

## *En guise de brève introduction*

*50 kilos de patates, un sac de sciure de bois, il te sortait  
25 litres de 3 étoiles à l'alambic. Un vrai magicien ce  
Jo. Et c'est pour ça que je me permets... .*

MICHEL AUDIARD – *Les tontons flingueurs*

Voici le recueil de la plupart des sujets d'informatique posés aux concours des grandes écoles scientifiques en 2020.

Le recueil est partagé en deux grandes parties. D'abord, les épreuves d'informatique option de nos classes, auxquels s'ajoutent les épreuves du CAPES d'informatique. Ensuite, les épreuves d'informatique commune. Dans une seconde version de ce recueil, une troisième partie regroupe les épreuves de modélisation comportant au moins une question d'informatique.

Comme l'an dernier, le bulletin des concours Informatique ne sera proposé aux adhérents que sous forme électronique.

Dans ce recueil, le nom du fichier contenant chaque sujet figure dans la table des matières et en tête de chaque page. Les fichiers sont disponibles sur le site de l'UPS, à l'adresse <https://ups-cpge.fr/ups.php?module=Maths&voir=recherche>.

Vous trouverez ce recueil, au format pdf, sur le site de l'UPS, à l'adresse <https://ups-cpge.fr/index.php?rubrique=146>.

Philippe Patte    [philippe.patte@prepas.org](mailto:philippe.patte@prepas.org)

*Octobre 2020*



## Constructions et explorations de labyrinthes

Nous nous intéressons ici à l'étude de labyrinthes. La partie I propose plusieurs méthodes pour construire des labyrinthes parfaits. La partie II étudie des algorithmes pour explorer des labyrinthes en présence de monstres. Les deux parties peuvent être traitées indépendamment.

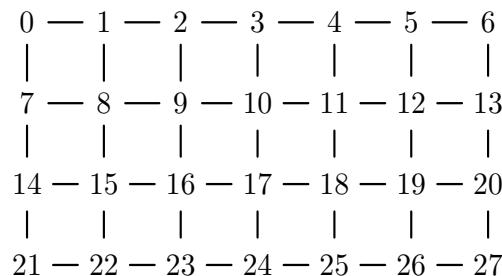
**Graphes.** Dans tout ce sujet, on considère des graphes non orientés, sans boucles. On note  $v-w$  l'arête reliant les sommets  $v$  et  $w$ . Un chemin de longueur  $n \geq 0$  du sommet  $v_0$  au sommet  $v_n$  est une séquence  $v_0-v_1-\cdots-v_{n-1}-v_n$  de sommets reliés par des arêtes. Dans tout l'énoncé, les chemins considérés sont simples, c'est-à-dire que leurs sommets sont tous distincts. Un graphe  $g$  est connexe si toute paire de sommets de  $g$  est reliée par un chemin.

Un sous-graphe d'un graphe  $g$  est un graphe dont l'ensemble des sommets est un sous-ensemble de celui de  $g$  et l'ensemble des arêtes est un sous-ensemble de celui de  $g$ . Pour un graphe  $g$  et un sous-ensemble  $X$  de sommets de  $g$ , on définit le sous-graphe de  $g$  induit par  $X$  comme le graphe dont les sommets sont  $X$  et dans lequel deux sommets sont reliés s'ils sont reliés dans  $g$ .

Pour deux entiers  $n$  et  $m$ , la grille de taille  $n \times m$  est le graphe dont les sommets sont  $\{0, 1, \dots, n \times m - 1\}$  et dont les arêtes sont les paires de sommets qui sont

- soit de la forme  $v-(v+1)$  pour  $0 \leq v < nm$  tel que  $v$  modulo  $m$  est distinct de  $m-1$ ,
- soit de la forme  $v-(v+m)$  pour  $0 \leq v < (n-1)m$ .

Par exemple, la grille de taille  $4 \times 7$  est le graphe suivant :



**Langage OCaml.** On peut créer des tableaux avec les deux fonctions `Array.make` et `Array.of_list`. L'appel de `Array.make n x` crée un tableau contenant  $n$  copies de l'élément `x`. L'appel de `Array.of_list l` crée un tableau contenant, dans l'ordre, les éléments d'une liste `l`. Les cases d'un tableau sont numérotées à partir de 0. La fonction `Array.length` renvoie la taille d'un tableau. Pour un tableau `a`, on accède à l'élément d'indice `i` avec `a.(i)` et on le modifie avec `a.(i) <- v`.

La fonction `List.iter: ('a -> unit) -> 'a list -> unit` est telle que l'appel de `List.iter f [a1 ; ... ; an]` applique la fonction `f` aux éléments `a1, ..., an` dans cet ordre. Elle est donc équivalente à `begin f a1; f a2; ...; f an; () end`.

Pour un entier  $n > 0$ , l'appel `Random.int n` renvoie un entier tiré aléatoirement dans l'ensemble  $\{0, 1, \dots, n - 1\}$ . On supposera que ce tirage est uniforme (tous les entiers sont équiprobables).

**Représentation d'un graphe en OCaml.** On représente un graphe en OCaml avec le type suivant :

```
type graphe = {
    n: int;           (* les sommets sont 0, 1, ..., n-1      *)
    adj: int list array;  (* adj.(v) est la liste des voisins de v *)
}
```

Les sommets sont les entiers  $\{0, 1, \dots, n - 1\}$ . Pour un sommet  $v$ , la liste d'adjacence `adj.(v)` contient tous les voisins de  $v$  dans un ordre arbitraire et sans doublons. On suppose donnée une fonction

```
ajoute_arete: graphe -> int -> int -> unit
```

telle que pour  $0 \leq v \leq g.n - 1$  et  $0 \leq w \leq g.n - 1$ , l'appel de `ajoute_arete g v w` ajoute au graphe `g`, si elle n'est pas déjà présente, une arête entre les sommets  $v$  et  $w$ , c'est-à-dire ajoute  $v$  à `g.adj.(w)` et  $w$  à `g.adj.(v)`. On se donne également une fonction

```
graphe_vide: int -> graphe
```

telle que `graphe_vide n` renvoie un nouveau graphe à  $n$  sommets et sans aucune arête (toutes les listes `adj.(v)` sont vides). On se donne enfin une fonction

```
aretes: graphe -> (int * int) array
```

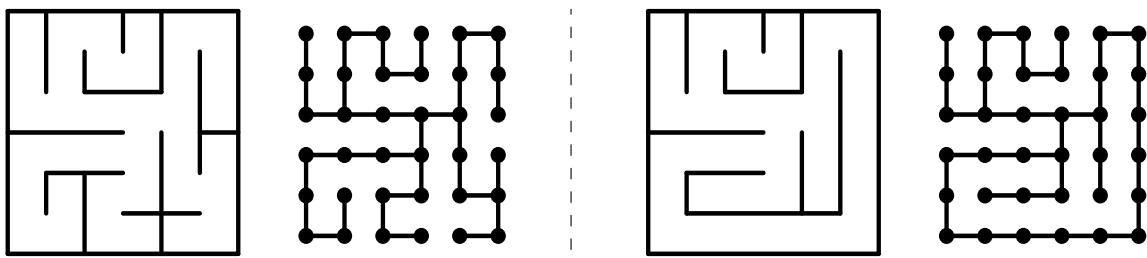
qui renvoie un tableau contenant toutes les arêtes d'un graphe donné, sous la forme de paires d'entiers. Le graphe étant non orienté, la paire  $(v, w)$  et la paire  $(w, v)$  apparaissent toutes les deux dans ce tableau.

**Complexité.** Par *complexité* (en temps) d'un algorithme  $A$  on entend le nombre d'opérations élémentaires (comparaison, addition, soustraction, multiplication, division, affectation, test, etc) nécessaires à l'exécution de  $A$  dans le cas le pire. Lorsque la complexité dépend d'un ou plusieurs paramètres  $\kappa_0, \dots, \kappa_{r-1}$ , on dit que  $A$  a une complexité en  $\mathcal{O}(f(\kappa_0, \dots, \kappa_{r-1}))$  s'il existe

une constante  $C > 0$  telle que, pour toutes les valeurs de  $\kappa_0, \dots, \kappa_{r-1}$  suffisamment grandes (c'est-à-dire plus grandes qu'un certain seuil), pour toute instance du problème de paramètres  $\kappa_0, \dots, \kappa_{r-1}$ , la complexité est au plus  $C f(\kappa_0, \dots, \kappa_{r-1})$ .

## Partie I. Construire des labyrinthes

On se donne un graphe  $g$  connexe. 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 cette partie, on cherche à construire des labyrinthes parfaits.

**Mélange de Knuth.** On commence par décrire l'algorithme du *mélange de Knuth* qui permet de permuter aléatoirement les éléments d'un tableau, et qui sera utilisé dans les trois algorithmes de construction de labyrinthe. Pour un tableau  $a$  de taille  $n$ , cet algorithme effectue  $n$  échanges : pour  $0 \leq i < n$ , la  $i$ -ième étape échange l'élément  $a.(i)$  avec l'élément  $a.(j)$ , pour un entier  $j$  pris au hasard entre 0 et  $i$  inclus.

**Question 1.** Écrire une fonction `melange_knuth: int array -> unit` qui reçoit en argument un tableau et le permute aléatoirement avec le mélange de Knuth.

On souhaite montrer que le mélange de Knuth effectue une permutation aléatoire uniforme du tableau, c'est-à-dire que toutes les permutations sont équi-probables.

**Question 2.** Soit  $a$  le tableau de taille  $n$  dans lequel  $a.(p) = p$  pour tout  $p$ . On appelle la fonction `melange_knuth` sur le tableau  $a$ , et on note  $x_p$  la valeur de la  $p$ -ième case de  $a$  après l'appel. Montrer que la probabilité que  $x_p$  soit égal à  $q$ , pour  $0 \leq p < n$  et  $0 \leq q < n$ , vaut

$$\Pr(x_p = q) = \frac{1}{n}.$$

On suppose que `Random.int` renvoie un entier aléatoire uniforme.

Indication : en notant toujours  $x_p$  la valeur courante de la  $p$ -ième case du tableau à l'étape  $i$ , montrer que l'algorithme satisfait l'invariant de boucle :

1. pour tous  $0 \leq p < i$  et  $0 \leq q < i$ , on a  $\Pr(x_p = q) = \frac{1}{i}$ ,
2. pour tout  $i \leq p < n$ , on a  $x_p = p$ .

**Parcours en profondeur aléatoire.** On rappelle qu'on considère un graphe  $g$  connexe et qu'on appelle labyrinthe sur  $g$  un sous-graphe connexe de  $g$  qui a le même ensemble de sommets que  $g$ . Pour construire un labyrinthe aléatoire sur  $g$ , on peut utiliser l'algorithme suivant. On effectue un *parcours en profondeur* (noté DFS par la suite) du graphe  $g$ . 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  $v — w$  au labyrinthe et on traite  $w$ .

**Question 3.** Écrire une fonction `labyrinthe1: graphe -> graphe` qui prend en argument un graphe  $g$  connexe et renvoie un labyrinthe sur  $g$  construit avec un DFS aléatoire. On suggère d'écrire le parcours en profondeur comme une fonction récursive. On ne demande pas de vérifier que  $g$  est connexe.

**Question 4.** Montrer que le labyrinthe construit par la fonction `labyrinthe1` est parfait.

Indication : identifier un invariant du parcours en profondeur réalisé par `labyrinthe1`.

**Classes disjointes.** On s'intéresse maintenant à une structure de données qui sera utilisée plus loin pour construire des labyrinthes par d'autres algorithmes. Cette structure de données représente une relation d'équivalence sur l'ensemble  $\{0, 1, \dots, n - 1\}$ . Pour cela on choisit un représentant arbitraire dans chaque classe d'équivalence et on se donne un tableau `lien` de taille  $n$  qui va permettre de retrouver ce représentant. Pour un représentant  $r$ , on a `lien.(r) = r`. Pour tout autre élément  $i$  de la classe de  $r$ , on aboutit à  $r$  en suivant les valeurs données par le tableau `lien` (c'est-à-dire que `lien.(i) = r`, ou `lien.(lien.(i)) = r`, ou `lien.(lien.(lien.(i))) = r`, etc). En particulier,  $i$  et `lien.(i)` sont toujours dans la même classe d'équivalence. Par exemple, le tableau

$$\text{lien} = \boxed{2 \mid 1 \mid 5 \mid 3 \mid 3 \mid 5 \mid 5}$$

représente la relation d'équivalence dont les classes sont  $\{0, 2, 5, 6\}$ ,  $\{1\}$  et  $\{3, 4\}$ , pour lesquelles on a choisi les représentants 5, 1 et 3. On se donne le type OCaml suivant pour cette structure de données :

```
type classes_disjointes = {
  lien: int array;
}
```

**Question 5.** Écrire une fonction `cd_trouve: classes_disjointes -> int -> int` qui prend en arguments une relation d'équivalence sur  $\{0, 1, \dots, n - 1\}$  et un entier  $i$ , avec  $0 \leq i < n$ , et renvoie le représentant de la classe de  $i$ .

On note que la complexité de `cd_trouve` dans le pire des cas est proportionnelle à la longueur du plus long chemin d'un élément à son représentant. Formellement, la longueur du chemin d'un élément  $i$  à son représentant  $r$  est définie comme le nombre d'éléments rencontrés en partant de  $i$  et en suivant les valeurs du tableau `lien` (en comptant  $i$  lui-même mais sans compter  $r$ ).

On veut maintenant pouvoir modifier la relation d'équivalence, en offrant une opération `cd_union` permettant de fusionner les classes d'équivalence de deux éléments `i` et `j` donnés. L'idée consiste à chercher les représentants `ri` et `rz` des classes d'équivalence de `i` et `j`, avec la fonction `cd_trouve`, puis de faire pointer l'un vers l'autre en modifiant le tableau `lien`. Le choix du représentant `ri` ou `rz` pour la classe fusionnée n'est pas anodin, car il affecte le coût des opérations `cd_trouve` ultérieures.

On introduit donc un second tableau `rang` de taille `n`. Pour chaque représentant `r`, la valeur de `rang.(r)` donne la longueur du plus long chemin d'un élément de la classe de `r` à `r`. Pour un élément `i` qui n'est pas un représentant, la valeur de `rang.(i)` n'est pas significative et ne sera jamais utilisée. Par exemple, les tableaux

$$\text{lien} = \boxed{2 \mid 1 \mid 5 \mid 3 \mid 3 \mid 5 \mid 5} \quad \text{et} \quad \text{rang} = \boxed{0 \mid 0 \mid 1 \mid 1 \mid 0 \mid 2 \mid 0}$$

représentent la relation d'équivalence dont les classes sont  $\{0, 2, 5, 6\}$ ,  $\{1\}$  et  $\{3, 4\}$ , mais les valeurs du tableau `rang` en positions 0, 2, 4, 6 ne sont pas significatives.

On modifie donc le type `classes_disjointes` de la façon suivante :

```
type classes_disjointes = {
    lien: int array;
    rang: int array;
}
```

Le code de la fonction `cd_trouve` reste inchangé.

**Question 6.** Écrire une fonction `cd_union: classes_disjointes -> int -> int -> unit` qui prend en arguments une relation d'équivalence sur  $\{0, 1, \dots, n - 1\}$  et deux entiers `i` et `j`, avec  $0 \leq i, j < n$ , et fusionne les classes d'équivalence de `i` et `j`. Lorsque `i` et `j` sont déjà dans la même classe, cette fonction ne fait rien. Sinon, elle choisit pour nouveau représentant celui dont le rang est maximal.

**Question 7.** On appelle rang d'une classe le rang de son représentant. Montrer que la fonction `cd_union` préserve les deux invariants suivants :

1. tout classe de rang  $k$  possède au moins  $2^k$  éléments;
2. dans une classe de rang  $k$ , la longueur du plus long chemin jusqu'au représentant est égale à  $k$ .

On se donne une fonction `cd_init: int -> classes_disjointes` qui prend en argument un entier  $n \geq 0$  et renvoie une relation d'équivalence sur  $\{0, 1, \dots, n - 1\}$  dont toutes les classes sont des singletons. Pour tout `i`, on a donc `lien.(i) = i` et `rang.(i) = 0`.

**Question 8.** Déduire de la question 7 que, pour une relation d'équivalence sur  $\{0, 1, \dots, n - 1\}$  construite à partir de `cd_init` et uniquement avec des appels à `cd_union`, la complexité de `cd_trouve` est en  $O(\log n)$ .

**Application à la construction d'un labyrinthe.** On utilise maintenant la structure de classes disjointes pour construire un labyrinthe parfait `h` à partir d'un graphe `g` connexe à `n` sommets. L'algorithme procède ainsi :

1. on construit une relation d'équivalence sur  $\{0, 1, \dots, n - 1\}$  avec `cd_init`;
2. on construit le tableau de toutes les arêtes du graphe `g`, puis on le mélange avec `melange_knuth`;
3. on parcourt ce tableau mélangé et, pour chaque arête  $v — w$ , si  $v$  et  $w$  ne sont pas dans la même classe d'équivalence, on ajoute l'arête  $v — w$  au labyrinthe `h` et on fusionne les classes de  $v$  et  $w$  avec `cd_union`.

**Question 9.** Écrire une fonction `labyrinthe2: graphe -> graphe` qui prend en argument un graphe `g` connexe et renvoie un labyrinthe sur `g` construit avec cet algorithme. On rappelle l'existence de la fonction `aretes` qui renvoie un tableau contenant toutes les arêtes d'un graphe donné, sous la forme de paires d'entiers. On ne demande pas de vérifier que `g` est connexe.

**Question 10.** Montrer que l'étape 3 de l'algorithme maintient l'invariant suivant : pour toute classe d'équivalence  $X$ , le sous-graphe de `h` induit par  $X$  est un labyrinthe parfait du sous-graphe de `g` induit par  $X$ . En déduire que le labyrinthe construit par la fonction `labyrinthe2` est parfait.

**Algorithme d'Eller.** On considère maintenant un troisième algorithme pour construire un labyrinthe parfait, dans le cas particulier où le graphe `g` de départ est la grille de taille  $n \times m$  définie dans le préambule. Comme pour l'algorithme précédent, on utilise une relation d'équivalence sur les sommets  $\{0, 1, \dots, n \times m - 1\}$  de la grille `g`. Initialement, toutes les classes d'équivalence sont des singltons. Dans la suite, on dit qu'on *connecte* deux sommets  $i$  et  $j$ , voisins dans `g`, pour dire qu'on fusionne leurs classes d'équivalence avec `cd_union` et qu'on ajoute l'arête  $i — j$  dans le labyrinthe en construction.

L'algorithme d'Eller procède en balayant la grille de haut en bas :

1. pour chaque ligne, sauf la dernière :
  - (a) on parcourt toutes les arêtes  $v — (v + 1)$  de cette ligne, dans un ordre aléatoire. Si les sommets  $v$  et  $v + 1$  ne sont pas dans la même classe d'équivalence, on les connecte avec probabilité  $1/2$ ;
  - (b) on choisit un sous-ensemble aléatoire de sommets de cette ligne qui contient au moins un sommet de chaque classe d'équivalence. On relie chacun des sommets de ce sous-ensemble avec le sommet situé juste en dessous sur la ligne suivante.

2. on parcourt toutes les arêtes  $v \rightarrow (v+1)$  de la dernière ligne, dans un ordre aléatoire. On connecte les sommets  $v$  et  $v + 1$  si ils ne sont pas dans la même classe d'équivalence.

**Question 11.** Montrer que l'algorithme d'Eller construit un labyrinthe parfait.

**Question 12.** Prouver que tout labyrinthe parfait sur la grille  $g$  peut être obtenu par l'algorithme d'Eller.

## Partie II. Résoudre un labyrinthe

On se pose maintenant la question de résoudre un labyrinthe, c'est-à-dire de chercher un chemin d'un sommet source `src` donné à un sommet destination `dst` donné. On suppose toujours que le labyrinthe est connexe. En revanche, on ne suppose pas le labyrinthe parfait, c'est-à-dire qu'il peut exister plusieurs chemins de la source à la destination. Dans cette partie, on considère trois variantes de la recherche d'un chemin.

**Files à deux bouts.** On commence par programmer une structure de données de file à deux bouts, qui sera utilisée dans les trois algorithmes de recherche de chemin. Une telle file est représentée par le type suivant :

```
type file = {
    contenu: int array;
    mutable debut: int;
    mutable taille: int;
}
```

La file contient `taille` éléments, stockés dans le tableau `contenu` à partir de l'indice `debut` et dans le sens des indices croissants. Le tableau a une taille `cap`, qui est la capacité maximale de la file. Si  $\text{debut} + \text{taille} \leq \text{cap}$ , les éléments de la file sont stockés de façon consécutive dans le tableau `contenu`, entre `debut` et `debut + taille - 1`. Sinon, le stockage des éléments se poursuit au début du tableau `contenu`, jusqu'à la position `debut + taille - 1 - cap`. Dans tous les cas, on a les inégalités suivantes :  $0 \leq \text{debut} < \text{cap}$  et  $0 \leq \text{taille} \leq \text{cap}$ .

Voici deux exemples de files de capacité 7 contenant 4 éléments (notés X) :

|   |   |   |   |   |   |   |  |
|---|---|---|---|---|---|---|--|
|   |   | X | X | X | X |   |  |
| X | X |   |   |   | X | X |  |

`cap = 7, debut = 2, taille = 4`

`cap = 7, debut = 5, taille = 4`

La file est construite par une fonction `file_vide: int -> file` qui prend la capacité en argument, puis manipulée avec les quatre opérations suivantes :

```
ajoute_debut: file -> int -> unit
retire_debut: file -> int
ajoute_fin : file -> int -> unit
retire_fin : file -> int
```

L'appel de `ajoute_debut f e` ajoute l'élément `e` au début de la file `f` (sans déplacer ou modifier les éléments déjà présents), en supposant que la file n'est pas pleine. L'appel de `retire_debut f` retire et renvoie l'élément du début de la file `f` (sans déplacer ou modifier les autres éléments), en supposant que la file n'est pas vide. Les deux opérations `ajoute_fin` et `retire_fin` opèrent de manière similaire à la fin de la file.

**Question 13.** Donner le code de la fonction `ajoute_debut`.

**Question 14.** Donner le code de la fonction `retire_debut`.

**Parcours en largeur.** Pour trouver la longueur d'un plus court chemin dans un labyrinthe à  $n$  sommets, on effectue un parcours en largeur, en partant du sommet source `src` et en s'arrêtant quand on trouve le sommet destination `dst`. On procède ainsi :

1. créer un tableau `distance` de taille  $n$ , initialisé avec la valeur  $-1$  dans toutes les cases ;
2. créer une file à deux bouts `f` de capacité  $n$  ;
3. ajouter le sommet source `src` dans `f` et initialiser `distance.(src)` à  $0$  ;
4. tant que la file `f` n'est pas vide :
  - (a) retirer l'élément `v` au début de `f` ;
  - (b) si `v = dst` alors on a terminé et la réponse est `distance.(dst)` ;
  - (c) sinon, pour chaque voisin `w` de `v` pour lequel `distance.(w)` vaut  $-1$ , ajouter `w` à la fin de `f` et donner à `distance.(w)` la valeur `distance.(v)+1`.

**Question 15.** Montrer que cet algorithme renvoie bien la longueur d'un plus court chemin de la source `src` à la destination `dst`. On rappelle que le graphe est supposé connexe.

**En croisant un minimum de monstres.** On suppose maintenant qu'il y a des monstres sur certains sommets du labyrinthe (les sommets `src` et `dst` pouvant, comme les autres, accueillir des monstres). Notre but est maintenant de chercher des chemins du sommet source `src` au sommet destination `dst` qui passent par un minimum de monstres (en comptant les monstres sur `src` et `dst` s'il y en a). Pour cela, on peut modifier l'algorithme de parcours en largeur donné plus haut de la façon suivante :

- le tableau `distance` contient maintenant, pour chaque sommet `v` déjà atteint, un couple  $(m, \ell)$  où  $m$  est le nombre de monstres et  $\ell$  la longueur d'un chemin de `src` à `v` qui passe par un minimum de monstres ;
- quand on atteint un nouveau sommet `w` pour la première fois, on le rajoute à la fin de la file `f` s'il y a un monstre sur le sommet `w` et au début sinon.

**Question 16.** Compléter le code de la fonction `minimum_monstres`: `graphe -> bool array -> int -> int -> int * int` ci-dessous qui implémente cet algorithme. Les quatre arguments sont le graphe `g`, un tableau de booléens `monstre` qui indique pour chaque sommet s'il y a un monstre, et les sommets source `src` et destination `dst`. La fonction renvoie le couple  $(m, \ell)$  où  $m$  est le nombre de monstres et  $\ell$  la longueur d'un chemin de `src` à `v` qui passe par un minimum de monstres.

```

let minimum_monstres g monstre src dst =
  let distance = Array.make g.n (-1,-1) in (* 1 *)
  let f = file_vide g.n in (* 2 *)
  ...
  let rec loop () = (* 4 *)
    let v = retire_debut f in (* a *)
    if v = dst then distance.(v) else begin (* b *)
      List.iter (fun w -> (* c *)
        ... (* pour chaque voisin w de v *)
      ) g.adj.(v);
      loop () (* d *)
    end
  loop ()

```

La valeur initiale  $(-1, -1)$  dans le tableau `distance` est utilisée pour indiquer qu'un sommet n'a pas encore été vu. La syntaxe de la fonction `List.iter` est rappelée dans le préambule. Les commentaires dans le code fourni font référence aux étapes de l'algorithme du parcours en largeur. On pourra se contenter de donner uniquement les morceaux de code correspondant aux étapes (3) et (4c).

**Question 17.** On admet que l'algorithme de la question 16 trouve un chemin qui passe par un minimum de monstres. En revanche, il ne donne pas nécessairement un chemin de longueur minimale parmi ceux qui passent par un minimum de monstres. Donner un exemple de labyrinthe pour lequel la fonction `minimum_monstres` trouve un chemin dont la longueur n'est pas minimale parmi ceux qui passent par un minimum de monstres.

**Minimum de monstres et longueur minimale.** On considère maintenant un algorithme qui cherche un chemin de longueur minimale parmi ceux qui passent par un minimum de monstres.

Le coût d'un chemin est défini comme la paire  $(m, \ell)$  où  $m$  est le nombre de monstres le long de ce chemin (y compris à ses extrémités) et  $\ell$  la longueur de ce chemin. On ordonne les coûts lexicographiquement :  $(m_1, \ell_1) < (m_2, \ell_2)$  si et seulement si  $m_1 < m_2$  ou  $m_1 = m_2$  et  $\ell_1 < \ell_2$ . La distance entre deux sommets est définie comme le coût minimal d'un chemin entre ces sommets.

Pour trouver la distance entre la source et la destination, l'idée consiste à effectuer plusieurs parcours en largeur successifs, pour des nombres de monstres rencontrés de plus en plus grand. On fait un premier parcours en largeur sur les chemins qui ne passent par aucun monstre. Si la destination est atteinte, on a bien trouvé un plus court chemin (par la question 15) ne passant par aucun monstre. Sinon, on a trouvé tous les monstres à distance de type  $(1, \ell)$ . On démarre alors un nouveau parcours en largeur depuis les monstres à distance  $(1, \ell)$  avec  $\ell$  minimale. Pendant ce parcours, on prend soin d'insérer au début de la file d'attente du parcours en largeur chaque monstre à distance de type  $(1, \ell)$  quand la longueur de chemin  $\ell$  est atteinte par le parcours en largeur. Si la destination est atteinte, on a trouvé un plus court chemin passant par un seul monstre. Sinon, on a trouvé tous les monstres à distance de type  $(2, \ell)$  et on continue.

Pour mettre en œuvre cet algorithme, on utilise trois files, **f**, **sources\_courantes** et **sources\_suivantes**. Par ailleurs, on utilise toujours un tableau **distance** qui contient maintenant, pour chaque sommet **v** déjà atteint, sa distance à la source sous la forme d'un couple  $(m, \ell)$ . On procède ainsi :

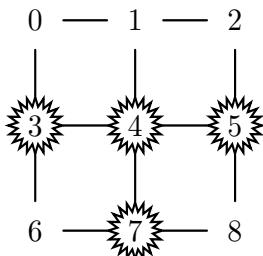
1. initialisation :

- créer un tableau **distance** de taille **n**, initialisé avec la valeur  $(-1, -1)$  dans toutes les cases,
- créer trois files **f**, **sources\_courantes** et **sources\_suivantes** de capacité **n**,
- ajouter le sommet source **src** dans **f** et initialiser **distance.(src)** à  $(0, 0)$  s'il n'y a pas de monstre sur le sommet **src** et à  $(1, 0)$  sinon ;

2. tant que la destination n'est pas atteinte :

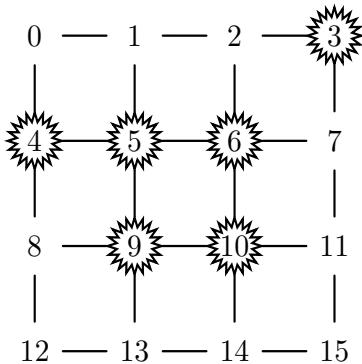
- (a) si la file **f** est vide,
  - i. si la file **source\_courantes** est vide, déplacer tous les éléments de **sources\_suivantes** dans **sources\_courantes**,
  - ii. déplacer tous les éléments de **sources\_courantes** avec  $\ell$  minimal dans **f** ;
- (b) retirer l'élément **v** au début de **f** et soit  $(m, \ell) = \text{distance.}(v)$  ;
- (c) si **v = dst** alors on a terminé et la réponse est  $(m, \ell)$  ;
- (d) tant qu'il y a au début de **sources\_courantes** un sommet à distance  $(m, \ell + 1)$ , le retirer de **sources\_courantes** et l'ajouter à la fin de **f** ;
- (e) pour chaque voisin **w** de **v** pour lequel **distance.(w)** vaut  $(-1, -1)$ ,
  - s'il n'y a pas de monstre sur **w**, ajouter **w** à la fin de **f** et donner à **distance.(w)** la valeur  $(m, \ell + 1)$  ;
  - s'il y a un monstre sur **w**, ajouter **w** à la fin de **sources\_suivantes** et donner à **distance.(w)** la valeur  $(m + 1, \ell + 1)$ .

Par exemple, on considère le labyrinthe de neuf sommets ci-dessous, avec **src** = 0 et **dst** = 8 et des monstres sur les sommets encerclés. Le tableau ci-dessous déroule l'algorithme sur cet exemple. La première colonne indique les étapes effectuées, la deuxième les affectations faites dans le tableau **distance** et les trois dernières l'état des trois files à la fin de chaque étape (on n'écrit rien lorsqu'il n'y a pas de changement).



| étape | affectations<br>distance                       | f           | sources_<br>courantes | sources_<br>suivantes |
|-------|--|-------------|-----------------------|-----------------------|
| 1     | $0 \leftarrow (0, 0)$                          | 0           | $\emptyset$           | $\emptyset$           |
| 2be   | $1 \leftarrow (0, 1)$<br>$3 \leftarrow (1, 1)$ | 1           |                       | 3                     |
| 2be   | $2 \leftarrow (0, 2)$<br>$4 \leftarrow (1, 2)$ | 2           |                       | 3, 4                  |
| 2be   | $5 \leftarrow (1, 3)$                          | $\emptyset$ |                       | 3, 4, 5               |
| 2a    |  | 3           | 4, 5                  | $\emptyset$           |
| 2bde  | $6 \leftarrow (1, 2)$                          | 4, 6        | 5                     |                       |
| 2bde  | $7 \leftarrow (2, 3)$                          | 6, 5        | $\emptyset$           | 7                     |
| 2be   | $8 \leftarrow (1, 4)$                          | 6, 8        |                       |                       |
| 2b    |  | 8           |                       |                       |
| 2bc   |  | $\emptyset$ |                       |                       |

**Question 18.** Dérouler de la même façon l'algorithme sur le labyrinthe à 16 sommets suivant, où les monstres sont sur les sommets encerclés, et avec  $\text{src} = 0$  et  $\text{dst} = 11$  :



**Question 19.** Montrer que cet algorithme renvoie bien la distance (pour le coût lexicographique défini plus haut) de la source  $\text{src}$  à la destination  $\text{dst}$ . On rappelle que le labyrinthe est supposé connexe. Le candidat pourra se contenter de donner une liste complète d'invariants permettant de déduire la correction de l'algorithme.

**Question 20.** Montrer qu'il est également possible de trouver un chemin de coût minimal dans le labyrinthe avec les monstres (pour le coût lexicographique défini plus haut) en construisant un graphe orienté pondéré et en utilisant un algorithme de plus court chemin (de type Dijkstra ou Floyd-Warshall).

\* \*  
\*

## Programmation fonctionnelle, sémantique et topologie

*Le sujet comporte 13 pages, numérotées de 1 à 13.*

*Pour le traitement des trois dernières parties du sujet, il est recommandé de détacher la page 13 afin que son contenu soit facilement consultable.*

---

Dans ce sujet, on considère un langage de programmation fonctionnelle minimalistre, proche mais différent d'OCaml. On définit rigoureusement la sémantique (signification) des programmes au moyen d'objets mathématiques usuels (e.g. entiers, fonctions, ordres partiels) et on l'utilise pour concevoir et prouver des programmes.

- La première partie introduit un système de types simples, ainsi que la notion d'ordre partiel complet. On y construit, pour chaque type, un certain ordre partiel complet.
  - La deuxième partie introduit le langage de programmation, et définit l'interprétation des programmes.
  - La troisième partie étudie des notions d'inspiration topologique issues de notre langage de programmation : on y parlera d'ouverts, de fermés et de compacts calculatoires.
  - La quatrième partie porte sur la preuve de compacité de l'espace de Cantor : on y démontrera qu'il existe un programme  $p$  qui permet de déterminer, pour tout programme  $q$ , si  $q$  termine sur toute suite de bits.
- 

## Notations et définitions

- On note  $\mathcal{P}(E)$  l'ensemble des parties d'un ensemble  $E$ .
- On note  $E \uplus F$  l'union disjointe des ensembles  $E$  et  $F$ .
- Soit  $E$  un ensemble et  $F$  un ensemble de sous-ensembles de  $E$ , c'est à dire  $F \subseteq \mathcal{P}(E)$ . On note  $\bigcap F$  l'intersection de tous les ensembles de  $F$  et  $\bigcup F$  l'union de tous les ensembles de  $F$ . Quand  $F$  est vide,  $\bigcap F = E$  et  $\bigcup F = \emptyset$ .
- La partie entière inférieure est notée  $\lfloor \cdot \rfloor$ . En particulier, si  $n \in \mathbb{N}$ , alors  $\lfloor n/2 \rfloor$  est le plus grand entier  $m \in \mathbb{N}$  tel que  $2m \leq n$ .
- Un **ordre partiel**  $(E, \leq_E)$  est un ensemble  $E$  équipé d'une relation binaire  $\leq_E$  qui est réflexive, transitive et antisymétrique. Dans ce contexte, on notera  $<_E$  l'ordre strict induit par  $\leq_E$ , dont on rappelle la définition :  $x <_E y$  quand  $x \leq_E y$  et  $x \neq y$ .
- Soit  $(E, \leq_E)$  un ordre partiel et  $(x_i)_{i \in \mathbb{N}}$  une suite d'éléments de  $E$ . On dit que  $y \in E$  est un **majorant** de la suite quand  $x_i \leq_E y$  pour tout  $i \in \mathbb{N}$ . On dit que  $y$  est une **borne supérieure** de la suite quand  $y$  est un majorant de la suite et qu'on a  $y \leq_E z$  pour tout majorant  $z$  de la suite.

## Partie I

On définit les **types** comme les expressions syntaxiques engendrées à partir des **types de base** `unit`, `bool` et `nat` au moyen de l'opérateur binaire  $(\_ \Rightarrow \_)$ , qui correspond intuitivement à l'opérateur  $(\_ \rightarrow \_)$  du langage OCaml. Par exemple, `nat` et `unit`  $\Rightarrow$  (`bool`  $\Rightarrow$  `bool`) sont des types. Dans la suite, les types seront représentés par la lettre  $\tau$ .

L'objectif de cette partie est de définir, pour tout type  $\tau$ , un certain ensemble partiellement ordonné  $([\![\tau]\!], \leq_\tau)$ . Dans la partie suivante, les programmes de type  $\tau$  seront interprétés comme des éléments de  $([\![\tau]\!])$ .

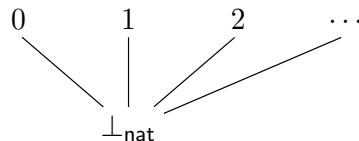
Étant donné un ordre partiel  $(A, \leq_A)$ , une **suite croissante** de  $A$  est une suite  $(a_i)_{i \in \mathbb{N}}$  d'éléments de  $A$  telle que, pour tout  $i \in \mathbb{N}$ ,  $a_i \leq_A a_{i+1}$ . On dit que  $(A, \leq_A)$  est un **ordre partiel complet** quand  $(A, \leq_A)$  est un ordre partiel et que toute suite croissante  $(a_i)_{i \in \mathbb{N}}$  d'éléments de  $A$  admet une borne supérieure, notée  $\sup_{i \in \mathbb{N}}^A a_i$ .

Étant donnés deux ordres partiels complets  $(A, \leq_A)$  et  $(B, \leq_B)$ , une application  $f : A \rightarrow B$  est dite **croissante** quand, pour tous  $a_1 \in A$  et  $a_2 \in A$  tels que  $a_1 \leq_A a_2$ , on a  $f(a_1) \leq_B f(a_2)$ . L'application  $f$  est dite **continue** quand elle est croissante et que, pour toute suite croissante  $(a_i)_{i \in \mathbb{N}}$  de  $A$ , on a  $f(\sup_{i \in \mathbb{N}}^A a_i) = \sup_{i \in \mathbb{N}}^B f(a_i)$ .

Définissons dans un premier temps  $([\![\tau]\!], \leq_\tau)$  pour les types de base. À cet effet, on se donne des constantes distinctes  $\perp_{\text{nat}}$ ,  $\perp_{\text{bool}}$ ,  $\perp_{\text{unit}}$ , `tt`, `ff` et `uu`. Les trois dernières correspondent intuitivement aux valeurs OCaml `true`, `false` et `()`. Les constantes  $\perp_\tau$  serviront à représenter l'absence de résultat des programmes de type  $\tau$  dont l'évaluation ne termine pas. On définit :

$$\begin{aligned} \mathbb{B} &\stackrel{\text{def}}{=} \{\text{tt}, \text{ff}\} & \mathbb{U} &\stackrel{\text{def}}{=} \{\text{uu}\} \\ [\![\text{nat}]\!] &\stackrel{\text{def}}{=} \mathbb{N} \uplus \{\perp_{\text{nat}}\} & [\![\text{bool}]\!] &\stackrel{\text{def}}{=} \mathbb{B} \uplus \{\perp_{\text{bool}}\} & [\![\text{unit}]\!] &\stackrel{\text{def}}{=} \mathbb{U} \uplus \{\perp_{\text{unit}}\} \end{aligned}$$

Pour chaque type de base  $b \in \{\text{unit}, \text{nat}, \text{bool}\}$  on définit enfin  $\leq_b$  comme suit : pour tous  $e_1, e_2 \in [\![b]\!]$ ,  $e_1 \leq_b e_2$  ssi  $e_1 = e_2$  ou  $e_1 = \perp_b$ . Il est clair que cela définit des ordres partiels ; inutile de le vérifier dans les réponses aux questions. L'ordre partiel  $\leq_{\text{nat}}$  peut être représenté graphiquement comme suit :



**Question 1.1.** On souhaite vérifier que  $([\![\text{nat}]\!], \leq_{\text{nat}})$  est un ordre partiel complet :

- a. Caractériser les suites croissantes de  $([\![\text{nat}]\!], \leq_{\text{nat}})$ .
- b. En déduire que toute suite croissante de  $([\![\text{nat}]\!], \leq_{\text{nat}})$  admet une borne supérieure.

On admet que  $([\![\text{unit}]\!], \leq_{\text{unit}})$  et  $([\![\text{bool}]\!], \leq_{\text{bool}})$  sont aussi des ordres partiels complets.

Étant donnés deux types  $\tau_1$  et  $\tau_2$  pour lesquels on suppose avoir construit les ordres partiels complets  $([\![\tau_1]\!], \leq_{\tau_1})$  et  $([\![\tau_2]\!], \leq_{\tau_2})$ , on définit  $[\![\tau_1 \Rightarrow \tau_2]\!]$  comme l'ensemble des applications continues de  $[\![\tau_1]\!]$  dans  $[\![\tau_2]\!]$  :

$$[\![\tau_1 \Rightarrow \tau_2]\!] \stackrel{\text{def}}{=} \{f : [\![\tau_1]\!] \rightarrow [\![\tau_2]\!] \mid f \text{ est continue}\}$$

On définit ensuite la relation  $\leq_{\tau_1 \Rightarrow \tau_2}$  en posant, pour tous  $f, g \in \llbracket \tau_1 \Rightarrow \tau_2 \rrbracket$  :

$$f \leq_{\tau_1 \Rightarrow \tau_2} g \quad \text{ssi} \quad f(e) \leq_{\tau_2} g(e) \text{ pour tout } e \in \llbracket \tau_1 \rrbracket$$

Enfin, on définit  $\perp_{\tau_1 \Rightarrow \tau_2}$  comme la fonction  $f : \llbracket \tau_1 \rrbracket \rightarrow \llbracket \tau_2 \rrbracket$  telle que  $f(e) = \perp_{\tau_2}$  pour tout  $e \in \llbracket \tau_1 \rrbracket$ .

On admettra que  $(\llbracket \tau_1 \Rightarrow \tau_2 \rrbracket, \leq_{\tau_1 \Rightarrow \tau_2})$  est un ordre partiel, et que  $\perp_{\tau_1 \Rightarrow \tau_2}$  est son plus petit élément.

**Question 1.2.** On considère dans cette question le cas particulier où  $\tau_1 = \text{nat}$  et  $\tau_2 = \text{unit}$ .

- a. Montrer que toute application croissante  $f : \llbracket \text{nat} \rrbracket \rightarrow \llbracket \text{unit} \rrbracket$  est continue.
- b. Combien y a-t-il d'éléments  $f \in \llbracket \text{nat} \Rightarrow \text{unit} \rrbracket$  tels que  $f(\perp_{\text{nat}}) \neq \perp_{\text{unit}}$  ?
- c. Donner, sans justifier, une suite strictement croissante de  $\llbracket \text{nat} \Rightarrow \text{unit} \rrbracket$ , c'est à dire une suite  $(f_i)_{i \in \mathbb{N}}$  telle que  $f_i <_{\text{nat} \Rightarrow \text{unit}} f_{i+1}$  pour tout  $i \in \mathbb{N}$ .

**Question 1.3.** Soient  $\tau_1$  et  $\tau_2$  des types tels que  $(\llbracket \tau_1 \rrbracket, \leq_{\tau_1})$  et  $(\llbracket \tau_2 \rrbracket, \leq_{\tau_2})$  sont des ordres partiels complets.

- a. Soit  $(f_i)_{i \in \mathbb{N}}$  une suite croissante de  $(\llbracket \tau_1 \Rightarrow \tau_2 \rrbracket, \leq_{\tau_1 \Rightarrow \tau_2})$ . Pour tout  $e \in \llbracket \tau_1 \rrbracket$ ,  $(f_i(e))_{i \in \mathbb{N}}$  est une suite croissante de  $(\llbracket \tau_2 \rrbracket, \leq_{\tau_2})$  par définition de  $\leq_{\tau_1 \Rightarrow \tau_2}$ , et admet donc une borne supérieure.

Montrer que l'application  $f' : \llbracket \tau_1 \rrbracket \rightarrow \llbracket \tau_2 \rrbracket$  définie par  $f'(e) = \sup_{i \in \mathbb{N}} f_i(e)$  est continue.

- b. Montrer que  $(\llbracket \tau_1 \Rightarrow \tau_2 \rrbracket, \leq_{\tau_1 \Rightarrow \tau_2})$  est un ordre partiel complet.

Tout type étant construit (en un nombre fini d'étapes) à partir des types de base au moyen de l'opérateur  $(\_ \Rightarrow \_)$ , les définitions et résultats précédents nous donnent pour chaque type  $\tau$  un ordre partiel complet  $(\llbracket \tau \rrbracket, \leq_\tau)$ .

## Partie II

Nous introduisons dans cette partie les programmes et leur interprétation. On suppose un ensemble infini dénombrable de **variables**, noté  $\mathcal{X}$  et dont les éléments seront représentés par les lettres  $x, y$  et  $z$  dans la suite. Les **programmes**, qui seront représentés par les lettres  $p$  et  $q$ , sont des expressions syntaxiques données par la grammaire suivante, où  $n$  dénote un entier arbitraire :

$$\begin{aligned} p ::= & \text{ uu} \\ & | \text{ tt} | \text{ ff} | \text{ if } p_0 \text{ then } p_1 \text{ else } p_2 | p_1 = p_2 \\ & | n | p_1 + p_2 | p_1 - p_2 | p_1 \times p_2 | p_0/2 \\ & | x | (\text{fun } x \rightarrow p_0) | (p_1 p_2) | (\text{rec } p_0) \\ & | (p_1 \|_{\text{u}} p_2) \end{aligned}$$

Cela signifie que l'ensemble des programmes est le plus petit ensemble tel que : les constantes **uu**, **tt** et **ff** sont des programmes ; tout  $n \in \mathbb{N}$  est un programme ; tout  $x \in \mathcal{X}$  est un programme ; si  $p_0, p_1$  et  $p_2$  sont des programmes alors **if**  $p_0$  **then**  $p_1$  **else**  $p_2$  aussi ; etc. Dans la plupart des cas, les constructions de programmes utilisent les mêmes notations qu'en OCaml, et la signification *intuitive* des constructions sera analogue à celle d'OCaml. Certaines autres constructions sont nouvelles. Dans tous les cas, une signification mathématique précise sera donnée plus loin.

Nous définissons ensuite une notion de typage pour nos programmes. Contrairement au cas du langage OCaml, un programme de notre langage admettra au plus un type. Afin de fixer l'unique type de chaque variable, on se donne une application  $t$  des variables dans les types. Nous supposerons que pour tout type  $\tau$  il existe une infinité de variables  $x$  telles que  $t(x) = \tau$ . La **relation de typage** exprimant qu'un programme  $p$  est de type  $\tau$ , notée  $p : \tau$ , est ensuite définie par les équivalences suivantes, pour tous programmes  $p_0, p_1, p_2$ , pour tout type  $\tau$ , et pour tous  $n \in \mathbb{N}$  et  $x \in \mathcal{X}$  :

$$\begin{aligned} \text{uu} : \tau &\Leftrightarrow \tau = \text{unit} \\ \text{tt} : \tau &\Leftrightarrow \tau = \text{bool} \\ \text{ff} : \tau &\Leftrightarrow \tau = \text{bool} \\ n : \tau &\Leftrightarrow \tau = \text{nat} \\ (\text{if } p_0 \text{ then } p_1 \text{ else } p_2) : \tau &\Leftrightarrow p_0 : \text{bool}, p_1 : \tau \text{ et } p_2 : \tau \\ (p_1 = p_2) : \tau &\Leftrightarrow \tau = \text{bool} \text{ et il existe } \tau' \in \{\text{unit}, \text{nat}, \text{bool}\} \text{ tel que } p_1 : \tau' \text{ et } p_2 : \tau' \\ p_1 + p_2 : \tau &\Leftrightarrow \tau = \text{nat}, p_1 : \text{nat} \text{ et } p_2 : \text{nat} \\ p_1 - p_2 : \tau &\Leftrightarrow \tau = \text{nat}, p_1 : \text{nat} \text{ et } p_2 : \text{nat} \\ p_1 \times p_2 : \tau &\Leftrightarrow \tau = \text{nat}, p_1 : \text{nat} \text{ et } p_2 : \text{nat} \\ p_0/2 : \tau &\Leftrightarrow \tau = \text{nat}, p_0 : \text{nat} \\ x : \tau &\Leftrightarrow t(x) = \tau \\ (\text{fun } x \rightarrow p_0) : \tau &\Leftrightarrow \text{il existe } \tau' \text{ tel que } p_0 : \tau' \text{ et } \tau = (t(x) \Rightarrow \tau') \\ (p_1 p_2) : \tau &\Leftrightarrow \text{il existe } \tau' \text{ tel que } p_1 : (\tau' \Rightarrow \tau) \text{ et } p_2 : \tau' \\ (\text{rec } p_0) : \tau &\Leftrightarrow p_0 : (\tau \Rightarrow \tau) \\ (p_1 \|_{\text{u}} p_2) : \tau &\Leftrightarrow \tau = \text{unit}, p_1 : \text{unit} \text{ et } p_2 : \text{unit} \end{aligned}$$

On pourra par exemple vérifier que  $(1 + (\text{fun } x \rightarrow x)) : \tau$  est faux quel que soit  $\tau$ . Ou encore, que  $(\text{fun } x \rightarrow (\text{fun } y \rightarrow (1 + x))) : (\text{nat} \Rightarrow t(y) \Rightarrow \text{nat})$  est vrai à condition que  $t(x) = \text{nat}$ .

**Question 2.1.** Soient  $x, y$  et  $z$  des variables quelconques, et  $\tau$  un type. Indiquer, sans justifier, sous quelles conditions sur  $\tau$ ,  $t(x)$ ,  $t(y)$  et  $t(z)$  on a  $(\text{fun } x \rightarrow \text{if } x = y \text{ then } z \text{ else } z) : \tau$ .

On considère désormais uniquement des programmes  $p$  bien typés, c'est à dire pour lesquels il existe  $\tau$  tel que  $p : \tau$ . Quand  $p$  est un programme bien typé, on note  $t(p)$  son unique type.

Pour donner un sens aux variables présentes dans un programme, on introduit la notion suivante : un **environnement** est une application  $\mathcal{E} : \mathcal{X} \rightarrow \bigcup_{\tau} [\tau]$  telle qu'on a  $\mathcal{E}(x) \in \llbracket t(x) \rrbracket$  pour tout  $x \in \mathcal{X}$ . Si  $\mathcal{E}$  est un environnement et  $e \in \llbracket t(x) \rrbracket$ ,  $\mathcal{E}[x \mapsto e]$  est l'environnement  $\mathcal{E}'$  tel que  $\mathcal{E}'(x) = e$  et  $\mathcal{E}'(y) = \mathcal{E}(y)$  pour tout  $y \neq x$ .

L'**interprétation** d'un programme  $p$  dans un environnement  $\mathcal{E}$ , notée  $\llbracket p \rrbracket^{\mathcal{E}}$ , est définie par les clauses de la figure 1, page 13, qu'il n'est pas nécessaire de consulter immédiatement. Dans ces clauses, on écrit simplement  $\llbracket p \rrbracket^{\mathcal{E}} = \perp$  pour  $\llbracket p \rrbracket^{\mathcal{E}} = \perp_{t(p)}$ . On admettra que, pour tout programme (bien typé)  $p$  et pour tout environnement  $\mathcal{E}$ , l'interprétation de  $p$  dans  $\mathcal{E}$  est bien définie et appartient à  $\llbracket t(p) \rrbracket$  :

$$\llbracket p \rrbracket^{\mathcal{E}} \in \llbracket t(p) \rrbracket$$

La figure 1 comporte de nombreux cas, dont certains sont assez techniques, et les intuitions (parfois trompeuses) issues de la pratique d'OCaml ne seront pas suffisantes pour en comprendre l'ensemble : le candidat est encouragé à ne pas s'y attarder trop, mais à en approfondir les différents cas au fil de l'énoncé. Pour cela, il est conseillé de séparer la page de la figure du reste de l'énoncé pour la garder à portée de main.

Intuitivement, l'interprétation d'un programme représente le résultat de son évaluation. Pour les types de base, une évaluation n'a jamais pour résultat  $\perp$ , mais cet élément spécial représente au contraire le résultat indéfini d'une évaluation qui ne termine pas.

Considérons par exemple le programme  $\text{fun } x \rightarrow x + y$ , de type  $\text{nat} \Rightarrow \text{nat}$  si  $x, y \in \mathcal{X}$  et  $t(x) = t(y) = \text{nat}$ . Par définition de l'interprétation,  $\llbracket \text{fun } x \rightarrow x + y \rrbracket^{\mathcal{E}}$  est l'application définie par  $\llbracket \text{fun } x \rightarrow x + y \rrbracket^{\mathcal{E}}(e) = \llbracket x + y \rrbracket^{\mathcal{E}[x \mapsto e]}$  pour tout  $e \in \llbracket \text{nat} \rrbracket$ . Si  $e = \perp_{\text{nat}}$  ou  $\mathcal{E}(y) = \perp_{\text{nat}}$ , on vérifie  $\llbracket \text{fun } x \rightarrow x + y \rrbracket^{\mathcal{E}}(e) = \perp_{\text{nat}}$ . Sinon,  $e \in \mathbb{N}$  et  $\mathcal{E}(y) \in \mathbb{N}$ , et l'on a  $\llbracket \text{fun } x \rightarrow x + y \rrbracket^{\mathcal{E}}(e) = e + \mathcal{E}(y)$ .

On remarque dans cet exemple que la valeur de  $\mathcal{E}$  sur les variables autres que  $y$  n'entre pas en jeu dans l'interprétation du programme. Cela correspond intuitivement au fait que seule la variable  $y$  fait référence à un objet extérieur au programme — on dit que cette variable est **libre**. À l'inverse, la variable  $x$  utilisée dans le sous-programme  $x + y$  est une référence interne au programme, correspondant à l'argument de la fonction définie — l'occurrence de  $x$  dans  $x + y$  est dite **liée** par la construction  $\text{fun } x \rightarrow \dots$  englobante.

On dit que l'interprétation d'un programme  $q$  est **indépendante** de l'environnement quand  $\llbracket q \rrbracket^{\mathcal{E}} = \llbracket q \rrbracket^{\mathcal{E}'}$  quels que soient  $\mathcal{E}$  et  $\mathcal{E}'$ . On notera simplement  $\llbracket q \rrbracket$  l'interprétation d'un tel programme dans un environnement arbitraire. Dans la suite du sujet, si on demande au candidat un programme  $q$  tel que  $\llbracket q \rrbracket$  satisfasse une certaine propriété, on attend un programme dont l'interprétation soit indépendante de l'environnement — ce qui est garanti si le programme est sans variable libre.

**Question 2.2.** Soit une variable  $x$  telle que  $t(x) = \text{nat}$ .

- a. On considère le programme  $p_e = (\text{fun } x \rightarrow ((2 \times (x/2)) = x))$ , de type  $\text{nat} \Rightarrow \text{bool}$ . Caractériser  $\{n \in \mathbb{N} \mid \llbracket p_e \rrbracket(n) = \text{tt}\}$  en justifiant par le calcul de  $\llbracket p_e \rrbracket$ .
- b. On définit l'application  $f_c : \mathbb{N} \rightarrow \mathbb{N}$  par  $f_c(n) = n/2$  si  $n$  est pair,  $f_c(n) = 3n + 1$  sinon. Donner un programme  $p_c : \text{nat} \Rightarrow \text{nat}$  tel que  $\llbracket p_c \rrbracket$  coïncide avec  $f_c$  sur  $\mathbb{N}$ . *On utilisera directement  $p_e$  plutôt que de recopier sa définition.*

On s'intéresse maintenant à l'interprétation des programmes de la forme  $\text{rec } p_0$ , permettant des définitions récursives de façon analogue au `let rec ...` d'OCaml. Un programme  $\text{rec } p_0$  est de type  $\tau$  quand  $p_0 : \tau \Rightarrow \tau$ . Puisque  $\llbracket p_0 \rrbracket^{\mathcal{E}}$  est une application continue de  $\llbracket \tau \rrbracket$  dans lui-même, la suite  $((\llbracket p_0 \rrbracket^{\mathcal{E}})^i(\perp_{\tau}))_{i \in \mathbb{N}}$  est croissante : on a en effet  $\perp_{\tau} \leq_{\tau} \llbracket p_0 \rrbracket^{\mathcal{E}}(\perp_{\tau})$  par minimalité de  $\perp_{\tau}$ , puis  $\llbracket p_0 \rrbracket^{\mathcal{E}}(\perp_{\tau}) \leq_{\tau} (\llbracket p_0 \rrbracket^{\mathcal{E}})^2(\perp_{\tau})$  par croissance de  $\llbracket p_0 \rrbracket^{\mathcal{E}}$ , et ainsi de suite. Toute suite croissante admettant une borne supérieure dans l'ordre partiel complet  $(\llbracket \tau \rrbracket, \leq_{\tau})$ , l'interprétation  $\llbracket \text{rec } p_0 \rrbracket^{\mathcal{E}}$  est bien définie.

**Question 2.3.** Soient  $f, n \in \mathcal{X}$  des variables telles que  $t(n) = \text{nat}$  et  $t(f) = (\text{nat} \Rightarrow \text{nat})$ . On pose :

$$p \stackrel{\text{def}}{=} \text{fun } f \rightarrow \text{fun } n \rightarrow \text{if } n = 0 \text{ then } 1 \text{ else } n \times f(n - 1)$$

On pourra vérifier que  $p$  est bien typé, de type  $(\text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat})$ .

- a. Soit  $f \in \llbracket \text{nat} \Rightarrow \text{nat} \rrbracket$  et  $e \in \llbracket \text{nat} \rrbracket$ . Exprimer  $\llbracket p \rrbracket(f)(e)$  en fonction de  $e$  et  $f$ .
- b. Calculer  $\llbracket p \rrbracket^{i+1}(\perp_{\text{nat} \Rightarrow \text{nat}})(e)$  pour tous  $i \in \mathbb{N}$  et  $e \in \llbracket \text{nat} \rrbracket$ .
- c. En déduire que  $\llbracket \text{rec } p \rrbracket(k) = k!$  pour tout  $k \in \mathbb{N}$ , et donner la valeur de  $\llbracket \text{rec } p \rrbracket(\perp_{\text{nat}})$ .

**Question 2.4.** On reprend la définition de  $f_c$  de la question 2.2. La conjecture de Collatz énonce que, pour tout  $n \in \mathbb{N}^*$ , il existe  $i \in \mathbb{N}$  tel que  $f_c^i(n) = 1$ . Autrement dit, la suite des itérées de  $f_c$  finirait toujours par atteindre 1, quelle que soit la valeur de départ non nulle.

- a. Donner un programme  $q : (\text{nat} \Rightarrow \text{unit}) \Rightarrow (\text{nat} \Rightarrow \text{unit})$  tel que, pour tous  $i \in \mathbb{N}$  et  $n \in \mathbb{N}^*$ ,  $\llbracket q \rrbracket^{i+1}(\perp_{\text{nat} \Rightarrow \text{unit}})(n) = \text{uu}$ ssi il existe  $k \leq i$  tel que  $f_c^k(n) = 1$ . *On utilisera directement le programme  $p_c$  de la question 2.2 sans en recopier la définition.*
- b. Donner un programme  $p : \text{nat} \Rightarrow \text{unit}$  tel que, pour tout  $n \in \mathbb{N}^*$ ,  $\llbracket p \rrbracket(n) = \text{uu}$ ssi il existe  $k \in \mathbb{N}$  tel que  $f_c^k(n) = 1$ . Justifier.

Soient  $\tau$  un type et  $x$  une variable de type  $\tau$ . On définit :

$$\text{DIV}_{\tau} \stackrel{\text{def}}{=} \text{rec } (\text{fun } x \rightarrow x)$$

On vérifie aisément que  $\text{DIV}_{\tau} : \tau$  et  $\llbracket \text{DIV}_{\tau} \rrbracket = \perp_{\tau}$ . Intuitivement,  $\text{DIV}_{\tau}$  est défini récursivement comme un objet  $x$  égal à lui même. L'évaluation de  $\text{DIV}_{\tau}$  déroule à l'infini cette définition, et ne termine donc pas.

Il existe donc des programmes  $p : \tau_1 \Rightarrow \tau_2$  tels que  $\llbracket p \rrbracket(e) = \perp_{\tau_2}$  pour tout  $e \in \llbracket \tau_1 \rrbracket$ . Si  $\tau_2$  est un type de base, l'évaluation de  $(p q)$  ne termine donc pas, quel que soit  $q : \tau_1$ . Réciproquement, il existe des programmes  $p : \tau_1 \Rightarrow \tau_2$  tels que  $(p q)$  termine toujours, même quand  $\llbracket q \rrbracket = \perp_{\tau_1}$ . Par exemple, si  $x$  est une variable de type  $\text{nat}$ , le programme  $(\text{fun } x \rightarrow 42) : \text{nat} \Rightarrow \text{nat}$  a pour interprétation l'application qui envoie tout élément de  $\llbracket \text{nat} \rrbracket$  sur l'entier 42. Intuitivement, ce programme peut terminer même si l'évaluation de son argument ne termine pas, puisque cet argument n'est pas utilisé. Par contre, on pourra vérifier que  $\text{fun } x \rightarrow \text{if } x = x \text{ then } 42 \text{ else } 13$  a pour interprétation une application  $f$  telle que  $f(\perp_{\text{nat}}) = \perp_{\text{nat}}$  — on notera aussi que cette application ne prend jamais la valeur 13, car  $\llbracket x = x \rrbracket^{\mathcal{E}} \in \{\text{tt}, \perp_{\text{bool}}\}$  pour tout  $\mathcal{E}$ .

Soit  $\tau$  un type. On dit qu'un élément  $e \in \llbracket \tau \rrbracket$  est **calculable** quand il existe un programme  $p : \tau$  tel que  $\llbracket p \rrbracket = e$ . Par extension, on dira qu'une application  $f : \llbracket \tau_1 \rrbracket \rightarrow \llbracket \tau_2 \rrbracket$  est calculable quand il existe  $p : \tau_1 \Rightarrow \tau_2$  tel que  $\llbracket p \rrbracket = f$ .

**Question 2.5.**

- Montrer que tous les éléments de  $\llbracket \text{unit} \Rightarrow \text{unit} \rrbracket$  sont calculables.
- On admet que l'ensemble des programmes est dénombrable. Montrer qu'il existe des éléments non calculables dans  $\llbracket \text{nat} \Rightarrow \text{unit} \rrbracket$ .

Dans la suite du sujet, nous chercherons à reconnaître certains éléments de  $\llbracket \tau \rrbracket$  au moyen de programmes de type  $\tau \Rightarrow \text{unit}$  : ces programmes devront renvoyer  $\text{uu}$  quand un élément est reconnu, et ne pas terminer sinon. Dans ce cadre, des notions de conjonction et de disjonction sur le type `unit` seront utiles.

**Question 2.6.** Donner un programme `unit_and` : `unit`  $\Rightarrow$  `unit`  $\Rightarrow$  `unit` tel que, pour tout environnement  $\mathcal{E}$  et pour tous  $p : \text{unit}$  et  $q : \text{unit}$ ,

$$\llbracket \text{unit\_and } p \ q \rrbracket^{\mathcal{E}} = \text{uu} \quad \text{ssi} \quad \llbracket p \rrbracket^{\mathcal{E}} = \llbracket q \rrbracket^{\mathcal{E}} = \text{uu}.$$

Par la suite, on écrira simplement  $p \&_u q$  au lieu de `unit_and`  $p$   $q$ .

On considère enfin l'opérateur  $(\_\parallel_u \_)$  de disjonction sur `unit`. La définition de  $\llbracket p_1 \parallel_u p_2 \rrbracket^{\mathcal{E}}$  en figure 1 indique que ce programme peut terminer (renvoyer  $\text{uu}$ ) dès que l'un de ses deux sous-programmes termine. Intuitivement, on peut penser que les deux sous-programmes sont évalués en parallèle, et que  $\text{uu}$  est renvoyé dès que l'une de ces deux évaluations termine.

**Question 2.7.**

- Soit  $x$  une variable telle que  $t(x) = (\text{bool} \Rightarrow \text{unit})$ . On pose :

$$p \stackrel{\text{def}}{=} \text{fun } x \rightarrow ((x \text{ tt}) \parallel_u (x \text{ ff}))$$

Caractériser  $\{f \in \llbracket \text{bool} \Rightarrow \text{unit} \rrbracket \mid \llbracket p \rrbracket(f) = \text{uu}\}$ .

- Soient  $x$ ,  $p'$ , et  $n$  des variables telles que  $t(x) = t(p') = (\text{nat} \Rightarrow \text{unit})$  et  $t(n) = \text{nat}$ . On pose :

$$p \stackrel{\text{def}}{=} \text{fun } x \rightarrow ((\text{rec } (\text{fun } p' \rightarrow \text{fun } n \rightarrow x \ n \parallel_u p' \ (n + 1))) \ 0)$$

Caractériser, sans justifier,  $\{f \in \llbracket \text{nat} \Rightarrow \text{unit} \rrbracket \mid \llbracket p \rrbracket(f) = \text{uu}\}$ .

## Partie III

Il existe de nombreux liens entre topologie et sémantique des langages de programmation. Nous nous proposons ici de dériver des notions topologiques à partir de notre langage de programmation.

On dit que l'application  $f : \llbracket \tau \rrbracket \rightarrow \llbracket \text{unit} \rrbracket$  est la **fonction caractéristique** de  $U \subseteq \llbracket \tau \rrbracket$  quand, pour tout  $e \in \llbracket \tau \rrbracket$ ,  $f(e) = \text{uu}$  ssi  $e \in U$ . Un ensemble  $U \subseteq \llbracket \tau \rrbracket$  est un **ouvert calculatoire** de  $\llbracket \tau \rrbracket$  quand la fonction caractéristique de  $U$  est calculable. Un ensemble  $U \subseteq \llbracket \tau \rrbracket$  est un **fermé calculatoire** de  $\llbracket \tau \rrbracket$  quand son complémentaire dans  $\llbracket \tau \rrbracket$  est un ouvert calculatoire de  $\llbracket \tau \rrbracket$ .

Par exemple,  $\llbracket \text{nat} \rrbracket \setminus \{\perp_{\text{nat}}, 13\}$  est un ouvert calculatoire de  $\llbracket \text{nat} \rrbracket$ . En effet, sa fonction caractéristique est l'interprétation du programme `fun x → if x = 13 then DIVunit else uu`.

**Question 3.1.** Soit  $\tau$  un type. Montrer que  $\llbracket \tau \rrbracket$  est à la fois un ouvert calculatoire et un fermé calculatoire de  $\llbracket \tau \rrbracket$ .

**Question 3.2.** Quels sont les ouverts calculatoires de  $\llbracket \text{unit} \rrbracket$  ?

**Question 3.3.** Soient  $\tau$  et  $\tau'$  des types,  $U$  un ouvert calculatoire de  $\llbracket \tau' \rrbracket$ , et  $f : \llbracket \tau \rrbracket \rightarrow \llbracket \tau' \rrbracket$  une application calculable. Montrer que la pré-image  $f^{-1}(U)$  de  $U$  par  $f$  est un ouvert calculatoire de  $\llbracket \tau \rrbracket$ .

**Question 3.4.** On considère un type  $\tau$  quelconque. Utiliser les opérateurs  $(\_||_{\text{u}}\_)$  et  $(\_&_{\text{u}}\_)$  pour montrer les propriétés suivantes.

- a. L'union de deux ouverts calculatoires de  $\llbracket \tau \rrbracket$  est encore un ouvert calculatoire de  $\llbracket \tau \rrbracket$ .
- b. L'intersection de deux ouverts calculatoires de  $\llbracket \tau \rrbracket$  est encore un ouvert calculatoire de  $\llbracket \tau \rrbracket$ .

Pour la prochaine question on pourra utiliser le résultat suivant sans le justifier :

**Propriété A.** Soient  $\tau_1$  et  $\tau_2$  des types quelconques. On pose  $\tau = (\tau_1 \Rightarrow \tau_2 \Rightarrow \text{unit})$ . Pour tous  $p : \tau \Rightarrow \tau$ ,  $e_1 \in \llbracket \tau_1 \rrbracket$  et  $e_2 \in \llbracket \tau_2 \rrbracket$ , on a

$$\llbracket \text{rec } p \rrbracket(e_1)(e_2) = \text{uu} \quad \text{ssi} \quad \exists i \in \mathbb{N}. (\llbracket p \rrbracket^i(\perp_\tau))(e_1)(e_2) = \text{uu}.$$

**Question 3.5.** On souhaite montrer qu'une union infinie d'ouverts calculatoires est encore un ouvert calculatoire, dans le sens suivant. Soient  $(U_i)_{i \in \mathbb{N}}$  une suite de parties de  $\llbracket \tau \rrbracket$  et  $p : \text{nat} \Rightarrow \tau \Rightarrow \text{unit}$  un programme tels que, pour tout  $i \in \mathbb{N}$ ,  $\llbracket p \rrbracket^i$  est la fonction caractéristique de  $U_i$ . Montrer que  $\bigcup_{i \in \mathbb{N}} U_i$  est un ouvert calculatoire de  $\llbracket \tau \rrbracket$ . *Indication : on pourra construire un programme intermédiaire `rec q : nat ⇒ τ ⇒ unit` dont on montrera qu'il satisfait, pour tous  $k \in \mathbb{N}$  et  $e \in \llbracket \tau \rrbracket$ ,  $\llbracket \text{rec } q \rrbracket(k)(e) = \text{uu}$  ssi  $e \in \bigcup_{i \geq k} U_i$ .*

Un sous-ensemble  $U \subseteq \llbracket \tau \rrbracket$  est **calculatoirement compact** dans  $\llbracket \tau \rrbracket$  s'il existe un programme  $\forall_U : (\tau \Rightarrow \text{unit}) \Rightarrow \text{unit}$  tel que, pour tout programme  $p : \tau \Rightarrow \text{unit}$ ,

$$\llbracket \forall_U p \rrbracket = \text{uu} \quad \text{ssi} \quad \llbracket p \rrbracket(e) = \text{uu} \text{ pour tout } e \in U.$$

Par exemple, l'ensemble  $E = \{40, 12\}$  est calculatoirement compact dans  $\llbracket \text{nat} \rrbracket$ , comme en témoigne le programme  $\forall_E = \text{fun } f \rightarrow (f \ 40) \ \&_{\text{u}} (f \ 12)$ , où  $f$  est une variable quelconque de type  $\text{nat} \Rightarrow \text{unit}$ .

**Question 3.6.** Soit  $\tau$  un type.

- a. Montrer que  $\emptyset$  est calculatoirement compact dans  $\llbracket \tau \rrbracket$ .
- b. Soit  $E \subseteq \llbracket \tau \rrbracket$  tel que  $\perp_\tau \in E$ . Montrer que  $E$  est calculatoirement compact dans  $\llbracket \tau \rrbracket$ .
- c. En déduire que tout fermé calculatoire de  $\llbracket \tau \rrbracket$  est calculatoirement compact dans  $\llbracket \tau \rrbracket$ .

**Question 3.7.** Soit  $U \subseteq \mathbb{N}$ . Montrer que  $U$  est calculatoirement compact dans  $\llbracket \text{nat} \rrbracket$  ssi  $U$  est fini. *Indication : on pourra écrire un certain programme  $p : \text{nat} \Rightarrow \text{unit}$  comme la borne supérieure d'une suite croissante.*

## Partie IV

Étant donnée une suite de booléens  $s = (s_i)_{i \in \mathbb{N}} \in \mathbb{B}^{\mathbb{N}}$  on définit  $f_s \in [\![\text{nat} \Rightarrow \text{bool}]\!]$  par  $f_s(\perp_{\text{nat}}) = \perp_{\text{bool}}$  et  $f_s(i) = s_i$  pour tout  $i \in \mathbb{N}$ . On définit l'espace de Cantor  $\mathbb{C}$  comme l'ensemble de ces applications :

$$\mathbb{C} \stackrel{\text{def}}{=} \{f_s \mid s \in \mathbb{B}^{\mathbb{N}}\}$$

L'objectif de cette partie est de montrer que  $\mathbb{C}$  est calculatoirement compact dans  $[\![\text{nat} \Rightarrow \text{bool}]\!]$ . On remarquera que  $\mathbb{C}$  contient des éléments non calculables. Réciproquement, il existe des éléments calculables de  $[\![\text{nat} \Rightarrow \text{bool}]\!]$  qui ne sont pas dans  $\mathbb{C}$ .

Dans la suite de l'énoncé, les programmes de type  $\text{nat} \Rightarrow \text{bool}$  seront représentés par la lettre  $q$ . Les programmes de type  $(\text{nat} \Rightarrow \text{bool}) \Rightarrow \text{unit}$  seront représentés par la lettre  $p$ .

Pour  $b \in \mathbb{B}$  et  $s \in \mathbb{B}^{\mathbb{N}}$  on note  $b::s$  la suite  $s' \in \mathbb{B}^{\mathbb{N}}$  telle que  $s'_0 = b$  et  $s'_{n+1} = s_n$  pour tout  $n \in \mathbb{N}$ . Pour tout programme  $q : \text{nat} \Rightarrow \text{bool}$  on définit l'opération analogue avec la même notation, où  $x$  est une variable de type  $\text{nat}$  :

$$b::q \stackrel{\text{def}}{=} \text{fun } x \rightarrow \text{if } x = 0 \text{ then } b \text{ else } q(x - 1)$$

Enfin, pour  $p : (\text{nat} \Rightarrow \text{bool}) \Rightarrow \text{unit}$  on définit, en prenant une variable  $y$  de type  $\text{nat} \Rightarrow \text{bool}$  :

$$p/b \stackrel{\text{def}}{=} \text{fun } y \rightarrow p(b::y)$$

Pour  $s \in \mathbb{B}^{\mathbb{N}}$  et  $k \in \mathbb{N}$  on note  $f_{s < k}$  l'application  $f' \in [\![\text{nat} \Rightarrow \text{bool}]\!]$  telle que  $f'(i) = s_i$  pour tout  $i$  tel que  $0 \leq i < k$ , et  $f'(e) = \perp_{\text{bool}}$  sinon.

**Question 4.1.** Soient un programme  $p : (\text{nat} \Rightarrow \text{bool}) \Rightarrow \text{unit}$  et  $s \in \mathbb{B}^{\mathbb{N}}$  tels que  $[\![p]\!](f_s) = \text{uu}$ . Montrer qu'il existe un entier  $k$  tel que  $[\![p]\!](f_{s < k}) = \text{uu}$ . Dans la suite du sujet, on notera  $k(s, p)$  le plus petit entier satisfaisant cette propriété.

**Question 4.2.** Soient un programme  $p : (\text{nat} \Rightarrow \text{bool}) \Rightarrow \text{unit}$ ,  $b \in \mathbb{B}$  et  $s \in \mathbb{B}^{\mathbb{N}}$  tels que  $[\![p]\!](f_{b::s}) = \text{uu}$ . Montrer, pour tout  $k \in \mathbb{N}$ , que  $[\![p/b]\!](f_{s < k}) = [\![p]\!](f_{b::s < 1+k})$ .

**Question 4.3.** Soit  $p : (\text{nat} \Rightarrow \text{bool}) \Rightarrow \text{unit}$  tel que, pour tout  $f \in \mathbb{C}$ ,  $[\![p]\!](f) = \text{uu}$ . On définit :

$$K(p) \stackrel{\text{def}}{=} \{k(s, p) \mid s \in \mathbb{B}^{\mathbb{N}}\}$$

- a. Soit  $s \in \mathbb{B}^{\mathbb{N}}$ . On pose  $i = k(s, p)$ . Montrer que

$$K(((p/s_0)/s_1) \dots /s_{i-1}) = \{0\}.$$

- b. Montrer que, si  $K(p)$  est non borné, alors il existe  $b \in \mathbb{B}$  tel que  $K(p/b)$  est encore non borné.

- c. En déduire que  $K(p)$  est borné.

La borne supérieure de  $K(p)$  sera notée  $|p|$  et appelée **module d'uniforme continuité** de  $p$ .

**Question 4.4.** Soit  $p : (\text{nat} \Rightarrow \text{bool}) \Rightarrow \text{unit}$  tel que, pour tout  $f \in \mathbb{C}$ ,  $\llbracket p \rrbracket(f) = \text{uu}$ .

- a. Soient  $b \in \mathbb{B}$  et  $s \in \mathbb{B}^{\mathbb{N}}$  tels que  $k(b::s, p) > 0$ . Montrer que  $k(s, p/b) < k(b::s, p)$ .
- b. Montrer que, si  $|p| > 0$ , on a  $|p/b| < |p|$  pour tout  $b \in \mathbb{B}$ .

**Question 4.5.** Montrer que  $\mathbb{C}$  est calculatoirement compact dans  $\llbracket \text{nat} \Rightarrow \text{bool} \rrbracket$ .



$$\begin{aligned}
 \llbracket c \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} c \quad \text{pour tout } c \in \mathbb{N} \cup \mathbb{B} \cup \{\text{uu}\} \\
 \llbracket x \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} \mathcal{E}(x) \\
 \llbracket p_1 = p_2 \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} \begin{cases} \perp_{\text{bool}} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} = \perp \text{ ou } \llbracket p_2 \rrbracket^{\mathcal{E}} = \perp \\ \text{tt} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} \neq \perp, \llbracket p_2 \rrbracket^{\mathcal{E}} \neq \perp \text{ et } \llbracket p_1 \rrbracket^{\mathcal{E}} = \llbracket p_2 \rrbracket^{\mathcal{E}} \\ \text{ff} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} \neq \perp, \llbracket p_2 \rrbracket^{\mathcal{E}} \neq \perp \text{ et } \llbracket p_1 \rrbracket^{\mathcal{E}} \neq \llbracket p_2 \rrbracket^{\mathcal{E}} \end{cases} \\
 \llbracket \text{if } p_0 \text{ then } p_1 \text{ else } p_2 \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} \begin{cases} \llbracket p_1 \rrbracket^{\mathcal{E}} & \text{si } \llbracket p_0 \rrbracket^{\mathcal{E}} = \text{tt} \\ \llbracket p_2 \rrbracket^{\mathcal{E}} & \text{si } \llbracket p_0 \rrbracket^{\mathcal{E}} = \text{ff} \\ \perp_{\tau} & \text{si } \llbracket p_0 \rrbracket^{\mathcal{E}} = \perp, \text{ où } \tau = t(p_1) = t(p_2) \end{cases} \\
 \llbracket p_1 + p_2 \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} \begin{cases} \perp_{\text{nat}} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} = \perp \text{ ou } \llbracket p_2 \rrbracket^{\mathcal{E}} = \perp \\ \llbracket p_1 \rrbracket^{\mathcal{E}} + \llbracket p_2 \rrbracket^{\mathcal{E}} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} \in \mathbb{N} \text{ et } \llbracket p_2 \rrbracket^{\mathcal{E}} \in \mathbb{N} \end{cases} \\
 \llbracket p_1 \times p_2 \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} \begin{cases} \perp_{\text{nat}} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} = \perp \text{ ou } \llbracket p_2 \rrbracket^{\mathcal{E}} = \perp \\ \llbracket p_1 \rrbracket^{\mathcal{E}} \times \llbracket p_2 \rrbracket^{\mathcal{E}} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} \in \mathbb{N} \text{ et } \llbracket p_2 \rrbracket^{\mathcal{E}} \in \mathbb{N} \end{cases} \\
 \llbracket p_1 - p_2 \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} \begin{cases} \perp_{\text{nat}} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} = \perp \text{ ou } \llbracket p_2 \rrbracket^{\mathcal{E}} = \perp \\ \max(0, \llbracket p_1 \rrbracket^{\mathcal{E}} - \llbracket p_2 \rrbracket^{\mathcal{E}}) & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} \in \mathbb{N} \text{ et } \llbracket p_2 \rrbracket^{\mathcal{E}} \in \mathbb{N} \end{cases} \\
 \llbracket p_0 / 2 \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} \begin{cases} \perp_{\text{nat}} & \text{si } \llbracket p_0 \rrbracket^{\mathcal{E}} = \perp \\ \lfloor \llbracket p_0 \rrbracket^{\mathcal{E}} / 2 \rfloor & \text{si } \llbracket p_0 \rrbracket^{\mathcal{E}} \in \mathbb{N} \end{cases} \\
 \llbracket p_1 \ p_2 \rrbracket^{\mathcal{E}} &\stackrel{\text{def}}{=} \llbracket p_1 \rrbracket^{\mathcal{E}}(\llbracket p_2 \rrbracket^{\mathcal{E}})
 \end{aligned}$$

$\llbracket \text{fun } x \rightarrow p_0 \rrbracket^{\mathcal{E}}$  est l'application  $f : \llbracket t(x) \rrbracket \rightarrow \llbracket t(p_0) \rrbracket$  telle que  $f(e) = \llbracket p_0 \rrbracket^{\mathcal{E}[x \mapsto e]}$  pour tout  $e \in \llbracket t(x) \rrbracket$

$$\llbracket \text{rec } p_0 \rrbracket^{\mathcal{E}} \stackrel{\text{def}}{=} \sup_{i \in \mathbb{N}} \llbracket \tau \rrbracket^{\mathcal{E}}((\llbracket p_0 \rrbracket^{\mathcal{E}})^i(\perp_{\tau})) \quad \text{où } \tau = t(\text{rec } p_0)$$

$$\llbracket p_1 \|_{\text{u}} p_2 \rrbracket^{\mathcal{E}} \stackrel{\text{def}}{=} \begin{cases} \perp_{\text{unit}} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} = \perp \text{ et } \llbracket p_2 \rrbracket^{\mathcal{E}} = \perp \\ \text{uu} & \text{si } \llbracket p_1 \rrbracket^{\mathcal{E}} = \text{uu} \text{ ou } \llbracket p_2 \rrbracket^{\mathcal{E}} = \text{uu} \end{cases}$$

FIGURE 1 – Définition de l'interprétation des programmes.

Les techniques de compression sans perte permettent d'encoder des données telles que des textes afin de les stocker en occupant un minimum d'espace. Elles s'appuient sur une estimation de la fréquence d'apparition des symboles qui composent le texte. Dans ce problème, nous étudions comment compter efficacement le nombre d'occurrences de chaque symbole dans un texte, que l'on entendra dans la suite comme une suite finie de numéros (par exemple, des numéros Unicode), et comment optimiser le fonctionnement d'un encodeur pour passer d'un texte à une suite de bits.

## Préliminaires

L'épreuve est composée d'un problème unique, comportant 33 questions. Après cette section de préliminaires, le problème est divisé en deux parties indépendantes. Pour répondre à une question, un candidat pourra réutiliser le résultat d'une question antérieure, même s'il n'est pas parvenu à établir ce résultat.

### Concernant la programmation

Il faudra coder des fonctions à l'aide du langage de programmation OCaml, tout autre langage étant exclu. Lorsque le candidat écrira une fonction, il pourra faire appel à d'autres fonctions définies dans les questions précédentes de la même sous-partie ; il pourra aussi définir des fonctions auxiliaires. Quand l'énoncé demande de coder une fonction, il n'est pas nécessaire de justifier que celle-ci est correcte, sauf si l'énoncé le demande explicitement. Si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile de tester si les hypothèses sont bien vérifiées dans le code de la fonction.

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple *n*) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple n).

On suppose codées les fonctions suivantes :

- `list_length`, de type `'a list -> int`, de complexité en temps linéaire, qui renvoie la longueur d'une liste,
- `list_append`, de type `'a list -> 'a list -> 'a list`, de complexité en temps linéaire en la longueur du premier argument, qui concatène deux listes,
- `array_make`, de type `int -> 'a -> 'a array`, de complexité en temps linéaire en la valeur du premier argument, qui crée un tableau d'une longueur spécifiée en premier argument et dont chacune des cases est initialisée à une valeur donnée en second argument,
- `array_length`, de type `'a array -> int`, de complexité en temps constante, qui renvoie la longueur d'un tableau.

On rappelle que `a mod b` renvoie le reste de la division euclidienne de *a* par *b*.

Dans les calculs de complexité en espace, on considérera qu'un entier occupe une place constante en mémoire quelle que soit sa valeur.

## Concernant les objets manipulés et leur type

L'intervalle des entiers compris entre  $a$  et  $b$  est noté  $\llbracket a, b \rrbracket$ .

La lettre  $\Sigma$  désigne un *alphabet fini ordonné* de cardinal  $\lambda$ ; ses lettres, numérotées dans l'ordre croissant, sont les éléments

$$\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_{\lambda-2}, \sigma_{\lambda-1}.$$

Par exemple, la norme *Unicode* est une norme fréquemment utilisée en informatique qui consiste à travailler sur un alphabet de 1 114 112 symboles. Elle permet d'identifier toute sorte de caractères (lettre de l'alphabet latin, idéogramme chinois, émoticône, symbole de l'alphabet phonétique international, etc.) par un numéro unique entre 0 et 1 114 111 quelle que soit la plate-forme informatique employée.

**Définitions :** Un mot sur un alphabet  $\Sigma$  est une suite finie de lettres de  $\Sigma$ . Leur ensemble est noté  $\Sigma^*$ , le mot vide est noté  $\varepsilon$ . Un mot possède plusieurs caractéristiques : la *longueur* d'un mot  $w \in \Sigma^*$ , notée  $|w|$ , est le nombre de lettres qui composent  $w$  ; la *fréquence d'une lettre*  $\sigma \in \Sigma$  dans un mot  $w \in \Sigma^*$ , notée  $|w|_\sigma$ , est le nombre d'occurrences de  $\sigma$  dans  $w$  ; la *valence* d'un mot  $w \in \Sigma^*$ , notée  $[w]$ , est le nombre de symboles distincts qui composent  $w$ .

Un *texte* est une suite finie d'entiers de l'intervalle  $\llbracket 0, \lambda - 1 \rrbracket$ . Le mot associé au texte de  $s$  entiers  $[u_1, u_2, u_3, \dots, u_s]$  est le mot  $\sigma_{u_1}\sigma_{u_2}\dots\sigma_{u_s} \in \Sigma^*$ .

**Indication OCaml :** On définit les types `unicode`, `texte` et la constante globale `lambda` par les déclarations suivantes :

```
type unicode = int;;
type texte = unicode list;;
let lambda = 1114112;;
```

Une fonction de l'alphabet  $\Sigma$  vers un ensemble  $X$  est représentée par un tableau de longueur  $\lambda$  de type `'a array` où `'a` est le type par lequel on représente les éléments de  $X$ .

## 1 Fonction de distribution des lettres d'un texte

Le but de cette partie est de produire une structure de données qui permette de vérifier la présence d'une lettre dans un mot et de stocker les valeurs non nulles de la *fonction de distribution* des fréquences des lettres dans un mot  $w \in \Sigma^*$  définie comme suit

$$\begin{cases} \Sigma & \rightarrow \mathbb{N} \\ \sigma & \mapsto |w|_\sigma \end{cases}$$

de trois manières différentes et de comparer les complexités en temps et en espace de chaque approche.

## 1.1 Avec un tableau exhaustif et une liste

- 1 – On définit une fonction `fonction1` par le code suivant

```
let rec fonction1 (t:texte) =
  match t with
  | [] -> array_make lambda 0
  | u::tprime -> let theta = fonction1 tprime in
    theta.(u)<-theta.(u)+1;
    theta;;
```

Inférer le type de la fonction `fonction1`. Expliquer ce que calcule `fonction1 t` et le démontrer par récurrence sur  $|t|$ .

**Définition :** Soit  $\Sigma'$  un sous-alphabet de l'alphabet  $\Sigma$ . On appelle *répartition* du sous-alphabet  $\Sigma'$  dans le mot  $w \in \Sigma^*$  toute liste constituée de tous les couples  $(\sigma, |w|_\sigma)$  tels que  $\sigma \in \Sigma'$  et  $|w|_\sigma > 0$ . Cette liste peut être dans un ordre quelconque. La répartition d'un sous-alphabet dans un texte est la répartition de ce sous-alphabet dans le mot associé au texte.

Par exemple, une répartition des lettres du mot *acad* est la liste  $[(a, 2), (c, 1), (d, 1)]$ .

**Indication OCaml :** En OCaml, on pourra se servir du type

```
type repartition = (unicode * int) list;;
```

- 2 – Écrire une fonction `cree_repartition`, de type `texte -> repartition` qui renvoie une répartition d'un texte en utilisant la fonction `fonction1` définie à la question 1.

- 3 – Déterminer la complexité en temps de la fonction `cree_repartition` en fonction du cardinal  $\lambda$  de l'alphabet et d'une caractéristique du texte donné en entrée.

- 4 – Déterminer la complexité en espace de la fonction `cree_repartition` en fonction du cardinal  $\lambda$  de l'alphabet.

- 5 – Donner sans justification un ordre de grandeur de la taille de la valeur de retour de `cree_repartition` en fonction d'une caractéristique du texte donné en entrée.

- 6 – Comparer les ordres de grandeur obtenus dans les trois questions précédentes quand le texte donné en entrée est le texte d'un courriel de la vie courante rédigé en langue française et  $\Sigma$  est l'alphabet Unicode.

## 1.2 Avec une table modulaire

**Définition :** Soient  $w \in \Sigma^*$  un mot et  $m \in \mathbb{N}^*$  un entier non nul. Pour tout entier  $\ell$  compris entre 0 et  $m - 1$ , on note  $\Sigma^{[\ell]}$  le sous-alphabet de  $\Sigma$  défini par

$$\Sigma^{[\ell]} = \{\sigma_u, u \in \llbracket 0, \lambda - 1 \rrbracket; (u - \ell) \text{ est divisible par } m\}.$$

Une *table modulaire de comptage d'ordre  $m$  du mot  $w$*  est un tableau de longueur  $m$  dont la  $\ell$ -ième case contient une répartition des lettres de  $\Sigma^{[\ell]}$  dans  $w$ .

Par exemple, avec l'alphabet  $\Sigma = \{a, b, c, d, e\}$  muni de l'ordre alphabétique et l'entier  $m = 3$ , une table modulaire de comptage d'ordre 3 du mot *acad* est le tableau de trois listes

$$\boxed{[(a, 2); (d, 1)] \mid [] \mid [(c, 1)]}.$$

□ 7 – Écrire une fonction `incremente_repartition`, de type `repartition -> unicode -> repartition` telle que `incremente_repartition r u` renvoie la répartition d'un mot qui contient la lettre  $\sigma_u$  une fois de plus qu'un mot de répartition  $r$ .

□ 8 – Écrire une fonction `cree_modulaire`, dont le type est `texte -> int -> repartition array`, qui utilise la fonction définie à la question 7, telle que la valeur de retour de `cree_modulaire t m` est une table modulaire de comptage d'ordre  $m$  du texte  $t$ .

Le recours à la fonction `fonction1` de la question 1 n'est pas autorisé pour répondre à cette question.

□ 9 – Écrire une fonction `valence`, de type `repartition array -> int`, qui renvoie la valence d'un mot à partir de sa table modulaire de comptage.

□ 10 – Soient  $m$  et  $v$  deux entiers non nuls et  $X_1, X_2, \dots, X_v$  des variables aléatoires discrètes à valeurs dans l'intervalle entier  $\llbracket 0, m - 1 \rrbracket$ , mutuellement indépendantes et qui suivent la loi de distribution uniforme. Soient encore un entier  $i_0$  fixé dans l'intervalle  $\llbracket 1, v \rrbracket$  et un entier  $\ell_0$  fixé dans l'intervalle  $\llbracket 0, m - 1 \rrbracket$ . Pour tout entier  $\ell$  compris entre 0 et  $m - 1$ , on appelle  $Z_\ell$  le cardinal

$$Z_\ell = |\{i \in \llbracket 1, v \rrbracket; X_i = \ell\}|.$$

Pour tout entier  $\ell$  compris entre 0 et  $m - 1$ , calculer l'espérance

$$\mathbb{E}[Z_\ell | X_{i_0} = \ell_0].$$

□ 11 – Soit  $w \in \Sigma^*$  le mot d'un texte  $t$  de valence  $v$ . En supposant que l'ensemble des  $v$  lettres distinctes présentes dans le mot  $w$  a été choisi uniformément dans l'ensemble des parties à  $v$  éléments de l'alphabet  $\Sigma$ , montrer que la complexité moyenne en temps de la fonction `cree_modulaire t m` est

$$O\left(m + |t| + \frac{(v - 1)|t|}{m}\right).$$

12 – Donner sans justification la complexité en espace de la fonction `cree_modulaire t m` en fonction de  $m$  et d'une caractéristique du texte  $t$ .

13 – Quelle valeur numérique de  $m$  suggérez-vous de choisir lorsque  $t$  est le texte d'un courriel de la vie courante rédigé en langue française et  $\Sigma$  est l'alphabet Unicode ?

### 1.3 Avec un tableau creux

**Définition :** Un *tableau creux de comptage du mot*  $w \in \Sigma^*$  est un quadruplet  $(v, F, I, A)$  formé d'un entier  $v$  et de trois fonctions  $F : \Sigma \rightarrow \mathbb{N}$ ,  $I : \Sigma \rightarrow \Sigma$  et  $A : \Sigma \rightarrow \Sigma$  tels que

- (i) l'entier  $v$  est la valence du mot  $w$ ,
- (ii) pour toute lettre  $\sigma \in \Sigma$  présente dans le mot  $w$ , il existe une lettre  $\tau \in \Sigma$  telle que  $\tau \leq \sigma_{v-1}$  et  $I(\tau) = \sigma$ .
- (iii) pour toute lettre  $\tau \in \Sigma$  telle que  $\tau \leq \sigma_{v-1}$ , on a  $A(I(\tau)) = \tau$ ,
- (iv) pour toute lettre  $\sigma \in \Sigma$  présente dans le mot  $w$ , on a  $F(\sigma) = |w|_\sigma$ .

Par exemple, avec l'alphabet  $\Sigma = \{a, b, c, d, e, f, g, h\}$ , le mot *bbbbbbbbbbehhhhhhh* admet comme tableau de comptage le quadruplet  $(v, F, I, A)$

|   | a | b  | c | d | e | f | g | h |
|---|---|----|---|---|---|---|---|---|
| F | * | 12 | * | * | 1 | * | * | 9 |
| I | b | e  | h | * | * | * | * | * |
| A | * | a  | * | * | b | * | * | c |

où chaque symbole \* représente une valeur particulière mais non significative.

**Indication OCaml :** En OCaml, on pourra se servir du type

```
type creux = int * int array * unicode array * unicode array;;
```

On pourra utiliser une fonction `make_creux`, de type `int -> creux` qui crée et renvoie un tableau creux de comptage du mot vide en temps constant, aucune hypothèse ne pouvant être faite sur le contenu des trois tableaux constituant la valeur de retour de cette fonction.

14 – Soit  $(v, F, I, A)$  un tableau creux de comptage du mot  $w \in \Sigma^*$ . Montrer que pour toute lettre  $\tau \in \Sigma$  avec  $\tau \leq \sigma_{v-1}$ , la lettre  $I(\tau)$  est une lettre présente dans le mot  $w$ .

15 – Soit  $(v, F, I, A)$  un tableau creux de comptage du mot  $w \in \Sigma^*$ . Montrer que pour toute lettre  $\sigma \in \Sigma$ , la lettre  $\sigma$  est présente dans le mot  $w$  si et seulement si on a  $A(\sigma) \leq \sigma_{v-1}$  et  $I(A(\sigma)) = \sigma$ .

- 16 – Écrire une fonction `est_present` de type `creux -> unicode -> bool` telle que la valeur de retour de `tableau_creux theta u` est vraie si et seulement si la lettre  $\sigma_u$  est une lettre présente dans un mot de tableau creux de comptage  $\theta$ . Justifier la correction et la terminaison de la fonction.
- 17 – Écrire une fonction `incremente_tableaucréux`, de type `creux -> unicode -> creux`, telle que la valeur de retour de `incremente_tableaucréux theta u` est un tableau creux de comptage d'un mot qui contient la lettre  $\sigma_u$  une fois de plus qu'un mot de tableau creux de comptage  $\theta$ .
- 18 – Écrire une fonction `cree_tableaucréux` de type `texte -> creux` qui renvoie le tableau creux de comptage du texte donné en argument.  
Le recours à la fonction `fonction1` de la question 1 n'est pas autorisé pour répondre à cette question.
- 19 – Déterminer la complexité en temps de la fonction `cree_tableaucréux`.
- 20 – Donner sans justification la complexité en espace de la fonction `cree_tableaucréux`.
- 21 – Est-il réaliste d'utiliser cette fonction pour le texte d'un courriel de la vie courante rédigé en langue française encodé avec l'alphabet Unicode lorsqu'on utilise un ordinateur ou un téléphone actuel ?

## 2 Un encodeur optimal

### 2.1 Codes et codages

**Définitions :** Un *code*  $c$  est une application

$$c : \Sigma \rightarrow \{0, 1\}^*$$

Le *codage* associé à un code  $c$  est l'application

$$e : \begin{cases} \Sigma^* & \rightarrow \{0, 1\}^* \\ \tau_1 \tau_2 \dots \tau_n & \mapsto c(\tau_1)c(\tau_2)\dots c(\tau_n) \end{cases}$$

Un *arbre de code* représentant un code  $c$  est un arbre binaire, dont l'ensemble des sommets est  $\Sigma$ , tel que

- pour toute lettre  $\sigma \in \Sigma$ , l'étiquette du sommet  $\sigma$  est  $c(\sigma)$ ,
- pour toutes lettres  $\sigma$  et  $\tau$  de l'alphabet  $\Sigma$ , si  $\tau$  appartient au sous-arbre gauche de  $\sigma$ , alors on a  $\tau < \sigma$  dans l'ordre de l'alphabet  $\Sigma$ ; si  $\tau$  appartient au sous-arbre droit de  $\sigma$ , alors on a  $\tau > \sigma$  dans l'ordre de l'alphabet  $\Sigma$ .

**Indication OCaml :** On adopte les déclarations de type suivantes afin de représenter les mots binaires et les arbres de code en OCaml :

```
type binaire = bool list;;
type code =
| Nil
| Noeud of unicode * binaire * code * code;;
```

- 22 – Écrire une fonction encodeur, de type  $\text{texte} \rightarrow \text{code} \rightarrow \text{bool list}$ , qui, à partir d'un texte  $t$  et d'un code  $c$ , renvoie le codage du mot du texte  $t$  par le codage associé au code  $c$ .

On note  $\text{prof}_{\mathcal{A}}(\sigma)$  la profondeur d'un sommet  $\sigma$  dans un arbre  $\mathcal{A}$ , c'est-à-dire le nombre de sommets de l'unique chemin entre le sommet  $\sigma$  et la racine de  $\mathcal{A}$ . En particulier, la racine  $\rho$  d'un arbre  $\mathcal{A}$  est de profondeur  $\text{prof}_{\mathcal{A}}(\rho) = 1$ .

- 23 – Pour un code  $c$  donné, exprimer la complexité en temps de la fonction encodeur à l'aide de  $L = \max_{\sigma \in \Sigma} |c(\sigma)|$ , des profondeurs des sommets de l'arbre de code  $\mathcal{A}$  et de la distribution des fréquences dans le mot  $w$  donnés en argument

## 2.2 Arbre de code optimal

On fixe un code  $c$  de l'alphabet  $\Sigma$  ainsi que des poids réels positifs sous la forme d'une application  $f : \Sigma \rightarrow \mathbb{R}_+$ . La *profondeur pondérée* d'un arbre de code  $\mathcal{A}$  représentant le code  $c$  est la somme

$$\text{Prof}(\mathcal{A}) = \sum_{\sigma \in \Sigma} f(\sigma) \text{prof}_{\mathcal{A}}(\sigma).$$

- 24 – Exprimer la profondeur pondérée  $\text{Prof}(\mathcal{A})$  d'un arbre de code  $\mathcal{A}$  en fonction de  $f$ , de la profondeur pondérée  $\text{Prof}(\mathcal{A}_g)$  du sous-arbre gauche  $\mathcal{A}_g$  de l'arbre de code  $\mathcal{A}$  et de la profondeur pondérée  $\text{Prof}(\mathcal{A}_d)$  du sous-arbre droit  $\mathcal{A}_d$  de l'arbre de code  $\mathcal{A}$ .

On dit qu'un arbre de code est *optimal* si sa profondeur pondérée est la plus petite des profondeurs pondérées des arbres de code représentant un code de l'alphabet  $\Sigma$ . Pour tous entiers  $u$  et  $v$  compris entre 0 et  $\lambda - 1$  avec  $u \leq v$ , on note  $c_{u,v}$  la restriction du code  $c$  à l'alphabet  $\Sigma_{u,v} = \{\sigma_u, \sigma_{u+1}, \dots, \sigma_{v-1}, \sigma_v\}$ . On note  $\Pi_{u,v}$  la profondeur pondérée d'un arbre de code représentant le code  $c_{u,v}$  optimal.

- 25 – Pour tout entier  $u$  compris entre 0 et  $\lambda - 1$ , calculer  $\Pi_{u,u}$ .  
Pour tous entiers  $u$  et  $v$  compris entre 0 et  $\lambda - 1$  vérifiant  $u < v$ , exprimer une relation entre  $\Pi_{u,v}$  et les quantités  $(\Pi_{u,r})_{u \leq r \leq v-1}$  et  $(\Pi_{r,v})_{u+1 \leq r \leq v}$ .

- 26 – Concevoir et présenter un algorithme qui reçoive en entrée un code  $c : \Sigma \rightarrow \{0, 1\}^+$  et une distribution de fréquences  $f : \Sigma \rightarrow \mathbb{N}$  sous la forme de tableaux et qui construise un arbre de code optimal en temps  $O(\lambda^3)$ . Il n'est pas exigé d'utiliser la syntaxe OCaml pour répondre à cette question. On décomposera l'algorithme en sous-programmes élémentaires dont on caractérisera les entrées, les sorties ou les effets suffisamment pour que l'établissement d'une preuve de correction de l'algorithme soit facile. Cette preuve de correction n'est pas exigée.  
On justifiera la complexité en temps.

## 2.3 Un arbre de code optimal calculé plus rapidement

Pour tous entiers  $u$  et  $v$  avec  $0 \leq u \leq v \leq \lambda - 1$ , on appelle  $r_{u,v}$  le plus grand entier  $r$  inférieur à  $\lambda - 1$  tel qu'il existe un arbre optimal pour le code  $c_{u,v}$  ayant la lettre  $\sigma_r$  comme racine. On se propose de démontrer, pour tout  $n$  compris entre 0 et  $\lambda - 2$ , l'encadrement

$$\forall 0 \leq u \leq v \leq \lambda - 2 \text{ avec } v - u \leq n, \quad r_{u,v} \leq r_{u,v+1} \leq r_{u+1,v+1}. \quad (\mathcal{E}(n))$$

- 27 – Déduire de l'encadrement  $\mathcal{E}(\lambda - 2)$  une modification de l'algorithme proposé en réponse à la question 26 afin que son temps d'exécution soit  $O(\lambda^2)$ . Détaillez le calcul de la complexité en temps.

Soient  $u$  et  $v$  deux entiers avec  $1 \leq u \leq v \leq \lambda - 2$ .

- 28 – Montrer que, dans un arbre de code qui représente le code  $c_{u,v}$ , le sommet qui contient la lettre  $\sigma_v$  ne possède jamais de fils droit.
- 29 – Dans cette question, on suppose que le poids  $f(\sigma_{v+1})$  est nul. Soit  $\mathcal{A}$  un arbre de code optimal pour le code  $c_{u,v}$  et  $\mathcal{A}'$  l'arbre obtenu en ajoutant à  $\mathcal{A}$  le sommet  $\sigma_{v+1}$  comme fils droit du sommet qui contient la lettre  $\sigma_v$ . Montrer que  $\mathcal{A}'$  est un arbre de code optimal pour le code  $c_{u,v+1}$ .

Dans les trois questions qui suivent, on suppose que les poids  $(f(\sigma_x))_{u \leq x \leq v}$  restent constants tandis que le poids  $f(\sigma_{v+1})$  est variable.

- 30 – Montrer que l'application

$$\pi_{u,v+1} : \begin{cases} \mathbb{R}_+ & \rightarrow \mathbb{R}_+ \\ f(\sigma_{v+1}) & \mapsto \Pi_{u,v+1} \end{cases}$$

est une application continue et affine par morceaux sur  $\mathbb{R}_+$  dont on précisera la pente pour chaque morceau et que, pour tout intervalle ouvert  $I \subseteq \mathbb{R}_+$  sur lesquels l'application  $\pi_{u,v+1}$  est affine, les indices  $(r_{u',v'})_{u \leq u' \leq v' \leq v+1}$  sont constants lorsque  $f(\sigma_{v+1})$  varie dans  $I$ .

Soit  $\alpha \in \mathbb{R}_+^*$  un point de changement de pente de l'application  $\pi_{u,v+1}$ , soit  $\beta^- < \alpha$  un réel tel que l'application  $\pi_{u,v+1}$  soit affine sur l'intervalle  $I^- = ]\beta^-, \alpha[$  et soit  $\beta^+ > \alpha$  un réel tel que l'application  $\pi_{u,v+1}$  soit affine sur l'intervalle  $I^+ = ]\alpha, \beta^+[$ .

On construit deux suites  $(d_k^-)$  et  $(d_k^+)$  par récurrence. Chaque suite s'arrête si le terme suivant n'est plus défini. Pour construire la première suite, on choisit  $f(\sigma_{v+1})$  dans l'intervalle  $I^-$  et on pose

$$\begin{cases} \text{si } d_k^- < v + 1, & d_0^- = r_{u,v+1} \\ & d_{k+1}^- = r_{d_k^- + 1, v+1} \end{cases}.$$

Pour construire la seconde suite, on choisit  $f(\sigma_{v+1})$  dans l'intervalle  $I^+$  et on pose

$$\begin{cases} \text{si } d_k^+ < v + 1, & d_0^+ = r_{u,v+1} \\ & d_{k+1}^+ = r_{d_k^+ + 1, v+1} \end{cases}.$$

Dans la mesure où les entiers  $r_{u,v}$  dépendent de la valeur de  $f(\sigma_{v+1})$ , les deux suites  $(d_k^-)$  et  $(d_k^+)$  ne sont pas forcément égales et ne dépendent que du fait que  $f(\sigma_{v+1})$  appartient à  $I^-$  ou à  $I^+$ .

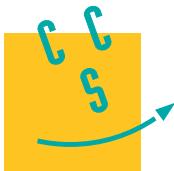
31 – Montrer que les suites  $(d_k^-)$  et  $(d_k^+)$  sont finies et que, en appelant  $m^-$  l'indice du dernier terme défini de la suite  $(d_k^-)$  et  $m^+$  l'indice du dernier terme défini de la suite  $(d_k^+)$ , on a  $m^- > m^+$ .

On suppose avoir démontré la validité de l'encadrement  $\mathcal{E}(n)$  pour un certain entier  $n$ . On suppose que les entiers  $u$  et  $v$  vérifient  $v - u \leq n + 1$ .

32 – En faisant l'hypothèse  $d_0^+ < d_0^-$ , montrer qu'il existe un indice  $s < m^+$  tel qu'on ait  $d_s^- = d_s^+$  et tel que, pour tout entier  $\ell$  compris entre 0 et  $s - 1$ , on ait  $d_\ell^+ < d_\ell^-$ . En déduire une contradiction en échangeant les descendants du sommet  $\sigma_{d_s^-}$  et du sommet  $\sigma_{d_s^+}$  dans certains arbres.

33 – Montrer que l'encadrement  $(\mathcal{E}(n))$  est satisfait pour tous entiers  $n$  avec  $n \leq \lambda - 2$ .

FIN DE L'ÉPREUVE



CONCOURS CENTRALE-SUPÉLEC

# Option informatique

MP

2020

4 heures

Calculatrice autorisée

## *Un système de vote*

Le seul langage de programmation autorisé dans cette épreuve est Caml.

Toutes les fonctions des modules `Array` et `List`, ainsi que les fonctions de la librairie standard (celles qui s'écrivent sans nom de module, comme `max` ou `incr` ainsi que les opérateurs comme `@`) peuvent être librement utilisées. Les candidats ne devront faire appel à aucun autre module.

En Caml, les matrices d'entiers sont représentées par des tableaux de tableaux, c'est-à-dire par le type `int array array`. L'expression `m.(i).(j)` permet d'accéder au coefficient de la  $i$ -ème ligne et de la  $j$ -ème colonne de la matrice `m`. Dans le texte, en dehors du code Caml, ce coefficient sera noté  $m[i, j]$ . On rappelle les définitions suivantes :

- `Array.make_matrix : int -> int -> 'a -> 'a array array` est telle que `Array.make_matrix n p v` renvoie une matrice de  $n$  lignes et  $p$  colonnes dont toutes les cases contiennent la valeur `v` ;
- `Array.length : 'a array -> int` est telle que `Array.length tab` renvoie le nombre d'éléments du tableau `tab`. Si `tab` est une matrice, c'est le nombre de lignes de `tab` ;
- `min : 'a -> 'a -> 'a` renvoie le minimum des deux valeurs en argument ;
- `max : 'a -> 'a -> 'a` renvoie le maximum des deux valeurs en argument ;
- l'opérateur `@` concatène deux listes. Par exemple, `[2; 0] @ [4; 2]` renvoie `[2; 0; 4; 2]`.

Dans ce problème, les graphes ont un ensemble de sommets de la forme  $\{0, 1, \dots, n-1\}$  et sont orientés complets et pondérés par des entiers : pour tout couple  $(i, j)$  de sommets distincts, il existe un arc de  $i$  à  $j$  de poids  $m_{i,j} \in \mathbb{Z}$ . Le graphe est représenté par sa matrice d'adjacence  $M = (m_{i,j})_{0 \leq i, j \leq n-1}$ , avec la convention que  $m_{i,i} = 0$  pour tout sommet  $i$  (il n'y a pas d'arc d'un sommet vers lui-même).

## I Vote par préférence

Nous considérons une élection, à laquelle se présentent  $n$  candidats. Chaque électeur inscrit sur son bulletin l'ensemble des candidats (tous les candidats sont classés), par ordre de préférence. L'ensemble des bulletins est rassemblé dans une urne. Le nombre de votants est noté  $p$ , l'urne contient donc  $p$  bulletins.

Les trois types de données suivants sont utilisés pour représenter un vote :

- `type candidat = int` ;; chaque candidat est désigné par un numéro de 0 à  $n-1$  ;
- `type bulletin = candidat list` ;; un bulletin de vote est une liste (ordonnée) de candidats ;
- `type urne = bulletin list` ;; une urne est un ensemble de bulletins de vote.

Par exemple, s'il y a trois candidats et qu'un électeur préfère le candidat 2, puis le candidat 0 et enfin considère que le candidat 1 est le moins souhaitable, son bulletin de vote sera `[2; 0; 1]`.

Une fois le vote effectué, on compare les résultats de deux candidats particuliers  $i$  et  $j$  en comptant le nombre de bulletins qui classent le candidat  $i$  avant le candidat  $j$ . On exprime le résultat de la comparaison entre les candidats  $i$  et  $j$  en calculant la différence entre le nombre de bulletins de vote qui placent  $i$  avant  $j$  et le nombre de ceux qui placent  $j$  avant  $i$ .

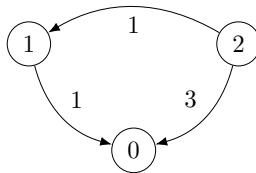
### I.A – Premier exemple

On considère une élection avec trois candidat et trois votants :  $n = 3$ ,  $p = 3$ . Les bulletins sont `[2; 0; 1]`, `[2; 1; 0]` et `[1; 2; 0]`. La comparaison entre le candidat numéro 0 et le candidat numéro 1 donne  $-1$  car le candidat 0 est placé une fois avant le candidat 1 et deux fois après.

**Q 1.** Écrire une fonction `duel : candidat -> candidat -> urne -> int`, telle que `duel i j u` renvoie la comparaison entre le candidat  $i$  et le candidat  $j$  à partir des bulletins contenus dans l'urne `u`.

On peut alors synthétiser le contenu d'une urne `u` en construisant le *graphe de préférence* des votants : ses sommets correspondent aux candidats et l'arc du sommet  $i$  vers le sommet  $j$  est pondéré par la comparaison entre le candidat  $i$  et le candidat  $j$ , c'est-à-dire la valeur de `duel i j u`.

La figure 1 donne le graphe de préférence obtenu à partir du vote de l'exemple 1 ainsi que sa matrice d'adjacence  $M$ . Afin d'alléger le schéma, seuls les arcs avec un poids strictement positif sont représentés.



$$M = \begin{pmatrix} 0 & -1 & -3 \\ 1 & 0 & -1 \\ 3 & 1 & 0 \end{pmatrix}$$

**Figure 1****I.B – Deuxième exemple**

On considère une élection avec trois candidats et 4 votants :  $n = 3$ ,  $p = 4$ . À l'issue du vote, le contenu de l'urne est  $\{[0; 1; 2], [1; 2; 0], [0; 2; 1], [0; 2; 1]\}$ .

**Q 2.** Tracer le graphe de préférence de l'urne (en ne dessinant que les arcs ayant un poids strictement positif) et donner sa matrice d'adjacence.

**I.C – Construction du graphe de préférence**

**Q 3.** Expliquer pourquoi la matrice d'adjacence d'un graphe de préférence est antisymétrique et pourquoi tous ses coefficients non-diagonaux ont la même parité.

Étant donné une urne  $U$ , on note  $\text{Mat}(U)$  la matrice du graphe de préférence associée à cette urne.

**Q 4.** Écrire une fonction `depouillement : int → urne → int array array` qui prend en paramètres le nombre de candidats  $n$  et une urne  $U$  et renvoie la matrice  $\text{Mat}(U)$ .

**I.D – Théorème de McGarvey**

Le but de cette sous-partie est de démontrer le théorème de McGarvey : « Pour toute matrice antisymétrique à coefficients pairs  $M$ , il existe une urne  $U$  telle que  $M = \text{Mat}(U)$ . »

Soient  $i$ ,  $j$  et  $n$  trois entiers naturels tels que  $i < n$ ,  $j < n$  et  $i \neq j$ . On note  $E_{i,j,n}$  la matrice carrée de taille  $n$  dont tous les coefficients sont nuls sauf les coefficients d'indices  $(i, j)$  et  $(j, i)$  qui valent respectivement 2 et  $-2$ .

**Q 5.** Soit  $n$  un nombre de candidats et  $i$  et  $j$  deux entiers naturels strictement inférieurs à  $n$ . Montrer qu'il existe une urne  $U_{i,j,n}$  contenant deux votes telle que  $\text{Mat}(U_{i,j,n}) = E_{i,j,n}$ .

**Q 6.** On considère deux urnes  $U_1$  et  $U_2$ . Exprimez  $M_3 = \text{Mat}(U_1 \cup U_2)$  en fonction de  $M_1 = \text{Mat}(U_1)$  et de  $M_2 = \text{Mat}(U_2)$ .

**Q 7.** Démontrer le théorème de McGarvey.

**Q 8.** Écrire une fonction `mccarvey : int array array → urne` prenant en paramètre une matrice antisymétrique  $M$  de coefficients tous pairs et renvoyant une urne  $U$  telle que  $M = \text{Mat}(U)$ .

**Q 9.** Estimer la complexité de la fonction `mccarvey` en fonction de  $n$ , la taille de la matrice (c'est-à-dire le nombre de candidats), et de  $q$ , le maximum des coefficients de la matrice.

**II Recherche du vainqueur**

L'objectif de cette partie est de déterminer le vainqueur, ou les vainqueurs *ex aequo*, d'un vote par préférence.

**II.A – Vainqueur de Condorcet**

On appelle *vainqueur de Condorcet* tout sommet tel que, dans le graphe de préférence, les arcs sortant de ce sommet ont tous un poids positif ou nul. Ainsi, dans le premier exemple de la partie I, le candidat 2 est un vainqueur de Condorcet.

**Q 10.** Expliquer pourquoi un vainqueur de Condorcet peut être qualifié de « vainqueur » de l'élection.

**Q 11.** Écrire une fonction `condorcet : int array array → candidat list` prenant en paramètre la matrice d'adjacence d'un graphe de préférence et renvoyant la liste des vainqueurs de Condorcet.

**Q 12.** En se plaçant dans le cas  $n = 3$  et  $p = 3$ , construire une urne pour laquelle il n'existe pas de vainqueur de Condorcet. Tracer le graphe de préférence correspondant.

**II.B – Graphe intermédiaire de Schulze**

À la fin du XX<sup>e</sup> siècle, Markus Schulze imagina une méthode permettant de déterminer un vainqueur à l'issue de n'importe quel vote par préférence.

On appelle *poids d'un chemin* du graphe de préférence, le **minimum** des poids des arcs constituant ce chemin. Ainsi, dans le premier exemple de la partie I, le poids du chemin  $2 \rightarrow 0 \rightarrow 1$  est  $-1$ .

Le *graphe intermédiaire de Schulze* est un graphe orienté pondéré complet dont les sommets sont les candidats et dont l'arc du sommet  $i$  vers le sommet  $j$  (distinct de  $i$ ) est pondéré par le **maximum** des poids de tous les chemins allant de  $i$  à  $j$  dans le graphe de préférence. Sa matrice d'adjacence est notée  $I$ .

**Q 13.** Pour  $i$  et  $j$ , candidats distincts, démontrer que  $I[i, j]$  est aussi le maximum des poids des chemins sans boucle de  $i$  à  $j$ , c'est-à-dire des chemins  $i = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_k = j$  avec  $i_0, i_1, \dots, i_k$  deux à deux distincts.

**Q 14.** Démontrer que  $\min(I[i, j], I[j, k]) \leq I[i, k]$  pour tout triplet  $(i, j, k)$  de sommets distincts.

**Q 15.** En adaptant l'algorithme de Floyd-Warshall, programmer une fonction `intermediaire : int array array -> int array array` prenant en paramètre la matrice d'adjacence d'un graphe de préférence et renvoyant la matrice d'adjacence du graphe intermédiaire de Schulze correspondant.

**Q 16.** Estimez la complexité de la fonction `intermediaire` en fonction de  $n$ , le nombre de candidats.

**Q 17.** Serait-il pertinent d'utiliser l'algorithme de Dijkstra au lieu de l'algorithme de Floyd-Warshall ? Argumenter la réponse.

#### II.C – Graphe de préférence de Schulze

Le *graphe de préférence de Schulze* est défini à partir du graphe intermédiaire de Schulze. Si  $I$  est la matrice d'adjacence du graphe intermédiaire de Schulze, alors la matrice d'adjacence  $S$  du graphe de préférence de Schulze est définie par  $S[i, j] = I[i, j] - I[j, i]$  pour tous entiers naturels  $i$  et  $j$  strictement inférieurs à  $n$ .

**Q 18.** Montrer que la matrice d'adjacence d'un graphe de préférence de Schulze est antisymétrique et que tous ses coefficients sont pairs.

**Q 19.** Écrire une fonction `graphe_schulze : int array array -> int array array` qui prend en paramètre un graphe intermédiaire de Schulze et qui renvoie le graphe de préférence de Schulze correspondant.

#### II.D – Vainqueur de Schulze

Un *vainqueur de Schulze* est un vainqueur de Condorcet dans le graphe de préférence de Schulze.

**Q 20.** Écrire une fonction `schulze : int -> urne -> candidat list` qui prend en paramètres le nombre de candidats et un ensemble de bulletins de vote et renvoie la liste des vainqueurs de Schulze.

**Q 21.** Estimer la complexité de la fonction `schulze` en fonction du nombre de candidats  $n$  et du nombre de votants  $p$ .

À partir d'un graphe de préférence de Schulze représenté par la matrice  $S$ , on définit la relation  $R_S$  entre les candidats comme suit :  $i R_S j$  si et seulement  $S[i, j]$  est strictement positif.

**Q 22.** Montrer que la relation  $R_S$  est transitive, c'est-à-dire que pour tous candidats  $i$ ,  $j$  et  $k$ , si  $i R_S j$  et  $j R_S k$  alors  $i R_S k$ .

Si  $I$  désigne la matrice d'adjacence du graphe intermédiaire, on pourra distinguer les cas  $I[i, j] \leq I[j, k]$  et  $I[i, j] > I[j, k]$ .

**Q 23.** Montrer que quelle que soit l'urne non vide considérée, il existe toujours au moins un vainqueur de Schulze.

### III Satisfiabilité d'une formule de logique propositionnelle

Étant donné une variable propositionnelle  $x$ , on appelle *littéral*, les formules  $x$  et  $\neg x$  ( $\neg x$  est la négation de  $x$ ). On appelle *clause* une disjonction de littéraux, par exemple  $x \vee \neg y \vee z$  est une clause ( $\vee$  signifie « ou »).

On appelle *conjonction de clauses*, une formule qui est la conjonction entre plusieurs clauses. Par exemple  $(x \vee \neg y \vee z) \wedge (x \vee \neg y)$  est une conjonction de clauses ( $\wedge$  signifie « et »).

On appelle *interprétation* une fonction qui à chaque variable propositionnelle associe une valeur de vérité (vrai ou faux).

On dit qu'une conjonction de clauses est *satisfiable* s'il existe une interprétation qui la rend vraie, c'est-à-dire qui rend vrai toutes ses clauses, autrement dit qui rend vrai au moins un littéral de chacune de ses clauses.

Le problème SAT consiste à déterminer si une conjonction de clauses est satisfiable :

**Entrées :** Une conjonction de clauses  $\varphi$ .

**Sortie :** Un booléen. Vrai si  $\varphi$  est satisfiable. Faux sinon.

**Q 24.** Montrer qu'il est possible de résoudre le problème SAT avec une complexité en  $O(2^n m)$  où  $n$  est le nombres de variables distinctes et  $m$  le nombre de littéraux de l'entrée (comptés avec leurs répétitions).

On considère un vote par préférence pour lequel on a réparti les candidats en trois groupes :

- un candidat  $c$ , appelé *champion* ;
- un ensemble  $A$  de candidats dits *automatiques* ;
- un ensemble  $B$  de candidats *optionnels*.

Le problème appelé CONTROL-ADD-ALT consiste à déterminer s'il est possible d'éliminer un certain nombre de candidats optionnels de façon à ce que  $c$  soit un vainqueur de Schulze de l'élection :

**Entrées :** Le candidat  $c$ . L'ensemble  $A$ . L'ensemble  $B$ . Le graphe de préférence  $G$  du vote sur l'ensemble des candidats  $\{c\} \cup A \cup B$ . Un budget  $k \in \mathbb{N}$ .

**Sortie :** Un booléen. Vrai, s'il est possible de choisir  $k$  candidats distincts  $b_1, \dots, b_k$  dans  $B$  tels que  $c$  est un vainqueur de Schulze de l'élection en considérant uniquement l'ensemble de candidats  $\{c\} \cup A \cup \{b_1, \dots, b_k\}$ . Faux sinon.

Pour obtenir le graphe de préférence de l'élection avec l'ensemble de candidats  $\{c\} \cup A \cup \{b_1, \dots, b_k\}$ , on prend le graphe de préférence sur l'ensemble des candidats  $\{c\} \cup A \cup B$  et on supprime les candidats de  $B$  qui n'ont pas été choisis (ainsi que les arrêtes entrantes et sortantes de ces candidats).

On appelle *instance* de CONTROL-ADD-ALT une entrée du problème CONTROL-ADD-ALT et instance de SAT une entrée du problème SAT.

On considère l'algorithme « Transformation d'une formule en élection » qui, étant donné une instance de SAT (c'est-à-dire une conjonction de clauses), construit une instance de CONTROL-ADD-ALT :

**Entrées :** Une conjonction de clauses  $\varphi$

**Sortie :** Un candidat  $c$  (le champion), un ensemble de candidats automatiques  $A$ , un ensemble de candidats optionnels  $B$ , un budget  $k$ , un graphe de préférence  $G$  sur l'ensemble de candidats  $\{c\} \cup A \cup B$

$\psi \leftarrow \varphi$

**pour tout**  $x$  variable propositionnelle apparaissant dans  $\varphi$  **faire**

$\psi \leftarrow \psi \wedge (x \vee \neg x)$

**fin pour**

Créer un nouveau candidat  $c$

Créer un graphe  $G$  avec un seul sommet  $c$

$A \leftarrow \emptyset, B \leftarrow \emptyset$

**pour tout**  $cl$  clause de  $\psi$  **faire**

Créer un nouveau candidat  $a_{cl}$  associé à  $cl$  et l'ajouter à  $G$  et à  $A$

Ajouter à  $G$  un arc de poids +4 de  $a_{cl}$  vers  $c$  et un arc de poids -4 en sens inverse.

**fin pour**

**pour tout**  $x$  variable propositionnelle apparaissant dans  $\psi$  **faire**

Créer un nouveau candidat  $b_x$  associé au littéral  $x$  et l'ajouter à  $G$  et à  $B$

Créer un nouveau candidat  $b_{\neg x}$  associé au littéral  $\neg x$  et l'ajouter à  $G$  et à  $B$

Ajouter à  $G$  deux arcs de poids +6 de  $c$  vers  $b_x$  et vers  $b_{\neg x}$  puis ajouter deux arcs de poids -6 en sens inverse.

**fin pour**

**pour tout** couple  $(li, cl)$  avec  $li$  un littéral de  $\psi$  et  $cl$  une clause de  $\psi$  contenant le littéral  $li$  **faire**

Ajouter à  $G$  un arc de poids +6 de  $b_{li}$  vers  $a_{cl}$  et un arc de poids -6 en sens inverse.

**fin pour**

Compléter le graphe  $G$  avec des arrêtes de poids nul pour le rendre complet.

$k \leftarrow$  le nombre de variables propositionnelles qui apparaissent dans  $\psi$ .

**renvoyer**  $c, A, B, G, k$ .

**Q 25.** Donner le graphe de préférence créé à partir de la formule  $(x \vee x) \wedge (\neg x \vee \neg x)$ . Est-il possible de choisir  $k = 1$  candidat parmi  $x$  et  $\neg x$  de sorte que  $c$  gagne ?

**Q 26.** Montrer que si on peut choisir  $k$  candidats optionnels de sorte à faire gagner le champion, alors, pour toute variable propositionnelle  $x$ , on a choisi  $x$  ou  $\neg x$  mais on n'a pas choisi  $x$  et  $\neg x$ .

**Q 27.** Justifier que  $\varphi$  est satisfiable si et seulement si l'instance de CONTROL-ADD-ALT créée par cet algorithme à partir de  $\varphi$  donne une réponse positive.

On dit qu'un algorithme est en temps polynomial en un paramètre  $m$ , si la complexité de cet algorithme est majorée par un polynôme en  $m$ .

**Conjecture.** Il n'existe pas d'algorithme en temps polynomial en nombre de littéraux de l'entrée qui résout SAT.

**Q 28.** Montrer que, si la conjecture précédente est vraie, alors il n'existe pas d'algorithme en temps polynomial en nombre de candidats qui résout CONTROL-ADD-ALT.

## Partie I - Logique et calcul des propositions

Dans la suite, les variables propositionnelles seront notées  $x_1, x_2, \dots$ . Les connecteurs propositionnels  $\wedge$  (conjonction),  $\vee$  (disjonction),  $\Rightarrow$  (implication) et  $\Leftrightarrow$  (équivalence) seront classiquement utilisés.

De même, la négation d'une variable propositionnelle  $x_i$  (respectivement d'une formule  $\mathcal{F}$ ) sera notée  $\neg x_i$  (resp.  $\neg \mathcal{F}$ ).

### I.1 - Définitions

**Définition 1** (Minterme, maxterme).

Soit  $(x_1 \cdots x_n)$  un ensemble de  $n$  variables propositionnelles.

- On appelle minterme toute formule de la forme  $y_1 \wedge y_2 \wedge \cdots y_n$  où pour tout  $i \in \{1, \dots, n\}$   $y_i$  est un élément de  $\{x_i, \neg x_i\}$ .
- On appelle maxterme toute formule de la forme  $y_1 \vee y_2 \vee \cdots y_n$  où pour tout  $i \in \{1, \dots, n\}$   $y_i$  est un élément de  $\{x_i, \neg x_i\}$ .

Les mintermes (respectivement maxtermes)  $y_1 \wedge y_2 \wedge \cdots y_n$  et  $y'_1 \wedge y'_2 \wedge \cdots y'_n$  (resp  $y_1 \vee y_2 \vee \cdots y_n$  et  $y'_1 \vee y'_2 \vee \cdots y'_n$ ) sont considérés identiques si les ensembles  $\{y_i, 1 \leq i \leq n\}$  et  $\{y'_i, 1 \leq i \leq n\}$  le sont.

**Q1.** Donner l'ensemble des mintermes et des maxtermes sur l'ensemble  $(x_1, x_2)$ .

**Définition 2** (Formes normales conjonctives et disjonctives).

Soit  $\mathcal{F}$  une formule propositionnelle qui s'écrit à l'aide de  $n$  variables propositionnelles  $(x_1 \cdots x_n)$ .

- On appelle forme normale conjonctive de  $\mathcal{F}$  toute conjonction de maxtermes logiquement équivalente à  $\mathcal{F}$ .
- On appelle forme normale disjonctive de  $\mathcal{F}$  toute disjonction de mintermes logiquement équivalente à  $\mathcal{F}$ .

**Définition 3** (Système complet).

Un ensemble de connecteurs logiques  $C$  est un système complet si toute formule propositionnelle est équivalente à une formule n'utilisant que les connecteurs de  $C$ .

Par définition,  $C = \{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$  est un système complet.

### I.2 - Le connecteur de Sheffer

On définit le connecteur de Sheffer, ou d'incompatibilité, par  $x_1 \diamond x_2 = \neg x_1 \vee \neg x_2$ .

**Q2.** Construire la table de vérité du connecteur de Sheffer.

**Q3.** Exprimer ce connecteur en fonction de  $\neg$  et  $\wedge$ .

**Q4.** Vérifier que  $\neg x_1 = x_1 \diamond x_1$ .

**Q5.** En déduire une expression des connecteurs  $\wedge, \vee$  et  $\Rightarrow$  en fonction du connecteur de Sheffer. Justifier en utilisant des équivalences avec les formules propositionnelles classiques.

**Q6.** Donner une forme normale conjonctive de la formule  $x_1 \diamond x_2$ .

**Q7.** Donner de même une forme normale disjonctive de la formule  $x_1 \diamond x_2$ .

**Q8.** Démontrer par induction sur les formules propositionnelles que l'ensemble de connecteurs  $C = \{\diamond\}$  est un système complet.

- Q9.** Application : soit  $\mathcal{F}$  la formule propositionnelle  $x_1 \vee (\neg x_2 \wedge x_3)$ . Donner une forme logiquement équivalente de  $\mathcal{F}$  utilisant uniquement le connecteur de Sheffer.

## Partie II - Le problème de Freudenthal (Informatique pour tous)

L'objectif de cette partie est de proposer une implémentation en langage Python d'une solution au problème de Freudenthal.

Hans Freudenthal (1905-1990), mathématicien allemand naturalisé néerlandais, spécialiste de topologie algébrique, est connu pour ses contributions à l'enseignement des mathématiques. En 1969, il soumet à une revue mathématique le problème suivant :

*Un professeur dit à ses deux étudiants Sophie et Pierre : "J'ai choisi deux entiers  $x$  et  $y$ , tels que  $1 < x < y$  et  $x + y \leq n$ . J'ai confié à Pierre la valeur  $\Pi$  du produit de  $x$  et  $y$ . J'ai confié à Sophie la valeur  $\Sigma$  de la somme de  $x$  et  $y$ . Pierre, Sophie, je vous demande de trouver  $x$  et  $y$ ."*

*Pierre et Sophie engagent alors le dialogue suivant :*

- *Pierre : "Je ne connais pas les nombres  $x$  et  $y$ ."*
- *Sophie : "Avant même que tu me le dises, je savais déjà que tu ne connaissais pas  $x$  et  $y$ ."*
- *Pierre : "Ah ! eh bien maintenant je connais  $x$  et  $y$ ."*
- *Sophie : "Très bien, mais moi aussi alors maintenant je connais  $x$  et  $y$ ."*

Dans la suite, on note  $\mathcal{N}_n = \{(x, y) \in \mathbb{N}^2, 1 < x < y \text{ et } x + y \leq n\}$ .

Si la discussion entre Sophie et Pierre semble stérile, une quantité importante d'informations est cependant échangée qui amène au bout du dialogue à la solution.

- Q10.** À quelle condition sur  $x$  et  $y$  Pierre aurait-il pu dire dès le début : "Je connais  $x$  et  $y$ " ?

Puisque Pierre ne peut répondre tout de suite, cela signifie que le produit  $\Pi$  peut s'écrire pour plusieurs couples d'entiers  $(x, y) \in \mathcal{N}_n$ .

- Q11.** Écrire une fonction `CoupleProd(n)` qui renvoie la liste des entiers  $P$  pour lesquels il existe au moins deux couples  $(x, y) \in \mathcal{N}_n$  tels que  $xy = P$ . Par exemple, `CoupleProd(9)=[12]` puisque  $12 = 3 \times 4 = 2 \times 6$  et qu'aucune autre valeur ne satisfait la propriété.

Sophie savait déjà que Pierre ne connaissait pas la réponse. C'est donc que, pour tout  $(x, y) \in \mathcal{N}_n$  qui satisfait  $x + y = \Sigma$ , le produit  $xy$  est dans la liste précédente.

- Q12.** Soit un entier  $S \leq n$ . Écrire une fonction `Prod(S, n)` qui renvoie, pour l'ensemble des  $(x, y) \in \mathcal{N}_n$  tels que  $x + y = S$ , la liste des entiers  $P = xy$ . Par exemple, `Prod(8, 9)` retourne `[12, 15]` puisque  $8 = 6 + 2 = 3 + 5$ .

- Q13.** Pour  $S \leq n$ , en déduire une fonction `Candidat_S(n)` qui renvoie la liste des entiers  $S$  tels que la liste `Prod(S, n)` est incluse dans la liste `CoupleProd(n)`.

Pierre peut maintenant déduire la valeur de  $\Sigma$  du fait qu'elle appartient à la liste renournée par la fonction `Candidat_S(n)`. Plus précisément, le produit  $\Pi$  n'apparaît dans la liste `Prod(S, n)` que pour une seule valeur de  $S$  de la liste `Candidat_S(n)`. Pour déterminer cet unique  $S$ , on recherche tout d'abord les produits  $P$  pour lesquels :

- il existe deux sommes  $S_1$  et  $S_2$  dans la liste `Candidat_S(n)` telles que  $S_1 < S_2$  ;
- $P$  apparaît dans les listes `Prod(S1, n)` et `Prod(S2, n)`.

**Q14.** Écrire une fonction `Double_P(n)` qui renvoie la liste des produits  $P$  satisfaisant ces deux conditions.

Il reste à construire une fonction `Reste_S(n)` permettant de ne retenir que les sommes  $S$  de la liste `Candidat_S(n)` pour lesquelles il existe un unique élément en commun entre les listes `Prod(S,n)` et `Double_P(n)`.

**Q15.** Écrire une fonction `Reste_S(n)` qui renvoie la liste de ces sommes.

Pour que Pierre conclue, il faut que la liste `Reste_S(n)` soit réduite à un singleton. Pour que Sophie conclue également, il lui suffit de rechercher les éléments de la liste `Prod(S,n)` qui ne sont pas dans `Double_P(n)`.

**Q16.** Pour  $S \leq n$ , écrire une fonction `Reste_P(S,n)` qui renvoie la liste de ces produits.

Les deux étudiants connaissent maintenant  $\Sigma$  et  $\Pi$ .

**Q17.** Pour  $S \leq n$ , écrire une fonction `Solution(P,S,n)` qui retourne le couple  $(x,y)$  recherché.

## Partie III - Mots de Lyndon et de de Bruijn

Cette partie comporte des questions nécessitant un code Caml. Pour ces questions, les réponses ne feront pas appel aux fonctionnalités impératives de langage (en particulier pas de boucles, pas de références).

On considère ici un alphabet totalement ordonné de  $k$  symboles, noté  $\Sigma$ .

### III.1 - Mots de Lyndon

#### III.1.1 - Définitions

##### Définition 4 (Mot).

*Un mot est une suite finie de longueur  $n$  de symboles  $\mathbf{m} = m_0 \cdots m_{n-1}$  où pour tout  $i$ ,  $m_i \in \Sigma$  et  $n \geq 1$  est la longueur de  $\mathbf{m}$ . On notera  $n = |\mathbf{m}|$ .*

On note  $\Sigma^n$  l'ensemble des mots de longueur  $n$  construits sur  $\Sigma$  et  $\Sigma^*$  l'ensemble des mots de longueur quelconque construits sur  $\Sigma$ . On note enfin  $\epsilon$  le mot vide.

Le type Caml choisi pour représenter un mot est une chaîne de caractères (`string`). Les éléments de  $\Sigma$  sont représentés par le type `char`.

Dans toute la suite, les seules fonctions/méthodes Caml sur les chaînes de caractères qui peuvent être utilisées sont :

- `S.[i]` (valeur du  $i^e$  caractère)
- l'opérateur de concaténation `^`
- la fonction `String.length : string -> int`  
`String.length s` retourne la longueur de `s`.
- la fonction `String.sub : string -> int -> int -> string`.  
`String.sub s start 1` retourne une chaîne de longueur 1 contenant la sous-chaîne de `s` qui commence en `start`.

**Définition 5** (Préfixe, suffixe).

Soient  $\mathbf{m} = m_0 \cdots m_{|m|-1}$  et  $\mathbf{p} = p_0 \cdots p_{|p|-1}$  deux mots de  $\Sigma^*$  :

- $\mathbf{mp} = m_0 \cdots m_{|m|-1}p_0 \cdots p_{|p|-1} \in \Sigma^{|m|+|p|}$  est le concaténé de  $\mathbf{m}$  et  $\mathbf{p}$ .
- $\mathbf{m}$  est un préfixe de  $\mathbf{p}$  si  $|m| < |p|$  et  $m_i = p_i$  pour  $0 \leq i \leq |m| - 1$ .
- $\mathbf{m}$  est un suffixe de  $\mathbf{p}$  si  $|m| < |p|$  et  $m_i = p_{|p|-|m|+i}$  pour  $0 \leq i \leq |m| - 1$ .

On définit alors la relation  $\prec$  par :

$$\mathbf{m} \prec \mathbf{p} \Leftrightarrow (\mathbf{m} \text{ est un préfixe de } \mathbf{p}) \text{ ou} \\ (\text{Il existe } k \in \llbracket 0, |m| - 1 \rrbracket \text{ tel que pour tout } i \in \llbracket 0, k - 1 \rrbracket, m_i = p_i \text{ et } m_k < p_k).$$

**Q18.** On donne le type

```
type comparaison = Inferieur | Egal | Superieur ;;
```

Écrire une fonction recursive `ordre : string -> string -> comparaison` telle que `ordre m p` est l'ordre relatif des mots  $\mathbf{m}$  et  $\mathbf{p}$ .

**Définition 6** (Ordre lexicographique).

La relation définie par :  $\mathbf{m} \leq \mathbf{p}$  si et seulement si  $\mathbf{m} = \mathbf{p}$  ou  $\mathbf{m} \prec \mathbf{p}$  est appelée ordre lexicographique sur  $\Sigma^*$ .

**Définition 7** (Conjugué).

Soit  $\mathbf{m} \in \Sigma^n$ . Un conjugué de  $\mathbf{m}$  est un mot de la forme  $m_i \cdots m_{n-1}m_0 \cdots m_{i-1}$ , pour  $i \in \llbracket 0, n - 1 \rrbracket$ . Par convention, le conjugué de  $\mathbf{m}$  pour  $i = 0$  est le mot  $\mathbf{m}$  lui-même.

**Q19.** Écrire une fonction Caml `conjugue : string -> int -> string` telle que `conjugue m i` retourne le conjugué de  $\mathbf{m}$  débutant par le  $i^{\text{e}}$  caractère de  $\mathbf{m}$ ,  $0 \leq i \leq |\mathbf{m}| - 1$ .

La notion de conjugaison induit une relation  $C$  définie sur  $\Sigma^*$  par  $\mathbf{m} C \mathbf{p}$  si et seulement si  $\mathbf{p}$  est un conjugué de  $\mathbf{m}$ .

**Q20.** Montrer que  $C$  est une relation d'équivalence.**Définition 8** (Collier).

Un collier est le plus petit mot dans l'ordre lexicographique d'une classe de mots équivalents par la relation  $C$ .

Un collier d'ordre  $n$  est dit périodique s'il peut s'écrire  $\mathbf{m}^l$ , où  $\mathbf{m} \in \Sigma^r$ ,  $r \geq 2$  et  $l > 1$ . Il est dit apériodique sinon, autrement dit si deux conjugaisons non triviales des membres de sa classe d'équivalence ne sont jamais égales.

**Définition 9** (Mot de Lyndon).

Un mot  $\mathbf{m} \in \Sigma^*$  est un mot de Lyndon si c'est un collier apériodique.

**Q21.** On suppose  $0 < 1$ . Pour les mots suivants, indiquer si ce sont ou non des mots de Lyndon. Dans le cas négatif, justifier votre réponse :

- (i). 0010011.
- (ii). 010011.
- (iii). 001001.

**Q22.** Écrire une fonction Caml `Lyndon : string -> bool` telle que `Lyndon m` renvoie `true` si  $\mathbf{m}$  est un mot de Lyndon. Cette fonction fera appel à une fonction récursive.

### III.1.2 - Génération de mots de Lyndon

Soit  $\mathbf{m} \in \Sigma^*$  un mot de Lyndon. Pour générer à partir de  $\mathbf{m}$  un mot de Lyndon  $\mathbf{q}$  de longueur au plus  $n \geq |\mathbf{m}|$  sur  $\Sigma$ , on utilise l'**algorithme 1**.

---

#### Algorithme 1 : Algorithme de génération d'un mot de Lyndon

---

**Données :**  $\mathbf{m} \in \Sigma^*$  un mot de Lyndon,  $n \geq |\mathbf{m}|$

**Résultat :**  $\mathbf{q}$  le mot de Lyndon généré à partir de  $\mathbf{m}$ .

\*\*\* *Etape 1* \*\*\*

Concaténer le mot  $\mathbf{m}$  à lui même jusqu'à obtenir un mot  $\mathbf{q}$  de longueur  $n$ . La dernière occurrence de  $\mathbf{m}$  pourra être tronquée pour arriver à un mot de longueur exactement  $n$ .

\*\*\* *Etape 2* \*\*\*

**tant que** le dernier symbole de  $\mathbf{q}$  est le plus grand symbole de  $\Sigma$  faire

- └ Ôter ce symbole de  $\mathbf{q}$ .

\*\*\* *Etape 3* \*\*\*

Remplacer le dernier symbole de  $\mathbf{q}$  par le symbole qui suit dans  $\Sigma$ .

**retourner**  $\mathbf{q}$ .

---

**Q23.** Donner l'indice dans  $\mathbf{m}$  de la  $i^e$  lettre de  $\mathbf{q}$  en fonction de  $i$  et  $|\mathbf{m}|$ .

**Q24.** Pour  $\Sigma = \{0, 1\}$ , on donne le mot de Lyndon  $\mathbf{m} = 00111$  et  $n = 9$ . Donner le mot de Lyndon généré par l'algorithme, en déroulant les différentes étapes produites par l'algorithme permettant d'aboutir au mot de Lyndon.

L'**algorithme 1** peut être utilisé pour générer tous les mots de Lyndon de longueur au plus  $n$ . Pour ce faire, on part du plus petit symbole de  $\Sigma$  et on itère les trois étapes (1-2-3) de l'algorithme jusqu'à arriver au mot vide.

**Q25.** Partant de  $\mathbf{m} = 0$  et toujours pour  $\Sigma = \{0, 1\}$ , construire par l'algorithme tous les mots de Lyndon de longueur au plus 4.

**Q26.** Donner la complexité de l'**algorithme 1** au pire des cas en nombre d'ajouts ou de suppressions de caractères.

### III.1.3 - Factorisation de mots de Lyndon

**Définition 10** (Factorisation de Lyndon).

Soit  $\mathbf{m} \in \Sigma^*$ . Une factorisation de  $\mathbf{m}$  est une suite  $\mathbf{m}_1, \dots, \mathbf{m}_l$  de mots de Lyndon telle que  $\mathbf{m} = \mathbf{m}_1 \cdots \mathbf{m}_l$  avec  $\mathbf{m}_1 \geq \mathbf{m}_2 \cdots \geq \mathbf{m}_l$ .

On admet le résultat suivant :

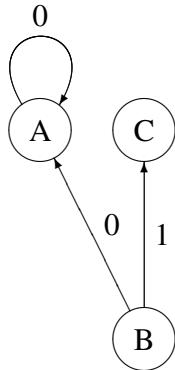
**Théorème 1** (Factorisation d'un mot).

Tout mot  $\mathbf{m} \in \Sigma^*$  admet une unique factorisation de Lyndon.

L'**algorithme 2** propose une méthode de factorisation d'un mot  $\mathbf{m}$  (on ne demande pas de le justifier). Le principe est d'itérer sur la chaîne des symboles de  $\mathbf{m}$  pour trouver le plus grand mot de Lyndon possible. Lorsqu'un tel mot est trouvé, il est ajouté à la liste  $\mathcal{L}$  des facteurs de  $\mathbf{m}$  et la recherche est poursuivie sur la sous-chaîne restante.

**Q27.** En utilisant l'**algorithme 2**, écrire une fonction `factorisation`  $\mathbf{m}$  qui réalise la factorisation d'un mot  $\mathbf{m}$  donné. Cette fonction fera appel à une ou des fonction(s) récursive(s).

**Algorithme 2 : Algorithme de factorisation d'un mot de Lyndon****Données :**  $\mathbf{m} \in \Sigma^*$  un mot de Lyndon.**Résultat :** la liste  $\mathcal{L}$  des mots de Lyndon décroissants de la factorisation de  $\mathbf{m}$ . $\mathcal{L} \leftarrow []$  $j \leftarrow 1$  $k \leftarrow 0$ **tant que**  $j \leq |\mathbf{m}|$  faire    **si**  $j = |\mathbf{m}|$  ou  $m_k > m_j$  alors         $\mathbf{p} = m_0 \cdots m_{j-k-1}$         Ajouter  $\mathbf{p}$  à  $\mathcal{L}$         Supprimer  $\mathbf{p}$  de  $\mathbf{m}$          $k \leftarrow 0$          $j \leftarrow 1$ **sinon**    **si**  $m_k = m_j$  alors         $k \leftarrow k + 1$          $j \leftarrow j + 1$ **sinon**         $k \leftarrow 0$          $j \leftarrow j + 1$ **retourner**  $\mathcal{L}$ .**III.2 - Mots de de Bruijn****III.2.1 - Définition****Définition 11** (Mot de de Bruijn).*Un mot de de Bruijn d'ordre n sur  $\Sigma$  est un collier qui contient tous les mots de  $\Sigma^n$  une et une seule fois.*Par exemple, pour  $\Sigma = \{a, b, c\}$  et  $n = 2$ ,  $\mathbf{m} = abacbbccca$  est un mot de de Bruijn puisqu'il contient une unique fois chaque mot de longueur 2 sur  $\Sigma$ , le mot 'aa' étant obtenu par circularité de  $\mathbf{m}$ .**Q28.** Donner la longueur d'un mot de de Bruijn en fonction de  $n$  et du nombre  $k$  de symboles de  $\Sigma$ .**III.2.2 - Graphe de de Bruijn****Définition 12** (Graphe de de Bruijn).*Le graphe de de Bruijn d'ordre n sur  $\Sigma$  est le graphe orienté  $\mathcal{B}(k, n) = (V, E)$  où :*-  $V = \Sigma^n$  est l'ensemble des sommets du graphe ;-  $E = \{(am, mb), a, b \in \Sigma, m \in \Sigma^{n-1}\}$  est l'ensemble des arcs orientés du graphe.*On value les arcs  $E$  par le dernier symbole du noeud terminal de chaque arc : ainsi  $(am, mb) \in E$  est étiqueté par  $b$ .*Dans ce graphe, certains arcs ont pour sommet initial et terminal un même sommet de  $V$ . Ces arcs sont appelés des boucles (**figure 1**).**Q29.** Donner l'ensemble des mots de longueur 3 sur  $\Sigma = \{0, 1\}$ . En déduire le graphe de de Bruijn  $\mathcal{B}(2, 3)$  associé.**Q30.** Montrer que le degré entrant et le degré sortant de chaque sommet est égal à  $k$ .**Q31.** En déduire le nombre d'arcs orientés  $|E|$  en fonction de  $k$  et  $n$ .



**Figure 1** - Exemple de graphe avec boucle sur le sommet A.

**Définition 13** (Successeur, prédécesseur).

Soit  $G = (V, E)$  un graphe orienté.  $v \in V$  est un successeur (respectivement prédécesseur) de  $u \in V$  si  $(u, v) \in E$  (resp.  $(v, u) \in E$ ).

**Q32.** Soient  $\mathcal{B}(k, n) = (V, E)$  et  $\mathbf{m} \in V$ . Montrer que tous les prédécesseurs de  $\mathbf{m}$  ont le même ensemble de successeurs.

**Q33.** Soit  $\mathbf{p} = (p_0 \cdots p_{n-1}) \in V$ . Donner l'ensemble des sommets  $\mathbf{m}$  tels que  $(\mathbf{p}, \mathbf{m}) \in E$ .

On suppose disposer de  $\mathcal{B}(k, n)$  et on souhaite construire  $\mathcal{B}(k, n + 1)$  à partir de ce graphe.

**Q34.** Proposer une méthode pour construire les sommets de  $\mathcal{B}(k, n + 1)$  à partir des arcs de  $\mathcal{B}(k, n)$ .  
Donner le sommet créé dans  $\mathcal{B}(2, 4)$  à partir de l'arc (001,010) de  $\mathcal{B}(2, 3)$ .

On dit que deux arcs  $e_1, e_2 \in E$  sont adjacents dans  $\mathcal{B}(k, n)$  si ces deux arcs s'écrivent  $e_1 = (\mathbf{m}, \mathbf{p})$  et  $e_2 = (\mathbf{p}, \mathbf{q})$  pour  $\mathbf{m}, \mathbf{p}, \mathbf{q} \in V$ .

**Q35.** Proposer de même une construction des arcs de  $\mathcal{B}(k, n + 1)$  en fonction des arcs adjacents de  $\mathcal{B}(k, n)$ . Donner l'arc de  $\mathcal{B}(2, 4)$  créé par les arcs adjacents de  $\mathcal{B}(2, 3)$  (001,011) et (011,110).

### III.2.3 - Construction des mots de de Bruijn

On propose ici trois algorithmes de construction de mots de de Bruijn.

#### III.2.3.1 - Construction à l'aide de $\mathcal{B}(k, n)$

Les mots de de Bruijn d'ordre  $n$  sur  $\Sigma$  peuvent être construits en parcourant  $\mathcal{B}(k, n)$ .

**Définition 14** (Circuit eulérien).

Soit  $G$  un graphe orienté. Un circuit eulérien est un chemin dont l'origine et l'extrémité coïncident et passant une fois et une seule par chaque arête de  $G$ .

**Q36.** Construire  $\mathcal{B}(2, 2)$  et trouver dans ce graphe un circuit eulérien. Vérifier que la concaténation des étiquettes lues au fil de ce circuit donne un représentant d'un mot de de Bruijn et donner son ordre sur  $\{0, 1\}$ .

On admet les résultats suivants :

- (i). un graphe  $G = (V, E)$  possède un circuit eulérien si et seulement si il est connexe et si, pour tout  $v \in V$ , le degré entrant de  $v$  et le degré sortant de  $v$  sont égaux.
- (ii). un circuit eulérien dans le graphe  $\mathcal{B}(k, n)$  correspond à un mot de de Bruijn.

**Q37.** En déduire qu'il existe au moins un mot de de Bruijn pour tout alphabet  $\Sigma$  et tout  $n$ .

Ainsi, la concaténation des étiquettes lues au fil d'un circuit eulérien de  $\mathcal{B}(k, n)$  donne un représentant d'un mot de de Bruijn d'ordre  $n + 1$  sur  $k$  symboles.

### III.2.3.2 - Construction à l'aide de l'algorithme Prefer One

Pour construire les mots de de Bruijn d'ordre  $n$  sur  $\Sigma = \{0, 1\}$ , on peut également utiliser l'**algorithme 3**, dit algorithme Prefer One.

#### Algorithme 3 : Algorithme Prefer One

**Données :**  $n, \Sigma$

**Résultat :**  $\mathbf{m}$  mot de de Bruijn de longueur  $n$  sur  $\Sigma$ .

$\mathbf{m} \leftarrow$  suite de  $n$  zeros

$\text{STOP} \leftarrow \text{false}$

**tant que**  $\text{STOP}=\text{false}$  faire

**Etape 1**

        Ajouter un 1 à la fin de  $\mathbf{m}$ .

**si** les  $n$  derniers symboles de  $\mathbf{m}$  n'ont pas été rencontrés auparavant **alors**

            Répéter Etape 1

**sinon**

            Retirer le 1 ajouté à la fin de  $\mathbf{m}$ .

            Passer à Etape 2

**Etape 2**

        Ajouter un 0 à la fin de  $\mathbf{m}$ .

**si** les  $n$  derniers symboles de  $\mathbf{m}$  n'ont pas été rencontrés auparavant **alors**

            Aller à l'Etape 1

**sinon**

$\text{STOP} \leftarrow \text{true}$

**retourner**  $\mathbf{m}$ .

Dans cet algorithme, "les  $n$  derniers symboles de  $\mathbf{m}$  n'ont pas été rencontrés auparavant" signifie que le mot composé des  $n$  derniers symboles de  $\mathbf{m}$  n'est pas un sous-mot de  $\mathbf{m}$ , situé entre le premier et le  $(|\mathbf{m}| - 1)^{\text{e}}$  symbole de  $\mathbf{m}$ .

**Q38.** Appliquer l'algorithme au cas  $n = 3$  et  $\Sigma = \{0, 1\}$ . Écrire les valeurs successives de  $\mathbf{m}$  au cours de l'exécution.

### III.2.3.3 - Construction à l'aide de la relation aux mots de Lyndon

La troisième construction utilise les notions de collier et de mots de Lyndon.

**Q39.** Donner, pour  $k = 2$  et  $n = 4$ , les classes d'équivalence de tous les mots binaires de longueur 4 pour la relation  $C$ . En déduire les colliers correspondants.

**Q40.** En déduire les mots de Lyndon de longueur 4 pour  $\Sigma = \{0, 1\}$ .

Plus généralement, les mots de de Bruijn et de Lyndon sont étroitement liés. On peut montrer que si l'on concatène, dans l'ordre lexicographique, les mots de Lyndon sur  $k$  symboles dont la longueur divise un

entier  $n$ , alors on obtient le mot de de Bruijn le plus petit, pour l'ordre lexicographique, de tous les mots de de Bruijn de longueur  $n$  sur  $k$  symboles.

**Q41.** Déduire de la question précédente le plus petit mot de de Bruijn pour  $n = 4$  et  $\Sigma = \{0, 1\}$ .

### III.2.4 - Application

La soirée a été longue, vous rentrez chez vous mais, au pied de votre immeuble, vous vous trouvez confronté à un sérieux problème : vous avez complètement oublié le code d'entrée à  $n$  chiffres de la porte. On suppose que le digicode de l'appartement est composé de  $1 \leq k \leq 10$  chiffres  $0, 1, \dots, k-1$ , qui constituent l'alphabet  $\Sigma$ .

Le digicode fonctionne de la façon suivante : vous tapez successivement sur les chiffres afin de composer un mot. À chaque nouveau symbole entré à partir du  $n$ -ième, le digicode teste le mot constitué par les  $n$  derniers chiffres pour voir s'il correspond au code secret.

Ainsi, par exemple pour  $n = 4$ , si vous tapez la séquence 021201, le digicode teste successivement 0212, 2120 et 1201.

Étant pressé de regagner votre lit, vous cherchez à taper un minimum de touches pour ouvrir la porte. On note  $\tilde{n}$  la longueur de la plus petite séquence de frappe de touches qui vous permet de rentrer dans l'immeuble.

**Q42.** Donner un encadrement de  $\tilde{n}$  :

- pour la borne supérieure, on considérera que l'on met bout à bout tous les mots possibles de  $n$  chiffres construits sur  $\Sigma$  ;
- pour la borne inférieure, on cherchera un mot sans redondance, c'est-à-dire qui contient une et une seule fois chaque mot de  $n$  chiffres.

**Q43.** Expliquer en une phrase en quoi les mots de de Bruijn peuvent vous aider. En vous inspirant de l'**algorithme 3**, donner une séquence la plus courte de chiffres à taper pour ouvrir à coup sûr la porte de votre immeuble, lorsque  $n = 2$  et  $k = 4$ .

**Q44.** Comparer alors le nombre maximum de frappes de touches du digicode à effectuer en utilisant les mots de de Bruijn, avec celui calculé par la méthode brute. Calculer ces nombres pour :

- $k = 4$  et  $n = 2$ .
- $k = 10$  et  $n = 4$ .

Que conclure quant à l'efficacité de votre stratégie ?

**FIN**

## Consignes Python

Tout code doit être écrit dans le langage Python.

- Tout code Python non indenté ne sera pas corrigé.
  - Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué ci-dessous.
  - Vous pouvez écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
- Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.

### Fonctions et méthodes autorisées

Vous pouvez utiliser la fonction range :

```
>>> for i in range(10):
...:     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...:     print(i, end=' ')
5 6 7 8 9
```

Sur les listes, vous pouvez utiliser la méthode append et la fonction len :

```
>>> help ( list . append )
Help on method_descriptor : append (...)
L. append ( object ) -> None -- append object to end of L

>>> help (len)
Help on built-in function len in module builtins : len (...)
len ( object )
Return the number of items of a sequence or collection .
```

Les matrices sont représentées par des listes de listes comme dans l'exemple ci-dessous :

```
>>> M1 = [[1, 10, 3, 0, 3, 10, 1],
...:         [1, 0, 1, 8, 1, 0, 1],
...:         [10, 9, 4, 1, 4, 9, 10],
...:         [10, 3, 7, 1, 7, 3, 10],
...:         [7, 8, 5, 1, 5, 8, 7]]
```

## Introduction

Le sujet de l'option informatique contient quatre exercices allant de l'écriture de fonctions à l'analyse de code. Le langage utilisé, dans tous les cas, est Python. Le premier exercice parle d'arbres binaires, les exercices 2 et 3 traitent de matrices, quant au 4ème il s'agit de coder le principe de Kaprekar à l'aide de sous-fonctions et de listes. Vous trouverez en annexe les différentes fonctions et structures de données autorisées.

### A) Arbres binaires : occurrences ... ( 2 points)

Un arbre binaire est une structure par nature récursive. Il peut être soit vide ( $\emptyset$ ), soit la composée d'un noeud racine et de deux sous-arbres ( $\langle o, G, D \rangle$ ) où  $o$  est le noeud racine, et  $G$  et  $D$  sont deux arbres binaires disjoints (respectivement sous-arbre gauche et sous-arbre droit). Un nœud peut donc être le père de 0, 1 ou 2 nœuds fils (respectivement gauche et droit) suivant que l'arbre dont il est racine soit la composée de 0, 1 ou deux sous-arbres non vide.

La taille d'un arbre binaire est définie comme étant le nombre de nœuds qui le composent et sa hauteur comme étant le maximum des hauteurs de ses nœuds : la hauteur d'un nœud étant égale à 0 s'il est la racine de l'arbre et à la hauteur de son père plus 1 dans tous les autres cas.

Notons qu'il est possible de représenter un arbre binaire sous forme d'occurrences. Dans ce cas, si un nœud a pour occurrence le mot  $\mu$ , alors son fils gauche aura pour occurrence le mot  $\mu 0$  et son fils droit le mot  $\mu 1$ .

Nous nous proposons donc de représenter un arbre binaire à l'aide d'une chaîne de caractères constituée des diverses occurrences de cet arbre, séparées les unes des autres par une virgule. Nous représenterons la racine par la chaîne vide.

Soit l'arbre  $B$  défini sous forme d'occurrences et représenté à l'aide d'une chaîne de caractères de la manière suivante :

$$B = ", 0, 1, 00, 01, 10, 11, 001, 101, 1010, 1011"$$

Comment sans représenter l'arbre  $B$  peut-on déterminer :

- 1)  $T(B)$  la taille de  $B$  ?
- 2)  $H(B)$  la hauteur de  $B$  ?
- 3) Le fait qu'un nœud de  $B$  soit une feuille (pas de fils) ?
- 4)  $LC(B)$  la longueur de cheminement de  $B$  (la somme des hauteurs des nœuds de l'arbre) ?

**B) Matrices : transposée ... (3 points)**

La matrice transposée d'une matrice  $\mathbf{A}$  est la matrice  $\mathbf{A}^T$  obtenue en échangeant les lignes et les colonnes de  $\mathbf{A}$ .

$$\mathbf{A} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \text{ alors } \mathbf{A}^T = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Écrire la fonction **transpose(M)** qui construit et retourne la matrice transposée de la matrice  $\mathbf{M}$  non vide.

**C) Matrices : symétrie ... (5 points)**

|    |   |    |    |   |
|----|---|----|----|---|
| 1  | 1 | 10 | 10 | 7 |
| 10 | 0 | 9  | 3  | 8 |
| 3  | 1 | 4  | 7  | 5 |
| 0  | 8 | 1  | 1  | 1 |
| 3  | 1 | 4  | 7  | 5 |
| 10 | 0 | 9  | 3  | 8 |
| 1  | 1 | 10 | 10 | 7 |

Mat1

|    |   |    |    |   |
|----|---|----|----|---|
| 1  | 1 | 10 | 10 | 7 |
| 10 | 0 | 9  | 3  | 8 |
| 3  | 1 | 4  | 7  | 5 |
| 3  | 1 | 4  | 7  | 5 |
| 10 | 0 | 9  | 3  | 8 |
| 1  | 1 | 10 | 10 | 7 |

Mat2

|    |   |    |    |   |
|----|---|----|----|---|
| 1  | 1 | 10 | 10 | 7 |
| 10 | 0 | 9  | 3  | 8 |
| 3  | 1 | 4  | 7  | 5 |
| 0  | 8 | 1  | 1  | 1 |
| 3  | 1 | 4  | 7  | 5 |
| 10 | 0 | 9  | 3  | 8 |
| 1  | 1 | 10 | 10 | 6 |

Mat3

Écrire la fonction **symetric(M)** qui vérifie si une matrice  $\mathbf{M}$  est symétrique selon l'axe horizontal (symétrie verticale).

*Exemples d'applications sur les matrices Mat1, Mat2 et Mat3 :*

```
>>> symetric(Mat1)
True

>>> symetric(Mat2)
True

>>> symetric(Mat3)
False
```

## D) Procédé de Kaprekar... (16 points)

Les exercices de cette partie sont indépendants. Le dernier exercice utilise les fonctions des exercices précédents, mais il n'est pas obligatoire de les avoir écrites.

Pour toutes les fonctions, on supposera que les paramètres sont valides : il n'y a pas de test à faire, pas d'exception à déclencher.

### 1) Test ... (1 point)

Soit la fonction **test** suivante :

```
1 def test(x, L):
2     i = len(L) - 1
3     while i >= 0 and L[i] != x:
4         i = i - 1
5     return i >= 0
```

Que fait cette fonction ?

### 2) Entiers <-> liste ... (6 points)

- a) Écrire la fonction **int\_to\_list(n,p)** qui convertit le nombre **n** (entier naturel à au plus **p** chiffres) en une liste de ses chiffres éventuellement complétés par des **0** pour atteindre **p** chiffres.

*Exemples d'applications (l'ordre des chiffres dans le résultat importe peu) :*

```
>>> int_to_list(27972,5)
[2, 7, 9, 7, 2]

>>> int_to_list(42,5)
[2, 4, 0, 0, 0]

>>> int_to_list(258,4)
[8, 5, 2, 0]
```

- b) Écrire la fonction **list\_to\_ints(L)** qui, à partir de la liste **L** non vide ne contenant que des chiffres (de 0 à 9), retourne le couple (*Left*,*right*), avec :

- **left** le nombre construit avec les chiffres de **L** lus de gauche à droite,
- **right** le nombre construit avec les chiffres de **L** lus de droite à gauche.

*Exemples d'applications :*

```
>>> list_to_ints([1,2,3,4,5,6])
(123456, 654321)
```

```
>>> list_to_ints([2,4,0,0,0,])
(24000, 42)
```

### 3) Histogramme et tri ... (4 points)

Nous travaillons ici avec des listes qui ne contiennent que des chiffres (de 0 à 9).

- a) Écrire la fonction **hist(L)** qui retourne un histogramme (sous la forme d'une liste) des chiffres présents dans la liste L.

Rappel : l'histogramme H est une liste telle que  $H[i]$  est le nombre d'occurrences de la valeur  $i$ . Par exemple dans la première application ci-dessous, il y a 5 valeurs 0, 3 valeurs 1, 3 valeurs 2, ...

*Exemples d'applications :*

```
>>> hist([1,5,9,3,0,1,2,0,1,0,2,5,0,5,2,0])
[5, 3, 3, 1, 0, 3, 0, 0, 0, 1]
```

```
>>> hist([1,0,1,0,1,0,1,0,1])
[4, 5, 0, 0, 0, 0, 0, 0, 0]
```

- b) Utiliser la fonction **hist** pour écrire la fonction **sort(L)** qui trie la liste L en ordre croissant (la fonction construit une nouvelle liste qui doit être renournée).

*Exemples d'application :*

```
>>> sort([1,5,9,3,0,1,2,0,1,0,2,5,0,5,2,0])
[0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 5, 5, 5, 9]
```

### 4) Kaprekar ... (5 points)

Procédé de Kaprekar :

Soit un entier n à p chiffres. Par exemple n = 6264.

- Prendre les p chiffres de n pour former deux nombres : le plus grand et le plus petit : 6642 et 2466,
- Les soustraire en complétant éventuellement par des 0 à gauche pour obtenir un nouveau nombre à p chiffres :  $n = 6642 - 2466 = 4176$ ,
- Recommencer le procédé avec le résultat :  $4176 \rightarrow 7641 - 1467 = 6174$ .

Le procédé de Kaprekar peut être utilisé avec un nombre quelconque. Dans tous les cas, quel que soit le nombre de chiffres, **on arrivera à une valeur déjà rencontrée** (dans les exemples donnés plus loin : on retrouve 6174 dans le premier, et 63 dans le deuxième), qui sera la valeur 0 si tous les chiffres sont identiques.

**Remarque :**

Nous travaillons ici toujours avec  $p$  chiffres. Cela signifie que si un résultat intermédiaire est inférieur à  $10^{p-1}$  (par exemple 999 lorsque  $p = 4$ ), les deux nombres seront construits à partir des chiffres du nombre complétés par des 0 (ici 999 et 9990).

La fonction à écrire **kaprekar(n,p)** prend en paramètre un entier naturel **n** et son nombre de chiffres **p** et applique le procédé de Kaprekar. Elle construit et retourne la liste les différentes valeurs calculées **c**, comme dans les exemples suivants.

*Exemples d'applications :*

```
>>> kaprekar(1574,4)
[1574, 6084, 8172, 7443, 3996, 6264, 4176, 6174, 6174]
// 6174 rencontré deux fois

>>> kaprekar(42,2)
[42, 18, 63, 27, 45, 9, 81, 63] // 63 rencontré deux fois

>>> kaprekar(666,3)
[666, 0, 0] // 0 rencontré deux fois
```

Vous pouvez utiliser les fonctions des questions 1 à 3 de cet exercice, même si vous ne les avez pas écrites.

Fin de l'énoncé

**Numéro de candidat:**

Document à insérer dans votre copie d'examen

**QCM**

Ce QCM est un peu particulier: chaque question peut comporter 0, 1, ou plusieurs bonnes réponses! Pour gagner des points, il faut répondre correctement (les mauvaises réponses feront perdre des points).

La majorité des questions suivantes sont formulées au singulier ou au pluriel, par commodité pour la grammaire française, mais sans corrélation directe avec le nombre de réponses correctes.

**1****? Cochez les phrases qui sont vraies**

- L'encodage ASCII est un code sur 7 bits
- le langage FORTRAN a plus de 50 ans et est encore utilisé
- Il n'y a pas de recherche académique en informatique
- Certains logiciels malveillants peuvent s'installer «sous» le système d'exploitation et même une réinstallation de la machine ne suffit pas à s'en débarrasser
- Il est possible de récupérer l'écran d'une machine à distance grâce aux fuites électromagnétiques. La seule façon de s'en prémunir est d'enfermer le système dans une cage de Faraday
- La monnaie virtuelle bitcoin est générée par des «miners» virtuels hébergés par le jeu MineCraft
- Avant 1992, il n'existe pas de led bleue
- Aucune réponse valide

**2****? Parmi les noms suivants, lesquels désignent un protocole réseau client-serveur**

- bittorrent
- html
- http
- mail
- telnet
- bitcoin
- Aucune réponse valide

### 3

❓ Associez les technologies aux entreprises en mettant le même numéro

1. Amazon

2. Facebook

3. Google

4. IBM

5. Microsoft

6. Oracle

AWS

Android

Azure

Mysql

Redhat Linux

Whatsapp

### 4

❓ En base de données, que fait la requête suivante

```
SELECT SUM(Bills.Amount) FROM Bills JOIN Users ON Bills.User=Users.Id  
WHERE Users.Name='Dujardin' and dueDate < date('2019-01-01');
```

- Elle affiche le total des factures dues par Dujardin avant le 1er janvier dernier
- Elle affiche le total des factures dues par Guillaume Dujardin après le 1er janvier dernier
- Elle affiche le total des factures dues par Dujardin entre le 1er janvier 2018 et le 1er janvier 2019

5

? Le premier disque dur, fabriqué par IBM, figure sur la photo suivante. Quelle était sa taille ?



- 50 Kilo octets
- 5 Mega octets
- 500 Mega octets
- 5 Giga octets
- 500 Giga octets
- Aucune réponse valide

6

? Qu'affiche le code python suivant

```
~ def puzzle(x):
    if x <= 0:
        return True
    else:
        return not(puzzle(x-3))

print(puzzle(503))
```

- True
- False
- RecursionError: maximum recursion depth exceeded in comparison
- Aucune réponse valide

7

❓ Qu'affiche le code python suivant

```
def b(y):
    if y % 4 == 0 :
        a = y % 100 != 0
    else:
        a = False
    if y % 400 == 0 :
        a = True
    if 1582 - y > 0 :
        a = y % 4
    return a

def pretty(x):
    if (b(x)):
        return "is"
    else:
        return "is not"

print(list(map(pretty, (2000, 1996, 2100, 1200, 2003))))
```

- ['is', 'is', 'is not', 'is not', 'is not']
- ['is', 'is', 'is', 'is', 'is not']
- ['is not', 'is not', 'is not', 'is not', 'is']
- ['is', 'is', 'is not', 'is', 'is']
- Aucune réponse valide

8

❓ Classez les langages de programmation suivants par date, du plus ancien (1) au plus récent (6)

- C
- Java
- JavaScript
- Lisp
- Python
- Rust

**9****? Cochez les algorithmes de chiffrement**

- Cesar
- ISF
- Neron
- RSA
- TVA
- Vigenere
- Aucune réponse valide

Cette épreuve est constituée de deux problèmes indépendants.

## Problème n° 1

Ce problème s'intéresse au processus de séquençage de l'ADN. Dans ce contexte, l'une des étapes importantes du séquençage permet de déterminer quelles sont les parties de la séquence d'ADN recherchée mais dans un ordre indéterminé. Le problème consiste alors à **reconstruire** la séquence complète à partir de ces parties. La suite de ce sujet propose une résolution algorithmique à ce problème.

Pour cela, nous représentons **une séquence**  $S_n$  d'ADN de longueur  $n$  par une suite de  $n$  lettres, chaque lettre appartenant à l'ensemble  $\{A, T, G, C\}$ .

*Exemple :  $S_9 = AGGTCAGGT$  est une séquence d'ADN de 9 lettres.*

### Recherche de mots dans une séquence d'ADN

Dans un premier temps nous allons chercher à déterminer quelles sont les parties de longueur fixe d'une séquence, appelées **mots de la séquence** par la suite.

Formellement, soient  $S_n$  une séquence d'ADN et  $l$  un entier tel que  $0 < l \leq n$ . On appelle **mot de  $S_n$  de longueur  $l$**  toute suite de  $l$  lettres contigües contenue dans la séquence  $S_n$ .

*Exemples : GTC est un mot de longueur 3 de la séquence  $S_9 = AGGTCAGGT$ , tandis que ATT n'en est pas un.*

1. **Combien de mots de longueur  $l$  existe-t-il dans une séquence de longueur  $n$  (en comptant les répétitions possibles) ?**
2. **Proposer un algorithme qui, étant donnés un entier  $n$ , une séquence d'ADN  $S_n$  de longueur  $n$  et un entier strictement positif  $l$  ( $l \leq n$ ), calcule la liste de tous les mots de longueur  $l$  de  $S_n$ .**

Une telle liste peut contenir plusieurs fois le même mot. Vous supposerez que la séquence d'ADN est contenue dans un tableau de caractères. Il n'est pas demandé de décrire les manipulations de listes que vous aurez éventuellement besoin d'effectuer, comme l'ajout d'un élément dans une liste par exemple.

*Exemple : l'algorithme appelé sur la séquence  $S_9 = AGGTCAGGT$  et  $l = 3$  doit calculer la liste [AGG, GGT, GTC, TCA, CAG, AGG, GGT].*

3. **Quelle est la complexité de votre algorithme en nombre d'opérations, en fonction de  $n$  et de  $l$  ?**

On s'intéresse à la question de déterminer tous les mots distincts de longueur  $l$  dans  $S_n$ . Nous cherchons ici à proposer une fonction Python. Une séquence  $S_n$  sera alors décrite comme une chaîne de caractères de longueur  $n$ .

*Exemple : la liste de tous les mots distincts de longueur 3 dans la séquence  $S_9 = "AGGT-CAGGT"$  où  $n = 9$  et  $l = 3$  est ["AGG", "GGT", "GTC", "TCA", "CAG"].*

4. Soient  $n$  et  $l$  deux entiers ( $n \geq l > 0$ ). Combien de mots distincts de longueur  $l$  existe-t-il au plus dans une séquence de longueur  $n$  ?
5. Proposer une structure de données en Python adaptée pour stocker les mots de la séquence sans répétition.
6. Proposer une fonction `liste_mots` en Python qui prend en arguments une séquence  $S_n$ , un entier  $n$  et un entier  $l$  ( $n \geq l > 0$ ), et qui renvoie la liste de tous les mots distincts de longueur  $l$  de  $S_n$ .

## Reconstruction d'une séquence d'ADN

L'une des difficultés lors du séquençage de l'ADN est qu'il faut reconstruire la séquence complète à partir de l'ensemble des mots de longueur  $l$  qui la composent, sans qu'aucune information sur leur position dans la séquence ne soit donnée.

Le reste de cet exercice va se concentrer précisément sur la résolution algorithmique de ce problème.

Soient  $S_n$  une séquence d'ADN de longueur  $n$  et  $l$  un entier tel que  $0 < l \leq n$ . On appelle  $\text{spectre}(S_n, l)$  l'ensemble des mots de longueur  $l$  distincts qui sont dans la séquence  $S_n$ .

*Exemple : étant donnée la séquence  $S_9 = AGTCAGGT$ ,  $\text{spectre}(S_9, 3) = \{\text{AGG}, \text{CAG}, \text{GGT}, \text{GTC}, \text{TCA}\}$ .*

Le problème de la **reconstruction** de la séquence d'ADN consiste, à partir de la connaissance d'un spectre  $\mathcal{SP}$  contenant des mots de longueur  $l$ , à déterminer une séquence d'ADN  $S$  compatible avec  $\mathcal{SP}$ , c'est-à-dire telle que  $\text{spectre}(S, l) = \mathcal{SP}$ . Plus formellement, on peut formuler le problème **Reconstruction** de la manière suivante :

**Données** : un entier  $l$ ; un ensemble  $\mathcal{SP}$  de mots de longueur  $l$ .

**Sortie** : une séquence d'ADN  $S$  telle que  $\text{spectre}(S, l) = \mathcal{SP}$ .

Pour résoudre ce problème, nous allons nous intéresser aux recouvrements existant entre les mots d'un spectre.

Soient  $M$  un mot de longueur  $n > 0$  et  $k$  un entier tel que  $0 < k \leq n$ , on appelle *préfixe* de  $M$  de longueur  $k$  le mot constitué des  $k$  premières lettres de  $M$ . De manière similaire, on appelle *suffixe* de  $M$  de longueur  $k$  le mot constitué des  $k$  dernières lettres de  $M$ .

Soient deux mots  $M_1$  et  $M_2$  de longueur  $n_1$  et  $n_2$  respectivement et soit  $k \leq \min(n_1, n_2)$  un entier strictement positif. Il existe un *recouvrement de longueur  $k$*  entre  $M_1$  et  $M_2$  si et seulement si le suffixe de  $M_1$  de longueur  $k$  est identique au préfixe de  $M_2$  de longueur  $k$ . Nous appellerons également *longueur du recouvrement maximal* entre  $M_1$  et  $M_2$ , noté  $\text{Irmax}(M_1, M_2)$  dans la suite, le plus grand entier  $k$  tel qu'il existe un recouvrement de longueur  $k$  entre  $M_1$  et  $M_2$ . Si aucun recouvrement n'existe entre  $M_1$  et  $M_2$ ,  $\text{Irmax}(M_1, M_2) = 0$ .

*Exemple :  $\text{Irmax}(\text{ACGG}, \text{CTAG}) = 0$  et  $\text{Irmax}(\text{ACGG}, \text{GGAT}) = 2$ .*

**7. Que valent :**

- $\text{Irm}_{\max}(ATGC, GGTA)$  ?
- $\text{Irm}_{\max}(TGGCGT, CGTAAATG)$  ?
- $\text{Irm}_{\max}(GCTAGGCTAA, AGGCTAAGTCGAT)$  ?
- $\text{Irm}_{\max}(TCTAGCCAGCTAGC, TAGCCAGCTAGCACT)$  ?

**Première modélisation**

Soient  $l \geq 2$  un entier et  $\mathcal{SP}$  un spectre contenant des mots de longueur  $l$ . Nous allons modéliser notre problème par un graphe orienté  $G = (V, E)$ , dans lequel :

- Chaque sommet de  $V$  correspond à un mot du spectre et chaque mot du spectre est représenté par un et un seul sommet.
- L'ensemble  $E$  contient un arc entre  $v_1 \in V$  et  $v_2 \in V$  si et seulement si  $\text{Irm}_{\max}(v_1, v_2) = l - 1$ .

**8. Quelle est la modélisation obtenue sous forme de graphe pour les spectres et longueurs suivants :**

- $\{GTGA, ATGA, GACG, CGTG, ACGT, TGAC\}$  et  $l = 4$  ?
- $\{TAC, ACC, ACG, CAC, CCG, CGT, CGC, GCA, GTA\}$  et  $l = 3$  ?

**9. Proposer une formulation du problème de reconstruction comme un parcours de graphe. De quel problème classique de théorie des graphes ce problème se rapproche-t-il ?****10. Proposer pour chacun des graphes obtenus dans la question 8 une solution au problème Reconstruction.****Deuxième modélisation**

Il est possible de modéliser autrement ce problème, toujours sous forme de graphe.

Nous modélisons maintenant les données ( $l \geq 2$  un entier et  $\mathcal{SP}$  un spectre contenant des mots de longueur  $l$ ) par un graphe orienté  $G = (V, E)$  où :

- $V$  est l'ensemble de tous les préfixes de longueur  $l - 1$  et de tous les suffixes de longueur  $l - 1$  des mots du spectre.
- L'ensemble  $E$  contient un arc entre les sommets  $v_1 \in V$  et  $v_2 \in V$  si et seulement si le spectre  $\mathcal{SP}$  contient un mot  $M$  de longueur  $l$  tel que  $v_1$  est le préfixe de  $M$  de longueur  $l - 1$  et  $v_2$  est le suffixe de  $M$  de longueur  $l - 1$ .

**11. Quelle est la modélisation obtenue sous forme de graphe pour les spectres suivants :**

- $\{GTGA, ATGA, GACG, CGTG, ACGT, TGAC\}$  et  $l = 4$  ?
- $\{TAC, ACC, ACG, CAC, CCG, CGT, CGC, GCA, GTA\}$  et  $l = 3$  ?

On rappelle que dans un graphe orienté  $G = (V, E)$ , un *chemin* d'origine  $u$  ( $\in V$ ) et d'extrémité  $v$  ( $\in V$ ) est défini par une suite d'arcs consécutifs reliant  $u$  à  $v$ . Un *circuit* est un chemin dont les deux extrémités sont identiques. Un chemin (resp. circuit) est dit *eulérien* s'il contient une fois et une seule chaque arc de  $G$ .

12. **Les graphes construits à l'aide de la modélisation de la question 11 contiennent-ils un circuit eulérien ? un chemin eulérien ?**
13. **En quoi un circuit ou un chemin eulérien dans un graphe construit avec cette modélisation constitue-t-il une solution au problème de la reconstruction de séquence d'ADN ?**

Dans la suite de l'exercice, nous nous restreignons au circuit eulérien qui est plus simple à déterminer qu'un chemin eulérien.

Soit  $(u, v) \in E$ . On dit que l'arc  $(u, v)$  est un arc entrant du sommet  $v$  et un arc sortant du sommet  $u$ . On définit le degré entrant d'un sommet  $v$ , noté  $d_e(v)$ , comme le nombre d'arcs entrants du sommet  $v$ . On définit le degré sortant d'un sommet  $u$ , noté  $d_s(u)$ , comme le nombre d'arcs sortants du sommet  $u$ .

14. **Montrer qu'un graphe orienté contient un circuit eulérien si et seulement si**  
 $\forall v \in V, d_e(v) = d_s(v)$ .

Dans la suite de l'exercice, un graphe sera représenté en Python par un dictionnaire dont les clefs sont des chaînes de caractères (str) représentant les sommets du graphe et les valeurs sont des listes de chaînes de caractères (list) contenant les sommets voisins de la clé.

15. **Écrire une fonction `presence_circuit_eulerien` en Python qui prend en arguments un graphe orienté  $G$  et qui détermine si  $G$  a un circuit eulérien.**

Nous supposons que les graphes considérés dans les questions 16, 17, 18, 19 et 20 comprennent un circuit eulérien.

16. **Écrire une fonction `construction_circuit` en Python qui prend en arguments un graphe orienté  $G$  et un sommet  $v$  quelconque de  $G$  et qui renvoie un circuit ayant pour origine le sommet  $v$ .**

17. **Écrire une fonction `enleve_circuit` en Python qui prend en arguments un graphe orienté  $G$  et un circuit  $C$  de  $G$  et qui supprime de  $G$  les arcs appartenant au circuit  $C$ .**

18. **Écrire une fonction `sommet_commun` en Python qui prend en arguments un graphe orienté  $G$  et un circuit  $C$  (qui n'est pas inclus dans  $G$ ) et qui renvoie un sommet commun de degré non nul appartenant à la fois à  $G$  et à  $C$ . Nous supposerons que  $G$  et  $C$  ont au moins un tel sommet en commun.**

19. **Écrire une fonction `fusion_circuits` en Python qui prend en arguments deux circuits  $C_1$  et  $C_2$  et un sommet commun à  $C_1$  et  $C_2$  et qui renvoie un circuit composé des arcs de  $C_1$  et des arcs de  $C_2$ .**

20. **À partir des fonctions Python que vous avez proposées aux questions 16, 17, 18 et 19, écrire une fonction `circuit_eulerien` en Python qui prend un entrée un graphe orienté  $G$  et qui renvoie un circuit eulérien de  $G$ .**

21. **Quelle est la séquence d'ADN déterminée par la fonction proposée à la question 20 sur le graphe obtenu à la question 11 avec le spectre  $\mathcal{SP} = \{TAC, ACC, ACG, CAC, CCG, CGT, CGC, GCA, GTA\}$  et  $l = 3$  ?**
  
22. **Proposer une autre séquence d'ADN compatible avec le spectre  $\mathcal{SP} = \{TAC, ACC, ACG, CAC, CCG, CGT, CGC, GCA, GTA\}$  et différente de la séquence déterminée à la question 21.**

## Problème n° 2

Préambule : les différentes parties de ce problème peuvent être traitées indépendamment les unes des autres, bien que les définitions et notations données au fur et à mesure du sujet soient communes à toutes les parties concernées.

Nous nous intéressons dans ce problème aux systèmes de gestion de base de données (SGBD), c'est-à-dire aux logiciels qui permettent de stocker et manipuler des informations contenues dans les bases de données. Les SGBDs implémentent différents mécanismes pour assurer les bonnes propriétés des bases de données, notamment la persistance des données en cas de panne, l'atomicité et la cohérence des transactions ainsi que la protection des données en fonction des droits des utilisateurs.

### 1 Interrogation et manipulation de bases de données

Vous disposez d'un site de critiques gastronomiques dans lequel les utilisateurs peuvent évaluer des restaurants. Ce site s'appuie sur un modèle relationnel composé de trois relations.

Les restaurants sont stockés dans la relation *Restaurant*. Chaque restaurant est identifié par son *rID* et a un *nom*, un *type* (de cuisine) et une *adresse*.

La relation *Evaluateur* décrit les évaluateurs qui sont identifiés par leur *eID* et décrits par leur *pseudonyme* et leur date d'inscription (*dateInscription*).

Enfin, la relation *Evaluation* décrit les appréciations faites par les évaluateurs sur les restaurants. Chaque évaluation est identifiée par l'ensemble d'attributs *rID*, *eID*, *dateEval* où *rID* référence un restaurant dans la relation *Restaurant* et *eID* référence un évaluateur dans la relation *Evaluateur*. Une évaluation contient également une *note*.

Ceci donne le schéma relationnel suivant :

- *Restaurant(rID, nom, type, adresse)*
- *Evaluateur(eID, pseudonyme, dateInscription)*
- *Evaluation(rID#, eID#, dateEval, note)*

On suppose que les tables, associées à ce schéma relationnel, viennent d'être créées avec :

/\* Création de la table Restaurant \*/

```
CREATE TABLE Restaurant(
    rID integer,
    nom varchar(50),
    type varchar(30),
    adresse varchar(50));
```

```

/* Création de la table Evaluateur */
CREATE TABLE Evaluateur(
    eID integer,
    pseudonyme varchar(30),
    dateInscription date);
/* Création de la table Evaluation */
CREATE TABLE Evaluation(
    rID integer,
    eID integer,
    dateEval date,
    note integer);

```

**1. Implanter en SQL les contraintes et requêtes suivantes.**

- (a)  $rID$  est la clé primaire de la relation *Restaurant*.
- (b)  $eID$  est la clé primaire de la relation *Evaluateur*. Le couple (*pseudonyme*, *dateInscription*) est également clé.
- (c) Déterminer tous les restaurants qui ont été évalués par l'évaluateur dont l' $eID$  est 135.
- (d) On souhaite obtenir les paires d'évaluateurs qui ont noté le même restaurant. Retourner le pseudonyme de ces deux évaluateurs en éliminant les duplications (( $a, b$ ) et ( $b, a$ ) représentent la même paire d'évaluateurs).
- (e) Pour tous les cas où la même personne note deux fois le même restaurant et donne une note plus élevée la seconde fois, retourner le pseudonyme de l'évaluateur et le nom du restaurant.
- (f) Trouver le pseudonyme de tous les évaluateurs qui ont fait au moins 3 évaluations.
- (g) Les restaurants (respectivement les évaluateurs) de la relation *Evaluation* doivent être contenus dans la relation *Restaurants* (respectivement *Evaluateur*) :  
 $Evaluation[rID] \subseteq Restaurant[rID]$  (respectivement  $Evaluation[eID] \subseteq Evaluateur[eID]$ ).

**2. Donner une définition des propriétés d'atomicité, de cohérence et de persistance énoncées en introduction du sujet.**

## 2 Inférence de dépendances fonctionnelles

On s'intéresse dans cette partie aux problèmes d'anomalies lors de mises à jour d'une base de données. De tels événements étant difficilement détectables, l'une des solutions possibles consiste à réduire les redondances présentes dans les bases de données.

Pour ce faire, l'une des approches usuelles consiste à étudier les *dépendances fonctionnelles* entre les attributs de la base de données :

**Définition 1** (Dépendance fonctionnelle). Soient  $R$  une relation et  $X$  et  $Y$  deux sous-ensembles de l'ensemble des attributs de  $R$ . On dit que, dans une instance de relation  $r$  définie sur le schéma  $R$ , il existe une dépendance fonctionnelle entre  $X$  et  $Y$  (ou que  $X$  détermine fonctionnellement  $Y$ ) si et seulement si pour tous  $n$ -uplets  $t_1$  et  $t_2$  de  $r$ ,  $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$ .

Une dépendance fonctionnelle entre  $X$  et  $Y$  est notée  $R : X \rightarrow Y$  (ou simplement  $X \rightarrow Y$  lorsque le schéma est implicite). La projection d'un  $n$ -uplet  $t$  sur les attributs  $X$ , notée  $t[X]$ , retourne les valeurs du  $n$ -uplet  $t$  pour chaque attribut de  $X$ .

Intuitivement, une dépendance fonctionnelle entre deux ensembles d'attributs  $X$  et  $Y$  exprime le fait que si l'on connaît la valeur des attributs  $X$  alors on peut retrouver sans aucune ambiguïté les valeurs correspondantes sur les attributs  $Y$ . Plus simplement, pour une valeur de  $X$  il ne correspond qu'une seule valeur de  $Y$  possible.

L'intérêt de l'étude des dépendances fonctionnelles réside dans le fait qu'il est possible, à partir de quelques dépendances fonctionnelles explicites, de déterminer par inférence l'ensemble des dépendances fonctionnelles valides dans un schéma relationnel.

Dans le cas général, une règle d'inférence s'exprime sous la forme d'une relation entre les *prémisses* et la *conclusion* de la règle, représentée graphiquement par :

$$\text{Règle d'inférence : } \frac{\text{Formule}_1 \dots \text{Formule}_n}{\text{Conclusion}}$$

Si les prémisses  $\text{Formule}_1, \dots, \text{Formule}_n$  sont vraies, alors Conclusion est vraie.

Dans le contexte des dépendances fonctionnelles, on dispose des règles d'inférence suivantes ( $X, Y, Z, W$  sont des sous-ensembles de l'ensemble des attributs de  $R$ , et pour deux ensembles  $X$  et  $Y$ ,  $XY$  désigne par abus de notation l'union de  $X$  et  $Y$  ( $XY = X \cup Y$ )) :

$$\text{Réflexivité } (\sigma_{Re}) : \frac{Y \subseteq X}{X \rightarrow Y}$$

$$\text{Augmentation } (\sigma_A) : \frac{X \rightarrow Y}{WX \rightarrow WY}$$

$$\text{Transitivité } (\sigma_T) : \frac{X \rightarrow Y \quad Y \rightarrow Z}{X \rightarrow Z}$$

Ces trois règles d'inférences forment le système d'inférence d'Armstrong. Le but de cette partie est d'étudier et d'utiliser le système d'inférence d'Armstrong.

**3. Expliquer ce qu'est une anomalie de mises à jour.**

**4. Rappeler ce qu'est une clé dans une base de données. En quoi une clé détermine-t-elle fonctionnellement tous les autres attributs d'un schéma relationnel ?**

**5. Montrer que la règle d'inférence suivante est fausse.**

$$\frac{X \rightarrow Y \quad Y \rightarrow Z}{Z \rightarrow X}$$

On dit qu'un système d'inférence est correct s'il ne permet pas de produire des dépendances fonctionnelles non valides.

**6. Est-ce qu'un système d'inférence contenant la règle d'inférence de la question 5 est correct ?**

7. Démontrer que les règles du système d'Armstrong (Réflexivité, Augmentation et Transitivité) sont correctes en exploitant la notion de dépendance fonctionnelle. Conclure sur le système d'inférence d'Armstrong.
8. Soit le schéma de relation  $R(A, B, C, D, E, F)$  (**A, B, C, D, E et F** étant les attributs de la relation R). En utilisant le système d'inférence d'Armstrong  $\{\sigma_{Re}, \sigma_A, \sigma_T\}$ , montrer que :
- $CE \rightarrow E$ .
  - La dépendance fonctionnelle  $BD \rightarrow C$  peut être inférée à partir de l'ensemble de dépendances fonctionnelles  $\{AB \rightarrow BC, D \rightarrow A\}$ .
  - La dépendance fonctionnelle  $AB \rightarrow F$  peut être inférée à partir de l'ensemble de dépendances fonctionnelles  $\{AB \rightarrow C, A \rightarrow D, CD \rightarrow EF\}$ .

### 3 Fermeture transitive

On s'intéresse dans la suite de l'exercice à déterminer l'ensemble des dépendances fonctionnelles que l'on peut dériver à partir des règles d'inférences du système d'Armstrong. Pour cela, on va s'intéresser à la notion de fermeture transitive.

Soient  $X$  un ensemble d'attributs et  $\Sigma$  un ensemble de dépendances fonctionnelles sur le schéma de relation  $R$ .  $\Sigma \models X \rightarrow Y$  signifie que l'ensemble de dépendances fonctionnelles  $\Sigma$  implique la dépendance fonctionnelle  $X \rightarrow Y$ . La *fermeture transitive* de  $X$  par rapport à  $\Sigma$ , notée  $X_{\Sigma, R}^+$  (ou simplement  $X^+$  lorsque  $\Sigma$  et  $R$  sont évidents) est définie de la façon suivante :

$$X_{\Sigma, R}^+ = \{A \in R \text{ tq } \Sigma \models X \rightarrow A\}.$$

Intuitivement, la fermeture transitive d'un ensemble d'attributs  $X$  correspond à tous les attributs qui sont déterminés fonctionnellement par  $X$ .

9. Soit l'algorithme 1 qui calcule la fermeture transitive d'un ensemble d'attributs par rapport à un ensemble de dépendances fonctionnelles.

---

**Algorithme 1 : Fermeture( $\Sigma, R, X$ )**

---

**Entrées :**  $\Sigma$  un ensemble de dépendances fonctionnelles,  $X$  un ensemble d'attributs, le schéma  $R$

**Sortie :**  $X^+$  (la fermeture de  $X$  par  $\Sigma$ )

```

1 Cl :=  $X$ ;
2 done := false;
3 while ( $\neg$ done) do
4   done := true;
5   forall  $W \rightarrow Z \in \Sigma$  do
6     if  $W \subseteq Cl$  and  $Z \not\subseteq Cl$  then
7        $Cl$  :=  $Cl \cup Z$ ;
8       done := false;
9 return  $Cl$ 

```

---

**Etant donnés**  $\Sigma = \{A \rightarrow D; AB \rightarrow E; BF \rightarrow E; CD \rightarrow F; E \rightarrow C\}$  et le schéma de relation  $R(A, B, C, D, E, F)$  (**A, B, C, D, E et F** étant les attributs de la relation R), que retourne l'algorithme pour les trois cas suivants ? :

- (a)  $X = F$
- (b)  $X = BF$
- (c)  $X = ABF$

**Que peut-on dire de l'ensemble d'attributs ABF ?**

10. Soit le lemme suivant :  $\Sigma \models X \rightarrow Y$  si et seulement si  $Y \subseteq X^+$ .

On considère l'ensemble  $\Sigma$  de dépendances fonctionnelles suivant défini sur le schéma de relation  $R(A, B, C, D, E, F, G)$  :

$$\Sigma = \{A \rightarrow BCD; ABE \rightarrow G; CD \rightarrow E; EG \rightarrow ABD; BE \rightarrow F; FG \rightarrow A\}.$$

- (a) **Montrer que**  $\Sigma \models A \rightarrow F$ .
- (b) **Montrer que**  $BEF$  n'est pas une clé de R.

On s'intéresse maintenant aux propriétés algébriques de la fermeture. En algèbre, on appelle *fermeture* une application  $\phi : \wp(E) \rightarrow \wp(E)$  où  $\wp(E)$  désigne l'ensemble des parties de  $E$ , qui est :

**Extensive** :  $X \subseteq \phi(X)$

**Croissante** :  $X \subseteq Y \Rightarrow \phi(X) \subseteq \phi(Y)$

**Idempotente** :  $\phi(\phi(X)) = \phi(X)$

11. Montrer que la fermeture  $X^+$  est bien une fermeture au sens algébrique du terme.

## 4 Equivalence de systèmes d'inférence

On dit qu'un système d'inférence est complet si toutes les dépendances valides peuvent être inférées par ce système. Le système d'inférence d'Armstrong est complet.

L'objectif de cette partie est de trouver un système d'inférence correct et complet de plus petite taille que le système d'Armstrong.

Considérons la nouvelle règle d'inférence suivante :

$$\text{Pseudo-transitivité } (\sigma_P) : \frac{X \rightarrow Y \quad WY \rightarrow Z}{WX \rightarrow Z}$$

12. Soit  $\Sigma$  un ensemble de dépendances fonctionnelles, montrer que toute preuve de  $\Sigma \models X \rightarrow Y$  utilisant la règle  $\sigma_P$  peut être transformée en une preuve n'utilisant que  $\sigma_A$  et  $\sigma_T$ .

13. Montrer que toute preuve de  $\Sigma \models X \rightarrow Y$  utilisant les règles  $\sigma_{Re}$ ,  $\sigma_A$  et  $\sigma_T$  peut être transformée en une preuve n'utilisant que  $\sigma_{Re}$  et  $\sigma_P$ .

14. En déduire que le système  $\{\sigma_{Re}, \sigma_P\}$  est correct et complet pour l'inférence des dépendances fonctionnelles.

**Préambule :** le sujet comporte 5 parties. L’ensemble du sujet porte sur une même thématique qui est la recherche d’une information satisfaisant certaines contraintes. Cette thématique est illustrée sur l’exemple du Mastermind, en parties II., III. et IV. Les notations sont communes à toutes les parties et sont introduites au fur et à mesure du sujet. Les parties peuvent être traitées de façon indépendante, sauf pour la partie III. où on pourra réutiliser ou admettre les notations et résultats de la partie II., et pour la partie IV. où on pourra réutiliser les notions, contenus, algorithmes et fonctions Python déterminés dans les parties II. et III.

Dans ce sujet, certains contenus sont à destination des élèves de lycée. Ces contenus sont en italique dans le sujet.

Pour certaines questions, une longueur de réponse attendue est indiquée. Nous vous invitons à respecter ces indications.

## Partie I. Le trésor et la fausse pièce

Vous avez proposé à vos classes de première en spécialité NSI, le projet ci-dessous.

*Vous êtes un ou une pirate qui vient de récupérer sa part du trésor constitué d'un beau tas de pièces d'or. Hélas, vous avez vent d'une supercherie : l'une des pièces n'est pas en or massif, mais en un alliage plus léger que l'or et est simplement plaquée or, pour qu'on ne puisse pas la distinguer visuellement des autres pièces. Le seul signe distinctif est qu'elle est plus légère que les autres. Vous n'avez pas à votre disposition un atelier de chimiste ni un laboratoire scientifique vous permettant d'étudier très précisément ces pièces, mais simplement une balance à plateaux sans aucun poids, comme celle donnée ci-dessous :*



*Pouvez-vous, avec cette simple balance, retrouver la fausse pièce ?*

I.1. L'une de vos classes a été beaucoup plus rapide que les autres et les élèves ont très rapidement proposé une réponse utilisant une approche par dichotomie. Pour alimenter leur curiosité et leur créativité, vous voulez les amener à une solution, dite *par trichotomie*, utilisant un découpage en 3 ensembles (contrairement à 2 ensembles utilisés avec l'approche dichotomique). Afin de les aider, vous préparez un énoncé donné ci-dessous (partie en italique) et que vous comptez leur distribuer. **Rédiger un corrigé de cet énoncé, tel que vous le proposeriez à vos élèves.**

*Exemples pour un petit nombre de pièces dans le trésor.*

1. *Expliquer comment procéder avec 2 pièces pour retrouver la fausse pièce.*
2. *Expliquer comment procéder en une seule pesée quand il y a 3 pièces.*
3. *Expliquer comment conclure en deux pesées quand il y a 4 pièces.*

*Examinons maintenant comment faire avec 9 pièces. On procède en divisant les 9 pièces en 3 tas, et selon la réponse de la balance on identifie le tas contenant la pièce plus légère, comme indiqué sur la figure 1.*

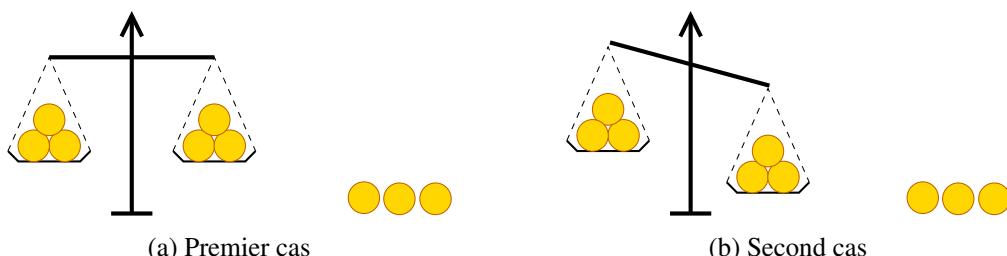


FIGURE 1 – Exemple avec 9 pièces.

*Dans le cas illustré en figure 1a, le tas contenant la pièce la plus légère est le tas qui n'est pas sur un plateau. On identifie, en une pesée supplémentaire, la pièce la plus légère parmi les 3 pièces de ce tas.*

*Dans le cas illustré en figure 1b, le tas contenant la pièce la plus légère est le tas qui est sur le plateau de gauche. On identifie, en une pesée supplémentaire, la pièce la plus légère parmi les 3 pièces de ce tas.*

*On peut donc conclure en deux pesées. Examinons maintenant le cas général d'un trésor de  $n$  pièces.*

Algorithme dans le cas général d'un trésor de  $n$  pièces ( $n$  est une donnée d'entrée de l'algorithme).

4. *Dans le cas où  $n$  est une puissance de 3 ( $n = 3^k$ ,  $k$  entier naturel) : écrire un algorithme itératif qui permet d'identifier, parmi  $n$  pièces, la pièce fausse qui est plus légère que les autres.*
  5. *Expliquer comment votre algorithme fonctionne avec 27 pièces.*
- Aide : on devrait obtenir le résultat en 3 pesées.
6. *Dans le cas d'un nombre quelconque de pièces, adapter l'algorithme précédent pour identifier la fausse pièce.*
  7. *Afin de comparer avec la recherche dichotomique, remplir le tableau suivant, en indiquant, pour chaque méthode et pour différentes valeurs de  $n$ , le nombre maximal de pesées.*

| <b><i>n</i></b> | <b><i>Dichotomie</i></b> | <b><i>Trichotomie</i></b> |
|-----------------|--------------------------|---------------------------|
| 2               |                          |                           |
| 3               |                          |                           |
| 6               |                          |                           |
| 8               |                          |                           |
| 9               |                          |                           |
| 16              |                          |                           |
| 27              |                          |                           |

*Quelle est, d'après vous, la méthode la plus efficace en nombre de pesées ?*

- I.2. La question 6 est difficile pour les élèves. En particulier, le terme *adapter* n'est pas suffisamment précis. **Proposer, en les justifiant, une ou plusieurs aides que vous pourriez proposer aux élèves afin de leur permettre de répondre à cette question.**  
(Répondre en 6 lignes maximum.)
- I.3. **En transposant les notions de variants et d'invariants, vues pour l'algorithme de recherche dichotomique dans un tableau trié en classe de 1<sup>ère</sup> NSI, donner à vos élèves un schéma de la preuve de correction de l'algorithme par trichotomie.**  
(Répondre en 10 lignes maximum.)

## Partie II. Mastermind : le joueur humain devine

Vous proposez à vos élèves de Terminale, en spécialité NSI, un projet qui réalise le jeu du Mastermind. Les règles du Mastermind sont fournies en annexe 1. Dans ce projet, les élèves travailleront sur deux situations :

- le joueur (humain) tente de découvrir une configuration de pions cachée par la machine (le joueur est alors appelé le décodeur et la machine le codificateur) ;
- le joueur (humain) cache une configuration de pions à la machine qui doit la découvrir (le joueur est alors appelé le codificateur et la machine le décodeur).

Dans cette partie II., on considère que le joueur est le décodeur et que la machine est le codificateur.

Dans la suite, on notera  $N$  le nombre de couleurs et  $P$  le nombre de pions cachés. On appellera *configuration* un ensemble de  $P$  pions dont les positions et les couleurs sont déterminées.  $C$  est le nombre maximal de configurations que le décodeur peut proposer.

Vous préparez un énoncé à distribuer aux élèves qui a pour but de les guider dans la réalisation de ce projet.

**II.1.** Parmi les structures de données de type liste, p-uplet et dictionnaire, vous choisissez la structure de données de liste pour représenter une configuration. **Donner les arguments qui seront présentés aux élèves pour justifier ce choix.**

**II.2.** **Donner la correction de l'énoncé, donné ci-dessous (partie en italique), qui sera proposé aux élèves.** Pour la question 2.d de l'énoncé, seules les lignes à compléter peuvent être données sur votre copie avec les numéros de lignes associés.

*1. La configuration choisie par le codificateur est générée aléatoirement.*

Écrire une fonction Python qui renvoie une configuration aléatoire selon une loi uniforme sur l'ensemble des configurations. L'en-tête de cette fonction sera le suivant :

**def genere\_configuration(n\_couleurs,n\_positions)**

Vous pourrez utiliser la fonction `randint` de la bibliothèque `random` donnée en annexe 2 pour choisir aléatoirement une couleur, qui est représentée par un entier entre 1 et  $N$ .

Il est possible d'écrire ce programme de façon itérative et de façon récursive. Vous écrirez les deux versions.

*2. Programmation, en Python, du jeu Mastermind quand le décodeur est le joueur humain*

(a) Écrire une fonction Python itérative calculant et retournant le nombre de pions bien placés entre deux configurations, c'est-à-dire les pions de même couleur et de même position entre les deux configurations (nombre de fiches noires). On l'appellera `n_bien_places` et son en-tête est :

**def n\_bien\_places(configuration\_1,configuration\_2)**

- (b) Donner un algorithme qui calcule le nombre de pions communs entre deux configurations (l'ordre des pions dans la configuration n'a pas d'importance). Si nécessaire, vous pourrez utiliser une structure de données auxiliaire en expliquant votre choix. La fonction Python qui implante cet algorithme sera appelé `n_pions_communns` et son en-tête est :

```
def n_pions_communns(configuration_1,configuration_2)
```

L'implantation de cette fonction n'est pas demandée.

- (c) En déduire une fonction Python renvoyant le nombre de fiches blanches et de fiches noires qui sont déterminées lorsque deux configurations sont comparées. Pour rappel, le nombre de fiches noires correspond au nombre de pions étant bien placés (et donc de la même couleur) et le nombre de fiches blanches correspond au nombre de pions de la bonne couleur qui ne sont pas bien placés. On appellera la fonction `n_noirs_blancs` et son en-tête est :

```
def n_noirs_blancs(configuration_1,configuration_2)
```

- (d) Compléter le squelette de la fonction Python interactive, donnée ci-dessous, permettant d'effectuer une partie de Mastermind dans laquelle le codificateur est la machine et le décodeur est le joueur humain. Cette fonction comprend la génération d'une configuration à faire deviner, la saisie des propositions successives de configurations par l'utilisateur et les réponses correspondantes de la machine via les fiches blanches et noires. Les "..." données dans le squelette sont à compléter et peuvent représenter des variables, des expressions, des instructions ou des suites d'instructions.

---

```
1 def joueur_devine(n_couleurs=6,n_positions=4,n_essais=10):
2
3     configuration_cachée =
4         genere_configuration(n_couleurs,n_positions)
5     print("Taille de la configuration cachée: ",n_positions)
6
7     i = ...      # nombre d'essais déjà réalisés
8     trouve = ... # variable booléenne qui indique si la
9                 # configuration a été trouvée
10
11    while ... :
12        ...
13        m = input("Essai "+str(i)+": ")
14        # le joueur entre une configuration
15        ... # conversion de la configuration entrée
16        # dans le bon type
17        print("    "+str(...)+" bien placés - "
18              +str(...)+" mal placés")
19        trouve = ...
20
21    if trouve :
22        return "Bravo vous avez trouvé en "
23                + str(...) + " essais"
24    else :
25        return "Perdu vous avez épuisé vos " + str(...)
26                + " tentatives, bonne réponse = "
27                + str(configuration_cachée)
```

---

II.3. L'année précédente, vous aviez aussi proposé, en TP, à une classe de terminale, de programmer le jeu du Mastermind avec 5 couleurs, 5 positions et un nombre maximal de 10 essais pour le décodeur, mais vous ne leur aviez pas fourni l'énoncé précédent que vous avez préparé cette année. Un groupe d'élèves vous a rendu la fonction donnée en annexe 4. **Analyser cette production, en se référant aux numéros de ligne si nécessaire.**  
*(Répondre en 20 lignes maximum.)*

II.4. **Proposer une remédiation pour ce groupe d'élèves sur les points qui vous paraissent importants.**

*(Répondre en 10 lignes maximum.)*

## Partie III. Mastermind : la machine devine

Dans cette partie, il s'agit d'étudier le cas où le joueur (humain) joue le rôle du codificateur et la machine celui du décodeur.

Vous réfléchissez aux différentes questions et activités que vous aimerez aborder avec vos élèves. Répondez aux questions suivantes.

III.1. Étude de l'ensemble de toutes les configurations

III.1.a. **Donner la formule du nombre de configurations possibles en fonction des paramètres  $N$  et  $P$  du jeu.**

III.1.b. **Avec les valeurs habituelles pour le jeu, qui sont  $N = 6$  et  $P = 4$  pour les plus jeunes joueurs et joueuses, et dans la version classique  $N = 8$  et  $P = 5$ , combien y a-t-il de configurations possibles ?** Vous pourrez vous aider de la table de calculs donnée en annexe 3.

III.1.c. **Est-il envisageable et raisonnable de créer et de mémoriser toutes les configurations dans ces deux cas de figures ?**

**Quelle comparaison pourriez-vous proposer à vos élèves pour qu'ils puissent arriver à la même conclusion que vous ?** Vous pourrez supposer qu'un entier est codé sur 64 bits.

III.2. Vous avez trouvé sur le Web un projet, traitant du Mastermind, déjà réalisé. Pour savoir si vous pouvez vous en inspirer, vous l'analysez. **Que fait la fonction Python suivante ? Prouver qu'elle n'omet aucune configuration.**

```
def mystere(n_couleurs = 6,n_positions = 4) :
    if ( n_positions == 0) :
        return []
    else :
        l = []
        for element in mystere(n_couleurs,n_positions-1):
            for couleur in range(1,n_couleurs+1) :
                l.append(element + [couleur])
    return l
```

Comme cela a été fait dans ce projet, vous décidez vous aussi d'utiliser une liste pour stocker toutes les configurations possibles.

- III.3. Donner un plan de cours, de niveau NSI Terminale, sur les structures de données linéaires, comme les listes, piles et files, en explicitant les exemples utilisés comme illustrations.** Un extrait du programme concernant ce sujet est donné dans la table 1.  
*(Répondre en 10 lignes maximum.)*

| Contenu                                     | Capacités attendues   | Commentaires   |
|---|---|--|
| Listes, piles, files : structures linéaires | Distinguer des structures par le jeu des méthodes qui les caractérisent. Choisir une structure de données adaptée à la situation à modéliser. | On distingue les modes FIFO (first in first out) et LIFO (last in first out) des piles et des files.<br>Les listes n'existent pas de façon native en Python. |

TABLE 1 – Extrait du programme de NSI Terminale - Structures de données.

- III.4.** Vous réfléchissez aux différents éléments à donner et aux programmes à demander aux élèves lorsque la machine tente de deviner la configuration choisie par le codificateur.
- III.4.a.** Vous supposez que la machine a proposé une configuration, nommée `tentative`, et a reçu du codificateur, en retour, le nombre de fiches noires, noté `n_bien_places`, et le nombre de fiches blanches, noté `n_mal_placés`.  
 Vous comptez demander à vos élèves un algorithme qui commence avec l'ensemble des configurations qui étaient encore candidates avant la proposition de la configuration `tentative` (ensemble qui sera appelé `configurations_possibles`) et qui le modifie pour qu'il ne contienne plus que l'ensemble des configurations qui pourront être candidates au tour suivant, une fois les fiches noires et blanches reçues du codificateur suite à la proposition de `tentative`. **Écrire cet algorithme pour pouvoir donner un corrigé à vos élèves.** La fonction Python correspondante s'appellera `reduction_configurations_possibles`. Il ne vous est pas demandé d'écrire cette fonction, mais seulement de **donner son en-tête avec ses paramètres d'entrée**.
- III.4.b.** **Quel(s) argument(s) donneriez-vous à vos élèves pour qu'ils choisissent la liste, parmi les structures linéaires de type liste, pile et file, afin de stocker l'ensemble des configurations possibles ?**
- III.4.c.** **Quel argument donneriez-vous à vos élèves pour qu'ils comprennent que n'importe quelle configuration peut être choisie dans l'ensemble `configurations_possibles` pour le tour suivant ?**
- III.4.d.** **Écrire le corrigé de la fonction Python interactive, demandée à vos élèves, permettant d'effectuer une partie où c'est la machine qui devine la configuration choisie par le joueur. Ce programme comprend la génération de toutes les configurations possibles par la machine, les propositions successives de la machine et les réponses correspondantes du joueur.**  
**Voici son en-tête :**
- ```
def machine_devine(n_couleurs=6,n_positions=4,n_essais=10)
```

## Partie IV. Mastermind : évaluation et analyse d'un projet

Vous donnez le projet de Mastermind à vos élèves. Vous les guidez avec les différentes étapes que vous avez établies dans les parties II. et III. Vous avez réservé 20h pour travailler sur ce projet avec vos élèves. À la fin du projet, les élèves devront rendre un programme qui permet d'effectuer une partie, avec le joueur (humain) qui peut être codificateur ou décodeur.

**IV.1. Proposer un barème qui vous permettra d'évaluer le travail rendu par les élèves.**

**Expliquer les différents éléments que vous comptez évaluer ainsi que le nombre de points attribués pour chacun de ces éléments.**

*(Répondre en 15 lignes maximum.)*

Une élève vous a rendu un projet dont un extrait figure en annexe 5. Elle est allée beaucoup plus loin que ce qui était attendu sur ce projet. Notamment, elle a fait jouer la machine contre elle-même et elle a estimé expérimentalement le nombre moyen d'essais effectués par la machine pour trouver la configuration cachée. Pour cela, la machine a généré des configurations aléatoires avec un nombre de couleurs  $N$  et une taille  $P$  fixés et l'élève a compté le nombre d'essais effectués par la machine pour trouver chaque configuration. Elle a limité à 10 le nombre maximal d'essais.

**IV.2. Que pensez-vous de son approche qui consiste à baser son estimation du nombre moyen d'essais effectués sur plusieurs configurations aléatoires ? Justifier votre réponse.**

**IV.3. Que pensez-vous de sa remarque : "... plus le nombre de boules et le nombre de couleurs sont élevés, plus le nombre de simulations est bas..." ? Justifier votre réponse.**

**IV.4. Dans la version pour jeunes joueurs du jeu,  $N = 6$  et  $P = 4$ , pensez-vous qu'il est raisonnable de fixer la limite du nombre d'essais à 10 ? Justifier votre réponse à partir des valeurs obtenues par l'élève sur le nombre moyen d'essais.**

**IV.5. Quel indicateur l'élève aurait-elle pu mesurer expérimentalement pour affiner la limite sur le nombre d'essais raisonnables à fixer ?**

**IV.6. Qu'est-ce que l'élève aurait pu mesurer pour analyser, expérimentalement, le coût de son algorithme, en plus du nombre d'essais effectués ?**

**IV.7. Vous aidez l'élève à réfléchir à la complexité du problème de recherche d'une configuration cachée. Pour cela, vous lui décrivez un algorithme qui trouve la configuration cachée en  $N + \frac{P(P-1)}{2}$  essais pour  $N$  couleurs et  $P$  positions avec  $N > P$ . Décrire cet algorithme. Quel est le nombre d'essais proposés par cet algorithme dans le cas du jeu pour jeunes joueurs et joueuses :  $N = 6$  et  $P = 4$  ? Et dans le cas classique  $N = 8$  et  $P = 5$  ?**

**IV.8. Que devrait en conclure l'élève sur le problème de recherche d'une configuration cachée ? Justifier votre réponse.**

**IV.9. En vous inspirant de l'idée de la partie I., montrer que le problème de recherche pour  $N$  couleurs et  $P$  positions ne peut pas être résolu, pour le pire cas, en moins de  $\log_b N^P$  où  $b$  est une base que l'on calculera.**

Indication : calculer le nombre de réponses différentes possibles, en termes de fiches noires et blanches, pour un essai.

## Partie V. Enjeu sociétal

Vous êtes sensible à la protection des données personnelles, sujet qui peut être abordé, entre autres, dans les thématiques Web et Réseaux sociaux du programme SNT. Vous avez l'intention de consacrer 1h30, en classe de Seconde, lors de l'enseignement de SNT, sur les questions d'anonymat et de ré-identification et plus généralement sur les risques liés aux données individuelles que l'on laisse sur le Web et en particulier sur les réseaux sociaux. Des extraits du programme de SNT sur ces thématiques sont donnés en annexe 6. Vous avez aussi trouvé un article, donné en annexe 6, qui traite du problème de la ré-identification et que vous allez exploiter pour votre cours.

**V.1. En quoi le problème de l'anonymat et de la ré-identification est-il similaire au principe du Mastermind, projet que vous avez travaillé avec vos élèves de Terminale ? Expliquer comment vous pourriez utiliser le Mastermind pour illustrer les principes d'anonymat et de ré-identification à vos élèves de Seconde.**

**V.2. Donner 3 illustrations que vous pourriez présenter aux élèves afin qu'ils prennent conscience de toutes les informations personnelles qu'ils peuvent être amenés à donner et qui peuvent permettre leur ré-identification.**

**V.3. Citer 2 contextes où le partage de données anonymisées est d'intérêt public.**

**V.4. Citer 2 situations où la ré-identification est problématique.**

**V.5. Citer 3 mini-activités sur machine qui pourraient aider les élèves à comprendre quelles informations personnelles sont stockées et comment ils peuvent essayer de réguler les traces qu'ils laissent dans leurs usages quotidiens.**

**V.6. Donner le plan de cours d'1h30 que vous projetez de faire sur ce sujet. Vous décrivez les différentes notions que vous comptez aborder et les illustrations associées le cas échéant, ainsi que les activités que vous comptez réaliser avec les élèves et l'ordre dans lequel vous comptez aborder ces notions et ces activités.**

*(Répondre en 20 lignes maximum.)*

**V.7. L'évaluation des acquis sur ce cours se fera par QCM. Proposer deux questions, avec 4 choix chacune, qui testeront, pour l'une, les connaissances acquises et pour l'autre, le savoir-faire (comme le raisonnement et la mise en œuvre par exemple). Justifier le choix des questions. Pour chaque question, justifier le choix des 4 réponses proposées. Vous indiquerez aussi si les réponses sont justes ou fausses.**

## Annexe 1 : Règles du Mastermind

Les règles du Mastermind présentées ci-dessous sont inspirées de Wikipedia (<https://fr.wikipedia.org/wiki/Mastermind> consulté le 24 octobre 2019). Le Mastermind, ou Master Mind, est un jeu de société pour deux joueurs dont le but est de trouver un code. C'est un jeu de réflexion, et de déduction, inventé par Mordecai Meirowitz dans les années 1970.

Le jeu d'origine se joue avec un joueur appelé codificateur et un joueur appelé décodeur.

**Déroulement du jeu :** le codificateur dispose de pions de  $N$  couleurs différentes. Il commence par placer  $P$  pions dont il a choisi les couleurs sans qu'ils soient vus de l'autre joueur à l'arrière d'un cache qui les masquera à la vue de celui-ci jusqu'à la fin de la partie. Il doit prendre soin de ne pas révéler la configuration qu'il a choisie, c'est-à-dire la couleur et la position des pions. Rien ne l'empêche de choisir plusieurs pions d'une même couleur dans une même configuration.

Le décodeur doit trouver quels sont les  $P$  pions choisis par le codificateur, c'est-à-dire leurs couleur et position. Le décodeur propose une configuration : il place  $P$  pions dans les trous de la première rangée la plus proche de lui, cf. Figure 2a.

Une fois les pions placés, le codificateur indique :

- le nombre de pions de la bonne couleur bien placés en utilisant le même nombre de fiches noires à côté de la rangée proposée par le décodeur ;
- puis le nombre de pions de la bonne couleur, mais mal placés, en insérant le même nombre de fiches blanches à côté des fiches noires.

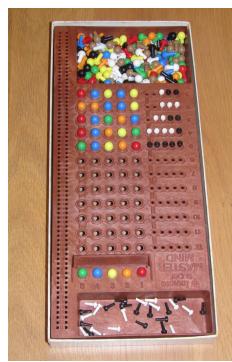
S'il n'y a aucune correspondance, le codificateur ne met aucune fiche.

Une fois que le codificateur a indiqué le nombre de fiches noires et le nombre de fiches blanches associés à la configuration proposée par le décodeur, ce dernier propose une nouvelle configuration. Le processus se répète jusqu'à ce que le décodeur ait trouvé la bonne configuration ou jusqu'à ce qu'il ait proposé  $C$  configurations qui ne correspondent pas à la configuration choisie par le codificateur.

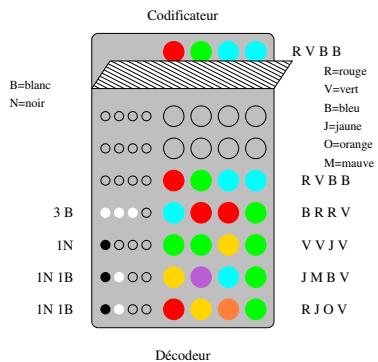
La figure 2 donne des exemples de déroulement de parties de Mastermind. Le Mastermind classique fonctionne avec 8 couleurs et 5 positions et la configuration doit être trouvée en au plus 12 coups.



(a) Vue du jeu côté décodeur



(b) Vue du jeu côté codificateur



(c) Schéma du jeu

FIGURE 2 – Exemples de Mastermind.

## Annexe 2 : Bibliothèque random

### Bibliothèque random de Python

Voici quelques explications pour utiliser la fonction `randint` de la bibliothèque `random` de Python.

#### Tirer aléatoirement un nombre entier

Pour tirer aléatoirement un nombre entier selon une loi uniforme, on peut utiliser la fonction `randint`. Cette fonction prend deux paramètres en entrée : une valeur minimale et une valeur maximale.

Voici un exemple pour tirer des entiers entre 1 et 5. Le premier paramètre doit être inférieur au second.

---

```
import random
print(random.randint(1, 5))
```

---

Le résultat sera soit 1, soit 2, soit 3, soit 4 ou 5.

## Annexe 3 : Table de calculs

La table suivante donne la valeur de  $x^y$  pour différentes valeurs de  $x$  et de  $y$ .

|         | $y = 2$ | $y = 3$ | $y = 4$ | $y = 5$ | $y = 6$ | $y = 7$ | $y = 8$  | $y = 9$   |
|---------|---------|---------|---------|---------|---------|---------|----------|-----------|
| $x = 2$ | 4       | 8       | 16      | 32      | 64      | 128     | 256      | 512       |
| $x = 3$ | 9       | 27      | 81      | 243     | 729     | 2187    | 6561     | 19683     |
| $x = 4$ | 16      | 64      | 256     | 1024    | 4096    | 16384   | 65536    | 262144    |
| $x = 5$ | 25      | 125     | 625     | 3125    | 15625   | 78125   | 390625   | 1953125   |
| $x = 6$ | 36      | 216     | 1296    | 7776    | 46656   | 279936  | 1679616  | 10077696  |
| $x = 7$ | 49      | 343     | 2401    | 16807   | 117649  | 823543  | 5764801  | 40353607  |
| $x = 8$ | 64      | 512     | 4096    | 32768   | 262144  | 2097152 | 16777216 | 134217728 |

## Annexe 4 : Production d'élèves

### Production d'élèves – partie II., question II.3

```

1 def verification():
2     # definit le nombre de couleurs justes
3     # et de couleurs qui ne sont pas dans la combinaison à trouver
4     bon=0
5     mauvais=0
6     tour=tour+1
7     if a==aa:
8         bon=bon+1
9     if aa!=a and aa!=b and aa!=c and aa!=d and aa!=e:
10        mauvais=mauvais+1
11    if b==bb:
12        bon=bon+1
13    if bb!=b and bb!=a and bb!=c and bb!=d and bb!=e:
14        mauvais=mauvais+1
15    if c==cc:
16        bon=bon+1
17    if cc!=c and cc!=b and cc!=a and cc!=d and cc!=e:
18        mauvais=mauvais+1
19    if d==dd:
20        bon=bon+1
21    if dd!=d and dd!=b and dd!=c and dd!=a and dd!=e:
22        mauvais=mauvais+1
23    if e==ee:
24        bon=bon+1
25    if ee!=e and ee!=b and ee!=c and ee!=d and ee!=a:
26        mauvais=mauvais+1
27    if a==aa and b==bb and c==cc and d==dd and e==ee:
28        # Si la combinaison est la même que celle à trouver,
29        # ouvre une fenêtre de victoire
30        fen_gagne=Toplevel()
31        fen_gagne.title("Mastermind")
32        text=Label(fen_gagne,text="Vous avez gagné !
33                                    Félicitations !")
34        text.pack()
35        global fen_gagne
36    if tour==10 :
37        # Si au 10eme tour, la combinaison n'a pas encore été
38        # trouvée, défaite
39        if aa!=a or bb!=b or cc!=c or dd!=d or ee!=e :
40            fen_perdu=Toplevel()
41            fen_perdu.title("Mastermind")
42            text=Label(fen_perdu,text=
43                        "C'est la 10e tentative sans succès,
44                        vous avez donc perdu la partie")
45            text.pack()
46            global fen_perdu
47    bon=str(bon)
48    mauvais=str(mauvais)
49    global tour,bon,mauvais

```

## Annexe 5 : Analyse d'un projet d'élève sur le Mastermind

### Analyse d'un projet d'élève sur le Mastermind – partie IV.

Extraits d'un projet d'élève sur le Mastermind (accessible à <https://issuu.com/wile/docs/mastermind>).

**Note :** dans cette production, la « version classique » correspond à la version pour les plus jeunes joueurs du Mastermind, avec 4 positions et 6 couleurs et une boule correspond à un pion.

#### 4. Analyse des résultats

##### 4.1. Le nombre de coups pour gagner

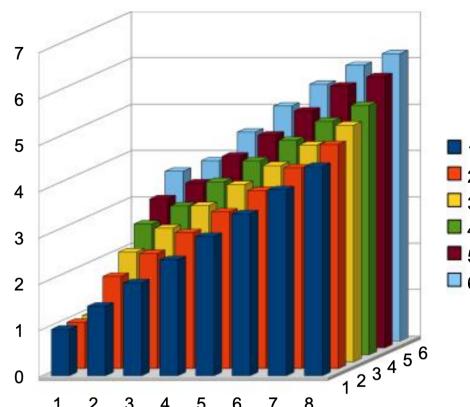
Grâce aux nombreuses simulations, j'ai pu découvrir le nombre de coups qu'il faut en moyenne pour découvrir la combinaison cachée en fonction du nombre de couleurs et du nombre de boules. J'ai effectué de 100 à 10'000 simulations par cas, cela dépend de la complexité de l'opération. Ainsi, plus le nombre de boules et le nombre de couleurs sont élevés, plus le nombre de simulations est bas, car sinon, cela prendrait trop de temps. Les valeurs sont donc moins précises pour ces cas.

*Moyenne du nombre de coups en fonction du nombre de boules et de couleurs*

|            | 1 boule | 2 boules | 3 boules | 4 boules | 5 boules | 6 boules |
|------------|---------|----------|----------|----------|----------|----------|
| 1 couleur  | 1.000   | 1.000    | 1.000    | 1.000    | 1.000    | 1.000    |
| 2 couleurs | 1.503   | 1.996    | 2.373    | 2.822    | 3.216    | 3.663    |
| 3 couleurs | 2.006   | 2.485    | 2.884    | 3.214    | 3.562    | 3.893    |
| 4 couleurs | 2.499   | 2.938    | 3.371    | 3.740    | 4.135    | 4.507    |
| 5 couleurs | 3.008   | 3.387    | 3.825    | 4.192    | 4.596    | 5.070    |
| 6 couleurs | 3.495   | 3.844    | 4.238    | 4.631    | 5.117    | 5.540    |
| 7 couleurs | 4.014   | 4.338    | 4.680    | 5.043    | 5.650    | 5.950    |
| 8 couleurs | 4.513   | 4.841    | 5.109    | 5.388    | 5.850    | 6.200    |

Rouge : Mastermind classique

*Graphique correspondant au tableau ci-dessus*



axe Z (verticalement) : nombre de coups

axe Y (en profondeur) : nombre de boules

axe X (horizontalement de gauche à droite) : nombre de couleurs

## Annexe 6

### Documents pour la partie V.

Cette annexe comprend un extrait du programme de SNT en Seconde concernant les thématiques du Web et des réseaux sociaux, ainsi qu'un article tiré du site The Conversation.

#### Extrait du programme de SNT en Seconde concernant la thématique du Web.

##### **Impacts sur les pratiques humaines**

Dans l'histoire de la communication, le Web est une révolution : il a ouvert à tous la possibilité et le droit de publier ; il permet une coopération d'une nature nouvelle entre individus et entre organisations : commerce en ligne, création et distribution de logiciels libres multi-auteurs, création d'encyclopédies mises à jour en permanence, etc. ; il devient universel pour communiquer avec les objets connectés.

Le Web permet aussi de diffuser toutes sortes d'informations dont ni la qualité, ni la pertinence, ni la véracité ne sont garanties et dont la vérification des sources n'est pas toujours facile. Il conserve des informations, parfois personnelles, accessibles partout sur de longues durées sans qu'il soit facile de les effacer, ce qui pose la question du droit à l'oubli. Il permet une exploitation de ses données, dont les conséquences sociétales sont encore difficiles à estimer : recommandation à des fins commerciales, bulles informationnelles, etc. En particulier, des moteurs de recherche permettent à certains sites d'acquérir de la visibilité sur la première page des résultats de recherche en achetant de la publicité qui apparaîtra parmi les liens promotionnels.

#### Extrait du programme de SNT en Seconde concernant la thématique des réseaux sociaux.

##### **Les données et l'information**

Les différents réseaux sociaux permettent l'échange d'informations de natures différentes : textes, photos, vidéos. Certains limitent strictement la taille des informations, d'autres autorisent la publication, mais de façon limitée dans le temps. Certains permettent l'adjonction d'applications tierces (plug-ins) qui peuvent ajouter des fonctionnalités supplémentaires.

Toutes les applications de réseautage social utilisent d'importantes bases de données qui gèrent leurs utilisateurs, l'ensemble des données qu'ils partagent, mais aussi celles qu'ils consentent à fournir (sans toujours le savoir), y compris sur leur vie personnelle.

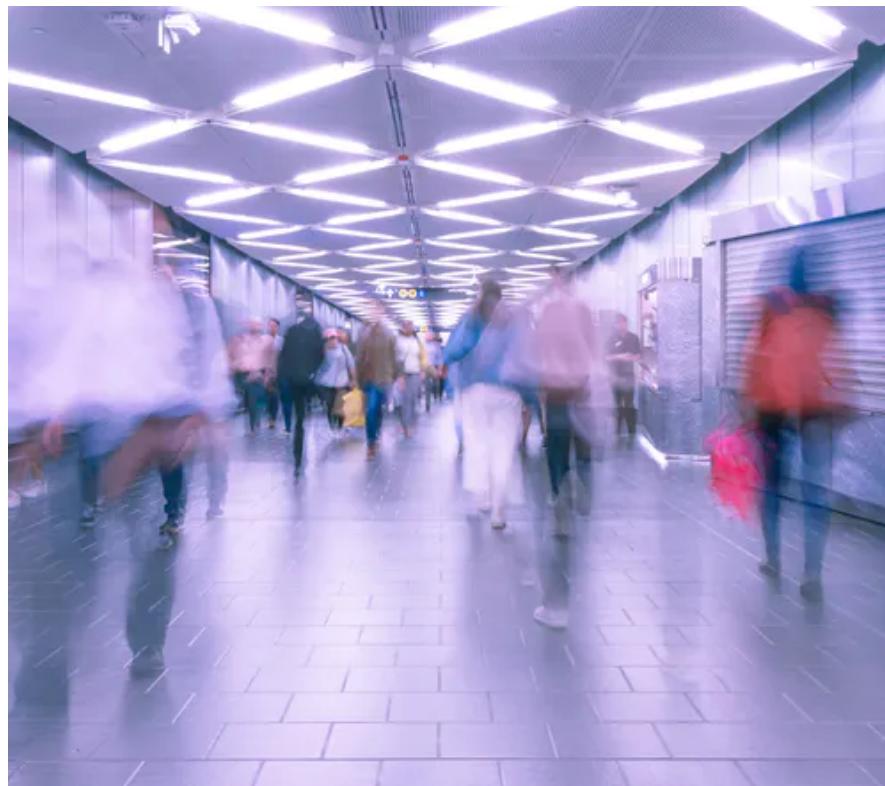
#### Article tiré du site The Conversation

Seules 3 pages de cet article sont données (la dernière page n'a pas été reproduite, elle contenait des suggestions de lecture et ne concernait pas le sujet que vous devez traiter).

Cet article comporte une erreur de frappe : il faut lire **font** à la place de **sont** dans la phrase *En réponse, les compagnies et organismes qui les collectent affirment souvent qu'elles le sont de manière « anonyme ».*

## THE CONVERSATION

L'expertise universitaire, l'exigence journalistique



Dans le métro. Photo by Martin Adams on Unsplash

## Données anonymes... bien trop faciles à identifier

17 septembre 2019, 21:01 CEST

Téléphones, ordinateurs, cartes de crédit, dossiers médicaux, montres connectées, ou encore assistants virtuels : chaque instant de nos vies – en ligne et hors ligne – produit des données personnelles, collectées et partagées à grande échelle. Nos comportements, nos modes de vie, s'y lisent facilement. Mais faut-il s'en inquiéter ? Après tout, ces données qui nous révèlent sont souvent anonymisées par les organismes qui les collectent. C'est du moins ce que l'on peut lire sur leurs sites. Leur travail est-il efficace ? Et les données anonymes le sont-elles vraiment ? Dans notre dernier article publié dans la revue *Nature Communications*, nous développons une méthode mathématique qui montre que c'est loin d'être acquis. Elle a pu nous amener à réidentifier des individus parmi des bases de données anonymes et fortement échantillonnées, remettant en question les outils utilisés actuellement pour partager les données personnelles à travers le monde.

### Matière première

D'abord, quelques ordres de grandeur. Ces dix dernières années, nos données personnelles ont été collectées à une vitesse inégalée : 90 % de celles circulant sur Internet ont été créées il y a moins de deux ans ! Objets connectés, informations médicales ou financières, réseaux sociaux, ces données sont

### Auteur



**Luc Rocher**

Doctorant, ingénierie mathématique,  
Université catholique de Louvain

la matière première de l’économie numérique comme de la recherche scientifique moderne. Mais, très vite, on a vu apparaître certaines dérives. Notamment les atteintes à la vie privée qui se sont multipliées. Témoin, parmi de nombreuses affaires, le scandale Cambridge Analytica... Depuis, 80 % des Européen·ne·s estiment avoir perdu le contrôle sur leurs données.

En réponse, les compagnies et organismes qui les collectent affirment souvent qu’elles le sont de manière « anonyme ». Par exemple, la société Transport for London (TfL), en charge du métro londonien, a entrepris de surveiller les déplacements des passagers sur le réseau via les signaux wifi « anonymes » de leurs téléphones portables. En Belgique, plus de 15 hôpitaux revendent les données confidentielles de leurs patients à une multinationale, Quintiles IMS, sous couvert d’anonymat. Enfin, en France, Orange et SFR ont revendu des données de géolocalisation en temps réel ou en différé, données là encore « anonymisées ».

Point intéressant, une donnée anonyme n’est plus considérée comme donnée personnelle. Elle échappe donc aux régimes de protection comme le RGPD en Europe. Partager des données personnelles anonymisées ne nécessite donc plus le consentement des participant·e·s... Puisqu’ils et elles sont anonymes !

## Ré-identification

Or, des chercheur·e·s et journalistes ont depuis longtemps montré que certaines données anonymes peuvent être ré-identifiées. Dans les années 1990, Latanya Sweeney avait pu ré-identifier les données médicales de William Weld (alors gouverneur du Massachusetts), sur base de son code postal, sa date de naissance et son genre. Deux journalistes allemands ont récemment ré-identifié l’historique de navigation d’un juge et d’un député, retrouvant leurs préférences sexuelles et leurs traitements médicaux dans des données anonymes obtenues en se faisant passer pour des acheteurs potentiels. Et, aux États-Unis, les dossiers fiscaux du président américain Trump ont pu lui être ré-attribués par le *New York Times* en utilisant des données anonymes publiées par le fisc américain, l’IRS.

Compagnies et gouvernements minimisent souvent ces ré-identifications. Leur ligne de défense : parmi des petites bases de données, toujours incomplètes, personne ne saura jamais si une ré-identification est correcte ou non et si des chercheur·e·s ou journalistes ont vraiment réidentifié la bonne personne.

Cela implique que l’organisme collecteur fasse un travail dit d’*échantillonage* sur la base de données. Ainsi, l’autorité de protection des données australienne [OAIC], suggère dans son guide de dés-identification que l’échantillonnage augmente « l’incertitude qu’une personne particulière fasse réellement partie d’une base de données anonyme ». Prenons un exemple pour expliquer cela. Admettons que votre employeur retrouve des données vous correspondant dans un échantillon de 10 000 patients, soit 1 % d’une large base de données médicales. Ces données – comprenant par exemple votre lieu et date de naissance, genre, statut marital, etc. – pourraient bien appartenir à une autre personne qui partage ces caractéristiques. Car cette base de données de 10 000 personnes ne représente que 0,015 % de la population française. Et ces données réidentifiées pourraient



Traitement d’échantillons viraux. Les données personnelles de santé sont parmi les plus sensibles. James Gathany/CDC

correspondre à n'importe quelle autre personne parmi les 99,985 % autres Français·e·s.

Échantillonner (partager par exemple 1 % d'une base de données) est ainsi une technique largement utilisée. Réduire la taille des données partagées permet de justifier que ces données sont anonymes, car personne ne pourra jamais prouver qu'une ré-identification est correcte.

### Un algorithme qui remet en question l'anonymat

Le problème ? Nos travaux démontrent au contraire qu'un algorithme peut apprendre à estimer, avec grande précision, si des données réidentifiées appartiennent bien à la bonne personne ou non.

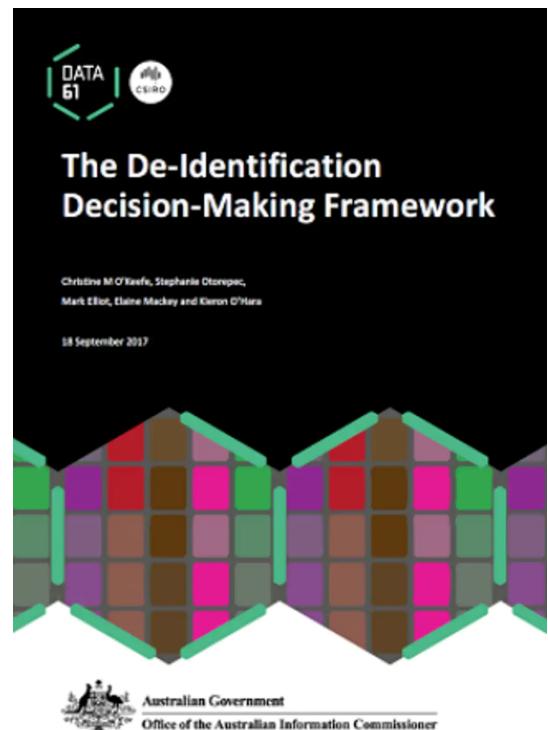
Il y a bien entendu, si c'est en France, de nombreux hommes trentenaires, habitant à Paris. Si je retrouve un seul homme de 30 ans parmi les données anonymes de 1 000 personnes, collectées et revendues par un cabinet d'assurance parisien, il y a peu de chance qu'elles correspondent à mon voisin Émeric. Les données correspondant à ces trois attributs (homme, 30 ans, habitant à Paris) seront sans doute celles d'un autre Français.

Mais au fur et à mesure que ces données s'enrichissent, qu'on apprend davantage de caractéristiques, il devient illusoire qu'une seconde personne ait les mêmes caractéristiques. Il y a ainsi sans doute un seul homme à Paris, né le 5 janvier 1989, roulant en vélo électrique et habitant avec ses deux enfants (deux filles) et un berger allemand : mon voisin Émeric.

Après avoir « appris » quelles caractéristiques rendent les individus uniques, notre algorithme génère des populations synthétiques pour estimer si un individu peut se démarquer parmi des milliards de personnes. Le modèle développé permettrait par exemple aux journalistes du New York Times de savoir à coup sûr si les dossiers identifiés appartenaient vraiment à Donald Trump.

Nos résultats montrent que 99,98 % des Américains seraient correctement ré-identifiés dans n'importe quelle base de données en utilisant 15 attributs démographiques. Les chiffres sont similaires à travers le monde (16 attributs en ajoutant la nationalité). Une quinzaine de caractéristiques qui suffisent à identifier un individu, ce n'est hélas pas beaucoup. Le « data broker » Acxiom, un courtier de données qui achète et qui revend nos données personnelles dans 60 pays, possède par exemple jusqu'à 5,000 attributs par personne.

Nos travaux remettent ainsi en question les pratiques actuelles utilisées pour dés-identifier des données personnelles. Cela interroge sur les limites de l'anonymisation : utiliser ainsi ces données protège-t-il toujours notre vie privée ? Alors que les standards d'anonymisation sont en passe d'être redéfinis par les pouvoirs publics, au niveau national et au sein de l'Union européenne, il est crucial pour ces standards d'être rigoureux, de promouvoir de meilleures méthodes de partage des données, et de prendre en compte tout risque futur. C'est à la fois important pour nos vies privées, pour la croissance de l'économie numérique et pour le dynamisme de la recherche scientifique.



Un guide pour protéger les données en Australie. Australian Government, CC BY



## IMAGES DE VAGUES ET DE STRUCTURES

---

### Préambule

Dans une production cinématographique en images de synthèse, les images sont créées une à une pour donner l'illusion du mouvement (sur le principe du dessin animé). Pour satisfaire les spectateurs, il est efficace de réaliser des images conformes aux équations de la physique.

Le sujet aborde la réalisation d'une scène montrant un bateau à moteur traversant un canal, créant un sillage à la surface de l'eau, et faisant osciller une gondole amarrée à proximité (figure 1).

Les programmes demandés sont à rédiger en langage Python 3. Si toutefois le candidat utilise une version antérieure de Python, il doit le préciser. Il n'est pas nécessaire d'avoir réussi à écrire le code d'une fonction pour pouvoir s'en servir dans une autre question. Les questions portant sur les bases de données sont à traiter en langage SQL.

### Notations mathématiques et physiques

On note  $\vec{V}$  un vecteur de  $\mathbb{R}^3$ , et  $V$  la variable qui lui est associée en Python. Dans l'ensemble du sujet, les vecteurs sont représentés par une liste de trois flottants. Par exemple, un vecteur dont les coordonnées sont  $x = 2$ ,  $y = 3$  et  $z = 1,5$  sera exprimé par :

**code Python**

```
1  V = [2., 3., 1.5]
```

Partout où cela est nécessaire, les variables sont considérées être exprimées dans les unités SI. Le repère  $(O, \vec{e_x}, \vec{e_y}, \vec{e_z})$  servant de référentiel est fixe par rapport au décor.

### Modèle de facettes

La scène contient plusieurs objets géométriques tri-dimensionnels (3D). Chaque objet géométrique est représenté de manière numérique par un maillage. On définit les termes suivants :

- **Maillage** : ensemble des facettes qui constituent la géométrie d'un objet. Un maillage sera représenté par une liste de facettes.
- **Facette** : polygone élémentaire qui constitue une partie de la surface d'un objet. Ici, toutes les facettes seront des triangles. Une facette sera représentée par une liste ordonnée de 3 sommets.
- **Sommet** : point délimitant une facette. Il peut être commun à une ou plusieurs facettes. Tout point sera représenté par son vecteur position de coordonnées  $(x,y,z)$ .

La figure 2(a) représente un exemple de maillage simple (tétraèdre), composé de 4 facettes (notées  $S_1$  à  $S_4$ ) et de 4 sommets (notés  $A$  à  $D$ ) avec  $AB = AD = AC = 1$ . Sa représentation en Python est donnée dans le paragraphe I.b.

**Organisation du sujet** Ce sujet se compose de trois parties indépendantes.

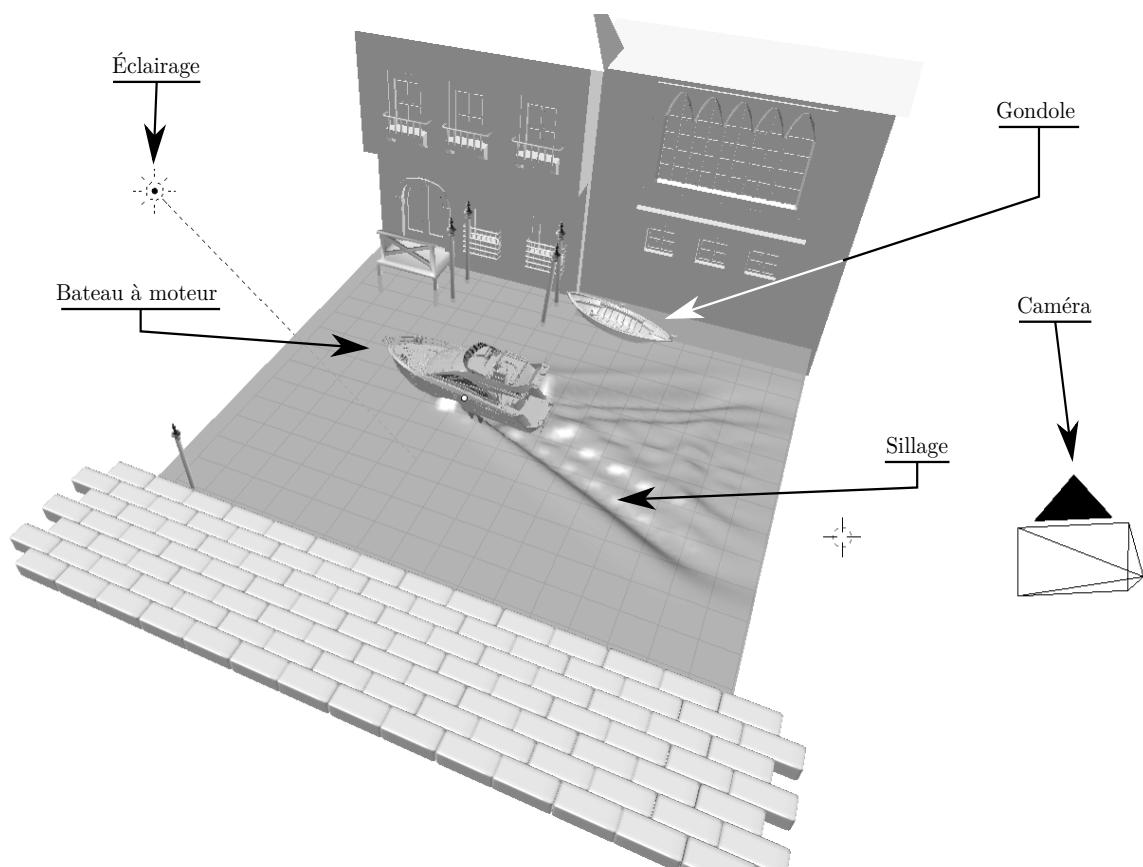
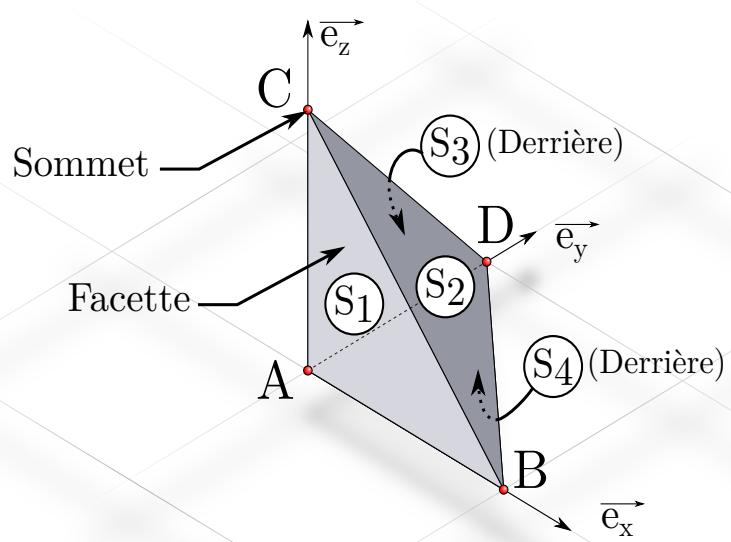
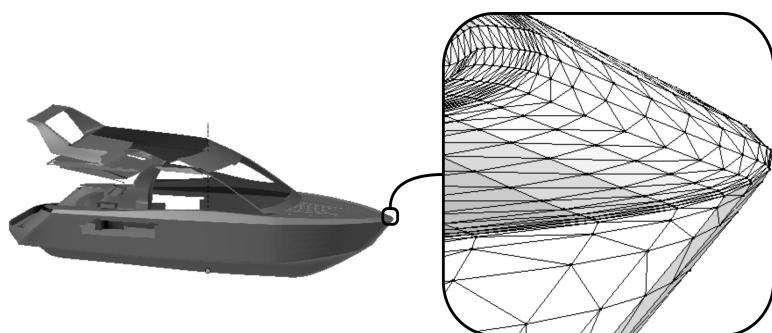


FIGURE 1 – Présentation de la scène étudiée.



(a) Un maillage simple : le tétraèdre



(b) Un maillage plus complexe : la coque d'un bateau.

FIGURE 2

## Partie I. Crédation d'un objet dans la scène

### I.a Chargement d'un modèle 3D à partir d'une base de données

On souhaite importer, dans Python, le modèle d'un bateau à moteur, élaboré préalablement. Ce modèle définit plusieurs maillages élémentaires (coque, bouée, échelle, moteur, etc.). Le fichier contenant ce modèle est une base de données relationnelle. Son schéma détaillé ci-dessous est illustré en figure 3 :

- relation **maillages\_bateau** : ensemble des maillages. Un maillage possède un identifiant (entier) et un nom (chaîne de caractères) :

**maillages\_bateau (id,nom)**

- relation **faces** : ensemble des facettes du modèle. Chaque facette est définie par un numéro unique, l'identifiant du maillage auquel elle appartient ainsi que les identifiants des sommets qui la composent. Tous sont des entiers.

**faces (numero,maillage,s1,s2,s3)**

- relation **sommets** : liste des sommets du modèle. Chaque sommet est défini par un identifiant (entier) et ses coordonnées dans l'espace par rapport au repère principal de la scène (flottant) :

**sommets (id,x,y,z)**

| <b>id</b> | <b>nom</b> |
|-----------|------------|
| 1         | coque      |
| 2         | bouée      |
| 3         | échelle    |
| 4         | moteur     |
| ...       | ...        |

(a) Relation  
**maillages\_bateau**

| <b>numero</b> | <b>maillage</b> | <b>s1</b> | <b>s2</b> | <b>s3</b> |
|---------------|-----------------|-----------|-----------|-----------|
| 1             | 3               | 1         | 2         | 3         |
| 2             | 3               | 2         | 4         | 3         |
| 3             | 2               | 3         | 12        | 5         |
| ...           | ...             | ...       | ...       | ...       |

(b) Relation **faces**

| <b>id</b> | <b>x</b> | <b>y</b> | <b>z</b> |
|-----------|----------|----------|----------|
| 1         | 0.0      | 0.0      | 0.0      |
| 2         | 1.0      | 0.0      | 0.0      |
| 3         | 0.0      | 1.0      | 0.0      |
| ...       | ...      | ...      | ...      |

(c) Relation **sommets**

FIGURE 3 – Exemples des relations de la base de données.

- Q1 – Proposez une requête SQL permettant de compter le nombre de maillages que contient le modèle du bateau.
- Q2 – Proposez une requête SQL permettant de récupérer la liste des numéros des facettes (numero) du maillage nommé « gouvernail ».
- Q3 – Expliquez ce que renvoie la requête SQL suivante :

Requête SQL

```

1  SELECT (MAX(x)-MIN(x))
2  FROM sommets AS s JOIN faces AS f JOIN maillages_bateau AS m
3  ON (s.id=f.s1 OR s.id=f.s2 OR s.id=f.s3) AND f.maillage=m.id
4  WHERE m.nom="coque"

```

### I.b Travail sur les facettes

À partir de requêtes sur la base de données précédente, on suppose avoir construit la variable Python suivante :

```
maillage_tetra = [ [[0.,0.,0.], [0.,0.,1.], [0.,1.,0.]],  
                   [[0.,0.,0.], [0.,1.,0.], [1.,0.,0.]],  
                   [[0.,0.,0.], [1.,0.,0.], [0.,0.,1.]],  
                   [[1.,0.,0.], [0.,1.,0.], [0.,0.,1.]] ]
```

Cette variable représente un maillage. L'exemple illustré en figure 2(a) représente le tétraèdre (le point A a pour coordonnées (0,0,0)).

□ Q4 – *À partir de la variable maillage\_tetra, écrire une expression Python permettant de récupérer la coordonnée y du premier sommet de la première facette.*

□ Q5 – *À quel élément, sur la figure 2(a), correspond maillage\_tetra[1] ?*

Dans le reste du sujet, on suppose qu'on dispose d'un module Python dont la documentation suit.

Help on module operations\_vectorielles :

## NAME

operations\_vectorielles

## DESCRIPTION

Ce module contient des fonctions qui implémentent des opérations usuelles sur les vecteurs. Sauf indication contraire explicite les arguments sont des vecteurs passés sous la forme de liste de trois réels.

## FUNCTIONS

**addition(V1, V2)**

Renvoie le vecteur correspondant à l'opération vectorielle V1+V2.

**aire(F)**

Renvoie l'aire d'une facette.

Argument :

F – une facette (liste de trois vecteurs)

**prod\_scalaire(V1, V2)**

Renvoie le produit scalaire de V1 avec V2.

**prod\_vectoriel(V1, V2)**

Renvoie le vecteur correspondant au produit vectoriel de V1 avec V2.

**soustraction(V1, V2)**

Renvoie le vecteur correspondant à l'opération vectorielle V1-V2.

**barycentre(F)**

Renvoie le vecteur position du barycentre d'une facette.

Argument :

F – une facette (liste de trois vecteurs)

## FILE

operations\_vectorielles.py

- Q6 – *On souhaite utiliser les fonctions de ce module depuis un autre fichier Python. Complétez le code ci-dessous afin qu'il fournisse le résultat attendu :*

code Python

```
1 from ??? import ?? as ?
2 vect_1 = [1., 2., 3.]
3 vect_2 = [2., 3., 4.]
4 scal12 = ps(vect_1, vect_2) #Calcul du produit scalaire de vect_1 avec vect_2
```

Dans ce module, une fonction n'a pas été documentée :

code Python

```
1 def mystere1(V):
2     return (V[0]**2 + V[1]**2 + V[2]**2)**0.5
```

□ Q7 – Que fait la fonction `mystere1` ?

□ Q8 – Créer la fonction `multiplie_scalaire`, prenant comme argument un flottant `a` et un vecteur `V` et renvoyant un nouveau vecteur correspondant à  $a \vec{V}$ .

La fonction `barycentre` (incomplète) est définie ci-dessous.

code Python

```
1 def barycentre(F):
2     G = [0,0,0]
3     for i in range(3): #Pour chaque point de F
4         ..... # Ligne à compléter
5         ..... # Ligne à compléter
6     return G
```

□ Q9 – Compléter les lignes 4 et 5 permettant de calculer le barycentre.

□ Q10 – Pour une facette  $F=(A,B,C)$  d'aire non-nulle, proposer une fonction normale, prenant comme argument une facette `F` et renvoyant le vecteur unitaire normal

$$\vec{n} = \frac{\vec{AB} \wedge \vec{AC}}{\|\vec{AB} \wedge \vec{AC}\|}$$

### I.c Liste des sommets

□ Q11 – Compte tenu de la représentation limitée des nombres réels en machine, deux sommets  $S_1$  et  $S_2$  supposés être au même endroit peuvent avoir des coordonnées légèrement différentes. Proposer une fonction `sont_proches`, prenant comme arguments deux sommets `S1` et `S2` (représentés par leur vecteur `position`) et un flottant positif `eps`, et qui renvoie `True` si  $S_1$  et  $S_2$  sont proches (i.e. si leur distance au sens de la norme Euclidienne est inférieure à `eps`) et `False` sinon.

Soient les fonctions suivantes :

code Python

```
1 def mystere2(S1, L):
2     for S2 in L:
3         if sont_proches(S1, S2, 1e-7):
4             return True
5     return False
6
7
8 def mystere3(maillage):
9     res = []
10    for facette in maillage:
11        for sommet in facette :
12            if not mystere2(sommet, res):
13                res.append(sommet)
14    return res
```

- Q12 – *Sous quelle condition la fonction mystere2 renvoie-t-elle True ?*
- Q13 – *Donner (sans justification) ce que renvoie mystere3(maillage\_tetra), dans le cas où maillage\_tetra est la variable définie précédemment.*
- Q14 – *Pour une liste L de longueur n, discuter la complexité de la fonction mystere2. En déduire la complexité de mystere3, pour un maillage contenant m facettes triangulaires. On distinguerá le meilleur et le pire des cas.*

## Partie II. Génération de vagues

Le bateau à moteur qui traverse le canal génère des vagues (appelées *sillage de Kelvin*). On ne dispose de formules analytiques décrivant ces ondes que dans des cas très simples, comme celui d'un bateau en translation rectiligne uniforme.

Le calcul numérique permettant d'évaluer la forme de ces vagues étant coûteux, il est réalisé par un programme extérieur. Ce dernier génère l'état du plan d'eau à chaque image de la scène (25 par seconde).

Pour chacune de ces images, on représente le plan d'eau par une grille régulière carrée, de taille  $(m+1) \times (m+1)$  (figure 4). Chaque case  $(i,j)$  de cette grille correspond à un sommet du maillage de la surface de l'eau, noté  $n_{i,j}$  de coordonnées  $(x_i, y_j)$  (avec  $(i,j) \in \llbracket 0, m \rrbracket^2$ ).

La hauteur (sur  $\vec{e}_z$ ) de  $n_{i,j}$  est notée  $h_{i,j}$ . L'ensemble de tous les  $h_{i,j}$  est stocké dans une liste de listes nommée `mat_h`, tel que `mat_h[i][j] = h_{i,j}`.

Le programme extérieur calcule ainsi `mat_h` pour chaque nouvelle image. L'ensemble de toutes les valeurs de `mat_h` est stocké dans une liste nommée `liste_vagues`.

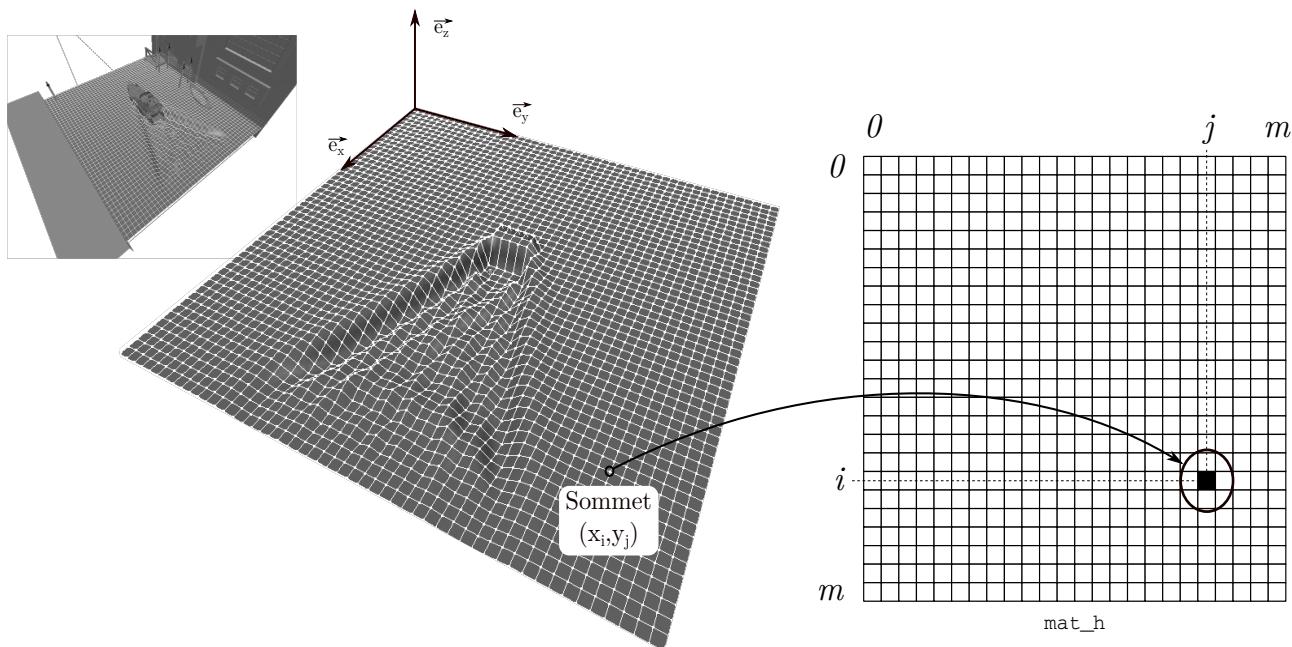


FIGURE 4 – Illustration du stockage de la hauteur d'eau dans un tableau.

La scène est composée de 350 images. Le plan d'eau est composé de  $200 \times 200$  sommets. Chaque hauteur  $h_{i,j}$  est un flottant codé sur 64 bits.

- Q15 – *Quel est l'espace occupé en mémoire vive par l'ensemble des données (en Mo).*

On souhaite enregistrer le contenu de `liste_vagues` dans un fichier afin de le transmettre au logiciel d'animation.

□ Q16 – *Écrire une fonction `mat2str` qui prend en argument une liste de listes (représentant un `mat_h`) et renvoie les données qu'elle contient sous forme d'une chaîne de caractères qui respecte le format suivant :*

$$\begin{array}{l} h_{01};h_{02}; \dots ;h_{0m} \\ | \\ h_{11}; \dots \quad ;h_{1m} \\ | \\ \dots \\ | \\ h_{m0}; \dots \quad ;h_{mm} \end{array}$$

*On rappelle que le retour à la ligne est codé par le caractère "\n".*

□ Q17 – *En s'appuyant sur `mat2str`, proposer un code Python qui permet de sauvegarder le contenu de `liste_vagues` dans un fichier nommé ■ `fichier_vagues.txt` ■ (dans le répertoire courant), en séparant la représentation de chaque `mat_h` par deux sauts de lignes consécutifs.*

Le fichier texte obtenu est jugé trop volumineux. On décide de recourir à des matrices creuses. Dans ce qui suit on désigne par matrice creuse une matrice dont seuls la valeur et l'emplacement des éléments non-nuls (c'est-à-dire significativement éloignés de zéro) sont enregistrés.

On propose d'utiliser le format de fichier nommé « Coordinate Format » qui consiste à stocker :

- une liste I, comportant les numéros de ligne de chaque élément non-nul,
- une liste J, comportant les numéros de colonne de chaque élément non-nul,
- une liste N, comportant la valeur de chaque élément non-nul.

Les valeurs d'une liste sont enregistrées sur une même ligne et séparées par des points-virgules. Chaque liste est séparée des autres par un retour à la ligne. On admet que les flottants sont écrits dans le fichier avec 15 caractères (tout compris).

□ Q18 – *Après avoir judicieusement les types des éléments contenus dans I, J puis N, estimer la taille (en octets) que prendra une matrice ayant p éléments non-nuls, au format « Coordinate Format », dans le fichier.*

□ Q19 – *En déduire à partir de combien d'éléments non-nuls il devient moins avantageux d'enregistrer une matrice creuse qu'une matrice complète classique.*

□ Q20 – *Proposer un code permettant de construire, pour un tableau `mat_h` donné, les listes Python I, J et N. On considérera nulles les hauteurs inférieures à  $10^{-3}$  (en valeur absolue).*

### Partie III. Mouvement de flottaison

La gondole amarrée sur le bord du canal perçoit les vagues générées par le bateau à moteur et effectue un mouvement pseudo-oscillant conséquence de sa flottaison. On étudie ici le mouvement de translation verticale de la gondole (on ne prendra en compte ni le tangage ni le roulis). On cherche à modéliser ce mouvement en calculant à un instant donné la force exercée par le fluide sur la gondole.

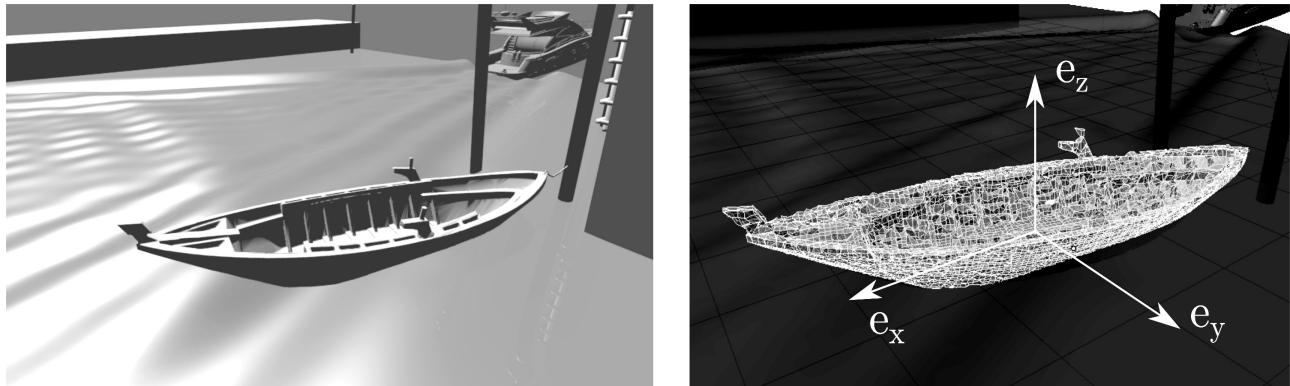


FIGURE 5 – Maillage de la gondole.

### III.a Estimation de la poussée d'Archimède

Seul le mouvement de translation verticale (selon la direction verticale  $\vec{e}_z$ ) est étudié.

On s'intéresse ici au maillage qui constitue la coque extérieure de la gondole. Certaines facettes sont émergées (*i.e.* leur barycentre est en dehors de l'eau), d'autres sont immergées (*i.e* leur barycentre est sous l'eau).

À chaque pas de temps, on suppose connue la fonction `hauteur(x,y)` qui, à tout point  $(x,y)$  du plan d'eau, associe la hauteur des vagues par rapport au repère de la scène. Le maillage composant la coque de la gondole est défini dans une liste de facette nommée `maillageG`, similaire à celle présentée dans la partie précédente.

**Q21 – Proposer une fonction `lister_FI` prenant comme argument un maillage M et renvoyant la liste des facettes immergées (*i.e dont le centre de gravité est sous la surface définie par hauteur*). On pourra utiliser les fonctions de la partie I.**

Pour calculer la poussée d'Archimède s'exerçant sur la coque du bateau, on doit calculer la résultante des forces dues à la pression, appliquées par l'eau sur chacune des facettes immergées.

On modélise la force appliquée par l'eau sur une facette  $i$  par :

$$\vec{F}_i = -S_i \times p(G_i) \vec{n}_i$$

avec :

- $S_i$  : l'aire de la facette,
- $\vec{n}_i$  : le vecteur normal sortant de la coque,
- $p(G_i)$  : la pression hydrostatique de l'eau sur la facette en son barycentre  $G_i$ .

Le théorème de Pascal détermine la pression de l'eau d'un point  $G$  en fonction de sa profondeur par rapport à la surface :

$$p(x_G, y_G, z_G) = \rho \cdot g \cdot (\text{hauteur}(x_G, y_G) - z_G)$$

avec :

- $\rho$  : masse volumique de l'eau ( $\rho \approx 1000$ )
- $g$  : accélération de la pesanteur (ici :  $g \approx 9,81$ )

**Q22 – Proposer une fonction `force_facette` prenant en argument une facette F, et renvoyant le vecteur force appliqué par l'eau sur cette facette. On pourra utiliser les fonctions définies précédemment.**

La force résultante sur toute la coque s'exprime par la somme de toutes les forces appliquées sur chaque facette immergée.

- Q23 – *Définir la fonction résultante prenant comme argument une liste L de facettes (supposées immergées), renvoyant la somme des forces sur l'axe  $\vec{z}$  de l'eau, appliquée sur l'ensemble des surfaces.*

### III.b Tri des facettes

On cherche à optimiser l'efficacité de la fonction `resultante` qui devra être utilisée intensément pour réaliser un grand nombre d'images. On remarque que la taille des facettes n'est pas homogène : la coque est composée de grandes facettes et de petites facettes. Les petites facettes représentent souvent des détails d'intérêt graphique n'apportant qu'une très faible contribution à la résultante des forces hydrostatiques.

Ainsi, une étude montre que la moitié des facettes représente à elle seule 99% de la surface totale de la coque. Pour alléger le processus, on souhaite donc trier les facettes par aire décroissante, afin de n'appliquer les calculs de la poussée d'Archimède qu'à la moitié d'entre elles (les plus grandes). On propose le code (incomplet) du tri-fusion ci-dessous :

code Python

```

1 def fusion(L1, L2):
2     # À compléter (sur une ou plusieurs lignes)
3
4 def trier_facettes(L):
5     # À compléter (sur une ou plusieurs lignes)
6
7 grandesFacettes = # À compléter

```

- Q24 – *Compléter la fonction fusion, prenant comme argument deux listes de facettes L1 et L2 (supposée chacune triée par aire décroissante) et renvoyant une nouvelle liste composée des facettes de L1 et L2 triées par aire décroissante.*

- Q25 – *Compléter la fonction récursive trier\_facettes, prenant comme argument une liste de facettes L, et renvoyant une nouvelle liste de facettes triées dans l'ordre des aires décroissantes, par la méthode du tri-fusion.*

- Q26 – *Affecter à une nouvelle variable grandesFacettes la liste des facettes de maillageG, privée de la moitié des facettes les plus petites (en cas de nombre impair d'éléments, on inclura la facette médiane).*

### III.c Mouvement vertical de la gondole

La gondole est attachée à un repère local dont l'origine est son centre de gravité (de coordonnées  $(x_G, y_G, z_G)$  et de vitesse verticale notée  $v$ ) par rapport au décor. Le principe fondamental de la dynamique en projection sur l'axe vertical  $\vec{e}_z$ , appliqué à la gondole énonce que :

$$\frac{dv}{dt} = \frac{1}{m} \times F_{eau \rightarrow gondole} - g$$

$$\frac{dz_G}{dt} = v$$

avec

- $m$  : masse de la gondole
- $F_{eau \rightarrow gondole}$  est la résultante des forces appliquées par l'eau sur la gondole (renvoyée par la fonction `resultante`, vue précédemment).

La position initiale de la gondole est  $z_{G0} = 0$ . Sa vitesse verticale initiale est  $v_0 = 0$ .

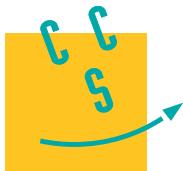
On souhaite estimer le mouvement par la méthode d'Euler. Pour ce faire, on utilise la fonction `nouvelle_hauteur` avant d'afficher chaque nouvelle image. Cette fonction a pour but de recalculer la hauteur (et la vitesse) de la gondole pour un nouveau pas de temps. Elle prend trois arguments :

- `posG` contiendra le vecteur position actuel du centre de gravité de la gondole au moment de l'appel (liste de trois flottants) ;
- `vitG` contiendra le vecteur vitesse actuel de ce même point au moment de l'appel (liste de trois flottants) ;
- `mailG` contiendra la liste des grandes facettes de la gondole (privée des petites, au sens de la question précédente), au moment de l'appel.

code Python

```
1 def nouvelle_hauteur(posG, vitG, mailG):  
2     dt=1.0/25.0 # Pas de temps correspondant à une image du film.  
3     facettes_immergees = lister_FI(mailG)  
4     posG = posG + .....      # à compléter  
5     vitG = vitG + .....      # à compléter  
6     return posG, vitG
```

Q27 – Compléter les lignes 4 et 5 du code précédent conformément à la méthode d'Euler.



CONCOURS CENTRALE-SUPÉLEC

# Informatique

**MP, PC, PSI, TSI**

3 heures

Calculatrice autorisée

**2020**

## *Photomosaïque*

Une *photomosaïque* (figure 1) est une image composée à la manière d'une mosaïque, où les fragments sont eux-mêmes des petites images, appelées *vignettes*. Elle est créée à partir d'une image appelée *image source*. Chaque vignette remplace une zone de même forme dans l'image source appelée *pavé*. Les vignettes sont fabriquées à partir d'une collection d'images appelée *banque* d'images.

L'intérêt est essentiellement artistique : vue de loin, une photomosaïque ressemble à l'image source ; en se rapprochant, on reconnaît les vignettes.



**Figure 1** Photomosaïque d'un surfeur composée de 1600 vignettes — de gauche à droite : l'image source<sup>1</sup>, la photomosaïque et 16 vignettes<sup>2</sup> (détail du pied)

De nombreux paramètres régissent la construction d'une photomosaïque et la qualité du résultat :

- la structure du pavage utilisé (nombre, forme et arrangement des vignettes) ;
- le nombre et la diversité des images de la banque ;
- les algorithmes mis en œuvre pour :
  - sélectionner les bonnes images dans la banque (partie III) ;
  - redimensionner les images (partie II) ;
  - placer les vignettes (partie IV).

Dans la suite du sujet, les photomosaïques sont construites sur des pavages rectangulaires réguliers, c'est-à-dire, constitués de vignettes rectangulaires, toutes de mêmes dimensions et juxtaposées bord à bord.

Les seuls langages de programmation autorisés dans cette épreuve sont Python et SQL. Pour répondre à une question, il est possible de faire appel aux fonctions définies dans les questions précédentes. Dans tout le sujet, on suppose que les modules `math`, `numpy`, `matplotlib.pyplot` et `random` ont été rendus accessibles grâce à l'instruction

```
import math, numpy as np, matplotlib.pyplot as plt, random
```

Si les candidats font appel à des fonctions d'autres bibliothèques, ils doivent préciser les instructions d'importation correspondantes.

Dans tout le sujet, le terme « liste » appliqué à un objet Python signifie qu'il s'agit d'une variable de type `list`. Les termes « vecteur » et « tableau » désignent des objets `numpy` de type `np.ndarray`, respectivement à une dimension ou de dimension quelconque. Enfin le terme « séquence » représente une suite itérable et indicable, indépendamment de son type Python, ainsi un tuple d'entiers, une liste d'entiers et un vecteur d'entiers sont tous trois des « séquences d'entiers ».

<sup>1</sup> Photo par « Sincerely Media », issue de <https://unsplash.com>.

<sup>2</sup> Vignettes issues de la banque <https://picsum.photos/images>.

Les entêtes des fonctions demandées sont annotés pour préciser les types des paramètres et du résultat. Ainsi,

```
def uneFonction(n:int, X:[float], c:str, u) -> np.ndarray:
```

signifie que la fonction `uneFonction` prend quatre paramètres `n`, `X`, `c` et `u`, où `n` est un entier, `X` une liste de nombres à virgule flottante, `c` une chaîne de caractères et le type de `u` n'est pas précisé. Cette fonction renvoie un tableau numpy.

Il n'est pas demandé aux candidats d'annoter leurs fonctions, la rédaction pourra commencer par

```
def uneFonction(n, X, c, u):
```

```
...
```

De façon générale, une attention particulière sera portée à la lisibilité, la simplicité et la clarté du code proposé. L'utilisation d'identifiants significatifs, l'emploi judicieux de commentaires seront appréciés.

Une liste de fonctions potentiellement utiles est fournie à la fin du sujet.

## I Pixels et images

### I.A – *Pixels*

Un pixel (contraction de l'anglais *picture element*) est un élément de couleur homogène utilisé pour représenter une image sous forme numérique. La teinte d'un pixel peut être représentée de plusieurs façons. Une méthode courante, basée sur la synthèse additive, consiste à la décomposer en trois composantes qui correspondent aux couleurs rouge, vert et bleu. On parle de représentation RGB (pour *red*, *green* et *blue*). Chacune des trois composantes donne l'intensité de la couleur correspondante dans la teinte finale, 0 indiquant l'absence de cette couleur. Ainsi, le triplet (0, 0, 0) désigne un pixel noir.

**Q 1.** On suppose que chacune des trois composantes RGB d'un pixel est représentée par un nombre entier positif ou nul, codé sur 8 bits. Combien de couleurs différentes peut-on représenter avec un tel pixel ?

Dans la suite, on représente un pixel par un vecteur (tableau numpy à une dimension) d'entiers de type `np.uint8` (entier non signé codé sur 8 bits) à trois éléments, correspondant respectivement à chacune des composantes RGB du pixel ; on utilise dans toute la suite le type `pixel` pour désigner un tel vecteur.

**Q 2.** Donner une instruction permettant de créer un vecteur correspondant à un pixel blanc.

Il est rappelé qu'en Python, comme dans beaucoup de langages de programmation, les opérations d'addition, soustraction, multiplication, division entière, modulo et élévation à la puissance (opérateurs `+`, `-`, `*`, `//`, `%`, `**`) appliquées à deux opérandes de même type fournissent un résultat du type de leurs opérandes. Cela peut conduire à un dépassement de capacité et à une erreur de calcul car, les dépassements de capacité étant par défaut « silencieux », ils ne produisent pas d'erreur lors de l'exécution du programme.

L'opérateur division (`/`) entre deux entiers produit toujours un résultat sous forme de nombre à virgule flottante même si la division est exacte (`12 / 2 → 6.0`). Il en est de même pour toute fonction faisant implicitement appel à cet opérateur comme `np.mean`.

**Q 3.** On pose `a = np.uint8(280)` et `b = np.uint8(240)`. Que valent `a`, `b`, `a+b`, `a-b`, `a//b` et `a/b` ?

Les fonctions numpy qui effectuent de manière répétitive des opérations élémentaires, si elles ne garantissent pas l'absence de dépassement de capacité, prennent la précaution d'utiliser pour leurs calculs intermédiaires et leur résultat un type compatible avec le type de base de la plus grande capacité possible. Par exemple le résultat de `np.sum(np.array([100, 200], np.uint8))` est de type `np.uint64` (entier non signé codé sur 64 bits) et vaut bien 300.

Pour représenter une image en niveau de gris, on peut se contenter d'une valeur par pixel, représentant l'intensité du gris entre le noir et le blanc. Pour convertir une image en couleurs en niveaux de gris, on peut remplacer chaque pixel par un seul entier, dont la valeur correspond à la meilleure approximation entière de la moyenne des trois composantes RGB du pixel.

**Q 4.** Écrire une fonction d'entête

```
def gris(p:pixel) -> np.uint8:
```

qui calcule le niveau de gris correspondant au pixel `p`.

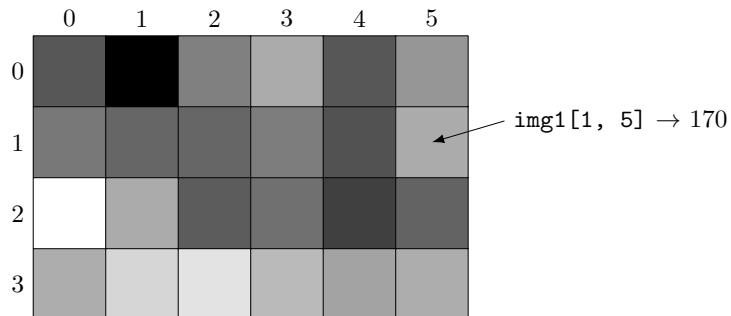
### I.B – *Images*

Une image en niveaux de gris de taille  $w \times h$  ( $w$  pixels de large,  $h$  pixels de haut) est associée à un tableau d'octets (type `np.uint8`) à deux dimensions, à  $h$  lignes et  $w$  colonnes. Chaque élément de ce tableau représente le niveau de gris du pixel correspondant. Ainsi le tableau à deux dimensions `img1`, défini par :

```
img1 = np.array([[ 85,    0, 127, 170,   85, 150],
                 [119, 102, 102, 123,   81, 170],
                 [255, 170,   90, 112,   63,  97],
                 [171, 212, 225, 186, 162, 171]], np.uint8)
```

définit une image de taille  $6 \times 4$ , représentée figure 2.

Dans toute la suite, on utilise le type `image` pour désigner un tableau d'octets à deux dimensions.

**Figure 2** Visualisation de l'image img1

Pour les images en couleurs, on ajoute une dimension pour représenter les trois composantes d'un pixel. L'instruction `source = plt.imread("surfer.jpg")` charge dans un tableau numpy l'image en couleurs contenue dans le fichier `surfer.jpg`. Les expressions `source.shape` et `source[0,0]` valent alors respectivement :

`(3000, 4000, 3)` et `np.array([144, 191, 221], np.uint8)`.

**Q 5.** Interpréter ces valeurs.

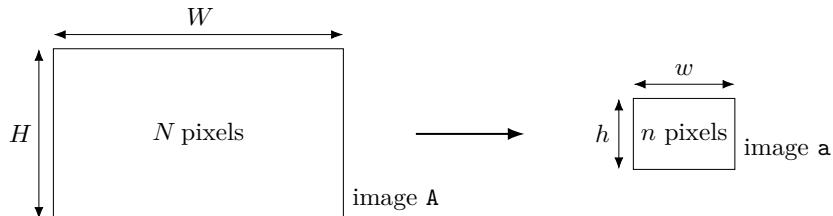
**Q 6.** Écrire une fonction d'entête

```
def conversion(a:np.ndarray) -> image:
```

qui génère une image en niveaux de gris correspondant à la conversion de l'image en couleurs `a`.

## II Redimensionnement d'images

On s'intéresse dans cette partie à plusieurs algorithmes de redimensionnement d'une image `A`, de taille  $W \times H$  ( $W$  pixels de large par  $H$  pixels de haut, on note  $N = HW$  son nombre total de pixels), en une image `a` de taille  $w \times h$  (on pose  $n = hw$ ). Nous nous intéresserons dans la suite uniquement à des images en niveau de gris.

**Figure 3** Redimensionnement d'image

### II.A – Le contexte

À l'occasion du mariage d'Alice et de Bernard, leurs amis souhaitent réaliser plusieurs photomosaïques sur des thèmes variés. Ils ont pour cela accumulé un grand nombre de photos au ratio 4:3, ce qui signifie que le rapport  $W/H$  vaut *exactement* 4/3. Les photomosaïques mesureront chacune deux mètres de large et seront constituées de  $40 \times 40 = 1600$  vignettes, toutes de même taille et au même ratio 4:3. Pour garder une bonne qualité d'impression, ils choisissent une résolution de 10 pixels par millimètre.

**Q 7.** Quelle taille de vignette ( $w \times h$ , en pixels) faut-il choisir ? Quelle sera alors la taille en pixels de la photomosaïque ?

### II.B – Algorithme d'interpolation au plus proche voisin

Cette interpolation est définie par la formule  $a(i, j) = A\left(\left\lfloor \frac{iH}{h} \right\rfloor, \left\lfloor \frac{jW}{w} \right\rfloor\right)$  où  $\lfloor x \rfloor$  désigne la partie entière de  $x$ .

**Q 8.** Écrire une fonction d'entête

```
def procheVoisin(A:image, w:int, h:int) -> image:
```

qui renvoie une nouvelle image correspondant au redimensionnement de l'image `A` à la taille `w × h` en utilisant l'interpolation au plus proche voisin.

**Q 9.** Quelle est sa complexité temporelle asymptotique ?

### II.C – Algorithme de réduction par moyenne locale

On suppose ici que les dimensions de l'image `a` divisent celles de l'image `A`:  $H/h$  et  $W/w$  sont entiers. Afin d'améliorer la qualité de la réduction, on propose la fonction `moyenneLocale`.

```

1 def moyenneLocale(A:image, w:int, h:int) -> image:
2     a = np.empty((h, w), np.uint8)
3     H, W = A.shape
4     ph, pw = H // h, W // w
5     for I in range(0, H, ph):
6         for J in range(0, W, pw):
7             a[I // ph, J // pw] = round(np.mean(A[I:I+ph, J:J+pw]))
8     return a

```

**Q 10.** Expliquer en quelques lignes son principe de fonctionnement.

**Q 11.** Donner sa complexité temporelle asymptotique.

#### **II.D – Optimisation de la réduction par moyenne locale**

Afin d'accélérer le calcul de la moyenne locale, on précalcule pour chaque image sa table de sommation. La table de sommation d'une image A de  $N$  pixels, représentée par le tableau  $A$  à  $H$  lignes et  $W$  colonnes, est le tableau  $S$  à  $H + 1$  lignes et  $W + 1$  colonnes, défini par

$$\forall l \in \llbracket 0, H \rrbracket, \quad \forall c \in \llbracket 0, W \rrbracket, \quad S(l, c) = \sum_{\substack{0 \leq i < l \\ 0 \leq j < c}} A(i, j),$$

la somme étant prise nulle quand elle ne comporte aucun terme.

**Q 12.** Le type `np.uint32` (entier non signé codé sur 32 bits) est-il suffisant pour stocker les éléments de  $S$  si l'image A comporte 50 millions de pixels ? Justifier.

**Q 13.** Écrire une fonction, de complexité temporelle asymptotique  $O(N)$ , d'entête

```
def tableSommation(A:image) -> np.ndarray:
```

qui calcule la table de sommation de l'image A.

On suppose à nouveau que les dimensions de l'image A divisent celles de l'image a :  $H/h$  et  $W/w$  sont entiers. On propose alors la fonction `réductionSommation1`, qui prend en paramètre l'image A et sa table de sommation S (`S = tableSommation(A)`), ainsi que les dimensions de l'image que l'on souhaite obtenir.

```

1 def réductionSommation1(A:image, S:np.ndarray, w:int, h:int) -> image:
2     a = np.empty((h, w), np.uint8)
3     H, W = A.shape
4     ph, pw = H // h, W // w
5     nbp = ph * pw
6     for I in range(0, H, ph):
7         for J in range(0, W, pw):
8             X = (S[I+ph, J+pw] - S[I+ph, J]) - (S[I, J+pw] - S[I, J])
9             a[I // ph, J // pw] = round(X / nbp)
10    return a

```

**Q 14.** Expliquer en quelques lignes le principe de fonctionnement de `réductionSommation1`.

**Q 15.** Donner sa complexité temporelle asymptotique.

**Q 16.** Montrer que la fonction `réductionSommation2` dont le code est fourni ci-dessous donne le même résultat que `réductionSommation1`.

```

1 def réductionSommation2(A:image, S:np.ndarray, w:int, h:int) -> image:
2     H, W = A.shape
3     ph, pw = H // h, W // w
4     sred = S[0:H+1:ph, 0:W+1:pw]
5     dc = sred[:, 1:] - sred[:, :-1]
6     dl = dc[1:, :] - dc[:-1, :]
7     d = dl / (ph * pw)
8     return np.uint8(d.round())

```

**Q 17.** Comparer les complexités asymptotiques en temps et en mémoire des deux versions de la fonction `réductionSommation`. Quel est l'avantage de la seconde version ?

#### **II.E – Synthèse**

**Q 18.** Discuter des cas d'usage respectifs de `procheVoisin`, `moyenneLocale` et `réductionSommation` pour redimensionner une image.

### III Sélection des images de la banque

Une première étape dans la conception d'une photomosaïque est le choix d'une image source et de vignettes. Cette partie est consacrée à la sélection d'images dans la banque.

Les images de la banque sont répertoriées dans une base de données dont le modèle physique est présenté figure 4, dans laquelle les clés primaires sont notées en italique.

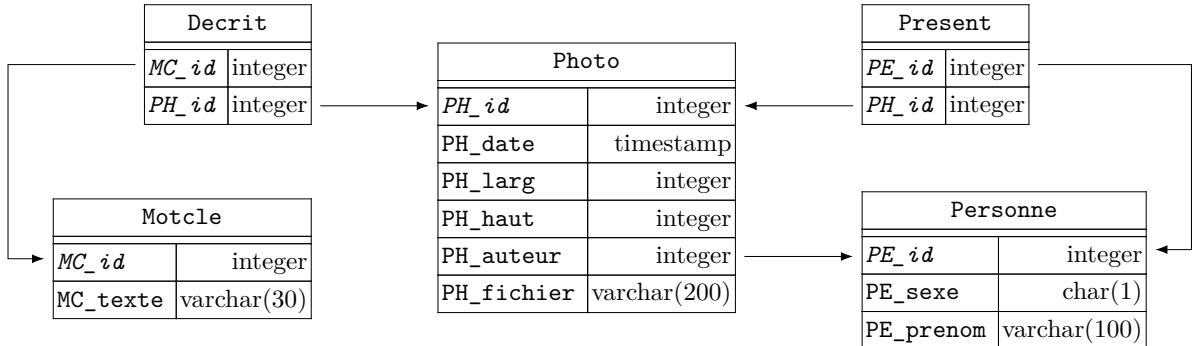


Figure 4 Structure physique de la base de données de photographies.

Cette base comporte les cinq tables listées ci-dessous avec la description de leurs colonnes :

- la table Photo répertorie les photographies
  - *PH\_id* identifiant (entier arbitraire) de la photographie (clé primaire)
  - *PH\_date* date et heure de la prise de vue
  - *PH\_larg*, *PH\_haut* largeur et hauteur de la photographie en pixels
  - *PH\_auteur* identifiant de l'auteur de la photographie
  - *PH\_fichier* nom du fichier contenant la photographie
- la table Personne des modèles et des photographes
  - *PE\_id* identifiant (entier arbitraire) de la personne (clé primaire)
  - *PE\_sexe* sexe de la personne ('M' ou 'F')
  - *PE\_prenom* prénom de la personne
- la table Motcle des mots-clés utilisés pour décrire une photographie
  - *MC\_id* identifiant (entier arbitraire) du mot-clé (clé primaire)
  - *MC\_texte* le mot-clé lui-même
- la table Decrit fait le lien entre les photographies et les mots-clés qui les décrivent, ses deux colonnes constituent sa clé primaire
  - *MC\_id* identifiant du mot-clé (décrivant la photographie)
  - *PH\_id* identifiant de la photographie (décrite par le mot-clé)
- la table Present fait le lien entre les photographies et les personnes qui y figurent, ses deux colonnes constituent sa clé primaire
  - *PE\_id* identifiant de la personne (figurant sur la photographie)
  - *PH\_id* identifiant de la photographie (représentant la personne)

#### III.A – Quelques requêtes

Pour réaliser les photomosaïques du mariage d'Alice et Bernard, on dispose de plus de 20 000 photographies répertoriées dans une base de données dont le modèle est celui de la figure 4.

**Q 19.** Écrire une requête SQL donnant les identifiants de toutes les photographies au ratio 4:3, c'est-à-dire dont le rapport largeur sur hauteur vaut exactement 4/3.

**Q 20.** Écrire une requête qui compte le nombre de photos prises par « Alice » ou « Bernard ».

**Q 21.** Écrire une requête qui fournit l'identifiant et la date des photographies prises avant 2006 et associées au mot-clé « surf ».

**Q 22.** Écrire une requête qui donne le prénom de l'auteur et l'identifiant de tous les selfies, c'est-à-dire les photographies sur lesquelles l'auteur est présent.

**Q 23.** Écrire une requête qui sélectionne toutes les photographies sur lesquelles sont présents « Alice » et « Bernard », à l'exclusion de toute autre personne.

***III.B – Internationalisation des mots-clés***

Afin de partager et d'enrichir la banque d'images, il a été décidé de faire évoluer la structure de la base de données afin de gérer les mots-clés dans différentes langues. Le cahier des charges de cette évolution stipule :

- l'ensemble des photographies sélectionnées à l'aide de mots-clés ne doit pas dépendre de la langue utilisée pour exprimer ces mots-clés ; autrement dit, les photographies décrites par le mot-clé « montagne » exprimé en français doivent être les mêmes que celles sélectionnées par les mots-clés « mountain » si la langue choisie est l'anglais, « Berg » pour l'allemand, « montaña » pour l'espagnol, etc. ;
- il doit être possible, pour cette nouvelle base de données, d'écrire une requête de recherche de photographies par mot-clé en spécifiant la langue utilisée pour exprimer le mot-clé de telle sorte que changer de langue se fasse en modifiant uniquement des constantes dans la clause WHERE.

**Q 24.** Proposer un nouveau modèle de base de données répondant à cette évolution du cahier des charges en ne détaillant que ce qui change (tables modifiées, nouvelles tables).

**Q 25.** Avec cette nouvelle base de données, écrire une requête qui permet de sélectionner les identifiants des photographies associées au mot-clé « mountain » exprimé en anglais.

**IV Placement des vignettes*****IV.A – Préparatifs***

On envisage ici le cas où la photomosaïque est homothétique de l'image source et constituée de  $p$  vignettes de haut sur  $p$  vignettes de large. Le nombre total de vignettes est donc  $r = p^2$ .

**Q 26.** Écrire une fonction d'entête

```
def initMosaïque(source:image, w:int, h:int, p:int) -> image:
```

qui prend en paramètre l'image source, les dimensions  $w$  et  $h$  d'une vignette et le nombre  $p$  de vignettes par côté. Cette fonction renvoie une version redimensionnée de `source`, de même taille que la photomosaïque finale. On rappelle qu'il est possible d'utiliser les fonctions définies précédemment.

On appelle désormais *pavé* chaque zone de cette image source redimensionnée, de taille  $w \times h$ , qui doit être remplacé par une vignette. Afin de comparer les vignettes et les pavés, on définit la distance  $L_1$  entre deux images  $a$  et  $b$  de même taille  $w \times h$  par :

$$L_1(a, b) = \sum_{\substack{0 \leq i < h \\ 0 \leq j < w}} |a(i, j) - b(i, j)|.$$

**Q 27.** Écrire une fonction d'entête

```
def L1(a:image, b:image) -> int:
```

qui calcule la distance  $L_1$  entre deux images de même taille, en prenant garde aux dépassements de capacité.

**Q 28.** Écrire une fonction d'entête

```
def choixVignette(pavé:image, vignettes:[image]) -> int:
```

qui prend en paramètre une image correspondant à un pavé et une liste de vignettes et qui renvoie l'indice  $i$  tel que  $L_1(\text{pavé}, \text{vignettes}[i])$  est minimal (ou l'un d'entre eux si plusieurs vignettes conviennent). Cette fonction ne doit pas modifier la liste des vignettes.

***IV.B – Méthode sans restriction du choix des vignettes***

**Q 29.** Écrire, à l'aide de ce qui précède, une fonction d'entête

```
def construireMosaïque(source:image, vignettes:[image], p:int) -> image:
```

qui construit une photomosaïque homothétique de `source` comportant  $p$  vignettes par côté.

**Q 30.** Déterminer sa complexité temporelle asymptotique en fonction de la taille  $n = hw$  des vignettes, du nombre  $r$  de vignettes dans la mosaïque et de la longueur  $q$  de la liste `vignettes`.

***IV.C – Améliorations***

Cette sous-partie demande de l'initiative de la part du candidat, qui peut être amené à définir de nouvelles variables, structures de données et fonctions. Il est demandé d'expliquer clairement la démarche utilisée, de préciser le rôle de chaque nouvelle fonction et variable introduite et de les illustrer, le cas échéant, par un schéma. Toute démarche pertinente, même non aboutie, sera valorisée. Le barème prend en compte le temps nécessaire à la résolution de cette sous-partie.

La méthode sans restriction proposée précédemment peut conduire à sélectionner répétitivement les mêmes vignettes et à mal les répartir. En particulier, une plage uniforme de l'image source conduit à l'accumulation de la même vignette dans cette zone de la photomosaïque.

**Q 31.** Proposer une stratégie de construction de photomosaïque permettant de sélectionner un maximum de vignettes différentes et, au cas où une vignette serait réutilisée, d'éviter que les différentes apparitions de la même vignette se retrouvent trop proches.

**Q 32.** Implanter cette stratégie sous la forme d'une fonction `belleMosaïque`, version améliorée de la fonction `construireMosaïque`, dont on définira les éventuels paramètres supplémentaires.

## Opérations et fonctions disponibles en Python et SQL

### Fonctions Python diverses

- `range(n)` itérateur sur les  $n$  premiers entiers ( $\llbracket 0, n - 1 \rrbracket$ ).  
`list(range(5)) → [0, 1, 2, 3, 4]`.
- `range(d, f, p)` où  $d$ ,  $f$  et  $p$  sont des entiers, itérateur sur les entiers  $(r_i = d + ip \mid r_i < f)_{i \in \mathbb{N}}$  si  $p > 0$  et  $(r_i = d + ip \mid r_i > f)_{i \in \mathbb{N}}$  si  $p < 0$ . Le paramètre  $p$  est optionnel avec une valeur par défaut de 1.  
`list(range(1, 5)) → [1, 2, 3, 4]` ; `list(range(20, 10, -2)) → [20, 18, 16, 14, 12]`.
- `s[d:f:p]` où  $s$  est une séquence et  $d$ ,  $f$  et  $p$  sont des entiers, désigne la séquence des éléments de  $s$  dont les indices correspondent à `range(d, f, p)`. Si  $s$  est d'un type de base (liste ou tuple), `s[d:f:p]` effectue une copie, si  $s$  est un tableau numpy, `s[d:f:p]` est une vue sur les éléments de  $s$  et peut être utilisé pour modifier  $s$ .  
`[0, 1, 2, 3, 4, 5][2:6:2] → [2, 4]` ; `(0, 1, 2, 3, 4, 5)[5:2:-2] → (5, 3)`.
- `random.randrange(a, b)` renvoie un entier aléatoire compris entre  $a$  et  $b-1$  inclus ( $a$  et  $b$  entiers).
- `random.random()` renvoie un nombre flottant tiré aléatoirement dans  $[0, 1[$  suivant une distribution uniforme.
- `random.choice(s)` renvoie un élément pris au hasard dans la séquence non vide  $s$ .
- `random.shuffle(L)` permute aléatoirement les éléments de la liste  $L$  (modifie  $L$ ).
- `random.sample(s, n)` renvoie une liste constituée de  $n$  éléments distincts de la séquence  $s$  choisis aléatoirement, si  $n \leq \text{len}(s)$  lève l'exception `ValueError`.
- `math.sqrt(x)` calcule la racine carrée du nombre  $x$ .
- `round(n)` arrondit le nombre  $n$  à l'entier le plus proche. Le résultat est de type `int` pour les types numériques de base. Pour les types de la bibliothèque numpy, le résultat a le même type que l'argument.
- `math.floor(x)` renvoie le plus grand entier inférieur ou égal à  $x$ .
- `math.ceil(x)` renvoie le plus petit entier supérieur ou égal à  $x$ .

### Opérations sur les listes

- `len(L)` donne le nombre d'éléments de la liste  $L$ .
- `L1 + L2` construit une liste constituée de la concaténation des listes  $L1$  et  $L2$ .
- `n * L` construit une liste constituée de la liste  $L$  concaténée  $n$  fois avec elle-même.
- `e in L` et `e not in L` déterminent si l'objet  $e$  figure dans la liste  $L$ . Cette opération a une complexité temporelle en  $O(\text{len}(L))$ .  
`2 in [1, 2, 3] → True` ; `2 not in [1, 2, 3] → False`.
- `L.append(e)` ajoute l'élément  $e$  à la fin de la liste  $L$ .
- `L.pop(i)` : renvoie l'élément à l'indice  $i$  de la liste  $L$  et le supprime de la liste.
- `L.remove(e)` supprime de la liste  $L$  le premier élément qui a pour valeur  $e$ , s'il existe. Cette opération a une complexité temporelle en  $O(\text{len}(L))$ .
- `L.insert(i, e)` insère l'élément  $e$  à la position d'indice  $i$  dans la liste  $L$  (en décalant les éléments suivants) ; si  $i \geq \text{len}(L)$ ,  $e$  est ajouté en fin de liste.
- `L.sort()` trie en place la liste  $L$  (qui est donc modifiée) en réordonnant ses éléments dans l'ordre croissant.

### Opérations sur les tableaux (np.ndarray)

- `np.array(s, dtype)` crée un nouveau tableau contenant les éléments de la séquence  $s$ . La taille de ce tableau est déduite du contenu de  $s$ . Le paramètre `dtype` précise le type des éléments du tableau créé.
- `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un tableau à une dimension de  $n$  éléments et un tableau à  $n$  lignes et  $m$  colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype`. Si le paramètre `dtype` n'est pas précisé, il prend la valeur `float`.
- `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens.
- `np.full(n, v, dtype)`, `np.full((n, m), v, dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur  $v$ .
- `a.ndim` nombre de dimensions du tableau  $a$ .

- `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions.
- `len(a)` taille du tableau `a` dans sa première dimension, équivalent à `a.shape[0]`.
- `a.size` nombre total d'éléments du tableau `a`.
- `a.dtype` type des éléments du tableau `a`.
- `a.flat` itérateur sur tous les éléments du tableau `a`.
- `np.ndenumerate(a)` itérateur sur tous les couples (`ind, v`) du tableau `a` où `ind` est un tuple de `a.ndim` entiers donnant les indices de l'élément `v`.
- `a.min()`, `a.max()` renvoie la valeur du plus petit (respectivement plus grand) élément du tableau `a` ; ces opérations ont une complexité temporelle en  $O(a.size)$ .
- `a.sum()` ou `np.sum(a)` calcule la somme de tous les éléments du tableau `a` ; cette opération a une complexité temporelle en  $O(a.size)$ .
- `a.sum(d)` ou `np.sum(a, d)` effectue la somme des éléments du tableau `a` suivant la dimension `d` ; le résultat est un nouveau tableau avec une dimension de moins que `a`.  
`a.sum(0)` → somme par ligne, `a.sum(1)` → somme par colonne, etc.
- `a.mean()` ou `np.mean(a)` renvoie la valeur moyenne de tous les éléments du tableau `a` ; le résultat est de type `np.float64`. Cette opération a une complexité temporelle en  $O(a.size)$ .
- `a.mean(d)` ou `np.mean(a, d)` effectue la moyenne des éléments du tableau `a` suivant la dimension `d` ; le résultat est un nouveau tableau avec une dimension de moins que `a`.  
`a.mean(0)` → moyenne par ligne, `a.mean(1)` → moyenne par colonne, etc.
- `a.round()`, `np.around(a)` crée un nouveau tableau de même forme et type que `a` en arrondissant ses éléments à l'entier le plus proche.

### *SQL*

- `T1 JOIN T2 USING (c1, c2, ...)` joint les deux tables `T1` et `T2` sur les colonnes `c1, c2...` qui doivent exister dans les deux tables ; équivalent à `T1 JOIN T2 ON T1.c1 = T2.c1 AND T1.c2 = T2.c2 AND ...`, sauf que les colonnes `c1, c2...` n'apparaissent qu'une fois dans le résultat.
- Les requêtes
  - (`SELECT ... FROM ... WHERE ...`) `INTERSECT` (`SELECT ... FROM ... WHERE ...`)
  - (`SELECT ... FROM ... WHERE ...`) `UNION` (`SELECT ... FROM ... WHERE ...`)
  - (`SELECT ... FROM ... WHERE ...`) `EXCEPT` (`SELECT ... FROM ... WHERE ...`)
 sélectionnent respectivement l'intersection, l'union et la différence des résultats des deux requêtes, qui doivent être compatibles : même nombre de colonnes et mêmes types.
- `EXTRACT(part FROM t)` extrait un élément de `t`, expression de type `date`, `time`, `timestamp` (jour et heure) ou `interval` (durée). `part` peut prendre les valeurs `year`, `month`, `day` (jour dans le mois), `doy` (jour dans l'année), `dow` (jour de la semaine), `hour`, etc.
- Les fonctions d'agrégation `SUM(e)`, `AVG(e)`, `MAX(e)`, `MIN(e)`, `COUNT(e)`, `COUNT(*)` calculent respectivement la somme, la moyenne arithmétique, le maximum, le minimum, le nombre de valeurs non nulles de l'expression `e` et le nombre de lignes pour chaque groupe de lignes défini par la cause `GROUP BY`. Si la requête ne comporte pas de clause `GROUP BY` le calcul est effectué pour l'ensemble des lignes sélectionnées par la requête.

• • • FIN • • •

## Détection d'obstacles par un sonar de sous-marin

### Partie I - Introduction

Pour détecter des obstacles sous l'eau, en l'absence de visuel direct, les sous-marins sont équipés de sonars. L'onde ultrasonore émise par le sonar est réfléchie sur l'obstacle et le signal en retour est détecté et analysé pour obtenir la distance de l'obstacle au sous-marin. Une des difficultés rencontrée lors de l'analyse du signal de retour est de déterminer si l'obstacle est une roche ou un objet métallique donc un danger potentiel.

Il existe des techniques d'aide à la décision faisant partie des algorithmes dits d'Intelligence Artificielle permettant d'analyser le signal de retour d'un sonar et de déterminer de quelle nature est l'obstacle.

#### Objectif

L'objectif du travail proposé est de découvrir des algorithmes d'Intelligence Artificielle en s'appuyant sur ce problème de détection d'obstacle. À partir d'une base de données comportant des mesures de sonar réalisées sur des roches et sur du métal, il s'agira d'être capable de déterminer à partir d'une mesure inconnue, si l'objet situé devant le sous marin est une roche ou un objet métallique.

Le sujet abordera les points suivants :

- analyse et représentation des données,
- construction des arbres de décisions,
- prédiction par la méthode « random forest ».

Dans tout le sujet, il sera supposé que les modules python `numpy`, `matplotlib.pyplot` sont déjà importés dans le programme (cf. Annexe), on utilisera donc les fonctions de ces modules sans préciser l'origine de celles-ci. Lorsqu'il est demandé de définir des vecteurs ou des matrices, il est demandé d'utiliser le type array du module `numpy`.

### Partie II - Analyse des données

#### II.1 - Présentation

Les données utilisées pour l'apprentissage de l'algorithme ont été obtenues suite à une campagne d'essais réalisés dans l'océan à partir des mesures d'un sonar situé à environ 10 mètres d'un cylindre métallique ou d'un cylindre en pierre pour différentes incidences angulaires du signal émis par le sonar.

Le signal émis par le sonar et le signal reçu sont numériques et représentés avec  $N$  valeurs sur une durée totale  $T_f$ . Les variables  $T_f$  et  $N$  sont définies dans le programme.

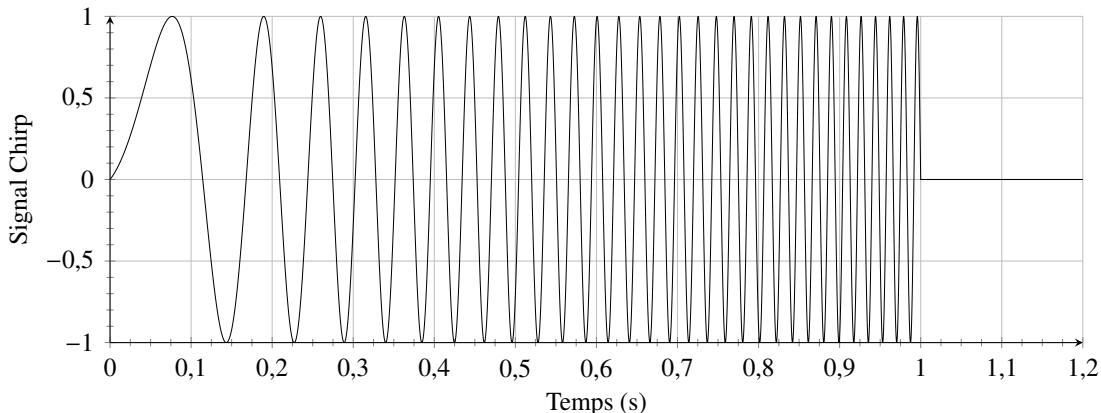
**Q1.** Écrire une instruction permettant de définir la variable `temps`, vecteur décrivant les instants de 0 à  $T_f$  uniformément répartis (exclu).

Le signal émis par le sonar (appelé CHIRP) est un signal modulé en fréquence linéairement en partant d'une fréquence  $f_0$  jusqu'à la fréquence  $f_1$  (**figure 1**), ceci permet d'atteindre de grandes portées et une bonne réception du signal lors du retour de l'onde malgré les bruits de mesure.

L'expression du signal émis est la suivante :

$$\begin{cases} e(t) = E_0 \sin(2\pi f_e(t)t) & \text{si } 0 \leq t \leq T \\ 0 & \text{si } T < t < T_f \end{cases}$$

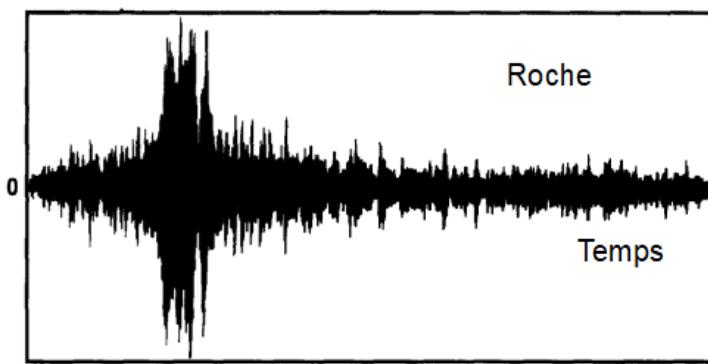
avec  $f_e(t) = f_0 + \frac{t}{T}\Delta f_e$  où  $f_0$  est la fréquence porteuse et  $\Delta f_e$  la bande de fréquences de modulation.



**Figure 1** - Exemple de signal émis de type CHIRP

- Q2.** Écrire une fonction `chirp(temps, f0, Deltafe, T, E0)` qui renvoie un vecteur de taille N correspondant au signal émis défini sur  $[0, T_f]$  avec `temps` le vecteur défini précédemment et `f0, Deltafe, T, E0` les paramètres intervenant dans le système d'équations définissant  $e(t)$ .

La réponse obtenue après réflexion sur un obstacle est de la forme donnée sur la **figure 2**.



**Figure 2** - Exemple de signal de retour (extrait de travaux de recherche)

Pour analyser ce signal et notamment savoir sur quel type de support a eu lieu la réflexion, une transformation de Fourier (discrète) est pertinente car elle permet de connaître les composantes fréquentielles importantes dans le signal. Cependant, compte tenu des bruits et de la localisation du signal de retour, on préfère utiliser une transformation de Fourier dite locale qui consiste à appliquer la transformée de Fourier non pas sur tout l'intervalle de temps d'étude mais uniquement sur plusieurs courtes périodes temporelles qui se chevauchent.

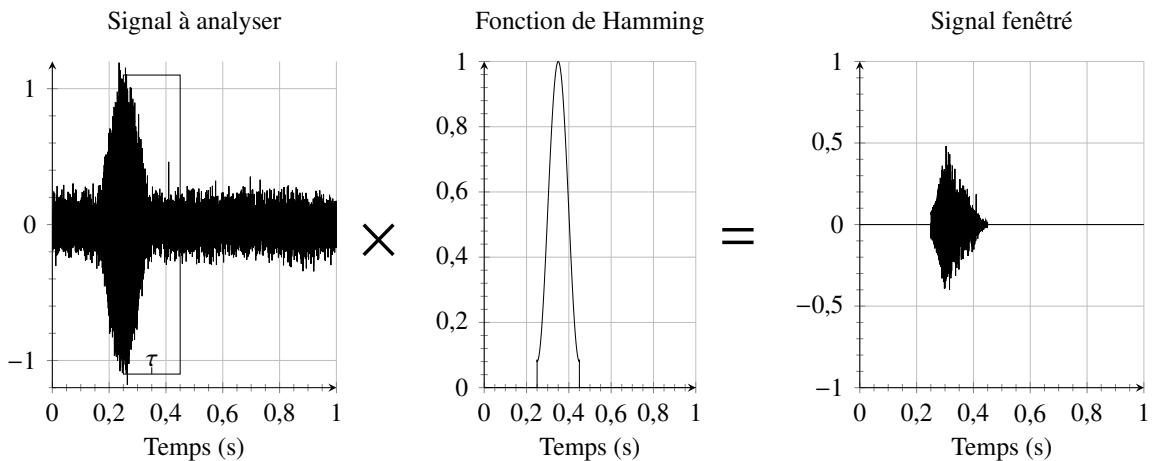
### Description de l'algorithme de transformée de Fourier locale

On note  $s_r(t)$  le signal temporel de retour mesuré après réflexion sur l'obstacle. Le signal numérique correspondant est représenté par un vecteur de taille  $N$  noté  $s$ . La période d'échantillonnage de ce signal est  $t_e = \frac{T_f}{N}$ .

Pour obtenir les intervalles de temps utilisés dans la méthode, on sélectionne un instant particulier ( $\tau$ ) et  $n < N$  instants. Ces instants seront répartis autour de la valeur  $\tau$  ( $n//2$  avant et  $n//2$  après). Pour extraire la portion de signal intéressante, on multiplie  $s_r(t)$  par une fonction particulière qui peut être un simple créneau centré autour de  $\tau$  ou des fonctions mathématiquement plus intéressantes comme celle de la **figure 3** (fonction de Hamming).

En appliquant la transformée de Fourier (FFT) au signal obtenu, on extrait un vecteur de fréquences de taille  $n_f = N//(n//2)$  et un vecteur de nombres complexes (de taille  $n_f$ ) dont le module correspond au spectre de Fourier. On répète le processus en décalant l'intervalle de temps sélectionné de  $n//2$  valeurs (pour un chevauchement de 50 %).

Finalement, on obtient alors une fonction discrète de deux variables  $S(f, t)$  où  $f$  est un vecteur de fréquences de taille  $n_f$  (le même pour toutes les fenêtres),  $t$  est un vecteur contenant les valeurs de  $\tau$  retenues de taille  $n_f$  et  $S$  l'ensemble des modules pour chaque fréquence et chaque instant  $\tau$ . Le tracé du module de  $S$  en fonction des deux vecteurs  $f$  et  $t$  est appelé spectrogramme (**figure 4**).



**Figure 3** - Principe de la transformée de Fourier locale d'un signal : sélection d'une partie du signal autour de  $t = \tau$  en multipliant le signal à analyser par une fonction de Hamming

La fonction `stft` utilisée pour réaliser la transformée de Fourier locale est disponible dans le module `scipy.signal`. Un extrait de la documentation est proposé sur la page suivante.

- Q3.** À partir des indications précédentes et de la documentation, donner l'instruction permettant de stocker dans les variables notées  $f$ ,  $t$ ,  $S$  le résultat de la transformée de Fourier discrète du signal numérique  $s$  en précisant bien les arguments retenus, sachant que l'on souhaite un recouvrement de 50 %, que le nombre de valeurs retenues autour de chaque instant est  $n$  et qu'on choisit une fenêtre de type Hamming. Donner également la taille des grandeurs  $f$ ,  $t$ ,  $S$  obtenues en retour en fonction de  $n$  et  $N$ .

**Description de la fonction stft**

```
scipy.signal.stft(x, fs=1.0, window='hann', nperseg=256, noverlap=None)
```

Compute the Short Time Fourier Transform (STFT).

STFTs can be used as a way of quantifying the change of a nonstationary signal's frequency and phase content over time.

Parameters :

**x** : array\_like

Time series of measurement values.

**fs** : float

Sampling frequency of the x time series. This value is used to define array of frequencies which length is equal to x length // (nperseg//2). Defaults to 1.0.

**window** : str

Desired window to use. If window is a string, it is passed to get\_window to generate the window values. Available windows are : boxcar, triang, hamming, hann, kaiser... Defaults to a Hann window.

**nperseg** : int

Length of each window. Defaults to 256.

**noverlap** : int, optional

Number of points to overlap between segments.

If None, nooverlap = nperseg // 2.

Returns :

**f** : ndarray

Array of sample frequencies.

**t** : ndarray

Array of segment times which length is equal to x length // (nperseg//2).

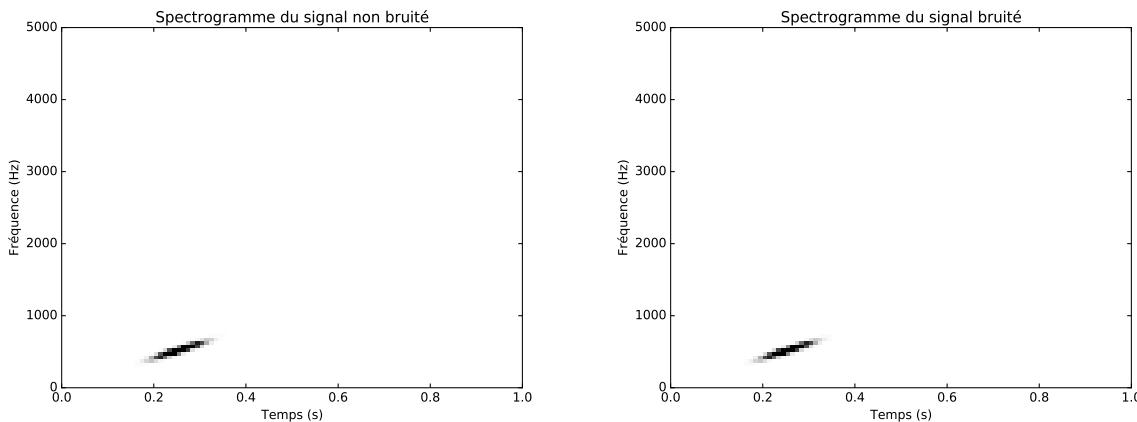
**S** : ndarray

STFT of x.

On teste la fonction sur le signal à analyser de la **figure 3**. Ce signal, qui a la même allure qu'une mesure expérimentale, a été construit à partir d'une fonction mathématique à laquelle un bruit a été rajouté.

Les spectrogrammes obtenus sont donnés sur la **figure 4**.

- Q4.** Commenter l'intérêt du spectrogramme pour analyser le contenu fréquentiel du signal d'origine et analyser succinctement la répartition obtenue entre  $f$  et  $t$ .



**Figure 4 - Spectrogramme d'un signal bruité et d'un signal non bruité**

Pour pouvoir comparer les résultats d'essais entre eux, il est préférable d'extraire l'enveloppe des spectrogrammes. Pour cela, on réalise une intégration numérique du signal tracé sur le spectrogramme :  $P(\eta) = \int_t \int_f S(t, f) w(t, f, \eta) df dt$  avec  $S(t, f)$  la fonction issue de l'analyse par transformée de Fourier locale et  $w$  une fonction représentant un ensemble de fenêtres localisées autour de la zone d'intérêt dans le spectrogramme (de tailles  $\Delta T$  et  $\Delta f$ ).  $\eta$  est un entier variant de 0 à  $m - 1$ , permettant d'obtenir un ensemble fini de  $m$  valeurs de l'enveloppe ( $m$  est un paramètre de l'analyse qui sera choisi par la suite).

On note  $t_\eta = \frac{\Delta T}{2} + \eta T$  et  $f_\eta = \frac{\Delta f}{2} + \eta \Delta f$ .

La fonction  $w(t, f, \eta)$  est définie par :

$$\begin{cases} w(t, f, \eta) = 1 & \text{si } -\Delta T/2 < t - t_\eta < \Delta T/2 \text{ et } -\Delta f/2 < f - f_\eta < \Delta f/2 \\ w(t, f, \eta) = 0 & \text{sinon} \end{cases}$$

- Q5.** Écrire une fonction  $w(t, f, \eta)$  qui renvoie 1 ou 0 en fonction des valeurs de  $t$ ,  $f$  et  $\eta$ .

On supposera que les paramètres  $\Delta T$ ,  $T$ , et  $\Delta f$  sont connus et utilisables directement dans la fonction (variables globales notées  $DT$ ,  $T$  et  $Df$ ).

On note  $t_i$  les valeurs du temps,  $i$  variant de 0 à  $n_f - 1$  et  $f_j$  les valeurs de fréquences,  $j$  variant de 0 à  $n_f - 1$  avec  $t_i = i dt$ ,  $f_j = j df$  où  $dt$  et  $df$  sont respectivement des pas de temps et de fréquence. La fonction à intégrer est notée  $S_{ij}$  avec  $i$  et  $j$  variant respectivement de 0 à  $n_f - 1$  et 0 à  $n_f - 1$ . Pour intégrer numériquement la fonction discrète  $S_{ij}$ , on utilise la formule :  $P_\eta = \sum_i \sum_j p_{ij} S_{ij} w(t_i, f_j, \eta) df dt$  avec  $p_{ij}$  poids (valeurs connues).

- Q6.** Écrire une fonction  $enveloppe(\eta, S, p, dt, df)$  qui renvoie la valeur de l'intégrale numérique en fonction de la valeur de  $\eta$ . On supposera connus les pas  $dt$  et  $df$ . Le tableau numpy  $S$  contient les valeurs de la fonction discrète  $S_{ij}$  et le tableau numpy  $p$  contient les valeurs des poids  $p_{ij}$ .

Pour chaque expérience (fonction de l'obstacle et de l'angle d'incidence de l'onde émise), on construit cette enveloppe. En prenant  $m = 60$ , on obtient 60 valeurs qui caractérisent une expérience. Ces valeurs seront utilisées dans les algorithmes de prédiction décrits dans la suite pour classer une nouvelle mesure.

De manière à pouvoir comparer les enveloppes, on normalise le vecteur  $P_\eta$  en ramenant le maximum à 1 et le minimum à 0 par une fonction affine.

- Q7.** Proposer une fonction `normalisation(P)` qui renvoie un vecteur de  $m$  valeurs comprises entre 0 et 1. On pourra utiliser les fonctions `min(x)` et `max(x)` avec  $x$  un vecteur.

## II.2 - Lecture des données

Les données à utiliser sont stockées dans un fichier texte où une ligne représente l'ensemble des valeurs associées à une expérience. Une expérience est caractérisée par 60 valeurs issues du traitement de la mesure (chaque valeur est un nombre réel entre 0 et 1) et un caractère donnant le groupe d'appartenance (un caractère M pour métal et R pour roche).

Premières lignes partielles du fichier, les points de suspension permettent de cacher les 56 données intermédiaires pour chaque ligne :

```
0.0200 ,0.0371 ,...,0.0090 ,0.0032 ,R
0.0453 ,0.0523 ,...,0.0052 ,0.0044 ,M
0.0262 ,0.0582 ,...,0.0095 ,0.0078 ,R
0.0100 ,0.0171 ,...,0.0040 ,0.0117 ,M
```

On souhaite lire ce fichier de données pour créer un tableau de type liste contenant des listes. Pour la dernière colonne, le caractère "R", respectivement "M", sera converti en la valeur 0.0, respectivement 1.0. La fonction suivante permet de lire le fichier de données et renvoie les données au format souhaité.

```
def lire donnees(nom_fichier):
    donnees = []
    with open(nom_fichier, 'r') as fichier:
        for ligne in fichier:
            a = ligne.split(',')
            b = []
            for i in range(len(a)-1):
                b.append(float(a[i]))
            if a[-1].strip() == "R":
                b.append(0.0)
            else:
                b.append(1.0)
            donnees.append(b)
    return donnees
```

- Q8.** À partir de la fonction `lire_donnees`, préciser la valeur de la variable `donnees` en se limitant aux deux premières lignes lues. Les fonctions usuelles sur les fichiers et chaînes de caractères et leur documentation sont rappelées dans l'**Annexe**.

Le fichier de données comporte  $n_x = 208$  enregistrements.

- Q9.** Donner la taille mémoire minimale nécessaire en octets pour stocker les données. On rappelle que Python stocke les nombres réels en format double précision par défaut.

### Partie III - Méthode des forêts aléatoires

La méthode des forêts aléatoires est une amélioration de la méthode des arbres de décision.

#### III.1 - Arbre de décision

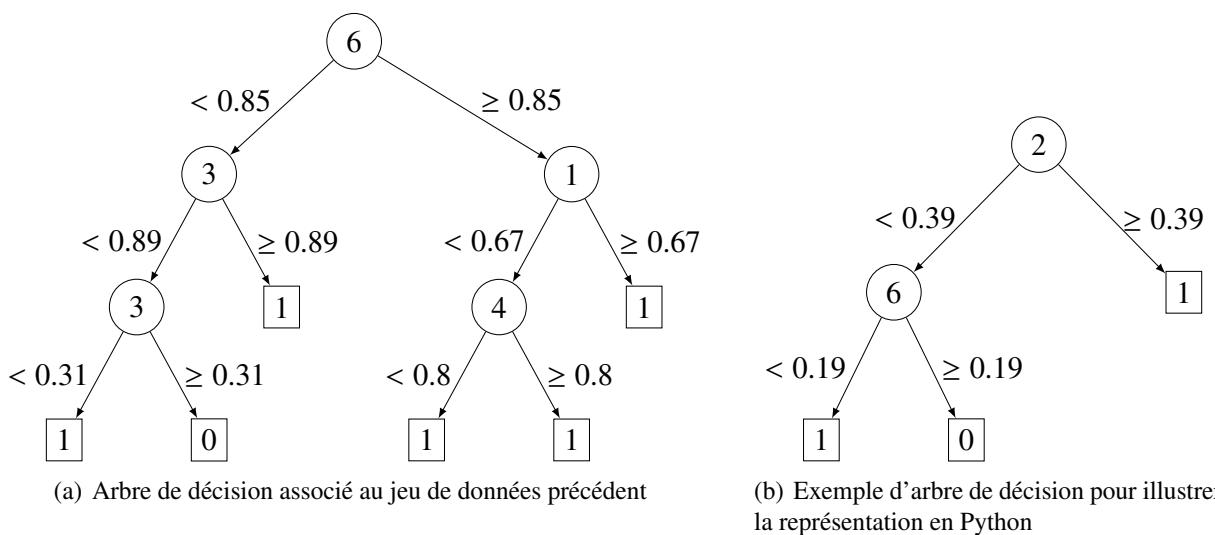
Un arbre de décision est une représentation qui permet de séparer les données en deux groupes, appelés nœuds de l'arbre, selon un critère objectif. Les sous-groupes (ou nœuds) sont ensuite séparés en deux selon un autre critère, puis on sépare à nouveau jusqu'à obtenir un nœud terminal appelé feuille.

Pour classer une nouvelle donnée, il suffit ensuite de suivre les différentes règles issues de la construction de l'arbre pour savoir dans quelle catégorie la placer.

Le principal problème rencontré est qu'un arbre de décision peut vite conduire à du surapprentissage si l'on tente de décrire parfaitement le jeu de données initial avec une donnée unique par feuille. Dans ce cas, il ne sera pas forcément possible de classer une nouvelle donnée. Il est donc souvent nécessaire d'arrêter la construction à un nombre maximal de séparations, on parle d'élaguer l'arbre.

Illustrons ces notions sur le jeu de données aléatoires suivant, la dernière colonne représentant le groupe d'appartenance :

|                                                               |
|---------------------------------------------------------------|
| 0.96, 0.95, 0.06, 0.08, 0.84, 0.74, 0.67, 0.31, 0.61, 0.61, 1 |
| 0.16, 0.43, 0.39, 0.72, 0.99, 0.95, 0.54, 0.44, 0.27, 0.04, 0 |
| 0.46, 0.32, 0.38, 0.89, 0.53, 0.56, 0.24, 0.02, 0.33, 0.14, 1 |
| 1.00, 0.67, 0.18, 0.89, 0.80, 0.73, 0.91, 0.76, 0.79, 0.35, 1 |
| 0.96, 0.16, 0.75, 0.72, 0.46, 0.53, 0.49, 0.92, 0.50, 0.83, 0 |
| 0.88, 0.90, 0.46, 0.57, 0.92, 0.72, 0.49, 0.22, 0.32, 0.70, 0 |
| 0.91, 0.27, 0.91, 0.31, 0.96, 0.71, 0.50, 0.52, 0.65, 0.59, 0 |
| 0.21, 0.51, 0.93, 0.62, 0.08, 0.82, 0.73, 0.91, 0.19, 0.74, 0 |
| 0.65, 0.27, 0.23, 0.88, 0.11, 0.52, 0.85, 0.24, 0.21, 0.88, 1 |
| 0.72, 0.03, 0.36, 0.17, 0.67, 0.08, 0.95, 0.03, 0.73, 0.02, 1 |



**Figure 5 - Différents arbres de décision**

Un arbre est représenté sur la **figure 5(a)**. Cet arbre est composé de nœuds représentés par des cercles, des liens représentés par des flèches avec condition et des feuilles représentées par des rectangles composés de 1 ou de 0.

Un nœud contient l'indice d'une des colonnes du tableau de données (les colonnes sont numérotées à partir de 0). Les liens permettent de parcourir l'arbre de décision. Pour une ligne du tableau de données, on prend la valeur de la colonne d'indice indiqué par le nœud et on évalue la condition du lien. Si la valeur est strictement plus petite que la valeur indiquée sur le lien, on descend vers la gauche, sinon on va vers la droite. Lorsqu'on ne peut aller plus loin en descendant, on aboutit à une feuille représentant le type d'obstacle détecté (0 pour roche, 1 pour métal).

En Python, on choisit de représenter un nœud par une liste de 4 éléments [`ind`, `val`, `gauche`, `droite`]. Le premier élément est l'indice de la colonne du tableau de données, le deuxième élément est la valeur permettant de faire le test pour descendre à droite ou à gauche. Le troisième élément est :

- soit le nœud de la branche de gauche (représentée elle-même par une structure de type nœud)
- soit la valeur terminale 0 ou 1 pour définir le groupe.

De même, le quatrième élément est soit le nœud de la branche de droite, soit la valeur terminale.

Par exemple, l'arbre de la **figure 5(b)** sera représenté en Python par la liste suivante :

`[2, 0.39, [6, 0.19, 1, 0], 1]`.

**Q10.** Donner la représentation en Python de l'arbre défini sur la **figure 5(a)** .

Soit une donnée non classée `a=[0.5, 0.2, 0.7, 0.4, 0.9, 0.25, 0.7, 0.7, 0.9, 0.2]`.

**Q11.** Déterminer, en justifiant, dans quel groupe cette donnée sera classée en utilisant l'arbre de la **figure 5(a)**. Expliquer le chemin parcouru dans l'arbre.

### III.2 - Construction d'un arbre

Pour construire l'arbre, la principale question à laquelle il faut répondre est de savoir comment séparer les données à chaque itération. Il existe différentes méthodes, souvent statistiques, qui permettent de réaliser cette séparation. Nous allons ici étudier l'algorithme CART (« classification and regression trees »), qui se base sur une grandeur appelée indice de concentration de Gini qui sera définie par la suite. Cet indicateur permettra de choisir la colonne et la valeur pour faire la séparation.

La fonction suivante permet de séparer les données en deux groupes en utilisant l'indice de concentration de Gini. Nous allons détailler dans la suite les fonctions intervenant dans cette fonction. Pour rappel, la variable `donnees` est un tableau constitué de  $n_x$  lignes et  $m+1$  colonnes, la dernière colonne contient la classe à laquelle appartient la ligne (qui correspond à une expérience).

```
def separe(donnees, p_var):
    # Initialisation des paramètres
    b_ind, b_val, b_gini = inf, inf, inf
    b_g, b_d = [], []
    m = len(donnees[0])-1
    # extractions d'indices aléatoires
    ind_var = indices_aleatoires(m, p_var)
    for ind in ind_var:
        for ligne in donnees:
            # séparation des données en deux groupes
            [gauche, droite] = test_separation(ind, ligne[ind], donnees)
            gini = Gini_groupes([gauche, droite])
            if gini < b_gini:
                b_ind, b_val, b_gini = ind, ligne[ind], gini
                b_g, b_d = gauche, droite
    return [b_ind, b_val, b_g, b_d]
```

La fonction renvoie l'indice de la colonne et la valeur retenus pour faire le test de séparation ainsi que les deux groupes gauche et droite (la structure identique à la valeur renvoyée par la fonction `lire_donnees` rappelée avant la question Q8).

**Q12.** Écrire une fonction `indices_aleatoires(m, p_var)` qui prend en arguments le nombre `m` correspondant au nombre de colonnes disponibles et `p_var` un nombre permettant de tirer aléatoirement `p_var` nombres parmi la liste des numéros de colonnes (`p_var < m`) et qui renvoie une liste de taille `p_var` contenant les numéros de colonnes tirés aléatoirement. Un numéro de colonne ne doit apparaître qu'une seule fois dans cette liste.

On utilisera la fonction `randrange(p)` qui renvoie un entier aléatoirement entre 0 et  $p - 1$ . Par exemple, si on choisit `p_var = 4` avec `m = 60`, on pourrait obtenir la liste d'indices aléatoires suivante [10, 50, 3, 24].

**Q13.** Écrire une fonction `[gauche,droite]=test_separation(ind, val, donnees)` qui prend en argument `ind` un numéro de colonne, `val` une valeur permettant de séparer les données et `donnees` le tableau contenant les données. Cette fonction renvoie une liste de deux éléments du même type que `donnees` : les lignes, dont la valeur de la colonne `ind` est inférieure strictement à `val`, sont stockées dans le groupe `gauche` et les autres dans le groupe `droite`. Les groupes `gauche` ou `droite` peuvent être vides.

L'indice de concentration de Gini pour un jeu de données d'un groupe `gr` (noté  $Gini_{gr}$ ) est calculé à l'aide de la formule suivante :  $Gini_{gr} = \sum_{i=0}^{k-1} 1 - p_i^2$  où `k` est le nombre de classes possibles, ici 2 (0 ou 1),  $p_i$  est la proportion des éléments du jeu de données appartenant à la classe `i`. On rappelle que la valeur de la classe est contenue dans la dernière colonne du tableau de données.

Pour obtenir l'indice de concentration de Gini total pour les deux groupes (gauche et droite), on réalise une somme des deux indices de concentration  $Gini_{gr}$  pondérée d'un coefficient de taille relative : taille du jeu de données du groupe divisé par le nombre de données des deux groupes (nombre de données totales).

**Q14.** Compléter les instructions notées 1 à 5 de la fonction `Gini_groupes` donnée sur le document réponse qui prend en argument `groupes` la liste contenant les deux groupes à tester et qui renvoie l'indice de concentration de Gini.

Lors de la construction de l'arbre, on peut fixer plusieurs critères permettant d'arrêter la construction en calculant une feuille :

- premier critère : quand le nombre de données à séparer est inférieur à une valeur que l'on notera `taille_min`,
- deuxième critère : quand le nombre de séparations a atteint une valeur maximale notée `sep_max`.

On donne à la feuille associée au jeu de données restant en fin de séparation la valeur du groupe majoritaire. Si la majorité des données est dans la classe 0 (roche) alors la feuille prendra la valeur 0, sinon elle prendra la valeur 1 (métal).

**Q15.** Écrire une fonction `feuille(data)` qui prend en argument un jeu de données `data` et qui renvoie la valeur de la classe majoritaire. La variable `data` est du même format que la variable `donnees`.

La fonction permettant de lancer la construction de l'arbre est la suivante :

```
def construire_arbre(data_train, sep_max, taille_min, p_var):
    arbre = separe(data_train, p_var)
    construit(arbre, sep_max, taille_min, p_var, 1)
    return arbre
```

avec :

- `data_train` des données dont on connaît déjà la classe ;
- `sep_max` : le nombre de séparations maximales pouvant être effectuées avant de placer une feuille ;
- `taille_min` : le nombre de données minimales sous lequel on impose de mettre une feuille plutôt que de séparer les données en deux ;
- `p_var` : le nombre de valeurs à tirer aléatoirement pour le calcul de l'indice de concentration de Gini.

La construction de l'arbre est réalisée récursivement avec la fonction `construit(arbre, sep_max, taille_min, p_var, ind_rec)` après avoir créé un arbre initial à deux branches avec la fonction `separe`.

La fonction `construit` prend en argument entre autres :

- `arbre` : structure de type noeud constituée de 4 éléments [`ind`, `val`, `gauche`, `droite`] ;
- `sep_max` ;
- `taille_min` ;
- `p_var` ;
- `ind_rec` : l'indice de récursivité donnant le nombre de séparations déjà effectuées.

La fonction `separe` peut renvoyer un groupe gauche ou droite vide ([]), cela signifie qu'il n'y a pas eu de critères permettant de séparer les données en deux groupes. Dans ce cas, il faut calculer la feuille terminale associée au groupe non vide et imposer la valeur de cette feuille à droite et à gauche. Les variables `gauche` et `droite` peuvent être des nombres 0 ou 1 (feuilles) ou des structures de type noeud (cf. question **Q10**).

Dans la fonction récursive, la variable `arbre` de type noeud est modifiée au cours des appels successifs.

**Q16.** Compléter les conditions numérotées 1 à 4 de la fonction récursive `construit` donnée sur le document réponse.

### III.3 - Test d'une prédiction sur un arbre simple

À partir des fonctions présentées précédemment, il est possible de construire un arbre de décision classique en prenant en compte toutes les colonnes disponibles pour faire les séparations. Pour le cas que l'on traite, on prendra ainsi `p_var = 60`.

Pour tester les performances de prédiction dans ce cas, on utilise un jeu de 100 données connues pour construire l'arbre. On réalise ensuite une prédiction sur un jeu de 50 données (non utilisées pour la construction mais dont on connaît la classe) : pour chaque donnée, on parcourt l'arbre jusqu'à arriver sur une feuille qui donnera le groupe prédit. On compare cette prédiction à la classe connue. On compte le nombre de succès pour en déduire un taux de réussite. On recommence cette analyse en faisant une nouvelle construction d'arbre à partir des 100 données (ce qui correspond aux prédictions notées 1, 2 et 3). On étudie également l'influence du nombre de données pour construire l'arbre en effectuant la même démarche pour des jeux de 125 et 150 données.

|                      | Arbre construit à partir de 100 données | Arbre construit à partir de 125 données | Arbre construit à partir de 150 données |
|----------------------|-----------------------------------------|-----------------------------------------|-----------------------------------------|
| Test de prédiction 1 | 79 %                                    | 74 %                                    | 88 %                                    |
| Test de prédiction 2 | 76 %                                    | 78 %                                    | 84 %                                    |
| Test de prédiction 3 | 74 %                                    | 78 %                                    | 77 %                                    |
| Temps moyen          | 0,42 s                                  | 0,64 s                                  | 0,86 s                                  |

**Tableau 1** - Pourcentages de réussite obtenus et temps de prédiction

Le **tableau 1** liste une synthèse des pourcentages de réussite obtenus ainsi que le temps pour réaliser une prédiction.

**Q17.** Au vu de ces quelques résultats d'analyse de la méthode des arbres de décision, indiquer de quels problèmes semblent souffrir cette méthode.

### III.4 - Algorithme des forêts aléatoires : « random forest »

Pour palier les problèmes observés précédemment, on utilise un algorithme des forêts aléatoires.

L'idée de l'algorithme des forêts aléatoires est de construire plusieurs arbres de décision (`n_arbres`) basés sur une vision partielle du problème en se limitant à quelques variables (d'où l'introduction de la fonction `indice_aleatoire` dans la partie précédente). En pratique, on utilise la racine carrée du nombre de variables, soit 7 au lieu de 60 dans notre exemple.

Ensuite, on réalise une prédiction sur les différents arbres construits et on associe à la donnée à classer la classe majoritaire issue des différentes prédictions élémentaires.

On suppose que l'on dispose des fonctions : `construire_foret` qui renvoie une liste d'arbres (non détaillée ici) et `prediction` qui pour un arbre connu et une donnée renvoie la valeur de sa classe.

**Q18.** Compléter les 4 instructions manquantes de la fonction récursive `prediction(arbre, donnee)` donnée sur le document réponse qui prend en argument un arbre de décision noté `arbre` de type `nœud` et une donnée à classer `donnee`. La fonction `isinstance(var, type)` renvoie `True` si `var` est du type `type`.

On note :

- `data_train`, les données d'entraînement de l'algorithme qui vont servir à construire les arbres ;
- `data_test`, les données permettant de tester l'efficacité de l'algorithme en comparant le classement proposé par rapport à la valeur connue.

**Q19.** Écrire une fonction `random_forest(data_train, data_test, sep_max, taille_min, n_arbres, p_var)` qui renvoie une liste contenant la classe de chaque donnée contenue dans la variable `data_test`.

## Conclusion

On réalise des tests de prédiction comme cela a été présenté en **sous-partie III.3**.

Le **tableau 2** liste une synthèse des pourcentages de réussite obtenus ainsi que le temps pour réaliser une prédiction avec une forêt de 20 arbres.

|                      | Arbre construit à partir de 100 données | Arbre construit à partir de 125 données | Arbre construit à partir de 150 données |
|----------------------|-----------------------------------------|-----------------------------------------|-----------------------------------------|
| Test de prédiction 1 | 88 %                                    | 84 %                                    | 91 %                                    |
| Test de prédiction 2 | 91 %                                    | 86 %                                    | 91 %                                    |
| Test de prédiction 3 | 86 %                                    | 83 %                                    | 88 %                                    |
| Temps moyen          | 0,23 s                                  | 0,35 s                                  | 0,5 s                                   |

**Tableau 2** - Pourcentages de réussite et temps de prédiction pour une forêt de 20 arbres

**Q20.** Conclure sur l'intérêt de cet algorithme des forêts aléatoires.

**FIN**



**Numéro  
d'inscription**

**Numéro  
de table**

**Né(e) le**   

**Nom :** \_\_\_\_\_

**Prénom :** \_\_\_\_\_

Filière : PSI

Session : 2020

## **Épreuve de : Informatique**

- Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer
  - Rédiger avec un stylo non effaçable bleu ou noir
  - Ne rien écrire dans les marges (gauche et droite)
  - Numérotter chaque page (cadre en bas à droite)
  - Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre

Emplacement  
QR Code

PSI5IN

# DOCUMENT RÉPONSE

#### **Q1 - Instruction pour la création de la variable temps**

## Q2 - Fonction chirp(temps, f0, Deltafe, T, E0)

**Q3** - Instructions pour stocker  $f$ ,  $t$ ,  $S$ . Taille de  $f$ ,  $t$ ,  $S$  en fonction de  $n$  et  $N$

**NE RIEN ÉCRIRE DANS CE CADRE**

**Q4** - Intérêt du spectrogramme



**Q5** - Fonction  $w(t, f, \eta)$



**Q6** - Fonction `enveloppe(eta, S, p, dt, df)`



## Q7 - Fonction normalisation(P)

**Q8** - Valeur de la variable donnees (deux premières lignes lues seulement)

**Q9 - Taille mémoire minimale pour stocker les données**

**Q10** - Représentation en Python de l'arbre de la **figure 5(a)**

## **Q11 - Classification de la donnée. Justification**

**Q12** - Fonction `indices_aleatoires(m,p_var)`

**Q13** - Fonction `[gauche, droite] = test_separation(ind,val,donnees)`



**Numéro  
d'inscription** | | | | |

**Numéro  
de table**

**Né(e) le**

**Nom :** \_\_\_\_\_

**Prénom :** \_\_\_\_\_

Filière : PSI

Session : 2020

## **Épreuve de : Informatique**

- Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer
  - Rédiger avec un stylo non effaçable bleu ou noir
  - Ne rien écrire dans les marges (gauche et droite)
  - Numérotter chaque page (cadre en bas à droite)
  - Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre

Emplacement  
QR Code

PSI5IN

**Q14 - Compléter les instructions notées 1 à 5**

```

def Gini_groupes(groupes):
    #nombre de données total
    n_donnees =                                     #instruction1

    gini = 0.0 #somme pondérée des indices Gini de chaque groupe
    for donnees in groupes:
        taille = len(donnees) #taille d'un groupe
        if taille != 0:
            gini_gr = 0.0
            for val in [0,1]:
                p=0
                for ligne in donnees :
                    if ligne[-1] == val:
                        #instruction2

                #instruction3

            gini_gr +=                                     #instruction4

            #ajout de gini_gr avec le poids relatif
            gini +=   #instruction5

    return gini

```

**NE RIEN ÉCRIRE DANS CE CADRE**

**Q15 - Fonction feuille(data)**

**Q16 - Compléter les conditions numérotées 1 à 4 de la fonction récursive construit**

```
def construit(arbre, sep_max, taille_min, p_var, ind_rec):
    gauche, droite = arbre[2], arbre[3]

    if : #condition1
        valeur = feuille(gauche + droite)
        arbre[2] = valeur
        arbre[3] = valeur
        return

    if : #condition2
        arbre[2], arbre[3] = feuille(gauche), feuille(droite)

    if : #condition3
        arbre[2] = feuille(gauche)
    else:
        arbre[2] = separe(gauche, p_var)
        construit(arbre[2], sep_max, taille_min, p_var, ind_rec+1)

    if : #condition4
        arbre[3] = feuille(droite)
    else:
        arbre[3] = separe(droite, p_var)
        construit(arbre[3], sep_max, taille_min, p_var, ind_rec+1)
```

## Q17 - Problèmes de la méthode

## **Q18 - Compléter les 4 instructions manquantes**

**Q19** - Fonction random\_forest(data\_train, data\_test, sep\_max, taille\_min, n\_arbres, p\_var)

## **Q20 - Conclusion sur l'intérêt de l'algorithme des forêts aléatoires**

## ANNEXE

## Rappels des syntaxes en Python

**Remarque** : sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : `from numpy import *`. Les indices de ces tableaux commencent à 0.

|                                                                                                                                  | Python                                                          |
|----------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| tableau à une dimension                                                                                                          | L=[1,2,3] (liste)<br>v=array([1,2,3]) (vecteur)                 |
| accéder à un élément                                                                                                             | v[0] renvoie 1 (L[0] également)                                 |
| ajouter un élément                                                                                                               | L.append(5) uniquement sur les listes                           |
| séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1                                                             | arange(0,10.1,0.1)                                              |
| définir une chaîne de caractères                                                                                                 | mot='Python'                                                    |
| taille d'une chaîne                                                                                                              | len(mot)                                                        |
| extraire des caractères                                                                                                          | mot[2:7]                                                        |
| éliminer le \n en fin d'une ligne                                                                                                | ligne.strip()                                                   |
| découper une chaîne de caractères selon un caractère passé en argument. On obtient une liste qui contient les caractères séparés | mot.split(',')'                                                 |
| ouverture d'un fichier en lecture                                                                                                | with open('nom_fichier','r') as file:<br>instructions avec file |

## Autour du séquençage du génome

Dans ce sujet, on s'intéresse à la recherche d'un motif dans une molécule d'ADN.

Une molécule d'ADN est constituée de deux brins complémentaires, qui sont un long enchaînement de nucléotides de quatre types différents désignés par les lettres A, T, C et G. Les deux brins sont complémentaires : "en face" d'un 'A', il y a toujours un 'T' et "en face" d'un 'C', il y a toujours un 'G'. Pour simplifier le sujet, on va considérer qu'une molécule d'ADN est une chaîne de caractères sur l'alphabet {A,C,G,T} (on s'intéresse donc seulement à un des deux brins). On parlera de séquence d'ADN.

### Partie I - Génération d'une séquence d'ADN

On considère la chaîne de caractère seq='ATCGTACGTACG'.

**Q1.** Que renvoie la commande seq[3] ? Que renvoie la commande seq[2:6] ?

Les fonctions que nous allons construire par la suite devront prendre en paramètre une chaîne de caractères ne contenant que des 'A', 'C', 'G' et 'T' (ceci correspond à une séquence d'ADN). Nous allons commencer par construire aléatoirement une séquence d'ADN.

Pour générer aléatoirement une séquence d'ADN composée de  $n$  caractères, on utilisera le principe suivant.

1. On commence par créer une chaîne de caractères vide.
2. Puis on tire aléatoirement  $n$  chiffres compris entre 1 et 4 et
  - si on obtient un 1, alors on ajoute un 'A' à notre chaîne de caractères ;
  - si on obtient un 2, alors on ajoute un 'C' à notre chaîne de caractères ;
  - si on obtient un 3, alors on ajoute un 'G' à notre chaîne de caractères ;
  - si on obtient un 4, alors on ajoute un 'T' à notre chaîne de caractères.
3. On renvoie la chaîne de caractères ainsi construite.

**Q2.** Écrire une fonction *generation()* qui prend en paramètre un entier  $n$  et qui renvoie une chaîne de caractères aléatoires de longueur  $n$  ne contenant que des 'A', 'C', 'G' et 'T'.

On pourra utiliser les fonctions *random()* ou *randint()* détaillées en **annexe**.

**Q3.** Que fait la fonction *mystere(seq)* qui prend en argument une séquence d'ADN 'seq' (une chaîne de caractères ne contenant que des 'A', 'C', 'G' et 'T') ?

Le code de la fonction *mystere()* se trouve dans le **DR 3**.

**Q4.** Quelle est la complexité de la fonction *mystere()* ?

Donner le nom de la variable permettant de montrer la terminaison de l'algorithme (on justifiera le raisonnement).

### Partie II - Recherche d'un motif

Soit une chaîne de caractères S='ACTGGTCACT', on appelle sous-chaîne de caractères de S une suite de caractères incluse dans S. Par exemple, 'TGG' est une sous-chaîne de S mais 'TAG' n'est pas une sous-chaîne de S.

#### Objectif

Rechercher une sous-chaîne de caractères M de longueur  $m$  appelée motif dans une chaîne de caractères S de longueur  $n$ .

Il s'agit d'une problématique classique en informatique, qui répond aux besoins de nombreuses applications.

On trouve plus de 100 algorithmes différents pour cette même tâche, les plus célèbres datant des années 1970, mais plus de la moitié ont moins de 10 ans.

Dans cette **partie**, nous allons d'abord nous intéresser à l'algorithme naïf (**sous-partie II.1**), puis à deux autres algorithmes : l'algorithme de Knuth-Morris-Pratt (**sous-partie II.2**), et un algorithme utilisant une structure de liste (**sous-partie II.3**) et enfin, aux fonctions de hachage (**sous-partie II.4**).

*Les différentes sous-parties sont indépendantes.*

### II.1 - Algorithme naïf

*Principe de l'algorithme naïf*

On parcourt la chaîne. À chaque étape, on regarde si on a trouvé le bon motif. Si ce n'est pas le cas, on recommence avec l'élément suivant de la chaîne de caractères.

Cet algorithme a une complexité en  $O(nm)$  avec  $n$ , la taille de la chaîne de caractère et  $m$ , la taille du motif.

**Q5.** Écrire une fonction *recherche()* qui à une sous-chaîne de caractères M et une chaîne de caractères S renvoie -1 si M n'est pas dans S, et la position de la première lettre de la chaîne de caractères M si M est présente dans S.

Cet algorithme doit correspondre à l'algorithme naïf.

**Q6.** Combien faut-il d'opérations pour chercher un motif de 50 caractères dans une séquence d'ADN en utilisant l'algorithme naïf ? On supposera qu'une séquence d'ADN est composée de  $3 \cdot 10^9$  caractères.

En combien de temps un ordinateur réalisant  $10^{12}$  opérations par seconde fait-il ce calcul ?

En génétique, on utilise des algorithmes de recherche pour identifier les similarités entre deux séquences d'ADN. Pour cela, on procède de la manière suivante :

- découper la première séquence d'ADN en morceaux de taille 50 ;
- rechercher chaque morceau dans la deuxième séquence d'ADN.

**Q7.** En utilisant les calculs précédents, combien de temps faut-il pour un ordinateur réalisant  $10^{12}$  opérations par seconde pour comparer deux séquences d'ADN avec l'algorithme naïf ?  
Vous semble-t-il intéressant d'utiliser l'algorithme de recherche naïf ?

### II.2 - Algorithme de Knuth-Morris-Pratt (1970)

Lorsqu'un échec a lieu dans l'algorithme naïf, c'est-à-dire lorsqu'un caractère du motif est différent du caractère correspondant dans la séquence d'ADN, la recherche reprend à la position suivante en repartant au début du motif. Si le caractère qui a provoqué l'échec n'est pas au début du motif, cette recherche commence par comparer une partie du motif avec une partie de la séquence d'ADN qui a déjà été comparée avec le motif. L'idée de départ de l'algorithme de Knuth-Morris-Pratt est d'éviter ces comparaisons inutiles. Pour cela, une fonction annexe qui recherche le plus long préfixe d'un motif qui soit aussi un suffixe de ce motif est utilisée.

Avant d'étudier l'algorithme de Knuth-Morris-Pratt (**sous-partie II.2.b**) nous allons définir les notions de préfixe et suffixe (**sous-partie II.2.a**).

### **II.2.a Préfixe et suffixe**

Un préfixe d'un motif M est un motif u, différent de M, qui est un début de M.

Par exemple, 'mo' et 'm' sont des préfixes de 'mot', mais 'o' n'est pas un préfixe de 'mot' .

Un suffixe d'un motif M est un motif u, différent de M, qui est une fin de M.

Par exemple, 'ot' et 't' sont des suffixes de 'mot', mais 'mot' n'est pas un suffixe de 'mot' .

**Q8.** Donner tous les préfixes et les suffixes du motif 'ACGTAC'.

**Q9.** Quel est le plus grand préfixe de 'ACGTAC' qui soit aussi un suffixe ?

Quel est le plus grand préfixe de 'ACAACA' qui soit aussi un suffixe ?

### **II.2.b Algorithme de Knuth-Morris-Pratt**

Nous rappelons que l'algorithme de Knuth-Morris-Pratt (KMP) est une fonction de recherche qui utilise une fonction annexe prenant en argument une chaîne de caractères M dont on notera la longueur *m*.

Cette fonction annexe, appelée *fonctionannexe()*, doit permettre, pour chaque lettre à la position i, de trouver le plus grand sous-mot de M qui finit par la lettre M[i] (c'est donc le plus grand suffixe de M[ :i+1]) qui soit aussi un préfixe de M.

Le code de *fonctionannexe()* se trouve à la question **Q11 du DR 6**.

**Q10.** Quel est le type de la sortie de la fonction *fonctionannexe()* ?

**Q11.** Une ou des erreurs de syntaxe s'est (se sont) glissée(s) dans la fonction *fonctionannexe()*.

Identifier la ou les erreur(s) et corriger la fonction pour qu'il n'y ait plus de message d'erreur quand on compile la fonction.

**Q12.** Décrire l'exécution de la fonction *fonctionannexe()* lorsque M='ACAACA' en précisant sur le **DR 6**, pour les six premiers tours dans la boucle while, à la sortie de la boucle, le contenu des variables : *i*, *j* et *F*.

**Q13.** Expliquer et commenter les groupements de lignes de l'algorithme KMP donnés dans le **DR 7**.

### **II.3 - Algorithme utilisant la structure de liste**

Une autre possibilité pour chercher un motif dans une chaîne de caractères (ou séquence d'ADN) est de construire une liste contenant tous les sous-motifs de notre chaîne, triés par ordre alphabétique, puis de faire la recherche dans cette liste.

Par exemple, à la chaîne 'CATCG', on peut lui associer la liste :

[ 'C', 'A', 'T', 'G', 'CA', 'AT', 'TC', 'CG', 'CAT', 'ATC', 'TCG', 'CATC', 'ATCG', 'CATCG' ] que l'on peut ensuite trier pour obtenir la liste :

[ 'A', 'AT', 'ATC', 'ATCG', 'C', 'CA', 'CAT', 'CATC', 'CATCG', 'CG', 'G', 'T', 'TC', 'TCG' ].

La première étape de cette méthode est donc de trier une liste.

**Q14.** Écrire une fonction *triinsertion()* de tri par insertion d'une liste de nombres.

**Q15.** Comment peut-on adapter la fonction *triinsertion()* à une liste de chaîne de caractères ?

Après avoir obtenu une liste triée, on peut faire une recherche dichotomique dans cette nouvelle liste.

**Q16.** Écrire une fonction *recherchedichotomique()* de recherche dichotomique dans une liste de nombres triés.

Quel est l'intérêt de ce type d'algorithmes (on parlera de complexité) ?

## II.4 - Fonction de hachage et évaluation de polynôme

### II.4.a Fonction de hachage, algorithme de Karp-Rabin

Certains algorithmes, comme l'algorithme de Karp-Rabin (1987), utilisent une fonction de hachage  $h$  qui à un motif renvoie une valeur numérique.

Voici un exemple de fonction de hachage :

- à chaque caractère de l'alphabet, on associe une valeur. Ici, on va associer à 'A' la valeur 0, à 'C' la valeur 1, à 'G' la valeur 2 et à 'T' la valeur 3. Pour un motif de taille  $n$ , on obtient donc une suite de chiffre  $a_{n-1} \dots a_1 a_0$ . Par exemple, à la chaîne 'TAGC', on lui associe la suite de chiffre 3021 ;
- cette suite de chiffre est considérée comme l'écriture d'un entier en base  $b$ , où  $b$  est le nombre de caractères présents dans l'alphabet. On a donc ici  $b = 4$  ;
- on calcule ensuite cet entier en base 10 (on calcule donc  $a_{n-1}b^{n-1} + \dots + a_1b^1 + a_0b^0$ ) ;
- puis on calcule le reste de la division euclidienne de ce nombre par 13.

**Q17.** On ne considère que des motifs de taille 3. Que renvoie la fonction de hachage avec les motifs 'CCC', 'ACG', 'GAG' ? On détaillera les calculs.

Dans cette fonction de hachage, nous avons besoin de transformer un entier en base  $b$  en un entier en base 10. On remarque que l'on peut éventuellement faire ce calcul en évaluant un polynôme.

### II.4.b Évaluation de polynôme, Algorithme de Hörner

Dans cette **sous-partie**, nous allons nous intéresser à l'évaluation d'un polynôme et de son coût lorsque l'on compte les multiplications, les additions et les affectations comme des opérations unitaires.

Soit  $P = \sum_{k=0}^n a_k X^k$  un polynôme, il sera représenté par la liste  $[a_n, \dots, a_0]$ .

**Q18.** Écrire une fonction *eval()* ayant pour paramètre un polynôme  $P$  (donc une liste de nombres  $[a_n, \dots, a_0]$ ) et un nombre  $b$ . Cette fonction doit renvoyer la valeur de  $P$  en  $b$ , c'est-à-dire calculant :

$$P(b) = \sum_{k=0}^n a_k b^k.$$

En admettant que le calcul de  $b^k$  utilise  $k - 1$  multiplications, on trouve une complexité quadratique. On peut être plus astucieux en utilisant l'algorithme de Hörner qui se base sur l'égalité suivante :

$$P(X) = (((\dots((a_nX + a_{n-1})X + a_{n-2})X + \dots)X + a_1)X + a_0).$$

Plus précisément, pour évaluer  $P$  en  $b$ , on commence par calculer  $a_n \times b + a_{n-1}$ , puis on multiplie le résultat par  $b$  et on ajoute  $a_{n-2}$ , etc. On trouvera alors une complexité linéaire.

**Q19.** Écrire une fonction itérative *hornerit()* ayant pour paramètres un polynôme  $P$ , sous forme de liste, ainsi qu'un réel  $b$ , et renvoyant  $P(b)$  en utilisant l'algorithme de Hörner.

**Q20.** Compléter la fonction *hornerrec()* pour avoir une fonction récursive qui évalue un polynôme en utilisant l'algorithme de Hörner.

### Partie III - Collection Française de Bactéries Phytopathogènes

La Collection Française de Bactéries Phytopathogènes (CFBP) possède deux bases de données :

- l'une, appelée Echantillon, qui permet de stocker les différents échantillons d'ADN ;
- l'autre, appelée Sequence, qui permet de mémoriser quelle personne est responsable de l'obtention de la séquence (i.e. de l'extraction d'un gène particulier dans les différents échantillons d'ADN).

Des extraits des tables Echantillon et Sequence sont données par les **tableaux 1 et 2**.

| Echantillon |             |          |              |
|-------------|-------------|----------|--------------|
| ADN         | Genre       | Espèce   | Sous-espèce  |
| 309         | Pseudomonas | syringae | morsprunorum |
| ...         |             |          |              |
| 3589        | Pseudomonas | syringae | vignae       |
| ...         |             |          |              |

**Tableau 1** - Table recensant toutes les séquences d'ADN dont dispose la CFBP

| Sequence |              |      |      |                     |         |
|----------|--------------|------|------|---------------------|---------|
| Code     | Date         | ADN  | Gène | Protocole           | Employé |
| A        | '01-03-2018' | 309  | gyrB | Spilker             | Dupont  |
| B        | '01-03-2018' | 2028 | recA | Cesbron and Manceau | Martin  |
| ...      | ...          | ...  | ...  | ...                 | ...     |
| AGZ      | '10-03-2018' | 2028 | leuS | Deletooste          | Martin  |
| ...      | ...          | ...  | ...  | ...                 | ...     |

**Tableau 2** - Table recensant tous les travaux réalisés à la CFBP en mars 2018

**Q21.** Définir le but et le résultat de la requête(1), écrite en SQL :

SELECT count(\*) FROM Sequence WHERE Date='01-03-2018'

**Q22.** Écrire en SQL la requête(2), donnée en algèbre relationnel :

$$\pi_{\text{ADN}}(\sigma_{\text{Gène} = \text{'leuS'}}(\text{Sequence})).$$

**Q23.** Écrire en SQL la requête(3) qui permet d'obtenir la liste des espèces étudiées par M. Martin le 10 mars 2018.

**Q24.** Écrire en SQL la requête(4) permettant d'obtenir le nombre d'échantillons prélevés par chaque employé.

**ANNEXE - Librairie numpy**

*random()* : génère un nombre pseudo-aléatoire compris entre 0 et 1.

*randint(a,b)* : génère un entier aléatoire tel que  $a \leq \text{randint}(a,b) < b$ .

**FIN**





**Numéro d'inscription**

**Nom :** \_\_\_\_\_

**Numéro de table**

**Prénom :** \_\_\_\_\_

**Né(e) le**

Emplacement  
QR Code

**Filière : TSI**

**Session : 2020**

### Épreuve de : Informatique

- Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer
- Réddiger avec un stylo non effaçable bleu ou noir
- Ne rien écrire dans les marges (gauche et droite)
- Numérotter chaque page (cadre en bas à droite)
- Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre

#### Consignes

J. 20 1173

## AUTOUR DU SÉQUENÇAGE DU GÉNOME

### DOCUMENT RÉPONSE

#### Q1

```
1 seq='ATCGTACGTACG'
2 seq[3]
```

Que renvoie cette commande Python ?

.....

```
1 seq='ATCGTACGTACG'
2 seq[2:6]
```

Que renvoie cette commande Python ?

.....

NE RIEN ÉCRIRE DANS CE CADRE

**Q2**

```
1 def generation(n):  
2     seq =
```

**Q3**

```
1 def mystere(seq):
2     a,b,c,d = 0,0,0,0
3     i = len(seq)-1
4     while i>= 0:
5         if seq[i]=='A':
6             a += 1
7             i -= 1
8         elif seq[i]=='C':
9             b += 1
10            i -= 1
11        elif seq[i]=='G':
12            c += 1
13            i -= 1
14        else:
15            d += 1
16            i -= 1
17    return [a*100/len(seq),b*100/len(seq),c*100/len(seq),d*100/len(seq)]
```

Réponse : .....  
.....

**Q4**

Complexité : .....

Terminaison : nom de la variable : .....

justification : .....

.....  
.....  
.....  
.....

**Q5**

1   **def** recherche(M,T):

**Q6**

Nombre d'opérations pour chercher un motif de 50 caractères : .....

Temps mis par l'ordinateur : .....

**Q7**

Temps pour comparer deux séquences d'ADN : .....

Est-ce intéressant d'utiliser cette méthode ? .....



Numéro  
d'inscription

Numéro  
de table

Né(e) le

Nom : \_\_\_\_\_

Prénom : \_\_\_\_\_

Emplacement  
QR Code

Filière : TSI

Session : 2020

Épreuve de : Informatique

- Consignes**
- Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer
  - Rédiger avec un stylo non effaçable bleu ou noir
  - Ne rien écrire dans les marges (gauche et droite)
  - Numérotter chaque page (cadre en bas à droite)
  - Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre

J. 20 1173

Q8

Préfixes de 'ACGTAC' : .....  
.....

Suffixes de 'ACGTAC' : .....  
.....

Q9

Plus grand préfixe de 'ACGTAC' qui soit aussi un suffixe : .....

Plus grand préfixe de 'ACAACA' qui soit aussi un suffixe : .....

Q10

Type de la variable en sortie : .....

**NE RIEN ÉCRIRE DANS CE CADRE**

### **Q11**

Corriger la fonction suivante pour qu'il n'y ait plus de message d'erreur quand on compile la fonction.

```

1   def fonctionannexe(M):
2       F=[0]
3       i=1
4       j=0
5       while i < m :
6           if M[i]==M[j] :
7               F.append(j+1)
8               i=i+1
9               j=j+1
10      else
11          if j>0 :
12              j=F[j-1]
13          else:
14              F.append(0)
15              i=i+1
16      return F

```

### **Q12**

Initialisation : i=1 ; j=0 ; F=[0]

Fin du premier passage dans la boucle while : i= ... ; j= ... ; F= .....;

Fin du deuxième passage dans la boucle while : i= ... ; j= ... ; F= .....;

Fin du troisième passage dans la boucle while : i= ... ; j= ... ; F= .....;

Fin du quatrième passage dans la boucle while : i= ... ; j= ... ; F= .....;

Fin du cinquième passage dans la boucle while : i= ... ; j= ... ; F= .....;

Fin du sixième passage dans la boucle while : i= ... ; j= ... ; F= .....;

### Q13

Algorithme KMP :

```
1  def KMP(M,T):
2      F=fonctionannexe(M)
3      i=0
4      j=0
5      while i < len(T) :
6          if T[i]==M[j]:
7              if j==len(M)-1:
8                  return(i-j)
9              else:
10                 i=i+1
11                 j=j+1
12             else:
13                 if j > 0:
14                     j=F[j-1]
15                 else:
16                     i=i+1
17     return -1
```

Explication de la ligne 2 : .....

.....

Explication des lignes 3 et 4 : .....

.....

Quelles lignes correspondent au cas où on a trouvé le mot ? .....

Que fait le programme dans ce cas ? .....

.....

Quelles lignes correspondent au cas où on a trouvé deux lettres identiques ? .....

.....

Que fait le programme dans ce cas ? .....

.....

Quelles lignes correspondent au cas où on a trouvé deux lettres différentes ? .....

.....

Que fait le programme dans ce cas ? .....

.....

.....

**Q14**

```
1 def triinsertion(L):
```

**Q15**

```
.....  
.....  
.....  
.....
```



**Numéro d'inscription**

**Nom :** \_\_\_\_\_

**Numéro de table**

**Prénom :** \_\_\_\_\_

**Né(e) le**

Emplacement  
QR Code

**Filière : TSI**

**Session : 2020**

**Épreuve de : Informatique**

- Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer
- Rédiger avec un stylo non effaçable bleu ou noir
- Ne rien écrire dans les marges (gauche et droite)
- Numérotter chaque page (cadre en bas à droite)
- Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre

**Consignes**

J. 20 1173

**Q16**

1 **def** recherchedichotomique(a,L):

Intérêt de ce type d'algorithme : .....

NE RIEN ÉCRIRE DANS CE CADRE

**Q17**

h('CCC') : .....

.....

h('ACG') : .....

.....

h('GAG') : .....

.....

**Q18**

1   **def eval(P,b):**

Q19

```
1 def hornerit(P,b):
```

Q20

```
1 def hornerrec(P,b):
2     if .....
3         return .....
4     else:
5         s = P[len(P)-1]
6         s1 = P[0:len(P)-1]
7         return .....
```

Q21

But de la requête(1) : .....  
.....

**Q22**

Requête(2) =

**Q23**

Requête(3) =

**Q24**

Requête(4) =



# Option informatique

---

## Banque X-E.N.S. (2020)

|                                                      |    |
|------------------------------------------------------|----|
| <b>Informatique A - XULCR (4 h) [i203m1e]</b>        | 3  |
| Constructions et explorations de labyrinthes         |    |
| <b>Informatique-Mathématiques (4 h) [m203mie]</b>    | 16 |
| Programmation fonctionnelle, sémantique et topologie |    |

## Concours Mines-Ponts (2020)

|                                     |    |
|-------------------------------------|----|
| <b>Informatique (3 h) [i20mmoe]</b> | 29 |
| Analyse fréquentielle d'un texte    |    |

## Concours Centrale-SupÉlec (2020)

|                                     |    |
|-------------------------------------|----|
| <b>Informatique (4 h) [i20cmoe]</b> | 38 |
| Un système de vote                  |    |

## Concours Communs Polytechniques (2020)

|                                     |    |
|-------------------------------------|----|
| <b>Informatique (4 h) [i20pmoe]</b> | 42 |
| Logique et calcul de propositions   |    |
| Problème de Freudenthal             |    |
| Mots de Lyndon et de de Bruijn      |    |

## EPITA/IPSA (2020)

|                                     |    |
|-------------------------------------|----|
| <b>Informatique (2 h) [i20wmoe]</b> | 51 |
| Un QCM et quatre exercices          |    |

## CAPES externe d'Informatique (2020)

|                                  |    |
|----------------------------------|----|
| <b>Épreuve 1 (5 h) [i20c31e]</b> | 62 |
| Problème 1 : Séquençage de l'ADN |    |
| Problème 2 : SGBD                |    |
| <b>Épreuve 2 (5 h) [i20c32e]</b> | 73 |
| Mastermind                       |    |

# Épreuves d'informatique

---

## Concours Mines-Ponts (2020)

|                                                         |    |
|---------------------------------------------------------|----|
| <b>Informatique (1 h 30), toutes filières [i20mice]</b> | 91 |
| Images de vagues et de structures                       |    |

## Concours Centrale-SupÉlec (2020)

|                                                      |     |
|------------------------------------------------------|-----|
| <b>Informatique (3 h), toutes filières [i20cice]</b> | 103 |
| Photomosaïque                                        |     |

## Concours Communs Polytechniques (2020)

|                                                  |     |
|--------------------------------------------------|-----|
| <b>Informatique (3 h), filière PSI [i20psce]</b> | 111 |
| Détection d'obstacles par un sonar de sous-marin |     |
| <b>Informatique (3 h), filière TSI [i20piue]</b> | 130 |
| Autour du séquençage du génome                   |     |