

Un graphe $G = (V_G, E_G)$ est dit d -dégénéré s'il existe un ordre v_1, \dots, v_n des sommets de G tel que, pour tout $i \leq n$, le sommet v_i au plus d voisins parmi $\{v_1, \dots, v_{i-1}\}$. La dégénérescence 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 -dégénéré 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 $\leq d$

⁵⁶InfoFonda 2026 partie I

Question 4 Donner un algo polynomial qui prend un graphe d -dégénéré et renvoie $(d + 1)$ -coloration

Coeur de k -domination⁵⁷

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 $\forall x \in V_G, |N(x) \cap X| < \lfloor |X|/k \rfloor$. Montrer que G n'admet pas d'ensemble quasi-dominant de X de taille au plus k .

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 $\exists v \notin S, |N(v) \cap Y| \geq \lfloor |Y|/k \rfloor$ **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 vizig⁵⁸

Soit $G = (V, E)$ un graphe, on note Δ le degré maximal d'un graphe, on dit que $\gamma : E \rightarrow \{1, \dots, k\}$ est une k -coloration d'arêtes 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 $(\Delta + 1)$ -colorable en arête.

Question 1 Montrer que tout graphe est $2\Delta - 1$ -arête colorable

Question 2 Montrer qu'un graphe ne peut pas être k -coloré pour $k < \Delta$.

Pour γ une coloration on note $\bar{\gamma}(u)$ l'ensemble des couleurs des arêtes d'extrémité u qui **ne** sont **pas** prise. Soient a, b deux couleurs, on définit $K(a/b)$ la **chaîne de Kempe** comme étant un ensemble d'arêtes maximal pour l'inclusion contenant que des arêtes 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.

⁵⁷InfoFonda 2026 partie IV

⁵⁸InfoFonda 2026 partie V

Question 4 Soit G biparti, γ une coloration partielle, $(u, v) \in E_G$ avec $a \in \bar{\gamma}(u)$ et $b \in \bar{\gamma}(v)$. 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 colorée d'une coloration partielle γ et soient a, b deux couleurs telle que $a, b \in \bar{\gamma}(u)$ et $b \in \bar{\gamma}(v)$. Montrer qu'il existe γ' qui colorie les mêmes arêtes telle que

- $\bar{\gamma}'(u) = \bar{\gamma}(u)$,
- $a \in \bar{\gamma}'(v)$ et
- $\bar{\gamma}'(x) = \bar{\gamma}(x)$ pour tout $x \in V \setminus \{u, v\}$ sauf au plus un qui vérifie alors $(\bar{\gamma}'(x) \setminus \bar{\gamma}(x)) \cup (\bar{\gamma}(x) \setminus \bar{\gamma}'(x)) = \{a, b\}$

Question 6 Soit γ une k -coloration partielle de G dont toutes les arêtes non colorées e_1, \dots, e_r sont toutes de la forme $e_i = (u, v_i)$. On suppose que $|\bar{\gamma}(u)| \geq r$, $|\bar{\gamma}(v_1) \cap \bar{\gamma}(u)| \geq 1$ et que pour tout $i \geq 2$, on a $|\bar{\gamma}(v_i) \cap \bar{\gamma}(u)| \geq 2$. 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 -arête 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 $(G - u + uu', k)$ 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 adjacentes 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 correct 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 $S_1, \dots, S_k \subseteq V_G$ 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

- Calcul des attracteurs. Définition des A_i^j dans l'algorithme du calcul des attracteurs.
- Stratégie gagnante, jeux à deux joueurs
- Zermelo (HP)

Petites questions

- Soit un jeu $G = (V, E)$ avec $V = \mathbb{N}$ et telle que les arêtes orientées 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.

Jeu de nim généralisé

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 $\{1, \dots, p\}$?
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 gagnante, alors il n'existe pas $N_0, T > 0, \forall n > N_0, n \in S_A \Leftrightarrow n + T \in S_A$.

Géographie dans les hypercubes⁵⁹

Soit $\Sigma = \{a, b, c\}$ et $n \in \mathbb{N}^*$. Alice et Bob jouent au jeu suivant : à tour de rôle, ils 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 choisit un mot quelconque au début. Le premier joueur ne pouvant pas jouer perd. On cherche à établir une stratégie gagnante.

On pose $w_a = \underbrace{a \dots a}_{n \text{ fois}}$

1. Dessiner le graphe des états pour $n = 1$ et $n = 2$. Qui gagne ?
2. Montrer que un joueur X possède une stratégie gagnante ssi et seulement si il en possède une pour le jeu où Alice commence par le mot w_a .

On supposera maintenant sans perte de généralité que le premier coup d'Alice est $a \dots a$. On pose $G_n = (V_n, E_n)$ le graphe défini par $V_n = \Sigma^n$ et $(u, v) \in E$ si u et v ne diffèrent que d'une lettre.

⁵⁹Le putnam de 2025

3. Montrer par récurrence sur $n \in \mathbb{N}^*$ que $G_n - w_a$ possède un couplage parfait
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

Géographie⁶⁰

Soit $G = (V, E)$ un graphe, on définit le jeu à deux joueurs (A)lice et (B)ob tel que à tour de rôle, 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 où G est un arbre pour savoir qui possède une stratégie gagnante.
5. On considère le cas où $V \subseteq \mathbb{N}^2$ 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?

Jeu de Shannon

Définition 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_2 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 $\|\mathcal{P}\| < 2(|\mathcal{P}| - 1)$. 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 couvrants 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.

Jeu du sous-mot

Soit Σ un alphabet, on considère le jeu à deux joueurs (dénote (A)lice et (B)ob) du sous-mot suivant: à tour de rôle, 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 peut plus dire de mot perd. Par exemple, sur $\Sigma = \{a, b\}$, la partie pourrait être $abb \rightarrow bbbb \rightarrow aab \rightarrow a \rightarrow bbb \rightarrow bb \rightarrow b$ et Alice perd (Alice étant la joueuse qui commence).

On admettra que ce jeu termine toujours (cette propriété vient du lemme d'Higman).

⁶⁰Marathon de math de Paris-Orsay 2021

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 $\bar{a} := b$ et $\bar{b} := a$, puis $\overline{w_1 \dots w_n} = \overline{w_1} \dots \overline{w_n}$. 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

Questions de Cours

- Expliquer l'algorithme des k -moyennes.
- Rapeller le fonctionnement de l'algorithme ID3

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?

Single-pass (CCINP 2024)⁶¹

On considère l'algorithme suivant pour catégoriser des données $X = \{\vec{x}_1, \dots, \vec{x}_n\} \subseteq \mathbb{R}^d$. On note $\delta(\vec{x}, \vec{y}) = \sum_{i=1}^d |\vec{x}_i - \vec{y}_i|$ 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 $\text{Cl}(\vec{x}, C)$ la classe dont le centre est le plus proche de \vec{x} pour δ .

Soit $\theta \in \mathbb{R}_+$, on considère :

```
C ← ∅
Pour i de 1 à n:
  Si ∀c̄ ∈ C, δ(x̄_i, c̄) > θ:
    C ← C ∪ {{x̄_i}} // On crée une nouvelle classe
  Sinon:
    C̄ ← Cl(x̄_i, C)
    C' ← C̄ ∪ {x̄_i}
    C ← C \ {C̄} ∪ {C'}
Renvoyer C
```

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 $\vec{x}_1, \dots, \vec{x}_n$ importe.

Question 3 Soit $\vec{x} \in X$, on note C_{ini} la classe de \vec{x} au moment où 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 admettra que $\frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq \ln(k)$

Question 4 Quel sont les avantages de cet algorithme par rapport aux k -moyennes? Proposer des améliorations à l'algorithme.

Clustering en dimension 1⁶²

Définition Soit $K \in \mathbb{N}^*$ et $E = \{x_0, \dots, x_{N-1}\}$ un ensemble de réels avec $x_0 \leq \dots \leq x_{N-1}$. Pour $X \subseteq E$, on note μ_X son centre et $S(X)$ son score, formellement défini par

⁶¹<https://prepas-mp2i.fr/documents/sujets/2024/CCINP-INFO.pdf>

⁶²tiré de https://mpi-lamartin.github.io/mpi-info/docs/ia/non_supervise/td_apprentissage

$$\mu_X := \frac{1}{|X|} \sum_{x \in X} x \quad S(X) := \sum_{x \in X} (x - \mu_X)^2$$

On pose $S(i, j) = S(\{x_i, \dots, x_j\})$. On cherche à résoudre l'algorithme des k -moyennes sur E de manière exacte. C'est à dire que l'on cherche une partition $\mathcal{P} = \{X_1, \dots, X_K\}$ de $\llbracket 0; N-1 \rrbracket$ 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 $\{x_0, \dots, x_n\}$ en k classes. Que vaut $I(n, 1)$?

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 partition 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: $v_1, \dots, v_n \subseteq \mathbb{R}^d$ une liste de points finis
Sortie: $X := \{x_1, \dots, x_k\} \subseteq V$ 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, 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, \dots, p_k les k points retourné par notre algorithme et p_1^*, \dots, 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 $\forall i, V_i \cap \{p_1, \dots, p_n\} \neq \emptyset$ 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 $S \subseteq A_1 \times \dots \times A_k \times C$ un ensemble fini d'apprentissage, et pour A_i de la forme $A_i = [\alpha, \beta]$, pour $k \in]\alpha, \beta[$, on défini $S_{<k}, S_{\geq k}$ 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 a 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 Quel 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

Lecteurs-Rédacteurs

On se place dans le cadre d'une base de données avec 2 types d'utilisateurs: les lecteurs et les rédacteurs. La base de données a ces deux contraintes:

- Plusieurs lecteurs doivent pouvoir lire la base de données en même temps ;
- Si un rédacteur accède à la base de données 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 a une potentielle 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 parallèles

On se propose d'implémenter un système de vote entre plusieurs fils. Chacun des k fils d'exécution exécutera la fonction `void vote(int valeur)` que l'on se propose d'implémenter. Une fois que tous les fils ont voté, **tout les fils** devront exécuter 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 tout les gagnant).

Par exemple, pour $k = 5$, si les 5 fils d'exécution exécutent `vote(2)`, `vote(4)`, `vote(3)`, `vote(4)`, `vote(3)` en parallèle, alors on doit s'attendre à ce que `resultat(3)` soit appelé par les 5 fils.

Question 1 Dans le cas où 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 où les votes sont dans $\llbracket 0; p \rrbracket$. 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 tableau d'entier partagé, mais aucun mutex ni 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 se 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 donnée qui est une généralisation pondéré des sémaphores. Elle aura trois opérations :

- `init(int capacite)` qui initialise et crée notre sémaphore pondéré avec `capacite` ressources
- `prendre(int k)` qui bloque le fil jusqu'à ce que l'on puisse allouer 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 garrantissant l'absence de famine utilisant un sémaphore, et une variable protégé par un mutex.

Question 4 Montrer l'absence 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 fil k peut se lock autant de fois qu'il veut. Notamment, cela permet d'exécuter ce code sans situation de blocage :

```
void example() {
    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 sémaphores: si un fil d'exécution exécute `p(semaphore)` et obtiens la ressource, alors il peut appeler `p(semaphore)` autant de fois qu'il veut sans être mis en attente ni faire décroître le compteur interne, et ce jusqu'à ce que le fil appelle `v(semaphore)`.

Question 2 Proposer une fonction qui, si elle est exécutée par deux fils, peut donner lieu à 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 tableau de booléens et un sémaphore classique.

Pont à voie unique

On cherche ici à modéliser un pont à voie unique qui ne peut être emprunté que par des voitures allant dans le même sens. On représentera cela par deux types de fils d'exécution : les fils ascendants et les fils descendants. 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 veut 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 absence de famine avec 2 sémaphores initialisés 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 présentes en même temps sur le pont. Modifier le code pour correspondre à ce critère.

Question 4 Et s'il le pont possédait k voies, tel qu'une seule ne peut être empruntée en même temps?

MPI: Algo avancé

Cours

- Union-find
- k -approx
- Algo probabiliste

Question de Cours

- Qu'est-ce qu'un algorithme de Monte-Carlo et de Las Vegas ?

Petites questions

- On choisit uniformément des entiers dans $\{1, \dots, n\}$ avec remise jusqu'à ce que tous les entiers ont été choisis. On note X la variable aléatoire du temps d'exécution. Montrer que $\mathbb{E}[X] = n \ln n + o(n \ln n)$
- Quel 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;  
}
```

C -approx du couplages maximum⁶³

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 $u', v' \notin S(M)$:
 $M \leftarrow (M \setminus \{u, v\}) \cup \{\{u', u\}, \{v, v'\}\}$
fin tant que
retourner M

1. Quel 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 $M \leftarrow (M \setminus P) \cup (P \setminus M)$. 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 .

Set cover

On considère le problème SET-COVER suivant :

Entrée: $n, m \in \mathbb{N}, U_1, \dots, U_n \subseteq \llbracket 1; m \rrbracket$
Sortie: Le plus petit I en cardinal tel que $\bigcup_{i \in I} U_i = \llbracket 1; m \rrbracket$

⁶³TD11 L3 ENS Lyon Algo 1

Dans toute cette colle on supposera que toutes les entrées sont telles que $\bigcup_{i \leq n} U_i = \llbracket 1; m \rrbracket$

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.

On considère maintenant que la sortie doit être un I qui maximise le nombre de $x \in \llbracket 1; m \rrbracket$ 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 choisit avec probabilité $\frac{1}{2}$ chaque U_i . On note N la variable aléatoire du nombre de $x \in \llbracket 1; m \rrbracket$ appartenant à un nombre impair d'ensembles choisis.

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 ?

Unique graphe infini aléatoire

On dit que deux graphes $G = (V, E)$ et $G' = (V', E')$ sont isomorphes s'il existe $\varphi : V \rightarrow V'$ bijective tel que $(u, v) \in E \Leftrightarrow (\varphi(u), \varphi(v)) \in E'$

On cherche à montrer que les graphes aléatoires infinis dénombrables sont isomorphes avec probabilité 1. On considère pour cela les variables aléatoires de Bernoulli $(X_{u,v})_{u,v \in \mathbb{N}}$ indépendantes de probabilité p qui représentent s'il existe une arête entre u et v . On note G_p cette famille de variables aléatoires.

Question 1 Quel est la probabilité que le graphe possède le chemin $(0, 1, 2, \dots, k)$?

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 à aucun V .

Question 2 Soit $U, V \subseteq \mathbb{N}$ fini, montrer que la probabilité que un $x \in \mathbb{N} \setminus (U \cup V)$ satisfait 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' possèdent la propriété de Rado, alors ils sont isomorphes.

Question 4 Généraliser la question 3 pour montrer que G_p est isomorphe à $G_{p'}$ pour $0 < p, p' < 1$ avec probabilité 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 $\mu : V \rightarrow \{1, \dots, k\}$. 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 $O(2^k \times |E|)$.

Question 3 En déduire un algorithme à erreur unilatérale de même probabilité que celle obtenue question 1. Quel type d'algorithme est-ce ?

Question 4 En déduire un algorithme à erreur unilatérale de probabilité $\frac{1}{2}$. Quel est son temps d'exécution?

Complexité en \log^* de Union Find⁶⁴

TODO

Complexité en α de Union-find⁶⁵

TODO

⁶⁴Colle de José

⁶⁵Algo 1, TD 4 ENS Lyon

MPI: Dédution naturelle

Cours

- Règle d'introduction et d'élimination du et
- Règle d'introduction et d'élimination du ou
- Règle d'introduction et d'élimination du non
- Règle d'introduction et d'élimination du implication
- Règle d'introduction et d'élimination du pour tout
- Règle d'introduction et d'élimination du il existe

Arbres simples en prop

Montrer les séquent suivant en logique minimale :

- $(p \wedge (p \rightarrow q)) \vdash q$
- $A \wedge B \rightarrow C \vdash A \rightarrow B \rightarrow C$ (la currification)
- $A \rightarrow B \rightarrow C \vdash A \wedge B \rightarrow C$ (la co-currification)
- $p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$ (le syllogisme de barbara)
- $\vdash p \wedge q \rightarrow q \wedge p$ (la commutativité du \wedge)
- $\vdash p \vee q \rightarrow q \vee p$ (la commutativité du \vee)
- $\vdash A \rightarrow B \vdash \neg B \rightarrow \neg A$ (le raisonnement par contraposé)
- $\vdash \neg\neg\neg A \rightarrow \neg A$ (meme si $\neg\neg A \rightarrow A$ n'est pas démontrable, ce séquent l'est !)
- $p \rightarrow q \vdash \neg(p \wedge \neg q)$ (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:

- $\forall x, \neg A \vdash \neg \exists x, A$ où $x \in VL(A)$
- $\exists x, \neg A \vdash \neg \forall x, A$ où $x \in VL(A)$

TODO: les sens inverses, quel logique il faut?

Equivalence entre les formules donnant la logique classique

Montrer que en logique intuitioniste, nous avons les jugements suivants :

- $A \vee \neg A \vdash ((A \rightarrow B) \rightarrow A) \rightarrow A$ (Tiers-exclu vers Peirce)
- $((A \rightarrow B) \rightarrow A) \rightarrow A \vdash A \vee \neg A$ (Peirce vers Tiers-exclu)
- $A \vee \neg A \vdash \neg\neg A \rightarrow A$ (Tiers-exclu vers élimination du double non)
- $\neg\neg A \rightarrow A \vdash A \vee \neg A$ (élimination du double non vers Tiers-exclu)

Arbres nécessitant de deviner

On définit $a \leftrightarrow b$ par $a \rightarrow b \wedge b \rightarrow a$.

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 logique propositionnelles l'opérateur \oplus (prononcé XOR) d'arité 2, tel que pour η une valuation, on a $\eta \models \varphi \oplus \psi$ si et seulement si $\eta \models \varphi$ et $\eta \not\models \psi$ ou bien $\eta \not\models \varphi$ et $\eta \models \psi$.

1. Proposer 2 règles 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 logique propositionnelles l'opérateur d'équivalence \leftrightarrow d'arité 2 avec les règles suivantes :

$$\frac{\Gamma, A \vdash B \quad \Gamma, B \vdash A}{\Gamma \vdash A \leftrightarrow B} \leftrightarrow_i \qquad \frac{\Gamma \vdash A \leftrightarrow B}{\Gamma \vdash A \rightarrow B} \leftrightarrow_l \qquad \frac{\Gamma \vdash A \leftrightarrow B}{\Gamma \vdash B \rightarrow A} \leftrightarrow_r$$

Soit μ une valuation, proposer une définition de $\mu \models A \leftrightarrow B$. 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 logique propositionnelles l'opérateur de Sheffer $A \uparrow B$ tel que $A \uparrow B \equiv \neg(A \wedge B)$.

1. Montrer que $A \rightarrow B \equiv A \uparrow (B \uparrow B)$
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 \perp .
4. Montrer que si $\vdash F$ est prouvable alors $\vdash F^*$ l'est aussi.

Sans implication

Soit F une formule propositionnelle, on défini par induction $\psi(F)$ la transformé sans implications de F par

$$\begin{aligned} \psi(A \vee B) &:= \psi(A) \vee \psi(B) & \psi(X) &:= X & \text{pour } X \text{ une variable} \\ \psi(A \wedge B) &:= \psi(A) \wedge \psi(B) & \psi(\neg A) &:= \neg\psi(A) \\ \psi(A \rightarrow B) &:= \neg\psi(A) \vee \psi(B) & \psi(\top) &:= \top, \psi(\perp) := \perp \end{aligned}$$

Montrer que $\vdash F$ est prouvable si et seulement si $\vdash \psi(F)$ est prouvable

Complétude⁶⁶

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

$$|\varphi|_\mu = \begin{cases} \varphi & \text{si } \mu \models \varphi \\ \neg\varphi & \text{sinon} \end{cases}$$

⁶⁶Le magnifique livre de Mme. Galatée Hemery !!!! <3

1. Soit F une formule et $\{X_1, \dots, X_n\}$ les variables propositionnelles de F . Montrer que le séquent suivant est démontrable en logique classique pour toute valuation μ :

$$|X_1|_\mu, \dots, |X_n|_\mu \vdash |A|_\mu$$

2. Montrer que en logique classique, si $\Gamma, X \vdash \varphi$ et $\Gamma, \neg X \vdash \varphi$ sont prouvables, alors $\Gamma \vdash \varphi$ l'est aussi.
3. Montrer que si A est une tautologie, alors pour toute valuation μ , on a $|X_1|_\mu, \dots, |X_n|_\mu \vdash A$ démontrable.
4. Conclure.

Sans négation

Soit F une formule propositionnelle, on définit par induction $\psi(F)$ la *transformé sans négation de F* par

$$\begin{aligned} \psi(A \vee B) &= \psi(A) \vee \psi(B) & \psi(X) &= X & \text{pour } X \text{ une variable} \\ \psi(A \wedge B) &= \psi(A) \wedge \psi(B) & \psi(\neg A) &= \psi(A) \rightarrow \perp \\ \psi(A \rightarrow B) &= \psi(A) \rightarrow \psi(B) & \psi(\top) &= \top, \psi(\perp) = \perp \end{aligned}$$

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 règles suivantes, indiquant que R est symétrique et n'est pas réflexif:

$$\frac{\Gamma \vdash R(x, y)}{\Gamma \vdash R(y, x)} \text{sym} \qquad \frac{}{\Gamma \vdash \neg R(x, x)} \text{nr}$$

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 tout 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 tout 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:

$$\begin{aligned} (A \vee B)^\perp &:= A^\perp \wedge B^\perp & X^\perp &:= \neg X & \text{pour } X \text{ une variable} \\ (A \wedge B)^\perp &:= A^\perp \vee B^\perp & (\neg A)^\perp &:= \neg A^\perp \\ (A \rightarrow B)^\perp &:= \neg A^\perp \vee B^\perp \end{aligned}$$

1. Montrer que si μ est une valuation, alors $\mu \models \neg F$ si et seulement si $\mu \models F^\perp$
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*.

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
- $\varphi \otimes \psi, \varphi \wp \psi$ sont des formules de la logique linéaire 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 munit ces formules des règles de déduction suivantes:

$$\frac{}{\vdash A, \overline{A}} \text{ax} \quad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \otimes \quad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \wp B} \wp$$

Attention, dans ces définition Γ n'est pas un ensemble mais une liste d'hypothèses (que l'on peut réordonnée). La multiplicité du nombre d'occurrence des formules compte, et la séparation Γ, Δ dans \otimes doit être une partition des hypothèses.

1. Montrer que le séquent $\vdash (A \otimes A) \wp (\overline{A} \wp \overline{A})$ est prouvable en logique linéaire multiplicative.

Soit $F \in \mathcal{L}$, on définit par induction le dual de F , noté F^\perp par $(X \otimes Y)^\perp = X^\perp \wp Y^\perp$, $(X \wp Y)^\perp = X^\perp \otimes Y^\perp$, $X^\perp = \overline{X}$ 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 supposant qu'il existe une preuve de toute ses prémisses, 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 Hilbert⁶⁷

(sujet à tester)

Si F est une formule, on dit que G est une sous-formule de F (noté $G \prec F$) si on peut remplacer des variables propositionnelles de F en d'autres formules pour obtenir G . Par exemple, la formule $(X \wedge Y) \rightarrow (X \wedge Y) \wedge (X \rightarrow Y)$ est une sous-formule de $A \rightarrow A \wedge B$ obtenu en remplaçant A par $X \wedge Y$ et B par $X \rightarrow Y$. On ne peut pas remplacer la même variable par deux formules différentes.

Pour R un ensemble de formules, on considère le système logique à la Hilbert sur R (on le note avec \triangleright_H) possédant les 2 règles suivantes:

$$\frac{}{\triangleright_R F} \text{Ax} \quad \text{pour } F \prec G, G \in R$$

$$\frac{\triangleright_R A \quad \triangleright_R A \rightarrow B}{\triangleright_R B} \rightarrow_e$$

Remarquer que comme R ne change jamais dans les règles, \triangleright_R est une sorte de système ou les axiomes sont les formules R avec uniquement la règle \rightarrow_e

⁶⁷Partie 4 du projet de rocq du cours PRFA 2025/2026

On fixe $H = \{A \rightarrow B \rightarrow A, (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)\}$. On souhaite montrer que $\triangleright_H X$ si et seulement si X est prouvable en logique minimale seulement avec les règles $\rightarrow_i, \rightarrow_e, Ax$.

1. Montrer le séquent $\triangleright_H (A \rightarrow B) \rightarrow (A \rightarrow A)$ est prouvable.
2. Montrer que si $\triangleright_X F$ est prouvable et que on a $F' \prec F$ et $X \subseteq X'$, alors $\triangleright_{X'} F'$ l'est aussi. En déduire que $\triangleright_H A \rightarrow A$ 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 $\triangleright_\Delta F$ est prouvable, alors $\Gamma \vdash F$ est prouvable en logique minimale seulement avec les règles $\rightarrow_i, \rightarrow_e, Ax$
4. Montrer les 2 faits suivants:
 - Si $\triangleright_\Delta B$ est prouvable alors $\triangleright_\Delta A \rightarrow B$ est prouvable
 - Si $\triangleright_\Delta A \rightarrow B \rightarrow C$ et $\triangleright_\Delta A \rightarrow B$ sont prouvable alors $\triangleright_\Delta A \rightarrow C$ est prouvable
5. Montrer que si $\triangleright_{\Delta \cup \{F\}} G$ est prouvable alors $\triangleright_\Delta F \rightarrow G$ 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_\wedge^i de la forme $[...] \rightarrow A \wedge B$ et deux formules $F_\wedge^{e,g}$ et $F_\wedge^{e,d}$ de la forme $A \wedge B \rightarrow [...]$ de telle sorte a ce que $\vdash \varphi$ soit démontrable en logique minimale avec les règles $Ax, \rightarrow_i, \rightarrow_e, \wedge_i, \wedge_e^g, \wedge_e^d$ si et seulement si $\triangleright_{H \cup \{F_\wedge^i, F_\wedge^{e,g}, F_\wedge^{e,d}\}} \varphi$ est prouvable.
8. Meme question pour le \vee et le \neg .

$\neg\neg$ -Traduction de Godel-Kolmogorov⁶⁸

On note \mathcal{L} l'ensemble des formules logique propositionnelle. Pour $F \in \mathcal{L}$, on défini inductivement la non-non traduction de F note $F^{\neg\neg}$ par :

$$\begin{aligned} (A \wedge B)^{\neg\neg} &:= A^{\neg\neg} \wedge B^{\neg\neg} & X^{\neg\neg} &:= \neg\neg X \quad \text{pour } X \text{ une variable} \\ (A \rightarrow B)^{\neg\neg} &:= A^{\neg\neg} \rightarrow B^{\neg\neg} & (\neg A)^{\neg\neg} &:= \neg A^{\neg\neg} \\ (A \vee B)^{\neg\neg} &:= \neg\neg(A^{\neg\neg} \vee B^{\neg\neg}) \end{aligned}$$

Pour $\Gamma = \{\Gamma_1, \dots, \Gamma_n\}$ on note $\Gamma^{\neg\neg} := \{\Gamma_1^{\neg\neg}, \dots, \Gamma_n^{\neg\neg}\}$. 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 défini la logique déductive multiplicative sur les formules constitué seulement de \wedge et \rightarrow par celle respectant les règles suivantes. Attention, pour l'axiome, on n'a pas de Γ :

$$\begin{array}{ccc} \frac{\Gamma, A \vdash' B}{\Gamma \vdash' A \rightarrow B} \rightarrow_i & \frac{\Gamma \vdash' A \rightarrow B \quad \Delta \vdash' A}{\Gamma \cup \Delta \vdash' B} \rightarrow_e & \frac{}{F \vdash' F} Ax \\ \frac{\Gamma \vdash' A \quad \Delta \vdash' B}{\Gamma \cup \Delta \vdash' A \wedge B} \wedge_i & \frac{\Gamma \vdash' A \wedge B}{\Gamma \vdash' A} \wedge_e^g & \frac{\Gamma \vdash' A \wedge B}{\Gamma \vdash' B} \wedge_e^d \end{array}$$

1. Montrer que le séquent $\vdash F$ est prouvable en logique minimale ssi $\vdash' F$ est prouvable en logique multiplicative.

⁶⁸Tiré de <https://www.lirmm.fr/~retore/LL/nonnon.pdf>

2. Proposer des règles pour l'introduction et l'élimination du \forall pour la logique multiplicative et traiter leur cas pour la question 1.

MPI: Classes de Complexités

Cours

- Problème de décision, d'optimisation, de maximisation/minimisation
- 3-SAT, SAT, certificat
- Définition de P, NP, NP-dur, NP-complet

Question de Cours

- Comment transformer un problème de maximisation en un problème de décision?
- Montrer que 3-SAT est NP-Complet

SAT-LIKE⁶⁹

Pour chacune des variantes de SAT suivantes, indiquer si c'est P ou NP-Complet :

- 2-SAT
- 4-SAT
- 3-SAT NAE (Not All Equal)
- 3-SAT OIT (exactly One in Three are true)
- XOR-SAT

Liste de potentielle réductions classique

TODO: 3-color, SOMME, 2-Partition, Hamiltonien, Arbre minimal couvrant $X \subseteq V$

Max 2-SAT

On considère 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 devient polynomial
3. Montrer que la version décisionnel du problème est NP-Complet

Langages NP-Complets sur $\Sigma = \{a\}$ ⁷⁰

Un langage L est dit unaire si c'est 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 entrées positives au problème SAT et S . Montrer que L est NP-Difficile ssi il existe une fonction f de complexité polynomiale telle que

$$\forall \varphi, \varphi \in SAT^+ \iff f(\varphi) \in L$$

On cherche à montrer le théorème suivant

Théorème de Berman: S'il existe un langage unaire NP-Dur alors $P = NP$

Soit $\varphi(x_1, x_2, \dots, x_n)$ une instance de SAT. On suppose que L est un langage unaire NP-Difficile.

2. On note $\varphi_{\top}(x_2, \dots, x_n) := \varphi(\top, x_1, \dots, x_n)$ et $\varphi_{\perp}(x_2, \dots, x_n) := \varphi(\perp, x_1, \dots, x_n)$. Montrer que

$$\varphi \in SAT^+ \iff \varphi_{\top} \in SAT^+ \vee \varphi_{\perp} \in SAT^+$$

⁶⁹Algo 1 ENS Lyon

⁷⁰Mallory Marin, doc de colles

On note $|\varphi|$ la taille d'une formule φ , et on suppose que $\forall i, |\top|, |\perp| < |x_i|$

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 $\varphi \in \text{SAT}^+ \iff \exists \varphi \in \Phi_i, \varphi \in \text{SAT}^+$
4. Montrer qu'il existe un polynôme p tel que $\sum_{\varphi \in \Phi_i} |\varphi| \leq p(|\varphi|)$
5. En déduire que $P = NP$, et donc le théorème de Berman.

Intersection d'automates⁷¹

On considère le problème NONEMPTY-REGULAR-INTERSECTION suivant:

Entrée: A_1, \dots, 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'opérateur 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 .

⁷¹Algo 1 ENS Lyon TD 8

MPI: Calculabilité

Question de Cours

- Rapeller le problème de l'arrêt. Montrer qu'il est indécidable.
- Donner un exemple de problème semi-décidable et montrer qu'il est semi-décidable.
- Donner un exemple de problème qui n'est pas semi-décidable et le prouver.

Indécidabilité

Montrer que les problèmes sont indécidables:

- **ε -Arrêt: Entrée:** Un programme P , **Sortie:** Est-ce que $P(\varepsilon)$ termine?
- **42-Arrêt: Entrée:** Un programme P , **Sortie:** Est-ce que $P(42)$ termine?
- **\neg -Arrêt: 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?
- **Arrêt- \exists : Entrée:** Un programme P prenant en entrée un naturel, **Sortie:** Est-ce qu'il existe un n tel que $P(n)$ termine?
- **Arrêt- \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?

Entrée bornée

Montrer que le problème suivant est indécidable:

- **Entrée:** Une fonction calculable $P : \mathbb{N} \rightarrow \mathbb{N}$ 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 chaîne 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 décidabilité⁷²

Montrer que pour toute fonction calculable $g : \text{String} \rightarrow \mathbb{N}$, il existe une fonction calculable $f : \text{String} \rightarrow \text{char}$ si complexe que pour tout programme P qui calcule f , il existe une entrée $u \in \text{String}$ sur laquelle $P(u)$ prend plus de $g(u)$ étapes de calcul.

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 .

Représentation d'ensembles infini

On se propose ici de créer une structure de donnée permettant de représenter un ensemble infini sous la forme d'une fonction de $'a \rightarrow \text{bool}$ déterministe et qui **termine toujours**. On définit donc le type suivant en OCaml :

```
type 'a set = 'a -> bool;;
```

⁷²tiré d'un oral d'Ulm 2021 https://a3nm.net/work/exams/ens/exercices_info_ulm_2021.pdf

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.