La Totale – Introduction

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

Les exos sont faits pour être indépendants les uns des autres, sauf mention contraire. Ils sont plutôt durs : n'hésitez pas à rajouter des questions intermédiaires si vous voulez les donner à des élèves. L'idée, c'est qu'ils correspondent à peu près à ce qu'on pourrait faire en une bonne colle. 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.

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

Contents

[1] de Maxime BRIDOUX

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

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

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

[9] Tiré du TD de la martinière de MPI

[10] Tiré du TD de la martinière de MPI

[11] Tiré du TD de la martinière de MPI

[12] oral info Ulm ENS 2019
[13] Oral de l'ENS Info 2019 et sujet de CCINP d'Informatique 2026
[14] Tiré de l'Info D 2026 de décembre
[15] Colle de José
[16] Le fabuleux cours de théorie des langages formels d'Olivier Carton
[17] Oral ENS 2022 C2 https://diplome.di.ens.fr/informatique-ens/annales/2022_InfoU-exercices.pdf
[18] Sujet de José
[19] ENS Ulm oral d'info, je ne sais plus de quand
[20] Oral de l'X
[21] Oral ENS Ulm, je ne sais plus quand

[22] TD11 L3 ENS Lyon Algo 1
[23] Tiré de Mallory Marin
[24] InfoFonda 2026 partie I
[25] InfoFonda 2026 partie IV
[26] InfoFonda 2026 partie V
[27] sujet original!
[28] Le putnam de 2025
[29] Marathon de math de Paris-Orsay 2021
[30] Putnam 2023
[31] Sujet original!
[32] https://prepas-mp2i.fr/documents/sujets/2024/CCINP-INFO.pdf
[33] tiré de https://mpi-lamartin.github.io/mpi-info/docs/ia/non_supervise/td_apprentissage

[34] TD11 L3 ENS Lyon Algo 1
[35] Le magnifique livre de Mme. Galatée Hemery !!!! <3
[36] Partie 4 du projet de rocq du cours PRFA 2025/2026
[37] Tiré de https://www.lirmm.fr/~retore/LL/nonnon.pdf
[38] Centrale Info 2026

[39] Mallory Marin, doc de colles

[40] Algo 1 ENS Lyon TD 8

[41] 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 intéressant. On pourrait quand meme faire l'analyse de la fast inverse square-root ou des bizzarerie avec des flottants.

Cours

- Paradigmes: impératif structuré, déclaratifs fonctionel, logique
- Savoir coder en C
- Différence entre compulé 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 contreole. Chemin faisable. Couverture des sommets, couverture des arretes. Test exaustif d'une boucle.
- Pointeurs, allocation, stack, heap, call stack, stack overflow.

Complexité

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

Par example, 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 fonctionelle

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 fonctionel (HP)

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

Mémoisation Automatique[42]

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

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

Suspension et listes infinies

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

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

Une suspension (de type `'a susp`) contient une fonction permettant de calculer une valeur de type `'a` quand cela sera nécéssaire. 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 utilliser 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.

[42] de Maxime BRIDOUX

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.
7. Ecrire une fonction `srev : 'a slist -> 'a slist` qui renverse une liste paresseuse de manière efficace. Quelle est la complexité des accès aux différents éléments de la liste renversée ?
8. Définir en OCaml une liste paresseuse qui contiens *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 queqonque 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 défini 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 modi er le code par continuatiobn pour qu'il n'utilise plus de fonctions anonymes, mais des valeurs d'un type somme OCaml qui représente les di férentes fonctions qui rentre en jeu. On appelle cela la *défonctionnalisation*. On reprends le code d'example pour la fonction `height1`

5. Compléter le code à trou suivant. K1, K2, K3 sont les fonctions anonymé marqué des commentaires `(*1*)`, `(*2*)` et `(*3*)` respectivement. La composition d'une suite de fonction 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 acces lists

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

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 poid 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 example, 6 s'écrit 011. Dans ce cas, la liste de longeur 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 etre sur le meme 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 subtilitées (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 complexitées 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 chercher à définir `val transform : 'a. 'a seq -> int -> ('a -> 'a) -> 'a seq` tel que `transform xs i f` applique `f` au $(i+1)$-ème élément de la liste `xs` et renvoie la nouvelle liste.

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

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

Trop cool, on a obtenu un `set` en $O(\log n)$ qui n'utillise 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 égalitées :

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

Question 1 On se donne le module suivant :

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

```
    | Some x -> f x
end
```

Montrer que `OptionMonad` est une monade

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

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

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

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

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

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

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

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

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

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

Montrer que c'est une monade.

Question 7 Que retourne le code suivant?

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

Question 8 Proposer une implémentation récursive naive de la fonction de fibonacci (sans ce soucier de l'explosion des appels récusif) en utillisant la monade `ContIntMonad`

Parsing avec des monades[43]

Attention: Ce sujet nécéssite 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 utillisant des combinateurs que l'on poura imbriquer les uns aux autres. Un parser sera une fonction de `char list -> ('a * char list) 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
```

[43] MPRI - FUN Exam 2026

L'idée est que un objet de type `'a parser` est une fonction qui prend une liste de charactère 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 charactère pas encore lu. En particulier, si la séquence est vide, c'est que le parsing a échoué. On dit qu'un charactère est consommé par le parser s'il est lu.

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

Question 1 Définir les fonctions suivantes :

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

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

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

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

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

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

On défini alors le combinateur suivant :

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

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

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

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

Question 7 Quel est le type et que fais `parse_d` 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 reconnait une suite arbitraire de concaténation de séquences reconnues par `p`

Question 10 Ecrire un parser qui reconnait exactement le plus grand préfixe de $\{0, 1\}$ et qui renvoie sa longeur. Le parser renvera une séquence d'exactement un élément $[(x, l)]$ avec x la longeur et l le reste de l'entrée. On n'utillisera que les combinateur précédemment utillisé

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

MP2I: Structures de données séquentielles

Cours

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

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 exaustive, algorithme de retour sur trace.
- Algorithme glouton et programmation dynamique.

Merge de liste

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

Calcul des différences

Proposer un algorithme qui prend un tableau d'entiers T de longeur 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 longeur n tel que $\forall i < N, T[i] = \sigma(i)$

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

Tableaux binaire

On considère le problème suivant :

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

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

Sous-tableau connexe

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

Le glouton par défault[44]

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

Prouver la correction de votre algorithme et donner sa complexité.

Recherche dans une matrice

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

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

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

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

Question 4 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 $n \times n$

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 longeur 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 utillisant un tableau cumulatif.

Question 4 On cherche maintenant a 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.

---

[44] Algo 1 ENS Lyon

[45] Algo 1 ENS Lyon

Multiplication rapide de polynome[4][5]

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 degrée $\leq n$ représenter par deux tableau de longeur n et qui renvoie le polynome produit.
2. Soit $n = 2m$, pour $P \in \mathbb{Z}_n[X]$, montrer que on peut décomposer $P = P_1 + X^m P_2$ avec $P_1, P_2 \in \mathbb{Z}_m[X]$.

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

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

Structure efficace pour représenter des ensembles

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

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

1. Proposer une implémentation de cette structure en utillisant des listes. Quels sont les complexitées 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 rapelle que un entier N à une taille de mémoire $\log_2(N)$.

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

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

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

3. Implémenter les fonctions `add` et `del` pour cette représentation. Quels sont les complexitées?
4. Proposer une implementation tel que `add` et `del` soient en $O(1)$ en complexité, mais que la structure soit toujours en $O(N)$ d'espace. La fonction `create` peut-etre 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 chainé en C.

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

2. Proposer un code C pour définir la liste cyclique qui représente $[1, 2, 3, 2, 3, ...]$ en n'utillisant que 3 maillons.
3. Proposer un algorithme pour tester si une liste est cyclique. On fera bien attention à ce que l'algorithme termine.

4. Donner une fonction `liste_t* reverse(liste_t* list);` qui inverse une liste simplement chainé en C. Que ce passe-t'il si on inverse une liste cyclique?
5. On cherche maintenant à calculer le nombre de maillons d'une liste cyclique. Proposer un algorithme en $O(n)$ de temps et $O(1)$ de mémoire.

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

Tableaux auto-référents

Pour T un tableau de longueur n, on définit l'histogramme de T noté H_T comme le tableau de longeur 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 example, $[1, 2, 1, 0]$ est un tableau autoréférent.

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

Question 2 Donner tout les tableaux auto-référent de longeur 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 exaustive `int count_autoref(int n);` qui donne le nombre de tableaux auto-référents de longeur n.

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

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

MP2I: Inductions & ordres[46]

Cours

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

Des ordres

Les ordres suivant sont-ils bien fondés?

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

Des ensembles inductifs

Proposer une définition inductive de chacun des ensemble suivants

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

Soustraction entière

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

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

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

Ordre sur les mots

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

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

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

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

[46] Maxime Bridoux, MP2I

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

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

Tri aléatoire

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

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

Langage de Dyck

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

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

Système MIU

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

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

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

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

1. **(∗)** Est-ce qu'il existe $A \subseteq \mathcal{P}(\mathbb{N})$ indénombrable tel que 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 défintion de la hauteur d'un arbre.
- Définition d'un arbre complet. Localement complet. Parfait.
- Donner la définition d'un tas min (et d'un tas max)
- Rapeller le principe du tri par tas et sa complexité.
- Donner la définition d'un arbre binaire de recherche.
- Arbre rouge-noir. Ajout dans un arbre rouge-noir.

Questions de programmation du cours:

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

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

Exercices à ajouter

- Comptage de sous-arbres: https://diplome.di.ens.fr/informatique-ens/annales/2017_InfoU-exercices.pdf

Nombre de sommets

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

Plus de feuilles

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

Contour d'un arbre

On considère la fonction contour suivante en OCaml :

```
type 'a tree = F | N of 'a tree * 'a * 'a tree;
let rec contour t = match t with
  | F -> []
  | N (g,x,d) -> [x] @ (contour g) @ [x] @ (contour d) @ [x]
```

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

Arbres d'intervalles

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

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

On 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'intervalle.

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

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

Question 5 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.

Accès dans un arbre parfait

On se donne le type C suivant

```
typedef struct Noeud *arb;
struct Noeud {
  int valeur;
  arb fils_g;
  arb fils_d;
};
```

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

6. Discutez de la complexité de `arb_trouve` et de potentiels moyens de l'améliorer. On pourra chercher un algorithme en $O(\ln n)$

Arbres 2-dimensionels

On cherche a crée une structure de donné 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 utillisé la relation d'ordre $(x, y) \leq' (x', y) \Leftrightarrow x \leq x'$ à la place?

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

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

4. Donner un arbre binaire 2-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 stoqué dans l'arbre trié selon la première composante?

Reconstruire l'arbre avec des parcours

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

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

Question 2 Montrer que le parcours T_{pref} et T_{suff} ne suffissent 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}

Arbre canonique[47]

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.

[47] ENS 2023 MPI Info A

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.

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

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

Question 4 Montrer la correction de votre algorithme

Arbres généraux

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

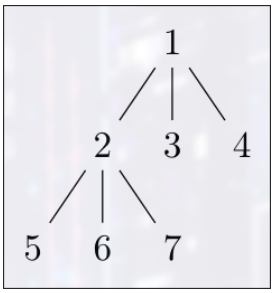
```
type bintree = | V | N of bintree * int * bintree
type tree = | N of int * tree list
```

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

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

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

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



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

Tableaux tri-coloré

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

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

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

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 longeur `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 a initialiser un corriger 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 $\Theta(n)$

Mots d'arbres

On se fixe un ensemble fini de lettres $\Sigma = \{a, b\}$. On défini un *mot* comme étant une suite finie de lettres de Σ. Le mot vide (la suite vide) sera noté ε, et la concaténation de deux mots u, v sera noté

par la concaténation uv. Pour w un mot, on note $|w|_a$ le nombre de a et $|w|_b$ le nombre de b. On définit par induction l'ensemble A par :

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

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

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

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

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

Ranger des 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 :

```c
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 ?

Arbre et oracle

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

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

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

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

Indépendant dans les arbres

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

```
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 poid maximal et le retourne.

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

Question 3 Proposer un algorithme ocaml qui calcule le poid maximal d'un ensemble fortement indépendant.

Question 4 Proposer un algorithme ocaml qui calcule le poid 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 ai

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

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

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

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

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

On suppose que on possède une fonction

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

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

MP2I: Graphes (introduction)

Cours

- Définition d'un graphe $G = (V, E)$ avec $E \subseteq \mathcal{P}_2(V)$. Graphe orienté avec $E \subseteq V^2$.
- Arrete, sommet, degrée, chemin, cycle, connexité, distance
- Graphes pondéré sur les sommets, sur les arretes
- Marche et circuit (HP)
- Arbre comme graphes connexes acyclique
- Parcours en largeur, parcours en profondeur
- Algorithme de Floyd-Warshall
- Algorithme de Dijsktra
- Graphes induits, sous-graphes (HP)
- Coloration, cliques, indépendants (HP)
- Graphes biparti

Questions de Cours

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

Exercices à rajouter

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

Autour des degrées

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

- $2|A| = \sum_{v \in V} \deg(v)$
- Le nombre de sommet de degrée impair est pair
- Il existe forcément deux sommets de même degrée.
- 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.

Petites questions en graphes connexes

Soit G un graphe connexe.

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

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 arrete 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 ai soit $(u, v) \in E$ soit $(v, u) \in E$ (mais pas les deux). On dit que dans un tournois, x est le roi si 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.

Graphes à unique sortie

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

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

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

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

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

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 $x \underset{T}{\preceq} y$ si x est un enfant de y

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

Conversions de DAG

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

On ce donne les types suivants en OCaml:

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

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

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

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

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

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

On supposera écrite la meme fonction pour les puits.

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

Chemin hamiltonien dans un tournois

On dit que $G = (V, E)$ un graphe orienté est un tournois si pour tout $(u, v) \in V^2$, on ai exactement une des deux arretes (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 $s_1, ..., s_n$ 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'asjascence.

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

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

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

Graphes d'Halin[48] 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 degrée 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.

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

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

Graphes d'Halin[49] 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 degrée 2 et C un cycle passant par toutes les feuilles.

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

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

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

Nombre cyclomatiques[50]

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

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

Calcul des triangles

Pour $G = (V, E)$ un graphe non orienté avec $V = \{1, ..., n\}$, on dit que $\{x, y, z\} \subseteq V$ est un *triangle* si $\{x, y\}, \{y, z\}, \{z, x\} \in E$. On cherche a 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 trié d'entiers de 1 à n, montrer que l'on peut calculer la liste triée des élément appartenant aux deux listes en temps $O(|L_1| + |L_2|)$.
3. Donner un algorithme en $O(|E| \times \Delta)$ pour calculer tout les triangles sans doublons d'un graphe G ou Δ est le degrée maximal de G. On supposera que G est donné sous la forme d'une liste d'adjascence.
4. Dans quels cas est-ce que l'algorithme de la question 4 est meilleur que celui de la question 2?

Graphes orienté semi-connexes[51]

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

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

---

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

[50] Tiré du TD de la martinière de MPI

[51] Tiré du TD de la martinière de MPI

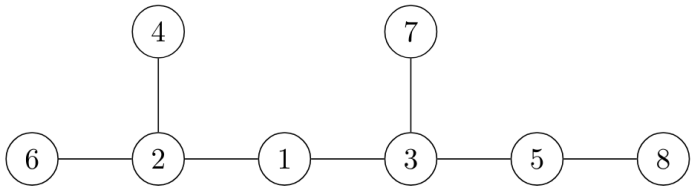
5. En considérant une décomposition en composante fortement connexe, proposer un algorithme pour tester si un graphe est semi-connexe.

Codage de Prufer[52]

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

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

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



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

[52] Tiré du TD de la martinière de MPI

MP2I: Logique Propositionelle

Cours

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

Questions de Cours

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

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

Equivalence

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

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

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

Retirer les constantes

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

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

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

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

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

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

Vers le 3-Fnc

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

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

Les variables propositionelles 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 utillisé.
3. Donner le code OCaml d'une fonction `val to_3 : fnc -> fnc` qui prend une formule en forme normale conjonctive et qui renvoie une formule équivalente ou chaque clause possède au plus 3 littéraux. On aura le droit d'introduire de nouvelles variables non utillisée initialement.

Push down not

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

```
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 propositionelle.

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.

Formule pour la bi-partition

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

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

Formule pour le centre d'un graphe

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

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

Formules linéaire

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

1. Montrer que si F est une formule propositionelle linéaire qui n'utillise pas $\bot$, alors il existe une valuation μ telle que $\mu \vDash F$
2. Dans le cas ou F est linéaire, proposer un algorithme polynomial qui compte le nombre de valuations satisfesant F

Equivalences

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

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

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

1. Montrer que $A \to B \equiv A \uparrow (B \uparrow B)$
2. Montrer que n'importe quel formule est équivalente à une formule n'utillisant 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'utillisant 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 utillisé une version polynomiale.

Indéterminé

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

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

Pour v une valuation et F une formule, on a:
- $v(F) = \top$ si quelque soit F' qui est F ou on a remplacé les ? par des valeurs $\top$ ou $\bot$, on a $v(F') = \top$
- $v(F) = \bot$ si quelque soit F' qui est F ou on a remplacé les ? par des valeurs $\top$ ou $\bot$, on a $v(F') = \bot$
- $v(F) = ?$ sinon

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

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

Formules duales

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

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

XOR SAT

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

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

Lemme d'interpolation de Craig

Soit F et B deux formules de la logique propositionelle et X une variable propositionelle, on note $F[X \leftarrow B]$ la formule ou l'on a remplacé la variable propositionelle 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 propositionelle tel que $F \rightarrow G$ est une tautologie.

2. Montrer que si F n'a aucune variable propositionelle en commun avec G, alors soit $\neg F$ soit F est une tautologie
3. Montrer que s'il existe au moins une variable propositionelle en commum 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 propositionelle de H apparait dans F et G

Compacité

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

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

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

1. Les ensembles suivants sont-ils satisfiables ?

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

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

2. Montrer que si A est satifiable alors il est finement satifiable.
3. $(*)$ Dans le cas ou l'ensemble des variables propositonnels de A est dénombrable, démontrer en construisant par récurrence une valuation que si A est finement satifiable alors A est satifiable.
4. Utilliser le théorème de compacité pour montrer que un graphe infini dénombrable est N coloriable ssi tout ses sous-graphes fini sont N coloriable.

MPI: Langages

Cours

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

Questions de Cours

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

Petites questions

1. Donner le plus petit langage clot par préfixe contenant tout les mots qui peuvent s'écrire de la forme $\{a^n b^n : n \in \mathbb{N}\}$
2. Soient $u, v \in \Sigma^*$ et $a, b \in \mathbb{N}$ tel que $u^a = v^b$. Montrer qu'il existe $w \in \Sigma^*$ et $p, q \in \mathbb{N}$ tel que $u = w^p$ et $v = w^q$
3. 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.

Opérations de langages

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

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

Cloture de langages

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

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

Suite de mots par récurrence

On défini 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 défini 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 $u[i...i+p-1]$ et $v[j...j+p-1]$ ont le plus grand nombre de caractères identiques.

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

Hypercubes et mots

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

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

Mots univers

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

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

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

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

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

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

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

Equations de mots (oral info LCR ENS 2022)

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

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

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

1. Donner toutes les solutions de l'équation $\boldsymbol{ab}x = x\boldsymbol{ba}$ sur $\Sigma = \{a, b\}$

TODO: continuer l'exos

Ensemble inévitables[53]

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

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

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

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

Mots sans carré[54]

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 contiens pas de u^3 en

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

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

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

Facteurs de points fixe de morphismes[55]

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

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

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

Soit f une substition primitive.

3. Montrer qu'il existe $k \in \mathbb{N}$ et $\alpha \in \Sigma$ tel que f^k soit α-prolongeable.
3. On défini $F(w)$ l'ensemble des facteurs de w pour $w \in \Sigma^*$. Soient $L = \bigcup_{n\in\mathbb{N}} F\big(f^{nk}(\alpha)\big)$ et $w \in L$, montrer que pour tout $N \in \mathbb{N}$, il existe un mot v avec $|v| > N$ tel que vw est un préfixe de $\lim_{n\to\infty} f^{kn}(\alpha)$. On dit alors que $\lim_{n\to\infty} f^{kn}(\alpha)$ est *uniformément récurrent.*

---

[53] oral info Ulm ENS 2019

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

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

$(*)$ 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 $\varphi : [\![0, |u|]\!] \longrightarrow [\![0, |v|]\!]$ strictement croissante tel que pour tout $i \in [\![0, |u|]\!]$ on ai $u_{\varphi(i)} = v_i$. On note cela $v \succcurlyeq u$.

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

MPI : Langages réguliers et Automates

Cours

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

Berry-Sethi

Que vaux les automates obtenu par la méthode de Berry-sethi appliqué aux expressions suivantes:

- $(a \mid b)^*$
- $(a \mid b)^* aaab$
- $(a \mid b)^*(c(ba \mid c))^*$

Langages réguliers

Soit L, L' deux langages réguliers. Montrer que les langages suivants sont réguliers :

- $\left\{w_1 w_1 w_2 w_2 ... w_{|w|} w_{|w|} : w \in L\right\}$
- $\{w_1 w_3 w_5 .... w_k : w \in L \text{ avec } k = |w| \text{ si } |w| \text{ impair et } k = |w| - 1 \text{ sinon}\}$
- $\overline{L} = \{\overline{u} : u \in L\}$ le langages des mirroirs
- $\bigcup_{w \in L} S(w)$ pour $S(w)$ l'ensemble des sous-mots de w
- $L/L' = \{w \in \Sigma^* \mid \exists x \in L', wx \in L\}$

Non régulier

Montrer que les langages suivant ne sont pas réguliers :

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

Petites questions

- Monter que l'on peut réécrire une expression régulière sans $\emptyset$ si le langage dénoté n'est pas vide.
- Soit L un langage reconnaisable 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 (respc. 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 $\mathcal{A}$ vérfie $L(\mathcal{A}) = \psi(L(\mathcal{A}))$

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

On considère un automate $\mathcal{A}$ pas forcément déterministe munit d'une fonction de cout $c : \delta \longrightarrow \mathbb{R}$ sur les transitions. Le cout d'un chemin est la somme des couts de chaque transition. Soit u un mot accepté par l'automate, on défini 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 $c : \delta \longrightarrow \mathbb{R}^+$ est positive, alors pour tout C, $L(\mathcal{A})_C$ est un langage régulier.
4. Toujours si $c : \delta \longrightarrow \mathbb{R}^+$, proposer un algorithme qui renvoie un des plus petits mots en longeur $u \in L(\mathcal{A})_C$. Pouvez-vous trouver une heuristique admissible et utilliser l'algorithme 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:

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

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

Sous-langage palindromique

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

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

Langage régulier à partir de l'automate[56]

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 ?*

[56] Colle de José

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

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

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

Langage rotationel

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

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

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

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 a créer les plus petits mots n-univers.

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

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

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

Astucieux langage[57]

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 $S(L) \coloneqq \{n \in \mathbb{N} : a^n \in L\}$. Réciproquement, pour $S \subseteq \mathbb{N}$, on pose $L(S) = \{a^n : n \in S\}$

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

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

Permuté

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

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

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

Langages surprenants

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

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

Automates partiellement ordonnés[58]

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?
2. Montrer que sur $\Sigma = \{a\}$, un langage L est partiellement ordonné si et seulement si il est fini ou il est le complémentaire d'un langage fini.
3. Donner un langage L partiellement ordonné de complémentaire non partiellement ordonné.

Commuteurs de Langages

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

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

1. Montrer que $[\![L_1; L_2]\!]$ n'est pas régulier pour $L_1 = \{a^n b^n : n \in \mathbb{N}\}$ et $L_2 = \{b^n a^n : n \in \mathbb{N}\}$. *Ind: vous n'etes pas obligé de calculer* $[\![L_1; L_2]\!]$
2. Soit L un langage reconnaisable 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 régulier, alors $[\![L_1; L_2]\!]$ l'est aussi.

Commutant de Langage[59]

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

1. Décrire $\text{Comm}(\text{abab})$
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 $\text{Comm}(L) \neq \emptyset$

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

[59] Sujet de José

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

Langage coupé au millieu

Soit Σ un alphabet et L un langage, on pose

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

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

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

Question 4 Soit L un langage reconnaisable 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 $\mathrm{TM}(L)$ aussi.

Langages fins

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

On cherche à charactériser les langages fins.

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

Langages épars[60]

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

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

Langages rationel infinis

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

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

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

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

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

Racine de langage[61]

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

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

Arithmétique de Presburger (+ déduction)

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

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

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

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

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

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

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

Cloture par sur-mot[62]

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

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

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

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

[61] Oral de l'X

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

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

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

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

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

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

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

Langages continuables et mots primitifs

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

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

MPI: Langages Hors-contexte

Cours

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

Question de Cours

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

To Do

- Sujet sur les automates à pile
- Sujet sur les images de Parikh

Langage hors-contexte

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

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

Paires différentes

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

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

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

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

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

Autant de lettres

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

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

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

1. Est-ce que la grammaire définissant L_+ est ambigue? Donner une grammaire pour L (sans preuve)
2. Montrer que L_+ est l'ensemble des mots w tel que $f(w) = 0$ et que pour tout w' préfixe de w on ai $f(w') \geq 0$

3. En remarquant qu'un mot de L_+ commence forcément par un a, donner une grammaire non-ambigue pour L_+.
4. Donner une grammaire non-ambigue pour L est hors-contexte

Forme normale de Chompsky

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

Lemme d'itération pour les langages hors-contexte

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

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

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

Circular shift

Soit L un langage. On définit

$$\mathrm{Circ}(L) = \{uv \mid vu \in L\}.$$

Question 1 Montrer que si L est régulier, alors $\mathrm{Circ}(L)$ l'est aussi.

On cherche maintenant à généraliser le résultat pour les langages hors-contexte. Soit $G = (\Sigma, \Gamma, R, S)$ une grammaire hors-contexte telle que $L = L(G)$. Pour $A \in \Gamma$, on pose

$$C_A = \{vu \in \Sigma^* \mid S \Rightarrow^* uAv\}$$

Question 2 Montrer que pour tout $A \in \Gamma$, C_A est hors-contexte.

Question 3 Montrer que pour tout $u, v \in \Sigma^*$ tel que $uv \in L \setminus \{\varepsilon\}$, il existe $A \in \Gamma$ et $x, y \in \Sigma^*$ et $x', y' \in (\Sigma \cup \Gamma)^*$ tel que $S \Rightarrow^* xAy$ et $A \Rightarrow^1 x'y'$ avec $xx' \Rightarrow^* u$ et $yy' \Rightarrow^* v$

Soit $A \to w_1 ... w_n$ une règle de G, pour tout $0 \leq i \leq n$, on pose $L_A^{i-} = \{u \in \Sigma^* \mid w_1 ... w_i \Rightarrow^* u\}$ et $L_A^{i+} = \{u \in \Sigma^* \mid w_{i+1} ... w_n \Rightarrow^* u\}$. On remarque qu'ils sont presque par définition hors conexte.

Question 4 En considérant le langage suivant, montrer que $\mathrm{Circ}(L)$ est hors-contexte :

$$E \cup \bigcup_{(A \to w_1 ... w_n) \in R} \bigcup_{0 \leq i \leq n} L_A^{i-} C_A L_A^{i+}$$

où $E = \{\varepsilon\}$ si $\varepsilon \in L$, et $E = \emptyset$ sinon.

Théorème de Chomsky-Schützenberger (plutot un sujet d'écrit)

Soit $n \in \mathbb{N}^*$, on défini $\Sigma_n = \{a_1, \overline{a_1}, ..., a_n, \overline{a_n}\}$. Les lettres a_i représentent des parenthèses ouvrantes et les lettres $\overline{a_i}$ représentent des parenthèses fermanantes.

On considère la grammaire $\mathcal{G}_n$ engendrée par les règles

$$S \to a_1 S \overline{a_1} S \mid ... \mid a_n S \overline{a_n} S \mid \varepsilon$$

On pose $D_n = L(\mathcal{G}_n)$ le langage des mots de dyck engendrée par $\mathcal{G}_n$.

1. Donner un arbre de dérivation pour le mot $a_1 a_2 \overline{a_2 a_1} a_3 \overline{a_3}$

On dit que $\varphi : \Sigma_1^* \to \Sigma_2^*$ et un morphisme de mot si $\forall uv \in \Sigma_1^*, \varphi(uv) = \varphi(u)\varphi(v)$

2. Donner une expression régulière dénotant $\varphi(D_1)$ pour φ telle que $\varphi(a_1) = aa$ et $\varphi(\overline{a_1}) = a$
3. Montrer que si L est régulier alors $\varphi(L)$ aussi. Meme question si L est hors-contexte.

On s'intéresse à montrer le théorème de Chomsky-Schützenberger :

> Un langage L est hors-contexte si et seulement il existe un langage régulier K,
> un langage de Dyck D_n et un morphisme alphabétique φ tels que $L = \varphi(D_n \cap K)$.

4. Montrer que l'intersection d'un langage régulier et d'un langage hors-contexte est hors-contexte. En déduire un sens du théorème.

On admet que tout grammaire peut etre mise en forme normale de Chompsky, c'est à dire que toutes les règles de la grammaire sont soit $X \to YZ$, soit $X \to \alpha$ ou soit $S \to \varepsilon$ avec $Y, Z \in \Gamma$ et $\alpha \in \Sigma$ et S le symbole initial

Soit $G = (\Sigma, \Gamma, S, R)$ une grammaire hors-contexte sous forme normale de chompsy. On ordonne les $k := |R|$ règles $r_1, ..., r_k$. On pose $G' = (\Sigma', \Gamma', S, R')$ avec :

$$\Sigma' = \Sigma \cup \{\overline{\alpha} : \alpha \in \Sigma\} \cup \bigcup_{i \in [k]} \left\{a_i, \overline{a_i}, b_i, \overline{b_i}, c_i, \overline{c_i}\right\}$$

Et les règles R' sont :
- $X \to a_i b_i Y \overline{b_i} c_i Z \overline{c_i a_i}$ pour chaque $r_i = X \to YZ$
- $X \to \alpha\overline{\alpha}$ pour toute règle de la forme $X \to \alpha$

5. Donner un morphisme de mot φ tel que $L(G) = \varphi(L(G'))$
6. Proposer un langage régulier K tel que $K \cap D_n = L(G)$. Conclure la preuve du théorème.

MPI: Graphes avancé

Cours

- Chemin alternant.
- Cliques et graphe biparti. Biparti ssi pas de cycle impair.
- Couplage maximal, maximum, parfait.
- Arbre couvrant, Kruksal
- Tri topologique, recherche de composante fortement connexes, Kosaraju
- Algo couplage max dans biparti
- Sous-graphe, graphe induit (HP)
- Multigraphe, hypergraphe (HP)
- k-coloration (HP)
- Graphes planaire (HP)

Question de Cours

- Montrer qu'un couplage est maximum si et seulement s'il n'existe pas de chemin alternant augmentant pour ce couplage.
- Montrer qu'un graphe est biparti ssi il ne contiens pas de cycles impair

Graphes à unique sortie

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

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

Question 2 Montrer que le graphe 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).

Kruskal en recherche locale

Soit $G = (V, E)$ un graphe connexe pondéré par $w : E \to \mathbb{R}_+$, on pose $\mathrm{Couv}(G)$ 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'arrete $e' \in E'$ de T et en y ajoutant l'arrete $e \notin E'$.

Question 1 Dans quel cas est-ce que $T[e' \leftarrow e]$ est aussi un arbre couvrant ?

Question 2 Montrer qu'une arrete $e \in E$ est choisie par tout 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 $T[e' \leftarrow e] \in \mathrm{Couv}(G)$, alors T est un arbre couvrant de poid minimal.

On considère alors l'algorithme qui commence sur un arbre couvrant quelquonque T et qui remplace T par $T[e' \leftarrow e]$ tant que c'est possible pour diminuer $w(T)$. C'est ce que on appelle un algorithme de *recherche locale* car il (localement) change l'arbre de manière à atteindre un arbre couvrant de poid 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 défini 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'arretes 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 longeur du plus long chemin dans G. On défini la *maille* d'un graphe G comme étant la plus petite longueur d'un cycle de G. On la note, si elle existe, par $\mathcal{M}(G)$. Si la maille existe on dit que le graphe est *maillé*.

Question 1 Soit d le diamètre d'un graphe maillé. Montrer que $\mathcal{M}(G) \leq 2d + 1$

Question 2 Montrer que si G est un graphe biparti maillé, alors $\mathcal{M}(G)$ est pair.

Question 3 Montrer qu'un graphe à n sommets qui contient au plus un cycle contient au plus n arêtes.

Question 4 Montrer que la maille d'un graphe à n sommets et au moins $n + 1$ arêtes est majorée par $\left\lfloor \frac{2n+2}{3} \right\rfloor$

Question 5 Soit $G = (V, E)$ un graphe maillé, on note $m = \mathcal{M}(G)$ et δ le plus petit degrée de G. Montrer que

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

Largeur de bande

Soit $G = (V, E)$ un graphe non orienté connexe. Le dimètre de G, noté $d(G)$, est la longeur du plus long chemin dans G. Pour une fonction dite *d'étiquetage* $f : V \to [\![1; |V|]\!]$ bijective, on définit la *largeur de* f *dans* G comme

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

La *largeur de bande de G*, notée $\psi(G)$, est la plus petite largeur d'une fonction d'étiquetage de G possible

Question 1 Calculer la largeur de bande du graphe de la question 1 (ignorer les poids).

Question 2 Que vaux la largeur de bande d'un cycle à $n \geq 3$ sommets?

Question 3 Soit $\Delta(G)$ le degrée 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 $p : V \longrightarrow \mathbb{R}^2$ tel que pour tout $(a, b), (a', b') \in E$, les segments ouverts $]p(a), p(b)[$ et $]p(a'), p(b')[$ ne se croisent pas. Soit $n \in \mathbb{N}$, on dénote par K_n le graphe complet à n sommets. On dénote par $K_{a,b}$ le *graphe biparti complet* contenant $a + b$ sommets, tel que tout les sommets de 1 à a soient relié à tout les sommets de $a + 1$ à b (sans aucune autre aretes).

Une *face* d'un graphe planaire est un cycle délimitant une zone de la représentation (aka une des parties connexe de $\mathbb{R}^2$ auquel on a retiré 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 degrée ≤ 5.

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

Couverture d'aretes et couplages

Soit $G = (V, E)$ un graphe connexe, on dit que $A \subseteq E$ est une *couverture d'arette* si pour tout $v \in V$, on a un $e \in A$ tel que recouvre v. On dit que A est une couverture d'arretes *minimale* si A est une couverture d'arrete de plus petit cardinal.
Un graphe *étoile* est un graphe tel qu'il existe un sommet u tel que toutes les arretes sont de la forme $\{u, x\}$.

On cherche à montrer que si M un couplage maximal et A une couverture d'arrete minimale, alors

$$|A| + |M| = |V|$$

Question 2 Montrer que s'il existe un couplage parfait M, alors c'est une couverture d'arretes minimale.

On fixe A une couverture d'arrete minimale.

Question 3 Montrer que les graphes étoiles sont exactement les graphes ou il n'existe pas de chemin de longueur strictement plus grande que 2. En déduire que A est une union disjointe de composantes connexe qui sont des graphes étoile.

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

Question 5 Montrer que $|A| + |M| = |V|$. *Indication: On montrer l'inégalité dans l'autre sens. On se fixe un couplage maximal M et on construit une couverture minimale X en prenant M et en ajoutant une arrete par sommet non couvert. Borner la taille de X.*

C-approx du couplages maximum[63]

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 $u', v' \notin S(M)$:
$\quad M \longleftarrow (M \setminus \{u, v\{\}\}) \cup \{\{u', u\}, \{v, v'\}\}$
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 longeur $2t + 1$, et si on en trouve un on effectue $M \longleftarrow (M \setminus P) \cup (P \setminus M)$. L'algorithme donné correspond donc au cas $t = 1$.

3. Donner le facteur d'approximation en fonction de t, et en déduire un *PTAS*, c'est-à-dire que pour tout $\varepsilon > 0$ on a un algorithme polynomial qui renvoie une $(1 - \varepsilon)$-approximation. On déterminera exactement la complexité en fonction de ε.

Calcul de triangles[64]

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

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

Question 4 Donner un algorithme en $O(|E| \times \Delta)$ pour calculer tout les triangles sans doublons d'un graphe G ou Δ est le degrée maximal de G. On supposera que G est donné sous la forme d'une liste d'adjascence.

[63] TD11 L3 ENS Lyon Algo 1
[64] Tiré de Mallory Marin

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 dégrée max du graphe ?

Planaire et degrée

Un graphe $G = (V, E)$ est dit k-régulier si $\forall v \in V, \deg(v) \geq k$. Un graphe G est dit planaire s'il existe une assignation $\varphi : V \longrightarrow \mathbb{R}^2$ tel que pour toute paire d'arête $(x, y), (x', y') \in E$, les segments $[\varphi(x); \varphi(y)]$ et $[\varphi(x'); \varphi(y')]$ ne se coupent pas.

Soit G connexe planaire. On définie f le nombre de face comme le nombre de composante connexe de $\mathbb{R}^2 \setminus \bigcup_{(x,y)\in E}[\varphi(x); \varphi(y)]$.

1. Montrer le théorème d'Euler: $|E| = |V| + f - 2$
2. En déduire que $|E| < 3|V|$
3. En déduire que si G est planaire alors il n'est pas 6-régulier.

Un Graphe dénombrable

On prend $\varphi : \mathbb{N} \longrightarrow \mathcal{P}_f(\mathbb{N})$ une bijection de $\mathbb{N}$ dans les parties finies de $\mathbb{N}$. Montrer qu'a isomorphisme près, et en retirant les boucles, le graphe $(\mathbb{N}, \{\{x, y\} : y \in \varphi(x)\})$ est unique.

Trouver l'isomorphisme pour φ_1, φ_2 deux bijections.

Graphes d-dégénéré[65]

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

Question 1 Montrer que si G contiens une clique de taille k alors $\mathcal{D}(G) \geq k - 1$

Question 2 Montrer que tout sous-graphe d'un graphe d-dégénérer est aussi d-dégénéré.

Question 3 Montrer que G est d-dégénéré si et seulement si $\forall S \subseteq V_G$, $G[S]$ possède un sommet de degré $\leq d$

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

Coeur de k-domination[66]

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 nécésairrement tout le graphe.

---

[65] InfoFonda 2026 partie I

[66] InfoFonda 2026 partie IV

Question 1 Soit X tel que $\forall x \in V_G, |N(v) \cap X| < \left\lfloor |X\frac{\downarrow}{k} \right\rfloor$. Montrer que G n'admet pas d'ensemble quasi-dominant de X de taille au plus k.

Un graphe est dit sans $K_{t,t}$ si le graphe biparti complet à deux parties de t sommets n'est pas présent comme sous-graphe. On considère l'algo suivant:

Entrée: Un graphe G sans $K_{t,t}$, $X \subseteq V_G$.
$Y \leftarrow G$
$S \leftarrow \emptyset$
tant que $\exists v \notin S, |N(v) \cap Y| \geq \left\lfloor |Y\frac{\downarrow}{k} \right\rfloor$ **faire :**
    $S \leftarrow S \cup \{v\}$
    $Y \leftarrow N(v) \cap Y$
fin tant que
retourner Y, S

Question 2 Montrer que si $|X| \geq 2tk^t$ alors l'algorithme termine en au plus $t-1$ itérations de la boucle.

On suppose que $|X| \geq 2tk^t$, et on se donne S, Y le résultat de l'algorithme

Question 3 Montrer que tout ensemble quasi-dominant de taille au plus k intersecte S. En déduire que, soit $y \in Y$, si D domine $G - y$ alors D domine G

Question 5 En déduire un algorithme polynomial qui prend un $k \geq 2$ et un graphe G sans $K_{t,t}$ qui retourne un coeur de k-domination de taille au plus $2tk^t$.

Théorème de vizig[67]

Soit $G = (V, E)$ un graphe, on note Δ le degrée maximal d'un graphe, on dit que $\gamma : E \longrightarrow \{1, ..., k\}$ est une *k-coloration d'aretes* si pour toute paire d'arrete e, e' partageant une extrémité, on a $\gamma(e) \neq \gamma(e')$. Une coloration est dite partielle si la fonction l'est.

On cherche à montrer que tout graphe est $(\Delta + 1)$-coloriable en arrete.

Question 1 Montrer que tout graphe est $2\Delta - 1$-arrete coloriable

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

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

Question 3 Montrer que $G[K(a/b)]$ est une union disjointe de cycle de longeur pair ou de chemins.

Question 4 Soit G biaprti, γ une coloration partielle, $(u, v) \in E_G$ avec $a \in \overline{\gamma}(u)$ et $b \in \overline{\gamma}(v)$. Montrer que si u et v ne peuvent pas etre dans la meme composante connexe de $K(a/b)$

Question 5 Montrer que si G est biparti alors G est $\Delta(G)$-arrete-colorable

Question 6 Soit $(u, v) \in E$ une arrete non coloré d'une coloration partielle γ et soient a, b deux couleurs telle que $a, b \in \overline{\gamma}(u)$ et $b \in \overline{\gamma}(v)$. Montrer qu'il existe γ' qui colorie les meme arretes telle que
- $\overline{\gamma'}(u) = \overline{\gamma}(u)$,
- $a \in \overline{\gamma'}(v)$ et

[67] InfoFonda 2026 partie V

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

Question 6 Soit γ une k-coloration partielle de G dont tout les arretes non coloré $e_1, ..., e_r$ sont toutess de la forme $e_i = (u, v_i)$. On suppose que $|\overline{\gamma}(u)| \geq r$, $|\overline{\gamma}(v_1) \cap \overline{\gamma}(u)| \geq 1$ et que pour tout $i \geq 2$, on ai $|\overline{\gamma}(v_i) \cap \overline{\gamma}(u)| \geq 2$. Montrer que G est k-colorable.

Question 7 En déduire que tout graphe est $(\Delta + 1)$-arrete-colorable.

Question 8 Montrer que pour tout $d > 1$ il existe un graphe G non d-arete colorable de degrée 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 degrée 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 degrée 2 relié $u, u' \neq v$. Montrer que (G, k) est une instance positive ssi $(G - u + \mathrm{uu}', k)$ l'est aussi
4. Soit G une instance et v un sommet avec une boucle. Montrer que (G, k) est une instance positive ssi $(G - v, k - 1)$ l'est aussi.

Soit G le graph obtenu après execution des deux règles précédentes autant de fois que possible. On a donc que le degrée minimum de G est 3. On note V_{3k} les $3k$ sommets de plus grand degrée. 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 ai
$$\sum_{v \in X \cup V_{3k}} \deg v \geq 3|X| + 3kd$$
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 arretes sont adjascente à un $s \in S$?

1. Montrer que pour tout k fixé, le problème est polynomial. *On admet qu'il est NP-Complet dans le cadre général*
2. Soit (G, k) une instance et v un sommet de degrée 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 degrée $\geq k + 1$. Montrer que (G, k) est une instance positive ssi $(G - v, k - 1)$ l'est aussi.

4. Montrer que après execution des deux règles précédentes autant de fois que possible, on ai $|V| \leq k^2 + k$
5. En déduire que VERTEX-COVER possède un algorithme correcte dont la complexité peut s'exprimer sous la forme $O(f(k)n^c)$ pour f, c que l'on donnera.

Avec une décomposition en couronne on peut réduire la taille de G par d'autres règles tel que $|V| \leq 3k$.

Avec de la programmation linéaire et de la relaxation on peut montrer qu'on peut réduire la taille de G tel que $|V| \leq 2k$.

FPT: Kernelisation de Edge clique cover

On considère le problème EDGE-CLIQUE-COVER suivant:

- **Entrée:** Un graphe $G = (V_G, E_G)$ et $k \in \mathbb{N}$
- Est-ce qu'il existe $S_1, ..., S_k \subseteq V_G$ tel que $G[S_i]$ soit une clique pour tout $i \leq k$ et que $E_G = \bigcup_i E(G[S_i])$

1. Montrer que pour $k = 1$ et $k = 2$ le problème est polynomial
2. Soit (G, k) une instance et v un sommet de degrée 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 execution des deux règles précédentes autant de fois que possible, on ai $|V| \leq 2^k$
4. En déduire que pour $k \leq \log_2(|V|)$, le problème EDGE-CLIQUE-COVER est polynomial.

MPI: Théorie des jeux

Cours

- Définition d'un jeu. Jeu d'accesibilité. Stratégie, stratégie gagnante.
- Calcul des attracteurs. Définition des A_i^j dans l'algorithme du calcul des attracteurs.
- Algorithme min-max, élagage α/β, verison avec heuristique.
- Algorithme $A*$
- Zermelo (HP)

Petites questions

- Soit un jeu $G = (V, E)$ avec $V = \mathbb{N}$ et telle que les arètes orienté sont toutes de la forme (x, y) avec $y < x$. On suppose de plus que $\forall x \in V, \deg_+ v = 0 \Rightarrow v \in F_1 \cup F_2$. Montrer que pour chaque état du jeu, un des deux joueurs possède toujours une stratégie gagnante.

Jeu de nim généralisé[68]

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 $A = \left\{2^i : i \in \mathbb{N}\right\}$ l'ensemble des puissances de 2 ? Et pour $\{1, ..., p\}$?
5. Montrer qu'il existe un ensemble A tel que si on note S_A l'ensemble des N tel que Alice possède une stratégie gaganante, alors il n'existe pas $N_0, T > 0, \forall n > N_0, n \in S_A \iff n + T \in S_A$.

Géographie dans les hypercubes[69]

Soit $\Sigma = \{a, b, c\}$ et $n \in \mathbb{N}^*$. Alice et Bob joue au jeu suivant : à tour de role, iels choisissent un mot dans Σ^n qui n'a pas déjà été choisi avant, et tel que il ne diffère que d'une lettre avec le dernier mot choisi. Alice choisi un mot quelquonque au début. Le premier joueur ne pouvant pas jouer perd. On cherche à établir une statégie gagnante.

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

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

[68] sujet original!

[69] Le putnam de 2025

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

3. Montrer par récurrence sur $n \in \mathbb{N}^*$ que $G_n - w_a$ possède une couplage parfais
4. En déduire que Alice possède une stratégie gagnante.
5. Généraliser pour $\Sigma = \{a, b, c, d\}$ et puis après pour tout alphabet fini non vide Σ

Géographie[70]

Soit $G = (V, E)$ un graphe, on définit le jeu à deux joueurs (A)lice et (B)ob tel que à tour de role, chaque joueur doit marquer un sommet non déjà marqué qui soit un voisin du dernier sommet marqué. Le premier joueur qui ne peut plus marquer de sommet perd. Le premier joueur (Alice) peut choisir ou commencer.

2. On considère C_n le cycle de longeur n. Pour quel $n \in \mathbb{N}$ est-ce que Alice a une stratégie gagnante sur C_n ?
3. Montrer que si le graphe possède un couplage parfait, alors Bob possède une stratégie gagnante. La réciproque est-elle vraie?
4. Proposer un algorithme dans le cas ou G est un arbre pour savoir qui possède une stratégie gagnante.
5. On considère le cas ou $V \subseteq \mathbb{N}^2$ et $\forall ((x, y), (x', y')) \in E, |x - x'| + |y - y'| = 1$ avec G connexe. A quelle condition sur $|V|$ est-ce que Alice possède une stratégie gagnante?

Devine le nombre de points fixes[71]

Alice et Bob joue au jeu suivant (TODO)

Jeu de Shannon

On considère un jeu avec un graphe $G = (V, E)$ non orienté qui peut posséder plusieurs arêtes entre deux sommets. Deux joueurs (A)lice et (B)ob, où Alice commence, choisissent alternativement une arête de G non encore choisie. Si, à un moment de la partie, les arêtes choisies par Bob forment un arbre couvrant de G alors Bob gagne. Sinon, Alice gagne.

Pour $\mathcal{P}$ une partition de V, on note $|\mathcal{P}|$ le nombre de parties et $\|\mathcal{P}\|$ le nombre d'arretes 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 arrete de T_1. Montrer qu'il existe e une arrete de T_2 tel que $T_1 - e_1 + e_2$ soit un arbre couvrant de G

Question 2 On suppose qu'il existe une partition $\mathcal{P}$ de V telle que $\|P\| < 2(|P| - 1)$. Montrer que Alice possède une stratégie gagnante.

Question 3 Soit T un arbre couvrant de G et e une arête de T. Soient T' et G' obtenus en contractant e dans T et G, c'est-à-dire en supprimant e et identifiant ses extrémités. Montrer que T' est un arbre couvrant de G'.

Théorème On admet le théorème de Tutte : G possède k arbres couvrant disjoints si et seulement si pour tout partition P de V, $\|P\| \geq k(|P| - 1)$

Question 4 Montrer que Bob a une stratégie gagnante si et seulement si G possède deux arbres couvrants disjoints.

---

[70] Marathon de math de Paris-Orsay 2021

[71] Putnam 2023

[72] Sujet original!

Jeu du sous-mot[72]

Soit Σ un alphabet, on considère le jeu à deux joueurs (dénoté (A)lice et (B)ob) du sous-mot suivant: a tour de role, chaque joueur dit un mot $w \in \Sigma^* \setminus \{\varepsilon\}$ qui ne contiens aucun des mots déjà dit en sous-mot. Le premier joueur qui ne peux plus dire de mot perd. Par exemple, sur $\Sigma = \{a, b\}$, la partie pourrait etre $abb \to bbbb \to aab \to a \to bbb \to bb \to b$ et Alice perd (Alice étant la joueuse qui commence).

On admettera que ce jeu termine toujours (cette propriété viens du lemme d'higman).

Question 1 On admet pour l'instant que Bob possède une stratégie gagnante pour $|\Sigma|$ pair. Montrer que Alice possède une stratégie gagnante pour $|\Sigma|$ impair.

Question 2 Pour $\Sigma = \{a, b\}$, on défini $\overline{a} := b$ et $\overline{b} := a$, puis $\overline{w_1 ... w_n} = \overline{w_1} ... \overline{w_n}$. Montrer que Bob possède une stratégie gagnante pour $\Sigma = \{a, b\}$.

Question 3 En généralisant la question précédente, montrer que Bob possède une stratégie gagnante pour $|\Sigma|$ pair.

Question 4 $(**)$ Montrer que le jeu termine toujours, autrement dit que si $(w_n)_{n \in \mathbb{N}} \in (\Sigma^*)^{\mathbb{N}}$ est une suite infinie de mots, alors il existe $i < j$ tel que w_i soit un sous-mot de w_j.

MPI: Apprentissage

Cours

- Différece entre apprentissage supervisé et non supervisé.
- Algorithme des k-plus proches voisins.
- Arbre de décision, Matrice de confusion.
- Entropie de Shannon, gain, algorithme ID3.
- Algorithme CHA (Classification hiérachique ascendente)
- L'Algorithme des k-moyennes.

Petites questions

1. Donner un algorithme qui ajoute un sommet $\vec{x} \in \mathbb{R}^k$ à un arbre k-dimensionnel. Quel est sa complexité?
2. Écrire une fonction `int* voisins1D(float x, float* X, int k, int n)` permettant de trouver efficacement les k plus proches voisins de x dans l'ensemble de n données X trié par ordre croissant, où chaque donnée est un réel (en dimension 1). La fonction renvoie un tableau d'entiers contenant les indices des k plus proches voisins de x dans X. Quelle est sa complexité?

Single-pass (CCINP 2024)[73]

On considère l'algorithme suivant pour catégoriser des données $X = \{\vec{x_1}, ..., \vec{x_n}\} \subseteq \mathbb{R}^d$. On note $\delta(\vec{x}, \vec{y}) = \sum_{i=1}^{d} |\vec{x}_i - \vec{x}_i|$ la distance entre $\vec{x}$ et $\vec{y}$. La distance de $\vec{x}$ à une classe C est la distance de $\vec{x}$ au centre de C. On note $\mathrm{Cl}(\vec{x}, C)$ la classe dont le centre est le plus proche de $\vec{x}$ pour δ.

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

$C \leftarrow \emptyset$
Pour i de 1 à n:
    Si $\forall \vec{c} \in C, \delta(\vec{x}_i, \vec{c}) > \theta$:
        $C \leftarrow C \cup \{\{\vec{x}_i\}\}$ // On crée une nouvelle classe
    Sinon:
        $\overline{C} \leftarrow \mathrm{Cl}(\vec{x}_i, C)$
        $\overline{C}' \leftarrow \overline{C} \cup \{\vec{x}_i\}$
        $C \leftarrow C \setminus \left\{\overline{C}\right\} \cup \left\{\overline{C}'\right\}$
Renvoyer C

Question 1 Proposer une implémentation en C de la fonction δ. On représentera un $\vec{x} \in \mathbb{R}^d$ par un `float* x`, et d sera traité comme une constante.

Question 2 Montrer que l'ordonnancement des $\vec{x}_1, ..., \vec{x}_n$ importe.

Question 3 Soit $\vec{x} \in X$, on note C_{ini} la classe de $\vec{x}$ au moment ou il a été ajouté, et C^* sa classe à la fin. Montrer que

$$\delta(\vec{x}, C^*) \leq \theta \ln(|C^*| - |C_{\text{ini}}|)$$

On admettera que $\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{k} \leq \ln(k)$

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

---

[73] https://prepas-mp2i.fr/documents/sujets/2024/CCINP-INFO.pdf
[74] tiré de https://mpi-lamartin.github.io/mpi-info/docs/ia/non_supervise/td_apprentissage

Clustering en dimension 1[74]

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

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

On pose $S(i, j) = S(\{x_i, ..., x_j\})$. On cherche à résoudre l'algorithme des k-moyennes sur E de manière exacte. C'est à dire que l'on cherche une partition $\mathcal{P} = \{X_1, ..., X_K\}$ de $[\![0; N-1]\!]$ qui minimise la somme des scores de chaque X_i

Question 2 Montrer que l'algorithme CHA ne permet pas de résoudre de manière exacte le problème. On pourra prendre $N = 4$ et $K = 2$.

Question 3 On note $I(n, k)$ le score minimale d'une partition de $\{x_0, ..., x_n\}$ en k classes. Que vaux $I(n, 1)$?

Question 4 Montrer que

$$\forall n > 0, \forall k > 1, I(n, k) = \min_{k-1 \leq m \leq n-1} (I(m, k-1) + S(m, n))$$

Question 5 En déduire une fonction `double inertie(double* E, int N, int K)` qui calcule le score minimale possible d'une parition de E en K classes non vides.

Question 6 Montrer que l'on peut calculer l'ensemble des $S(i, j)$ en $O(N^2)$

k-centres

Définition On fixe $V \subseteq \mathbb{R}^d$ un jeu de donnée. Pour $\vec{y} \in \mathbb{R}^d$ on définit $d(\vec{y}, V) = \max_{\vec{x} \in V} \|\vec{y} - \vec{x}\|$. On cherche à choisir k points de V tel qu'ils minimise $\max_{v \in V} d(v, X)$. On regarde donc le problème k-CENTRE suivant:

Entrée: $v_1, ..., v_n \subseteq \mathbb{R}^d$ une liste de points finis **Sortie:** $X := \{x_1, ..., x_k\} \subseteq V$ qui minimise $\max_{v \in V} d(v, X)$

Question 1 Rapeller le fonctionnement de l'algorithme des k-moyennes. Quel sont les différences avec le problème des k-centres?

Question 2 On cherche à montrer que l'algorithme du cours de la question ne donne pas de bonne approximation du problème des k-centres. Pour cela on pose $k = 2$, $d = 1$ et on pose N points en 0, N points en 1 et un point en D. Montrer que pour des bonnes valeurs de N et D on peut avoir le ratio $\frac{\text{opt-k-moyennes}}{\text{opt-k-centre}}$ arbitrairement large.

Question 3 Proposer un algorithme glouton pour le problème des k-centres.

Question 4 On note $p_1, ..., p_k$ les k points retrourné par notre algorithme et $p_1^*, ..., p_k^*$ la solution optimale. On partitionne $V = \bigsqcup_{i \in \mathbb{N}} V_i$ avec V_i l'ensemble des points les plus proches de p_i^*. Montrer que

- Si $\forall i, V_i \cap \{p_1, ..., p_n\} \neq \emptyset$ alors on a une 2-approximation
- Sinon, on a aussi une 2-approximation

Algorithme ID3 avec intervalles

On cherche ici à effectuer l'algorithme ID3 pour des attributs non discret (un intervalle des réels par example). Pour cela, on considère des attributs $(A_i)_i$ tel que A_i est soit fini, soit $A_i = [\alpha_i, \beta_i]$, et un ensemble de classes C fini. Pour $S \subseteq A_1 \times .... \times A_k \times C$ un ensemble fini d'apprentisage, et pour

A_i de la forme $A_i = [\alpha, \beta]$, pour $k \in]\alpha, \beta[$, on 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

Cours

- Fil d'execution, non déterminisme, opération atomique.
- Mutex, Sémaphore.
- L'algorithme de Peterson
- L'algorithme de la boulangerie de Lamport
- Problème du rendez-vous, producteur-consommateur, du diner des philosophes.

Lecteurs-Rédacteurs

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

- Plusieurs lecteurs doivent pouvoir lire la base de données en même temps ;
- Si un rédacteur accède à la base de donné alors personne d'autres ne peut l'utilliser.

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'utilliser 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 utilliser 2 mutex?

Question 2 Montrer qu'il y une pottentielle famine des rédacteurs avec cette solution.

Question 3 Proposer une solution avec un compteur protégé par un mutex, et deux sémaphores qui résoud ce problème de famine.

Votes en parralèlles

On se proposer d'implémenter un système de vote entre plusieurs fils. Chacun des `k` fils d'execution executera la fonction `void vote(int valeur)` que l'on se propose d'implémenter. Une fois que tout les fils ont voté, **tout les fils** devront executer la fonction `resultat(int result)` avec `result` la valeur la plus voté par les fils (en cas d'égalité, prendre la valeur la plus petite dans tout les gagnant).

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

Question 1 Dans le cas ou les votes sont soit 0 soit 1, proposer une implémentation de `void vote(int valeur);` à base d'un mutex, un sémaphore initialisé à 0 et de deux variables entières partagés.

Question 2 On considère le cas ou les votes sont dans $[\![0; p]\!]$. Proposer une solution avec un tableau de mutex de longueur p, un compteur protégé par un mutex et un sémaphore initialisé à 0.

Question 3 Proposer une solution avec un tableau de longeur k, le nombre de fils, n'utillisant 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 utillisant un tableaux d'entier partagé, mais aucun mutex nrei sémaphore. On suppose le tableau initialisé à 0.

Salle de spectacle

On considère une salle de spectacle qui peut être utilisée par deux groupes de musiciens: les violonistes et les guitaristes. Chaque musicien se prépare au spectacle en un temps aléatoire. Dès que

tous les musiciens d'un groupe sont prêts, alors ils devront tous jouer ensemble (tous les violonistes ou bien tous les guitaristes jouent). Une fois leur performance faite, ils retournent en préparation.

On suppose qu'il y a N violonistes et M guitariste.

1. Proposer une modélisation de ce problème où chaque musicien est représenté par un fil d'exécution exécutant soit la fonction `violon` soit la fonction `guitare`, sans ce soucier des problèmes de synchronisation. On représentera le fait que tous les musiciens d'un groupe partent jouer par un **unique** appel à une fonction `void spectacle(bool est_violon);`
   On écrit les fonctions `violon` et `guitare`.
2. Quels peuvent être les problèmes rencontrés ?
3. Proposer une solution utilisant 2 sémaphores et 3 mutexs.
4. Et s'il y avait k groupes de musiciens ?

Sémaphore pondéré

On cherche à impémenter une structure de donnée qui est une généralisation pondéré des sémaphores. Elle aura trois opérations :

- `init(int capacite)` qui initiallisé et crée notre sémaphore pondéré avec `capacite` ressources
- `prendre(int k)` qui bloque le fil jusqu'à ce que l'on puisse alloué un total de k ressources
- `vendre(int k)` qui libère k ressources

Question 1 On propose la suivante première implémentation de notre sémaphore pondéré à l'aide d'un sémaphore normal `sem`:

```
void init(int k) {init(sem,k);}

void prendre(int k) {
  for (int i = 0;i<k;i++) {P(sem);};
}

void vendre(int k) {
  for (int i = 0;i<k;i++) {V(sem);};
}
```

Expliquer pourquoi cette implémentation peut donner à des situations d'interblocages

Question 2 Proposer une implémentation sans interblocage utillisant un compteur protégé par un mutex. Cette solution peut-elle etre source de famine?

Question 3 Proposer une solution garentissant l'abscence de famine utillisant un sémaphore, et une variable protégé par un mutex.

Question 4 Montrer l'abscence d'interblocage et de famine.

Mutex ré-entrant

On cherche à implémenter un mutex réentrant: c'est un mutex sauf que si la ressource est allouée à un fil k, alors le fils k peut se lock autant de fois qu'il veux. Notamment, cela permet d'executer ce code sans situation de blocage :

```
void example() {
  lock(mutex_reentrant);
  lock(mutex_reentrant);
  unlock(mutex_reentrant);
}
```

Question 1 Proposer une implémentation d'un tel mutex réentrant en utillisant seulement un mutex classique, un entier partagé et de l'attente active.

On cherche maintenant à faire la meme chose pour des sémaphore: si un fil d'execution execute `p(semaphore)` et obtiens la ressource, alors il peut appeller `p(semaphore)` autant de fois qu'il veux sans etre mis en attente ni faire décroire le compteur interne, et ce jusqu'à ce que le fil appelle `v(semaphore)`.

Question 2 Proposer une fonction qui, si elle est executé par deux fils, peut donner lieu a un interblocage des deux fils avec un sémaphore classique, mais qui ne donne pas lieu à un interblocage avec un sémaphore réentrant.

Question 3 Proposer une implémentation d'un tel sémaphore réentrant avec un tableaux de booléens et un sémaphore classique.

Pont à voie unique

On cherche ici a modéliser un pont à voie unique qui ne peut etre que emprunter par des voitures allant dans le meme sens. On représentera cela par deux types de fils d'execution : les fils ascendant et les fils descendant. Chaque fil répète en boucle l'execution de `request_bridge(b);` puis de `exit_bridge(b);` avec `b` un booléen valant `true` iff le fil est ascendant. On veux s'assurer que pour chaque fil, lors de sa présence dans la section critique (entre `request_bridge(b);` et `(b);`), il n'y a que des fils allant dans la meme direction.

Question 1 Proposer une solution utillisant un mutex, un compteur partagé et de l'attente active. Expliquer pourquoi il peut avoir une famine.

Question 2 Donner une solution ayant une abscence de famine avec 2 sémaphores initialisé à 1 et un entier. On pourra s'inspirer de la solution avec de l'attente active.

Question 3 On suppose maintenant qu'un maximum de 5 voitures peuvent etre présentent en meme temps sur le pont. Modifier le code pour correspondre à ce critère.

Question 4 Et s'il le pont possèdait k voies, tel qu'une seule ne peut etre empruntée en meme temps?

MPI: Proba & Approx

Cours

- Algorithme déterministe, de Las Vegas, Monte Carlos.
- Problème de décision, approximation, instance positive, instance négative.
- Fonction de cout, algorithme d'approximation.

Question de Cours

- Qu'est-ce qu'un algorithme de Monte-Carlo et de Las vegas ?
- Comment transformer un problème d'approximation en un problème de décision?

Petites Questions

- Soit P un problème de minimisation et $\overline{P}$ le problème de décision associé. On suppose que $\overline{P}$
- On choisi uniformément des entiers dans $\{1, ..., n\}$ avec remis j'usqu'à ce que tout les entiers ont été choisi. On note X la variable aléatoire du temps d'execution. 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;
}
```

2-approximation du circuit eulérien (TSP)

Soit $G = (V, E)$ un graphe non orienté pondéré par $w : E \to \mathbb{R}_+^*$. On note pour $c := (c_i)_{i \leq n}$ un chemin $w(c) = \sum_{i<n} w((c_i, c_{i+1}))$ son poids. Similairement, on 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 $d : V^2 \longrightarrow \mathbb{R}_+$ une distance (respecte les axiomes de la distance).
- **Sortie** Le poid minimum d'un cycle passant par tout les sommets.

On note $w(C)$ pour C un chemin (ou arbre) son poid. On pose T^* un arbre couvrant de G' de poid minimal et I l'ensemble des sommets impair de T^*. On note w^{OPT} le poid optimal d'une instance du problème.

Question 1 Montrer que $w(T^*) < w^{\mathrm{OPT}}$. En déduire une 2-approximation du problème en utillisant un parcours de T^*

Question 2 Montrer que $|I|$ est pair et en déduire que G' restraint à I possède un couplage de poid minimal que l'on notera M.

Question 3 Montrer que si G est un graphe connexe tel que $\forall v \in V, \deg v \in 2\mathbb{N}$, alors G possède un cycle eulérien (passant par toutes les aretes). En déduire que le multigraphe $T^* + M$ possède un cycle eulérien que l'on notera h.

Question 4 Montrer que le problème du voyageur de commerce admet une $\frac{3}{2}$-approximation.

C-approx du couplages maximum[75]

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 $u', v' \notin S(M)$:
$\quad M \longleftarrow (M \setminus \{u, v\{\}\}) \cup \{\{u', u\}, \{v, v'\}\}$
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 longeur $2t + 1$, et si on en trouve un on effectue $M \longleftarrow (M \setminus P) \cup (P \setminus M)$. L'algorithme donné correspond donc au cas $t = 1$.

3. Donner le facteur d'approximation en fonction de t, et en déduire un *PTAS*, c'est-à-dire que pour tout $\varepsilon > 0$ on a un algorithme polynomial qui renvoie une $(1 - \varepsilon)$-approximation. On déterminera exactement la complexité en fonction de ε.

Vertex cover

Soit $G = (V, E)$ un graphe. On dit que $S \subseteq V$ est *une couverture* si pour toute arrete $e \in E$, au moins une des extrémitées de e est dans S. On s'intéresse au problème VERTEX-COVER suivant :

- **Entrée:** $G = (V, E)$ un graphe
- **Sortie:** Une couverture C de taille minimale en cadinal

1. Donner la version problème de décision de VERTEX-COVER.

On considère un algorithme glouton qui, tant qu'il reste des arretes au graphe, en choisit une et retire les deux sommets du graphe. On note C les arretes choisies la fin.

2. Montrer que si S est une couverture, alors $|S| \leq 2|C|$
3. En déduire une 2 approximation de VERTEX-COVER

On considère l'algorithme qui, tant que le graphe possède encore des arretes, prend un sommet de degrée non nul au hasard uniformément, le choisi et retire toutes les arretes relié à ce sommet.

4. Quel est ce type d'algorithme ? Soit $G = (V, E)$ un graphe, on note V_f la variable aléatoire de l'ensemble choisi à la fin de l'execution de l'algorithme sur G. Montrer que

[75] TD11 L3 ENS Lyon Algo 1

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

Set cover

On considère le problème SET-COVER suivant :

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

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

Question 1 Résoudre le problème pour les instances de la forme $n \in \mathbb{N}; m := n + 1; U_i = \{i, i + 1\}$

Question 2 Donner une $\log(m)$-approximation de SET-COVER. *Ind: On fera un algorithme glouton et on pourra utilliser le fait que* $\frac{1}{2} + ... + \frac{1}{n} \leq \log n$

On considère maintenant que la sortie doit etre un I qui maximise le nombre de $x \in [\![1; m]\!]$ tel que x appartienne à un nombre impair d'ensembles de $(U_i)_{i \in I}$

Question 3 Soit $(U_i)_{i \leq n}$ une instance du problème. On choisi avec probabilité $\frac{1}{2}$ chaque U_i. On note N la variable aléatoire du nombre de $x \in [\![1; m]\!]$ appartennant à un nombre impair d'ensembles choisi.

Question 4 Montrer que $\mathbb{E}[N] = \frac{m}{2}$

Question 5 En déduire un algorithme un algorithme de Las Vegas qui est une 2-approximation. Existe t'il une 2-approximation déterministe ?

Unique graphe infini aléatoire

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

On cherche à montré que les graphes aléatoire infinis dénombrable sont isomorphes avec probabilité 1. On considère pour cela les variables aléatoire de bernoulli $\left(X_{u,v}\right)_{u,v \in \mathbb{N}}$ indépendantes de probabilité p qui représente s'il existe une arrete entre u et v. On note G_p cette famille de variables aléatoire.

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

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 a aucun V.

Question 2 Soit $U, V \subseteq \mathbb{N}$ fini, montrer que la probabilité que un $x \in \mathbb{N} \setminus (U \cup V)$ satisfaite la propriété de rado est non nul. En déduire l'existence d'un tel x avec probabilité 1.

Question 3 Montrer que si G et G' possède la propriété de Rado, alors ils sont isomorphes.

Question 4 Généraliser la question 3 pour montrer que la probabilité que G_p soit isomorphe a $G_{p'}$ est 1 si $0 < p, p' < 1$.

$(\Delta + 1)$-approx de Maximum independant set

Question de Cours Rapeller la définition d'un algorithme de monte-carlo et d'un algorithme de las-vegas.

Question 1 Montrer que si L est un langage hors-contexte alors $\overline{L} = \{\overline{w} : w \in L\}$ (l'ensemble des mirroirs de L) l'est aussi.

On considère le problème MIS suivant :

Entrée: Un graphe $G = (V, E)$ non orienté
Sortie: Un $I \subseteq V$ de cardinal maximal tel que $E \cap I^2 = \emptyset$

Pour $G = (V, E)$ un graphe, on note $\Delta(G) = \max_{v \in V} \deg(v)$.

Question 2 Donner la version problème de décision de MIS

Question 3 Donner un $(\Delta(G) + 1)$-approximation de MIS qui tourne en $O(|S|^2)$

On considère l'algorithme qui, tant que le graphe possède encore des arretes, prend un sommet de degrée non nul au hasard uniformément, le choisi et retire toutes les arretes relié à ce sommet.

Question 4 Quel est ce type d'algorithme ? Soit $G = (V, E)$ un graphe, on note V_f la variable aléatoire de l'ensemble choisi à la fin de l'execution de l'algorithme sur G. Montrer que

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

Coloration aléatoire

Question de Cours Montrer que les langages réguliers sont hors-contexte

Soit $k \in \mathbb{N}$ fixé. On regarde le problème du k-PATH :

Entrée: Un graphe $G = (V, E)$ non orienté
Sortie: Est-ce qu'il existe un chemin de longeur k ?

On définit une k-coloration comme une fonction $\mu : V \longrightarrow \{1, .., k\}$. On dit qu'un chemin est arc-en-ciel si tout les sommets sont de couleur différente.

Question 1 Quelle est la probabilité qu'un chemin de longeur k soit arc-en-ciel?

Question 2 Donner un algorithme de programmation dynamique qui, soit G et une k-coloration, vérifie si G possède un chemin arc-en-ciel en $O\big(2^k \times |E|\big)$.

Question 3 En déduire un algorithme à erreur unilateralle de meme probabilité que celle obtenu question 1. Quel type d'algorithme est-ce?

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

K-Centres

On fixe $V \subseteq \mathbb{R}^d$ un jeu de donnée. Pour $\vec{y} \in \mathbb{R}^d$ on définit $d(\vec{y}, V) = \max_{\vec{x} \in V} \|\vec{y} - \vec{x}\|$. On cherche à choisir k points de V tel qu'ils minimise $\max_{v \in V} d(v, X)$. On regarde donc le k-CENTRES suivant:

- **Entrée:** $v_1, ..., v_n \subseteq \mathbb{R}^d$ une liste de points finis
- **Sortie** : un $X \coloneqq \{x_1, ..., x_k\} \subseteq V$ qui minimise $\max_{v \in V} d(v, X)$

Question 1 On cherche à montrer que l'algorithme du cours de la question ne donne pas de bonne approximation du problème des k-centres. Pour cela on pose $k = 2$, $d = 1$ et on pose N points en 0, N points en 1 et un point en D. Montrer que pour des bonnes valeurs de N et D on peut avoir le ratio $\frac{\text{opt-k-moyennes}}{\text{opt-k-centre}}$ arbitrairement large.

Question 2 Proposer un algorithme glouton pour le problème des k-centres.

Question 3 On note $p_1, ..., p_k$ les k points retrourné par notre algorithme et $p_1^*, ..., p_k^*$ la solution optimale. On partitionne $V = \bigsqcup_{i \in \mathbb{N}} V_i$ avec V_i l'ensemble des points les plus proches de p_i^*. Montrer que

- Si $\forall i, V_i \cap \{p_1, ..., p_n\} \neq \emptyset$ alors on a une 2-approximation
- Sinon, on a aussi une 2-approximation

MPI: Déduction naturelle

Cours

- Arbre d'inférence, contexte Γ, séquent prouvable, variables libres, liées.
- Règle d'introduction et d'élimination de chaque opérateur $\wedge, \vee, \neg, \to, \forall, \exists$
- Syllogisme de barbara, modus ponens.
- Etre capable de montrer la correction des règles
- Logique minimale, intuitionniste (+ $\bot_e$), classique (HP)

Arbres à faire

Montrer les séquent suivant en logique minimale :

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

Arbres des lois de morgan

Montre les séquents suivant en logique minimale :

- $\neg(A \vee B) \vdash \neg A \wedge \neg B$
- $\neg A \wedge \neg B \vdash \neg(A \vee B)$
- $\neg A \vee \neg B \vdash \neg(A \wedge B)$

Montrer le séquent $\neg(A \wedge B) \vdash \neg A \vee \neg B$ en logique classique

Arbre des lois de morgan avec quantificateurs

Montrer les séquents suivants:

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

TODO: Pour les sens inverses, quelle logique il faut?

Equivalence entre les formules donnant la logique classique

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

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

Arbres nécésitant de deviner

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

Montrer que les séquents suivant sont prouvables en utilisant la logique intuitioniste:

- $\vdash \neg\neg(A \vee \neg A)$
- $\vdash \neg(A \leftrightarrow \neg A)$

Affaiblissement

Montrer que si le séquent $\Gamma \vdash A$ est démontrable alors $\Gamma, \Delta \vdash A$ l'est aussi.

XOR

On rajoute au langage des formules logique propositionelles l'opérateur $\oplus$ (prononcé XOR) d'arité 2, tel que pour η une valuation, on a $\eta \vDash \varphi \oplus \psi$ si et seulement si $\eta \vDash \varphi$ et $\eta \nvDash \psi$ ou bien $\eta \nvDash \varphi$ et $\eta \vDash \psi$.

1. Proposer 2 régles d'introduction et 1 règle d'élimination pour l'opérateur $\oplus$.
2. Montrer la correction de la règle, et utilliser la pour montrer la commutativité du $\oplus$.

Equivalence

On rajoute au langage des formules logique propositionelles l'opérateur d'équivalence $\leftrightarrow$ d'arité 2 avec les règles suivantes :

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

Soit μ une valuation, proposer une définition de $\mu \vDash A \leftrightarrow B$. Montrer la correction de la règle, et utilliser la pour montrer la commutativité de l'équivalence.

L'opérateur de Sheffer

On rajoute au langage des formules logique propositionelles *l'opérateur de Sheffer* $A \uparrow B$ tel que $A \uparrow B \equiv \neg(A \wedge B)$.

1. Montrer que $A \to B \equiv A \uparrow (B \uparrow B)$
2. Proposer une règle d'introduction et une règle d'élimination pour l'opérateur de Sheffer et montrer qu'elles sont correcte.
3. Montrer que pour chaque formule F il existe une formule F^* équivalente n'utillisant que l'opérateur de sheffer et $\top$ ou $\bot$.
4. Montrer que si $\vdash F$ est prouvable alors $\vdash F^*$ l'est aussi.

Sans implication

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

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

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

Complétude[76]

On cherche à montrer que si F une formule propositionelle est vraie pour tout modèle, alors $\vdash F$ est démontrable en logique classique.

Pour μ une valuation et φ une formule on pose

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

[76] Le magnifique livre de Mme. Galatée Hemery !!!! <3

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

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

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

Sans négation

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

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

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

Théorie déductive des graphes

On se donne le langage de la logique du premier ordre auquel on a rajouté une relation R d'arité 2. En plus des règles concernant la déduction naturelle, on rajoute les règle suivantes, indiquant que R est symétrique et n'est pas reflexif:

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

On peut interpréter alors une formule F de cette théorie comme une formule sur des graphes non orienté sans boucle : soit $G = (V, E)$ un graphe, on a $(x, y) \in E$ si et seulement si $R(x, y)$. Par exemple, la formule "$\forall x, R(s, x)$" est la formule qui indique que s est connecté à 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:

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

1. Montrer que si μ est une valuation, alors $\mu \vDash \neg F$ si et seulement si $\mu \vDash F^\perp$
2. Montrer par induction sur les formules F de la logique propositionelles 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 défini par induction les formules de la logique linéaire par le fait que :

- X et $\overline{X}$ sont des formules pour X une variable propositionelle
- $\varphi \otimes \psi, \varphi \parr \psi$ sont des formules de la logique linéiare pour φ, ψ deux formules de la logique linéaire

L'on note $\mathcal{L}$ l'ensemble des formules de la logique linéaire multiplicative. L'on munit ces formules des règles de déduction suivantes:

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

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

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

Soit $F \in \mathcal{L}$, on défini par induction le dual de F, noté $F^\perp$ par $(X \otimes Y)^\perp = X^\perp \ \parr \ Y^\perp$, $(X \parr Y)^\perp = X^\perp \otimes Y^\perp$, $X^\perp = \overline{X}$ et $\overline{X}^\perp = X$

2. Montrer que $\vdash F, F^\perp$ pour $F \in \mathcal{L}$.

3. $(*)$ Montrer que si $\vdash \Gamma, F$ et $\vdash \Delta, F^\perp$, alors $\vdash \Gamma, \Delta$

Contexte On dit qu'une règle déductive est *admissible* si en suppossant qu'il existe une preuve de toute ses prémisse, il existe une preuve de la conclusion. Une règle est admissible si elle est "inutile" au système mais est plus puissante qu'une règle dérivable (qui est elle juste qu'une macro). On a donc montré que la règle cut est admissible :

$$\frac{\vdash \Gamma, F \quad \vdash \Delta, F^\perp}{\vdash \Gamma, \Delta}\ \text{cut}$$

Et avec ça, on a montré le théorème bien compliqué que s'il existe une preuve utillisant la règle cut, il existe une sans.

Correspondance du système à la Hilbert[77]

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

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

$$\frac{}{\rhd_R F}\ \text{Ax} \qquad \text{pour } F \prec G, G \in R$$

$$\frac{\rhd_R A \qquad \rhd_R A \to B}{\rhd_R B} \to_e$$

Remarquer que comme R ne change jamais dans les règles, $\rhd_R$ en une sorte de système ou les axiomes sont les formules R avec uniquement la règle $\underset{e}{\to}$

On fixe $H = \{A \to B \to A, (A \to B \to C) \to (A \to B) \to (A \to C)\}$. On souhaite montrer que $\rhd_H X$ si et seulement si X est prouvable en logique minimale seulement avec les règles $\underset{i}{\to}$, $\underset{e}{\to}$, Ax.

[77] Partie 4 du projet de rocq du cours PRFA 2025/2026

1. Montrer le séquent $\rhd_H (A \to B) \to (A \to A)$ est prouvable.
2. Montrer que si $\rhd_X F$ est prouvable et que on a $F' \prec F$ et $X \subseteq X'$, alors $\rhd_{X'} F'$ l'est aussi. En déduire que $\rhd_H A \to A$ est prouvable avec une bonne instantiation de B dans la question précédente.

On fixe Γ un ensemble de formules propositionelle et $\Delta = \Gamma \cup H$

3. Montrer que si $\rhd_\Delta F$ est prouvable, alors $\Gamma \vdash F$ est prouvable en logique minmale seulement avec les règles $\underset{i}{\to}$, $\underset{e}{\to}$, Ax
4. Montrer les 2 faits suivants:
   - Si $\rhd_\Delta B$ est prouvable alors $\rhd_\Delta A \to B$ est prouvable
   - Si $\rhd_\Delta A \to B \to C$ et $\rhd_\Delta A \to B$ sont prouvable alors $\rhd_\Delta A \to C$ est prouvable
5. Montrer que si $\rhd_{\Delta \cup \{F\}} G$ est prouvable alors $\rhd_\Delta F \to G$ l'est aussi.
6. En déduire la réciproque de la question 3

On peut s'amuser à ajouter les autres opérateurs !

7. Proposer une formule $F^i_\wedge$ de la forme $[...] \to A \wedge B$ et deux formules $F^{e,g}_\wedge$ et $F^{e,d}_\wedge$ de la forme $A \wedge B \to [...]$ de telle sorte a ce que $\vdash \varphi$ soit démontrable en logique minimale avec les règles Ax, $\underset{i}{\to}$, $\underset{e}{\to}$, $\wedge_i$, $\wedge^g_e$, $\wedge^d_e$ si et seulement si $\rhd_{H \cup \left\{F^i_\wedge, F^{e,g}_\wedge, F^{e,d}_\wedge\right\}} \varphi$ est prouvable.
8. Meme question pour le $\vee$ et le $\neg$.

$\neg\neg$-Traduction de Godel-Kolmogorov[78]

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

$$(A \wedge B)^{\neg\neg} := A^{\neg\neg} \wedge B^{\neg\neg} \qquad X^{\neg\neg} := \neg\neg X \quad \text{pour } X \text{ une variable}$$
$$(A \to B)^{\neg\neg} := A^{\neg\neg} \to B^{\neg\neg} \qquad (\neg A)^{\neg\neg} := \neg A^{\neg\neg}$$
$$(A \vee B)^{\neg\neg} := \neg\neg(A^{\neg\neg} \vee B^{\neg\neg})$$

Pour $\Gamma = \{\Gamma_1, ..., \Gamma_n\}$ on note $\Gamma^{\neg\neg} := \{\Gamma_1^{\neg\neg}, ..., \Gamma_n^{\neg\neg}\}$. On note $\Gamma \vdash_i F$ s'il existe une preuve de $\Gamma \vdash F$ en logique intuitioniste (sans utilliser 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 seulements de $\wedge$ et $\to$ par celle respectant les règles suivantes. Attention, pour l'axiome, on n'a pas de Γ:

$$\frac{\Gamma, A \vdash' B}{\Gamma \vdash' A \to B} \underset{i}{\to} \qquad \frac{\Gamma \vdash' A \to B \qquad \Delta \vdash' A}{\Gamma \cup \Delta \vdash' B} \underset{e}{\to} \qquad \frac{}{F \vdash' F}\ \text{Ax}$$

$$\frac{\Gamma \vdash' A \qquad \Delta \vdash' B}{\Gamma \cup \Delta \vdash' A \wedge B} \wedge_i \qquad \frac{\Gamma \vdash' A \wedge B}{\Gamma \vdash' A} \wedge^g_e \qquad \frac{\Gamma \vdash' A \wedge B}{\Gamma \vdash' B} \wedge^d_e$$

1. Montrer que le séquent $\vdash F$ est prouvable en logique minimale ssi $\vdash' F$ est prouvable en logique multiplicative.
2. Proposer des règles pour l'introduction et l'élimination du $\vee$ pour la logique multiplicative et traiter leur cas pour la question 1.

[78] Tiré de https://www.lirmm.fr/~retore/LL/nonnon.pdf

Modèle de Kripke

Soit F une formule de variables libre $\mathcal{V}$, on apelle un **Modèle de Kripke** un arbre $T = (V, E)$ munit d'une fonction $\rho : V \longrightarrow \mathcal{P}(\mathcal{V})$ tel que pour tout enfant e d'un parent p, on ai $\rho(p) \subseteq \rho(e)$. On note $\mathcal{E}(v)$ les noeuds accesible depuis v (les petit-enfants). On aura toujours $v \in \mathcal{E}(v)$

On défini par induction le fait qu'une formule F est vérifié par un noeud $v \in V$ d'un modèle de Kripke $\mathcal{M}$ (noté $\mathcal{M}, v \vDash F$) par:
- $\mathcal{M}, v \vDash X$ si $X \in \rho(v)$
- $\mathcal{M}, v \vDash A \wedge B$ si $\mathcal{M}, v \vDash A$ et $\mathcal{M}, v \vDash B$
- $\mathcal{M}, v \vDash A \vee B$ si $\mathcal{M}, v \vDash A$ ou $\mathcal{M}, v \vDash B$
- $\mathcal{M}, v \vDash A \to B$ si pour tout $e \in \mathcal{E}(v)$, on a que $\mathcal{M}, e \vDash A$ implique $\mathcal{M}, e \vDash B$
- $\mathcal{M}, v \vDash \neg A$ si pour tout $e \in \mathcal{E}(v)$, on a pas $\mathcal{M}, e \vDash A$

On note $\mathcal{M} \vDash F$ pour dit que pour tout $v \in V$ on ai $\mathcal{M}, v \vDash F$, et l'on notera $\vDash F$ si pour tout $\mathcal{M}$ on ai $\mathcal{M} \vDash F$.

1. Donner un modèle de Kripke $\mathcal{M}$ tel que $\mathcal{M} \nvDash X \vee \neg X$
2. Montrer par induction que si $\Gamma \vdash F$ est prouvable en logique intuitioniste, alors pour tout modèle de Kripke tel que pour tout $A \in \Gamma$ on ai $\mathcal{M} \vDash A$, on aura nécessairement $\mathcal{M} \vDash F$
3. En déduire que l'on ne peut pas démontrer $A \vee \neg A$ ni $\neg\neg A \to A$ en logique intuistioniste.

Logique modale[79]

On cherche ici à représenter des formules déontiques, c'est-à-dire des formules représentant plusieurs situations analogues qui se déroule en meme temps et qui sont liées. Soit $\mathcal{V}$ un ensemble de variables propositionelle, on définit par induction l'ensemble des formules déontique $\mathcal{D}$ comme :
- Si $X \in \mathcal{V}$ alors $X \in \mathcal{D}$,
- Pour tout $\varphi, \psi \in \mathcal{D}$, les formules $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \to \psi)$ et $\neg\varphi$ sont des formules déontique,
- Si $\varphi \in \mathcal{D}$ alors $\Box\varphi \in \mathcal{D}$ et $\Diamond\, \varphi \in \mathcal{D}$.

La formule $\Box\varphi$ se lit "partout φ" et la formule $\Diamond\, \varphi$ se lit "quelquepart φ"

1. Donner la formule représentant "si partout $A \to B$, et si quelquepart A, alors quelquepart B".

On cherche maintenant à attribuer une valeur de vérité à de telle formules. Soit F une formule de variables propositionnelles V, on définit un **modèle de Kripke de F** comme un couple $(\mu_1, ..., \mu_n)$ tel que pour tout $i \leq n$ on ai $\mu_i : V \longrightarrow \{\top, \bot\}$ une valuation. Soit $\mathcal{M} = (\mu_1, ..., \mu_n)$ un modèle de Kripke, on définit $\mathcal{M} \vDash_i \varphi$ par induction :

$$
\begin{aligned}
&\mathcal{M} \vDash_i A && \text{si } A \in V \text{ et } \mu_i(A) = \top \\
&\mathcal{M} \vDash_i \varphi \wedge \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ et } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \varphi \vee \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ ou } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \varphi \to \psi && \text{si } \mathcal{M} \vDash_i \varphi \text{ implique } \mathcal{M} \vDash_i \psi \\
&\mathcal{M} \vDash_i \neg\varphi && \text{si } \mathcal{M} \vDash_i \varphi \text{ est faux} \\
&\mathcal{M} \vDash_i \Diamond\, \varphi && \text{si } \exists j, \mathcal{M} \vDash_j \varphi \\
&\mathcal{M} \vDash_i \Box\varphi && \text{si } \forall j, \mathcal{M} \vDash_j \varphi
\end{aligned}
$$

Dans ce cas on dit que $\mathcal{M}$ *satisfait* φ au i-ème monde. Si $\mathcal{M} = (\mu_1, ..., \mu_n)$ et que pour tout i, $\mathcal{M} \underset{i}{\vDash} \varphi$ on dira que $\mathcal{M}$ satisfait φ. Si pour tout modèle $\mathcal{M}$ de Kripke on a $\mathcal{M} \vDash \varphi$, on dit que φ est une *tautologie*, et on note cela $\vDash \varphi$.

[79] Centrale Info 2026

2. Pour chacune des formules φ suivante, indiquer s'il existe un modèle la satisfaisant, s'il existe un modèle satisfaisant $\neg\varphi$ et si elles sont des tautologies :

$\varphi_1 = "A \wedge \Diamond \neg A"$ $\varphi_2 = "\Box(A \wedge \Diamond \neg A)"$ $\varphi_3 = "\Box(\Box A \to \Box A)"$

3. Montrer que $\mathcal{M} \vDash_i \Diamond P$ est vrai si et seulement $\mathcal{M} \vDash_i \neg(\Box\neg P)$ est vrai.
4. En déduire un algorithme qui transforme une fomule déontique en une formule équivalente n'utillisant pas $\Diamond$.

On étudie maintenant que les formules déontiques sans $\Diamond$. On rajoute aux règles de la déduction naturelle de la logique classique les 3 règles suivantes :

$$\frac{\vdash \varphi}{\vdash \Box\varphi} D_1 \qquad \frac{\Gamma \vdash \Box(\varphi \to \psi)}{\Gamma \vdash \Box\varphi \to \Box\psi} D_2 \qquad \frac{\Gamma \vdash \Box\varphi}{\Gamma \vdash \neg\Box\neg\varphi} D_3$$

5. Montrer que $\vdash \neg(\Box A \wedge \Box\neg A)$
6. Montrer que $\vdash \Box(A \wedge B) \leftrightarrow (\Box A \wedge \Box B)$
7. Montrer que le système est correct, c'est-à-dire que si $\vdash \varphi$ est démontrable pour φ une formule sans $\Diamond$ alors $\vDash \varphi$ est vrai.

MPI: Classes de Complexités

Cours

- Problème de décision, d'optimisation, de maximisation/minimisation. Comment transformer un problème d'optimisation en un probblème de décision.
- Taille d'une instance.
- Définition d'une classe de complexité. Définition de P, NP, NP-Dur, NP-Complet
- Réduction polynomiale
- k-SAT, SAT, certificat

Question de Cours

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

Max 2-SAT

On consdière le problème d'optimisation suivant :

Entrée: φ une formule en FNC **Sortie:** Une valuation φ qui maximise le nombre de clauses satisfaites

1. Donner la version problème de décision
2. Montrer que si k est le nombre de clauses le problème deviens polynomial
3. Montrer que la version décisionnel du problème est NP-Complet

SUBSET-SUM

On considère le problème SUBSET-SUM suivant :

- **Entrée** $n \in \mathbb{N}$ et $a_1, a_2, ..., a_n, S \in \mathbb{N}$
- **Sortie:** Est-ce qu'il existe $I \subseteq [\![1; n]\!]$ tel que $\sum_{i \in I} a_i = S$

1. Pour les suites suivante de $(a_n)_n$ et les valeurs de S suivantes, indiquer si le problème est satisfiable ou non :
   - $(a_n)_n = (1, 2, 4, 8, 16), S = 7$
   - $(a_n)_n = (31, 24, 2, 43, 12, 12, 18), S = 29$
   - $(a_n)_n = (1001, 1010, 101, 100, 11), S = 1111$
2. Montrer que le problème SOMME est dans la classe NP
3. Donner un algorithme qui en $O(nS)$ résoud le problème

On cherche à prouver que le problème précédent est NP-Complet. Pour cela on considère une généralisation du problème sur des k-uplets intitulé SUBSET-SUM-VECT:

- **Entrée** $k, n \in \mathbb{N}$ et $a_1, a_2, ..., a_n, S \in \mathbb{N}^k$
- **Sortie:** Est-ce qu'il existe $I \subseteq [\![1; n]\!]$ tel que $\sum_{i \in I} a_i = S$

4. Montrer que on peut réduire le problème **SOMME-VECT** au problème **SOMME**
5. Montrer que **SOMME-VECT** est NP-Complet à partir de 3-SAT.
6. Est-ce que avec la question 3 on a démontré que P=NP? Justifier.

INDEPENDANT-SET et CLIQUE

On considère les deux problèmes CLIQUE-MAX et INDEPENDANT-SET suivant:

CLIQUE:

- **Entrée:** Un graphe $G = (V, E)$ non orienté
- **Sortie:** Le plus grand $S \subseteq V$ en cardinal tel que $V^2 \subseteq E$

INDEPENDANT-SET :
- **Entrée:** Un graphe $G = (V, E)$ non orienté et un $k \in \mathbb{N}$
- **Sortie:** Est-ce qu'il existe $S \subseteq V$ tel que $S^2 \cap E = \emptyset$ avec $|S| \geq k$?

1. Donner la version problème de décision de CLIQUE. Montrer que elle est NP.
2. Donner une réduction polynomiale de INDEPENDANT-SET dans la version porblème de décision de CLIQUE.
3. En réduisant depuis 3-SAT, et en considérant un graphe avec $3k$ noeuds avec k le nombre de clauses, montrer que CLIQUE est NP-Complet.

MIN-COLOR-PATH

On considère le problème MIN-COLOR-PATH suivant :
- **Entrée:** $G = (V, E)$ un graphe, $s, t \in V$ deux sommets et $c : V \longrightarrow \mathbb{N}$ une coloration
- **Sortie:** Un chemin $s = x_1, .., x_n$ de s à t qui minimise $\text{Card}(\{c(x_1), ..., c(x_n)\})$

1. Donner la version problème de décision, et montrer qu'elle est NP.
2. Donner un algorithme pour résoudre le problème si G est un cycle.
3. Montrer que le problème de décision associé à MIN-COLOR-PATH est NP-Complet.

On suppose maintenant que l'on rajoute la condition $|\text{Im}(c)| \leq k$ avec $k \in \mathbb{N}$ fixé.

4. Montrer maintenant que le problème est P.

On considère l'algorithme qui prend des couleurs uniformément au hasard dans $\text{Im}(c)$ jusqu'à ce qu'il existe un chemin de s à t

5. Quel est ce type d'algorithme? Proposer des exemples arbitrairement grand ou cet algorithme renvoie une solution d'éspérence en $O(n)$ au lieu de $O(1)$.

DOMINATING-SET

Soit $G = (V, E)$, on dit que $S \subseteq V$ est un *ensemble dominant* si tout les sommets de V sont soit dans S, soit voisin d'un sommet de S. On note $\gamma(G)$ le cardinal minimum d'un ensemble dominant de G.

On considère le problème DOMINATING-SET suivant :
- **Entrée:** $G = (V, E)$ un graphe
- **Sortie:** $\gamma(G)$

1. Donner la version problème de décision de DOMINATING-SET. Montrer qu'elle est NP.
2. Montrer que $\gamma(G) \leq |V|/2$
3. Montrer que la version problème de décision de DOMINATING-SET est NP-Complet.
4. Montrer que si l'on restrain l'entrée du problème à des arbres, alors le problème est dans P.
5. Pour G un graphe, on note Δ_G son degré maximal. Donner une Δ_G-approximation de DOMINATING-SET.
6. Donner similairement une $\log(|V|)$-approximation. Est-ce qu'une des approximations fait les deux?

Langages NP-Complets sur $\Sigma = \{a\}$[80]

Un langage L est dit unaire si c'ets un langage sur un alphabet à une lettre, donc si $L \subseteq \{a\}^*$. L est dit *NP-Difficile* si le problème de tester si $x \in L$ est NP-Difficile. Dans ce problème, la taille de l'entrée est $|x|$.

[80] Mallory Marin, doc de colles

1. On note SAT^+ l'ensemble des entrée positives au problème SAT et S. Montrer que L est NP-Difficile ssi il existe une fonction f de complexité polynomiale telle que

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

On cherche à montrer le théorème suivant

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

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

2. On note $\varphi_\top(x_2, ..., x_n) \coloneqq \varphi(\top, x_1, ..., x_n)$ et $\varphi_\bot(x_2, ..., x_n) \coloneqq \varphi(\bot, x_1, ..., x_n)$. Montrer que

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

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

3. Soit φ une formule à n variables. Donner un algorithme qui, pour tout $i \leq n$, construit un ensemble de formules Φ_i tel que toute formule de Φ_i contient $n - i$ variables et $\varphi \in \mathrm{SAT}^+ \iff \exists \varphi \in \Phi_i, \varphi \in \mathrm{SAT}^+$
4. Montrer qu'il existe un polynome p tel que $\sum_{\varphi \in \Phi_i} |\varphi| \leq p(|\varphi|)$
5. En déduire que $\mathrm{P} = \mathrm{NP}$, et donc le théorème de Berman.

Intersection d'automates[81]

On considère le problème NONEMPTY-REGULAR-INTERSECTION suivant:

Entrée: $A_1, ..., A_n$ une liste d'automate sur $\Sigma = \{a, b\}$
Sortie Est-ce que

$$\bigcap_{i \leq n} L(A_i) \neq \emptyset$$

1. Montrer que NONEMPTY-REGULAR-INTERSECTION est NP
2. Donner une instance du problème tel que le plus petit mot dans l'intersection est de longeur exponentielle en la taille de l'entrée.
3. Montrer que si $\Sigma = \{a\}$ alors le problème est polynomial.
4. Montrer que NONEMPTY-REGULAR-INTERSECTION est NP-Complet à partir de 3-SAT

Sheffer

On définit l'opréateur de sheffer $X \uparrow Y \coloneqq \neg(X \wedge Y)$ et l'opérateur $X \oplus Y \coloneqq (X \vee Y) \wedge (\neg X \vee \neg Y)$. 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 $\uparrow$.

[81] Algo 1 ENS Lyon TD 8

MPI: Calculabilité

Cours

- Problème décidable, indécidable
- Le problème de l'arret est indécidable
- Réduction calculable
- Problèmes semi-décidable, co-semi-décidable (HP).
- Fonction calculable (HP)

Question de Cours

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

Indécidabilité

Montrer que les problèmes sont indécidables:

- **ε-Arret: Entrée:** Un programme P, **Sortie:** Est-ce que $P(\varepsilon)$ termine?
- **42-Arret: Entrée:** Un programme P, **Sortie:** Est-ce que $P(42)$ termine?
- **$\neg$-Arret: Entrée:** Un programme P et une entrée w, **Sortie:** Est-ce que $P(w)$ ne termine pas?
- **Equivalence: Entrée:** Deux programmes P, Q prenant en entrée des naturels, **Sortie:** Est-ce que pour tout $n \in \mathbb{N}$, $P(n)$ termine ssi $Q(n)$ termine?
- **Arret-$\exists$: Entrée:** Un programme P prenant en entrée un naturel, **Sortie:** Est-ce qu'il existe un n tel que $P(n)$ termine?
- **Arret-$\forall$: Entrée:** Un programme P prenant en entrée un naturel, **Sortie:** Est-ce que pour tout $n \in \mathbb{N}$ on a $P(n)$ qui termine?

Entrée bornée

Montrer que le problème suivant est indécidable:

- **Entrée:** Une fonction calculable $P : \mathbb{N} \to \mathbb{N}$ qui termine toujours.
- **Sortie:** Est-ce qu'il existe un n tel que $P(n) = 0$?

L'indécidable est partout

Soit S un ensemble de string, on considère le problème P_S suivant:

- **Entrée:** s une chaine de charactères
- **Sortie:** Est-ce que $n \in S$?

1. Montrer qu'il existe des S tel que P_S est indécidable.
2. Montrer que si C est infini, alors il existe $K \subseteq C$ tel que P_K est indécidable. Est-ce vrai si C est fini?

Complexité et décidabilité[82]

Montrer que pour toute fonction calculable $g : \text{String} \to \mathbb{N}$, il existe une fonction calculable $f : \text{String} \to \text{char}$ 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'execution de ce code affiche le `string` w.

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

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 -> bool` déterministe et qui **termine toujours**. On défini donc le type suivant en OCaml :

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

1. Donner un ensemble $A \subseteq \mathbb{N}$ que l'on ne pourra pas représenter avec notre structure. Le nombre d'ensembles non représentable est-t'il fini ? Dénombrable ? Indénombrable ?
2. Montrer que la fonction `val is_empty : int set -> bool` qui à un ensemble donné indique s'il est vide ou non n'est pas calculable.
3. Soit `val P: int set set`, montrer qu'il existe un $N \in \mathbb{N}$ tel que pour tout `val x: int set`, dans le calcul de `P x`, `P` n'évalue `x` que sur entrées inférieure à N. Est-ce vrai avec `val P: (int -> int) set` ?
4. ($*$) Montrer que la fonction `val is_empty : (int set set) -> int set` qui à un `int set set` donné indique s'il est vide ou non est calculable.