Exercices d'informatique
MP2I - MPI

Milan Gonzalez-Thauvin, ENS de Lyon

2022 - 2023

Table des matières

1 Méthodes de programmation

1.1 Entrainement OCaml récursif

Exponentiation
On cherche à implémenter une fonction puissance $(x, n) \mapsto x^n$. On ne travaille qu'avec des entiers.

1. Implémenter en Ocaml un algorithme naïf. La signature sera `expo : int -> int -> int`. Donner sa complexité et prouver sa correction et sa terminaison.
2. Implémenter en Ocaml un algorithme plus efficace utilisant la parité de l'exposant. La signature sera `expo : int -> int -> int`. Prouver sa correction et sa terminaison. Combien de multiplications sont éffectuées lorsque n est une puissance de deux ?
3. Implémenter une fonction `quarante_deux : int -> int` qui calcule x^{42}.

Expressions *Source : [5]*
On donne un type `type expr = I of int | Plus of expr*expr | Mult of expr*expr`

1. Représenter $(1 + 2) * 6$ avec ce type en OCaml, ainsi que sous forme d'arbre.
2. Écrire une fonction d'évaluation d'expression `eval : expr -> int`. Donner une preuve (rapide mais rigoureuse) de correction.
3. On introduit la notation polonaise inverse (NPI) : Il s'agit d'une notation pour représenter des formules arithmétiques où les opérateurs se situent juste après les opérandes (et pas entre, comme habituellement). On se limitera ici aux entiers, à l'addition et à la multiplication. Calculer $(3)((1)(12)+)\times$ puis écrire $(1 + 2) * (8 + 7)$ en NPI.
4. Avec un type `type token = I of int | Plus | Mult` et `type NPI = token list` écrire une fonction `eval : NPI -> int` qui évalue une expression en NPI. On admet pour cela le résultat de la question suivante.
5. Prouver que des parenthèses sont inutiles en notation NPI si les entiers sont vus comme des objets insécables. Autrement dit, les seules parenthèses nécessaires seraient celles qui entourent les différents chiffres d'un même entier.

Fibonacci récursif terminal
La suite de Fibonacci est définie par la relation de récurrence suivante :
$F_0 = 0, \quad F_1 = 1$, et $F_n = F_{n-1} + F_{n-2}$ pour $n \geq 2$.

1. Donner une fonction OCaml `fibo: int -> int` qui calcule un terme de la suite à partir de son indice. Donner une intuition sur la complexité de votre fonction.
2. Donner une fonction récursive terminale qui calcule elle aussi les termes de la suite de Fibonacci. Analyser sa complexité.

On souhaite étendre cette fonction à toute suite récurrente linéaire. Ainsi, on souhaite pouvoir calculer n'importe quel terme u_n pour u définie par la relation de récurrence

$$\forall n \geq 0 \quad u_{n+p} = a_0 u_n + a_1 u_{n+1} + \cdots + a_{p-1} u_{n+p-1}$$

avec $p \in \mathbb{N}^*$ et a_i et u_i connus pour $i \in [0, p-1]$.

3. Vérifier que la suite de Fibonacci est un bien un cas particulier de la définition ci dessus.
4. Coder en OCaml une fonction `srl: int list -> int list -> int -> int` qui calcule les termes d'une suite récurrente linéaire définie par ses a_i et u_i pour $i \in [0, p-1]$, si possible en récursif terminal. Si ce n'est pas possible, expliquer pourquoi.
   Remarque : pourquoi mettre les arguments dans cet ordre et pas dans l'ordre inverse ?
5. Quelle est la complexité de votre fonction ?

1.2 Entrainement OCaml impératif

Chiffrement de César
On appelle chiffrement de César la façon de coder un message en décalant les lettres d'une certaine quantité. On considère ici uniquement les 26 lettres de l'alphabet plus l'espace (soit 27 caractères) que l'on ordonne alphabétiquement et circulairement ($A \to B \to \ldots \to Z \to \square \to A \to \ldots$). Par exemple, "CHAMPOLLION" serait codé "HMFRUTQQNTS" avec un décalage de 5.

1. Donner un programme qui prend en entrée un entier (le décalage) et une chaine de caractère, et qui renvoie le message codé. Votre fonction aura une signature `encode : int -> string -> string`. On utilisera pour cela les fonctions `Char.code : char -> int` et `Char.chr : int -> char` qui associent un caractère à son code ASCII (ou réciproquement), par exemple 65 pour `'A'`, 66 pour `'B'`, etc... Ensuite, implémenter (intelligemment) la fonction de décodage.
2. Avez vous une idée de comment déchiffrer un très long message (écrit en français) sans posséder le décalage utilisé ? (et sans tester les 27 possibilités)

Clavier cassé *Source : NWERC22*
On possède un clavier cassé qui agit de la façon suivante : il écrit soit le caractère sur lequel l'utilisateur a appuyé, soit deux fois ce même caractère. De plus, il alterne de comportement entre chaque appui : si il fonctionne correctement à un moment, il doublera la saisie la fois d'après. On veut un programme OCaml qui détermine si une certaine chaine de caractère aurait pu être écrite avec ce clavier. Ce programme aura pour signature `casse : string -> bool`. Par exemple, `foot` ou encore `MMA` auraient pu être écrit avec le clavier cassé, mais pas `judo`. On pourra admettre que seules les 26 lettres de l'alphabet et le caractère espace sont présent dans la chaine en question. On interdit ici l'utilisation de la récursivité.

2 Structures de données

2.1 Représentation d'objets

Représentation des entiers *Source : [1]*
On s'intéresse ici à la représentation des entiers dans différentes bases. On ne demande de ne coder aucune fonction qui se trouve déjà dans le module `List` de OCaml (en particulier `List.rev` ou `List.map`).

1. Comment représenter un entier dans une base donnée ? Implémenter en OCaml un type `repr` qui contiendra la représentation d'un entier et la base utilisée.
2. Écrire deux fonctions `repr_to_int : repr -> int` et `int_to_repr : int -> int -> repr` qui passe du type `int` de OCaml au votre.
3. On suppose avoir écrit plus haut une fonction `hex_digit_to_int : char -> int` qui transforme un digit hexadécimal en entier entre 0 et 15. Implémenter une fonction `hex_to_bin : char list -> repr` qui transforme une représentation hexadécimal (convention : digit de poids fort en premier) en binaire avec votre représentation. Indication : Ne pas chercher trop loin, cela peut se faire en une ligne avec les questions précédentes et une bonne utilisation du fonctionnel).
4. Écrire une fonction qui additionne deux nombres dans une même base, sans utiliser les fonctions précédentes.
5. (Bonus) Écrire une fonction qui multiplie deux nombres dans une même base (on s'autorise la fonction de la question 4).

Représentation d'ensembles $\star$ *Inspiré de : [6]*
On souhaite trouver une structure de donnée pour représenter des sous-ensembles d'un ensemble à n éléments. On se limite pour l'instant aux ensembles contenus dans $[\![0, n-1]\!]$. Une idée serait de représenter un sous-ensemble X par un seul entier dans $[\![0, 2^n - 1]\!]$: un entier k appartient à X ssi le k-ème bit de X est 1.

1. On pose $n = 10$. Combien "vaut" $\{4, 2\}$?

2. Que représentent les $X = 2^k$ pour $k \in [\![0, n-1]\!]$.
3. MP2I On cherche maintenant à utiliser astucieusement cette structure de donnée pour implémenter des opérations élémentaires sur les sous-ensembles. Par exemple, montrer que si X et Y représentent des sous-ensembles, alors $X|Y$ représente $X \cup Y$ (avec | l'opérateur binaire *ou* bit-à-bit). De la même façon, trouver quelle opération effectuer pour savoir si $X \subset Y$. Enfin montrer que $-X\&X$ représente $\{\min X\}$ (avec & l'opérateur binaire *et* bit-à-bit). *Indication : pour la dernière partie, penser au complément à deux.*

On remarque désormais que cette structure fonctionne pour n'importe quel ensemble de n éléments : il suffit de les ordonner (dans une liste par exemple) et de changer "k appartient à X" par "le k-ème élément appartient à X".

4. Donner une fonction OCaml `int_to_set: 'a list -> int -> 'a list` qui à partir d'un ensemble de base et d'un entier X génère le sous ensemble associé.
5. Donner une fonction qui résout le problème du Subset Sum : à partir d'un ensemble de n entiers positifs et d'un entier cible v, déterminer si il existe un sous-ensemble tel que la somme soit égale à v. Cette fonction est-elle récursive terminale ?
6. Recommencer sans la structure de donnée présentée. Votre fonction est-elle récursive terminale ? Comparer les complexités.

2.2 Arbres enracinés

Requêtes ⋆ *D'après : NWERC22*

On considère un graphe à n sommets (numérotés de 0 à $n-1$) initialement circulaire (il existe une arête entre i et $i+1$ pour tout $0 \leq i < n-1$ ainsi qu'entre $n-1$ et 0). Donner un algorithme qui, à partir de requêtes chronologiques de la forme :

1. `suppression de l'arête entre` i `et` $i+1$ `(modulo` n`)`
2. `ajout de l'arête entre` i `et` $i+1$ `(modulo` n`)`
3. i `et` j `sont-ils dans la même composante connexe ?`

répond correctement à chaque requête de type 3. On supposera que toutes les requêtes sont licites (pas de double arête ou pas de suppression d'une arête inexistante par exemple). On demande une complexité temporelle moyenne après chaque requette en $\mathcal{O}(\log(n))$. Implémenter cet algorithme en OCaml après validation.

Profondeur moyenne d'un ABR ⋆ ⋆ *Source : gtl.csa.iisc.ac.in/dsa/node92.html*

On cherche à étudier le comportement d'un ABR lors d'une utilisation *générique*. On va s'intéresser plus particulièrement à la profondeur moyenne des noeuds lorsque l'ordre des n éléments insérés est aléatoire : ils sont tirés uniformément et sans remise dans un ensemble très grand, par exemple $[|-2^{32}; 2^{32}-1|]$.

1. Montrer que le problème se ramène à l'insertion (dans l'ordre) des éléments $\sigma(1), \sigma(2), \ldots, \sigma(n)$ pour σ une permutation tiré uniformément dans $\mathfrak{S}_n$.
2. Soit $P(n)$ la profondeur moyenne d'un noeud d'un ABR comportant n éléments (tirés aléatoirement comme décrit en question 1). Soit i le premier élément inséré. Cet élément sera donc la racine de l'arbre. Montrer que la profondeur moyenne d'un noeud d'un ABR dont i est la racine est
$$P(n,i) = 1 + \frac{i-1}{n}P(i-1) + \frac{n-i}{n}P(n-i)$$
3. Montrer que la profondeur moyenne $P(n)$ est
$$P(n) = 1 + \frac{2}{n^2}\sum_{i=0}^{n-1} iP(i)$$
4. Montrer que
$$\sum_{i=1}^{n-1} i\log(i) \leq \frac{n^2}{2}\log(n) - \frac{n^2}{8}$$
Indication : on pourra séparer la somme en deux parties

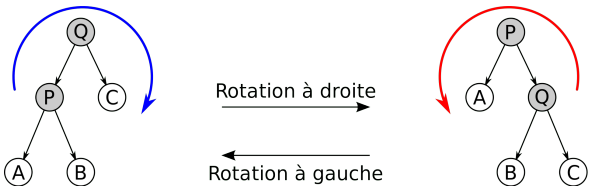
5. Montrer par récurence forte sur n que $P(n) \leq 1 + 4\log(n)$

Arbres AVL ⋆⋆ *Source : [2]*

Un arbre AVL est un arbre binaire de recherche dans lequel chaque noeud stocke sa hauteur (plus long chemin vers une feuille) et tel que la différence de hauteur entre les deux sous-arbres de n'importe quel noeud est au plus 1.

1. Soit A un AVL avec n éléments. Montrer que la hauteur de A est en $\mathcal{O}(\log(n))$
2. Soit A' l'arbre A auquel on a ajouté l'élément x comme dans un ABR. A' est il forcément un ABR ? Un AVL ?
3. Soit D l'ensemble des noeuds de A' qui ne vérifient pas la proprété d'equilibrage d'un AVL. Montrer que la différence de hauteur des deux sous arbres de tout noeud de D est exactement 2.

La rotation d'un ABR est une opération qui permet de changer la structure d'un tel arbre tout en préservant la propriété dichotomique fondamentale. Concrètement, il s'agit de choisir un noeud de l'arbre (dit "racine") et d'appliquer les changements décrits dans l'image ci-dessous (P et Q représentent des noeuds et A, B et C des sous-arbres).



(image : wikipedia)

4. Montrer que (comme attendu) une rotation appliquée à un ABR (en prenant n'importe quel noeud comme racine) reste un ABR.
5. Montrer que A' peut être transformé en un arbre AVL en appliquant deux rotations.

2.3 Structures complexes

File fonctionnelle *Source : [5]*

On souhaite construire un type `'a file` représantant une file d'élément `'a`. Une file est une structure de donnée qui permet de stocker des éléments puis de les retirer, de telle sorte que les éléments sortent de la file dans le même ordre que lorsqu'ils sont rentrés (*First In First Out*), un peu comme dans une file d'attente. Pour cela, on utilisera le paradigme de programmation fonctionnelle Implémenter le type file et les fonctions suivantes :

```
vide: 'a file
taille : 'a file -> int
est_vide : 'a file -> bool
ajoute : 'a file -> 'a -> bool
pop : 'a file -> 'a
iter: 'a file -> ('a -> unit) -> 'a file
```

On pourra représenter une file sous forme de deux listes qui feront office de début et de fin de la file : on enfile un élément en l'ajoutant en tête de la première liste et on défile un élément en retirant la tête de la deuxième liste (si elle est non vide, sinon on transvase la première dans la deuxième).

File impérative *Source : [5]*

On souhaite construire un type `'a file` représentant une file d'élément de type `'a`. Une file est une structure de donnée de type FIFO (First In First Out) : on veut faire sortir les éléments dans l'ordre dans lequel ils sont arrivés.

L'idée est d'implémenter cette file sous forme d'un tableau circulaire de taille fixe (on suppose connue à l'avance le nombre maximal que la file contiendra). L'état de la file ne sera donné que par deux entiers : l'indice du premier élément de la file et le nombre d'élément de la file. On utilisera sans surprise le paradigme de programmation impératif. Le type `'a file` sera le suivant :

```
type 'a file = {mutable debut : int; mutable taille : int; mutable tab : 'a array}
```

Implémenter les fonctions suivantes :

```
init : 'a -> int -> 'a file'
```

(initialise une file vide, l'élément de type `'a` va juste servir à initialiser le tableau, et l'entier représente la capacité maximale de la file)

```
capacite : 'a file -> int (donne la capacite max de la file)
est_pleine : 'a file -> bool (verifie si la file a atteint cette capacite max ou non)
taille : 'a file -> int (donne la taille de la file)
est_vide : 'a file -> bool (verifie si la file est vide)
ajoute : 'a file -> 'a -> unit (ajoute un element a la file)
pop : 'a file -> 'a (retire un element de la file et le renvoie)
```

Unir et trouver ⋆⋆ MPI *Source : [2]*

On définit en `OCaml` le type `unir_trouver` et les fonctions associées ainsi :

```
type unir_trouver = int array ;;

let initialisation n =
  Array.init n (fun x -> x) ;;

(* TODO: val trouver : unir_trouver -> int -> int = <fun> *)

let meme_classe ut x y =
  (trouver ut x = trouver ut y) ;;

(* TODO: val unir : unir_trouver -> int -> int -> unit = <fun> *)
```

1. Donner une implémentation possible des deux fonctions manquantes. Aucune optimisation n'est demandée.
2. Donner un jeu de tests qui permet de vérifier que la structure d'unir et trouver a été codée correctement.
3. Donner les complexités (dans le pire cas) des fonctions `unir` et `trouver`

On utilise une heuristique, l'union-by-rank. On associe à chaque element x un entier $\mathrm{rk}(x)$ qui est une borne supérieure de la hauteur de l'arbre enraciné en ce nœud.

4. Effectuer les modifications nécessaires au code précédent pour inclure cette heuristique.
5. Donner une autre heuristique pour améliorer la structure d'unir et trouver.
6. Montrer que le rang d'un nœud est strictement inférieur à celui de son père (sauf si il représente sa classe). De plus, le rang d'un nœud est inférieur ou égal à $\lfloor \log(m+1) \rfloor$, où m est le nombre d'opérations unir effectuées.
7. En déduire la nouvelle complexité des fonctions `unir` et `trouver`.

3 Algorithmique

3.1 Analyse de correction et de complexité

Domination *Source : [3]*

Soit c_n une suite telle que $c_n = \mathcal{O}(n^\alpha)$. Donner une suite (raisonnable) qui domine $\sum_{k=1}^{n} c_k$.

Recherche linéaire ⋆ *Source : [3]*

On se trouve face à un mur infini à droite et à gauche. Il existe une unique porte qui permet de traverser le mur, mais on ne sait pas de quel coté elle se trouve. Quelle stratégie adopter pour atteindre la porte en marchant $\mathcal{O}(d)$ mètres, où d est la distance entre la porte et notre position initiale. En d'autres termes, exhiber une constante C telle que votre stratégie trouve la porte en moins de $C \cdot d$ mètres (pour d assez grand).

Tombouctou ⋆ *Source : [2]*

Bob roule sur l'autoroute en voiture. Il part de Grenoble (kilomètre 0) avec le plein d'essence et va à

Tombouctou. La i-ème station-service est au kilomètre k_i. Son autonomie (plein d'essence) est de d kilomètres, et Bob veut s'arrêter un minimum de fois. On suppose que toutes les stations essence ont de l'essence. La dernière station se situe à Tombouctou.

1. Trouver un algorithme qui affiche les indices des stations dans lesquelles s'arrêter. Donner le paradigme utilisé et le décrire brièvement.
2. Prouver rigoureusement l'algorithme.

3.2 Paradigmes avancés

Eléments dominants ⋆ *Source : [2]*

Soit S un ensemble de n points du plan $(x_i, y_i), 1 \leq i \leq n$, à coordonnées entières distinctes deux à deux. On dit qu'un point (x, y) domine un point (x', y') si $x > x'$ et $y > y'$. Un point de S est dit maximal s'il n'est dominé par aucun point de S.

1. La relation binaire de domination est-elle une relation d'ordre ? Si oui, justifier et préciser l'ensemble, si non, la modifier pour qu'elle le devienne, puis mêmes questions.
2. Proposer un algorithme efficace pour déterminer les points maximaux de S. On demande une complexité en $\mathcal{O}(n \log n)$.

Dichotomie ⋆ *Inspiré de : [2]*

Dans cet exercice, on demande des programmes en OCaml ou des algorithmes en pseudo-code (au choix).

1. Écrire une fonction `bisect: (int -> bool) -> int -> int` qui étant donné une fonction f (premier argument) et une borne N (deuxième argument), renvoie le plus petit entier i tel que $f(i) = \top$ et $i \leq N$. On impose à f d'être croissante (avec la convention $\bot < \top$), définie sur $\mathbb{N}$ et telle que $f(\llbracket 0; N \rrbracket) = \{\bot, \top\}$. On demande une fonction en $\mathcal{O}(\log N)$ et récursive terminale. Prouver la correction de la fonction.
2. Donner un algorithme qui résout le problème suivant : étant donné n objets de poids respectifs $(p_1, \ldots, p_n)$ et m sacs (de taille infini), trouver comment répartir les objets dans les sacs en minimisant le poids sac le plus lourd. On impose en plus de cela que l'ordre des objets soit respecté (on ne peut pas placer l'objet j avant l'objet i si $i < j$) et idem pour les sacs (une fois qu'on a fini de remplir un sac, celui-ci ne sera plus utilisable par la suite). La sortie sera un entier : le poids du sac le plus lourd. On demande une complexité en $\mathcal{O}(n \log P)$ avec $P = \sum_{i=1}^{n} p_i$.

Plus grand carré ⋆

On se donne un quadrillage de taille $N \times M$ dans lequel certaines cases sont grisées. On veut connaitre la taille du plus grand carré blanc, c'est à dire qui ne recouvre aucune des cases grisées. Donner un algorithme qui détermine cette taille à partir de N, M et p coordonnées (celles des cases noires grisées, sans doublons) en complexité $\mathcal{O}(N \times M)$.

Des rectangles et des carrés ⋆

On part d'un rectangle de taille $n \times m$ et on cherche à découper ce rectangle en plusieurs carrés (pas forcément de même taille). Pour cela, on s'autorise uniquement à un couper un réctangle en deux (soit verticalement soit horizontalement). Combien de découpes sont nécéssaires au minimum ? Par exemple, pour un rectangle de taille 4×3, 3 découpes sont nécéssaires.

Sous suite croissante ⋆ ⋆ *Source : [6]*

On souhaite connaitre la taille maximale d'une sous suite croissante d'une suite donnée de taille n. Par exemple, la suite $(6, 1, 4, 9, 5, 11)$ admet $(1, 4, 9, 11)$ comme sous suite croissante et n'admet pas de sous suite croissante de taille 5. La taille recherchée est donc 4.

1. Proposer un algorithme qui résout le problème en $\mathcal{O}(n^2)$ et prouver sa correction.
   Indication : on pourra maintenir un tableau `b` *tel que* `b[k]` *soit la longueur de la plus longue sous suite croissante se terminant à l'indice* `k`

2. Améliorer l'algorithme pour obtenir une complexité de $\mathcal{O}(n\log(n))$
Indication : on pourra maintenir un tableau `b` *tel que* `b[k]` *soit le dernier élément de la plus longue sous suite de longueur* `k`

Lâchers d'œufs ⋆ ⋆ ⋆ *Source : [4]*
On se trouve devant un immeuble de n étages, et on dispose de k œufs strictement identiques. On cherche une stratégie qui permette de déterminer à coups sur l'étage à partir duquel les œufs se cassent. Un œuf qui ne s'est pas cassé peut être réutilisé par la suite.

1. Donner une stratégie optimale (en nombre de lâchers) pour les cas $k = 1$ et $n = k$. On ne demande pas de preuve.
2. Donner une stratégie pour le cas $k = 2$ et $\mathcal{O}(\sqrt{n})$ lâchers.
3. Écrire un algorithme qui détermine le nombre minimal de lâchers nécessaires dans le cas général. Cet algorithme aura une complexité en $\mathcal{O}\left(n^2k\right)$.

On note $H(k,p)$ le plus grand entier n tel qu'il existe une stratégie utilisant au plus k œufs et au plus p lâchers. On notera par la suite $L(n,k)$ le nombre minimal de lâchers d'une stratégie qui résout le problème.

4. Montrer que $\forall(k,n,p) \in \mathbb{N}, H(k,p) \geq n \Leftrightarrow L(n,k) \leq p$. En déduire (en utilisant la relation de récurrence trouvé en question 3) que $\forall k,p \geq 1, H(k,p) = H(k-1,p-1) + H(k,p-1) + 1$.
5. En déduire que $\forall k,p \geq 1, H(k,p) = \sum_{i=1}^{k} \binom{p}{i}$. On admettra que $H(k,p)$ peut se calculer en $\mathcal{O}(k)$.
6. Bonus. On pose $k' = \min(k, \lceil \log_2(n+1) \rceil)$. Écrire un algorithme qui détermine le nombre minimal de lâchers nécessaires toujours dans le cas général. Cet algorithme aura une complexité en $\mathcal{O}(k' \log k' + \log n)$.

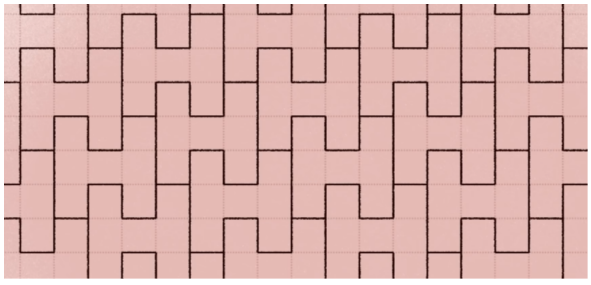
3.3 Intelligence artificielle étude des jeux

Tic-tac-toe ⋆ *Inspiré de "Ce que vous ne savez pas sur le morpion" - Aline Parreau*
On s'intéresse ici à une version généralisé du jeu du morpion :

— Le quadrillage considéré est de taille infinie (et pas 3×3)
— Pour gagner, le premier joueur doit aligner N ronds (et pas 3)
— Pour gagner, le deuxième joueur doit empêcher le premier joueur de gagner avec une stratégie qui fait durer une partie jusqu'à l'infini

1. Donner une stratégie gagnante pour le premier joueur lorsque $N \leq 3$

On cherche maintenant une stratégie gagnante pour le second joueur lorsque $N = 9$ On décide de paver le quadrillage avec des H imbriqués les uns dans les autres :



2. Montrer que toute ligne de 9 cases consécutives recouvre entièrement la barre du milieu d'un des H du pavage. On admet que toute colonne (resp diagonale) de 9 cases consécutives recouvre entièrement la barre de droite d'un des H du pavage (resp une des diagonale d'un des H du pavage). En déduire une propriété sur le pavage qui, si elle est conservée tout au long du jeu, permet au deuxième joueur de gagner.

3. On impose au premier joueur de toujours jouer au milieu d'un H avant d'attaquer les autres cases. Montrer que le deuxième joueur possède une stratégie gagnante. *Indication : lorsque le premier joueur attaque le milieu d'un H, répondre par la case juste à droite*
4. On impose au premier joueur de ne jamais jouer au milieu d'un H en premier. Montrer que le deuxième joueur possède une stratégie gagnante. *Indication : Cette fois-ci, attaquer la case du milieu*
5. Conclure.

3.4 Algorithmes probabilistes, algorithmes d'approximation

Pile ou face ⋆ MPI
Le but de cet exercice est de transformer un pile ou face honnête en pile ou face biaisé et vice versa. On ne demande pas d'algorithmes efficaces ni même qui terminent à coup sur.

1. Supposons qu'un programme `biaise` renvoie 1 avec une probabilité p inconnue (potentiellement non rationelle) et 0 avec une probabilité $(1-p)$. Donner un algorithme `honnete` qui renvoie 1 ou 0 de façon parfaitement équiprobable. Le seul accès à l'aléa provient évidement de `biaise`. En moyenne, combien d'appels à `biaise` seront effectués ?
2. À l'inverse, donner un algorithme qui étant donné un rationnel $0 < \rho < 1$ et une source honnête de pile ou face, renvoie 1 avec probabilité ρ et 0 avec probabilité $1-\rho$. Cette fois, on veut que l'algorithme termine à coup sur. Donner alors sa complexité dans le pire cas.

Polynôme nul ⋆ MPI
Dans cet exercice, un polynôme est représenté par une expression parenthésée quelconque utilisant les caractères `X, +, -, *, (, )` et des entiers. Par exemple $(X+10)*(X-10)-X*X+100$. On prendra comme taille d'une formule (utile pour calculer la complexité des programmes) le nombre de caractères de cette dernière (un entier compte pour un caractère).

1. Donner un algorithme naïf et déterministe qui détermine si un polynôme est nul ou non. Donner sa complexité.
2. Démontrer le lemme de Schwartz-Zippel pour les polynomes dans $\mathbb{R}$ à une variable : *Soit P un polynôme de degré d et S un sous-ensemble fini non vide de $\mathbb{R}$. Si s est un élément tiré uniformément dans S, alors* $\Pr[P(s)=0] \leq d/|S|$.
3. En déduire un algorithme probabiliste qui détermine si un polynôme est nul ou non. Étudier sa complexité et son taux de succès, et comparer avec l'algorithme de la question 1.

3.5 Divers

Min et max ⋆ *Source : [4]*
On cherche à déterminer le minimum et/ou maximum d'une liste *d'entiers distincts* de taille n. On rappelle que `min_int` (resp. `max_int`) est le plus petit (resp. grand) entier en OCaml, et donc que l'on peut le considérer comme égal à $-\infty$ (resp. ∞).

1. Écrire une fonction Ocaml `maxi : int list -> int` qui détermine l'indice du maximum de cette liste. Donner sa complexité et prouver sa correction et sa terminaison.
2. Soit un algorithme résolvant le problème précédent. Montrer qu'il existe une liste de taille n pour laquelle cet algorithme requiert au moins $n-1$ comparaisons. On supposera que l'algorithme n'a pas accès à la liste directement, seulement à une fonction `compare(i,j)` qui prend en argument deux indices et qui renvoie l'indice du maximum entre les deux éléments de la liste.
3. Écrire maintenant un algorithme (en pseudo code) qui permet de déterminer l'indice du maximum et l'indice du minimum en utilisant au plus $\lceil 3n/2 \rceil - 2$ comparaisons. Prouver sa correction et sa terminaison.

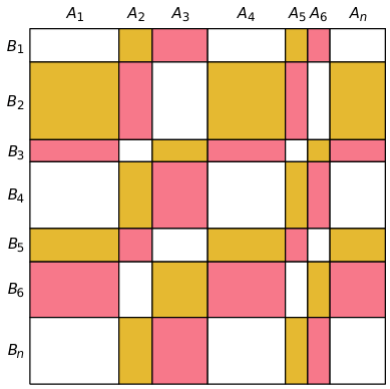
Maximum 2D ⋆ *Source : [2]*

On a un tableau d'entiers à 2 dimensions A (de taille $N \times N$), dont on sait que chaque ligne et chaque colonne est triée dans l'ordre croissant. On veut écrire un programme en C qui détermine si un entier x appartient à la matrice (on supposera que les variables `N` et `A` ont déjà été définies plus haut dans le programme. On pourra afficher "oui" ou "non".

1. Trouver un algorithme naïf et l'implémenter en C. On demande une complexité en $\mathcal{O}\left(n^2\right)$.
2. Comment améliorer (facilement) l'algorithme pour obtenir une complexité en $\mathcal{O}\left(n \log(n)\right)$?
3. Trouver un algorithme qui résout le problème en $\mathcal{O}\left(n\right)$. Prouver la correction et la terminaison à l'aide de variant et invariant de boucle. Si il reste du temps, implémenter l'algorithme en C.

Gatô ⋆ *Source : SWERC 2017*

Iskander le boulanger est en train de décorer un énorme gâteau, en recouvrant la surface rectangulaire du gâteau de glaçage. Pour cela, il mélange du sucre glace avec du jus de citron et du colorant alimentaire, afin de produire trois sortes de glaçage : jaune, rose et blanc. trois types de glaçage : jaune, rose et blanc. Ces couleurs sont identifiées par les chiffres 0 pour le jaune, 1 pour le rose, et 2 pour le blanc. Pour obtenir un joli motif, il divise la surface du gâteau en bandes verticales de largeur $A_1, A_2, \ldots, A_n$. centimètres, et en bandes horizontales de hauteur $B_1, B_2, \ldots, B_n$ centimètres, pour un nombre entier positif n. Ces bandes divisent la surface du gâteau en $n \times n$ rectangles. L'intersection de la bande verticale i et de la bande horizontale j horizontale j a pour numéro de couleur $(i + j)$ mod 3 pour tout $1 \leq i, j \leq n$. Pour préparer le glaçage, Iskander veut connaître la surface totale en centimètres carrés à colorer pour chacune des trois couleurs et vous demande de l'aide.



1. On pose $n = 10000$. Donner un algorithme et analyser sa complexité et terminaison. L'implémenter en OCaml.
2. On pose $n = 1000000$. Donner un algorithme et analyser sa complexité.
3. L'implémenter en OCaml (sur machine ? et le tester avec le jeu de test suivant)

```
n = 7
A = [|6 2 4 5 1 1 4|]
B = [|2 5 1 4 2 3 4|]
```

Solution : 155, 131 et 197.

Rectanges & Co ⋆ *Source : NWERC 2022*

On se donne trois entiers strictement positifs l, w, n et on cherche à savoir si il est possible de partitionner un rectangle de taille $l \times w$ en n rectangles de même surface.

1. Donner une condition nécessaire et suffisante pour que ce soit possible.
2. Donner un programme OCaml qui affiche une répartition possible en utilisant les n premières lettres de l'alphabet, en supposant que la CNS est vérifiée et que $n \leq 26$. On utilisera pour cela les fonctions `Char.code : char -> int` et `Char.chr : int -> char` qui associent un caractère à son code ASCII (ou réciproquement), par exemple 65 pour `'A'`, 66 pour `'B'`, etc... Par exemple, pour $l = 6, w = 15, n = 9$ on pourrait afficher :

```
GGGGGBBBBBBBBB
GGGGGAAAAAAAAA
IIIIIIIIIEEEEE
FFFFFFFFFEEEEE
CCCCCDDDDDHHHHH
CCCCCDDDDDHHHHH
```

Somme nulle ⋆ ⋆ *Source : SWERC 2022, problème H*

Dans cet exercice, on se donne deux entiers $0 < a \leq b$ ainsi qu'une chaine de n caractères contenant uniquement les caractères `+` et `-`. On se prête au jeu suivant : en partant d'un compteur $c = 0$, on parcourt la chaine, et pour chaque `+` (resp `-`) rencontré, on a le choix entre ajouter (resp soustraire) la valeur a ou la valeur b au compteur. Le but est d'arriver à la fin du jeu avec un compteur égal à 0.

1. Trouver un algorithme qui résout ce problème en $\mathcal{O}(n)$ puis l'implémenter en OCaml. Il aura pour signature `somme_nulle : int -> int -> string -> unit` et devra afficher "possible" ou "impossible" en fonction de si il existe une façon de gagner ou non. Par exemple, si $a = 3$ et $b = 1$, il est impossible de gagner avec la chaine `+-+` alors que c'est possible avec la chaine `-+++`. *Indication : considérer un vecteur contenant toutes les valeurs atteignables avec uniquement des additions (et idem avec des soustractions).*
2. Même question mais cette fois, on considère n couples (a, b) au lieu de seulement un, et on veut savoir au cas par cas si il est possible de gagner. La complexité restera en $\mathcal{O}(n)$.

Classe Majoritaire ⋆ ⋆ *Source : [4]*

Soit $E = \{x_0, \ldots, x_{n-1}\}$ un ensemble à n éléments et $\mathcal{R}$ une relation d'équivalence inconnue sur E. Une classe d'équivalence est dite *majoritaire* si son cardinal est strictement supérieur à $\lfloor \frac{n}{2} \rfloor$. On dispose d'une fonction `EnRelation` qui permet de savoir si deux éléments sont en relation. Les algorithmes demandés en OCaml et prennent en entrée un tableau de taille n représentant l'ensemble E.

1. Donner la définition d'une relation d'équivalence, puis d'une classe d'équivalence.
2. Écrire un programme en OCaml qui prend en argument le tableau et un entier i et renvoie la taille de la classe de x_i.
3. Écrire un programme en OCaml qui détermine s'il existe une classe majoritaire en utilisant $\mathcal{O}\left(n^2\right)$ appels à `EnRelation`.
4. Soit $(B_1, \ldots, B_k)$ une partition de E et $\mathcal{R}_i$ la relation induite par $\mathcal{R}$ sur B_i. Montrer que si C est une classe majoritaire pour $\mathcal{R}$, alors il existe $i \in \{1, \ldots, k\}$ tel que $\mathcal{R}_i$ admette $C_i = C \cap B_i$ comme classe majoritaire.
5. En considérant une partition de E en $k = \lceil \sqrt{n} \rceil$ parties de cardinal au plus $\lceil \sqrt{n} \rceil$, écrire un algorithme en pseudo-code qui détermine s'il existe une classe majoritaire en utilisant $\mathcal{O}\left(n^{3/2}\right)$ appels à `EnRelation`.
6. Supposons qu'il existe deux éléments $x, y \in E$ qui ne sont pas dans la même classe pour $\mathcal{R}$. Montrer que si $\mathcal{R}$ admet une classe majoritaire C, alors la relation induite par $\mathcal{R}$ sur $E \setminus \{x, y\}$ admet une classe majoritaire, et que celle ci est incluse dans C.
7. En utilisant la question précédente, écrire un algorithme utilisant au plus $2n$ appels à `EnRelation` qui détermine si il existe un classe majoritaire.

4 Graphes

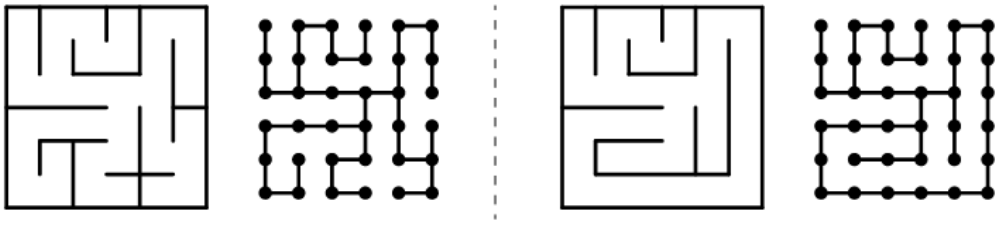
4.1 Graphes

Composantes

Écrire une fonction `composante : int list array -> int list list` qui prend en argument un graphe donné par ses listes d'adjacence et renvoie une liste de listes d'entiers correspondant aux différentes composantes connexes du graphe.

Cycles
Écrire une fonction `cycle: int list array -> bool` qui détermine si un graphe non orienté connexe donné par ses listes d'adjacence admet un cycle.

Labyrinthe ⋆ *D'après : Info A X-ENS 2020*
On se donne un graphe G de n sommets, numérotés de 0 à $n-1$. On appelle labyrinthe sur G un sous-graphe connexe de G qui a le même ensemble de sommets que G. Par exemple, on peut partir d'un graphe G dont les sommets sont les cases d'une grille rectangulaire et dont les arêtes correspondent aux cases adjacentes (nord, sud, est, ouest). Les arêtes du labyrinthe correspondent alors aux ouvertures permettant de passer d'une case à une autre. Les arêtes de G qui ne sont pas dans le labyrinthe correspondent aux murs. Voici deux exemples :



On dit qu'un labyrinthe est parfait lorsqu'il existe un et un seul chemin entre toute paire de sommets. Le labyrinthe de gauche dans l'exemple ci-dessus est parfait, alors que celui de droite ne l'est pas. Dans tout l'exercice, on pourra utiliser sans justification les algorithmes vus en cours.

1. Quelle propriété importante vérifie un labyrinthe parfait ?

On s'intéresse désormais aux labyrinthes quelconques, et on cherche des chemins permettant de relier un sommet de départ `src` à un sommet d'arrivée `dst`, en minimisant une métrique ou une autre.

2. Donner un algorithme linéaire qui détermine la taille du chemin le plus court pour rejoindre la sortie. Préciser les structures de donnée utilisées.

Maintenant, on considère en plus un ensemble de sommets *piégés* qu'il faudra éviter *de préférence*. Ils seront représentés par un tableau de booléens de taille n par exemple.

3. Donner un algorithme linéaire qui détermine le nombre minimum de pièges par lesquelles il faut passer pour rejoindre la sortie. Préciser les structures de donnée utilisées.
4. Donner un algorithme qui détermine le nombre minimum de pièges par lesquelles il faut passer ainsi que la taille du plus court chemin parmis ceux qui passent par le nombre minimum de pièges. En d'autre terme, on souhaite minimiser (p, l) par ordre lexicographique, où p représente le nombre de piège et l la taille d'un chemin de `src` à `dst`. On renverra donc un couple d'entiers (p, l). Préciser les structures de donnée utilisées ainsi que la complexité de votre algorithme.

Dominos ⋆ MPI *Source : [2]*
On s'intéresse à un circuit de dominos. Chaque domino est numéroté entre 1 et N, et on représente un circuit de dominos par une liste de M couples (u, v), qui signifie qui si le domino u tombe, alors il entraine le domino v dans sa chute (attention, ça ne va pas dans les deux sens), et ainsi de suite. Donner un algorithme en $\mathcal{O}(M+N)$ qui détermine le nombre minimum de domino qu'il faut pousser pour que tous les dominos tombent.

Ordre Topologique ⋆ MPI
Donner un algorithme en $\mathcal{O}(V+E)$ qui trouve un ordre topologique à base de suppression recursive de sommet sans parents dans un graphe orienté sans cycle.

Graphe Eulerien ⋆ ⋆ *inspiré de : [2]*
Dans un graphe non orienté $G = (V, E)$, un *chemin eulérien* est un chemin qui passe une fois et une seule par chaque arrête. Si le sommet de départ est identique au sommet d'arrivée, on parle alors de *cycle eulérien*.

1. Le lemme des poignées de main affirme que la somme des degrés des sommets d'un graphe $G = (V, E)$ non orienté vaut deux fois son nombre d'arête. En d'autres termes :

$$\sum_{v \in V} \deg(v) = 2|E|$$

   Démontrer ce lemme.
2. Démontrer le théorème suivant : un graphe connexe admet un cycle eulérien si et seulement si le degré de chacun de ses sommets est pair.
3. En déduire une condition nécésaire et suffisante pour qu'un graphe connexe admette un chemin eulérien. La démontrer en utilisant la question précédente.
4. Combien d'arêtes (au minimum) doivent être ajoutées à un graphe G pour qu'il admette un cycle eulérien ?
5. Donner un algorithme en $\mathcal{O}(|V| + |E|)$ qui trouve un cycle eulérien dans un graphe qui en admet un. Le graphe est donné sous forme de liste d'adjacence.

Formule d'Euler pour les graphes planaires. ⋆ ⋆

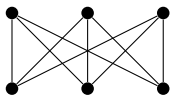
Un graphe *planaire* est un graphe qui peut se représenter sur un plan sans qu'aucune arête n'en croise une autre.

1. Donner une représentation planaire du graphe suivant :
   — Sommets : les cases d'un échiquier de taille 3×4
   — Arêtes : il existe une arête entre deux sommets si le mouvement d'un cavalier pour aller de l'un à l'autre est autorisé.

   En déduire qu'il existe une façon de placer un cavalier sur l'échiquier 3×4 puis de le faire passer par toutes les cases une fois et une seule.

On définit (naïvement) une *face* pour une représentation planaire d'un graphe planaire. Il s'agit d'une surface entouré d'arêtes (y compris la face dite "extérieure", ou comme si le graphe avait été dessiné sur une spère). Dans tous les graphes planaires que l'on considèrera par la suite, on notera n son nombre de sommet, a son sombre d'arête et f son nombre de face. On admet que le nombre de face ne dépend pas de la représentation planaire.

2. Démontrer la formule d'Euler pour les graphes planaires connexes : $n - a + f = 2$.
3. Les graphes complets à 4, 5 et 6 sommets sont-ils planaires ?
4. Le graphe $K_{3,3}$ suivant est-il plannaire ?

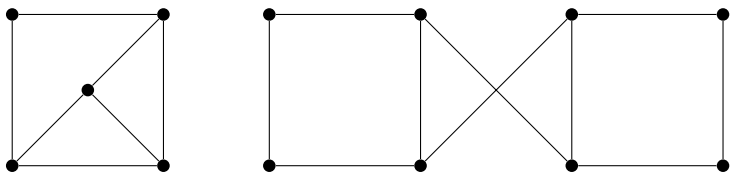


5. Démontrer la formule suivante pour un graphe planaire connexe de plus de 3 sommets : $|E| \leq 3|V| - 6$.

Théorème des 6 couleurs. ⋆ ⋆ *Inspiré de : [4]*

Un graphe non orienté $G = (V, E)$ est dit k-coloriable s'il existe une fonction $c : V \mapsto [\![1; k]\!]$ telle que pour toute arête $\{u, v\} \in E$, $c(u) \neq c(v)$. On appelle *nombre chromatique de* G le plus petit entier k tel que G soit k-coloriable.

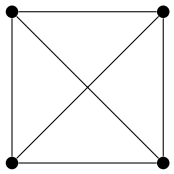
1. Donner le nombre chromatique des graphes suivants :



2. Donner le nombre chromatique d'un arbre.

Un graphe *planaire* est un graphe qui peut se représenter sur un plan sans qu'aucune arrête n'en croise une autre.

3. Le graphe suivant est-il planaire ?



On admet la formule suivante pour un graphe planaire connexe de plus de 3 sommets : $|E| \leq 3|V| - 6$.

4. Montrer qu'un graphe planaire a au moins un sommet de degré inférieur ou égal à 5.
5. En déduire que tout graphe planaire est 6-coloriable.

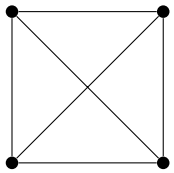
Théorème des 5 couleurs. ⋆ ⋆ ⋆ *Inspiré de : [4]*

Un graphe non orienté $G = (V, E)$ est dit k-coloriable s'il existe une fonction $c : V \mapsto [\![1; k]\!]$ telle que pour toute arête $\{u, v\} \in E$, $c(u) \neq c(v)$.

1. Quels sont les graphes 1-coloriables ?
2. Montrer que pour tout entier k il existe des graphes non k-coloriable.

Un graphe *planaire* est un graphe qui peut se représenter sur un plan sans qu'aucune arrête n'en croise une autre.

3. Le graphe suivant est-il planaire ?



On admet la propriété suivante : tout graphe planaire admet au moins un sommet de degré inférieur ou égal à 5.

4. Soit c un coloriage d'un graphe $G = (V, E)$ et $s, t \in V$. Montrer que l'une des deux conditions suivantes est réalisée :
   — Il existe une chaîne alternée avec deux couleurs reliant s à t.
   — Il existe un coloriage avec le même nombre de couleurs que c pour lequel s et t sont de la même couleur.
5. En déduire que tout graphe planaire est 5-coloriable.

4.2 Arbres, arbres couvrant

Diviser pour régner MPI *Source : [2]*

On considère un algorithme de type diviser-pour-régner pour calculer un arbre couvrant de poids minimum : Découper l'ensemble de sommets V en deux sous-ensembles V_1 et V_2 de même cardinal (à un près), et appliquer l'algorithme sur les sous-graphes induits par V_1 et V_2. Enfin, utiliser l'arête de poids minimum de la coupe (V_1, V_2) pour relier les deux arbres obtenus. L'algorithme est-il correct ? Si oui, prouver qu'il retourne un arbre couvrant de poids minimum et donner sa complexité. Sinon, donner un exemple où l'algorithme échoue.

Diamètre d'un arbre ⋆

Un arbre est un graphe non orienté, connexe et sans cycle. Le diamètre d'un graphe connexe est la

distance maximale entre deux points du graphe (chaque arête a un poids de 1). Les algorithmes de cet exercice sont à donner en pseudo-code, les graphes sont représentés par liste d'adjacence, et les complexités sont données en fonction de n le nombre de sommet du graphe.

1. Donner un algorithme quadratique qui détermine le diamètre d'un graphe.
2. Donner un algorithme linéaire qui détermine si un graphe est un arbre.
3. Donner un algorithme linéaire qui détermine le diamètre d'un arbre. On pourra enraciner ce dernier arbitrairement et utiliser une fonction récursive.
4. On se donne maintenant l'algorithme suivant :
   — Choisir un noeud arbitraire x.
   — Effectuer un parcours en largeur depuis ce point, et noter y le dernier noeud visité.
   — Effectuer un parcours en largeur depuis y, noter z le dernier noeud visité et d la distance entre y et z.
   — Renvoyer d.

   Cet algorithme fonctionne-t-il sur un graphe connexe quelconque ?
5. Prouver la correction de l'algorithme de la question précédente dans le cas d'un arbre.

Codage de Prüfer ⋆ *Source : [4]*

Le codage de Prüfer d'un arbre T est une suite d'entier construite à partir de l'algorithme suivant :

```
Donnees : Arbre T
Tant qu'il reste plus de deux sommets dans l'arbre T
    Identifier la feuille v de l'arbre ayant le numero minimum
    Ajouter a la suite P le seul sommet s adjacent a v dans l'arbre T
    Enlever de l'arbre T le sommet v et l'arete incidente a v
Fin Tant que
```

1. Donner le codage de Prüfer de l'arbre suivant :

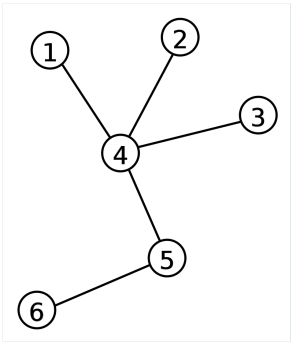


Image : Wikipedia

2. Donner un arbre qui admet $(1, 1, 1, 1, 1)$ comme codage de Prüfer.
3. Montrer qu'un sommet est de degré k dans l'arbre T si et seulement si s apparait $(k-1)$ fois dans le codage de Prüfer de l'arbre T.
4. Donner un algorithme qui reconstruit un T à partir de son codage de Prüfer et de son tableau des degrés.
5. En déduire le théorème de Cayley : *Le nombre d'arbres différents que l'on peut construire sur n sommets numérotés, avec $n > 1$ est égal à n^{n-2}.*

Labyrinthe 2 ⋆ MPI *Inspiré de : Info A X-ENS 2020*

Dans cet exercice on s'intéresse à la génération aléatoire de labyrinthes parfaits. Pour un graphe donné, un labyrinthe parfait est un sous graphe dans lequel tout couple de sommets est relié par un unique chemin. On génère habituellement des labyrinthes en partant d'un graphe connexe duquel on déduit un sous graphe qui est un labyrinthe parfait, en gardant les mêmes sommets.

1. Donner une meilleure définition d'un labyrinthe parfait.
2. Donner un labyrinthe parfait issu de la grille de taille 4×4.

3. Le mélange de Knuth est un algorithme qui permet de permuter aléatoirement les éléments d'un tableau. L'algorithme peut être décrit par la fonction Ocaml suivante :

```
let melange_knuth a =
  (* Source : Olivier Brunet et Eric Detrez *)
  let n = Array . length a in
  for i = 1 to n - 1 do
    let j = Random . int ( i + 1) in
    if j < i
    then ( let tmp = a .( j ) in
      a .( j ) <- a .( i ) ;
      a .( i ) <- tmp ) done ;;
```

   Montrer que si on restreint l'entrée aux tableaux a de taille n tels que $\forall i \in [0, n-1], a.(i) = i$, alors `melange_knuth` permet de générer des permutations aléatoires uniformément distribuées sur $\mathfrak{S}_n$.

Dans toute la suite du sujet, on suppose donc que l'on peut permuter aléatoirement les éléments d'un tableau.

4. Maintenant on cherche à générer des labyrinthes parfaits à partir de graphes connexes. Une façon de procéder est d'effectuer un DFS depuis un sommet quelconque. Lorsqu'un sommet v est traité, on permute aléatoirement la liste de ses voisins et on la parcourt. Pour chaque voisin w de v qui n'a pas encore été visité, on ajoute l'arête (v, u) au labyrinthe et on traite w. Montrer que cet algorithme renvoie effectivement un labyrinthe parfait.
5. Décrire brièvement l'algorithme de Kruskal. Comment l'appliquer à un graphe connexe pour obtenir un labyrinthe parfait aléatoire ?

Cluster ⋆ MPI *Source : [2]*

On se donne un ensemble V de N objets et une fonction distance $d : V \times V \to \mathbb{Q}^+$ telle que $d(x, y) = 0 \iff x = y$. Le clustering consiste à créer plusieurs groupes pour placer l'objet de la manière la plus pertinente possible. Nous proposons d'utiliser la distance pour créer de tels groupes : les objets proches les uns des autres doivent appartenir au même groupe et les objets éloignés les uns des autres doivent appartenir à des groupes différents.

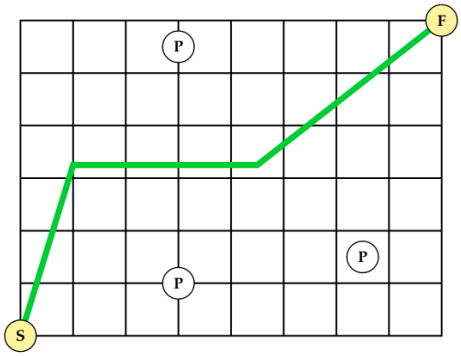
Le k-clustering de V est une partition de V en k classes non vides $V_1, \ldots, V_k$. L'espacement d'un k-clustering est définie par la distance minimale entre deux objets de classes différentes. Un objectif naturel est de déterminer les classes qui maximisent ce paramètre.

Étant donné $k \in \mathbb{N}$, V un ensemble d'objets et la fonction distance, écrire un algorithme permettant de calculer un k-clustering avec espacement maximal. Montrer que votre algorithme est correct et donner sa complexité.

Épidémie ⋆ ⋆ *Source : SWERC 2020, problème C*

En période d'épidémie, on s'intéresse à la distance maximale qu'il est possible de garder avec les personnes qui nous entourent. Ici, on cherche un algorithme qui calcule cette distance dans le contexte particulier suivant :

On se situe dans une pièce rectangulaire, au niveau d'un coin (de coordonnée $(0, 0)$ par exemple) et on souhaite rejoindre le coin opposé en conservant cette distance de sécurité. On donne trois entiers X et Y (la taille de la pièce), N le nombre de personne dans la pièce, ainsi qu'une liste de N coordonnées (x_i, y_i) représentant les positions des N personnes. En d'autres termes, on cherche la distance maximale que l'on peut conserver avec les autres personnes de la pièce pour rejoindre le coin opposé). Dans l'exemple ci-dessous, la distance recherchée est 2.25 :



Si la distance maximale recherchée n'est pas entière, un arrondi supérieur ou inférieur est accepté. Donner un algorithme (en pseudo-code) qui résout ce problème. Après vérification, l'implémenter dans langage de votre choix.

Arbres isomorphes ⋆ ⋆ *D'après : [4]*

Le but de cet exercice est de trouver un algorithme quadratique qui détermine si deux arbres sont isomorphes. Deux arbres enracinés (T, r) et (T', r') sont *isomorphes* (on note $(T, r) \sim (T', r')$) si il existe ϕ une application qui à chaque noeud de T associe un noeud de T', et ce en respectant la propriété de parenté (c'est à dire que si u est le père de v alors $\phi(u)$ est le père de $\phi(v)$. En particulier, $\phi(r) = r'$. Informellement, deux arbres enracinés sont isomorphes si ils ont la même racine et la même forme.

1. La relation $\sim$ est elle une relation d'équivalence ?
2. Donner un algotithme quadratique qui détermine si deux arbres enracinés sont isomorphes. *Indication : associer à chaque arbre un mot caractéristique sur deux lettres (disons a et b) tel que deux arbres sont isomorphes ssi ils ont le même mot*

On considère désormais des arbres non-enracinés (aucun noeud ne fait office de racine). Deux arbres non-enracinés T et T' sont *isomorphes* si il existe ϕ une application qui à chaque noeud de T associe un noeud de T', et ce en respectant la propriété de voisinage (c'est à dire que si u est voisin de v alors $\phi(u)$ est voisin de $\phi(v)$.

3. Donner un algotithme cubique qui détermine si deux arbres non-enracinés sont isomorphes.
4. Dans un arbre non-enraciné, l'excentricité d'un noeud correspond à la distance entre ce noeud et la feuille la plus lointaine. On définit C le centre d'un arbre comme l'ensemble des noeuds minimaux lorsqu'ils sont triés par excentricité. Montrer que le centre de chaque arbre contient 1 ou 2 éléments.
5. On suppose que l'on connait un algorithme linéaire qui trouve le centre d'un arbre non-enraciné. Donner un algorithme quadratique qui détermine si deux arbres non-enracinés sont isomorphes.

4.3 Couplages

Mots croisés ⋆ ⋆ MPI *Inspiré d'un problème Prologin*

On considère dans cet exercice des grilles de mots croisés. Il s'agit d'un quadrillage de taille $N \times M$ dans lequel certaines cases sont coloriées (les "cases noires"). Le but est de compléter cette grille (i.e. remplir toutes les cases blanches) grâce aux définitions données en annexe.

Dans tout l'exercice on pourra utiliser le théorème de König : *Pour tout graphe biparti G, le cardinal maximal d'un couplage de G est égal au cardinal minimal d'une couverture de G.* On rappelle qu'une couverture d'un graphe est un ensemble de sommets C tel que chaque arête du graphe est incidente à au moins un sommet de C.

1. Démontrer que dans un graphe quelconque, le cardinal d'une couverture est toujours supérieur ou égal au cardinal d'un couplage.
2. Pour une grille donnée, on voudrait connaitre la taille du plus grand carré blanc. Donner un algorithme qui détermine cette taille à partir de N, M et p coordonnées (celles des "cases noires"). Donner la complexité de votre algorithme.

3. Un éditeur souhaite rendre ses grilles plus difficiles. Plutôt que de donner une définition pour chaque mot de la grille, il se réserve le droit de retirer celles qui sont superflues. En d'autres termes, si un mot est complètement déterminé par les mots qui le croisent, il n'est pas nécessaire de donner sa définition. Donner un algorithme qui détermine le nombre minimum de définition que l'éditeur aura à expliciter (à partir d'une grille sous le même format que précédemment). On demande une complexité polynomiale en le nombre de définitions initiales.

Théorème de König détaillé ⋆ ⋆ ⋆ MPI *Source : [4]*

Soit $G = (V, E)$ un graphe et C un couplage de ce graphe. Un sommet de G est C-insaturé s'il n'est extrémité d'aucune arête de C. Un chemin C-augmentant est un chemin de G dont les extremités sont C-insaturés et dont les arêtes sont alternativement dans C et $V \setminus C$. Une couverture de G est un ensemble de sommets K tel que chaque arête du graphe est incidente à au moins un sommet de K.

1. Démontrer que dans un graphe quelconque, le cardinal d'une couverture est toujours supérieur ou égal au cardinal d'un couplage.

On suppose désormais G biparti, c'est à dire que l'ont peut partitionner V en A et B tels que toute arête de G relie un sommet de A et un sommet de B.

2. Soit Z l'ensemble des sommets de G accessible depuis au moins un sommet C-insaturé de A par un chemin alternant arêtes de $A \setminus C$ et arêtes de C. Montrer que toute arête de C admet soit ses deux extrémités dans Z, soit aucune.
3. Soit $K = (A \setminus Z) \cup (B \cap Z)$. Montrer que K est une couverture de G.
4. Montrer que tout sommet de K est une extrémité d'une arête de C. On admet pour cela le lemme de Berge : *Un couplage C dans un graphe G est maximum si et seulement s'il n'y a pas de chemin C-augmentant qui alterne entre les arêtes dans et en dehors du couplage C.*
5. En déduire le théorème de König : *Pour tout graphe biparti G, le cardinal maximal d'un couplage de G est égal au cardinal minimal d'une couverture de G.*
6. Donner un algorithme qui détermine la taille de la couverture minimale d'un graphe biparti. Quelle est sa complexité ?

5 Logique

2SAT ⋆ ⋆ MPI

On s'intéresse au problème 2SAT : à partir d'une formule sous forme normale conjonctive avec deux littéraux par clauses, déterminer si la formule est satisfiable ou non. On se limitera aux algorithmes polynomiaux, mais on ne cherchera pas à optimiser ces derniers.

1. Donner un algorithme qui résout le problème si l'on remplace "forme normale conjonctive" par "forme normale disjonctive". L'hypothèse de deux littéraux par clause est-elle nécessaire ?

Le graphe d'implication d'une formule 2SAT est un graphe orienté non pondéré dont les sommets sont les variables et les négations de variables (les deux à chaque fois) qui composent la formule. Les arcs représentent eux les implications induites par les clauses : si $(x \vee y)$ est une clause alors on le graphe contiendra les arcs $\neg x \implies y$ et $\neg y \implies x$.

2. Représenter le graphe d'implication de $(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$.
3. Donner une condition nécessaire et suffisante sur le graphe d'implication d'une formule 2SAT pour que cette dernière soit satisfiable.
4. En déduire un algorithme qui résout le problème. Prouver sa correction et sa complexité. Peut on adapter l'algorithme en enlevant l'hypothèse de deux littéraux par clauses ?

6 Bases de données

Modèle Entité Association MP2I

Des acteurs (représentés par un identifiant unique, un nom, un prénom, et éventuellement un nom de scène) jouent dans des films (représentés par un identifiant unique, un titre, une année de sortie, et un réalisateur). Sur une plateforme de streaming, ces derniers sont vus par des spectateurs (représentés par un pseudo et un prénom), qui peuvent alors publier une (seule) critique (composée d'une note entre 0 et 5, éventuellement un commentaire et une date de publication).

1. Proposer un modèle entité / association permettant de représenter les données décrites ci-dessus. Ne pas hésiter à demander des précisions.
2. En déduire un modèle relationnel.

Donner les requêtes SQL permettant de connaitre :

3. Toutes les évaluations des films réalisés par Steven Spielberg, avec le prénom du spectateur et le titre du film.
4. La liste des films (titre et année) qui n'ont jamais reçu de commentaires (mais potentiellement des notes).
5. Le film ayant la meilleure note moyenne. Donner le titre du film et la note moyenne.

7 Langages et automates

7.1 Langages formels

Lemme d'Arden ⋆ *Source : [1]*

Soit $\Sigma = \{a, b\}$. On considère l'équation $L = \{a\}L \cup \{b\}$ d'inconnue $L \subset \Sigma^*$.

1. Trouver une solution de cette équation.
2. Montrer qu'il s'agit de l'unique solution

Soit Σ un alphabet quelconque et A, B deux langages sur Σ. On considère l'équation $L = AL \cup B$ d'inconnue $L \subset \Sigma^*$.

3. Trouver une solution de cette équation.
4. Montrer que si $\varepsilon \notin A$ alors il s'agit de l'unique solution.

Mots primitifs ⋆ ⋆ *Source : [4]*

1. Soient x, y, z, et t quatre mots tels que $xy = zt$. Montrer qu'il existe un unique mot u tel que l'une des conditions suivante est réalisée.
   - $x = zu$ et $t = uy$
   - $z = xu$ et $y = ut$
2. Soient x et y deux mots. Montrer que les propriétés suivantes sont équivalentes.
   1. $xy = yx$
   2. Il existe un mot z et deux entiers p et q tels que $x = z^p$ et $y = z^q$
   3. Il existe n et m non tous les deux nuls tels que $x^m = y^n$

   Indication : $1 \implies 2 \implies 3 \implies 1$
3. Soit u un mot non vide. Montrer que les propriétés suivantes sont équivalentes.
   1. Le seul mot dont u est une puissance est u
   2. Pour tous mots x et y tels que $u = xy = yx$, on a $x = \varepsilon$ ou $y = \varepsilon$

Un tel mot u est dit primitif

4. Soit $u = xy$ un mot primitif. Montrer que le mot yx est aussi primitif.

7.2 Automates

Barman aveugle *Source : [1]*

1. Donner un automate fini non déterministe reconnaissant le langage des mots se terminant par ab (sur l'alphabet $\Sigma = \{a, b\}$. Le déterminiser.
2. Un barman aveugle joue au jeu suivant avec un client : il a devant lui un plateau sur lequel sont disposés quatre verres formant un carré. Chacun de ces verres peut être retourné ou non, sans que le barman ne le sache. Le but de ce dernier est de s'arranger pour que tous les verres soient tournés dans le même sens. Pour ce faire, il peut à chaque tour choisir l'une des trois actions suivantes :
   — Tourner l'un des verres
   — Tourner deux verres voisins
   — Tourner deux verres opposés

   Modéliser ce jeu par un automate et donner une stratégie gagnante pour le Barman.

Factorielle $\star$ MPI *Source : [2]*
On s'intéresse à $\mathcal{L} = \{a^{n!} | n \in \mathbb{N}\}$ sur $\Sigma = \{a, b\}$.

1. Le langage $\mathcal{L}$ est-il rationnel ?
2. On considère $\mathcal{L}' = b^+\mathcal{L} \cup \{a\}^*$. Vérifie-t-il la conclusion du lemme de l'étoile ? Est-il rationnel ? *Indication : considérer* $\mathcal{L}'' = ba^* \cap \mathcal{L}'$

Langages reconnaissables $\star\star$ *Source : [4]*
Soit Σ un alphabet et $\mathcal{L}, \mathcal{L}'$ des langages reconnaissables.

1. Soit (Q, q_0, F, δ) un automate et soit $q, q' \in Q$. Montrer que le langage $\mathcal{L}_{q,q'} = \{u \in \Sigma^*, \delta^*(q, m) = q'\}$ est reconnaissable.
2. Montrer que le langage $\sqrt{\mathcal{L}} = \{u \in \Sigma^*, u^2 \in \mathcal{L}\}$ est reconnaissable.
3. Montrer que le langage $\mathcal{L} : \mathcal{L}' = \{u \in \mathcal{L}, \exists u' \in \mathcal{L}', |u| = |u'|\}$ est reconnaissable.
4. Montrer que le langage $\frac{\mathcal{L}}{2} = \{u \in \Sigma^*, \exists u' \in \Sigma^*, |u| = |u'|, uu' \in \mathcal{L}\}$ est reconnaissable.
5. Montrer que le langage $\{u \in \Sigma^*, u\bar{u} \in \mathcal{L}\}$ est reconnaissable.

Lemme de l'étoile et extensions $\star\star$ MPI *Source : [4]*

1. (Cours) Énoncer le lemme de l'étoile (ou lemme de pompage) et le démontrer.
2. (Cours) Le langage $\mathcal{L} = \{a^n b^n | n \in \mathbb{N}^*\}$ est-il rationnel ? (sur $\Sigma = \{a, b, c\}$)
3. Le langage $\mathcal{L}' = c^+\mathcal{L} \cup \{a, b\}^*$ vérifie-t-il la conclusion du lemme de l'étoile ? Est-il rationnel ? (Avec toujours $\Sigma = \{a, b, c\}$). *Indication : que vaut* $c\{a, b\}^* \cap \mathcal{L}'$ *?*
4. Démontrer la version forte du lemme de l'étoile : Si $\mathcal{L}$ est reconnaissable alors il existe un entier N tel que pour tout mot w de $\mathcal{L}$, et pour toute factorisation $w = uw'v$, avec w' de longueur $|w'| \geq N$, il existe une factorisation $w' = xyz$ telle que
   1. $0 < |y| \leq N$ et
   2. $uxy^*zv \subset L$
5. Montrer que $\mathcal{L}'$ ne vérifie pas la conclusion du lemme précédent. Qu'en déduisez vous ?
6. Démontrer le lemme de l'étoile *par bloc* : Si $\mathcal{L}$ est reconnaissable alors il existe un entier N tel que pour tout mot w de $\mathcal{L}$, et pour toute factorisation $w = uw_1w_2 \dots w_Nv$ (où les w_i sont non vides), il existe deux entiers k et l vérifiant
   1. $0 \leq k < l \leq N$ et
   2. $uw_1 \dots w_k(w_{k+1} \dots w_l)^*w_{l+1} \dots w_Nv \subset L$
7. (Bonus) Justifier que le lemme de l'étoile *par bloc* est plus fort que les deux autres

Myhill Nerode ⋆ ⋆ ⋆ *Inspiré de : Wikipedia*

Le théorème de Myhill-Nerode donne une condition nécessaire et suffisante pour qu'un langage formel soit un langage rationnel, c'est-à-dire reconnaissable par un automate fini.
Étant donné un langage $\mathcal{L}$ et deux mots x et y, on dit qu'un mot z sépare x et y si un et un seul des mots xz et yz est dans le langage $\mathcal{L}$. Deux mots x et y sont inséparables s'il n'existe pas de mot z qui les sépare.
On définit une relation $R_{\mathcal{L}}$ sur les mots, appelée relation de Myhill-Nerode, par la règle : $xR_{\mathcal{L}}y$ si et seulement si x et y sont inséparables.

1. Montrer que $R_{\mathcal{L}}$ définit bien une relation d'équivalence sur les mots.

Le nombre de classes est appelé l'index de la relation. Il peut être fini ou infini. On énonce maintenant le théorème de Myhill Nerode : un langage $\mathcal{L}$ est rationnel si et seulement si la relation $R_{\mathcal{L}}$ est d'index fini.

2. Utiliser le théorème précédent pour montrer que $\{a^n b^n | n \in \mathbb{N}^*\}$ n'est pas rationnel.
3. Montrer le sens direct du théorème.
4. Montrer le sens indirect. *Indication : construire un automate fini comme suit. Les états sont les classes de l'équivalence $R_{\mathcal{L}}$. L'état initial est la classe d'équivalence du mot vide, et la fonction de transition mène, pour un état p et une lettre a, à l'état q qui contient le mot xa, où x est un mot quelconque de p. Un état de l'automate est final s'il contient un mot de $\mathcal{L}$.*
5. Comparer le nombre de classe d'équivalence de $\mathcal{L}$ et le nombre d'état d'un AFD reconnaissant $\mathcal{L}$.

8 Décidabilité et classes de complexité

Liste de problèmes NP-Complets

On donne ici une liste de problèmes NP-complets que l'on pourra utiliser sans justification dans cette partie.

— **SAT** : étant donnée une formule de logique propositionnelle sous FNC, existe-t-il une assignation des variables propositionnelles qui rend la formule vraie ?
— **3-SAT** : étant donnée une formule de logique propositionnelle sous FNC avec 3 littéraux par clause, existe-t-il une assignation des variables propositionnelles qui rend la formule vraie ?
— **Subset-Sum (SS)** : étant donné un ensemble fini S d'entiers positifs et un entier objectif t, existe-t-il un sous-ensemble $S' \subset S$ tel que la somme des éléments de S' soit t ?
— **2-Partition (2P)** : étant donné un multiensemble S d'entiers naturels strictement positifs de taille n, existe-t-il une partition de S en deux sous-ensembles S_1 et S_2 tels que la somme des éléments de S_1 soit égale à la somme des éléments de S_2 ?
— **3-Partition (3P)** : étant donné un multiensemble S d'entiers naturels strictement positifs de taille n, existe-t-il une partition de S en trois sous-ensembles S_1, S_2 et S_3 tels que les sommes des éléments de S_i soient égale pour $i \in \{1, 2, 3\}$?
— **Clique** : étant donné un graphe $G = (V, E)$ non orienté et un entier k, existe-t-il $V' \subset V, |V'| = k$ tel que $u, v \in V' \implies (u, v) \in E$?
— **Circuit Hamiltonien (CH)** : étant donné un graphe $G = (V, E)$, contient-il un circuit hamiltonien (c.à.d un cycle passant par chaque sommet exactement une fois) ?

Questions de cours (rapide) MPI *Source : [2]*

Soient P_1 et P_2 deux problèmes de décision, et supposons qu'on connaisse une transformation polynomiale (une réduction) de P_1 en P_2.

1. Si $P_1 \in \mathsf{P}$, a-t-on $P_2 \in \mathsf{P}$?
2. Si $P_2 \in \mathsf{P}$, a-t-on $P_1 \in P$?
3. Si $P_1 \in \mathsf{NPC}$, a-t-on $P_2 \in \mathsf{NPC}$?
4. Si $P_2 \in \mathsf{NPC}$, a-t-on $P_1 \in \mathsf{NPC}$?

5. Si on connait une transformation polynomiale de P_2 en P_1, ces deux problèmes sont-ils NPC ?
6. Si $P_1, P_2 \in \mathsf{NPC}$, existe-t-il une transformation polynomiale de P_2 en P_1 ?
7. Si $P_1 \in \mathsf{NP}$, a-t-on $P_2 \in \mathsf{NPC}$?

2-Partition & Co MPI *Source : [2]*

On définit le problème de décision **2-Partition** : *étant donné un multiensemble S d'entiers naturels strictement positifs de taille n, existe-t-il une partition de S en deux sous-ensembles S_1 and S_2 tels que la somme des éléments de S_1 soit égale à la somme des éléments de S_2 ?*
Montrer que l'on peut réduire (polynomialement) **2-Partition** aux problèmes de décisions suivants :

1. **2-Partition-n-Pair** : Comme **2-Partition** mais on impose n pair.
2. **2-Partition avec même cardinal** : Comme **2-Partition** mais on impose que S_1 et S_2 soient de même cardinal.
3. **2-Partition mais approx** : Comme **2-Partition** mais on veut que $|\sum_{x\in S_1} x - \sum_{y\in S_2} y| \leq 1$.

Paires interdites MPI *D'après : [2]*

Montrer que le problème suivant est NP-complet : étant donné un entier n, et m paires (distinctes) d'entiers $(i, j) \in [1, n], i \neq j$, existe-t-il un graphe circulaire (i.e. connexe et tous les sommets de degré deux) de taille n tel qu'aucune paire d'entier ne représente une arête du graphe ?

Roue MPI *Source : [2]*

Montrer que le problème suivant est NP-complet : étant donnés un graphe $G = (V, E)$ et un entier $K \leq 3$, déterminer si G contient une roue de taille K, i.e. un ensemble de $K + 1$ sommets $w, v_1, v_2, \ldots, v_K$ tels que $(v_i, v_{i+1}) \in E$ pour $1 \leq i < K$; $(v_K, v_1) \in E$ et $(v_i, w) \in E$ pour $1 \leq i \leq K$ (w est le centre de la roue).

Réductions depuis Subset-Sum ⋆ MPI *Source : [2]*

On définit les problèmes de décision suivants :

— **Subset-Sum** : *Étant donné un ensemble fini S d'entiers strictement positifs et un entier objectif t, existe-t-il un sous-ensemble $S' \subset S$ tel que la somme des éléments de S' soit t ?*

— **2-Partition** : *étant donné un multiensemble S d'entiers naturels strictement positifs de taille n, existe-t-il une partition de S en deux sous-ensembles S_1 et S_2 tels que la somme des éléments de S_1 soit égale à la somme des éléments de S_2 ?*

— **3-Partition** : *étant donné un multiensemble S d'entiers naturels strictement positifs de taille n, existe-t-il une partition de S en trois sous-ensembles S_1, S_2 et S_3 tels que les sommes des éléments de S_i soient égale pour $i \in \{1, 2, 3\}$?*

On rappelle que pour réduire (polynomialement) P_1 à P_2 il faut donner un algorithme polynomial qui résout P_1 avec l'aide d'un oracle pour P_2 (on lui donne une instance et il répond correctement en temps constant).

1. Montrer que l'on peut réduire **Subset-Sum** à **2-Partition**
2. Montrer que l'on peut réduire **2-Partition** à **3-Partition**

Sac à dos ⋆ MPI *D'après : [2]*

On définit le problème du sac à dos de la façon suivante : étant donné n objets de valeurs respectives v_i, de poids respectifs p_i et un sac à dos pouvant contenir un poids maximal de C, quelle est la valeur maximal de ce sac à dos rempli ? Plus formellement, étant donné deux entiers n et C, ainsi que deux familles d'éléments $(p_i)_{i\in[1,n]}, (v_i)_{i\in[1,n]}$, déterminer

$$\max_{S\subset[1,n]/\sum_{i\in S} p_i \leq C} \sum_{i\in S} v_i$$

1. Étudier rapidement la possibilité d'une exploration exhaustive (complexités, heuristiques). Estimer alors le nombre d'objet à partir duquel cette méthode n'est plus envisageable.
2. Donner un algorithme qui résout ce problème en complexité $\mathcal{O}(nC)$. *Indication : utiliser le paradigme de programmation dynamique.*

3. Ramener ce problème à un problème de décision, puis montrer que ce dernier est NP-complet. Pourquoi n'est-ce pas incohérent avec la question précédente ?

Ordonnancement ⋆ MPI *Source : [2]*

Montrer que le problème suivant est NP-complet : On exécute n taches indépendantes $T_i, 1 \leq i \leq n$, sur un seul processeur. La durée de T_i est a_i, et T_i doit s'exécuter sans interruption dans l'intervalle $[r_i, d_i]$ (r_i est la date de disponibilité ou release time et d_i est l'échéance ou deadline). Tous les nombres a_i, r_i et d_i sont entiers. On donne une constante entière D et on veut décider si on peut exécuter les n taches dans l'intervalle $[0, D]$.

Clique Régulier ⋆ MPI *Source : [2]*

Montrer que le problème suivant est NP-complet : étant donné un graphe $G = (V, E)$ *régulier* non orienté et un entier k, existe-t-il $V' \subset V, |V'| = k$ tel que $u, v \in V' \implies (u, v) \in E$. On pourra partir de **Clique**.

Clique ⋆ ⋆ MPI *Source : Olivier Bournez (Cours INF423 de l'X)*

1. En partant de **3-SAT**, montrer que le problème **Stable** est NP-complet : étant donné un graphe $G = (V, E)$ non orienté et un entier k, existe-t-il $V' \subset V, |V'| = k$ tel que $u, v \in V' \implies (u, v) \notin E$.
2. En déduire que le problème **Clique** est NP-complet.

3SAT & Co ⋆ ⋆ MPI *Source : [2]*

On rappelle le problème de décision **3SAT** : *étant donnée une formule de logique propositionnelle sous FNC avec 3 littéraux par clause, existe-t-il une assignation des variables propositionnelles qui rend la formule vraie ?*

Montrer que l'on peut réduire (polynomialement) 3SAT aux problèmes de décisions suivants :

1. **3-SAT NAE** (not all equal), où l'on impose que les trois littéraux de chaque clause ne soient pas tous à la même valeur.
   Indication : transformer les clauses $C_i = a_i \vee b_i \vee c_i$ *en* $C'_i = a_i \vee b_i \vee z_i$ *et* $C''_i = c_i \vee \bar{z_i} \vee f$.
2. **3-SAT OIT** (one in three), où l'on impose qu'exactement un littéral soit à vrai dans chaque clause.
   Indication : transformer les clauses $C_i = a_i \vee b_i \vee c_i$ *en* $C'_i = a_i \vee x_i \vee y_i$ *et* $C''_i = \bar{b_i} \vee x_i \vee x'_i$ *et* $C'''_i = \bar{c_i} \vee y_i \vee y'_i$.

A Entrainement aux oraux

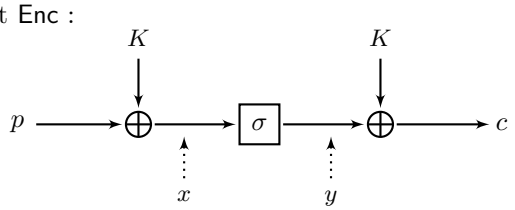
A.1 X-ENS

Cryptanalyse d'un chiffre par bloc

Un chiffre par bloc est une famille de permutations indéxée par une clef (souvent secrète). Par exemple, pour une permutation σ et un entier n donnés, le chiffre par bloc Even-Mansour est la famille de fonctions définies sur $\{0,1\}^n$ par

$$(\mathsf{Enc}_K^\sigma : p \mapsto \sigma(p \oplus K) \oplus K)_{K \in \{0,1\}^n}$$

avec $\oplus$ le ou exclusif bit-à-bit. On pose en plus de cela les valeurs intermédiaires $x \coloneqq p \oplus K$, $y \coloneqq \sigma(x)$ et $c \coloneqq y \oplus K = \mathsf{Enc}_K^\sigma(p)$. On appellera alors c le *chiffré* de p. Le schéma suivant illustre le fonctionnement de la fonction de chiffrement Enc :



On fixe $n \in \mathbb{N}$, on pose $N = 2^n$ et on note $\mathcal{F}_n$ l'ensemble des fonctions de $\{0,1\}^n$ dans $\{0,1\}^n$.

1. Vérifier que pour une permutation σ donnée, la famille de fonction Even-Mansour est bien un chiffre par bloc.
   Rappel : le ou exclusif est associatif, commutatif, et $\forall \alpha \in \{0,1\}^n$, $\alpha \oplus \alpha = 0$.

Dans la suite de l'exercice, on cherche une attaque sur un chiffre par bloc Even-Mansour, c'est à dire qu'on fixe une clef secrète $K \in \{0,1\}^n$ et on cherche un algorithme qui retrouve cette clef à partir de données de la forme $(p, \mathsf{Enc}(K,p))$. Concrètement, l'algorithme pourra appeler jusqu'à $\mathcal{O}(D)$ fois (avec $D \ll N$) une fonction boite-noire `oracle` qui renverra le chiffré de ce qui lui aura été passé en argument. On cherchera naturellement à minimiser D, la complexité temporelle et le cout mémoire de nos attaques.

Comme Even-Mansour dépend d'une permutation σ, on suppose dans toute la suite de l'exercice que cette dernière a été choisi arbitrairement. Pour rappel, σ est publique et connue de tous.

2. On choisit arbitrairement $\Delta \neq 0$ et on définit deux fonctions sur $\{0,1\}^n$:

   $$\varphi_1 \coloneqq p \mapsto \mathsf{Enc}_K(p) \oplus \mathsf{Enc}_K(p \oplus \Delta) \qquad \varphi_2 \coloneqq x \mapsto \sigma(x) \oplus \sigma(x \oplus \Delta)$$

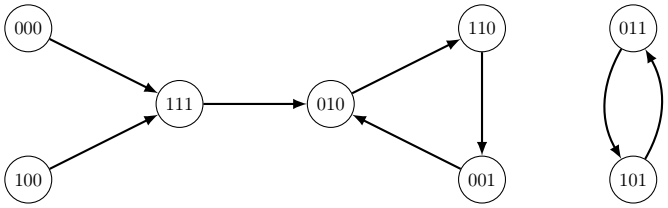
   Montrer que pour tout $p, x \in \{0,1\}^n$, $p \oplus K = x \implies \varphi_1(p) = \varphi_2(x)$.
   On supposera par la suite que la réciproque est vraie avec forte probabilité, c'est à dire que trouver p et x tels que $\varphi_1(p) = \varphi_2(x)$ permet de retrouver K en calculant $p \oplus x$.
3. On étudie maintenant l'attaque différentielle (Algorithme 1, page suivante).
   Quelle structure de donnée (ligne 1) proposez vous pour cette attaque ? Quelle quantité de mémoire (environ) est nécessaire ? Si cela a un sens, donner l'espérence du temps d'execution. On supposera pour cela que les fonctions φ_1 et φ_2 se comportent comme des fonctions tirés aléatoirement et uniformément dans $\mathcal{F}_n$, et que X ne contient pas de doublons.

Soit $f \in \mathcal{F}_n$. On définit $G_f = (S, A)$ le *graphe orienté de la fonction* f par

— $S = \{0,1\}^n$

— $A = \{(x, f(x)) \mid x \in \{0,1\}^n\}$

En d'autres termes, G est le graphe orienté dont les sommets sont l'ensemble des mots de n bits, et les arcs sont les transitions d'un mot à un autre par la fonction f. Voici un exemple de graphe fonctionnel pour $n = 3$:



Algorithme 1 Attaque Différentielle

Entrée: $\Delta \in \{0,1\}^n, D > 0$
Sortie: K

Initialiser X une structure de donnée adaptée ▷ à déterminer
pour $i \leftarrow 1$ à D **faire**
    Tirer p_i aléatoirement et uniformément dans $\{0,1\}^n$
    Calculer $p'_i \leftarrow p_i \oplus \Delta$, $c_i \leftarrow \texttt{oracle}(p_i)$ et $c'_i \leftarrow \texttt{oracle}(p'_i)$
    Stocker $(c_i \oplus c'_i, p_i)$ dans X
fin pour
tant que `Vrai` **faire**
    Tirer x_i aléatoirement et uniformément dans $\{0,1\}^n$
    Calculer $x'_i \leftarrow x_i \oplus \Delta$, $y_i \leftarrow \sigma(x_i)$ et $y'_i \leftarrow \sigma(x'_i)$
    si $(y_i \oplus y'_i, \_)$ match un élément de X **alors**
        Soit $\widetilde{p}$ le deuxième élément du tuple ci-dessus
        retourne $K \coloneqq \widetilde{p} \oplus x_i$
    fin si
fin tant que

4. On pose le jeu probabiliste suivant :

Algorithme 2 Promenade sur le graphe

Entrée: $D > 0$
Sortie: `Succès` ou `Échec`

Tirer f aléatoirement et uniformément dans $\mathcal{F}_n$
Tirer p aléatoirement et uniformément dans $\{0,1\}^n$
pour $i \leftarrow 1$ à D **faire**
    $p \leftarrow f(p)$
fin pour
Tirer x aléatoirement et uniformément dans $\{0,1\}^n$
pour $i \leftarrow 1$ à $2D$ **faire**
    $x \leftarrow f(x)$
    si $x = p$ **alors**
        retourne `Succès`
    fin si
fin pour
retourne `Échec`

Montrer que la probabilité de succès de l'Algorithme 2 peut être minorée par $\frac{D^2}{2N}$. On supposera pour cela que les chemins ne bouclent jamais, c'est à dire que les p calculés sont tous distinct (et pareil pour les x).
Indication : On pourra évaluer f de façon paresseuse, c'est à dire que tirer f aléatoirement et uniformément dans $\mathcal{F}_n$ revient à ne pas définir f initialement ; et pour chaque appel à f sur une entrée inconnue, tirer une sortie aléatoirement et uniformément dans $\{0,1\}^n$.

5. En gardant les notations des questions précédentes, on définit deux autres fonctions sur $\{0,1\}^n$:

$$\psi_1 \coloneqq p \mapsto p \oplus \varphi_1(p) \qquad \psi_2 \coloneqq x \mapsto x \oplus \varphi_2(x)$$

Montrer que pour tout $p, x \in \{0,1\}^n$, $p \oplus K = x \implies \psi_1(p) \oplus K = \psi_2(x)$.
En déduire que les graphes de ψ_1 et de ψ_2 sont isomorphes, c'est à dire qu'il existe une bijection $\chi : \{0,1\}^n \to \{0,1\}^n$ telle que $\forall p, x \in \{0,1\}^n$, $\chi(p) = x \implies \chi(\psi_1(p)) = \psi_2(x)$.

6. En déduire une attaque sur le chiffre par bloc Even-Mansour par analogie avec le jeu de la question 4. On supposera pour cela que ψ_1 et ψ_2 se comportent aussi comme des fonctions aléatoires (c.à.d. tirées aléatoirement et uniformément dans $\mathcal{F}_n$). Cette attaque aura une complexité temporelle en $\mathcal{O}(D)$, une probabilité de succès similaire à celle du jeu de la question 4, et utilisera une mémoire constante (deux emplacements mémoire par exemple). Comment améliorer la probabilité de succès de l'attaque sans augmenter D ?

Sources

[1] *TD MP**. Lycée Blaise Pascal, Clermont-Ferrand. 2018-2020.

[2] *TD L3 Informatique Fondamentale*. ENS de Lyon. 2020-2021.

[3] L. Albert. *Cours et exercices d'informatique : classes préparatoires, 1er et 2nd cycles universitaires*. Passeport pour l'informatique. Vuibert, 1998.

[4] Ismael Belghiti, Roger Mansuy, and Jill-Jênn Vie. *Les clés pour l'info : ENS et Agrégation*. 2016.

[5] Maxime Cautres and Pierre Goutagny. *Colles d'informatique MP2I, Lycée du Parc*. 2021-2022.

[6] C. Dürr and J.J. Vie. *Programmation efficace : les 128 algorithmes qu'il faut avoir compris et codés en Python au cours de sa vie*. Références sciences. Ellipses, 2016.