

INFORMATIQUE 2023



Union des Professeurs
de classes préparatoires
Scientifiques

Sommaire

Informatique commune	1
X – ENS B (XULSR) (2h) - MP, PC, PSI [i223m2e]	
Gestion de versions de grands textes	1
Mines-Ponts (2h) - MP, PC, PSI [i23mmce]	
La typographie informatisée	15
CCINP (3h) - PC, PSI [i23psue]	
Reconnaissance optique de caractères	24
CCINP (3h) - TPC, TSI [i23piue]	
Gestion de Tests dans une entreprise	52
Banque PT (4h) [i23bpt]	
Élasticité d'ADN	76
MPI et MP option informatique	96
X – ENS C (XULSR) (4h) - MPI [i223mpiec]	
Ordonnabilité des espaces métriques	96
X – ENS A (XULSR) (4h) - MPI, MP [i223m1ea]	
Compression entropique	108
X – ENS Info fondamentale (XULSR) (4h) - MPI, MP [i23m3eif]	
Concision et ambiguïté	121
Mines-Ponts (3h) - MPI [i23mmmp1e]	
Listes à accès direct	131
Mines-Ponts (3h) - MPI [i23mmmp12e]	
Complexité de Kolmogoroff	141
Mines-Ponts (3h) - MP [i23mmoel]	
Le jeu du hanjie	154
Centrale-Supélec (4h) - MPI [i23cmpie]	
Problème du voyageur de commerce et processus d'édition sur des arbres	166
Centrale-Supélec (4h) - MP [i23cmoe]	
Transformations sur des langages et représentation d'ensembles d'entiers	173
CCINP (4h) - MPI [i23pmpie]	
Palindromes, traversée de rivière et calcul d'une coupe minimum d'un graphe . . .	178

CCINP (4h) - MP [i23pmoe]	
Partie I (option informatique) : sélection du $(k + 1)$ e plus petit élément - Partie II (informatique commune) : recherche d'une clique de célébrités - Partie III (option informatique) : étude d'une famille d'automates	186
CAPES et agrégation d'informatique	194
CAPES externe d'informatique - Épreuve 1 [i23c31e]	
Résolution de logigrammes	194
CAPES externe d'informatique - Épreuve 2 [i23c32e]	
Systèmes d'exploitation et processus, gestion de données	203
Agrégation externe d'informatique - Épreuve 1 [i23ag1e]	
Partie 1 : provenance en bases de données - Partie 2 : ponts d'un graphe - Partie 3 : architecture des ordinateurs	219
Agrégation externe d'informatique - Épreuve 2 [i23ag2e]	
Test d'égalité de langages rationnels	238
Agrégation externe d'informatique - Épreuve 3 [i23ag3e]	
Option A : livraisons par des véhicules - Option B : réseaux de neurones formels	259

En guise d'introduction

Oser et faire. Il est plus facile de demander le pardon après, que la permission avant.

Grace Hopper

Le recueil 2023 comporte trois parties. La première contient les épreuves d'informatique de l'enseignement commun à toutes nos filières hors MPI. La deuxième contient les épreuves de MPI et de l'option informatique de MP. Enfin, la troisième contient les épreuves des CAPES et agrégation d'informatique.

Le bulletin des concours Informatique n'est proposé aux adhérents que sous forme électronique. Dans ce recueil, le nom du fichier contenant chaque sujet figure dans le sommaire 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>. Ce bulletin et ses prédecesseurs sont à votre disposition sur le site de l'UPS à l'adresse <http://prepas.org/ups.php?rubrique=146>.

Amélie Stainer - Septembre 2023
amelie.stainer@prepas.org

Gestion de versions de grands textes

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve. Le langage de programmation sera **obligatoirement Python**.

Dans ce sujet, on s'intéresse à des textes de grande taille auxquels plusieurs auteurs apportent des modifications au cours du temps. Ces textes peuvent par exemple être des programmes informatiques développés par de multiples auteurs. Il est important de pouvoir efficacement gérer les différentes versions de ces programmes au cours de leur développement et limiter le stockage et la transmission d'informations redondantes. Nous allons pour cela nous intéresser à une notion de *différentiels* entre textes.

Complexité. La complexité, ou le temps d'exécution, d'une fonction P est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc...) nécessaires à l'exécution de P dans le cas le pire. Lorsque la complexité dépend d'un ou plusieurs paramètres $\kappa_1, \dots, \kappa_r$, on dit que A a une complexité en $\mathcal{O}(f(\kappa_1, \dots, \kappa_r))$ s'il existe une constante $C > 0$ telle que, pour toutes les valeurs de $\kappa_1, \dots, \kappa_r$ suffisamment grandes (c'est-à-dire plus grandes qu'un certain seuil), pour toute instance du problème de paramètres $\kappa_1, \dots, \kappa_r$, la complexité est au plus $C \cdot f(\kappa_1, \dots, \kappa_r)$.

Lorsqu'il est demandé de donner la complexité d'un programme, le candidat devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Rappels concernant le langage Python. Ce sujet utilise les types Python listes et dictionnaires, mais seules les opérations mentionnées ci-dessous sont autorisées dans vos réponses. Quand une complexité est indiquée avec un symbole (*), cela signifie que nous faisons une hypothèse simplificatrice sur sa complexité. La justification de cette simplification est hors-programme.

Si $l, l1, l2$ désignent des listes en Python :

- `len(l)` renvoie la longueur de la liste l , c'est-à-dire le nombre d'éléments qu'elle contient. Complexité en $\mathcal{O}(1)$.
- `l1 == l2` teste l'égalité des listes $l1$ et $l2$. Complexité en $\mathcal{O}(n)$ avec n le minimum de `len(l1)` et `len(l2)`.
- $l[i]$ désigne le i -ème élément de la liste l , où l'indice i est compris entre 0 et `len(l)-1`. Complexité en $\mathcal{O}(1)$.
- $l[i:j]$ construit la sous-liste $[l[i], \dots, l[j-1]]$. Complexité en $\mathcal{O}(j-i)$. L'usage des variantes $l[i:]$ à la place de $l[i:len(l)]$, et de $l[:j]$ à la place de $l[0:j]$ est aussi autorisé.

- `l.append(e)` modifie la liste `l` en lui ajoutant l'élément `e` en dernière position. Complexité en $\mathcal{O}(1)$ (*).
- `l.pop()` renvoie le dernier élément de la liste `l` (supposée non vide) et supprime l'occurrence de cet élément en dernière position dans la liste. Complexité en $\mathcal{O}(1)$ (*).

On pourra aussi utiliser la fonction `range` pour réaliser des itérations.

Si `d` est un dictionnaire Python :

- `{key_1: v_1, ..., key_n: v_n}` crée un nouveau dictionnaire en associant chaque valeur `v_i` à une clé `key_i`. Complexité en $\mathcal{O}(n)$ (*).
- `d[key]` renvoie la valeur associée à la clé `key` dans `d` et lève une erreur si la clé `key` n'est pas présente. Complexité en $\mathcal{O}(1)$ (*).
- `d[key] = v` modifie `d` pour associer la valeur `v` à la clé `key`, même si la clé `key` n'est pas présente dans `d` initialement. Complexité en $\mathcal{O}(1)$ (*).
- `key in d` teste si la clé `key` est présente dans `d`. Complexité en $\mathcal{O}(1)$ (*).

Sauf mention contraire, les fonctions à écrire ne doivent pas modifier leurs entrées.

La structure de données *texte*. Dans ce sujet, on appelle *texte* une liste de caractères. Par exemple, `['b', 'i', 'n', 'g', 'o']` est un texte de longueur 5.

Partie I : Différentiels par positions fixes

Dans cette partie, nous traitons le problème avec une hypothèse simplificatrice : les textes comparés ont toujours la même taille.

Question 1. Sans utiliser le test `==` sur les listes, écrire une fonction `textes_égaux(texte1, texte2)` qui teste si deux textes sont égaux. Donner la complexité de cette fonction.

Exemples

```
>>> textes_égaux(['v', 'i', 's', 'a'], ['v', 'a', 'i', 's'])
False
>>> textes_égaux(['v', 'i', 's', 'a'], ['v', 'i', 's', 'a'])
True
```

Dans la suite de ce sujet, on pourra utiliser `==` sur les listes plutôt que cette fonction.

Si deux textes ne sont pas égaux mais ont la même longueur n , on souhaite compter le nombre de positions qui diffèrent, c'est à dire déterminer combien il existe de positions i ($0 \leq i < n$) telles que les caractères en position i sont différents dans les deux textes.

Question 2. Écrire une fonction `distance(texte1, texte2)` qui calcule cette quantité. On supposera que les deux textes ont le même nombre de caractères. Donner la complexité de cette fonction.

Exemples

```
>>> distance(['v', 'i', 's', 'a'], ['v', 'a', 'i', 's'])
3
>>> distance(['a', 'v', 'i', 's'], ['v', 'i', 's', 'a'])
4
```

Question 3. En vous aidant d'un dictionnaire dont les clés sont des caractères, écrire une fonction `aucun_caractère_commun(texte1, texte2)` qui renvoie `True` si et seulement si l'ensemble des caractères qui apparaissent dans `texte1` est disjoint de l'ensemble des caractères qui apparaissent dans `texte2`. Les deux textes peuvent avoir ici des longueurs différentes. Cette fonction devra avoir une complexité $\mathcal{O}(\text{len}(\text{texte1}) + \text{len}(\text{texte2}))$.

Exemples

```
>>> aucun_caractère_commun(['a', 'v', 'i', 's'], ['v', 'i', 's', 'a'])
False
>>> aucun_caractère_commun(['a', 'v', 'i', 's'], ['u', 'r', 'n', 'e'])
True
```

Nous introduisons maintenant une structure de données spécifique pour représenter un différentiel par positions fixes entre deux textes.

La FIGURE 1 présente un exemple de couple de textes (`texte1, texte2`) qui diffèrent sur 4 *tranches* (représentées par des zones grisées sur la figure). En dehors des tranches, les textes sont égaux.

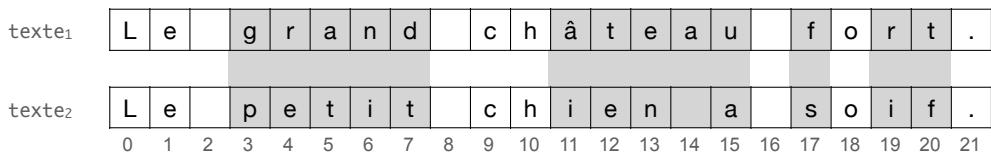


FIGURE 1 – Exemple de couple (`texte1, texte2`) dont on veut calculer le différentiel (sur des positions fixes).

La structure de données *tranche*. Une *tranche* est un dictionnaire avec trois clés '`début`', '`avant`' et '`après`'. La valeur associée à la clé '`début`' est le premier indice de la tranche, les

textes associés aux clés 'avant' et 'après' représentent les textes (**de même longueur**) de la tranche avant et après modification. Dans la suite de cette partie, on s'appuiera sur les fonctions suivantes pour manipuler cette structure.

```
def tranche(arg_début, arg_avant, arg_après):
    return {'début': arg_début, 'avant': arg_avant, 'après': arg_après}

def début(tr):
    return tr['début']

def après(tr):
    return tr['après']

def avant(tr):
    return tr['avant']

def fin(tr):
    return début(tr) + len(après(tr))
```

Nous ne fournissons pas de fonction pour modifier une tranche car nous souhaitons traiter cette structure de données comme une structure *immuable*¹.

On peut représenter le *différentiel* de la FIGURE 1, par la liste suivante :

```
[tranche(3, ['g', 'r', 'a', 'n', 'd'], ['p', 'e', 't', 'i', 't']),
 tranche(11, ['â', 't', 'e', 'a', 'u'], ['i', 'e', 'n', ' ', 'a']),
 tranche(17, ['f'], ['s']),
 tranche(19, ['r', 't'], ['i', 'f'])]
```

La structure de données *différentiel*. Un *différentiel* est une liste (potentiellement vide) de tranches $[\text{tr}_1, \dots, \text{tr}_k]$ représentant des modifications touchant des zones distinctes d'un texte, telle que

- $\text{début}(\text{tr}_1) < \text{fin}(\text{tr}_1) < \dots < \text{début}(\text{tr}_k) < \text{fin}(\text{tr}_k)$
- pour tout $j \in [1, k]$, pour tout $i \in [0, \text{len}(\text{avant}(\text{tr}_j)) - 1]$, $\text{avant}(\text{tr}_j)[i] \neq \text{après}(\text{tr}_j)[i]$

Existence et unicité d'un différentiel par positions fixes. Pour deux textes `texte1` et `texte2` de même longueur n , il existe un unique différentiel $[\text{tr}_1, \dots, \text{tr}_k]$ tel que :

1. On rappelle qu'une structure *immuable* est une structure qui n'est jamais modifiée. C'est par exemple le cas des chaînes et des tuples en Python.

- si $k > 0$, alors $0 \leq \text{début}(\text{tr}_1)$ et $\text{fin}(\text{tr}_k) \leq n$
- pour tout $j \in [1, k]$, $\text{texte}_1[\text{début}(\text{tr}_j) : \text{fin}(\text{tr}_j)] = \text{avant}(\text{tr}_j)$
- pour tout $j \in [1, k]$, $\text{texte}_2[\text{début}(\text{tr}_j) : \text{fin}(\text{tr}_j)] = \text{après}(\text{tr}_j)$
- pour tout $i \in [0, n-1]$, si $i \notin \bigcup_{1 \leq j \leq k} [\text{début}(\text{tr}_j), \text{fin}(\text{tr}_j) - 1]$, alors $\text{texte}_1[i] = \text{texte}_2[i]$

Cet unique différentiel est appelé le *differentiel de texte_2 vis-à-vis de texte_1* .

Toutes les propriétés précédentes sur les différentiels assurent les propriétés intuitives suivantes :

- les tranches sont présentées par indices de début croissants, sans se chevaucher, ni se toucher ;
- chaque tranche tr couvre un intervalle de positions $[\text{début}(\text{tr}), \text{fin}(\text{tr}) - 1]$ sur lequel texte_1 et texte_2 diffèrent à chaque position, et dont les sous-textes sur ces intervalles correspondent à $\text{avant}(\text{tr})$ pour texte_1 et $\text{après}(\text{tr})$ pour texte_2 .

Question 4. Écrire une fonction `differentiel(texte1, texte2)` qui calcule le différentiel du texte texte_2 vis-à-vis du texte texte_1 , supposés de même longueur. La complexité attendue est $\mathcal{O}(\text{len}(\text{texte}_1))$. Justifier cette complexité.

Question 5. Écrire une fonction `applique(texte1, diff)` qui, étant donné un texte texte_1 et un différentiel diff , renvoie un texte texte_2 tel que diff soit le différentiel de texte_2 vis-à-vis de texte_1 . On supposera que le différentiel diff contient des tranches cohérentes avec la taille et le contenu du texte texte_1 . Donner et justifier la complexité.

Pour reconstruire l'ancienne version d'un texte à partir d'un différentiel, nous allons nous appuyer sur la notion de différentiel inversé.

Question 6. Écrire une fonction `inverse(diff)` telle que pour tous textes $\text{texte}_1, \text{texte}_2$ de même longueur, si diff désigne `differentiel(texte1, texte2)`, alors `applique(texte2, inverse(diff)) = texte1` et `inverse(inverse(diff)) = diff`. Donner sa complexité.

La structure de données *texte versionné*. Nous représentons un texte versionné par un dictionnaire contenant la version courante du texte, comme valeur associée à la clé '`courant`', et l'historique des différentiels qui ont mené jusqu'à cette version dans une pile² de différentiels associée à la clé '`historique`'. Dans la suite de cette partie, on s'appuiera sur les fonctions suivantes pour manipuler cette structure.

2. Une pile est ici implémentée par une liste Python.

```

def versionne(texte):
    return {'courant' : texte, 'historique' : [] }

def courant(texte_versionné):
    return texte_versionné['courant']

def remplace_courant(texte_versionné, texte):
    texte_versionné['courant'] = texte

def historique(texte_versionné):
    return texte_versionné['historique']

```

Contrairement à la structure immuable de tranche, nous nous autorisons cette fois à modifier la structure de texte versionné, en particulier la pile qu'elle contient via les opérations `historique(texte_versionné).append(diff)` et `historique(texte_versionné).pop()`.

Question 7. Écrire les fonctions `modifie(texte_versionné, texte)` et `annule(texte_versionné)` qui assurent les deux opérations de base attendues sur un texte versionné `texte_versionné`. La fonction `modifie(texte_versionné, texte)` modifie `texte_versionné` pour lui ajouter une nouvelle version correspondant au texte `texte`, en supposant qu'il a la même longueur n que le texte courant. La taille de l'historique augmente alors de 1. La fonction ne renvoie rien. La fonction `annule(texte_versionné)` modifie `texte_versionné` en annulant l'effet de la dernière modification effectuée et renvoie la nouvelle valeur courante du texte. La taille de l'historique diminue alors de 1. On suppose que la pile des différentiels n'est pas vide lors de cet appel. Donnez les complexités de ces deux fonctions.

Exemples

```

>>> texte_versionné = versionne(['a', 'v', 'i', 's'])
>>> modifie(texte_versionné, ['v', 'i', 's', 'a'])
>>> modifie(texte_versionné, ['v', 'i', 't', 'a'])
>>> modifie(texte_versionné, ['l', 'i', 's', 'a'])
>>> assert courant(texte_versionné) == ['l', 'i', 's', 'a']
>>> assert historique(texte_versionné) == [
    différentiel(['a', 'v', 'i', 's'], ['v', 'i', 's', 'a']),
    différentiel(['v', 'i', 's', 'a'], ['v', 'i', 't', 'a']),
    différentiel(['v', 'i', 't', 'a'], ['l', 'i', 's', 'a'])
]
>>> annule(texte_versionné)
['v', 'i', 't', 'a']
>>> annule(texte_versionné)
['v', 'i', 's', 'a']
>>> annule(texte_versionné)
['a', 'v', 'i', 's']

```

Partie II : Différentiels sur des positions variables

Dans cette partie, nous nous intéressons à des différentiels de textes dont les longueurs ne sont plus forcément égales. Nous adaptons pour cela la définition de *tranche* et de *différentiel*. La FIGURE 2 présente un exemple de couple (`texte1`, `texte2`) dont on va représenter le différentiel par une liste de *tranches* (représentées par des zones grisées sur la figure). Cette fois, les tranches désignent des portions de textes qui ne sont pas nécessairement de la même longueur, ni alignées.

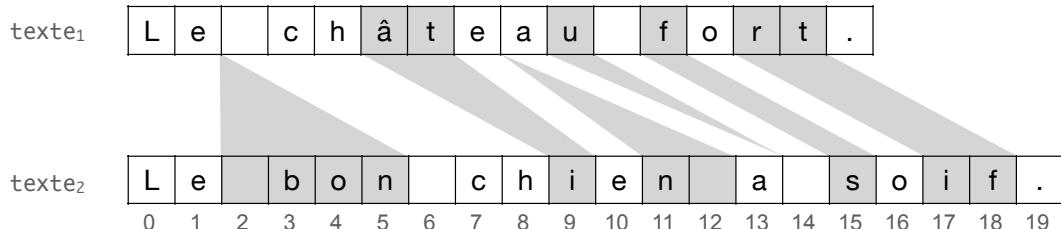


FIGURE 2 – Exemple de couple (`texte1`, `texte2`) dont on veut calculer le différentiel sur des positions variables.

Nouvelle structure de données *tranche*. Un différentiel d'un texte `texte2` vis-à-vis d'un texte `texte1` est toujours une liste de tranches mais chaque tranche comporte maintenant 4 clés :

- la clé 'début_avant' représente la position i d'un sous-texte *avant* qui a été supprimé de `texte1`;
- la clé 'avant' est associée au texte *avant* ;
- la clé 'début_après' représente la position dans `texte2` d'un sous-texte *après*, qui a été ajouté à la place du sous-texte *avant* en position i dans `texte1` ;
- la clé 'après' est associée au texte *après*.

Dans la suite de cette partie, on s'appuiera sur les fonctions suivantes pour manipuler cette structure.

```
def tranche(arg_début_avant, arg_avant, arg_début_après, arg_après):
    return {'début_avant': arg_début_avant,
            'avant': arg_avant,
            'début_après': arg_début_après,
            'après': arg_après}

def début_avant(tr):
    return tr['début_avant']
```

```

def début_après(tr):
    return tr['début_après']

def après(tr):
    return tr['après']

def avant(tr):
    return tr['avant']

def fin_avant(tr):
    return début_avant(tr) + len(avant(tr))

def fin_après(tr):
    return début_après(tr) + len(après(tr))

```

Nouvelle structure de données *différentiel*. Un différentiel est une liste (potentiellement vide) de tranches $[tr_1, \dots, tr_k]$ telle que

- $\text{début_avant}(tr_1) \leq \text{fin_avant}(tr_1) < \dots < \text{début_avant}(tr_k) \leq \text{fin_avant}(tr_k)$
- $\text{début_après}(tr_1) \leq \text{fin_après}(tr_1) < \dots < \text{début_après}(tr_k) \leq \text{fin_après}(tr_k)$
- pour tout $j \in [1, k]$, $\text{aucun_caractère_commun}(\text{avant}(tr_j), \text{après}(tr_j)) = \text{True}$
- pour tout $j \in [1, k]$, $\text{len}(\text{avant}(tr_j)) > 0$ ou $\text{len}(\text{après}(tr_j)) > 0$

Notion de différentiel *valide vis-à-vis de deux textes*. Pour deux textes texte_1 et texte_2 de même longueur n , un *différentiel valide de texte_2 vis-à-vis de texte_1* est une liste $\text{diff} = [tr_1, \dots, tr_k]$ de tranches telle que :

- si $k > 0$, alors $0 \leq \text{début_avant}(tr_1)$ et $\text{fin_avant}(tr_k) \leq \text{len}(\text{texte}_1)$
- si $k > 0$, alors $0 \leq \text{début_après}(tr_1)$ et $\text{fin_après}(tr_k) \leq \text{len}(\text{texte}_2)$
- pour tout $j \in [1, k]$, $\text{texte}_1[\text{début_avant}(tr_j) : \text{fin_avant}(tr_j)] = \text{avant}(tr_j)$
- pour tout $j \in [1, k]$, $\text{texte}_2[\text{début_après}(tr_j) : \text{fin_après}(tr_j)] = \text{après}(tr_j)$
- pour tout $j \in [1, k - 1]$, les sous-textes $\text{texte}_1[\text{fin_avant}(tr_j) : \text{début_avant}(tr_{j+1})]$ et $\text{texte}_2[\text{fin_après}(tr_j) : \text{début_après}(tr_{j+1})]$ sont égaux
- si $k = 0$ alors les textes texte_1 et texte_2 sont égaux
- si $k > 0$ alors $\text{texte}_1[0 : \text{début_avant}(tr_1)] = \text{texte}_2[0 : \text{début_après}(tr_1)]$ et $\text{texte}_1[\text{fin_avant}(tr_k) : \text{len}(\text{texte}_1)] = \text{texte}_2[\text{fin_après}(tr_k) : \text{len}(\text{texte}_2)]$

On peut représenter le *différentiel* de la FIGURE 2, par la liste suivante :

```
[  

tranche( 2, [], 2, [' ', 'b', 'o', 'n']),  

tranche( 5, ['â', 't'], 9, ['i']),  

tranche( 8, [], 11, ['n', ' ']),  

tranche( 9, ['u'], 14, []),  

tranche(11, ['f'], 15, ['s']),  

tranche(13, ['r', 't'], 17, ['i', 'f'])  

]
```

On admet que, comme dans la partie précédente, on peut écrire des fonctions `applique` et `inverse` satisfaisant les mêmes propriétés que précédemment sur cette nouvelle notion de différentiel. On définit le *poids* d'un différentiel comme la somme des longueurs des sous-textes `avant(tr)` et `après(tr)` pour toutes les tranches `tr` qui le composent.

Question 8. Écrire une fonction `poids(diff)` qui calcule le poids d'un différentiel `diff`. Donner sa complexité.

Exemple

```
>>> poids([tranche(0, ['b'], 0, ['t', 'r', 'o', 't', 't']),  

           tranche(2, ['c', 'y', 'c', 'l'], 6, ['n'])])
```

11

On s'intéresse à la *distance d'édition*³ entre deux textes. Dans ce sujet, on définit cette distance comme le nombre minimal de suppressions et d'insertions de caractères pour passer d'un texte à un autre. On peut facilement se convaincre que cette distance coïncide avec le poids minimal possible pour un différentiel entre les deux textes.

Nous allons calculer cette distance par programmation dynamique. Pour deux textes `texte1` et `texte2` fixés, et pour $0 \leq i \leq \text{len}(\text{texte}_1)$ et $0 \leq j \leq \text{len}(\text{texte}_2)$, on note $M[i][j]$ la distance d'édition pour passer de `texte1[0 : i]` à `texte2[0 : j]`. La matrice⁴ M est appelée *matrice de distance d'édition* entre `texte1` et `texte2`.

La FIGURE 3 présente la matrice M pour `texte1 = ['A', 'B', 'C', 'D', 'C', 'E', 'F']` et `texte2 = ['U', 'A', 'B', 'C', 'C', 'X', 'Y', 'Z']`.

Question 9. Donnez une équation de récurrence qui exprime $M[i + 1][j + 1]$ en fonction de $M[i][j]$, $M[i][j + 1]$, $M[i + 1][j]$, `texte1[i]` et `texte2[j]`, pour $0 \leq i < \text{len}(\text{texte}_1)$ et $0 \leq j < \text{len}(\text{texte}_2)$. Justifier brièvement la validité de cette équation, sans rédiger une preuve complète.

Question 10. Écrire une fonction `levenshtein(texte1, texte2)` de complexité polynomiale qui renvoie la matrice M . Préciser sa complexité.

On utilisera l'instruction `M = [[0 for j in range(m)] for i in range(n)]` pour initialiser une matrice M de n lignes et m colonnes avec des zéros.

3. Cette distance est communément appelée *distance de Levenshtein*.

4. Dans ce sujet, nous représenterons ces matrices par des listes de listes d'entiers.

	U	A	B	C	C	X	Y	Z	
A	0	1	2	3	4	5	6	7	8
B	1	2	1	2	3	4	5	6	7
C	2	3	2	1	2	3	4	5	6
D	3	4	3	2	1	2	3	4	5
E	4	5	4	3	2	3	4	5	6
F	5	6	5	4	3	2	3	4	5
G	6	7	6	5	4	3	4	5	6
H	7	8	7	6	5	4	5	6	7

FIGURE 3 – Exemple de matrice de distance d'édition

Question 11. Écrire une fonction `différentiel(texte1, texte2, M)` qui calcule un différentiel du texte `texte2` vis-à-vis du texte `texte1`, en s'aidant de la matrice de distance `M` donnée par `levenshtein(texte1, texte2)`. Le différentiel renvoyé doit être de poids minimal. La fonction devra avoir une complexité $\mathcal{O}(\text{len}(\text{texte1}) + \text{len}(\text{texte2}))$. Justifier cette complexité et expliquer brièvement pourquoi le différentiel calculé satisfait les propriétés attendues par un différentiel. On pourra s'aider de la FIGURE 3 pour comprendre quel parcours suivre dans la matrice `M`.

Si on se place dans un scénario de *travail collaboratif* où deux auteurs différents modifient en parallèle le même texte `texte`, il est nécessaire de pouvoir *fusionner* leur travail. Nous notons `texte1` le nouveau texte obtenu après le travail du premier auteur sur `texte` et `diff1` le différentiel correspondant. De même, nous notons `texte2` le texte obtenu après le travail du deuxième auteur sur le même texte `texte`, et `diff2` le différentiel correspondant.

Exemple

```
>>> texte =
    ['l', 'e', ' ', 'c', 'h', 'a', 't', ' ', 'a', ' ', 's', 'o', 'i', 'f']
>>> texte1 =
    ['l', 'e', ' ', 'c', 'h', 'a', 't', ' ', 'a', ' ', 't', 'r', 'è', 's',
     ' ', 's', 'o', 'i', 'f']
>>> texte2 =
    ['l', 'e', ' ', 'c', 'h', 'i', 'e', 'n', ' ', 'a', ' ', 's', 'o', 'i', 'f']
>>> diff1 = différentiel(texte, texte1, levenshtein(texte, texte1))
>>> assert diff1 == [tranche(9, [], 9, [' ', 't', 'r', 'è', 's'])]
>>> diff2 = différentiel(texte, texte2, levenshtein(texte, texte2))
>>> assert diff2 == [tranche(5, ['a', 't'], 5, ['i', 'e', 'n'])]
```

Pour fusionner le travail des deux auteurs, on apporte des modifications à `diff2` de façon à ce que le texte final, qui inclut les modifications des deux auteurs, soit exprimable comme l'application du différentiel `diff1`, puis de la nouvelle version de `diff2` sur le texte initial. Dans l'exemple précédent, le texte final attendu est : `['l', 'e', ' ', 'c', 'h', 'i', 'e', 'n', ' ', 'a', ' ', 't', 'r', 'è', 's']`.

Nous allons être prudents en nous assurant au préalable que les modifications apportées ne concernent pas les même zones du texte initial.

Question 12. Écrire une fonction `conflit(diff1, diff2)` qui prend en argument deux différentiels `diff1` et `diff2` et renvoie `True` si et seulement s'il existe une tranche `tr1` dans `diff1` et une tranche `tr2` dans `diff2` telles que

$$[\text{début_avant}(\text{tr}_1), \text{fin_avant}(\text{tr}_1)] \cap [\text{début_avant}(\text{tr}_2), \text{fin_avant}(\text{tr}_2)] \neq \emptyset$$

Cette fonction devra avoir une complexité $\mathcal{O}(\text{len}(\text{diff1}) + \text{len}(\text{diff2}))$ que l'on justifiera.

Question 13. Écrire une fonction `fusionne(diff1, diff2)` qui renvoie un nouveau différentiel représentant la mise à jour de `diff2`. Il est attendu que `poids(fusionne(diff1, diff2)) = poids(diff2)`. On suppose que les deux différentiels `diff1` et `diff2` ne sont pas en conflit. Cette fonction devra avoir une complexité $\mathcal{O}(\text{len}(\text{diff1}) + \text{len}(\text{diff2}))$.

Exemple

```
>>> assert not conflit(diff1, diff2)
>>> print(applique(applique(texte, diff1), fusionne(diff1, diff2)))
['l', 'e', ' ', 'c', 'h', 'i', 'e', 'n', ' ', 'a', ' ', 't', 'r', 'è', 's', ' ', 
 's', 'o', 'i', 'f']
```

Partie III : Calcul de différentiels par calcul de plus courts chemins

Dans cette partie on souhaite exprimer le problème de calcul de distance d'édition comme un problème de calcul de plus court chemin dans un graphe orienté pondéré. Pour deux textes `texte1` et `texte2`, on considère une grille de dimension $(\text{len}(\text{texte}_1)+1) \times (\text{len}(\text{texte}_2)+1)$ dont chaque cellule est un sommet du graphe. On appelle *sommet* un couple (i, j) tel que $0 \leq i \leq \text{len}(\text{texte}_1)$ et $0 \leq j \leq \text{len}(\text{texte}_2)$. Chaque sommet (i, j) aura au plus trois arcs sortants vers des sommets parmi $(i+1, j)$, $(i, j+1)$ et $(i+1, j+1)$. On appelle *entrée* du graphe le sommet $(0, 0)$ et *sortie* le sommet $(\text{len}(\text{texte}_1), \text{len}(\text{texte}_2))$.

La FIGURE 4 présente la matrice de distance d'édition pour `texte1 = ['b', 'i', 'e', 'n']` et `texte2 = ['b', 'o', 'n', 'n', 'e']`, ainsi que le graphe associé, sans les poids des arcs.

Le graphe ne sera jamais explicitement représenté, mais nous sommes en mesure de **calculer** l'ensemble des arcs sortants de chaque sommet.

Question 14. Écrire une fonction `successeurs(texte1, texte2, sommet)` qui renvoie une liste de couples `(voisin, distance)`, de taille au plus 3, représentant les sommets destinations des arcs sortant du sommet `sommet`, avec les poids associés.

L'existence et la pondération des arcs devra permettre d'assurer la correspondance suivante entre le graphe et la matrice de distance d'édition de `texte1` et `texte2` : pour tout sommet (i, j) du graphe, $M[i][j]$ coïncide avec la longueur d'un plus court chemin de $(0, 0)$ à (i, j) .

Démontrer cette propriété avec une récurrence.

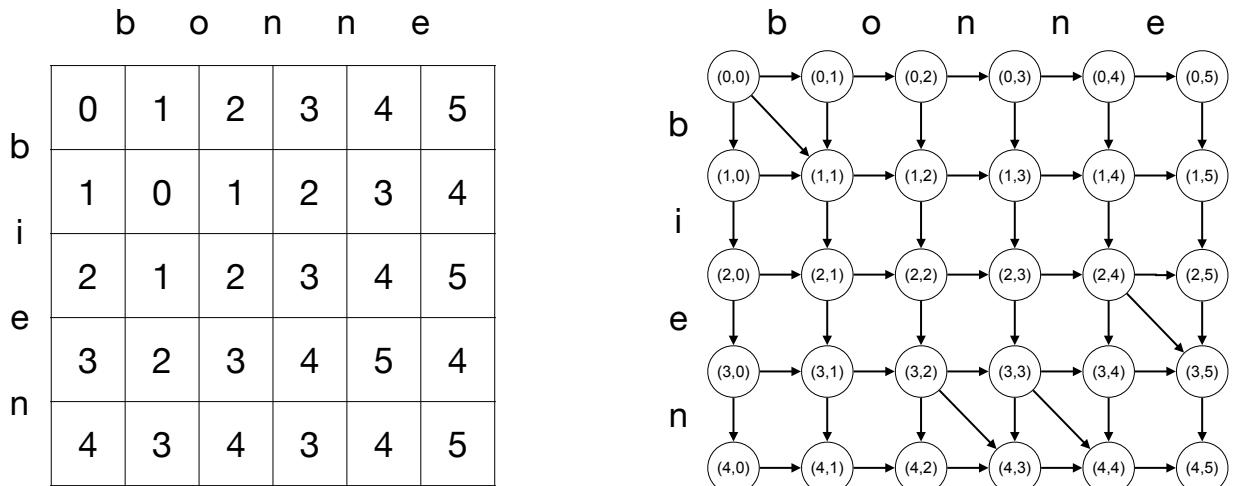


FIGURE 4 – Exemple de matrice de distance d'édition et de graphe associé (les poids des arcs ont été volontairement omis).

Exemple (les poids des arcs sont ici remplacés par ...)

```
>>> texte1 = ['b', 'i', 'e', 'n']
>>> texte2 = ['b', 'o', 'n', 'n', 'e']
>>> successeurs(texte1, texte2, (2,4))
[((3, 4), ...), ((2, 5), ...), ((3, 5), ...)]
```

Pour calculer un plus court chemin, on peut utiliser une variante l'algorithme de Dijkstra, présentée dans la FIGURE 5. Il s'appuie sur une structure de données de file de priorité sur les sommets (i, j) du graphe, dont on ne précise pas l'implémentation mais dont on précise ici la complexité des différentes opérations élémentaires.

- La fonction `vide()` construit une file vide (de cardinal 0) en $\mathcal{O}(1)$.
- La fonction `est_vide(file)` teste si la file `file` est vide en $\mathcal{O}(1)$.
- La fonction `extraire_min(file)` supprime l'élément de priorité minimale dans la file `file` et le renvoie. En cas d'égalité de priorités, elle renvoie le sommet (i, j) le plus petit pour l'ordre lexicographique⁵ parmi les sommets de priorité minimale. Sa complexité est en $\mathcal{O}(\log(\text{cardinal}(\text{file})))$.
- La fonction `ajoute(file, sommet, priorité)` ajoute à la file `file` un sommet `sommet` avec une priorité `priorité`. L'opération augmente de 1 le cardinal de la file si le sommet n'est pas déjà présent avec cette priorité. Sa complexité est en $\mathcal{O}(\log(\text{cardinal}(\text{file})))$.

5. On rappelle que l'ordre lexicographique \prec sur les paires est défini par $(i_1, j_1) \prec (i_2, j_2)$ si et seulement si $i_1 < i_2$ ou $(i_1 = i_2$ et $j_1 < j_2)$.

```

def dijkstra(texte1, texte2):
    entrée = (0, 0)
    sortie = (len(texte1), len(texte2))
    file = vide()
    dist = {}
    vue = {}
    horloge = 0
    ajoute(file, entrée, 0)
    dist[entrée] = 0
    while not est_vide(file):
        sommet = extraire_min(file)
        if not sommet in vue:
            vue[sommet] = horloge
            horloge +=1
        if sommet == sortie:
            dist_final = {sommet: dist[sommet] for sommet in vue}
            return dist_final
        for voisin, distance in successeurs(texte1, texte2, sommet):
            d = dist[sommet] + distance
            if not voisin in dist or d < dist[voisin]:
                dist[voisin] = d
                ajoute(file, voisin, d)
    assert False

```

FIGURE 5 – Une variante de l'algorithme de Dijkstra.

Question 15. En vous appuyant sur les propriétés de l'algorithme de Dijkstra vues en cours, expliquer pourquoi l'utilisation de la fonction `dijkstra` permet de calculer la distance d'édition entre `texte1` et `texte2`. Préciser ce que contient le dictionnaire `dist_final` renvoyé, en caractérisant soigneusement l'ensemble des clés de ce dictionnaire.

Question 16. Donner la complexité de la fonction `dijkstra` et commenter son intérêt par rapport à l'algorithme de programmation dynamique de la partie II.

On s'intéresse maintenant à l'algorithme A^* , présenté dans la FIGURE 6. Il s'appuie sur une fonction heuristique `h` qui *estime* la distance de chaque sommet à la sortie du graphe. On admet que cet algorithme renvoie un dictionnaire `dist_final` tel que `dist_final[sortie]` est la longueur d'un plus court chemin de l'entrée à la sortie du graphe, si la fonction heuristique `h` utilisée est *admissible*, c'est à dire si pour tout sommet `s` du graphe, `h(texte1, texte2, s)` est inférieure ou égale à la longueur pondérée d'un plus court chemin de `s` jusqu'à la sortie du graphe.

Question 17. Donner une fonction `h` qui satisfait cette hypothèse, avec une complexité en $\mathcal{O}(1)$, et qui permet un gain de temps de calcul (vis-à-vis du nombre de sommets extraits de la file avant de rencontrer la sortie) sur l'exemple `texte1 = ['A', 'B', 'C']`, `texte2 = ['B', 'X']`. Justifier en comparant les dictionnaires `dist_final` renvoyés par les deux algorithmes sur cet exemple.

```

def astar(texte1, texte2):
    entrée = (0, 0)
    sortie = (len(texte1), len(texte2))
    file = vide()
    dist = {}
    vue = {}
    horloge = 0
    ajoute(file, entrée, 0)
    dist[entrée] = 0
    while not est_vide(file):
        sommet = extraire_min(file)
        vue[sommet] = horloge
        horloge += 1
        if sommet == sortie:
            dist_final = {sommet: dist[sommet] for sommet in vue}
            return dist_final
        for voisin, distance in successeurs(texte1, texte2, sommet):
            d = dist[sommet] + distance
            if not voisin in dist or d < dist[voisin]:
                dist[voisin] = d
                ajoute(file, voisin, d + h(texte1, texte2, voisin))
    assert False

```

FIGURE 6 – Algorithme A^* .

* *
*

A2023 – INFO

**ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.**

Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle International).

CONCOURS 2023**ÉPREUVE D'INFORMATIQUE COMMUNE**

Durée de l'épreuve : 2 heures

L'usage de la calculatrice ou de tout dispositif électronique est interdit.

Cette épreuve est commune aux candidats des filières MP, PC et PSI.

*Les candidats sont priés de mentionner de façon apparente
sur la première page de la copie :*

INFORMATIQUE COMMUNE

L'énoncé de cette épreuve comporte 8 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Les sujets sont la propriété du GIP CCMP. Ils sont publiés sous les termes de la licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Pas de Modification 3.0 France. Tout autre usage est soumis à une autorisation préalable du Concours commun Mines Ponts.



La typographie informatisée

Ce sujet explore quelques aspects de la typographie informatisée. Il aborde la gestion de polices vectorielles, leur manipulation, leur tracé, l'affichage de texte et la justification d'un paragraphe. Il va du texte à la page via le pixel. Les questions posées peuvent dépendre des questions précédentes. Toutefois, une question peut être abordée en supposant les fonctions précédentes disponibles, même si elles n'ont pas été implémentées. Les questions de programmation seront traitées en Python.

Partie I – Préambule

La typographie est l'art d'assembler des caractères afin de composer des pages en vue de leur impression ou de leur affichage sur un écran, en respectant des règles visuelles qui rendent un texte agréable à lire. Elle requiert des efforts importants, avantageusement simplifiés par le recours à l'outil informatique.

Donald Knuth, prix Turing 1974 notamment pour la monographie *The Art of Computer Programming*, en est un pionnier. Lassé de la piètre qualité de la typographie proposées pour son ouvrage, il développe les logiciels TeX pour la mise en page et METAFONT pour la gestion de polices. Leslie Lamport, prix Turing 2013 pour ses travaux sur les systèmes distribués, a écrit L^AT_EX qui facilite l'utilisation de TeX. L^AT_EX est largement utilisé dans l'édition scientifique, y compris pour la composition du présent document.

Donald Knuth récompense toute personne qui signale une nouvelle erreur dans un de ses ouvrages par un chèque d'un montant de un hexa dollar, c'est-à-dire 100 cents où 100 est interprété en base hexadécimale (base 16).

Q1 Quel montant est effectivement versé en dollars par Donald Knuth pour une nouvelle erreur trouvée ?

Voici la définition de quelques termes utiles pour la suite :

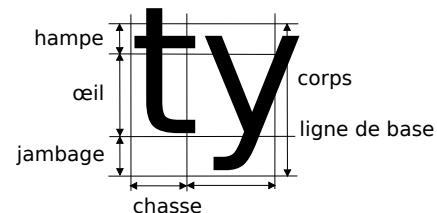
- un **caractère** est un signe graphique d'un système d'écriture, par exemple le *caractère latin a majuscule* « A ». Le standard Unicode donne à chaque caractère un nom et un identifiant numérique, appelé *point de code*, que nous appellerons ci-après simplement *code*. Le code de « A » dans la représentation Unicode est 65. La version 13.0 publiée en mars 2020 répertorie 143 859 caractères couvrant 154 systèmes d'écriture, modernes ou historiques comme les hiéroglyphes ;
- un **glyphe** est un dessin particulier représentant un caractère, par exemple pour le *caractère latin a majuscule* : A (roman) A (italique) A (caligraphié) A (gras), A (courrier)…
- une **police** de caractères est un ensemble coordonné de glyphes incluant différentes variantes (style roman ou italique, graisse...) et permettant de représenter un texte complet dans un système d'écriture. La police de ce document est *Computer Modern*, la police par défaut de L^AT_EX ;
- une **famille** est un groupe de polices. La classification Vox-ATypI, proposée par Maximilien Vox en 1952 et adoptée par l'Association typographique internationale, contient 11 familles. La police *Computer Modern* fait partie de la famille *Didone*.

Le corps du glyphe est sa hauteur, la chasse est sa largeur.

Le corps est décomposé en trois parties : l'œil qui contient typiquement les petites lettres, le jambage et la hampe qui recouvrent les dépassements en dessous ou au dessus de l'œil. La limite inférieure de l'œil est la ligne de base. Elle définit l'alignement des caractères. La chasse peut être fixe (polices monospaces) ou variable.

La description vectorielle d'un glyphe est définie de la façon suivante :

- un **point p** est repéré par ses coordonnées (abscisse, ordonnée) dans le plan orthonormé classique, et sera représenté par une liste de deux flottants ;
- une **multi-ligne l** est une séquence de points reliés par des segments, représentée par une liste de points, éventuellement restreinte à un seul point ;



- la **description vectorielle** v d'un glyphe est un ensemble non vide de multi-lignes, représenté par une liste de multi-lignes.

Les descriptions vectorielles seront supposées normalisées de sorte que la ligne de base corresponde à l'ordonnée 0, que la hauteur de l'œil soit 1, et enfin que le glyphe soit collé à l'abscisse 0, sans dépassement vers les abscisses négatives.

Concrètement, la description vectorielle d'un glyphe est une liste de listes de listes de 2 flottants. À titre d'illustration, voici une description vectorielle d'un glyphe, composée de deux multi-lignes .

```
| v = [ [ [ 0.25, 1.0 ], [ 0.25, -1.0 ], [ 0.0, -1.0 ] ], [ [ 0.25, 1.25 ] ] ]
```

- Q2** Dessiner ce glyphe. De quel caractère s'agit-il ?

Partie II – Gestion de polices de caractères vectorielles

Une base de données stocke les informations liées aux polices de caractères dans 4 tables ou relations.

Famille décrit les familles de polices, avec **fid** la clé primaire entière et **fnom** leur nom.

Police décrit les polices de caractères disponibles, avec **pid** la clé primaire entière, **pnom** le nom de la police et **fid** de numéro de sa famille.

Caractere décrit les caractères, avec **code** la clé primaire entière, **car** le caractère lui-même, **cnom** le nom du caractère.

Glyphe décrit les glyphs disponibles, avec **gid** la clé primaire entière, **code** le code du caractère correspondant au glyphe, **pid** le numéro de la police à laquelle le glyphe appartient, **groman** un booléen vrai pour du roman et faux pour de l'italique et **gdesc** la description vectorielle du glyphe.

Voici un extrait du contenu de ces tables.

Famille	
fid	fnom
1	Humane
2	Garalde
3	Réale
4	Didone
5	Mécane
6	Linéale
...	...

Police		
pid	pnom	fid
1	Centaur	1
2	Garamond	2
3	Times New Roman	3
4	Computer Modern	4
...
21	Triangle	6
...

Caractere		
code	car	cnom
65	A	lettre majuscule latine a
66	B	lettre majuscule latine b
...
97	a	lettre minuscule latine a
98	b	lettre minuscule latine b
99	c	lettre minuscule latine c
...

Glyphe				
gid	code	pid	groman	gdesc
1	65	20	True	[[[0, 0], [1, 2], [2, 0]], [[0.5, 1], [1.5, 1]]]
2	65	20	False	[[[0, 0], [2, 2], [2, 0]], [[1, 1], [2, 1]]]
...
501	97	21	True	[[[0, 0], [0.5, 1], [1, 0], [0, 0]]]
502	98	21	True	[[[0, 2], [0, 0], [1, 0.5], [0, 1]]]
503	99	21	True	[[[1, 1], [0, 0.5], [1, 0]]]
504	100	21	True	[[[1, 2], [1, 0], [0, 0.5], [1, 1]]]
...

- Q3** Proposer une requête en SQL sur cette base de données pour compter le nombre de glyphs en roman (cf. description précédente).

- Q4** Proposer une requête en SQL afin d'extraire la description vectorielle du caractère A dans la police nommée Helvetica en italique.

- Q5 Proposer une requête en SQL pour extraire les noms des familles qui disposent de polices et leur nombre de polices, classés par ordre alphabétique.

Pour la suite, la requête de la question 4 est supposée paramétrée et encapsulée dans une fonction `glyphe(c, p, r)` qui renvoie la description vectorielle du caractère `c` dans la police `p` en roman ou italique selon le booléen `r`, de sorte que l'appel à `glyphe("a", "Helvetica", False)` répond à la question 4.

Partie III – Manipulation de descriptions vectorielles de glyphs

L'avantage de la description vectorielle de glyphs est qu'il est possible de réaliser des opérations sur les glyphs sans perte d'information. On peut réaliser simplement un agrandissement des glyphs, une déformation de glyphe pour en créer un nouveau etc. Cette partie propose des fonctions pour analyser et modifier des descriptions vectorielles.

Dans un premier temps, des fonctions sont créées pour extraire des informations sur des glyphs. Deux fonctions utilitaires sont implémentées.

- Q6 Implémenter la fonction utilitaire `points(v: [[[float]]]) -> [float]` qui renvoie la liste des points qui apparaissent dans les multi-lignes de la description vectorielle `v` d'un glyphe.

```
v = [ [ [ 0, 0 ], [ 1, 1 ] ], [ [ 0, 1 ], [ 1, 0 ] ] ]
print(points(v))                                # affiche la liste [ [ 0, 0 ], [ 1, 1 ], [ 0, 1 ], [ 1, 0 ] ]
```

- Q7 Implémenter la fonction utilitaire `dim(l:[[[float]]], n:int) -> [float]` qui renvoie la liste des éléments d'indice `n` (en commençant à 0) des sous listes de flottants, dont on supposera qu'ils existent toujours.

```
l = [ [ 1, 2 ], [ 3, 4 ], [ 5, 6 ], [ 7, 8 ] ]
print(dim(l, 1))                                # affiche la liste [ 2, 4, 6, 8 ]
```

On cherche à déterminer les dimensions (largeur et hauteur) d'un glyphe donné de manière à pouvoir les modifier par la suite si nécessaire.

- Q8 Implémenter la fonction `largeur(v: [[[float]]]) -> float` qui renvoie la largeur de la description vectorielle `v`. Il faudra utiliser les fonctions utilitaires précédentes ainsi que les fonctions `max` et `min` de Python appliquées à des listes.

- Q9 Implémenter la fonction `obtention_largeur(police:str) -> [float]` qui renvoie une liste de largeurs pour toutes les lettres minuscules romanes et italiques (uniquement les 26 lettres non accentuées de `a` à `z`) de la police `police` dans l'ordre a roman, a italique, b roman, b italique...

On souhaite dériver automatiquement de nouvelles représentations vectorielles de glyphs à partir de représentations existantes.

Python permet de passer simplement des fonctions en paramètre d'autres fonctions. Par exemple, la fonction `applique` ci-après renvoie une nouvelle liste constituée en appliquant la fonction `f` à tous les éléments de la liste 1.

```
def applique(f:callable, l:[]) -> []:
    return [ f(i) for i in l ]

def incremente(i:int) -> int:
    return i + 1

print(applique(incremente, [ 0, 5, 8 ]))      # affiche la liste [ 1, 6, 9 ]
```

- Q10 En se basant sur l'exemple de la fonction `applique`, implémenter une fonction utilitaire `transforme(f:callable, v: [[[float]]]) -> [[[float]]]` qui prend en paramètres une fonction `f`, une description vectorielle `v` et qui renvoie une nouvelle description vectorielle construite à partir de `v` en appliquant la fonction `f` à chacun des points et en préservant la structure des multi-lignes. La fonction `f` passée en argument transforme un point en un autre point.

Soit la fonction `zzz` qui renvoie un nouveau point calculé de la façon suivante :

```

1 | def zzz(p:[float])->[float]:
2 |     return [ 0.5 * p[0], p[1] ]

```

□ **Q11** Expliquer comment est modifiée une description vectorielle v par `transforme(zzz, v)`. Préciser l'effet obtenu sur un glyphe.

□ **Q12** Implémenter la fonction `penche(v: [[[float]]])->[[[float]]]` qui renvoie une nouvelle description vectorielle correspondant à un glyphe penché vers la droite, obtenue en modifiant comme suit les coordonnées des points (x, y) :

- la nouvelle abscisse est $x + 0.5 * y$;
- la nouvelle ordonnée reste y .

Partie IV – Rasterisation

La rasterisation est la transformation d'une image vectorielle en image matricielle. Cette opération est indispensable notamment pour afficher à l'écran une image vectorielle.

Dans cette partie, il s'agit d'analyser comment représenter le segment entre deux points d'une représentation vectorielle par des pixels encrés dans une image bitmap.

Le module `PIL` (Python Image Library) fournit le sous module `Image` :

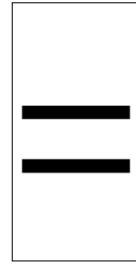
- `im = Image.new(mode, size, color=0)` alloue une nouvelle image matricielle, de type bitmap si `mode` vaut "1"; le tuple `size` donne la largeur et la hauteur de l'image en pixels ; le paramètre facultatif `color` précise la couleur par défaut des pixels, en bitmap 1 pour blanc et 0 pour noir ;
- `im.putpixel((x, y), 1)` attribue la valeur 1 au pixel de coordonnées (x, y) de l'image `im` ;
- `im.save(nom_fichier)` sauvegarde l'image dans un fichier dont on donne le nom ;
- `im.show()` affiche l'image dans une fenêtre graphique.

Attention, dans le domaine graphique (images, impressions, écrans), et contrairement aux conventions mathématiques usuelles, le pixel $(0, 0)$ est le coin en haut à gauche de l'image, l'axe des ordonnées est dirigé vers le bas. Ainsi, le code suivant crée une image bitmap rectangulaire 50×100 blanche avec deux barres horizontales formant un signe « = ».

```

1 | from PIL import Image
2 |
3 | im = Image.new("1", (50, 100), color=1)
4 | for y in range(60, 65):
5 |     for x in range(5, 45):
6 |         im.putpixel((x, y), 0)
7 |         im.putpixel((x, y-20), 0)
8 | im.save("egal.png")

```



La fonction `trace_quadrant_est` implémente une partie de l'algorithme de tracé continu de segment proposé par Jack E. Bresenham en 1962.

```

1 | from PIL import Image
2 | from math import floor      # renvoie l'entier immédiatement inférieur
3 |
4 | def trace_quadrant_est(im:img, p0:(int), p1:(int)):
5 |     x0, y0 = p0
6 |     x1, y1 = p1
7 |     dx, dy = x1-x0, y1-y0
8 |     im.putpixel(p0, 0)
9 |     for i in range(1, dx):
10 |         p = (x0 + i, y0 + floor(0.5 + dy * i / dx))
11 |         im.putpixel(p, 0)
12 |     im.putpixel(p1, 0)
13 |
14 |     im = Image.new("1", (10, 10), color=1)
15 |     trace_quadrant_est(im, (0, 0), (6, 2))
16 |     trace_quadrant_est(im, (9, 8), (1, 9))
17 |     trace_quadrant_est(im, (3, 0), (5, 8))
18 |     im.show()

```

- **Q13** Préciser les coordonnées des pixels de l'image encrés (pixels que l'on met en noir) par l'exécution de la ligne 15 ?
- **Q14** Préciser les coordonnées des pixels de l'image encrés par l'exécution de la ligne 16 ? Indiquer d'où vient le problème rencontré. Proposer une assertion à mettre en début de fonction pour éviter ce problème.
- **Q15** Préciser les coordonnées des pixels de l'image encrés par l'exécution de la ligne 17 ? Expliquer le problème rencontré et à quoi il est dû.
- **Q16** En s'inspirant du code de la fonction `trace_quadrant_est`, implémenter une nouvelle fonction `trace_quadrant_sud` qui règle le problème précédent.
- **Q17** Implémenter la fonction `trace_segment(im:Image, p0:(int), p1:(int))` qui trace un segment continu entre les pixels `p0` et `p1` sur l'image `im` supposée assez grande. Cette fonction devra opérer correctement si les deux points passés en arguments sont égaux.

Partie V – Affichage de texte

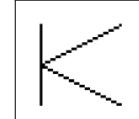
La fonction `trace_segment` va maintenant permettre de tracer sur une page (comprendre une image matricielle) des glyphes à partir de leur description vectorielle.

- **Q18** Implémenter la fonction `position(p:(float), pz:(int), taille:int)->(int)` qui renvoie les coordonnées du point `p` (point d'un glyphe de coordonnées flottantes) en un point dans une page (pixel de coordonnées entières) de manière à ce que le point `(0, 0)` de la description vectorielle soit en position `pz` sur la page, et que l'œil de taille normalisée 1 du glyphe fasse `taille` pixels de hauteur. Prendre garde à la bonne orientation du glyphe sur la page. Vérifier que, pour la taille limite 1, l'œil du glyphe fait bien 1 pixel de hauteur.
- **Q19** Implémenter la fonction `affiche_car(page:Image, c:str, police:str, roman:bool, pz:(int), taille:int)->int` qui affiche dans l'image `page` le caractère `c` dans la police `police`, en roman ou italique selon la valeur du booléen `roman`, et renvoie la largeur en pixel du glyphe affiché. Pour rappel, la fonction `glyphe(c:str, police:str, roman:bool)` charge la description vectorielle du caractère `c` dans la police `police` pour la variante `roman`.

```

1 from affiche import affiche_car
2 from PIL import Image
3 page = Image.new("1", (50, 50), color=1)
4 avance = affiche_car(page, "K", "Triangle", True, [ 10, 40 ], 16)
5 page.save("K.png")

```

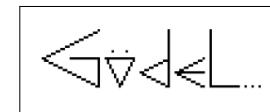


- **Q20** Implémenter la fonction `affiche_mot(page:Image, mot:str, ic:int, police:str, roman:bool, pz:(int), taille:int)->int` qui affiche la chaîne de caractères `mot` dans les mêmes conditions, chaque glyphe étant séparé du suivant par `ic` pixels, et renvoie la position du dernier pixel de la dernière lettre dans la page.

```

1 from affiche import affiche_mot
2 from PIL import Image
3 page = Image.new("1", (110, 50), color=1)
4 avance = affiche_mot(page, "Gödel...", 2, "Triangle", True, [ 10, 35 ], 13)
5 page.save("goedel.png")

```



De la même manière, on pourrait implémenter une fonction `affiche_ligne(page:Image, ligne:[str], ic:int, im:int, police:str, roman:bool, pz:(int), taille:int)` qui afficherait la liste de mots `ligne` en les séparant de `im` pixels.

Partie VI – Justification d'un paragraphe

L'objectif de cette dernière partie est d'afficher un paragraphe de manière harmonieuse en le justifiant, c'est-à-dire en alignant les mots sur les bords gauche et droit de la zone d'écriture de la page.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse Lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas Leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci Luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit.

FIGURE 1 – Illustration de la justification de paragraphe pour différentes polices

Le paragraphe à justifier est constitué d'une liste de mots (chaînes de caractères). La difficulté est de placer des espaces entre les mots et de découper la liste en sous-listes pour que les lignes soient équilibrées (pas de ligne avec beaucoup d'espaces à la fin ou entre les mots par exemple). Pour simplifier le problème, on considère que le paragraphe est constitué d'une liste `1mots` d'entiers correspondant aux longueurs de chaque mot du paragraphe dans l'ordre d'apparition des mots (`1mots[i]` correspond au nombre de caractères du mot d'indice `i`). On note `L` le nombre de caractères et espaces que peut contenir une ligne au maximum. Il faut au minimum un espace entre deux mots d'une même ligne, on suppose que cet espace minimal correspond à un caractère.

On propose dans un premier temps l'algorithme glouton suivant :

```

1  def glouton(Lmots:[int],L:int)->[[int]]:
2      lignes=[]
3      nligne=[]
4      l=0
5      for c in Lmots :
6          if (c + 1) > L:
7              lignes.append(nligne)
8              nligne=[c]
9              l=c+1
10         else:
11             l=l+c+1
12             nligne.append(c)
13     lignes.append(nligne)
14  return lignes

```

□ Q21 Expliquer en une ou deux phrases le principe de l'algorithme et pourquoi il est dit glouton.

Cet algorithme fournit une solution mais qui n'est pas nécessairement optimale. Si on teste cet algorithme sur le paragraphe suivant extrait du lorem ipsum : `ut enim ad minima veniam` pour une longueur de ligne maximale `L=10`, le résultat obtenu est le découpage noté a). Si on utilise une méthode de programmation dynamique, on obtient le découpage noté b).

- a) Découpage obtenu par l'algorithme glouton
- b) Découpage obtenu par programmation dynamique

<code>ut enim ad</code>	<code>ut enim</code>
<code>minima</code>	<code>ad minima</code>
<code>veniam</code>	<code>veniam</code>

Pour évaluer la pertinence du placement d'espaces et de retours à la ligne, on définit une fonction coût à minimiser pour que la répartition soit la plus harmonieuse possible.

Cette fonction coût correspond au nombre d'espaces disponibles sur une ligne élevé au carré, si on commence la ligne du mot i au mot j inclus sur cette même ligne :

$$cout(i, j) = (L - (j - i) - \sum_{k=i}^j lmots[k])^2$$

Cette fonction prend la valeur ∞ lorsque la somme des longueurs des mots de i à j est supérieure à L (ce qui signifie qu'on ne peut pas placer les mots i à j sur une même ligne).

La fonction python suivante correspond à l'implémentation de cette fonction coût.

```

1 | def cout(i:int,j:int,lmots:[int],L:int)->int:
2 |     res=sum(lmots[i:j+1])+(j-i)
3 |     if res>L:
4 |         return float("inf")
5 |     else:
6 |         return (L-res)**2

```

□ **Q22** Évaluer pour les deux découpages a) et b) de l'exemple, ce que renvoie la fonction coût pour chacune des lignes en précisant les indices i et j pour chaque ligne (`lmots=[2,4,2,6,6]`). Conclure sur l'algorithme qui donne la solution la plus harmonieuse en sommant les différents coûts par ligne pour le découpage a) puis pour le découpage b).

La méthode de programmation dynamique consiste dans un premier temps à déterminer une équation de récurrence (équation de Bellman). Le problème peut être reformulé de la manière suivante :

Si on suppose connue la solution pour placer les mots jusqu'à un indice i , le nouveau problème consiste à placer correctement les changements de lignes du mot i jusqu'à la fin. La question est donc de savoir où placer le changement de ligne à partir du mot d'indice i , de manière à minimiser la fonction coût.

On note $d(i)$ le problème du placement optimal de changement de ligne jusqu'à l'indice i . L'équation de Bellman correspondante est alors :

$$d(i) = \min_{i < j \leq n} (d(j) + cout(i, j - 1))$$

Un algorithme récursif naïf correspondant à la résolution de ce problème est le suivant.

```

1 | def algo_recuratif(i:int,lmots:[int],L:int)->int:
2 |     if i==len(lmots):
3 |         return 0
4 |     else:
5 |         mini=float("inf")
6 |         for j in range(i+1,len(lmots)+1):
7 |             d=algo_recuratif(j,lmots,L)+cout(i,j-1,lmots,L)
8 |             if d<mini:
9 |                 mini=d
10 |         return mini

```

□ **Q23** Proposer une modification de la fonction `algo_recuratif` pour rendre celle-ci plus efficace en introduisant une mémoisation.

On définira la nouvelle fonction récursive `progd_memo(i:int,lmots:[int],L:int,memo:{int:int})` avec la variable `memo`, dictionnaire initialisé en dehors de la fonction par `memo={len(m):0}`

On donne finalement une fonction utilisant la méthode de calcul de bas en haut. Cette fonction renvoie le coût optimal au problème de découpage de texte global de manière équivalente à la fonction `progd_memo`.

```

1 | def progd_bashaut(lmots:[int],L:int)->int:
2 |     M=[0]*(len(lmots)+1)
3 |     for i in range(len(lmots)-1,-1,-1):
4 |         mini,indi=float("inf"),-1
5 |         for j in range(i+1,len(lmots)+1):
6 |             d=M[j]+cout(i,j-1,lmots,L)
7 |             if d<mini:
8 |                 mini,indi=d,j
9 |             M[i]=mini
10 |     return M[0]

```

□ **Q24** Analyser, en fonction de n nombres de mots, la complexité temporelle asymptotique associée à l'algorithme récursif naïf (fonction `algo_recursif`) ainsi qu'à l'algorithme de programmation dynamique de bas en haut (fonction `progd_bashaut`) et conclure sur l'intérêt de l'algorithme de programmation dynamique. On ne comptera que les opérations de type additions/soustractions. Il n'est pas attendu de développements mathématiques poussés, les résultats peuvent être donnés sans justification.

On modifie légèrement la fonction `progd_bashaut` en ajoutant un argument `t` en plus des variables précédentes pour extraire les valeurs d'indices de découpe de lignes.

```

1  def progd_bashaut(lmots:[int],L:int,t:[int])->int:
2      M=[0]*(len(lmots)+1)
3      for i in range(len(lmots)-1,-1,-1):
4          mini,indi=float("inf"),-1
5          for j in range(i+1,len(lmots)+1):
6              d=M[j]+cout(i,j-1,lmots,L)
7              if d<mini:
8                  mini,indi=d,j
9              t[i]=indi
10             M[i]=mini
11     return M[-1]

```

`t[i]` est la valeur de l'indice dans la liste `lmots` correspondant au placement optimisé sur une même ligne des mots d'indice `i` jusqu'à `t[i]` exclus. Cette liste `t` est modifiée en place dans la fonction. Pour l'exemple étudié précédemment (cas b)), on a ainsi obtenu la liste : `t=[2, 3, 4, 4, 5]`.

On dispose de la liste des mots (chaînes de caractères cette fois-ci) notée `mots`.

La fonction `lignes(mots:[str],t:[int],L:int)->[[str]]` doit renvoyer une liste de listes de mots (chaque sous-liste correspond à une ligne) en fonction de la liste `t` donnée par l'algorithme. La fonction `lignes(["Ut","enim","ad","minima","veniam"],[2,3,4,4,5],10)` renvoie :

```
[[["Ut","enim"],["ad","minima"],["veniam"]].
```

De même, en prenant,

```

t=[2, 3, 5, 5, 5, 6, 7, 9, 11, 11, 11]
L=15
mots=[["Lorem","ipsum","dolor","sit","amet,","consectetur","adipiscing",
"elit.", "Sed", "non", "risus ."]

```

la fonction `lignes(mots,t,L)` renvoie :

```

[[["Lorem","ipsum"],
["dolor","sit","amet,"],
["consectetur"],
["adipiscing"],
["elit.", "Sed"],
["non", "risus ."]]

```

□ **Q25** Proposer une implémentation de cette fonction `lignes(mots:[str],t:[int],L:int)`

Il reste à écrire une fonction `formatage(lignesdemots: [[str]],L:int)` qui renvoie une chaîne de caractères correspondant à la justification du paragraphe à partir des listes de mots par ligne `lignesdemots` et de la longueur maximale `L` d'une ligne en termes de caractères et espaces. Les retours à la ligne seront représentés par le symbole "`\n`". Les espaces devront être répartis équitablement entre les mots pour que la justification se fasse bien entre la marge gauche et la marge droite (en respectant la longueur `L` maximale imposée). On obtient par exemple pour `L=10` :

```
ut    enim
ad  minima
veniam
```

□ **Q26** Proposer une implémentation de cette fonction `formatage(lignesdemots: [[str]],L:int)->str` qui renvoie la chaîne de caractères justifiée.

Fin de l'épreuve.



ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE PC

INFORMATIQUE

Durée : 3 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.
- Ne pas utiliser de correcteur.
- Écrire le mot FIN à la fin de votre composition.

Les calculatrices sont interdites.

Le sujet est composé de trois parties.

L'épreuve est à traiter en langage **Python** sauf pour les bases de données.

Les différents algorithmes doivent être rendus dans leur forme définitive sur le Document Réponse dans l'espace réservé à cet effet en respectant les éléments de syntaxe du langage (les brouillons ne sont pas acceptés).

La réponse ne doit pas se limiter à la rédaction de l'algorithme sans explication, les programmes doivent être expliqués et commentés.

Énoncé et Annexe : 16 pages

Document Réponse (DR) : 12 pages

Seul le Document Réponse doit être rendu dans son intégralité.

Reconnaissance optique de caractères

Introduction

La reconnaissance optique de caractères (OCR) existe depuis de nombreuses années mais les récents travaux d'intelligence artificielle (apprentissage profond) ont considérablement augmenté les performances de la reconnaissance de documents.

L'objectif du travail proposé est de découvrir différentes étapes de la numérisation d'un document en explorant plusieurs algorithmes utilisés pour obtenir au final un document éditabile conforme à l'original.

Le sujet abordera les points suivants :

- acquisition d'un document et pré-traitement dans le but d'obtenir une image numérique pertinente ;
- reconnaissance du contenu qui correspond à l'extraction du texte et de sa structure ;
- reconnaissance des caractères par identification à l'aide d'une base de données.

Partie I - Acquisition d'un document

L'acquisition du document est obtenue généralement par balayage optique. Le résultat est rangé dans un fichier de points (pixels) dont la taille dépend de la résolution.

Une image en couleurs est stockée dans une matrice `imgC` de p lignes (pixels en hauteur), q colonnes (pixels en largeur) dont chaque élément est un triplet. Chaque valeur du triplet de couleur (rouge, vert, bleu) est un entier compris entre 0 et 255. La résolution est exprimée en nombre de pixels par pouce (ppp). La valeur d'un pouce est environ égale à 2,5 cm.

Q1. Chaque entier représentant une couleur est représenté, en binaire, sous la forme d'un mot constitué de bits 0 et de 1. Donner la taille de ce mot pour qu'il puisse représenter tous les entiers compris entre 0 et 255. Indiquer les dimensions (en pixels) d'une image en couleurs au format A4 (21 cm x 29,7 cm) pour une résolution de 300 ppp. En déduire alors la taille en bits du fichier image correspondant.

Pour diminuer la taille du document afin de pouvoir plus facilement le traiter, on réalise tout d'abord une conversion en niveaux de gris de l'image.

L'image en niveau de gris est une matrice `imgG` à p lignes et q colonnes où chaque valeur est un entier entre 0 (pixel noir) et 255 (pixel blanc).

La formule utilisée pour déterminer la valeur d'un pixel gris en fonction des trois couleurs d'un pixel (R rouge, G vert, B bleu) est la suivante : $pixGris = 0,299 * R + 0,587 * G + 0,114 * B$.

De manière générale, on nomme le type `array` pour représenter une matrice sous la forme d'une liste de listes dont les éléments de la liste interne pourront être des triplets pour les images en couleurs ou des entiers pour les images en niveau de gris.

On introduit les fonctions :

- `dimension(img:array)` -> tuple qui renvoie le triplet $(p, q, 3)$ pour une image en couleurs et le triplet $(p, q, 1)$ pour une image en niveau de gris ;
- `initialise(p:int, q:int, valeur:int)` -> array qui renvoie une image de dimensions (p, q) où tous les pixels sont initialisés à une même valeur `valeur`.

On donne la fonction permettant de convertir en niveau de gris l'image en couleurs.

```
def conversion_gris(imgC :array )->array :
    n0,n1,_ = dimension(imgC)
    img = initialise(p,q,0)
    for i in range(n0):
        for j in range(n1):
            r,g,b = imgC[i][j]
            val = 0.299 * r + 0.587 * g + 0.114 * b
            img[i,j] = int(val)
    return img
```

Q2. Donner la complexité de la fonction `conversion_gris(imgC:array)->array`.

La première étape du prétraitement est la **binarisation**. Cela consiste à remplacer les pixels en niveaux de gris par des pixels noirs (valeur 0) ou blanc (valeur 255) uniquement. Pour cela, la valeur du pixel gris est comparée à une valeur seuil notée `seuil`.

Q3. Proposer une fonction `binarisation(imgG:array, seuil:int)->array` qui convertit une image en niveau de gris en image en noir et blanc en imposant une valeur 255 pour tout pixel de valeur strictement supérieure au seuil.

La difficulté de cette technique de binarisation est le choix de la valeur seuil pour des images ayant des problèmes d'éclairage. Nous verrons que la technique de restauration étudiée par la suite peut être utilisée pour remplacer la binarisation par seuil standard.

Partie II - Reconnaissance du document

II.1 - Rotation de l'image

L'image scannée peut avoir un problème de rotation qu'il convient de corriger afin d'appliquer l'algorithme de reconnaissance des caractères (**figure 1**).

Beauté, limpidité, pureté, sérénité... La liste des louanges attribuées aux lacs naturels d'altitude est aussi longue que leur nombre est grand.
 Creusés çà et là par les "géants de glace" pour la plupart, ces joyaux de la montagne, aux couleurs chatoyantes et surprenantes, ont longtemps servi de toile de fond aux récits des conquêtes des sommets qui les couronnent.
 C'est l'avènement du tourisme en montagne et plus particulièrement l'en-gouement pour la randonnée qui les sortira de l'ombre et leur donnera tout l'éclat qu'ils méritent.
 Écosystèmes fragiles, ces réserves d'eau qui n'ont de potentiel que leur magie, sont les miroirs de nos belles montagnes qu'aucune pierre n'est enco-re venue briser... pour notre plus grand bonheur que nous vous invitons à partager.

Figure 1 - Image avec un problème de rotation lors de l'acquisition numérique

Le paramétrage de l'image pour la rotation est donné sur la **figure 2**. L'image est de dimension (p, q) (possède p lignes et q colonnes). Pour faire tourner le point de coordonnées (i, j) autour du point O centre de l'image d'un angle α , on applique une rotation à l'aide d'une matrice de rotation :

$$\begin{pmatrix} n_i - p//2 \\ n_j - q//2 \end{pmatrix} = \begin{pmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} i - p//2 \\ j - q//2 \end{pmatrix}.$$

On obtient alors un nouveau point de coordonnées (n_i, n_j) .

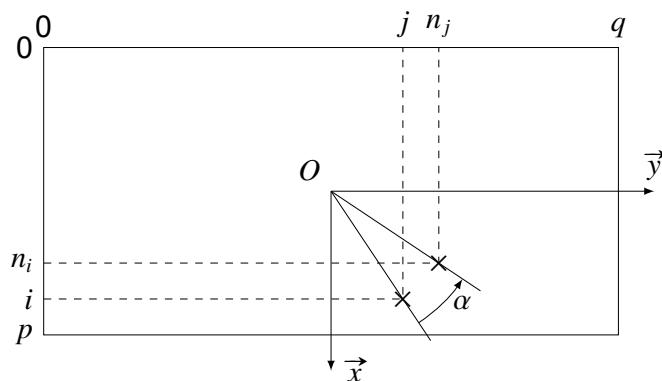


Figure 2 - Paramétrage de l'image pour la rotation

Naïvement, on pourrait penser que pour réaliser la rotation, il suffit de parcourir chaque pixel de l'image intiale en lui appliquant la rotation définie précédemment. Mais les indices étant des entiers, on se rend compte que certains pixels de la nouvelle image ne sont jamais calculés (**figure 3**) et qu'il peut apparaître des problèmes de dépassement de taille d'image.

Beauté, limpideur, pureté, sérénité...

Figure 3 - Rotation naïve de l'image initiale

L'algorithme de rotation consiste donc, pour chaque pixel de la nouvelle image de coordonnées (n_i, n_j) , à trouver ses coordonnées (i, j) par une rotation d'angle $-\alpha$ dans l'image initiale. La position du pixel virtuel ainsi trouvée est en fait un couple de réels (x, y) . Le pixel virtuel est ainsi entouré de 4 pixels dans l'image initiale dont les abscisses sont comprises entre `int(x)` et `int(x)+1` et les ordonnées entre `int(y)` et `int(y)+1` (**figure 4**).

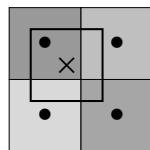


Figure 4 - Illustration du pixel trouvé entouré des 4 pixels voisins dans l'image initiale

Pour trouver la valeur du pixel virtuel, on utilise la valeur des 4 pixels voisins en réalisant une approximation bilinéaire qui consiste :

- en prenant les deux pixels voisins de la première ligne, à trouver la valeur du niveau de gris du pixel virtuel en supposant une évolution linéaire selon la coordonnée y entre le pixel de gauche et le pixel de droite ;
- à faire de même en prenant les pixels de la deuxième ligne ;
- enfin en travaillant sur la coordonnée x , à supposer une évolution linéaire entre les deux valeurs trouvées aux deux étapes précédentes.

On dispose d'une fonction :

```
lineaire(x:float, x0:int, x1:int, pix0:int, pix1:int) -> float
```

qui renvoie le flottant `val`, approximation linéaire au point `x` des valeurs `pix0` prise au point `x0` et `pix1` prise au point `x1`.

Si les coordonnées du point virtuel (x, y) se situent en dehors de l'image, alors la valeur du pixel sur l'image tournée sera prise de couleur blanche, c'est-à-dire égale à 255.

- Q4.** Choisir la fonction `bilineaire(im:array, x:float, y:float) -> int` parmi les quatre propositions, données dans le **DR**, permettant de respecter les spécifications.
- Q5.** Compléter la fonction `rotation(im:array, angle:float) -> array` donnée dans le **DR** qui prend en argument une image en niveau de gris et un angle en degré et qui renvoie une nouvelle image tournée de l'angle `angle` donné en degré. On veillera à initialiser l'image par une image complètement blanche (pixels de valeur 255).

On suppose définie une fonction :

```
prod_matrice_vecteur(M: array, v: list) -> list
```

qui renvoie le vecteur colonne (sous forme de liste) résultat de la multiplication de la matrice `M` par le vecteur colonne `v`.

Une manière d'implémenter la fonction linéaire est la suivante :

```
def linéaire(x :float, x0 :int, x1 :int, pix0 :int, pix1 :int)-> float :
    return (x-x0)*(pix1-pix0)/(x1-x0) + pix0
```

Il se trouve qu'en pratique, si on utilise des tableaux Numpy pour représenter les matrices, on peut être tenté de forcer les entiers à être du type `uint8` qui correspond à un entier non signé (d'où le « u » pour « unsigned ») stocké sur 8 bits.

Q6. Donner une raison pour laquelle il serait intéressant de se contraindre à 8 bits et expliquer le gain qu'il pourrait en découler en pratique.

Expliquer quel problème pourrait apparaître en réfléchissant au résultat de la soustraction $18 - 23$ où 18 et 23 sont tous deux des entiers non signés sur 8 bits et où le résultat est lui aussi obligatoirement un entier non signé sur 8 bits.

En utilisant une telle structure (où `pix0` et `pix1` sont des entiers de type `uint8`), on se retrouve avec l'image pixellisée de la **figure 5**, ce qui n'est effectivement pas un résultat voulu, l'image attendue étant donnée sur la **figure 6**. L'angle choisi pour cette rotation n'est pas la valeur optimale assurant l'horizontalité du texte.

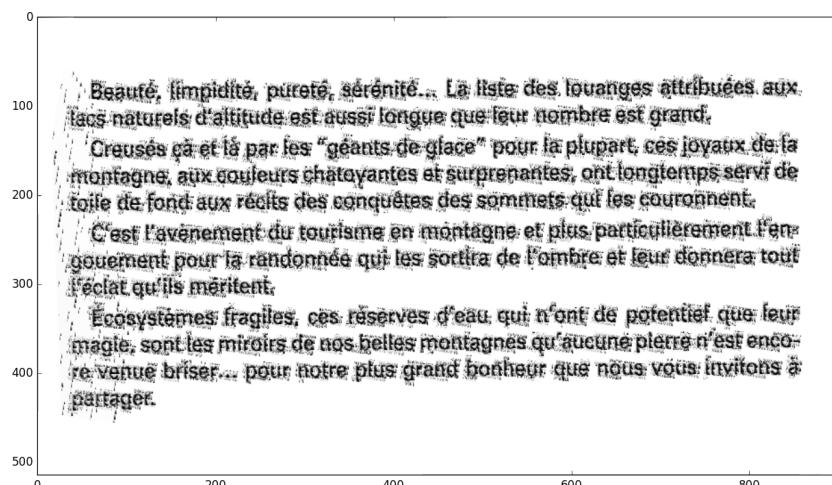


Figure 5 - Problème de pixellisation lors de la rotation

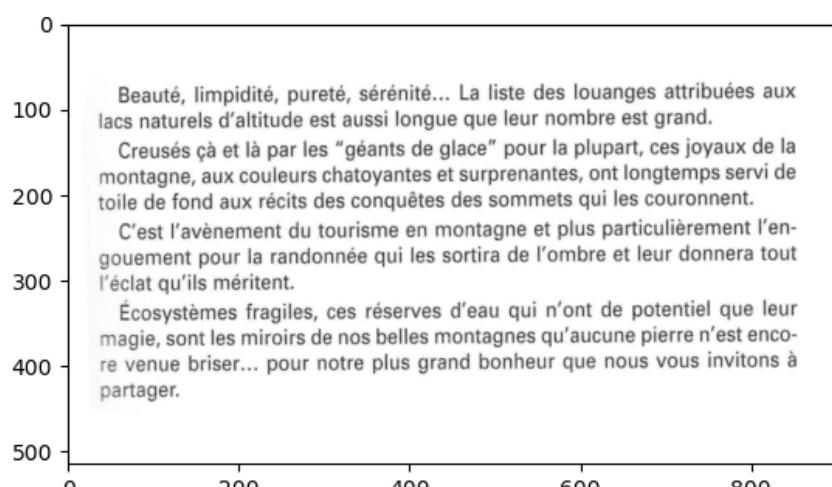


Figure 6 - Figure attendue après la rotation

II.2 - Segmentation

La segmentation consiste à découper l'image en plusieurs éléments de manière à pouvoir ensuite traiter chacun des éléments. Il faut dans l'image pouvoir dissocier les lignes, les mots puis les lettres. L'idée est de construire la liste du nombre de pixels noirs par ligne.

On peut ensuite détecter les lignes en sélectionnant les zones où il y a majoritairement des pixels blancs, ce qui correspond aux zones sans texte.

On applique ensuite le même principe pour détecter les mots et les lettres en comptant les pixels blancs verticalement.

On travaille sur une image binarisée, c'est-à-dire ne contenant que des pixels blancs (255) ou des pixels noirs (0).

- Q7.** Proposer une fonction `histo_lignes(im:array)->list` qui prend en argument une image binarisée et renvoie une liste contenant le nombre de pixels noirs de chaque ligne sans utiliser la fonction `count`.

La fonction appliquée au texte précédent, après rotation, renvoie la liste présentée sous forme d'un histogramme sur la **figure 7**.

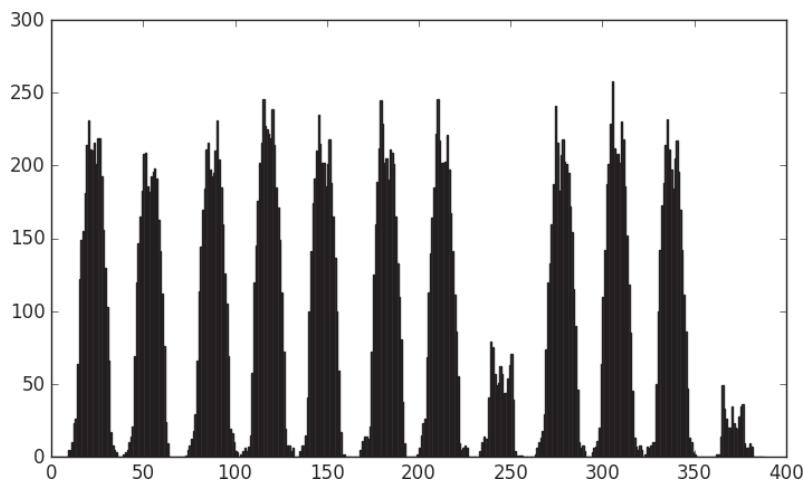


Figure 7 - Histogramme de détection des lignes

On peut observer des blocs de pixels noirs correspondant bien aux lignes. Il suffit maintenant de détecter le début d'un bloc comme étant un élément nul suivi d'un élément non nul et de détecter la fin d'un bloc comme étant un élément nul précédé d'un élément non nul.

- Q8.** Compléter sur le **DR** la fonction `detecter_lignes(liste:list)->list` prenant en argument une liste contenant le nombre de pixels noirs par ligne de l'image et qui renvoie une liste de couples (début ligne, fin ligne).

Cette fonction appliquée à notre exemple renvoie : `[[8, 36], [38, 64], [73, 102], [102, 132], [134, 160], [167, 193], [198, 227], [232, 257], [262, 291], [293, 322], [322, 351], [361, 382]]`.

En appliquant cette détection de ligne directement sur l'image mal orientée, il en résulte une erreur de détection. En effet, si on observe l'histogramme dans ce cas (**figure 8**), on constate qu'il n'y a plus de zones avec des pixels blancs détectées.

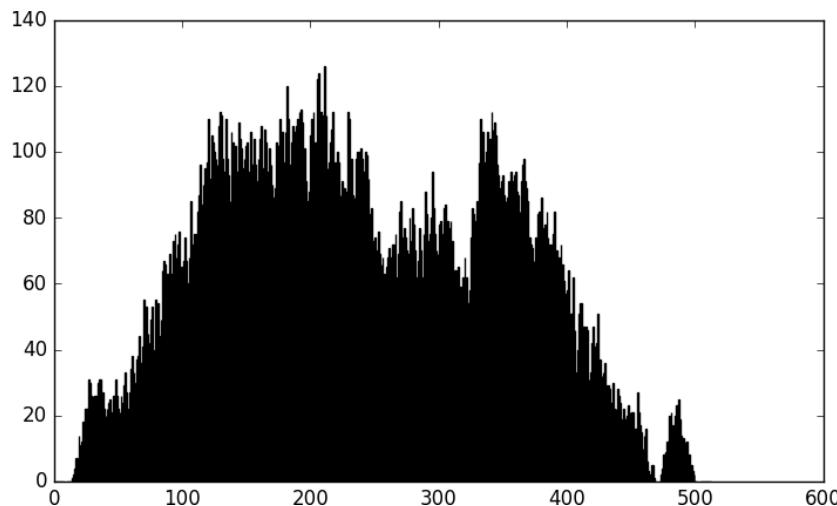


Figure 8 - Histogramme de détection des lignes sur la figure mal orientée

Il est donc nécessaire d'implanter un algorithme permettant de détecter automatiquement la bonne orientation en travaillant sur la maximisation du nombre de 0 dans la liste fournie par la fonction `histo_ligne`. On suppose que dans l'intervalle des angles de recherche, la fonction possède un unique maximum (pas d'extremum local).

L'algorithme peut être décrit de la manière suivante :

- partant d'un intervalle de départ $[a, b]$ avec les angles a et b , on calcule :
 - le nombre de 0 de la liste fournie par la fonction `histo_ligne` pour les deux orientations a et b ,
 - le nombre de 0 de la liste fournie par la fonction `histo_ligne` pour l'orientation du milieu, noté $c = \frac{a+b}{2}$;
- on itère tant que l'intervalle de recherche $[a, b]$ est plus grand qu'un epsilon donné :
 - on calcule le nombre de 0 pour l'orientation au milieu, noté ac , de l'intervalle $[a, c]$,
 - on calcule le nombre de 0 pour l'orientation au milieu, noté cb , de l'intervalle $[c, b]$,
 - on cherche où se situe le maximum entre ac , c ou cb ,
 - on en déduit le nouvel intervalle de recherche, comme étant celui entourant le maximum. Par exemple, si le maximum est en c , alors le nouvel intervalle sera $[ac, cb]$.

Q9. Justifier que cet algorithme se termine.

Donner le nom de la méthode utilisée pour réaliser cet algorithme et préciser en justifiant le nombre d'itérations nécessaires pour obtenir la solution avec une précision notée ε .

Q10. Compléter la fonction `rotation_auto(im:array, a:float, b:float) -> array` qui prend en argument une image `im` et les angles initiaux `a` et `b` et qui renvoie l'image avec la bonne orientation.

Après avoir séparé les lignes, en appliquant une méthode similaire, on peut extraire les caractères sur chacune de ces lignes. La **figure 9** montre les premiers caractères détectés par l'algorithme.

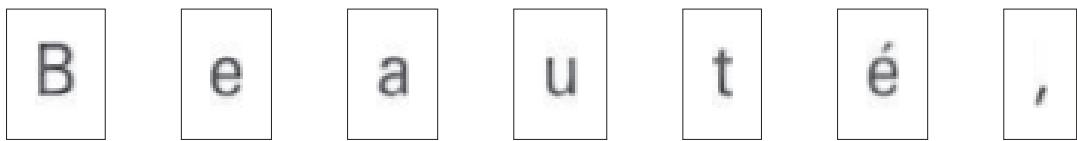


Figure 9 - Premiers caractères détectés par l'algorithme

II.3 - Restauration d'image

Les images de caractères peuvent être bruitées compte tenu d'une mauvaise résolution ou de parasites apparaissant pendant un scan. De même, la technique de binarisation proposée initialement ne donne pas toujours un résultat correct si le seuil est mal choisi.

La méthode du flot maximal (ou méthode de la coupe minimale) reposant sur la représentation par un graphe de l'image à restaurer est souvent utilisée pour pallier ces problèmes.

La librairie `maxflow` disponible sous Python propose des fonctions déjà existantes pour traiter une image bruitée.

La fonction globale de traitement de l'image est la suivante :

```
import numpy
import maxflow
def graph_cut(img :array )->array :
    img = numpy.array(img) #Conversion en array de Numpy pour un usage
    plus facile ensuite
    g = maxflow.Graph[int]()
    #création du graphe
    nodeids = g.add_grid_nodes(dimension(img))
    g.add_grid_edges(nodeids, 5)
    g.add_grid_tedges(nodeids, img, 255-img)
    g.maxflow()
    sgm = g.get_grid_segments(nodeids)
    img2 = numpy.int_(numpy.logical_not(sgm))
    return img2
```

L'objet des questions de cette sous-partie est de comprendre chaque ligne de cette fonction et d'illustrer la méthode sur un exemple basique d'une image test (3x3) constituée de 9 pixels en niveau de gris (pixels compris entre 0 (noir) et 255 (blanc)).

1 : 0	2 : 210	3 : 190
4 : 20	5 : 100	6 : 200
7 : 10	8 : 5	9 : 255

Figure 10 - Exemple d'image à restaurer

Les valeurs des pixels de l'exemple sont les suivantes :

```
[ [ 0, 210, 190 ],
  [ 20, 100, 200 ],
  [ 10, 5, 255 ] ]
```

La méthode utilise la représentation par graphe pondéré constitué de n sommets et m arêtes. Chaque sommet correspond à un pixel de l'image. `nodeids` est donc l'ensemble des sommets du graphe correspondant à l'image de taille `dimension(img)`.

Les arêtes reliant deux sommets sont ensuite construites à l'aide de l'instruction `g.add_grid_edges(nodeids, 5)` entre un sommet et ses potentiels 4 voisins adjacents. À chaque arête e reliant deux sommets, un poids $w(e)$ de valeur fixe 5 est associé. Cette pondération va représenter la capacité maximale du flot définie par la suite.

Q11. Représenter le graphe correspondant à l'image de (3x3) pixels en précisant sur chaque arête la capacité maximale de 5.

Pour mettre en place la méthode de flot maximal, il est nécessaire d'introduire deux sommets supplémentaires (appelés source S et puits P) qui sont reliés par des arêtes à tous les sommets précédents. Sur chaque arête entre le sommet S et les sommets "pixels" on utilise les valeurs des pixels comme poids, et sur les arêtes entre les sommets "pixels" et le sommet P on utilise le complément à 255 des valeurs des pixels. C'est le rôle de la ligne `g.add_grid_tedges(nodeids, img, 255-img)`.

Q12. Compléter sur le DR la partie supérieure de la matrice de capacités correspondant au graphe complet de l'exemple en prenant l'ordre suivant pour les sommets : S, 1, 2, ..., 9, P avec pour valeurs les poids précédemment introduits pour chaque arête. Le poids est nul si le sommet i n'est pas relié au sommet j .

La fonction `g.maxflow()` calcule le flot maximal, ce qui permettra par la suite de partitionner les sommets.

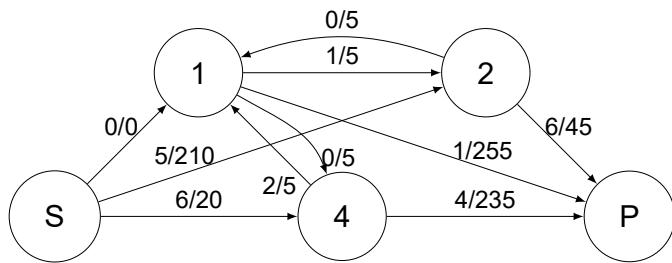
Le flot est une notion similaire à un flux de fluide qui s'écoulerait de la source vers le puits. Mathématiquement, le flot est une fonction f définie de l'ensemble des arêtes $e \in E$ vers l'ensemble des réels \mathbf{R} . Cette fonction vérifie les propriétés suivantes :

- $\forall e = (p, q) \in E$ (avec p, q deux sommets), $f(p, q) = -f(q, p)$, le flot dans le sens q vers p est l'opposé du flot dans le sens p vers q ;
- pour tout sommet p autre que S et P : $\sum_{e=(p,j) \in E} f(e) = 0$, la somme des flots arrivant et sortant d'un sommet est nulle, ce qui est similaire à la loi de Kirchoff;
- pour toute arête $e \in E$, $f(e) \leq w(e)$, le flot ne peut pas dépasser la capacité maximale définie initialement.

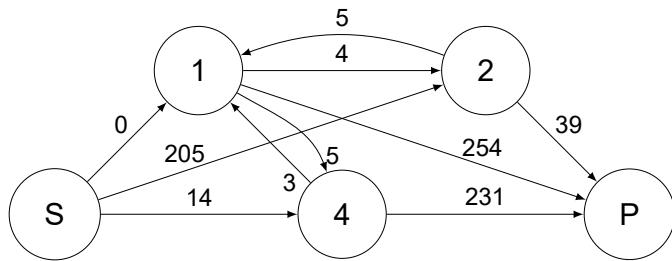
On pourrait définir une matrice de flots similaire à la matrice de capacités qui contiendrait les valeurs des flots au lieu des capacités.

On passe du graphe non orienté que nous venons de décrire à un graphe orienté. Les arêtes faisant intervenir la source sont alors orientées de la source vers les sommets (flot sortant de la source); celles faisant intervenir le puits sont orientées des sommets vers le puits (flot entrant dans le puits); les arêtes entre des sommets i et j correspondant à des pixels sont dédoublées (une de i vers j , l'autre de j vers i) et ont chacune une capacité maximale égale à 5. La **figure 11** montre un exemple de flot sur une partie seulement du graphe de l'exemple étudié. Les étiquettes de la forme i/j représentent pour i la valeur du flot et pour j la valeur de la capacité maximale.

Le flot est maximal lorsque les flots partant de la source S sont maximaux tout en respectant toutes les règles précédentes. On dit qu'une arête est saturée lorsque le flot de cette arête est égal à sa capacité.

**Figure 11** - Exemple d'extrait de graphe avec flot

Pour déterminer le flot maximal, une méthode possible consiste à saturer des arêtes. Pour cela, on utilise un graphe complémentaire appelé graphe résiduel, obtenu à partir du graphe de flot sur lequel on indique sur chaque arête $e \in E$ la capacité résiduelle (dans un sens et dans l'autre) : $r(e) = w(e) - f(e)$. Si une arête est étiquetée 0 sur le graphe résiduel, alors il n'est plus possible d'emprunter cette arête pour construire le chemin de longueur minimal. La **figure 12** montre le graphe résiduel associé au graphe avec flot de la **figure 11**.

**Figure 12** - Exemple de graphe résiduel

On utilise l'algorithme d'Edmonds-Karp : à partir du flot nul, on cherche itérativement un plus court chemin C (c'est-à-dire un chemin où la somme des étiquettes du graphe résiduel en parcourant les arêtes le constituant est minimal et comportant le moins d'arêtes) de la source au puits sur lequel il n'y a pas d'arête saturée (c'est-à-dire un chemin pour lequel aucun des arêtes correspondantes du graphe résiduel n'est pondérée par 0). On rajoute alors autant de flots que possible à ce chemin (c'est-à-dire on sature l'arête qui a une capacité résiduelle minimale).

L'algorithme de recherche du flot maximal est le suivant en pseudo-code :

```

Initialisation :
poser f(e) = 0 pour toute arête e
définir le graphe résiduel initial
définir un chemin C de S à P dans le graphe résiduel de longueur minimale
tant qu'il existe un chemin C de S à P dans le graphe résiduel faire
    prendre un chemin C de longueur minimale
    a = min(r(e)| e dans C)
    pour tout e dans C faire
        f(e) = f(e) + a
    fin pour
    mettre à jour le graphe résiduel
fin tant que
  
```

Q13. Appliquer cet algorithme sur les graphes du **DR** en précisant à chaque étape le chemin choisi et la valeur de l'augmentation du flot jusqu'à sa terminaison. Le graphe de gauche représente le graphe de flot et le graphe de droite le graphe résiduel. On donne le graphe initial avec un flot nul ainsi que le graphe résiduel associé.

Pour transformer l'image en niveau de gris en une image noir et blanc, c'est-à-dire pour séparer les pixels entre ceux qui prennent la valeur 0 et ceux qui prennent la valeur 255, on va réaliser une coupe dans le graphe des pixels. On définit l'ensemble A contenant la source S et certains sommets ainsi que l'ensemble B contenant le puits P et les sommets restants.

La capacité de la coupe est la somme des capacités des arcs orientés de A vers B.

Par exemple, supposons que nous ayons coupé le graphe entre les ensembles A = {S, 1, 2} et B = {P, 4}. En sommant les capacités maximales des arêtes orientées allant d'un sommet de A vers un sommet de B, on obtient une capacité de coupe de $20+5+255+45 = 325$.

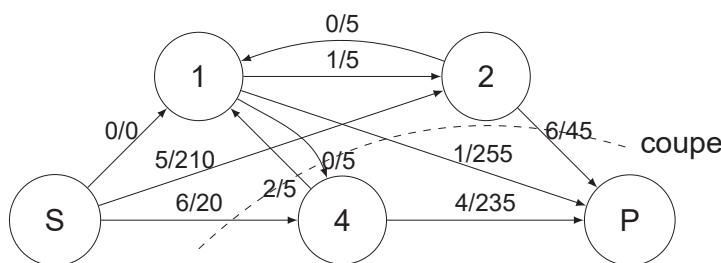


Figure 13 - Coupe dans un graphe

L'algorithme d'Edmonds-Karp permet de construire un flot maximal, c'est-à-dire un flot dont la somme des arêtes arrivant au puits est maximale. Le théorème du "flot maximal et coupe minimale" assure que la valeur de ce flot maximal est égale à la valeur de coupe minimale.

Pour réaliser cette coupe, on met dans l'ensemble A la source S et tous les sommets accessibles, depuis S, par des arêtes non saturées ; on met dans l'ensemble B les sommets restants.

L'appel `g.get_grid_segments(nodeids)` renvoie une liste indiquant, pour chacun des sommets, s'il appartient ou non au même ensemble que la source.

Q14. Dans l'exemple précédent, indiquer les deux ensembles A et B en précisant la valeur du flot maximal et en vérifiant que la capacité de coupe réalisée correspond bien à une valeur égale à celle du flot maximal.

Lorsqu'on applique la fonction précédente sur l'image d'un caractère, on obtient la nouvelle image de la **figure 14**.

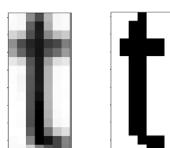


Figure 14 - Caractère scanné et caractère après traitement par la fonction `graph_cut`

Q15. Indiquer pour cette image de la **figure 14** à quoi correspondent les ensembles A et B. Analyser le résultat obtenu.

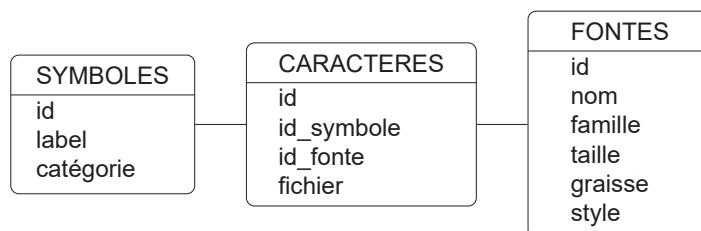
Partie III - Détermination des caractères

Une fois les images de lettres isolées, il s'agit de reconnaître la lettre correspondante. Différentes méthodes peuvent être employées. Nous allons étudier une méthode d'apprentissage automatique basée sur les K plus proches voisins.

Le principe de cette méthode consiste à comparer chaque caractère à un ensemble de caractères définis dans une base de données.

III.1 - Analyse de la base de données de caractères

La base de données contient des informations sur chaque caractère selon le type de fonte, la taille, la graisse... Trois tables sont utilisées.



La table SYMBOLES contient les attributs suivants :

- id : identifiant d'un symbole (entier), clé primaire ;
- label : nom du symbole ("A", "a", "1", "é", "!" ...) (chaîne de caractères) ;
- catégorie : parmi majuscule, minuscule, chiffre, spécial (dont accent) (chaîne de caractères).

La table CARACTERES contient les attributs suivants :

- id : identifiant d'un caractère (entier), clé primaire ;
- id_symbole : identifiant du nom du symbole (entier) ;
- id_fonte : identifiant du type de fonte (entier) ;
- fichier : nom du fichier image correspondant (chaîne de caractères).

La table FONTES contient les attributs suivants :

- id : identifiant d'une fonte (entier), clé primaire ;
- nom : nom de la fonte ("Arial", "Times new roman", "Calibri", "Zurich", ...) (chaîne de caractères) ;
- famille : nom de la famille dont fait partie la fonte ("humane", "garalde", "réale", "didone", "scripte", ...) (chaîne de caractères) ;
- taille : dimension en hauteur des caractères en pixels (entier) ;
- graisse : type de graisse ("léger", "normal", "gras", "noir", ...) (chaîne de caractères) ;
- style : type de style ("romain", "italique", "ombré", "décoratif", ...) (chaîne de caractères).

Q16. Écrire une requête SQL permettant d'extraire les identifiants des fontes dont le nom est "Zurich", de style "romain" et dont la taille est comprise entre 10 et 16 pixels.

Q17. Écrire une requête SQL permettant d'extraire tous les noms de fichiers des caractères qui correspondent au symbole de label "A".

Q18. Écrire une requête SQL permettant d'indiquer le nombre de caractères correspondant à la fonte "Zurich", de style "romain" et dont la taille est comprise entre 10 et 16 pixels groupés selon les labels des symboles.

III.2 - Classification automatique des caractères

Dans la suite du sujet, on suppose qu'on dispose d'une liste `fichiers_car_ref` contenant les noms des fichiers images d'un grand nombre de caractères ayant des fontes proches de celles du texte scanné. Le nom de chaque fichier est défini de la manière suivante :

```
nomFonte + "_" + nomCatégorie+taillePolice + "_" + idSymbole + ".png"
```

Les catégories sont définies par la liste :

```
categories = ["majuscules", "minuscules", "chiffres", "special"].
```

Les symboles considérés sont définis par la liste :

```
symboles = ["ABCDEFGHIJKLMNOPQRSTUVWXYZ", "abcdefghijklmnopqrstuvwxyz",
```

```
"0123456789", ".:;,(!?)éèàçùêûâ"]. On compte 79 symboles différents.
```

Exemple : `Zurich Light BT_majuscules18_10.png` pour la majuscule K de la police Zurich Light BT en taille 18.

On introduit la fonction suivante :

```
def lire_symbole_fichier(nomFichier:str)->str:
    car = nomFichier.split('_')
    num = car[2].split('.')[0]
    var = car[1][len(car[1])-2]
    ind = categories.index(var)
    return symboles[ind][int(num)]
```

Pour une liste L, `L.index(val)` renvoie la position de val dans la liste L.

Q19. Indiquer ce que valent les variables `car`, `num`, `var`, `ind` et ce qui est renvoyé par la fonction si `nomFichier="Zurich Light BT_majuscules18_10.png"`.

Toutes les images des caractères de référence sont lues et stockées sous forme de tableaux array. On définit un dictionnaire `carac_ref` dont les clés seront les symboles apparaissant dans la liste `symboles` (par exemple "A", "a", ...). À chaque clé sera associée une liste de tableaux array représentant des images.

La commande `img.imread(nomFichier)` permet de lire le fichier image `nomFichier` et de stocker le tableau array à deux dimensions qui représente l'image dans la variable `img`.

Q20. Écrire une fonction `lire_donnees_ref(fichiers_car_ref:list)->dict` qui prend en argument la liste des noms de fichiers images `fichiers_car_ref` et qui renvoie le dictionnaire contenant tous les tableaux catégorisés.

Un caractère à identifier est également stocké sous forme d'un tableau array nommé `carac_test`. On suppose que les dimensions de ce tableau et de tous les tableaux du dictionnaire `carac_ref` sont les mêmes.

La méthode d'identification utilisée est celle des K plus proches voisins. Elle consiste à calculer une distance entre l'image du caractère à identifier et toutes les images de référence. En notant (i, j) les coordonnées d'un pixel dans le tableau représentant l'image, p_{ij} le pixel associé à l'image du caractère à identifier et q_{ij} celui d'un caractère de référence, on calcule pour chaque caractère de référence la distance $d = \sqrt{\sum_{i,j} (p_{ij} - q_{ij})^2}$.

Les distances d sont stockées dans un dictionnaire `distances` où, pour chaque clé égale à un symbole de la liste `symboles`, on associe une liste de distances pour chaque image de référence de ce symbole.

Q21. Écrire une fonction `distance(im1:array, im2:array)->float` qui calcule la distance entre les deux images `im1` et `im2` supposées de même dimension.

Q22. Écrire une fonction `calcul_distances(carac_ref:dict, carac_test:array) ->dict` qui prend en argument le dictionnaire des tableaux catégorisés et un tableau associé au caractère à tester et qui renvoie le dictionnaire des distances.

La suite consiste à déterminer les K plus petites distances et extraire les clés correspondantes, puis parmi ces clés déterminer la clé majoritaire. Une méthode envisageable est de trier les distances par ordre croissant pour prendre les K premiers éléments. On suppose qu'il y a au total n images de caractères de référence sur l'ensemble des symboles.

Q23. En se plaçant dans le pire des cas, indiquer le nom d'une méthode de tri performante envisageable, en précisant sa complexité temporelle en fonction de n .

Une méthode plus efficace est envisagée pour extraire directement les K plus petits éléments. Elle consiste à construire par tri par insertion la liste de taille K . L'algorithme correspondant est donné dans le **DR**.

Q24. Compléter les 3 zones manquantes dans cet algorithme.

Q25. Préciser la complexité temporelle asymptotique dans le pire des cas de cet algorithme en fonction de n et de K . Comparer avec l'utilisation d'un tri classique sachant que n est grand et K ne dépassera pas 5.

Q26. Écrire une fonction `symbole_majoritaire(voisins:list) ->str` qui à partir de la liste voisins renvoyée par la fonction `Kvoisins` renvoie le symbole majoritaire.

On teste l'algorithme sur les caractères extraits dans la partie précédente ("Beauté,").

On obtient les résultats suivants.

Nombre de voisins K	Type d'éléments dans la base de données	Nombre d'éléments dans la base n	Caractères obtenus
1	fonte similaire au texte analysé	79 images correspondant aux 79 symboles	"Bssi!-, "
4	fonte similaire au texte analysé	79 images correspondant aux 79 symboles	"Bssi!-, "
1	40 fontes proches de celle du texte analysé	40*79 images correspondant aux 79 symboles	"Bsauté,"
4	40 fontes proches de celle du texte analysé	40*79 images correspondant aux 79 symboles	"Bsauté,"
1	40 fontes pour 8 polices différentes	320*79 images correspondant aux 79 symboles	"Beauté,"
4	40 fontes pour 8 polices différentes	320*79 images correspondant aux 79 symboles	"Beauté,"

Q27. Commenter les résultats obtenus.

ANNEXE

Rappels des syntaxes en Python

Fonctionnalités	Python
détermination du nombre de zéros dans la liste X	X.count(0)
définir une chaîne de caractères	mot = 'Python'
taille d'une chaîne	len(mot)
extraire des caractères (avec le même fonctionnement des indices que pour les extractions de sous-listes)	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. Dans l'exemple ci-contre, on découpe à chaque occurrence du caractère ","	mot.split(',')
ouverture d'un fichier en lecture et lecture des données (data est une liste de chaînes de caractères dont la taille est le nombre de lignes du fichier lu)	with open('nom_fichier','r') as f: data = f.readlines()

FIN

 CONCOURS COMMUN CCINP	Numéro d'inscription <input type="text" value=" "/>	Nom : _____
	Numéro de table <input type="text" value=" "/>	Prénom : _____
	Né(e) le <input type="text" value=" / / "/>	
		Filière : PC
	Épreuve de : INFORMATIQUE	
	Consignes <ul style="list-style-type: none"> • 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 	

PC5IN

DOCUMENT RÉPONSE

Ce Document Réponse doit être rendu dans son intégralité.

Q1 - Taille du mot en bits. Dimensions en pixels d'une image. Taille en bits du fichier image

--

Q2 - Complexité de la fonction `conversion_gris(imgC:array)->array`

--

NE RIEN ÉCRIRE DANS CE CADRE

Q3 - Fonction binarisation(imgG:array, seuil:int)->array

Q4 - Choix de la fonction bilineaire(im:array, x:float, y:float)->int

```
def bilineaire(im :array ,x :float ,y :float )->int :
    x0 = int(x)
    x1 = x0+1
    y0 = int(y)
    y1 = y0+1
    a = lineaire(y,y0,y1,im[x0][y0],im[x1][y1])
    b = lineaire(y,y0,y1,im[x1][y0],im[x0][y1])
    c = lineaire(x,x0,x1,a,b)
    return int(c)
```

```

def bilineaire(im :array ,x :float ,y :float )->int :
    x0 = int(x)
    x1 = x0+1
    y0 = int(y)
    y1 = y0+1
    a = lineaire(y,y0,y1,im[x0][y0],im[x0][y1])
    b = lineaire(y,y0,y1,im[x1][y0],im[x1][y1])
    c = lineaire(x,x0,x1,a,b)
    return int(c)

```

```
def bilineaire(im :array ,x :float ,y :float )->int :
    x0 = int(x)
    x1 = x0+1
    y0 = int(y)
    y1 = y0+1
    a = lineaire(y,y0,y1,im[x0][y0],im[x0][y1])
    b = lineaire(y,y0,y1,im[x0][y0],im[x1][y0])
    c = lineaire(x,x0,y1,a,b)
    return int(c)
```

```

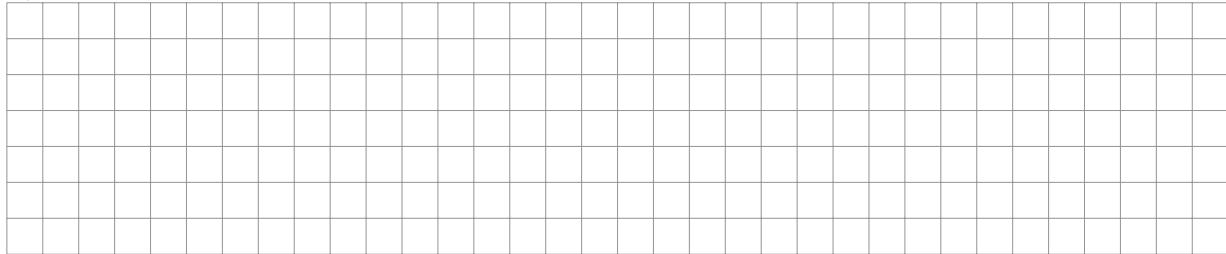
def bilineaire(im :array ,x :float ,y :float )->int :
    x0 = int(x)
    x1 = x0+1
    y0 = int(y)
    y1 = y0+1
    a = lineaire(y,y0,y1,im[x0][y0],im[x0][y1])
    b = lineaire(y,y0,y1,im[x1][y0],im[x0][y1])
    c = lineaire(y,y0,y1,a,b)
    return int(c)

```

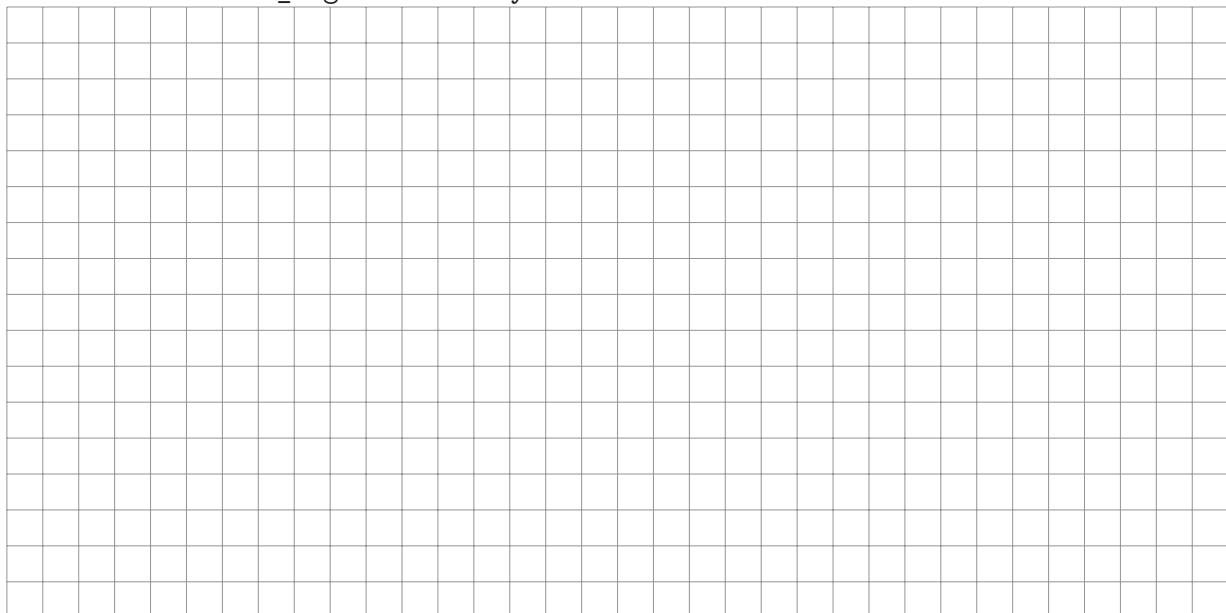
Q5 - Fonction `rotation(im:array, angle:float)->array`

```
def rotation(im :array , angle :float)->array :
    p,q,_ = dimension(im)
    imr = .....
    angr = .....
    matR = .....
    for ni in range(.....):
        for nj in range(.....):
            x, y = .....
            x = x + .....
            y = y + .....
            if .....:
                .....
    return imr
```

Q6 - Étude de la fonction linéaire



Q7 - Fonction `histo_lignes(im:array)->list`



Q8 - Fonction detecter_lignes(liste:list)->list

```
def detecter_lignes(liste :list) -> list :
    lignes = []
    i = 0
    deb = -1 #contient -1 tant qu'on parcourt des lignes de pixel blanc
    while ..... :
        #début d'une suite de lignes contenant des pixels noirs
        if ..... :
            deb = .....
        #fin d'une suite de lignes contenant des pixels noirs
        elif ..... :
            .....
            deb = .....
        .....
    return lignes
```

Q9 - Justification de la terminaison et nom de la méthode. Nombre d'itérations nécessaires

 CONCOURS COMMUN CINP	Numéro d'inscription <input type="text" value=" "/>	Nom : _____
	Numéro de table <input type="text" value=" "/>	Prénom : _____
	Né(e) le <input type="text" value=" "/> <input type="text" value=" "/> <input type="text" value=" "/>	
	Filière : PC	Session : 2023
	Épreuve de : INFORMATIQUE	
	Consignes <ul style="list-style-type: none"> • 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 	

PC5IN

Q10 - Fonction rotation_auto(im:array, a:float, b:float)->array

```

def nb_zeros(im :array , angle :float )->int :
    imr = rotation(im , angle)
    ligne = histo_ligne(imr)
    f=ligne .count(0)
    return f

def rotation_auto(im :array , a :float , b :float )->array :
    c = (a+b)/2
    fc = nb_zeros(im ,c)
    while b-a > 0.1 :#plus grand que 0.1 degré

        ac = .....
        fac = .....
        cb = .....
        fcb = .....
        maxi = max(fac ,fc ,fcb )

        if ..... == maxi :
            b = c
            c = ac
            fc = fac

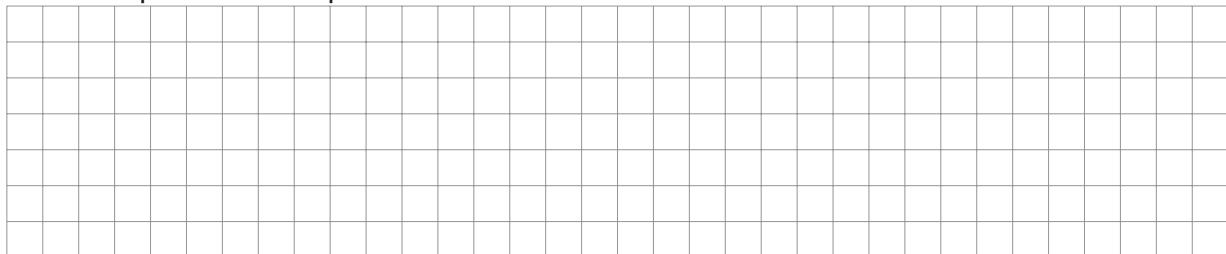
        elif ..... == maxi :
            a = ac
            b = cb
        else :
            a = .....
            c = .....
            fc = .....

    return rotation(im ,(b+a)/2)

```

NE RIEN ÉCRIRE DANS CE CADRE

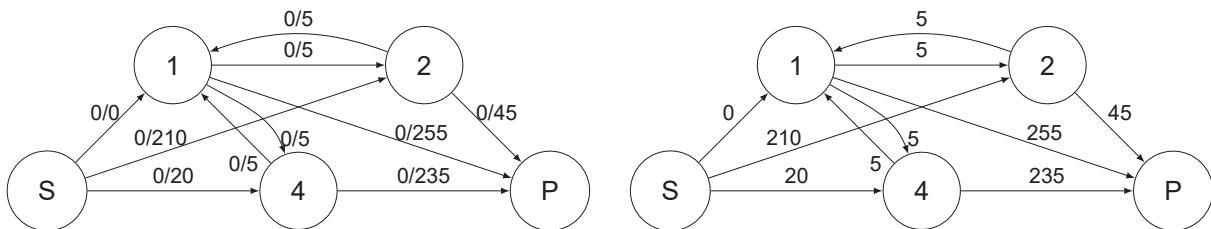
Q11 - Graphe de l'exemple de taille 3x3



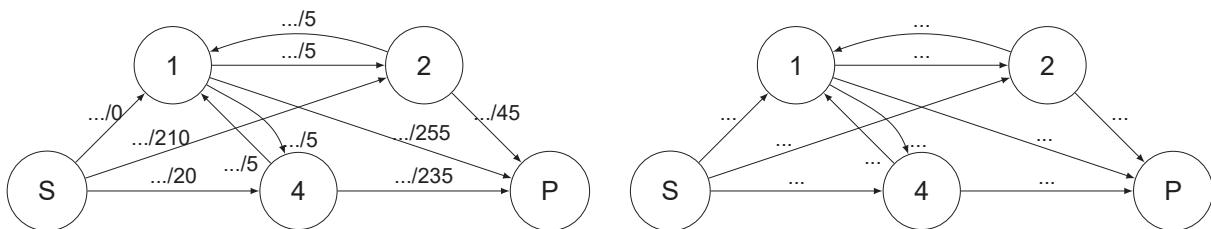
Q12 - Compléter la partie supérieure de la matrice de capacités

	S	1	2	3	4	5	6	7	8	9	P
S											
1	-										
2	-	-									
3	-	-	-								
4	-	-	-	-	-						
5	-	-	-	-	-	-					
6	-	-	-	-	-	-	-				
7	-	-	-	-	-	-	-	-			
8	-	-	-	-	-	-	-	-	-		
9	-	-	-	-	-	-	-	-	-		
P	-	-	-	-	-	-	-	-	-	-	

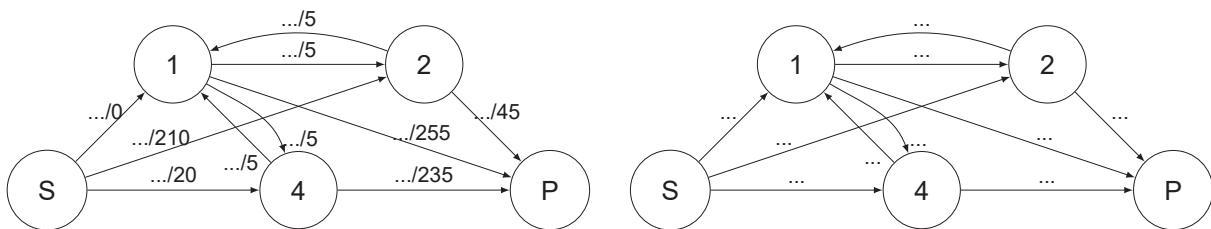
Q13 - Compléter les graphes de flot à gauche et résiduel à droite. Pour chaque étape, préciser le chemin choisi et la valeur de l'augmentation du flot



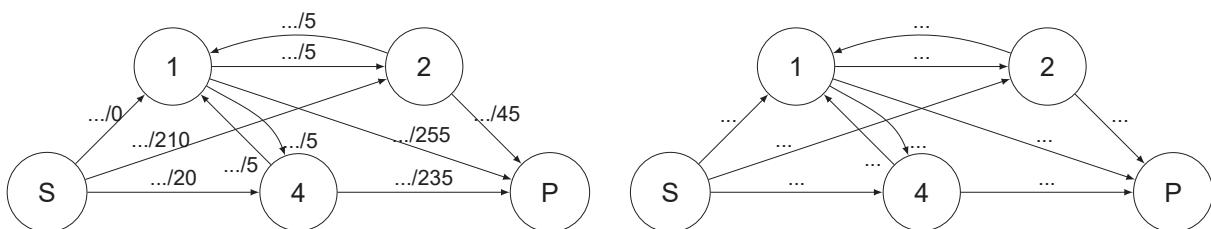
Chemin choisi :



Chemin choisi :



Chemin choisi :



Q14 - Ensembles A et B. Flot maximal. Coupe choisie et capacité de coupe

Q15 - Correspondance ensembles A et B. Analyse

Q16 - Requête SQL

Q17 - Requête SQL

Q18 - Requête SQL

Q19 - Contenu des variables car, num, var, ind. Retour de la fonction

car :

var :

num :

ind :

Retour de la fonction :

CINP CONCOURS COMMUN	Numéro d'inscription	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Nom : _____
	Numéro de table	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Prénom : _____
	Né(e) le	<input type="text"/> <input type="text"/> <input type="text"/> / <input type="text"/> <input type="text"/> <input type="text"/> / <input type="text"/> <input type="text"/> <input type="text"/>	
	Filière :	PC	Session : 2023
	Épreuve de : INFORMATIQUE		
Emplacement QR Code			
Consignes	<ul style="list-style-type: none"> • 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 		

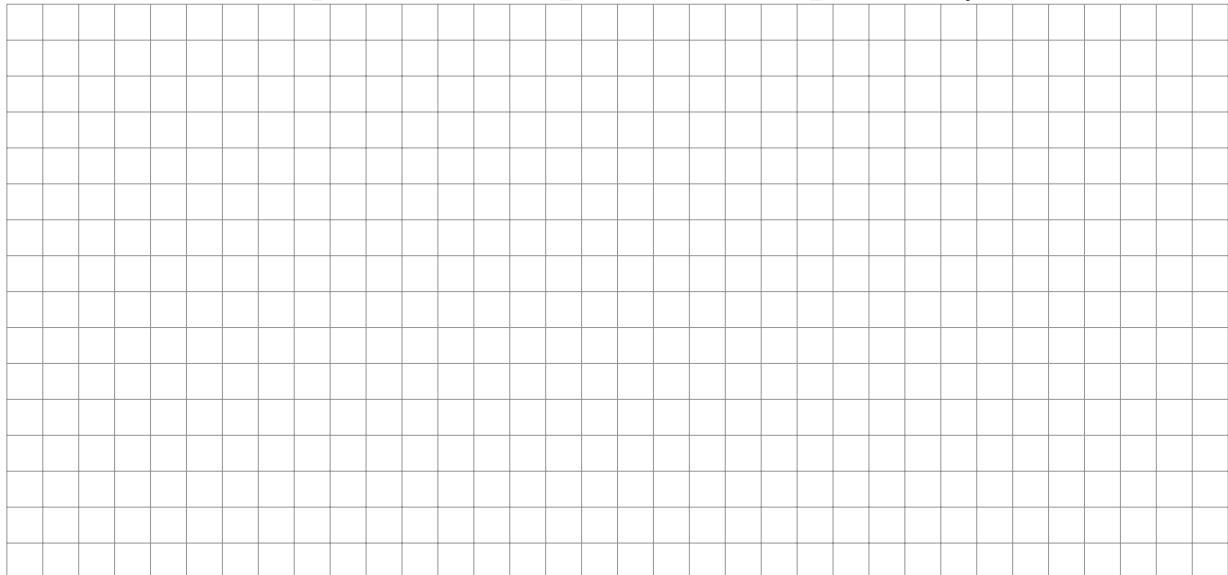
PC5IN

Q20 - Fonction lire_donnees_ref(fichier_car_ref:list) ->dict

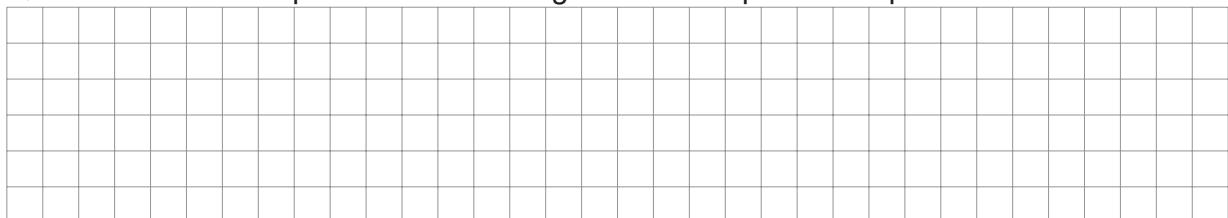
Q21 - Fonction `distance(im1:array, im2:array) -> float`

NE RIEN ÉCRIRE DANS CE CADRE

Q22 - Fonction `calcul_distances(carac_ref:dict, carac_test:array)->dict`



Q23 - Méthode de tri performante envisageable et complexité temporelle



Q24 - Compléter les 3 zones manquantes

```
def Kvoisins(distances :dict ,K:int)->list :
    voisins = [(float("inf"), "") for k in range(K)]
    for lettre in distances:
        d = distances[lettre]
        for j in range( ..... ):
            if ..... :
                k = len(voisins)-1
                while ..... :
                    voisins[k] = voisins[k-1]
                    k = k - 1
                voisins[k] = [d[j], lettre]
    return voisins
```

Q25 - Complexité en fonction de n et K . Comparaison



Q26 - Fonction symbole_majoritaire(voisins:list) ->str



Q27 - Commentaires

FIN

SESSION 2023

TSI5IN



ÉPREUVE SPÉCIFIQUE - FILIÈRE TSI

INFORMATIQUE

Durée : 3 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.
- Ne pas utiliser de correcteur.
- Écrire le mot FIN à la fin de votre composition.

Les calculatrices sont interdites.

Le sujet est composé de cinq parties, pouvant être traitées indépendamment.

Important : vous pouvez utiliser les fonctions des questions précédentes, même si vous ne les avez pas toutes implémentées.

Vous devez répondre directement sur le Document Réponse, soit à l'emplacement prévu pour la réponse lorsque celle-ci implique une rédaction, soit en complétant les différents programmes en langage Python.

Le sujet comporte :

- le texte du sujet : 10 pages ;
- les Annexes : 3 pages ;
- le Document Réponse (**DR**) : 8 pages.

Seul le Document Réponse est à rendre dans son intégralité.

Gestion de Tests dans une entreprise

Une entreprise d'e-commerce vend des meubles tous identifiés par une référence et par un QR code¹. Tous les clients sont identifiés par leur numéro de sécurité sociale. Tous les achats s'effectuent à l'aide d'un numéro de carte de crédit. Cette entreprise met en œuvre différents tests afin d'éviter les erreurs de numéros de sécurité sociale, de numéros de carte de crédit ou de QR codes.

Partie I - Tests de code de sécurité sociale

En France, le numéro de sécurité sociale correspond au numéro d'inscription au répertoire national d'identification des personnes physiques (RNIPP). Il est formé du numéro d'inscription (NIR) à 13 chiffres et d'une clé de contrôle à 2 chiffres. Le NIR, créé à partir de l'état civil, est composé de la façon suivante :

- Sexe (1^{er} chiffre) ;
- Année de naissance (les deux chiffres suivants) ;
- Mois de naissance (les deux chiffres suivants) ;
- Lieu de naissance (les cinq chiffres ou caractères suivants - 2 chiffres² du code du département de naissance, suivis de 3 chiffres du code commune officiel de l'Insee³) ;
- Numéro d'ordre permettant de distinguer les personnes nées au même lieu à la même période (les 3 chiffres suivants).

Les deux derniers chiffres, compris entre 01 et 97, permettent de déterminer la clé, appelée aussi "clé de contrôle", qui permettra de contrôler l'exactitude du numéro de sécurité sociale.

Pour obtenir cette clé, on détermine tout d'abord, le reste de la division par 97 du nombre formé par les 13 premiers chiffres. La clé correspond au résultat de ce nombre retranché de 97.

Exemple : soit le numéro de sécurité sociale à 13 chiffres : "2 91 01 75 018 002". Le reste de la division de 2910175018002 par 97 est égal à 29. La clé est constituée du résultat : 97-29 = 68. Le numéro de sécurité sociale complet est donc : "2 91 01 75 018 002 68".

Dans cette partie, le numéro de sécurité sociale de 13 chiffres est une chaîne de caractères composée uniquement de chiffres avec des espaces de séparation entre les différents éléments constituant ce numéro. On ne prendra pas en compte le cas de la Corse.

Ne pas oublier qu'il est toujours possible de transformer un nombre entier en une chaîne de caractères composées de chiffres (fonction *str*) et réciproquement (fonction *int*), (**annexe 3**).

- Q1.** Écrire la fonction *num_secu* qui, à partir de la chaîne de caractères d'un numéro de sécurité sociale, donne le numéro sous la forme d'un entier. Le programme devra parcourir la chaîne de caractères représentant le numéro de sécurité sociale en supprimant les caractères d'espacement, puis la transformer en un nombre entier. Cette fonction a un paramètre de type *string* et retourne une valeur de type *int*.

1. Un QR code (Quick Response code) désigne un type de code-barres en deux dimensions, lequel se compose de modules noirs disposés dans un carré à fond blanc (voir **figure 1**).
2. Pour simplifier le problème, nous supposons que les deux départements corse 2A et 2B sont représentés par le code 20 comme avant 1976.
3. Institut national de la statistique et des études économiques.

Exemple :

```
>>> num_secu("2 91 01 75 018 002")
2910175018002
```

- Q2.** Écrire la fonction *clef* qui détermine la valeur de la clé d'un numéro de sécurité sociale. Cette fonction a un paramètre de type *int* et retourne un élément de type *int*.

Exemple :

```
>>> clef(2910175018002)
68
```

- Q3.** Écrire la fonction *num_secu_complet* qui détermine le numéro complet de sécurité sociale. Cette fonction a un paramètre de type *int* et retourne un élément de type *int*.

Exemple :

```
>>> num_secu_complet(2910175018002)
291017501800268
```

- Q4.** Écrire la fonction *test_num_secu* qui détermine si un numéro de sécurité sociale est correct. Cette fonction a un paramètre de type *string* et retourne un élément de type *bool*.

Exemples :

```
>>> test_num_secu('2 91 01 75 018 002 68')
True
>>> test_num_secu('2 91 01 75 018 002 93')
False
```

Partie II - Test de numéro de carte de crédit

Pour savoir si un numéro de carte de crédit est valide, on utilise très souvent l'algorithme de Luhn⁴. Comme pour le numéro de sécurité sociale, il y a une clé appelée somme de contrôle (checksum en anglais) qui fait partie du numéro d'une carte de crédit. Ce numéro est un entier composé de 16 chiffres. Le dernier chiffre est la clé qui permet de contrôler l'exactitude du numéro.

Le principe de l'algorithme de Luhn est le suivant. On commence toujours par le chiffre se trouvant le plus à droite. Ce chiffre sera le premier élément de la liste dites des "indices impairs". Puis on complète cette liste en prenant un chiffre sur deux du numéro de carte bancaire, toujours en le lisant de la droite vers la gauche.

Pour la liste des chiffres "d'indices pairs", on commence par le deuxième chiffre le plus à droite du numéro de la carte de crédit, on se déplace de la droite vers la gauche comme pour la liste précédente et on construit la liste, en prenant un chiffre sur deux. Pour les nombres de cette liste des indices pairs, on double tous les chiffres. Si un nombre est supérieur à 9, on réalise la somme des deux chiffres qui le composent (exemple si on obtient 16, on additionne 1 et 6 pour avoir 7). Par conséquent, tous les nombres des deux listes sont

4. L'algorithme de Luhn, ou code de Luhn, ou encore formule de Luhn est aussi connu comme l'algorithme "modulo 10".

composés uniquement de chiffres compris entre 0 et 9. On calcule alors la somme totale des chiffres de ces deux listes. Si cette somme est un multiple de 10, alors le numéro de la carte de crédit est valide.

Exemple : soit 4762 un nombre (on se limite à 4 chiffres mais le raisonnement est identique pour un nombre à 16 chiffres). Appliquons-lui la formule de Luhn. On commence par le chiffre 2, celui se trouvant le plus à droite. Le nombre 4762 se transforme en deux listes correspondant aux indices impairs et pairs soit [2, 7] et [6, 4]. Puis en deux autres listes, la liste des indices impairs inchangés [2, 7] et la liste des indices pairs doublés [12, 8]. La somme des éléments de ces deux listes est égale à 20 car la liste des indices pairs [12, 8] se réduit en la liste [3, 8] du fait de la sommation des chiffres constituant le nombre 12. La somme des éléments des deux listes obtenues [2, 7] et [3, 8] est bien égale à 20. Ce résultat est un multiple de 10, le nombre 4762 est donc correct au sens de l'algorithme de Luhn.

On aurait pu raisonner sur une seule liste et obtenir le même résultat. 4762 se transforme en la liste [8, 7, 12, 2] puis en la liste [8, 7, 3, 2] après réduction. La somme des chiffres 8+7+3+2 est égale à 20.

Q5. Écrire une fonction *num_en_liste* qui transforme un nombre entier en une liste de chiffres. Cette fonction a un paramètre de type *int* et retourne un élément de type *list*.

Exemple :

```
num_en_liste(4532015112830465)
[4, 5, 3, 2, 0, 1, 5, 1, 1, 2, 8, 3, 0, 4, 6, 5]
```

Q6. Écrire une fonction *tuple_pairs_impairs* qui détermine un tuple représentant la liste des chiffres d'indice pair et la liste des chiffres d'indice impair d'un numéro de carte de crédit. Le chiffre le plus à droite de ce numéro est considéré comme le premier chiffre d'indice impair. Cette fonction a un paramètre de type *int* et retourne un *tuple* composé de deux éléments de type *list*.

Exemple :

```
tuple_pairs_impairs(4532015112830465)
([6, 0, 8, 1, 5, 0, 3, 4], [5, 4, 3, 2, 1, 1, 2, 5])
```

Q7. Écrire une fonction *cree_dico* qui, à partir d'un numéro de carte de crédit, crée un dictionnaire avec deux clés nommées '*pair*' et '*impair*'. La clé '*pair*' est constituée de la liste des nombres d'indice pairs du numéro de la carte de crédit et la clé '*impair*' de la liste des nombres d'indice impairs.

Exemple :

```
>>> cree_dico(4532015112830465)
{'pair': [6, 0, 8, 1, 5, 0, 3, 4], 'impair': [5, 4, 3, 2, 1, 1, 2, 5]}
```

Q8. Écrire une fonction *traitement_nb_pairs* qui multiplie par 2 tous les chiffres de la liste associée à la clé '*pair*'. Si un chiffre est supérieur à 9, il faut réaliser la somme des deux chiffres qui le composent. Cette fonction a un paramètre de type dictionnaire et retourne un dictionnaire.

Remarque : la partie correspondant à la clé '*impair*' n'est pas modifiée par le traitement de cette fonction.

Exemple :

```
>>> un_dico=cree_dico(4532015112830465)
>>> traitement_nb_pairs(un_dico)
{'pair': [3, 0, 7, 2, 1, 0, 6, 8], 'impair': [5, 4, 3, 2, 1, 1, 2, 5]}
```

- Q9.** Écrire une fonction *test_num_carte_credit* qui utilise l'algorithme de *Luhn* pour savoir si un numéro de carte de crédit est correct. Vous devez utiliser la fonction *traitement_nb_pair* pour sa réalisation. Cette fonction a un paramètre de type *int* et retourne une valeur de type *bool*.

Exemple :

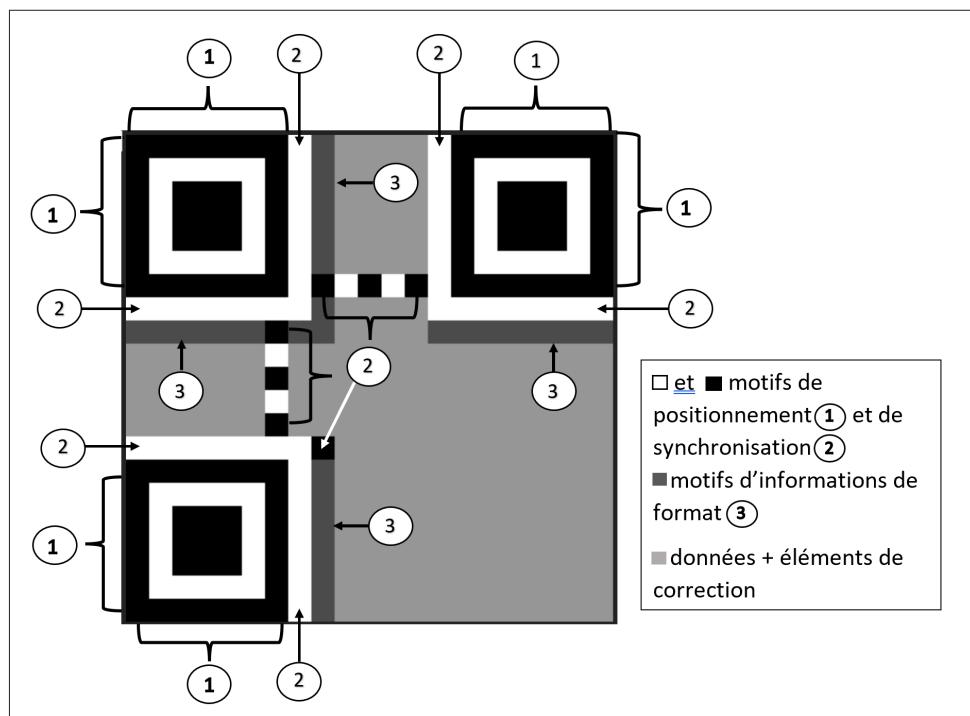
```
>>> test_num_carte_credit(4532015112830465)
True
```

Partie III - Tests de QR code

Les QR codes ont été inventés en 1994, par Masahiro Hara, un ingénieur de l'entreprise japonaise Denso-Wave. Cette invention a permis d'assurer le référencement des pièces détachées dans les usines Toyota. Les QR codes sont constitués essentiellement de pixels noirs et blancs codés dans le format RGB (**annexe 1**). Cependant, il existe des QR codes bicolores mais avec un jeu de couleurs très contrastées. Les QR codes peuvent être partiellement raturés ou déchirés car un de leurs avantages est qu'ils peuvent accepter un certain taux d'erreurs, entre 7 % et 30 % suivant la version du QR code. Il existe 40 versions qui peuvent stocker entre 10 et 7089 caractères numériques. Nous nous restreignons ici à la version 1 qui utilise une matrice de 21*21 pixels pour sa représentation.

En fait sur la **figure 1**, l'image du QR code correspond à une matrice de 420*420 pixels (**programme P1**), alors que la matrice initiale d'un QR code de version 1 ne compte que 21*21 pixels. Pour qu'un QR code soit plus visible, on a créé la notion de module qui correspond à un bloc de pixels identiques pour représenter un pixel du QR code initial. C'est un mécanisme de zoom pour que le QR code soit visible. Un module a une taille de 20*20 pixels. Chaque module représente globalement une valeur binaire : 1 pour le blanc et 0 pour le noir. Attention : ne pas confondre la taille d'un QR code (ici, 420*420) avec la taille d'un module (ici, 20*20) [la valeur 420 correspond à 21*20]. Enfin, un QR code est constitué de différents éléments : des motifs de positionnement (3 blocs de 7×7 pixels), des motifs de synchronisation (6 zones blanches de séparation et 11 pixels de couleur blanc et noir), des motifs de format d'information et une zone comprenant les données utilisateurs avec des motifs de correction (**figure 2**).

Dans la suite de cette partie, nous n'utiliserons que les QR codes de version 1.

**Figure 1** - QR code version 1**Figure 2** - Organisation d'un QR code

Q10. Écrire une fonction *init* qui réalise l'initialisation d'une liste de dimension *n* où chaque élément est également une liste de dimension *n*. Cette liste de listes représente ainsi une matrice de taille *n* × *n*. Cette fonction a un paramètre de type *int* et retourne une liste de listes qui représente un QR code initialisé avec des valeurs 0.

Exemple :

```
>>> init(4)
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

On donne le programme **P1** suivant : ([annexe 1](#) pour la description du module *Gestion_QRCode*).

P1

```

1  from Gestion_QRCode import *
2
3  img=open("./Image/ccinp.png").....# Lecture de l'image
4  img.show().....# Affichage de l'image (figure 1)
5  largeur,hauteur=img.size.....# La taille de l'image (largeur, hauteur)
6  position = (largeur,hauteur) .....# Résultat : (420, 420)
```

Q11. Écrire la fonction *charge_valeur* qui a pour but de réduire les données de l'image dans une liste de listes de dimension 21*21. Attention : l'image correspondant à un QR code représente une liste de listes de dimension 420*420 dont on veut réduire tous les blocs constitués de 20*20 pixels à un seul pixel pour avoir à partir de l'image une liste de listes de dimension 21*21. Ne pas oublier que tous les pixels d'un bloc sont identiques. Cette fonction a un paramètre de type image et retourne une liste de listes de triplets (couleur des pixels).

Indication : utiliser la fonction *getpixel* du module Python *Gestion_QRCode* (voir sa définition dans l'[annexe 1](#)).

On prend comme bloc de positionnement celui représenté dans la **figure 3**.



Figure 3 - Bloc de positionnement

Q12. Écrire une fonction *cree_bloc* qui crée un bloc de positionnement. Cette fonction, qui n'a pas de paramètre, retourne une liste de listes de dimension 7*7.

Exemple :

```

>>> cree_bloc()
[[0, 0, 0, 0, 0, 0, 0],
 [0, 1, 1, 1, 1, 1, 0],
 [0, 1, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0],
 [0, 1, 1, 1, 1, 1, 0],
 [0, 0, 0, 0, 0, 0, 0]]
```

Q13. Écrire une fonction *test_bloc* qui teste si un bloc de positionnement (rappel : il y en a trois) est bien représenté pixel par pixel dans un QR code. Cette fonction a 3 paramètres : les coordonnées *x* et *y* donnant la position du début d'un bloc de positionnement d'un QR code (toujours les coordonnées du pixel le plus haut et à gauche) et la liste de listes de dimension 21*21 associée au même QR code. Cette fonction retourne un booléen.

Remarque : on cherche à tester si un bloc de positionnement d'un QR code n'a pas subi une modification. Les coordonnées du pixel le plus haut et à gauche pour le premier bloc sont égales à (0,0), pour le second bloc à (0,14) et pour le troisième bloc à (14,0).

Exemples :

```
>>> test_bloc(0,0, mat1)
True
>>> test_bloc(1,3, mat1)
False
```

Q14. On considère qu'un QR code est bien positionné lorsque ses 3 blocs de contrôle sont effectivement présents en haut à gauche, en haut à droite et en bas à gauche (comme sur la **figure 1**). Écrire une fonction *test_QRcode* qui permet de tester si un QR code est bien positionné. Cette fonction a pour paramètre une matrice de dimension 21*21 et retourne un booléen.

Exemple :

```
>>> test_QRcode(mat1)
True
```

Lors de la lecture d'un QR code par un appareil dédié (scanner, caméra ou autre) le processus de lecture permet de placer un QR code dans l'une des quatre positions possibles, comme illustré dans la **figure 4**. Cela dépend bien évidemment de l'orientation du QR code lors de sa lecture.

On se propose de faire tourner un QR Code par rotation successive de 90° afin qu'il puisse se trouver dans la bonne position comme celui de la **figure 1**.

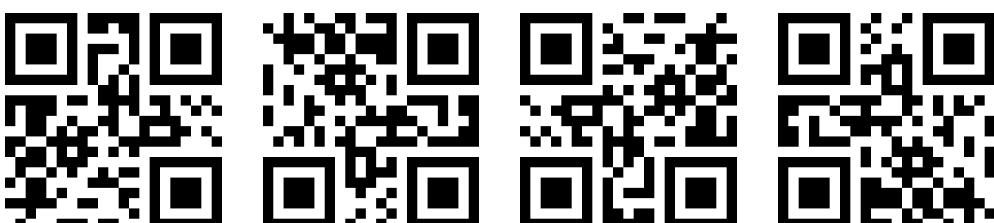


Figure 4 - Les 4 positions possibles lors de la lecture d'un QR code

Q15.

Écrire une procédure⁵ *tourHoraire* qui réalise une rotation de 90°, dans le sens des aiguilles d'une montre, des 4 éléments du QR code. La fonction a trois paramètres, les coordonnées *x* et *y* d'un élément de la liste de listes et une liste de listes de dimension 21*21.

5. Une procédure est une fonction qui retourne la valeur *None* mais cette valeur n'est pas destinée à être utilisée ou à être capturée.

Exemple :

```
>>> tourHoraire(0,1, mat1)
```

On se limite à un exemple d'une liste de listes de dimension 4*4 pour expliquer le fonctionnement, mais ce serait la même chose pour une liste de listes de dimension 21*21. Si on prend les 4 éléments (b , h , o , i) de la liste de listes **table 1** de la **figure 5**, b doit se trouver, après une rotation de 90°, à la place de l'élément h , l'élément h à la place de l'élément o , l'élément o à la place de l'élément i et l'élément i à la place de l'élément b . Le résultat de la transformation est illustré dans la **table 2** de la **figure 5**. Le mécanisme doit s'exécuter de la même manière sur les autres éléments de la **table 2**. Le résultat de la transformation finale est illustré dans la **table 3** de la **figure 5**.

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

Table 1

a	i	c	d
e	f	g	b
o	j	k	l
m	n	h	p

Table 2

m	i	e	a
n	j	f	b
o	k	g	c
p	l	h	d

Table 3

Figure 5 - Simple rotation

Q16. Écrire la procédure *rotationHoraire* qui réalise la rotation de 90° d'un QR code. Cette procédure a un seul paramètre, une liste de listes de dimension 21*21.

Par exemple, dans la **figure 4** cette fonction réalisera la première rotation de 90° du QR code.

Q17. Connaissant les 4 positions possibles lors de la lecture d'un QR code par un appareil dédié, écrire la procédure *QRcode_posi* qui positionne correctement un QR code. Cette procédure a un seul paramètre, une liste de listes de dimension 21*21.

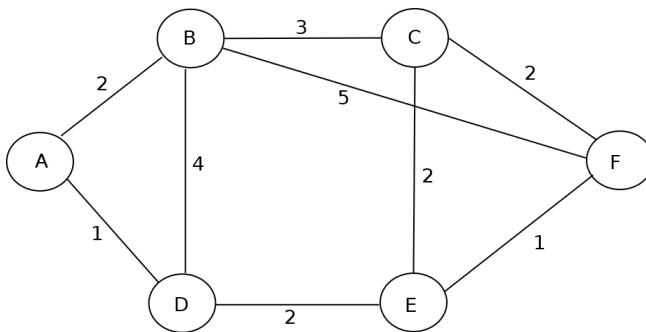
Indication : utiliser les fonctions *rotationHoraire* et *test_QRcode* .

Partie IV - Gestion réseau

Cette entreprise possède de nombreux magasins dans le monde entier. Des serveurs ont été placés dans tous les pays et sont nommés par des lettres.

Les différentes informations envoyées dans le réseau circulent de serveur en serveur. Les serveurs sont représentés par des nœuds, **figure 6**, et plusieurs routes sont possibles entre chacun d'eux. Le poids associé aux arêtes correspond à la valeur du temps de transmission entre deux nœuds du graphe multiplié par un facteur correctif. L'entreprise souhaite optimiser les temps de transmission entre deux nœuds du réseau en utilisant l'algorithme de Dijkstra.

L'algorithme de Dijkstra permet de déterminer les plus courts chemins à partir d'un sommet unique $s \in S$ vers les autres sommets d'un graphe pondéré orienté ou non $G = (S, A)$, avec S un ensemble de sommets et A un ensemble d'arêtes qui sont des paires de sommets. Toutes les arêtes de G sont de poids positif.

**Figure 6** - Organisation des serveurs**Principe de l'algorithme de Dijkstra**

Entrée :

 $G(S, A)$: un graphe pondéré, d : le sommet de départ à partir duquel on veut déterminer les plus courts chemins aux autres sommets, P : construction d'un sous-graphe tel que la distance entre un sommet de P depuis d soit définie et soit un minimum dans le graphe G , $Parent$: tableau pour noter les sommets par où on passe. Parent est utilisé comme le tableau des précédents de chaque sommet, initialisé avec un élément n'appartenant pas à S , M : tableau où les indices représentent les sommets du graphe : 0 désigne le sommet "A", 1 désigne le sommet "B", 2 désigne le sommet "C", etc...

Les éléments de ce tableau sont corrigés au fur et à mesure de l'algorithme afin d'obtenir les distances les plus courtes du sommet de départ à un sommet du graphe.

Début :

 $P \leftarrow \emptyset$ $M[d] \leftarrow 0$ // la distance de d à lui-même est égale à 0 $M[s] \leftarrow +\infty$ pour chacun des sommets du graphe autre que d Tant qu'il existe un sommet qui ne soit pas dans P Choisir un sommet s hors de P de plus petite distance $M[s]$ Ajouter s à P Pour chaque sommet u hors de P mais voisin de s si $M[u] > M[s] + \text{poids}(s,u)$ $M[u] = M[s] + \text{poids}(s,u)$ $\text{Parent}[u] = s$ // le sommet s est le prédecesseur du sommet u

Fin du pour

Fin tant que

Fin

En utilisant l'algorithme de Dijkstra, on souhaite déterminer tous les plus courts chemins, en terme de temps de communication, en partant du sommet A du graphe de la **figure 6**.

Il est plus simple de réaliser l'exécution de l'algorithme de Dijkstra avec un tableau particulier que l'on nomme $M+$. Ce tableau est dans le **DR**. Une première colonne précise l'évolution des distances d'un sommet spécifique depuis le sommet de départ lors de l'exécution de l'algorithme. Chaque ligne correspond à une étape de l'algorithme. Chaque ligne donne les valeurs des distances courantes des sommets depuis le sommet de départ avec éventuellement une mise à jour si une distance est plus petite que celle déjà calculée.

Dans le tableau $M+$, l'élément $B(2_A)$ correspond à la colonne choisie qui est B , à la valeur 2 du chemin et au sommet précédent A . Sur la ligne de l'élément $B(2_A)$ on choisit la valeur ③ de la colonne E pour l'étape suivante.

Q18. Compléter les 3 lignes manquantes du tableau $M+$ sur le **DR**.

Q19. Donner la valeur du plus court chemin entre "A" et "F". Expliquer comment on obtient cette valeur à l'aide du tableau $M+$. Expliciter le chemin le plus court trouvé pour aller de "A" à "F".

Partie V - Requêtes SQL

L'entreprise possède une base de données nommée *Gestion_Entreprise* constituée de trois tables : clients, produits et ventes. Les contenus de ces tables se trouvent en **annexe 2**.

La **table "clients"** est constituée de 5 champs :

- *id* : de type INTEGER – clé primaire auto-incrémente ;
- *num_secu* : de type INTEGER – entier de 15 chiffres ;
- *nom* : de type TEXT ;
- *prenom* : de type TEXT ;
- *num_CB* : de type INTEGER.

La **table "produits"** est constituée de 5 champs :

- *id* : de type INTEGER – clé primaire auto-incrémente ;
- *ref_produit* : de type INTEGER ;
- *nom_produit* : de type TEXT ;
- *qrcode* : de type TEXT ;
- *prix* : de type DECIMAL.

La **table "ventes"** est constituée de 3 champs :

- *date* : de type TEXT ;
- *ref_produit* : de type INTEGER ; – clé étrangère, pointe vers la clé primaire *id* de la table *produits* ;
- *num_client* : de type INTEGER ; – clé étrangère, pointe vers la clé primaire *id* de la table *clients*.

Les dates dans cette table sont définies par une chaîne de 10 caractères suivant le format *année-mois-jour*. Exemple de dates : "2019-06-01", "2022-12-30".

Q20. Écrire, en SQL, la requête (1) qui permet d'obtenir le numéro de carte de crédit de toutes les personnes référencées dans la base de données de l'entreprise dont le numéro de sécurité sociale commence par 2. On utilisera le caractère "_" comme le séparateur des milliers. Par exemple 10000000 sera réécrit comme 10_000_000.

Q21. Écrire, en SQL, la requête (2) permettant d'obtenir le nom et le prénom de toutes les personnes ayant effectué un achat avec un résultat sans doublon.

Q22. Écrire, en SQL, la requête (3) qui permet d'obtenir les produits associés à chaque numéro de carte de crédit du client et qui ont été vendus entre le 1 juin 2020 et le 30 juillet 2020. On rappelle que SQL compare les variables de type TEXT grâce à l'ordre lexicographique : par exemple "13-06-1989 < 13-07-1999" est vrai.

Annexe 1

Module "Gestion_QRCode"

fonction Gestion_QRCode.open(fp)

- * Paramètre :
 - fp : nom de fichier (chaîne de caractères) représentant une image sous différents formats tels que PPM, PNG, JPEG, GIF, TIFF et BMP.
- * Retour :
 - retourne une variable qui est un descripteur d'image (un objet image).

Attention : la valeur retournée n'est en aucun cas une liste de listes ou une liste de listes similaire à celle de la bibliothèque *Numpy*.

Exemple :

```
img=Gestion_QRCode.open("uneImage.png")
```

fonction Gestion_QRCode.Show()

- * Retour :
 - retourne la valeur None.⁶

Cette fonction affiche l'image dans n'importe quelle visionneuse d'images.

Exemple :

```
img.show()
```

attribut Gestion_QRCode.size

Permet de connaître la taille de l'image en pixels, le résultat est un tuple (largeur, hauteur).

Exemple :

```
print(img.size)
>>> (360, 160)      # soit largeur = 360 pixels et hauteur = 160 pixels
```

fonction Gestion_QRCode.getpixel(x,y)

- * Paramètres :
 - x : la coordonnée x du pixel référencé ;
 - y : la coordonnée y du pixel référencé.
- * Retour :
 - retourne les attributs de la couleur du pixel, au format RVB⁷

La valeur retournée est un tuple (r,v,b) correspondant à la couleur du pixel : le plus souvent (0,0,0) pour un pixel noir et (255,255,255) pour un pixel blanc pour un QR code.

Exemple :

```
(r,v,b)= img.getpixel(100,30)
>>> (0, 0, 0)      # pour un pixel noir
```

6. La valeur *None* est une valeur qui correspond à l'absence de valeur.

7. Le système RVB (Rouge, Vert, Bleu), ou en anglais RGB (Red, Green, Blue), permet de coder les couleurs en informatique. Un écran informatique est composé de pixels représentant une couleur au format RVB. La composante R est codée sur 8 bits de 0 à 255 en décimal. Il en va de même pour les composantes suivantes. Le codage des couleurs va du plus foncé au plus clair.

Annexe 2

Base de données "Gestion_Entreprise"

Le contenu des tables *clients*, *produits* et *ventes* de la base de données *Gestion_Entreprise* est donné ci-après.

Table "clients"

id	num_secu	nom	prenom	num_CB
1	286128817863441	Eldyn	Sophie	6767342589219928
2	298082934500890	Gomez	Maria	2324563490665454
3	298082934500896	Ruiza	Flor	9889454573204522
4	109086723487917	Kovitz	Boris	6789543778653678
5	175105642102321	Mottreff	Erwan	4745342178563217
6	189027511732543	Settin	Michel	7856432167453492
7	191017511318196	Valérie	Georges	8787564521392354
...

Table "produits"

id	ref_produit	nom_produit	qrcode	prix
1	27	Buffet chêne	27.jpg	320
2	102	Chaise rustique	102.jpg	65
3	453	Table ronde	453.jpg	75
4	756	Table ovale	756.jpg	120
5	921	Coffret Bali	921.jpg	170
...

Table "ventes"

date	ref_produit	num_client
2020-05-15	2	2
2020-06-17	3	1
2020-06-21	3	7
2020-07-19	4	5
2020-08-19	5	5
2020-09-05	4	6
...

Annexe 3

Rappels des syntaxes en Python

Définir une liste.	<code>L = [1, 2, 3] >>> L[0] 1</code>
Définir une liste de listes.	<code>LL = [[1, 2], [3, 4], [5, 6], [7, 8]] >>> LL[1] [3, 4]</code>
Ajouter un élément à la fin d'une liste.	<code>L.append(5) LL.append([9, 10])</code>
Convertir un nombre entier en une chaîne de caractères.	<code>>>> str(12345) '12345'</code>
Convertir une chaîne de caractères en un nombre entier. Attention : si la chaîne de caractères contient un espace, l'appel de la fonction <code>int</code> provoque une erreur.	<code>>>> int('12345') 12345</code>
$a//b$ donne le quotient de la division euclidienne de a par b .	<code>>>> 10//3 3</code>
$a\%b$ donne le reste de la division euclidienne de a par b .	<code>>>> 10%3 1</code>
Définir une chaîne de caractères.	<code>mot='Python'</code>
Longueur d'une chaîne.	<code>len(mot)</code>
Le slicing permet d'extraire des éléments d'une liste ou d'une chaîne.	<code>si L = [6, 8, 5, 3, 7] L[2 : 4] --> [5, 3] L[: 3] --> [6, 8, 5] L[3 :] --> [3, 7] L[::-2] --> [6, 5, 7] L[2 : -1] --> [5, 3] L[-1] --> 7 mot[2 : 7] -->'thon'</code>

FIN

	Numéro d'inscription		Nom :	
	Numéro de table		Prénom :	
	Né(e) le			
	Filière : TSI		Session : 2023	
	Épreuve de : INFORMATIQUE			
	Consignes <ul style="list-style-type: none"> • 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) • Numéroter chaque page (cadre en bas à droite) • Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre 			

TSI5IN

DOCUMENT RÉPONSE

Seul ce document est à rendre dans son intégralité.

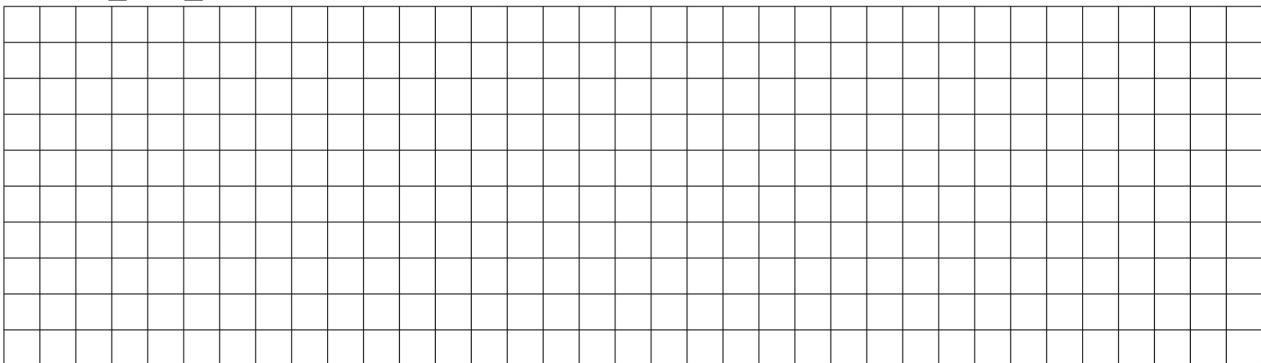
Q1. Fonction num_secu

Q2. Fonction clef

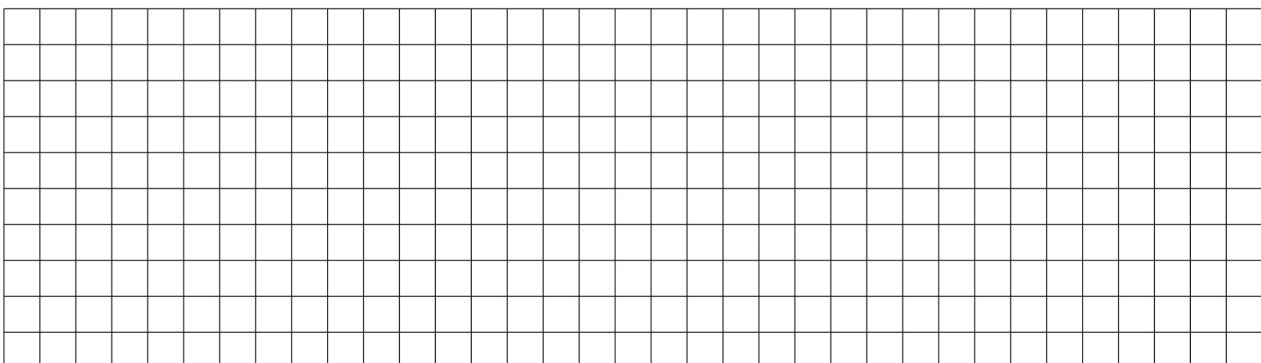
Q3. Fonction num_secu_complet

NE RIEN ÉCRIRE DANS CE CADRE

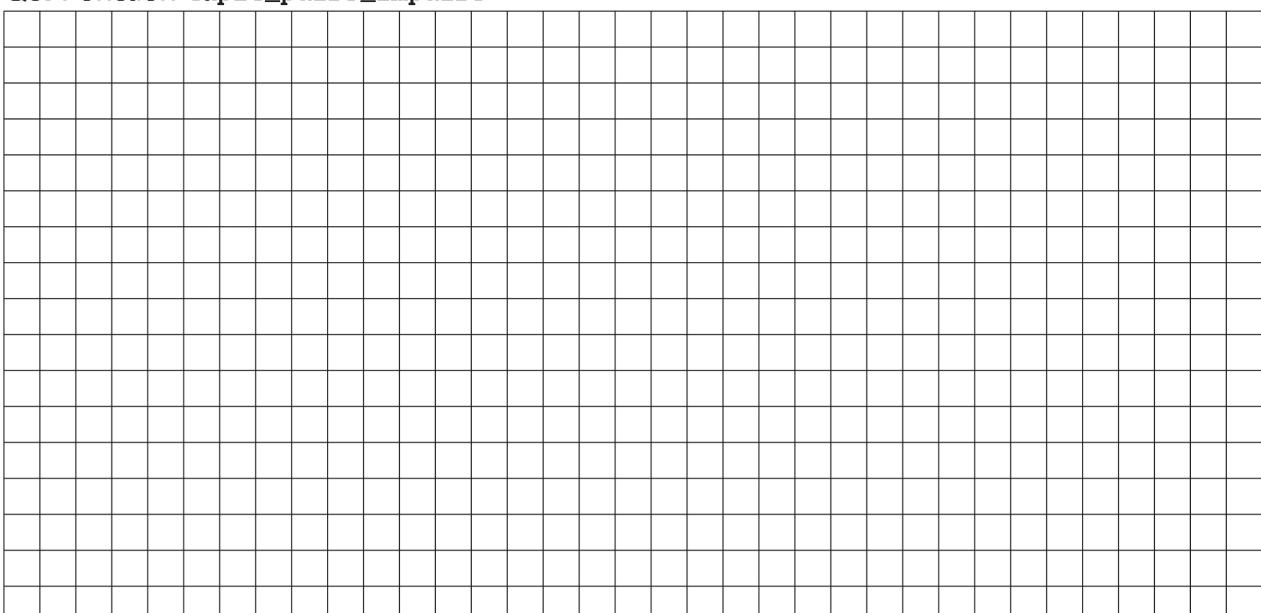
Q4. test_num_secu



Q5. Fonction num_en_liste



Q6. Fonction tuple_pairs_impairs



Q7. Fonction cree_dico

Q8. Fonction traitement_nb_pairs

Q9. Fonction test_num_carte_credit

Q10. Fonction init

Q11. Fonction charge_valeur

Q12. Fonction cree_bloc

Q13. Fonction test_bloc

	CONCOURS COMMUN	Numéro d'inscription	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Nom :	<input type="text"/>
		Numéro de table	<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	Prénom :	<input type="text"/>
		Né(e) le	<input type="text"/> <input type="text"/> / <input type="text"/> <input type="text"/> / <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>		
Emplacement QR Code	Filière : TSI				Session : 2023
	Épreuve de : INFORMATIQUE				
Consignes	<ul style="list-style-type: none"> • 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 				

TSI5IN

Q14. Fonction test_QRcode

Q15. Fonction tourHoraire

NE RIEN ÉCRIRE DANS CE CADRE

Q16. Fonction rotationHoraine

Q17. Fonction QRcode_posi

Q18. Compléter les 3 lignes manquantes du tableau $M+$

	A	B	C	D	E	F
étape initiale	0	∞	∞	∞	∞	∞
$A(0)$	-	2	∞	(1)	∞	∞
$D(1_A)$	-	(2)	∞	-	3	∞
$B(2_A)$	-	-	5	-	(3)	7
$E(3_D)$						

Q19. Donner et expliquer la valeur obtenue du plus court chemin entre "A" et "F".
Explicité le chemin le plus court entre "A" et "F".

Q20. Requête 1

Q21. Requête 2

Q22. Requête 3



Epreuve d’Informatique et Modélisation de Systèmes Physiques

Durée 4 h

Si, au cours de l’épreuve, un candidat repère ce qui lui semble être une erreur d’énoncé, d’une part il le signale au chef de salle, d’autre part il le signale sur sa copie et poursuit sa composition en indiquant les raisons des initiatives qu’il est amené à prendre.

L’usage de calculatrices est interdit.

AVERTISSEMENT

La présentation, la lisibilité, l’orthographe, la qualité de la rédaction, la clarté et la précision des raisonnements entreront pour une **part importante** dans l’appréciation des copies. En particulier, les résultats non justifiés ne seront pas pris en compte. Les candidats sont invités à encadrer les résultats de leurs calculs.

Un document en annexe se trouve en page 16 du sujet.

CONSIGNES :

- Composer lisiblement sur les copies avec un stylo à bille à encre foncée : bleue ou noire.
- L’usage de stylo à friction, stylo plume, stylo feutre, liquide de correction et dérouleur de ruban correcteur est interdit.
- Remplir sur chaque copie en MAJUSCULES toutes vos informations d’identification : nom, prénom, numéro inscription, date de naissance, le libellé du concours, le libellé de l’épreuve et la session.
- Une feuille, dont l’entête n’a pas été intégralement renseigné, ne sera pas prise en compte.
- Il est interdit aux candidats de signer leur composition ou d’y mettre un signe quelconque pouvant indiquer sa provenance.

PARTIE MODÉLISATION

Notations :

- $\vec{A} \wedge \vec{B}$: produit vectoriel de \vec{A} avec \vec{B}
- $\vec{A} \cdot \vec{B}$: produit scalaire de \vec{A} avec \vec{B}
- $\frac{\partial A}{\partial x} \Big|_{x_0}$: dérivée partielle de A par rapport à x , évaluée en x_0

Lors de la réalisation d'un test PCR (Polymerase Chain Reaction), une portion d'ADN double brin est copiée (amplifiée) un très grand nombre de fois par une enzyme (la polymérase).

Toutes les séquences d'ADN double brin ne sont pas éligibles pour cette méthode. Les propriétés physiques de l'ADN dépendent de la séquence de paires de bases (nommées G,T,A ou C) et si la séquence que l'on souhaite copier donne lieu à un objet trop rigide le cheminement de la polymérase le long du morceau d'ADN risque de ne pas se dérouler correctement.

Ce sujet propose d'aborder les méthodes utilisées pour essayer de prédire ainsi que de mesurer une partie des propriétés mécaniques de l'ADN.

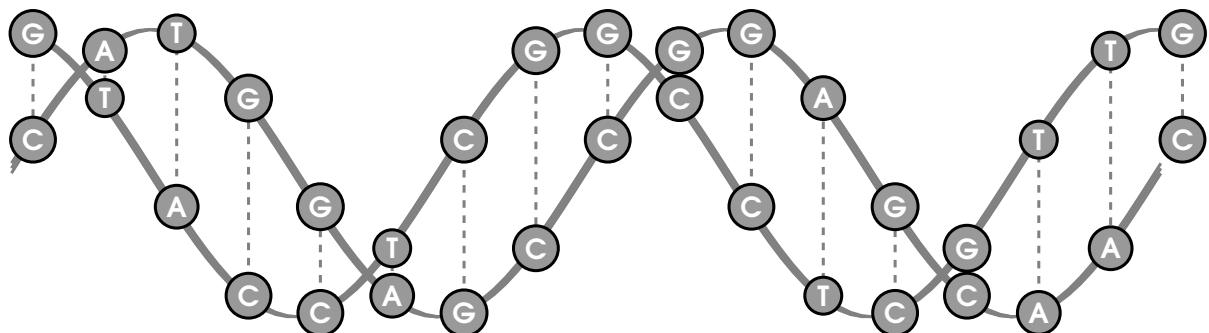


FIGURE 1 – Double hélice d'ADN. Chaque brin est constitué d'une séquence de paires de bases reliées par des liaisons covalentes (en gris). Les deux brins sont reliés par des liaisons hydrogène (en pointillés) qui sont moins résistantes que les liaisons covalentes.

PARTIE I : MODÈLE NAÏF DE L'ÉLASTICITÉ DE BRINS D'ADN COURTS (<30 PAIRES DE BASES)

On modélise d'abord un simple brin d'ADN par une succession de bases reliées par des ressorts.

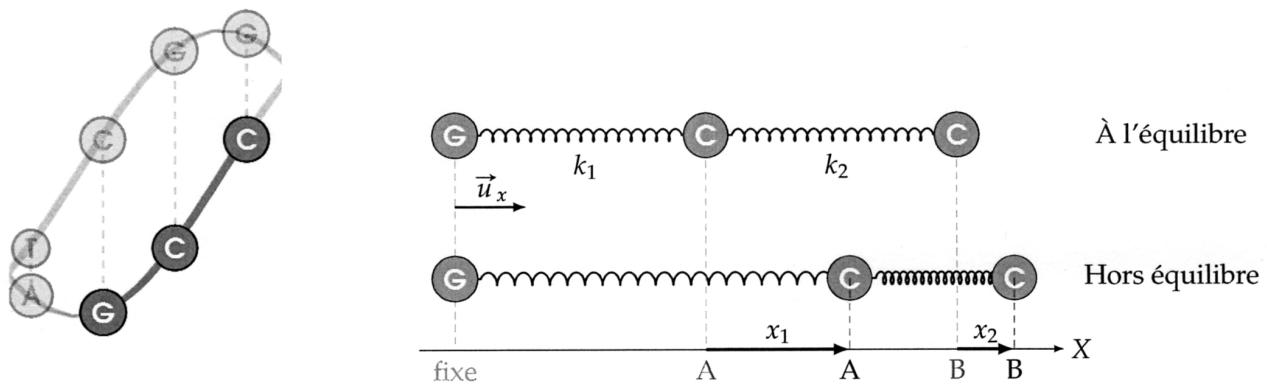


FIGURE 2 – Une portion d'un simple brin d'ADN (ici "G-C-C") est modélisée par une succession de deux ressorts.

On repère le déplacement de A par rapport à sa position d'équilibre par x_1 , et le déplacement de B par rapport à l'équilibre par x_2 . On pourra considérer que la longueur à vide de ces ressorts est la longueur qui sépare les points A et B lorsqu'ils sont à l'équilibre, on note cette longueur l_0 . La raideur d'une liaison covalente G-C est notée k_1 et celle d'une liaison covalente C-C est notée k_2 . Dans tout l'énoncé, les effets de la gravité sont négligés.

1. Les deux points (A et B) sont déplacés de leur position au repos. Faire l'inventaire des forces qui s'exercent sur le point A . Donner leur expression vectorielle. Faire de même pour le point B . Donner vos réponses en fonction de x_1 , x_2 et des paramètres de l'énoncé.

On tire sur le point B avec une force $\vec{F}_0 = F_0 \vec{u}_x$ jusqu'à ce que l'ensemble se retrouve dans une nouvelle position d'équilibre.

2. Montrer que tout se passe comme si le point B était attaché à l'origine via un unique ressort dont on donnera la raideur.

On suppose que les liaisons covalentes qui relient une paire de base (par exemple G-G, G-C, T-A, etc) ont toutes une raideur équivalente identique. Dans l'exemple de la figure 2, cela signifie que $k_1 = k_2 = k$.

3. Quelle est la raideur d'un brin de séquence : GCTGAGG ?

Un double brin d'ADN est en fait une succession de "paires de base". Chaque base est reliée à une base complémentaire (G avec C, T avec A) via des liaisons hydrogène :

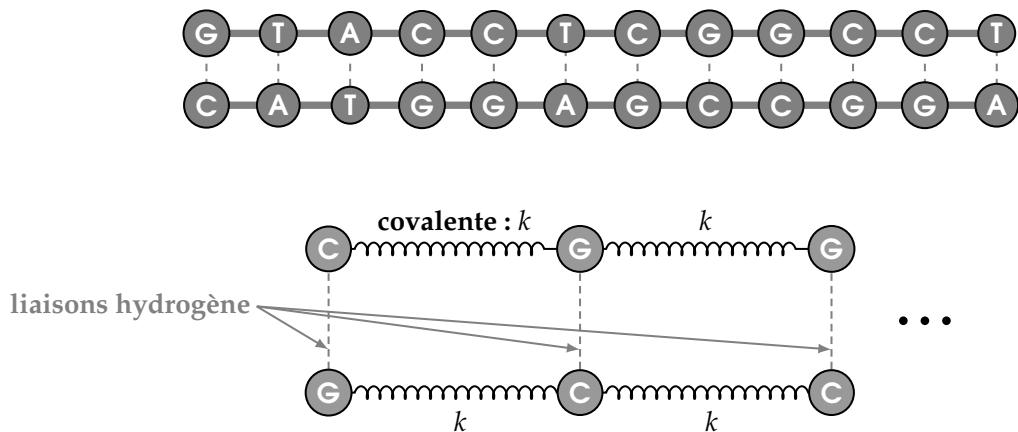


FIGURE 3 – Schématisation d'un morceau d'ADN double brin.

On néglige l'effet des liaisons hydrogène qui relient deux bases opposées. On tire sur une extrémité du double brin, l'autre restant attachée.

4. Quelle est la raideur d'un double brin constitué de 7 paires de bases, exprimée en fonction de k ?

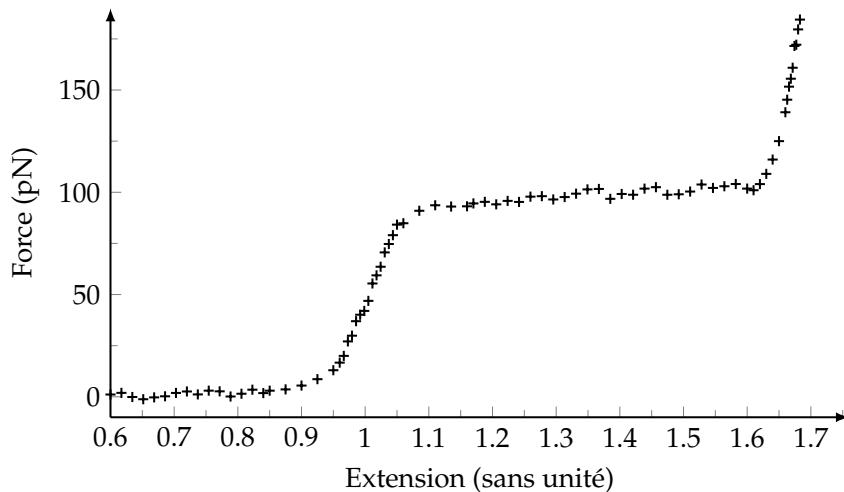


FIGURE 4 – Force de rappel (en pN) versus extension (allongement relatif) d'un morceau d'ADN double brin. [D'après Caron et al., "DNA : an extensible molecule"].

5. Sur quel domaine d'extension le modèle proposé ci-dessus semble-t-il valide ?
6. Sachant que la longueur à vide de séparation entre deux paires de bases est de l'ordre de 3 pm, donner un ordre de grandeur de k , la raideur d'une unique liaison covalente.

PARTIE II : ÉLASTICITÉ D'UN LONG BRIN D'ADN (> 200 PAIRES DE BASES)

Ce modèle devient inapproprié pour les longues molécules d'ADN. La flexibilité de celui-ci fait qu'on peut le considérer comme une chaîne de bâtonnets rigides, chaque bâtonnet ayant une centaine de paires

de bases de longueur environ.

Les forces dites "entropiques" dues aux collisions avec les molécules environnantes (typiquement, des molécules d'eau) font que, en pratique, on observe, pour un brin d'ADN dont les deux extrémités sont séparées d'une distance x , une énergie potentielle :

$$U(x) = U_0 + \frac{1}{2}k' \frac{x^2}{\langle r^2 \rangle}$$

ou $k' = 3 k_b T$ est une constante qui dépend uniquement de la température, et $\langle r^2 \rangle$ est décrit plus bas. Le brin d'ADN est modélisé comme une chaîne de petits fragments rigides de longueur $a \approx 30 \text{ nm}$.

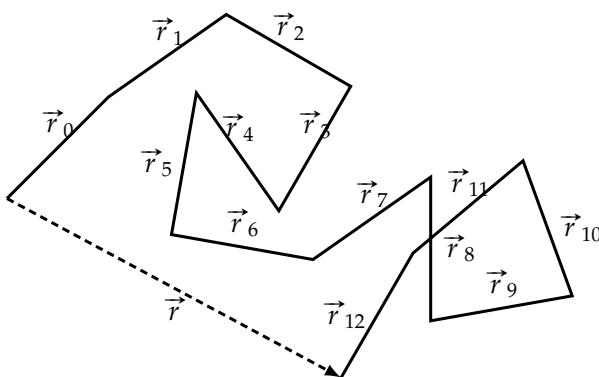


FIGURE 5 – Modélisation d'un long morceau d'ADN sous la forme d'une chaîne de bâtonnets.

On appelle \vec{r}_i le vecteur déplacement associé au fragment i . On considère la direction des bâtonnets comme indépendantes les unes des autres, c'est à dire que l'angle $\theta_{i,j}$ formé par les vecteurs \vec{r}_i et \vec{r}_j est aléatoire, réparti uniformément entre 0 et 2π , si $i \neq j$.

7. Quelle est la valeur moyenne $\langle \vec{r}_i \cdot \vec{r}_j \rangle$ du produit scalaire $\vec{r}_i \cdot \vec{r}_j$ lorsque $i \neq j$?

8. Quelle est, dans ce cas, la valeur moyenne $\langle \vec{r} \rangle$ de $\vec{r} = \sum_{i=1}^N \vec{r}_i$? Même question pour la valeur de moyenne $\langle r^2 \rangle$ de $r^2 = \vec{r} \cdot \vec{r}$.
9. En envisageant une déformation unidimensionnelle, en déduire l'expression de la force élastique exercée par un brin d'ADN de longueur totale $L = Na$, et dont les extrémités sont séparées de x . Donner votre réponse en fonction de k_b , T , a , x et L .

PARTIE III : MANIPULATION DES MOLÉCULES D'ADN

Pour mesurer ces forces, une méthode fréquemment utilisée est la pince optique. Cette méthode consiste à attacher le brin d'ADN à une petite bille de polystyrène, et la bille de polystyrène est elle-même piégée par un faisceau lumineux. L'objectif de cette partie est de comprendre comment un faisceau lumineux peut piéger la bille.

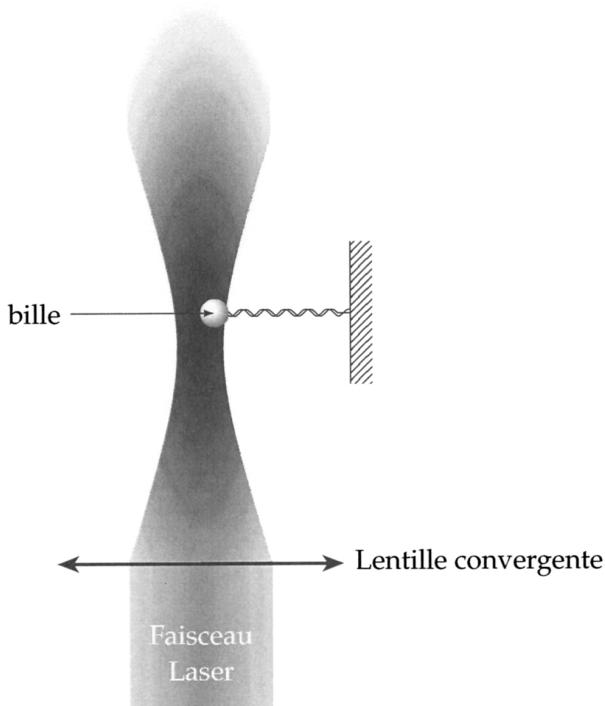


FIGURE 6 – Schéma expérimental d'une pince optique. Le niveau de gris indique l'intensité lumineuse du faisceau laser.

Lorsque la bille est illuminée par une lumière cohérente (celle d'un laser par exemple), un pôle chargé positivement apparaît sur la bille, ainsi qu'un pôle négatif du fait de la nature électromagnétique de la lumière. Du point de vue électrique, tout se passe comme si la bille était constituée de deux points ($M_1, -q$), et ($M_2, +q$). La neutralité de la bille garantie que les deux charges sont exactement opposées l'une de l'autre.

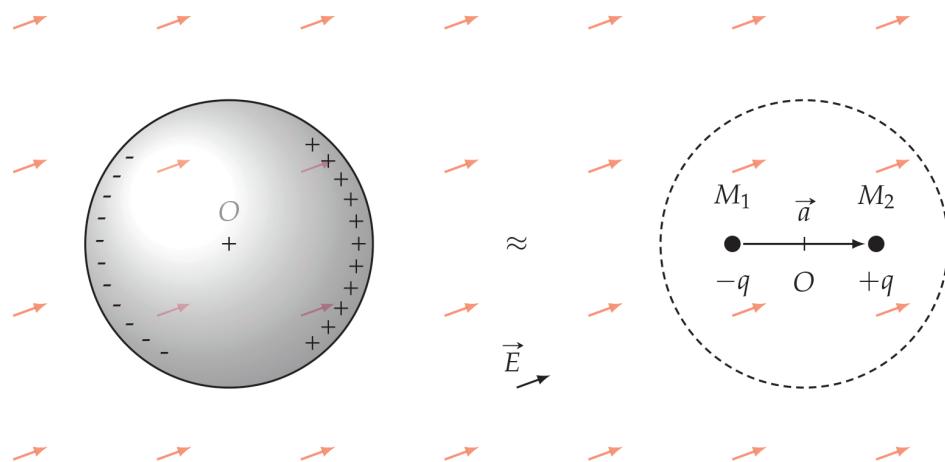


FIGURE 7 – Modélisation de la distribution de charge sur la bille.

On note O le centre de la bille, et $\vec{a} = \overrightarrow{M_1 M_2}$. Dans tout le problème, on néglige le champ électrique que génèrent les deux charges situées en M_1 et M_2 par rapport au champ électrique extérieur.

10. On soumet cette bille à un champ électrique extérieur \vec{E} uniforme. Quelle est la force électrique exercée sur la charge située en M_1 ? Quelle est la force électrique exercée sur la charge située en M_2 ? En déduire la force électrique totale \vec{F}_E exercée sur la bille ?

La charge située en M_1 est possiblement en mouvement à une vitesse \vec{v}_1 . De même pour M_2 qui se déplace à la vitesse \vec{v}_2 .

11. On soumet cette bille à un champ magnétique extérieur \vec{B} uniforme. Quelle est la force magnétique exercée sur la charge située en M_1 ? Quelle est la force magnétique exercée sur la charge située en M_2 ? En déduire que la force magnétique totale \vec{F}_B exercée sur la bille s'exprime sous la forme :

$$\vec{F}_B = \frac{d}{dt}(q\vec{a}) \wedge \vec{B}$$

On modélise le caractère local de l'éclairage lumineux de la pince optique par un champ électrique d'amplitude non-uniforme. Plus exactement, on suppose que le champ électrique prend la forme suivante :

$$\vec{E}(x, y, z, t) = E_0 e^{-\frac{x^2}{\sigma^2}} \cos(\omega t - kz) \vec{u}_y$$

où σ est une constante connue.

12. Quelle est la direction de propagation ainsi que la polarisation de cette onde de champ électrique ?

Lorsque le champ électromagnétique n'est pas trop intense, on peut montrer que les forces électriques et magnétiques exercées sur la bille se mettent sous la forme forme :

$$\vec{F}_{EM} = \vec{F}_1 + \vec{F}_2$$

avec

$$\begin{cases} \vec{F}_1 = \frac{1}{2}\alpha \operatorname{grad}(\vec{E}^2) \\ \vec{F}_2 = \alpha \frac{d}{dt} (\vec{E} \wedge \vec{B}) \end{cases}$$

où α est une constante caractéristique du matériau de la bille et de l'environnement.

Enfin, on rappelle la définition de moyenne temporelle $\langle A \rangle$ d'une fonction de période T :

$$\langle A \rangle = \frac{1}{T} \int_{t_0}^{t_0+T} A(t) dt$$

13. Donner l'expression de la moyenne temporelle de la force \vec{F}_1 exercée sur la bille.

14. Donner l'expression de la moyenne temporelle de la dérivée d'une fonction périodique : $\left\langle \frac{df}{dt} \right\rangle$. En déduire la valeur moyenne de la force \vec{F}_2 exercée sur la bille.

15. Calculer l'intensité lumineuse $I(x, y, z)$ associée à ce champ électromagnétique.

16. En déduire que la force électromagnétique moyenne peut se mettre sous la forme :

$$\vec{F}_{EM} = -\kappa x \vec{u}_x$$

où κ est proportionnel à l'intensité lumineuse.

17. Faire un schéma qui illustre l'orientation de la force électromagnétique moyenne exercée sur la bille, lorsqu'elle est dans un profil lumineux conforme à la figure 6.

PARTIE INFORMATIQUE

On pourra dans la suite du sujet utiliser les objets python de type liste ou les objets de type `numpy.ndarray` pour représenter une liste ou un tableau. Les listes entrées en argument seront non vides. Un document en annexe rappelle les opérations et fonctions usuelles des librairies `numpy` et `matplotlib`. On considère que ces librairies ont été importées avec le code :

```
import numpy as np
import matplotlib.pyplot as plt
```

PRINCIPE D'UNE MESURE D'ÉLASTICITÉ D'ADN

On s'intéresse dans cette partie du sujet à la mesure de l'effort qui s'exerce sur un brin d'ADN dont on souhaite déterminer les caractéristiques mécaniques. Le dispositif utilisé est un piège optique tel que décrit dans la première partie. La puissance du laser est imposée à une valeur constante. Le brin d'ADN est fixé à la bille par une de ses extrémités. L'autre extrémité est liée à un plateau mobile. La force agissant sur la bille est déduite par la mesure interférométrique de la position de la bille par rapport au centre du piège. Le capteur renvoie une tension U proportionnelle au déplacement et donc à la force exercée par le montage sur la bille. Pour déplacer finement l'échantillon (100 μm d'amplitude maximale), le plateau mobile est actionné par des cales piézo-électriques associées à une mesure capacitive de déplacement. Le déplacement est asservi, ce qui permet d'imposer une position avec une précision au nanomètre.

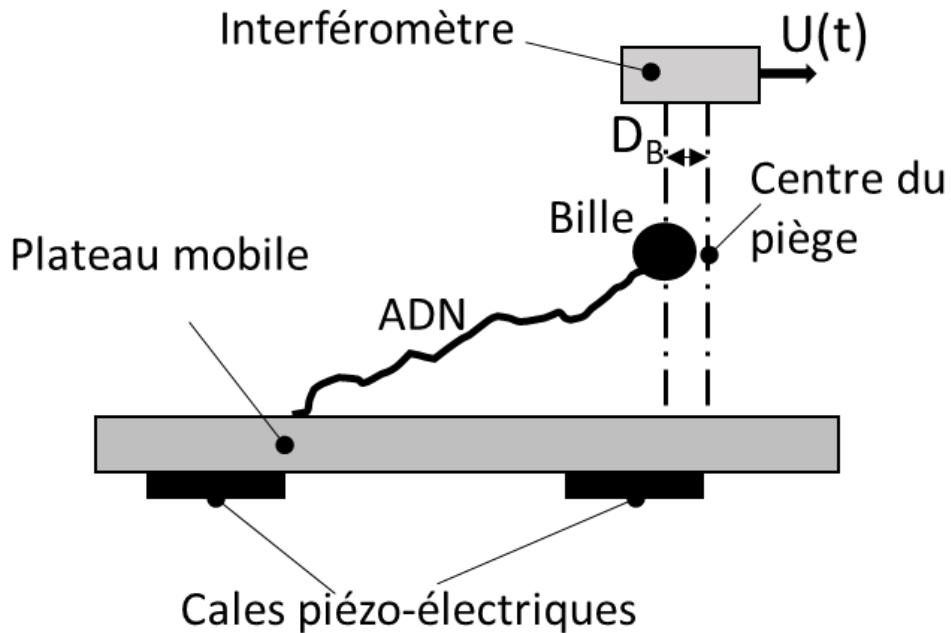


FIGURE 8 – Dispositif de mesure

On donne figure 9 les caractéristiques de la tension U en fonction du déplacement de la bille par rapport au centre optique (chaque courbe correspond à des billes de diamètres différents).

18. Pour quel intervalle de déplacement de la bille D_b la tension U est-elle une loi affine du déplacement ?
 Donner alors a et b tels que : $U = aD_b + b$.

La tension U est numérisée et codée par un entier naturel sur 16 bits.

19. Quel est l'ordre de grandeur de la résolution de la mesure de D_b permise par cette mesure ?

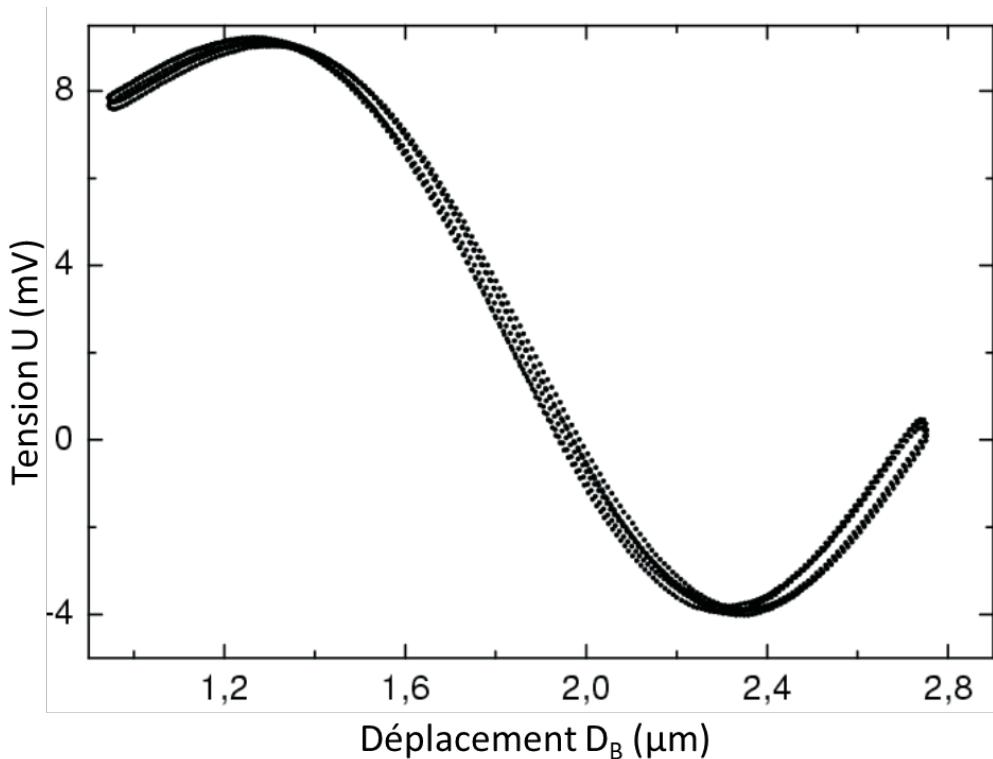


FIGURE 9 – Tension mesurée en fonction du déplacement de la bille.

Le diamètre de 1 μm des billes est de l'ordre de grandeur de la longueur d'onde du laser et ne permet donc pas d'utiliser simplement les règles de l'électromagnétisme, ni de l'optique géométrique pour calculer la force exercée par la lumière sur la bille. Il faut donc procéder à une calibration expérimentale. Nous allons étudier deux méthodes différentes de calibration.

FONCTIONS DE BASE :

Nous allons définir des fonctions qui devront être utilisées dans la suite du sujet. Les fonctions `sum()` et `mean()` sont proscrites.

20. Écrire une fonction `somme(a)` qui prend pour argument une liste et retourne la somme de ses éléments.
 21. Écrire une fonction `moyenne(a)` qui prend pour argument une liste et retourne la moyenne de ses éléments.

On rappelle que la variance ν est la moyenne des écarts à la moyenne au carré :

$$\nu = \frac{1}{n} \sum_{i=0}^{n-1} (x_i - \bar{X})^2 = \left(\frac{1}{n} \sum_{i=0}^{n-1} x_i^2 \right) - \bar{X}^2$$

où \bar{X} désigne la moyenne des x_i

22. Écrire une fonction `variance(a)` qui prend pour argument une liste et retourne la variance de ses éléments.

MÉTHODE 1 : OSCILLATIONS FORCÉES DE LA BILLE DANS UN LIQUIDE

Le principe de l'expérience consiste à capturer une bille dans le piège, à faire osciller l'échantillon et enfin, à mesurer la réponse de la bille. La figure 10 représente sur un même graphique la mesure du déplacement de la bille via la tension U ainsi que le déplacement sinusoïdal imposé par les cales piézo-électriques. Dans un intervalle de fréquence f bien choisi, l'amplitude du mouvement de la bille est proportionnelle au mouvement de l'échantillon et en quadrature de phase avec le déplacement imposé, de telle sorte que :

$$|D_b| = 2\pi f \frac{\gamma}{\kappa} |D_p|$$

avec D_p le déplacement imposé, κ la raideur du piège et γ le coefficient de frottement de l'eau. Cette méthode donne *in fine* la constante K telle que la force recherchée F vérifie : $F = KU$

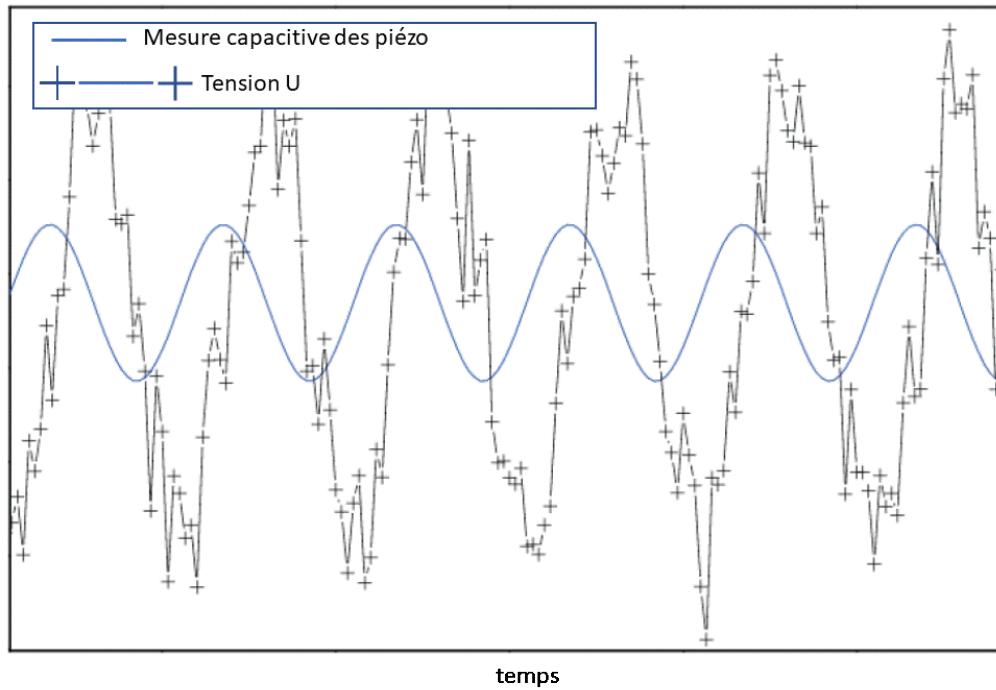


FIGURE 10 – Évolution temporelle des tensions mesurées avec la méthode 1.

On réitère la mesure pour différentes fréquences f puis l'on cherche la corrélation entre U et la fréquence f par une régression linéaire via une minimisation de l'erreur au carré comme représenté Figure 11

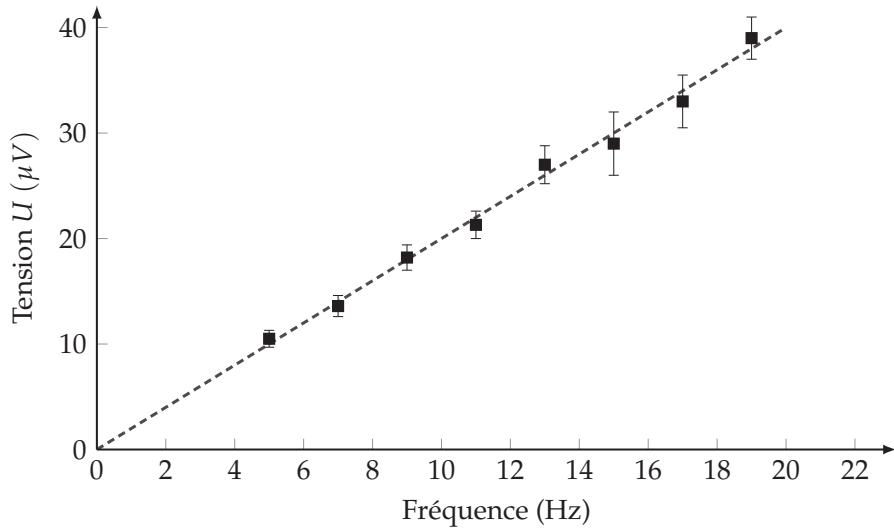


FIGURE 11 – Tensions mesurées par la méthode 1. Régression linéaire en fonction de la fréquence.

Le principe de la méthode des moindres carrés linéaires consiste à chercher la solution qui minimise la somme des erreurs au carré entre les données expérimentales et une fonction de la forme :

$$y(x) = \sum_{j=0}^{m-1} a_j f_j(x)$$

où les fonctions f_j sont connues, et où les coefficients a_j sont les inconnues du problème. On cherche alors les coefficients a_j qui minimisent e :

$$e = \sum_{i=0}^{n-1} \left(y_i - \sum_{j=0}^{m-1} a_j f_j(x_i) \right)^2$$

Notons qu'il faut que n , la taille des données soit telle que $n > m$.

Pour faire une régression linéaire, on choisit donc :

$$\begin{cases} y(x) = a_0 f_0(x) + a_1 f_1(x) \\ f_0(x) = 1 \\ f_1(x) = x \end{cases}$$

Et donc :

$$e = \sum_{i=0}^{n-1} (y_i - a_0 - a_1 x_i)^2$$

23. Montrer que :

$$a_1 = \frac{\sum_{i=0}^{n-1} x_i \sum_{i=0}^{n-1} y_i - n \sum_{i=0}^{n-1} x_i y_i}{\left(\sum_{i=0}^{n-1} x_i \right)^2 - n \sum_{i=0}^{n-1} x_i^2}$$

et

$$a_0 = \bar{Y} - a_1 \bar{X}$$

On admet que a_1 peut également s'écrire :

$$a_1 = \frac{\sum_{i=0}^{n-1} (x_i - \bar{X}) (y_i - \bar{Y})}{\sum_{i=0}^{n-1} (x_i - \bar{X})^2}$$

24. Écrire une fonction `regression_lineaire(x,y)` qui prend pour arguments deux listes d'abscisses x et d'ordonnées y et qui retourne les coefficients a_0 et a_1 tels que définis ci-dessus.
25. Évaluer l'ordre de la complexité d'un appel à la fonction `regression_lineaire` en fonction de la taille des données n .
26. Écrire un script qui permet de tracer la figure ci-dessus à partir de la liste des tensions U et des fréquences f correspondantes, les barres d'erreur ne sont pas à représenter.

Afin d'améliorer la corrélation on peut effectuer plusieurs mesures de la tension U pour une même fréquence f . On retiendra alors la moyenne des mesures pour une fréquence et l'on pourra évaluer la qualité de la mesure à cette fréquence en calculant la variance. On considère à présent que l'on dispose d'une liste f de dimension n et d'une liste de liste U de dimension $n \times l$. $U[i]$ est donc la liste des l mesures effectuées à la fréquence $f[i]$.

27. Écrire une fonction `traitement(a)` qui prend pour argument une liste de liste de dimension $n \times l$ et retourne une liste de liste de même dimension calculant la moyenne et la variance pour tout élément de a . On aura donc `len(traitement(a)[i])=2`.

On peut généraliser la méthode de régression linéaire estimée par les moindres carrés, pour tout ordre m par :

$$a = (J^T V J)^{-1} J^T V y$$

où l'on a appelé

- $a = \begin{bmatrix} a_0 \\ \vdots \\ a_{m-1} \end{bmatrix}$ les coefficients de la régression
- $y = \begin{bmatrix} y_0 \\ \vdots \\ y_{n-1} \end{bmatrix}$ et $x = \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix}$ les listes des données expérimentales,

- $J = \begin{bmatrix} f_0(x_0) & \dots & f_{m-1}(x_0) \\ \vdots & & \vdots \\ f_0(x_{n-1}) & \dots & f_{m-1}(x_{n-1}) \end{bmatrix}$ la matrice "Jacobienne".
- $V = \begin{bmatrix} \frac{1}{\nu_0^2} & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & \frac{1}{\nu_{n-1}^2} \end{bmatrix}$ est la matrice (diagonale) des poids, ν_i étant la variance de la mesure y_i .

Les mesures y_i et leur variance ν_i seront fournies dans une même liste de listes yv telle que $yv[i]$ retourne $[y_i, \nu_i]$

28. Écrire une fonction `regression_lineaire_gen(x, yv, listf)` qui prend pour argument une liste d'abscisses x , une liste de liste d'ordonnées et leur variances yv , une liste de fonction `listf` et qui retourne les coefficients a de la régression linéaire comme défini ci-dessus. On rappelle qu'une documentation partielle est disponible en annexe.
29. Évaluer l'ordre de la complexité d'un appel à la fonction `regression_lineaire_gen()` pour m et n quelconques. Comparer à la question 25. pour $m = 2$.

MÉTHODE 2 : ANALYSE DU MOUVEMENT BROWNIEN D'UNE BILLE PIÉGÉE

On analyse dans cette méthode le mouvement brownien de la bille. Une équation approchée de ce mouvement est donnée par l'équation de Langevin :

$$\gamma \frac{dx}{dt} + kx = F_\ell$$

avec γ et k des constantes.

La force de Langevin $F_\ell(t)$ correspond à un bruit blanc. Son "spectre de puissance" correspond au module de la fonction de transfert harmonique au carré. Il est noté : $S_f(\omega)$ et a pour expression :

$$S_f(\omega) = |\hat{F}_\ell(j\omega)|^2 = 4\gamma k_b T$$

avec k_b la constante de Boltzmann, et T la température.

30. En transformant l'équation de Langevin dans le domaine de Laplace harmonique (à condition initiale nulles) avec $p = j\omega$, exprimer le spectre de puissance du mouvement brownien de la bille $S_x(\omega) = |\hat{x}(j\omega)|^2$ en fonction de $S_f(\omega)$, γ et k .
31. Montrer que $S_x(f) = \frac{4k_b T}{\pi^2 \gamma} \frac{1}{f^2 + f_c^2}$ où $f = \frac{\omega}{2\pi}$ est la fréquence du signal. Donner l'expression de f_c .

Le déplacement de la bille est mesuré par le même dispositif que pour la méthode 1 et l'on dispose de son image en tension U . On réalise une série de mesures à partir desquelles on établit numériquement la densité spectrale S_U liée à la tension U . La Figure 12 représente la densité spectrale déterminée expérimentalement et une régression de cette densité telle que nous allons l'établir.

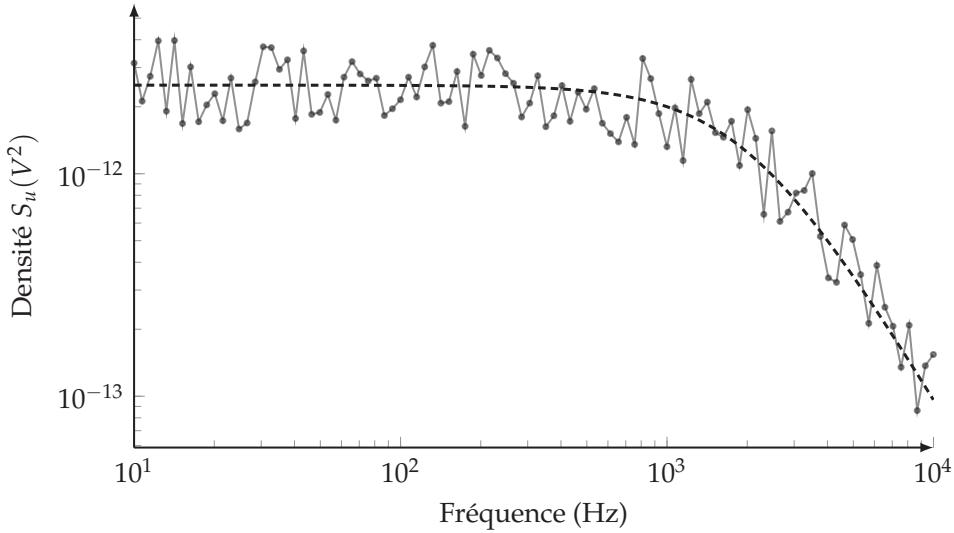


FIGURE 12 – Densité spectrale et identification

En exploitant les calculs précédents on montre que la force exercée sur la bille est liée à la tension par :

$$F = U \sqrt{2\pi\gamma k_b T \frac{f_c}{A}}$$

avec $A = \int S_u(f) df$ l'aire sous la courbe du spectre de puissance expérimental S_u . On dispose d'une liste S_u et d'une liste f de dimension n qui contiennent respectivement les valeurs numériques du spectre de puissance et de la fréquence correspondante. On souhaite à présent déterminer f_c et A .

32. Écrire une fonction `aire(x,y)` qui détermine l'aire sous la courbe définie par une liste d'abscisses x et d'ordonnées y de même dimension. Vous expliciterez clairement la méthode choisie.

Pour évaluer f_c nous allons dans un premier temps chercher la fonction de la forme (voir question 31.) :

$$G(f, a_0, a_1) = \frac{a_0}{a_1^2 + f^2}$$

qui minimise l'erreur au carré avec le spectre S_u expérimental. Le problème de minimisation étant cette fois non linéaire, on applique la méthode de descente du gradient qui consiste, comme pour la méthode de Newton, à rechercher itérativement les coefficients a_0 et a_1 qui annulent le gradient de l'erreur définie par :

$$e = \sum_{i=0}^{n-1} (S_{ui} - G(f_i, a_0, a_1))^2$$

Le principe de l'algorithme est le suivant :

- On choisit d'initialiser a_0 et a_1 à deux valeurs $a_{0,0}$ et $a_{1,0}$.
- On choisit le pas h de résolution et e_{rr} la condition d'arrêt du gradient.

- On calcule de façon itérative les coefficients a_0 et a_1 :

$$a_{0,k+1} = a_{0,k} - h \frac{\partial e}{\partial a_0} \Big|_{(a_{0,k}, a_{1,k})}$$

$$a_{1,k+1} = a_{1,k} - h \frac{\partial e}{\partial a_1} \Big|_{(a_{0,k}, a_{1,k})}$$

- L'algorithme s'arrête lorsque :

$$h \left(\frac{\partial e}{\partial a_0} \Big|_{(a_{0,k}, a_{1,k})} + \frac{\partial e}{\partial a_1} \Big|_{(a_{0,k}, a_{1,k})} \right) \leq e_{rr}$$

33. Donner les expressions analytiques de $\frac{\partial G}{\partial a_0}$ et $\frac{\partial G}{\partial a_1}$.
34. Écrire les fonctions $G(f, a0, a1)$, $dG_da0(f, a0, a1)$ et $dG_da1(f, a0, a1)$ qui retournent respectivement la valeur de G , $\frac{\partial G}{\partial a_0}$ et $\frac{\partial G}{\partial a_1}$ en fonction de leurs arguments.
35. Donner les expressions analytiques de $\frac{\partial e}{\partial a_0}$ et $\frac{\partial e}{\partial a_1}$, en fonction de $\frac{\partial G}{\partial a_0}$ et $\frac{\partial G}{\partial a_1}$ et des données.
36. Écrire une fonction `methode_gradient(G, dG_da0, dG_da1, Su, f, h, a00, a10, err)` qui détermine et retourne les paramètres optimaux a_0 et a_1 .
37. En considérant que i_T itérations sont nécessaires pour atteindre la condition d'arrêt de l'algorithme, donner la complexité de la méthode du gradient en fonction de la taille n des données expérimentales
38. Écrire une fonction `force(Su, f, h, a00, a10)` qui détermine et retourne la force exercée sur la bille.

Fin de l'énoncé

DOCUMENTATION PARTIELLE

Fonction plot

```
matplotlib.pyplot.plot(*args, **kwargs)
```

Plot lines and/or markers to the Axes. args is a variable length argument, allowing for multiple x, y pairs with an optional format string. For example, each of the following is legal :

```
plot(x, y) # plot x and y using default line style and color
plot(x, y, 'bo') # plot x and y using blue circle markers
plot(x, y, 'gs') # ditto, but with green square markers
plot(y) # plot y using x as index array 0..N-1
plot(y, 'r+') # ditto, but with red plusses
```

En plus de la fonction `plot`, le module `matplotlib.pyplot` propose diverses fonctions dédiées à la mise en forme des graphiques. En voici quelques-unes :

- `xlabel(s)` : écrit le contenu de la chaîne `s` comme étiquette des abscisses.
- `ylabel(s)` : écrit le contenu de la chaîne `s` comme étiquette des ordonnées.
- `title(s)` : écrit le contenu de la chaîne `s` comme titre du graphique.
- `legend(L)` : donne une légende au graphique. `L` doit être une liste de chaînes : `L[0]` est la légende de la première courbe, `L[1]` de la deuxième, etc.

Opérations de base sur les tableaux de type numpy :

- `numpy.array(u)` crée un nouveau tableau contenant les éléments de la séquence `u`. La taille et le type des éléments de ce tableau sont déduits du contenu de `u`.
- `numpy.empty(n, dtype)`, `numpy.empty((n, m), dtype)` crée respectivement un vecteur à `n` éléments ou un tableau à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype` qui peut être un type standard (`bool`, `int`, `float`, ...) ou un type spécifique `numpy` (`numpy.int16`, `numpy.float32`, ...). Si le paramètre `dtype` n'est pas précisé, il prend la valeur `float` par défaut.
- `numpy.zeros(n, dtype)`, `numpy.zeros((n, m), dtype)` fonctionne comme `numpy.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens.
- `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`.
- `numpy.transpose(a)` renvoie le transposé du tableau `a`.
- `numpy.dot(a, b)` calcule le produit matriciel des tableaux `a` et `b`.
- `numpy.linalg.inv(a)` renvoie l'inverse du tableau `a`, lève l'exception `ValueError` si `a` n'est pas un tableau carré à deux dimensions et `LinAlgError` si `a` n'est pas inversible.

Ordonnabilité des espaces métriques

Le sujet comporte 12 pages, numérotées de 1 à 12.

Début de l'épreuve.

Dans ce sujet, on s'intéresse au problème d'ordonner les éléments d'un espace métrique de sorte que deux éléments successifs soient à distance bornée.

Ce sujet est constitué de quatre parties. La première partie est formée de préliminaires utiles dans le reste du sujet, avec des questions de programmation en C et OCaml. La deuxième partie propose une implémentation en C d'une structure de données utilisée notamment dans la troisième partie. Cette troisième partie considère les graphes, munis de la distance de plus court chemin, comme espace métrique, avec implémentations en C. La quatrième partie, enfin, indépendante des deux précédentes, considère l'ordonnabilité des langages réguliers pour une certaine distance d'édition ; cette partie contient de la programmation en OCaml.

Dans les questions de programmation en C ou OCaml, on n'utilisera pas de fonctions qui ne sont pas incluses dans la bibliothèque standard du langage. Pour les codes en C, on pourra supposer que les en-têtes `<stdbool.h>`, `<stdio.h>`, `<stdlib.h>` et `<assert.h>` ont été inclus.

Espaces métriques et d -ordres

Dans ce sujet, on s'intéresse à des collections de données discrètes que l'on cherche à explorer ou engendrer de proche en proche. On modélise ce cadre général à l'aide des notions d'*espace métrique*, de *d -suite* et de *d -ordre* :

Définition 1 (espace métrique). *Un espace métrique $\mathcal{M} = (X, \delta)$ est constitué d'un ensemble dénombrable X dont les éléments sont appelés points et d'une distance sur X , c'est-à-dire une application $\delta : X \times X \rightarrow \mathbb{N}$ satisfaisant :*

Symétrie. Pour tout $(x, y) \in X^2$, $\delta(x, y) = \delta(y, x)$.

Séparation. Pour tout $(x, y) \in X^2$, $\delta(x, y) = 0$ si et seulement si $x = y$.

Inégalité triangulaire. Pour tout $(x, y, z) \in X^3$, $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$.

Pour un espace métrique $\mathcal{M} = (X, \delta)$ et $X' \subseteq X$, on note $\mathcal{M}[X']$ le couple $(X', \delta|_{X'})$, où $\delta|_{X'}$ dénote la restriction de δ au domaine $X' \times X'$. On observe que $\mathcal{M}[X']$ est encore un espace métrique, que l'on appellera sous-espace de \mathcal{M} .

Définition 2 (d -suite). *Pour $d \in \mathbb{N}^*$, une d -suite \mathbf{s} dans un espace métrique $\mathcal{M} = (X, \delta)$ est une suite (finie ou infinie dénombrable) $\mathbf{s} = x_1, x_2, \dots$ de points de \mathcal{M} telle que :*

- \mathbf{s} ne contient pas de doublons, c'est-à-dire que pour tous x_i, x_j de la suite, si $i \neq j$ alors $x_i \neq x_j$;
- deux points consécutifs de la suite sont à distance au plus d ; formellement, pour tous x_i, x_{i+1} de la suite on a $\delta(x_i, x_{i+1}) \leq d$.

On dit que \mathbf{s} commence en x_1 et, dans le cas où \mathbf{s} est finie et a n éléments, que \mathbf{s} termine en x_n .

Définition 3 (d -ordre, d -ordonnable). *Pour $d \in \mathbb{N}^*$, un d -ordre de \mathcal{M} est une d -suite dans \mathcal{M} contenant tous les points de \mathcal{M} . L'espace \mathcal{M} est dit d -ordonnable lorsqu'il existe un d -ordre de \mathcal{M} . On dit que \mathcal{M} est ordonnable lorsque \mathcal{M} est d -ordonnable pour un certain d .*

Distances d'édition sur les mots

On fixe dans tout le sujet l'*alphabet* $\Sigma := \{a, b\}$ ayant pour seules lettres a et b . Un *mot* w est une suite finie de lettres $w = \alpha_1 \cdots \alpha_n$. La longueur de w , notée $|w|$, est n . On note ε le mot vide, de longueur nulle. On note Σ^* l'ensemble des mots sur Σ . Un *langage* est un sous-ensemble de Σ^* .

Un espace métrique d'intérêt, sur l'ensemble des mots d'un langage, est donné par deux distances d'édition sur les mots : la distance *push-pop* et la distance *push-pop-droite*, que l'on définit ci-après.

Définition 4 (distance push-pop). *La distance push-pop, dénotée δ_{pp} , est définie de la manière suivante : pour $w, w' \in \Sigma^*$, $\delta_{\text{pp}}(w, w')$ est le nombre minimal d'opérations nécessaires pour passer de w à w' , où les opérations autorisées sont :*

- pour un mot w , insérer la lettre $\alpha \in \Sigma$ à la fin, ce qui donne le mot $w\alpha$;
- pour un mot w , insérer la lettre $\alpha \in \Sigma$ au début, ce qui donne le mot αw ;
- pour un mot de la forme $w\alpha$ avec $\alpha \in \Sigma$, supprimer la dernière lettre, ce qui donne le mot w ;
- pour un mot de la forme αw avec $\alpha \in \Sigma$, supprimer la première lettre, ce qui donne le mot w .

Exemple 1. Les mots à distance push-pop 1 du mot aab sont aa (on a supprimé la dernière lettre), $aaba$ (on a ajouté un a à la fin), $aabb$ (on a ajouté un b à la fin), ab (on a supprimé la première lettre), $aaab$ (on a ajouté un a au début) et $baab$ (on a ajouté un b au début).

Définition 5 (distance push-pop-droite). *La distance push-pop-droite, dénotée δ_{ppr} , est définie de la même manière que δ_{pp} mais seules les insertions et suppressions à la fin du mot sont autorisées.*

Exemple 2. Les mots qui sont à distance push-pop-droite 1 du mot aab sont aa (on a supprimé la dernière lettre), $aaba$ (on a ajouté un a à la fin) et $aabb$ (on a ajouté un b à la fin).

Algorithmes d'énumération push-pop et push-pop-droite

Lorsqu'on travaillera sur les mots, on considérera parfois des programmes produisant une suite (potentiellement infinie) de mots de Σ^* , en utilisant les instructions spéciales suivantes : `popL()`, `popR()`; `pushL(α)` et `pushR(α)` pour $\alpha \in \Sigma$; et `output()`.

Le programme dispose, comme état interne, d'une liste L d'éléments de Σ , qui est interprétée comme un mot de Σ^* . La liste L est initialement vide et représente donc le mot vide. Voici ce qu'il se passe lorsque le programme utilise les instructions spéciales :

- `popL()` a pour effet de supprimer la première lettre de la liste (et ne peut être appelée que si la liste est non vide) ;
- `popR()` a pour effet de supprimer la dernière lettre de la liste (et ne peut être appelée que si la liste est non vide) ;
- `pushL(α)` a pour effet d'ajouter la lettre $\alpha \in \Sigma$ en début de liste ;
- `pushR(α)` a pour effet d'ajouter la lettre $\alpha \in \Sigma$ en fin de liste ;
- `output()` produit le mot étant actuellement représenté par la liste (par exemple, il l'affiche en sortie du programme).

On souligne que la liste L n'est accessible par le programme que via ces instructions spéciales. De plus, on fait l'hypothèse que chacune des instructions `popL`, `popR`, `pushL` et `pushR` a une complexité en $O(1)$.

Un tel programme *produit* alors la suite de mots w_1, w_2, \dots , où w_1 est le premier mot produit par le programme lors de son exécution, w_2 le second et ainsi de suite.

On appellera un tel programme un *programme push-pop*. De manière similaire, un programme *push-pop-droite* est un programme push-pop qui n'utilise pas les instructions `popL()` ni `pushL(α)` pour $\alpha \in \Sigma$.

Exemple 3. On suppose que les fonctions C pushL, pushR, popL, popR, output ont été prédéfinies et manipulent toutes une même variable représentant la liste. Le programme C suivant est un programme push-pop-droite qui produit la suite de mots w_1, w_2, \dots où w_i est $(aa)^{i-1}b$:

```
int main(void)
{
    pushR('b'); output();
    while (true) {
        popR(); pushR('a'); pushR('a'); pushR('b'); output();
    }
}
```

On remarque que ce programme ne termine jamais.

En OCaml, on suppose qu'on dispose d'un type `lettre` défini par :

```
type lettre = A | B
```

On représente alors les mots de Σ^* par des listes OCaml `lettre list`, dont la tête correspond à la première lettre du mot. Par exemple, le mot aab est représenté par la liste `[A;A;B]`. On suppose avoir accès en OCaml à des fonctions de type `pushR : lettre -> unit`, `pushL : lettre -> unit`, `popR : unit -> unit`, `popL : unit -> unit`, ainsi que `output : unit -> unit`, qui représentent les instructions spéciales.

Exemple 4. Le programme OCaml push-pop suivant produit la même suite que le programme C push-pop-droite de l'exemple 3 :

```
let main () =
    pushR B; output ();
    while true do
        pushL A; pushL A; output ();
    done
```

Partie I : Préliminaires

Question I.1. Soit $\mathcal{M} = (X, \delta)$ un espace métrique tel que X est un ensemble fini. Montrer que \mathcal{M} est ordonnable.

Question I.2. Écrire une fonction OCaml

```
delta_ppr : lettre list -> lettre list -> int
```

tenant en entrée deux listes l_1, l_2 représentant des mots w_1, w_2 et renvoyant $\delta_{\text{ppr}}(w_1, w_2)$, la distance push-pop-droite entre w_1 et w_2 . On attend de cette fonction que sa complexité soit linéaire en $|w_1| + |w_2|$.

Question I.3. On considère la suite $s := w_1, w_2, \dots$, où w_i est a^{i^2} . Écrire un programme push-pop-droite en C qui produit la suite s .

Question I.4. Montrer que $\mathcal{M}_{\text{pp}} := (\Sigma^*, \delta_{\text{pp}})$ et $\mathcal{M}_{\text{ppr}} := (\Sigma^*, \delta_{\text{ppr}})$ sont des espaces métriques.

On s'intéresse maintenant aux sous-espaces de \mathcal{M}_{pp} et \mathcal{M}_{ppr} , c'est-à-dire (comme indiqué dans la Définition 1) aux espaces métriques de la forme $\mathcal{M}_{\text{pp}}[L] = (L, \delta_{\text{pp}|L})$ ou $\mathcal{M}_{\text{ppr}}[L] = (L, \delta_{\text{ppr}|L})$, pour L un langage. Par exemple, la suite produite par les programmes push-pop des exemples 3 et 4 est un 4-ordre pour $\mathcal{M}_{\text{ppr}}[L]$ et un 2-ordre pour $\mathcal{M}_{\text{pp}}[L]$, où $L := (aa)^*b$.

Question I.5. Écrire un programme push-pop en C qui produit un d -ordre pour $\mathcal{M}_{\text{pp}}[L]$, où $L := a^*b^* \mid b^*a^*$ et un certain $d \in \mathbb{N}$. Expliquer comment fonctionne ce programme.

Question I.6. Écrire un programme push-pop en C qui produit un 1-ordre pour $\mathcal{M}_{\text{pp}}[L]$, où $L := a^*b^*$. [Indice : on peut visualiser le langage L comme la grille $\mathbb{N} \times \mathbb{N}$, où à la position (i, j) correspond le mot $a^i b^j$.] Ne pas hésiter à faire un dessin pour se faire comprendre.

Question I.7. Soit $L := b^* \mid ab^*$.

- Écrire un programme push-pop en C qui produit un 1-ordre pour $\mathcal{M}_{\text{pp}}[L]$.
- Prouver que $\mathcal{M}_{\text{ppr}}[L]$ n'est pas ordonnable.

Question I.8. Un *chemin hamiltonien* dans un graphe non-orienté est un chemin (aussi appelé *chaîne*) qui visite tous les sommets du graphe exactement une fois. Un *graphe grille* est un graphe de la forme $G = (V, E_V)$ avec V une partie finie de $\mathbb{N} \times \mathbb{N}$ et $E_V = \{\{(i, j), (i', j')\} \mid (i, j), (i', j') \in V \text{ et } |i - i'| + |j - j'| = 1\}$, autrement dit, deux sommets sont connectés par une arête s'ils sont à distance 1 dans la grille $\mathbb{N} \times \mathbb{N}$.

On définit le problème de décision HamGrille de la manière suivante : en entrée on a un graphe grille G et la sortie est OUI si G possède un chemin hamiltonien et NON sinon. On admettra que ce problème est NP-complet.

Pour t un entier naturel non nul, on définit le problème de décision \mathcal{P}_t , de la façon suivante : \mathcal{P}_t prend en entrée un ensemble fini de mots L et la sortie est OUI si $\mathcal{M}_{\text{pp}}[L]$ est t -ordonnable, NON sinon. On fixe maintenant un entier naturel $t \geq 1$ arbitraire. Montrer que pour cet entier t , le problème \mathcal{P}_t est NP-complet.

Partie II : Implémentation en C des programmes push-pop

Dans cette partie, on souhaite réaliser une implémentation en langage C des opérations des programmes push-pop sur une structure de données `liste` codant une liste doublement chaînée. Le type de la structure de données `liste` est défini comme suit :

```
typedef struct chainon_s chainon;

struct chainon_s
{
    int val;
    chainon *prec;
    chainon *suiv;
};

struct liste_s
{
    chainon *premier;
    chainon *dernier;
};

typedef struct liste_s liste;
```

En particulier, les lettres sont donc représentées par des entiers (on rappelle qu'en C, une variable de type `char` peut être implicitement interprétée comme variable de type `int` sans perte d'informations) ; cela permettra de réutiliser cette structure `liste` dans d'autres contextes.

Dans l'implémentation de cette structure de données, on veillera à ce qu'un pointeur dont la valeur n'est pas `NULL` pointe toujours vers un espace mémoire valide et pas, par exemple, vers une donnée libérée.

Question II.1. Définir et initialiser une variable globale `lg` représentant la liste chaînée globale manipulée par les programmes push-pop ; on souhaite que cette liste soit initialement vide.

Question II.2. Programmer une fonction `est_vide` de prototype `bool est_vide(void)` renvoyant `true` si `lg` représente une liste vide, `false` sinon.

Question II.3. Programmer des fonctions `pushL` et `pushR` de prototypes `void pushL(int)` et `void pushR(int)` implémentant les opérations `pushL` et `pushR` sur la liste représentée par `lg`. On utilisera une assertion pour vérifier que l'allocation dynamique de mémoire est bien réalisée. Quelle est la complexité en temps de ces deux fonctions ?

Question II.4. Programmer des fonctions `popL` et `popR` de prototypes `int popL(void)` et `int popR(void)` implémentant les opérations `popL` et `popR` sur la liste représentée par `lg` (chacune renvoyant également la valeur extraite de la liste). On utilisera une assertion pour s'assurer que la liste n'est pas vide et on veillera à libérer la mémoire devenue inutile. Quelle est la complexité en temps de ces deux fonctions ?

Question II.5. Programmer des fonctions `output` et `vide_liste` de prototypes respectifs `void output(void)` et `void vide_liste(void)`; `output` affiche le mot codé par la liste de caractères sur la sortie standard (suivi d'un retour à la ligne), tandis que `vide_liste` vide la liste (retire tous ses éléments) et libère la mémoire dynamique associée. Quelle est la complexité en temps de ces deux fonctions ?

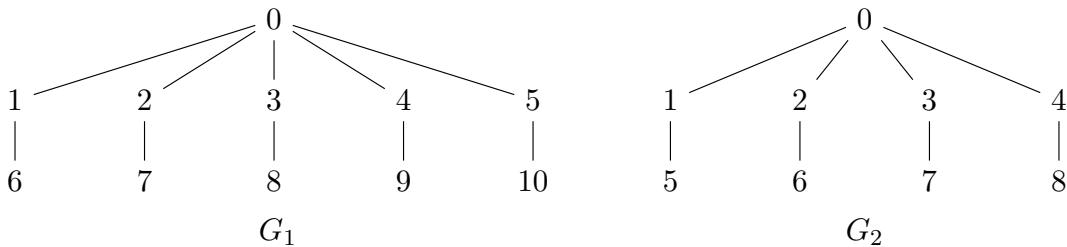
Partie III : Ordonnabilité des graphes

Dans cette partie on s'intéresse à des espaces métriques définis à partir de graphes non-orientés connexes. Pour un graphe non-orienté connexe $G = (V, E)$ (où V est l'ensemble fini non-vide de noeuds du graphe et E son ensemble d'arêtes), on note $\delta_G : V \times V \rightarrow \mathbb{N}$ la fonction qui à un couple de noeuds (u, v) de V associe la longueur d'un plus court chemin de u à v dans G .

L'objectif est de montrer que les espaces de la forme $\mathcal{M}_G := (V, \delta_G)$ sont 3-ordonnables et d'étudier des algorithmes qui calculent de tels ordres. On travaillera en langage C.

Question III.1. Soit $G = (V, E)$ un graphe non-orienté connexe. Montrer que le couple $\mathcal{M}_G := (V, \delta_G)$ est effectivement un espace métrique.

Question III.2. On considère les graphes G_1 et G_2 représentés ci-dessous.



- a. Donner un 3-ordre de \mathcal{M}_{G_1} (aucune justification n'est attendue).
- b. Donner un 2-ordre de \mathcal{M}_{G_2} (aucune justification n'est attendue).

On souhaite calculer un arbre couvrant d'un graphe non-orienté connexe. Un *arbre* est un graphe orienté acyclique $T = (V_T, E_T)$ avec V_T un ensemble fini non vide de noeuds et $E_T \subseteq V_T \times V_T$ un ensemble d'arcs tel que :

- il existe un unique noeud $r \in V_T$, appelé la *racine* de T tel que pour tout $v \in V_T$, $(v, r) \notin E_T$;
- pour tout noeud $u \in V_T$ qui n'est pas la racine, il existe un unique noeud $v \in V_T$ tel que $(v, u) \in E_T$ et on dit que u est un *fils* de v .

On définit maintenant un *arbre couvrant* d'un graphe non-orienté connexe $G = (V, E)$ comme un arbre $T = (V', E')$ tel que :

- $V' = V$;
- pour tout $(u, v) \in E'$, $\{u, v\} \in E$.

On va travailler avec la représentation en matrice d'adjacence des graphes : on suppose que le nombre de noeuds N est prédéfini en C comme une constante globale \mathbb{N} , que les noeuds de G sont $\{0, \dots, N - 1\}$ et que G est représenté par un tableau bidimensionnel `bool graphe[N][N]` où `g[u][v]` et `g[v][u]` valent tous les deux `true` si $\{u, v\}$ est une arête de G , et `false` sinon.

L'arbre couvrant T sera également représenté par un tableau `bool arbre[N][N]`, où cette fois `arbre[u][v]` vaut `true` si v est un fils de u et `false` sinon.

Question III.3. Écrire une fonction

```
void calcule_arbre_couvrant(bool graphe[N][N], bool arbre[N][N])
```

qui remplit la structure `arbre` pour que celle-ci représente un arbre couvrant quelconque du graphe G représenté par `graphe`, avec le noeud 0 comme racine. On supposera G connexe et on ne vérifiera pas ce point dans le code. On pourra utiliser les fonctions définies dans la partie II pour gérer une file (ou une pile) de noeuds à traiter. Justifier brièvement la correction de votre algorithme.

On supposera par la suite que `arbre` représente effectivement un arbre couvrant comme indiqué plus haut et que le noeud 0 est la racine de celui-ci.

Question III.4. On considère la fonction C suivante :

```
void visite(bool arbre[N][N], int noeud, bool mystere) {
    if(!mystere)
        pushR(noeud);
    for (int fils = 0; fils < N; ++fils) {
        if (arbre[noeud][fils])
            visite(arbre, fils, !mystere);
    }
    if(mystere)
        pushR(noeud);
}
```

On suppose que la liste `lg` utilisée par la fonctions `pushR` est initialement vide et qu'on appelle `visite(arbre, 0, false)` sur un arbre `arbre` de racine 0.

- Que vaut le troisième argument `mystere` lors d'un appel récursif à `visite` avec comme deuxième argument un noeud n de l'arbre, en fonction de la position de n dans l'arbre ?
- Que contient la liste `lg` à la fin de l'exécution de ce programme sur le graphe G_2 de la question III.2, vu comme un arbre enraciné en 0 ?

Question III.5. Montrer que l'appel `visite(a, 0, false)`, lorsque `a` est la représentation d'un arbre couvrant d'un graphe non-orienté connexe G de racine 0, calcule (dans la liste `lg`) un 3-ordre pour \mathcal{M}_G .

Question III.6. Quelle est la complexité en temps de l'appel à `visite(a, 0, false)`, en fonction du nombre N de noeuds ? Peut-on améliorer cette complexité (par exemple en changeant notre méthode de représentation des graphes) ?

Question III.7. On observe que la fonction `visite` est une fonction récursive.

- Quel(s) problème(s) cela pourrait-il poser ?
- Proposer une méthode pour rendre ce calcul non récursif. On ne s'attend pas ici à du code, mais à une explication qui puisse être facilement transformée en du code.

Partie IV : Ordonnabilité des langages réguliers pour la distance push-pop-droite

Dans cette partie on souhaite caractériser les langages réguliers ordonnable pour la distance push-pop-droite. On travaillera en OCaml.

Automates et langages réguliers. Le terme *automate* désignera systématiquement un automate fini déterministe. Formellement, un automate $A = (Q, q_{\text{init}}, F, \text{trans})$ consiste en un ensemble fini Q d'*états*, un état initial q_{init} , un ensemble $F \subseteq Q$ d'*états finaux*, ainsi qu'une *fonction de transition* $\text{trans} : Q \times \Sigma \rightarrow Q \cup \{\perp\}$, où \perp signifie que la transition n'est pas définie. Un *chemin* dans A d'un état $q \in Q$ à un état $q' \in Q$ est une suite finie de la forme $q_1, \alpha_1, q_2, \alpha_2, \dots, \alpha_{n-1}, q_n$ où les q_i sont des états de Q et les α_i des lettres de Σ , telle que $q = q_1$, $q' = q_n$ et telle que pour tout $i \in \{1, \dots, n-1\}$ on a $q_{i+1} = \text{trans}(q_i, \alpha_i)$; l'*étiquette* d'un tel chemin est alors le mot $\alpha_1 \dots \alpha_{n-1}$. En particulier, il y a toujours un chemin de longueur nulle, avec étiquette ε , entre n'importe quel état et lui-même (la suite comprenant ce seul état). Le *langage accepté par* A , noté $L(A)$, est l'ensemble des mots qui étiquettent un chemin depuis q_{init} jusqu'à un état final. Un langage est *régulier* s'il est accepté par un automate ou, de manière équivalente, s'il est représenté par une expression régulière.

On rappelle qu'un automate $A = (Q, q_{\text{init}}, F, \text{trans})$ est *émondé* si tout état q est à la fois accessible depuis l'état initial (c'est-à-dire qu'il y a un chemin depuis q_{init} à q) et co-accessible depuis un état final (c'est-à-dire qu'il y a un chemin depuis q vers un état final). Pour tout automate A , on peut calculer en temps linéaire un automate A' émondé tel que $L(A) = L(A')$.

Automates poêle. D'après la question I.1, si L est fini alors $\mathcal{M}_{\text{ppr}}[L]$ est ordonnable. **On supposera L infini dans cette partie.** On définit dans ce qui suit la notion d'*automate poêle-à-frirre* (ou *automate poêle*), puis on montre que $\mathcal{M}_{\text{ppr}}[L]$ est ordonnable si et seulement si n'importe quel automate émondé acceptant L est un automate poêle.

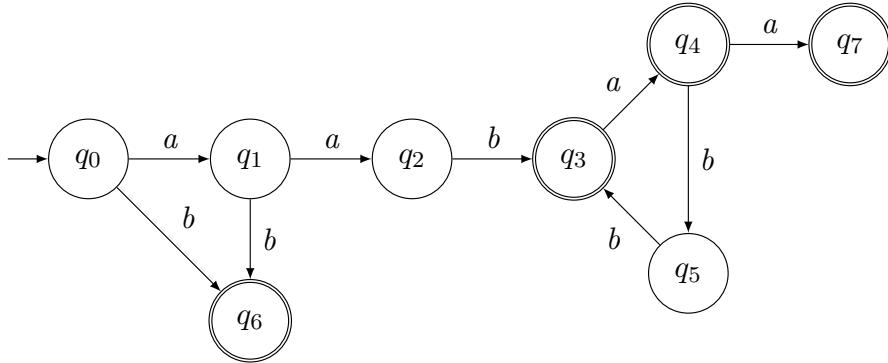
Soit $A = (Q, q_{\text{init}}, F, \text{trans})$ un automate. Un *cycle* dans A est un chemin de longueur non nulle dans A depuis un état q vers lui-même sans répétition d'état (à part q). On observe que ce chemin est possiblement de longueur 1, dans le cas où $\text{trans}(q, \alpha) = q$ pour $\alpha \in \Sigma$. On note que si $q_1, \alpha_1, q_2, \alpha_2, \dots, \alpha_{n-1}, q_1$ est un cycle, alors $q_2, \alpha_2, \dots, \alpha_{n-1}, q_1, \alpha_1, q_2$ et plus généralement $q_i, \alpha_i, \dots, \alpha_{n-1}, q_1, \alpha_1, \dots, \alpha_{i-1}, q_i$ pour tout $2 \leq i \leq n-2$ sont également des cycles : on dit que ce sont les mêmes cycles à *permutation près*.

Un *chemin strict vers un cycle* dans A est un chemin sans répétition d'état depuis l'état initial jusqu'à un état faisant partie d'un cycle tel que tous les états sauf le dernier ne font pas partie d'un cycle dans A ; formellement, c'est un chemin $q_1, \alpha_1, \dots, \alpha_{n-1}, q_n$ pour $n \in \mathbb{N}$ avec $q_1 = q_{\text{init}}$ tel que q_n fait partie d'un cycle et, pour $1 \leq i < n$, q_i ne fait partie d'aucun cycle. En particulier, si q_{init} fait partie d'un cycle alors le seul tel chemin est de longueur nulle.

On dit de A qu'il est *pseudo-acyclique* s'il a au plus un cycle à permutation près. On note qu'un automate tel que $\text{trans}(q, a) = \text{trans}(q, b) = q$ n'est jamais pseudo-acyclique, car q, a, q et q, b, q sont deux cycles différents.

Un automate A est alors un *automate poêle* s'il est pseudo-acyclique et a un unique chemin strict vers un cycle.

Exemple 5. Considérons l'automate ci-dessous :



Son ensemble d'états est $\{q_0, \dots, q_7\}$, son état initial q_0 , ses états finaux q_3, q_4, q_6 et q_7 ; les transitions non spécifiées par des flèches sont non définies. Cet automate est clairement déterministe et il est aisément vérifiable qu'il est émondé.

L'automate est également pseudo-acyclique : il comporte en effet un unique cycle, le cycle $q_3, a, q_4, b, q_5, b, q_3$ (qu'on peut également écrire $q_4, b, q_5, b, q_3, a, q_4$ ou encore $q_5, b, q_3, a, q_4, b, q_5$). Et comme il a un unique chemin strict vers un cycle (le chemin $q_0, a, q_1, a, q_2, b, q_3$), c'est un automate poêle.

Si une transition de q_3 à q_4 étiquetée par b était ajoutée, ce ne serait plus un automate pseudo-acyclique car $q_3, a, q_4, b, q_5, b, q_3$ et $q_3, b, q_4, b, q_5, b, q_3$ seraient deux cycles différents. De même, si une transition étiquetée par a de q_2 à l'un des états du cycle était ajoutée, l'automate aurait deux chemins stricts distincts vers un cycle et ne serait donc plus un automate poêle.

Question IV.1. Proposer une méthode permettant de déterminer si un automate émondé est un automate poêle, en supposant n'importe quelle représentation raisonnable des automates (on supposera l'automate émondé). On ne demande pas du code, mais on attend une explication qui puisse être facilement transformée en du code.

Lorsque A est un automate poêle, on appelle *état d'entrée* l'état qui se trouve à la fin de l'unique chemin strict vers l'unique cycle de A (cet état peut être l'état initial si celui-ci fait partie du cycle). Dans l'exemple 5, l'état q_3 est l'état d'entrée.

Question IV.2. Si A est un automate poêle, montrer que $L(A)$ peut s'écrire de la forme

$$F \mid uv^*F'$$

où F et F' sont des ensembles finis de mots et u, v sont des mots avec $v \neq \varepsilon$.

Question IV.3. On suppose que A est un automate poêle. Écrire un programme push-pop-droite en OCaml qui calcule un d -ordre pour $\mathcal{M}_{\text{ppr}}[L(A)]$, pour un certain $d \in \mathbb{N}$ que l'on ne cherchera pas à calculer, étant donné les mots u, v et ensembles F, F' de la question précédente représentés en OCaml par des variables :

```

u : lettre list
v : lettre list
f : (lettre list) list
f' : (lettre list) list
  
```

On a ainsi établi que lorsqu'un langage L est accepté par un automate poêle alors il est ordonnable pour la distance push-pop-droite. On montre dans la suite de cette partie que c'est en fait une condition nécessaire, dans le sens où si $\mathcal{M}_{\text{ppr}}[L]$ est ordonnable, alors n'importe quel automate émondé pour L est un automate poêle.

Pour ce faire, on considère l'arbre infini T_Σ de Σ^* défini ainsi : les noeuds de T_Σ sont les mots de Σ^* , le mot vide est la racine de T_Σ et si $w \in \Sigma^*$ alors wa et wb sont les fils de w dans T_Σ . Une *branche infinie* B dans T_Σ est une suite infinie de la forme w_1, w_2, \dots , où $w_1 = \varepsilon$ et $w_{i+1} = w_i \alpha_i$ avec $\alpha_i \in \Sigma$ pour tout $i \in \mathbb{N}$. Pour $w \in \Sigma^*$, on note T_w le sous-arbre infini de T_Σ enraciné en w .

Pour un langage L , une *branche lourde* est une branche infinie $B = w_1, w_2, \dots$ dans T_Σ telle que pour tout $i \in \mathbb{N}$, T_{w_i} contient une infinité de mots de L .

Question IV.4. Montrer que, pour n'importe quel langage L , si L est infini alors L a (au moins) une branche lourde.

Question IV.5. Montrer que, pour n'importe quel langage L , si $\mathcal{M}_{\text{ppr}}[L]$ est ordonnable alors L a au plus une branche lourde.

Question IV.6. Soit A un automate émondé qui a au moins deux chemins stricts (distincts) vers des cycles (potentiellement identiques) et soient u et u' des mots étiquetant deux tels chemins.

- a. Montrer que u n'est pas un préfixe de u' et que u' n'est pas un préfixe de u .
- b. Montrer que $L(A)$ a au moins deux branches lourdes.

Question IV.7. Soit A un automate émondé qui a un seul chemin strict vers un cycle. Montrer que si A n'est pas pseudo-acyclique alors $L(A)$ a au moins deux branches lourdes. En déduire que pour un langage régulier L infini, $\mathcal{M}_{\text{ppr}}[L]$ est ordonnable si et seulement si n'importe quel automate émondé A acceptant L est un automate poêle.

Fin du sujet.

**ECOLE POLYTECHNIQUE
ECOLES NORMALES SUPERIEURES**

CONCOURS D'ADMISSION 2023

MARDI 18 AVRIL 2023

14h00 - 18h00

FILIÈRE MP-MPI - Epreuve n° 4

INFORMATIQUE A (XULSR)

Durée : 4 heures

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve

Cette composition ne concerne qu'une partie des candidats de la filière MP, les autres candidats effectuant simultanément la composition de Physique et Sciences de l'Ingénieur. Pour la filière MP, il y a donc deux enveloppes de Sujets pour cette séance.

Compression entropique

Le sujet comporte 11 pages, numérotées de 1 à 11.

Vue d'ensemble du sujet.

Ce sujet s'intéresse à la compression et à la décompression de mots sur un alphabet fini \mathcal{S} de cardinal $|\mathcal{S}| \geq 2$. Plus précisément, on se donne une fonction q de \mathcal{S} dans \mathbb{N} et on s'intéresse aux mots de \mathcal{S}^* tels que chaque lettre $\sigma \in \mathcal{S}$ a exactement $q(\sigma)$ occurrences dans ces mots. On note \mathcal{S}^q l'ensemble de ces mots. Remarque : les mots de \mathcal{S}^q ont pour longueur $N = \sum_{\sigma \in \mathcal{S}} q(\sigma)$.

Une lettre de \mathcal{S} peut être représentée avec $\lceil \log_2 |\mathcal{S}| \rceil$ bits, où $\lceil x \rceil$ désigne la partie entière supérieure de x , c'est-à-dire l'entier tel que $\lceil x \rceil - 1 < x \leq \lceil x \rceil$. Si l'on concatène les bits représentant chacune des lettres d'un mot de \mathcal{S}^q , on peut donc représenter ce mot de façon non ambiguë à l'aide d'une séquence de bits de taille

$$\sum_{\sigma \in \mathcal{S}} q(\sigma) \cdot \lceil \log_2 |\mathcal{S}| \rceil = \lceil \log_2 |\mathcal{S}| \rceil \cdot N.$$

Il s'agit d'une représentation non compressée.

La théorie de l'information affirme qu'il faut au moins $\sum_{\sigma \in \mathcal{S}} q(\sigma) \cdot \log_2(N/q(\sigma))$ bits pour représenter tous les mots de \mathcal{S}^q de façon non ambiguë. Ce sujet explore quelques façons de compresser les mots pour s'approcher de cette borne théorique.

La partie I s'intéresse à la fonction q , c'est-à-dire au comptage des lettres d'un mot. La partie II étudie les arbres binaires dont les feuilles sont étiquetées par des lettres de \mathcal{S} . La partie III utilise ces arbres pour (dé)compresser des mots de \mathcal{S}^* . La partie IV s'intéresse aux arbres qui donnent les meilleurs taux de compression pour les mots de \mathcal{S}^q . Certains de ces arbres ont une représentation compacte ; c'est l'objet de la partie V. D'autres arbres sont encore plus compacts, mais au prix d'un moindre taux de compression, comme le montre la partie VI. Finalement, la partie VII attaque le problème d'une façon complètement différente afin de s'approcher un peu plus du taux de compression théorique optimal.

Les différentes parties sont indépendantes : il n'est pas nécessaire d'avoir répondu aux questions d'une partie pour répondre aux questions d'une autre partie ; cependant les notions abordées dans une partie sont souvent utiles aux parties suivantes.

Rappels d'OCaml

On rappelle ici quelques opérations de base sur les tableaux :

- `Array.length t` renvoie la longueur du tableau `t`.
- `Array.make n v` crée un tableau de `n` cases qui sont toutes initialisées avec `v`.
- `Array.of_list l` renvoie un tableau initialisé avec les valeurs de la liste `l`, tandis que `Array.to_list t` réalise l'opération inverse.
- La case numéro `i` du tableau `t` peut être accédée avec `t.(i)`. Les cases sont numérotées à partir de zéro.

Partie I. Comptage d'occurrences

Un texte non compressé est un mot de \mathcal{S}^* . Dans la suite, à chaque fois qu'il s'agira de définir une fonction en OCaml, les lettres de \mathcal{S} seront représentées par des entiers et un mot de \mathcal{S}^* sera représenté par un tableau d'entiers (`int array`) ou une liste d'entiers (`int list`) en fonction des questions. La première étape consiste à calculer le nombre d'occurrences $q(\sigma)$ de chaque lettre σ dans un mot.

Question I.1. Supposons $\mathcal{S} = [0, 255]$. Définissez une fonction OCaml `occurrences : int list -> int array` qui reçoit en argument un mot représenté par une liste d'entiers et renvoie le nombre d'occurrences $q(\sigma)$ de chaque lettre $\sigma \in \mathcal{S}$ sous forme d'un tableau $\llbracket q(0); q(1); \dots; q(255) \rrbracket$.

Il peut être intéressant de ne considérer que le sous-ensemble des lettres qui ont au moins une occurrence. Dans la question suivante, on ne représentera pas q par $\llbracket q(0); q(1); \dots; q(|\mathcal{S}| - 1) \rrbracket$ mais par un tableau de paires $\llbracket (\sigma_1, q(\sigma_1)); (\sigma_2, q(\sigma_2)); \dots \rrbracket$ avec $\{\sigma_1, \sigma_2, \dots\} = \{\sigma \in \mathcal{S} \mid q(\sigma) \neq 0\}$ et $\sigma_1 < \sigma_2 < \dots$

Question I.2. Définissez une fonction OCaml `nonzero_occurrences : int array -> (int * int) array` qui passe de la représentation de q de la question **I.1** (tableau $\llbracket q(0); q(1); \dots; q(|\mathcal{S}| - 1) \rrbracket$) à la représentation sous forme d'un tableau de paires. Cette fonction devra avoir une complexité temporelle en $O(|\mathcal{S}|)$.

Partie II. Arbres binaires

Soit \mathcal{T}_S l'ensemble des arbres binaires dont les feuilles sont en bijection avec un ensemble fini S . Autrement dit, un arbre de \mathcal{T}_S a exactement $|S|$ feuilles ; chacune de ses feuilles est étiquetée par un élément de S ; toutes les feuilles sont étiquetées par des éléments différents de S . Ces arbres peuvent être construits comme suit :

- Une feuille étiquetée par x est notée $F(x)$. C'est un arbre de $\mathcal{T}_{\{x\}}$.
- Si g et d sont des arbres appartenant respectivement à \mathcal{T}_{S_1} et \mathcal{T}_{S_2} , alors l'arbre dont le sous-arbre gauche est g et le sous-arbre droit est d , noté $N(g, d)$, est un arbre de $\mathcal{T}_{S_1 \cup S_2}$ à condition que $S_1 \cap S_2 = \emptyset$.

Remarque : un tel arbre binaire possède exactement $|S| - 1$ noeuds internes. Par ailleurs, on étiquettera implicitement les arêtes par 0 ou 1 suivant qu'elles mènent à un sous-arbre gauche (0) ou droit (1).

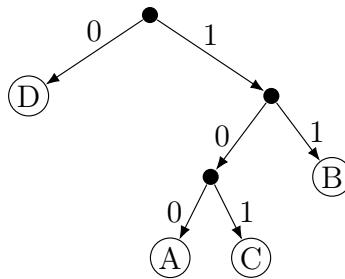


FIGURE 1 – Exemple d'arbre de $\mathcal{T}_{\{A,B,C,D\}}$.

Question II.1. Montrez que l'ensemble $\mathcal{T}_{\{A,B,C,D\}}$ contient 120 arbres.

Le type OCaml utilisé pour représenter les arbres est le suivant :

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

Étant donnés un arbre t de \mathcal{T}_S et un élément σ de S , on note $\ell_t(\sigma)$ la profondeur de la feuille étiquetée par σ , c'est-à-dire le nombre d'arêtes entre la racine de t et cette feuille. Par exemple, avec l'arbre de la figure 1, on a $\ell_t(A) = 3$.

Comme précédemment, soit q une fonction de S dans \mathbb{N} . Pour tout arbre $t \in \mathcal{T}_{S'}$ avec $S' \subseteq S$, on note $c_q(t)$ la somme pondérée suivante :

$$c_q(t) = \sum_{\sigma \in S'} q(\sigma) \ell_t(\sigma).$$

Question II.2. Supposons $S = [0, n - 1]$. Définissez une fonction OCaml `cq : tree -> int array -> int` qui reçoit deux arguments, un arbre $t \in \mathcal{T}_S$ et un tableau $\llbracket q(0); q(1); \dots; q(n-1) \rrbracket$ représentant q , et qui renvoie la valeur de $c_q(t)$. On cherchera à écrire une fonction efficace. Donnez et justifiez sa complexité temporelle.

Étant donné un arbre t de \mathcal{T}_S , on représente une lettre $\sigma \in S$ par la séquence des bits obtenus en parcourant t de la racine vers la feuille $F(\sigma)$. Par exemple, dans l'arbre de $\mathcal{T}_{\{A,B,C,D\}}$ de la figure 1, D est représenté par 0 tandis que A est représenté par 100.

Question II.3. Définissez une fonction OCaml `get_path : int -> tree -> int list` qui reçoit en argument un entier $\sigma \in \mathcal{S}$ et un arbre $t \in \mathcal{T}_{\mathcal{S}}$ et qui renvoie la liste de 0 et 1 représentant σ dans t . Donnez et justifiez la complexité temporelle de cette fonction.

Considérons les séquences de bits définies de la façon suivante. Elles commencent par $k \geq 0$ bits valant 1, suivis d'un bit à 0, suivis de k bits arbitraires, ce que l'on notera $1^k 0 (0 \mid 1)^k$. Les dix premières séquences par ordre lexicographique sont donc 0, 100, 101, 11000, 11001, 11010, 11011, 1110000, 1110001, 1110010. L'arbre dont les branches constituent ces séquences et dont les feuilles sont étiquetées de gauche à droite par des entiers croissants commence comme illustré sur la figure 2.

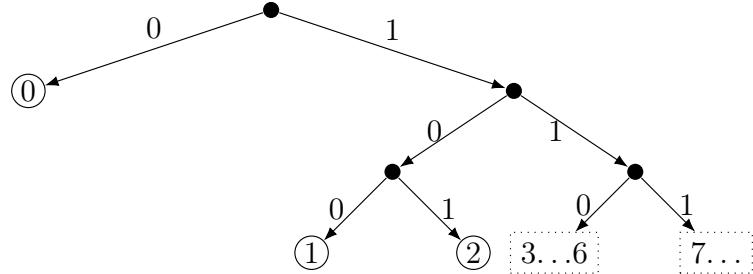


FIGURE 2 – Arbre des séquences $1^k 0(0 \mid 1)^k$.

Remarque : l'étiquette de chaque feuille correspond à l'indice de la séquence quand elles sont triées par ordre lexicographique. Les séquences croissant indéfiniment, l'arbre est *a priori* infini. C'est pourquoi la question suivante se limite aux séquences de bits $1^k 0(0 \mid 1)^k$ pour lesquelles k n'excède pas une certaine borne ℓ , afin d'obtenir un arbre de profondeur bornée $2\ell + 1$.

Question II.4. Définissez une fonction OCaml `integers : int -> tree` qui prend un entier $\ell \geq 0$ en argument et renvoie l’arbre dont les branches sont les séquences de la forme $1^k 0(0 \mid 1)^k$ avec $k \leq \ell$, ainsi que la séquence $1^{\ell+1}$. Les feuilles seront étiquetées de gauche à droite par les entiers 0, 1, 2, etc.

Partie III. Codes préfixes

Étant donnés un alphabet \mathcal{S} et un arbre t de $\mathcal{T}_{\mathcal{S}}$, un mot de \mathcal{S}^* peut être représenté par la concaténation des séquences de bits représentant chacune de ses lettres dans t , de la gauche vers la droite. Supposons que l'arbre t est l'exemple de la figure 1. Le mot « ADBDCD » est alors représenté par la séquence de bits 100|0|11|0|101|0. Remarque : les barres verticales ne servent qu'à rendre l'exemple plus lisible ; elles ne font pas partie de la séquence de bits et ne sont pas nécessaires pour retrouver le mot initial.

Question III.1. Soit t un arbre de $\mathcal{T}_{\mathcal{S}}$. Étant donnée une séquence de bits, montrez qu'il existe au plus un mot de \mathcal{S}^* qui est représenté par cette séquence.

Question III.2. Supposons $\mathcal{S} \subseteq \mathbb{N}$. Définissez une fonction OCaml `decomp1 : int list -> tree -> int list` qui reçoit en argument une liste d'entiers 0 ou 1 et un arbre de $\mathcal{T}_{\mathcal{S}}$ et qui renvoie la liste des éléments de \mathcal{S} représentée par cette liste de bits. On sera attentif au cas où la liste en argument ne représente aucun mot de \mathcal{S}^* . Donnez et justifiez la complexité temporelle de cette fonction.

Cette fonction est dite de *décompression*. Supposons que \mathcal{S} est $\{A, B, C, D\}$ et que la fonction q donne le nombre d'occurrences de chaque lettre dans le mot « ADBDCD », par exemple $q(D) = 3$. Dans ce cas, $c_q(t)$ vaut 11 pour l'arbre exemple de la Figure 1. Il s'agit de la longueur de la séquence 10001101010 représentant le mot « ADBDCD ». Plus généralement, $c_q(t)$ est la longueur de n'importe quelle séquence représentant un mot de \mathcal{S}^q compressé avec l'arbre t . Il s'avère qu'il n'existe aucun arbre $t' \in \mathcal{T}_{\mathcal{S}}$ tel que $c_q(t') < 11$; t est donc optimal.

Étant donnée une fonction q , l'objectif de la partie suivante sera de construire un des arbres de $\mathcal{T}_{\mathcal{S}}$ offrant la meilleure *compression* des mots de \mathcal{S}^q , c'est-à-dire un arbre qui minimise c_q .

Partie IV. Arbres optimaux

Soit q une fonction dont le domaine inclut \mathcal{S} . Un arbre de $\mathcal{T}_{\mathcal{S}}$ est dit *optimal* pour (q, \mathcal{S}) s'il minimise c_q parmi tous les arbres de $\mathcal{T}_{\mathcal{S}}$. Autrement dit, un arbre optimal $t \in \mathcal{T}_{\mathcal{S}}$ vérifie $c_q(t) = \min_{t' \in \mathcal{T}_{\mathcal{S}}} c_q(t')$. De tels arbres optimaux existent et, dès lors que $|\mathcal{S}| \geq 2$, ils ne sont pas uniques.

Question IV.1. Soit $t = N(g, d) \in \mathcal{T}_{\mathcal{S}}$ un arbre optimal pour (q, \mathcal{S}) . Soit \mathcal{S}_g l'ensemble des lettres qui étiquettent les feuilles du sous-arbre g . Montrez que g est un arbre de $\mathcal{T}_{\mathcal{S}_g}$ optimal pour (q, \mathcal{S}_g) .

Malheureusement, cette propriété ne permet pas d'en déduire un algorithme de type « diviser pour régner » pour trouver l'arbre optimal. En effet, il faudrait que l'algorithme puisse efficacement deviner comment partitionner \mathcal{S} entre les feuilles du sous-arbre gauche et celles du sous-arbre droit.

Question IV.2. Supposons qu'il existe une lettre $\sigma_0 \in \mathcal{S}$ telle que $q(\sigma_0) > \sum_{\sigma \in \mathcal{S} \setminus \{\sigma_0\}} q(\sigma)$. Montrez que, pour tout arbre de $\mathcal{T}_{\mathcal{S}}$ optimal pour (q, \mathcal{S}) , la feuille $F(\sigma_0)$ est à profondeur 1, c'est-à-dire directement attachée à la racine.

Question IV.3. Soient σ_1 et σ_2 deux éléments différents de \mathcal{S} tels que, pour tout $\sigma \in \mathcal{S} \setminus \{\sigma_1, \sigma_2\}$, on a $q(\sigma) \geq q(\sigma_1)$ et $q(\sigma) \geq q(\sigma_2)$. Soit $\mathcal{S}' = \mathcal{S} \setminus \{\sigma_1, \sigma_2\} \cup \{\sigma_3\}$ avec σ_3 un tout nouvel élément. On définit q' de telle sorte que $q'(\sigma_3) = q(\sigma_1) + q(\sigma_2)$ et $\forall \sigma \in \mathcal{S} \setminus \{\sigma_1, \sigma_2\}$, $q'(\sigma) = q(\sigma)$.

1. Montrez qu'il existe un arbre t de $\mathcal{T}_{\mathcal{S}}$ optimal pour (q, \mathcal{S}) ayant un nœud $N(F(\sigma_1), F(\sigma_2))$.
2. Soit un arbre t' de $\mathcal{T}_{\mathcal{S}'}$ optimal pour (q', \mathcal{S}') . Montrez que l'arbre $t \in \mathcal{T}_{\mathcal{S}}$ obtenu en remplaçant dans t' la feuille $F(\sigma_3)$ par $N(F(\sigma_1), F(\sigma_2))$ est optimal pour (q, \mathcal{S}) .

On définit la fonction \bar{q} en étendant la fonction q aux arbres de la façon suivante. Pour une feuille, $\bar{q}(F(\sigma))$ vaut $q(\sigma)$. Pour un nœud interne, $\bar{q}(N(g, d))$ vaut récursivement $\bar{q}(g) + \bar{q}(d)$. Remarque : en général, $c_q(t) \neq \bar{q}(t)$.

La question précédente montre qu'il est possible de construire un arbre optimal pour (q, \mathcal{S}) à l'aide de l'algorithme suivant. On initialise un ensemble d'arbres E avec toutes les feuilles $F(\sigma)$ avec $\sigma \in \mathcal{S}$. À chaque étape, on retire de E deux arbres t_1 et t_2 qui minimisent \bar{q} ; puis on ajoute à E l'arbre $N(t_1, t_2)$. On répète cette procédure jusqu'à ce qu'il ne reste qu'un seul arbre dans E . Il s'agit alors d'un arbre optimal pour (q, \mathcal{S}) .

Question IV.4. Implantez l'algorithme décrit ci-dessus en définissant les deux fonctions OCaml suivantes : `insert` et `optimal`.

1. La fonction `insert` : `(int * tree) -> (int * tree) list -> (int * tree) list` insère dans une liste son premier argument. La liste en argument est supposée triée par rapport à la première composante de chacune de ses paires. La liste renvoyée doit l'être aussi.
2. La fonction `optimal` : `(int * int) list -> tree` renvoie un arbre optimal. La liste passée en argument est de taille $|\mathcal{S}|$; elle contient toutes les paires $(\sigma, q(\sigma))$ pour $\sigma \in \mathcal{S}$; ces paires sont triées par valeur croissante de $q(\sigma)$.
3. Donnez et justifiez la complexité temporelle de la fonction `optimal`.

Les deux questions suivantes montrent que, quand la liste initiale des $(\sigma, q(\sigma))$ est triée par valeur croissante de $q(\sigma)$, de simples listes et/ou tableaux suffisent pour écrire une fonction `optimal` dont la complexité temporelle est en $O(|\mathcal{S}|)$.

Question IV.5. Montrez que, lors de l'exécution de l'algorithme décrit ci-dessus, les arbres $t = N(t_1, t_2)$ sont ajoutés à l'ensemble E par valeur croissante de $\bar{q}(t)$.

Question IV.6. Expliquez comment écrire la fonction `optimal` pour que sa complexité temporelle soit linéaire. Le code OCaml n'est pas demandé.

Cette partie a permis de calculer un arbre qui minimise la longueur c_q de la séquence de bits représentant un mot de \mathcal{S}^q . Mais pour décompresser ce mot, il faut disposer de l'arbre et donc l'avoir stocké quelque part. Parmi tous les arbres optimaux, il est donc important d'en choisir un qui nécessitera le moins de bits pour être représenté ; c'est l'objet de la partie suivante.

Partie V. Arbres canoniques

On suppose maintenant qu'il existe un ordre alphabétique noté $<$ sur les éléments de \mathcal{S} . Pour $\mathcal{S} \subseteq \mathbb{N}$, il s'agira de l'ordre usuel sur \mathbb{N} . Soit t un arbre de $\mathcal{T}_{\mathcal{S}}$. On définit la relation \prec_t entre éléments de \mathcal{S} de la façon suivante : pour toute paire $(\sigma_1, \sigma_2) \in \mathcal{S}^2$,

$$\sigma_1 \prec_t \sigma_2 \Leftrightarrow \ell_t(\sigma_1) < \ell_t(\sigma_2) \text{ ou } (\ell_t(\sigma_1) = \ell_t(\sigma_2) \text{ et } \sigma_1 < \sigma_2).$$

L'arbre t est dit *canonique* si, en parcourant les feuilles de la gauche vers la droite, leurs étiquettes respectent la relation \prec_t . Autrement dit, plus une feuille est à gauche, plus elle est proche de la racine ; et les étiquettes des feuilles à une même profondeur sont triées de gauche à droite par ordre alphabétique.

L'arbre de la figure 1 vérifie bien la deuxième partie de la propriété : les feuilles à une même profondeur sont triées de gauche à droite par ordre alphabétique. Mais il ne respecte pas la première partie : la feuille étiquetée par A est plus profonde que celle étiquetée par B alors qu'elle est plus à gauche. Cet arbre n'est donc pas canonique. L'arbre suivant est par contre canonique :

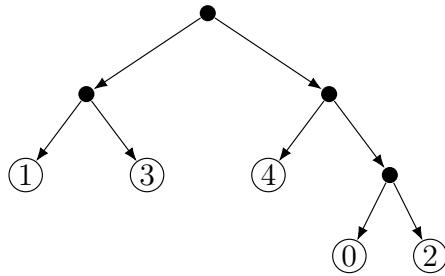


FIGURE 3 – Exemple d'arbre canonique.

Un arbre canonique peut être représenté par deux tableaux d'entiers. Le premier tableau associe à chaque indice i le nombre de lettres $\sigma \in \mathcal{S}$ telles que $\ell_t(\sigma) = i$. Le deuxième tableau contient les éléments de \mathcal{S} obtenus en parcourant les feuilles de l'arbre de gauche à droite. Ainsi, l'arbre de $\mathcal{T}_{[0,4]}$ de la figure 3 est représenté par les deux tableaux $\llbracket 0; 0; 3; 2 \rrbracket$ et $\llbracket 1; 3; 4; 0; 2 \rrbracket$.

Question V.1. Définissez une fonction OCaml `canonical : int array -> int array -> tree` qui, étant donnés deux tels tableaux, renvoie l'arbre canonique qu'ils représentent, s'il existe. Donnez et justifiez la complexité temporelle de cette fonction.

Partie VI. Arbres alphabétiques

L’arbre de la figure 1 était optimal pour le mot « ADBDCD ». Un autre arbre optimal pour ce mot est le suivant :

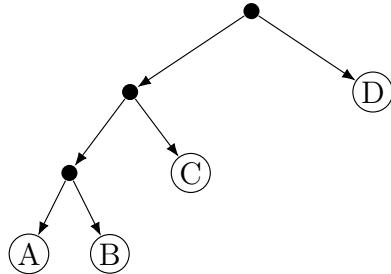


FIGURE 4 – Arbre à la fois optimal pour le mot « ADBDCD » et alphabétique.

Il n’est pas canonique mais il a une propriété très intéressante : ses feuilles sont dans l’ordre alphabétique. Cela signifie que lors du stockage de l’arbre, il n’est pas nécessaire de stocker les étiquettes des feuilles, il suffit de stocker la forme de l’arbre. Un tel arbre est dit *alphabétique*.

Plus généralement, un arbre de \mathcal{T}_S est *alphabétique* si, en parcourant ses feuilles de gauche à droite, les étiquettes apparaissent dans l’ordre alphabétique. On note \mathcal{A}_S l’ensemble de ces arbres. On dit d’un arbre qu’il est *alphabétique-optimal* pour (q, S) s’il minimise c_q parmi tous les arbres de \mathcal{A}_S . Remarque : un arbre alphabétique-optimal pour (q, S) n’est pas nécessairement optimal pour (q, S) .

Soit $n \in \mathbb{N}$ et $S = [0, n - 1]$. On se donne une fonction $q : S \rightarrow \mathbb{N}$. Étant donnés deux entiers i et j tels que $0 \leq i \leq j < n$, on note $m_{i,j}$ la valeur de $c_q(t)$ pour n’importe quel arbre t de $\mathcal{A}_{[i,j]}$ alphabétique-optimal pour $(q, [i, j])$. En particulier, si t est un arbre de \mathcal{A}_S alphabétique-optimal pour (q, S) , il vérifie $c_q(t) = m_{0,n-1}$.

Question VI.1. Exprimez la valeur de $m_{i,j}$ en fonction des valeurs de $m_{i',j'}$ avec $[i', j'] \subsetneq [i, j]$. Déduisez en une fonction OCaml `alpha_optimal : int array -> tree` qui calcule un arbre de $\mathcal{A}_{[0,n-1]}$ alphabétique-optimal pour $(q, [0, n - 1])$. La fonction q est donnée par le tableau de taille n passé en argument. La complexité temporelle de la fonction devra être en $O(n^3)$.

Un arbre alphabétique de $\mathcal{A}_{[0,n-1]}$ peut être représenté par la séquence de $2n - 1$ bits obtenue en effectuant un parcours en profondeur préfixe, c’est-à-dire que la racine d’un sous-arbre est visité avant ses feuilles et qu’un sous-arbre gauche est parcouru avant un sous-arbre droit. Pour chaque nœud, les chiffres 0 et 1 indiquent respectivement s’il s’agit d’un nœud interne ou d’une feuille. Par exemple, l’arbre de la figure 4 est représenté par la séquence 0001111.

Question VI.2. Définissez une fonction OCaml `alpha : int array -> tree` qui reçoit en argument un tableau de 0 et 1 et renvoie l’arbre alphabétique correspondant. On sera attentif au cas où le tableau en argument ne représente aucun arbre. Cette fonction devra avoir une complexité temporelle en $O(n)$ avec n la taille du tableau.

Partie VII. Codes arithmétiques

Soient un alphabet \mathcal{S} et une fonction $q : \mathcal{S} \rightarrow \mathbb{N}$. Comme précédemment, \mathcal{S}^q est le sous-ensemble des mots de \mathcal{S}^* qui contiennent exactement $q(\sigma)$ occurrences de σ pour chaque $\sigma \in \mathcal{S}$.

Question VII.1. Supposons que \mathcal{S} est $\{A, B, C\}$ et que q satisfait $q(A) = 15$, $q(B) = 4$, $q(C) = 1$.

1. Combien de bits faut-il pour représenter un mot de \mathcal{S}^q en utilisant un arbre optimal pour (q, \mathcal{S}) ?
2. Combien y a-t-il de mots dans \mathcal{S}^q ? On pourra exprimer ce nombre sous forme d'un produit de nombres premiers.
3. Montrez que 17 bits suffisent pour représenter chaque entier entre 0 et $|\mathcal{S}^q| - 1$ (et donc n'importe quel mot de \mathcal{S}^q). Remarque : $15 \cdot 17 = 2^8 - 1$.

L'exemple de la question précédente montre que, dans certains cas, par exemple quand une lettre est bien plus fréquente que les autres, l'utilisation d'un code préfixe n'est pas l'approche optimale. Il faut alors se tourner vers d'autres mécanismes de compression.

On se restreint maintenant au cas où $\mathcal{S} = [0, n - 1]$. Soit $N = \sum_{\sigma \in \mathcal{S}} q(\sigma)$. La fonction $E_q : \mathbb{N} \times \mathcal{S} \rightarrow \mathbb{N}$ est définie de la façon suivante pour $q(\sigma) \neq 0$:

$$E_q(x, \sigma) = \lfloor x/q(\sigma) \rfloor \cdot N + (x \bmod q(\sigma)) + \sum_{0 \leq k < \sigma} q(k),$$

où $\lfloor a \rfloor$ désigne la partie entière de a .

La fonction E_q peut être étendue en une fonction $C_q : \mathbb{N} \times \mathcal{S}^* \rightarrow \mathbb{N}$ qui travaille sur les mots de la façon suivante :

$$C_q(x, \sigma_0 \sigma_1 \dots \sigma_k) = E_q(\dots E_q(E_q(x, \sigma_0), \sigma_1), \dots \sigma_k).$$

Ainsi, après avoir choisi x arbitrairement, un mot $\sigma_0 \dots \sigma_{N-1} \in \mathcal{S}^q$ peut être compressé en un entier $y = C_q(x, \sigma_0 \dots \sigma_{N-1})$. Il est alors possible de retrouver le mot original à partir de y en calculant progressivement σ_{N-1} , σ_{N-2} , etc, jusqu'à σ_0 .

Considérons par exemple le mot « 0120000 » ($N = 7$, $q = [[5, 1, 1]]$). Si l'on part de $x = 0$, sa version compressée sera l'entier 151 :

$$0 \xrightarrow{0} 0 \xrightarrow{1} 5 \xrightarrow{2} 41 \xrightarrow{0} 57 \xrightarrow{0} 79 \xrightarrow{0} 109 \xrightarrow{0} 151$$

où $x \xrightarrow{\sigma} y$ signifie $E_q(x, \sigma) = y$.

Question VII.2. Définissez une fonction OCaml `decomp2 : int array -> int -> int * int array` qui reçoit en argument un tableau représentant q et deux entiers y et k , et renvoie un entier x et un tableau $\llbracket \sigma_0, \sigma_1, \dots, \sigma_{k-1} \rrbracket$ tels que $E_q(x, \sigma_0 \sigma_1 \dots \sigma_{k-1}) = y$.

Le nombre de bits de l'entier $C_q(x, \sigma_0 \dots \sigma_{N-1})$ est proche de l'optimum théorique $\sum_{\sigma \in S} q(\sigma) \cdot \log_2(N/q(\sigma))$ mais un problème se pose en pratique. Sauf pour de tout petits mots sur de tout petits alphabets, le calcul de $C_q(x, \sigma_0 \dots \sigma_{N-1})$ va nécessiter de manipuler des entiers très grands. Pour éviter ce problème, une solution consiste, avant chaque appel à E_q , à se débarrasser d'un certain nombre k_i de bits de poids faible. Plus précisément, on construit les suites (x_n) et (y_n) suivantes :

$$\begin{cases} y_i &= \lfloor x_i / 2^{k_i} \rfloor \\ x_{i+1} &= E_q(y_i, \sigma_i) \end{cases}$$

Comme précédemment, la valeur de x_0 est fixée arbitrairement. Le mot $\sigma_0 \dots \sigma_{N-1}$ peut alors être représenté par d'une part x_N et d'autre part la concaténation des séquences de bits suivantes, les bits de poids forts apparaissant en premier :

$$\underbrace{x_{N-1} \bmod 2^{k_{N-1}}; \dots; \underbrace{x_2 \bmod 2^{k_2}}_{k_2 \text{ bits}}; \underbrace{x_1 \bmod 2^{k_1}}_{k_1 \text{ bits}}}.$$

Pour chaque i en ordre décroissant, le décompresseur déduit de x_{i+1} les valeurs de y_i et σ_i (par exemple avec `decomp2`, question VII.2, avec $k = 1$), puis il lit k_i bits dans le mot compressé et les concatène à y_i pour obtenir x_i , et ainsi de suite jusqu'à avoir décompressé tout le mot.

Dans la suite, on supposera que N est une puissance de deux : $N = 2^K$. On supposera par ailleurs que le processeur n'est efficace que pour des entiers ne dépassant pas 2^B pour un certain B bien plus grand que K . En particulier, $x_{i+1} = E_q(y_i, \sigma_i)$ ne doit jamais atteindre cette borne 2^B lors de la compression.

Pour que le taux de compression se rapproche de l'optimum théorique, il faut que chaque k_i soit le plus petit possible (idéalement 0) et donc que chaque y_i soit le plus grand possible. Par ailleurs, les différents k_i ne font pas partie du mot compressé. Autrement dit, en ne connaissant que y_i , le décompresseur doit pouvoir deviner le nombre k_i de bits qui doivent être lus dans le mot compressé pour reconstruire x_i . Une solution correcte mais mauvaise serait, par exemple, de fixer tous les k_i à la constante K ; le décompresseur pourrait alors deviner trivialement les k_i mais le mot résultant ne serait absolument pas compressé.

Question VII.3. Proposez une façon pour le compresseur de choisir k_i en fonction de x_i et σ_i . Expliquez comment le décompresseur choisit k_i en fonction de y_i et prouvez que ce choix correspond bien à celui du compresseur. Si cela a une importance, expliquez comment le compresseur doit choisir x_0 .

Remarque : la contrainte imposant que N soit une puissance de deux ne pose aucune difficulté en pratique. En effet, en matière de compression entropique, la seule chose qui importe est que $q(\sigma)/N$ soit proche de la proportion de la lettre σ dans le mot à compresser.

Fin du sujet.

ECOLES NORMALES SUPERIEURES

CONCOURS D'ADMISSION 2023

**VENDREDI 21 AVRIL 2023
14h00 - 18h00**

FILIERES MP et MPI

Epreuve n° 10

INFO-FONDAMENTALE (ULSR)

Durée : 4 heures

*L'utilisation des calculatrices n'est pas
autorisée pour cette épreuve*

Épreuve d'informatique fondamentale

Concision et ambiguïté

Le sujet porte sur les automates finis, dont la définition est rappelée dans le préambule. Le sujet s'intéresse à la propriété d'ambiguïté des automates finis, propriété également définie dans le préambule. La première partie lie les notions de déterminisme et d'ambiguïté d'un automate fini en utilisant la notion de miroir d'un automate. La deuxième partie amène à définir un algorithme qui teste si un automate est ambigu et vous demande de déterminer sa complexité asymptotique. La troisième et la quatrième partie étudient la concision des automates non ambigus et ambigus.

Les parties 1 et 2 sont indépendantes ; il est conseillé de les traiter en premier car les parties 3 et 4 en dépendent. Il est permis d'admettre les réponses à certaines questions pour répondre aux suivantes.

Préambule

On note Σ un *alphabet* fini, c'est à dire un ensemble fini de symboles appelés *lettres*. Un *mot* sur Σ est une suite finie de lettres. On note Σ^n l'ensemble des mots de longueur n sur Σ et Σ^* l'ensemble de tous les mots sur Σ . Soient u, v deux mots sur Σ , on note $u \cdot v$ la *concaténation* de u et v . Un *langage* sur Σ est un sous-ensemble de Σ^* .

Un automate sur Σ est un tuple $\mathcal{A} = (Q, T, I, F)$ où :

- Q est un ensemble fini de symboles appelés *états* ;
- $T \subseteq Q \times \Sigma \times Q$ est appelé ensemble des *transitions* ;
- $I \subseteq Q$ est l'ensemble des états *initiaux* ;
- $F \subseteq Q$ est l'ensemble des états *finaux*.

La Figure 1 représente graphiquement trois automates. Les symboles dans Q sont encerclés, avec deux cercles pour les symboles dans F . Une transition $(q, a, q') \in T$ est représentée par une flèche étiquetée par $a \in \Sigma$, allant de l'état source q à l'état destination q' . Les états initiaux sont indiqués par une flèche sans état source.

Un calcul de \mathcal{A} sur un mot $w = a_0 \dots a_{n-1} \in \Sigma^*$ est une suite finie d'états $q_0 \dots q_n \in Q^*$ telle que $q_0 \in I$ et pour tout $i < n$, $(q_i, a_i, q_{i+1}) \in T$. Un tel calcul est dit *acceptant* si $q_n \in F$. On dit alors que \mathcal{A} accepte w . Le langage de \mathcal{A} , noté $L(\mathcal{A})$, est l'ensemble des mots acceptés par \mathcal{A} .

Un automate \mathcal{A} est dit *déterministe* si $|I| \leq 1$ et pour tout $(q, a) \in Q \times \Sigma$, $|\{q' \mid (q, a, q') \in T\}| \leq 1$.

Un automate \mathcal{A} est dit *complet* si $|I| \geq 1$ et pour tout $(q, a) \in Q \times \Sigma$, $|\{q' \mid (q, a, q') \in T\}| \geq 1$.

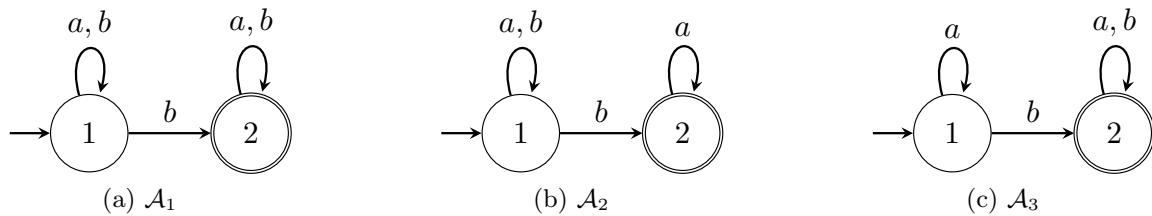


FIGURE 1 – Trois automates reconnaissant le même langage

Soient $w \in \Sigma^*$ et \mathcal{A} un automate. Le mot w est dit *ambigu* pour \mathcal{A} s'il existe deux calculs acceptants ρ et ρ' de \mathcal{A} sur w avec $\rho \neq \rho'$. On définit $d_{\mathcal{A}}(w)$, le *degré d'ambiguïté* de w dans \mathcal{A} , comme étant le nombre de calculs acceptants différents de \mathcal{A} sur w . Ainsi, w est ambigu pour \mathcal{A} si et seulement si $d_{\mathcal{A}}(w) > 1$. On note $\text{Amb}(\mathcal{A})$ l'ensemble des mots ambigus pour \mathcal{A} . L'automate \mathcal{A} est dit *ambigu* si $\text{Amb}(\mathcal{A}) \neq \emptyset$.

Ce sujet s'intéresse tout particulièrement aux automates *non ambigu*s.

1 Déterminisme et ambiguïté

Question 1.1 Les trois automates de la Figure 1 acceptent le même langage. Donnez-en, sans justification, une description intuitive.

Question 1.2 Pour chacun des automates de la Figure 1, dites s'il est déterministe ou non déterministe. Justifiez vos affirmations.

Question 1.3 Pour l'automate \mathcal{A}_1 de la Figure 1, calculez $\text{Amb}(\mathcal{A}_1)$ et déduisez en si \mathcal{A}_1 est ambigu ou non. Faites de même pour les automates \mathcal{A}_2 et \mathcal{A}_3 .

Soit \mathcal{A} un automate. On note $\tilde{\mathcal{A}} = (\tilde{Q}, \tilde{T}, \tilde{I}, \tilde{F})$ l'*automate miroir* de \mathcal{A} , défini par :

- $\tilde{Q} = Q$;
- $\tilde{I} = F$;
- $\tilde{F} = I$;
- $\tilde{T} = \{(t, a, s) \mid (s, a, t) \in T\}$.

Un automate est dit *co-déterministe* si son automate miroir est déterministe.

Question 1.4 Soit \mathcal{A} un automate, $w \in L(\mathcal{A})$ un mot accepté par \mathcal{A} et $q_0 \dots q_n$ un calcul acceptant de \mathcal{A} sur w . Montrez qu'il existe un mot \tilde{w} tel que $q_n \dots q_0$ soit un calcul acceptant de $\tilde{\mathcal{A}}$ sur \tilde{w} .

Question 1.5 Montrez qu'un automate \mathcal{A} est ambigu si et seulement si $\tilde{\mathcal{A}}$ est ambigu.

Question 1.6 Montrez que si un automate \mathcal{A} est déterministe, alors \mathcal{A} n'est pas ambigu.

Question 1.7 Montrez que si un automate \mathcal{A} est co-déterministe, alors \mathcal{A} n'est pas ambigu.

Question 1.8 Pour chacune des questions suivantes, donnez un automate \mathcal{A} ayant au plus 4 états et respectant les propriétés demandées. Justifiez vos réponses.

- (i) \mathcal{A} est non ambigu mais ni déterministe, ni co-déterministe ;
- (ii) $L(\mathcal{A})$ est infini et $\text{Amb}(\mathcal{A}) = L(\mathcal{A})$;
- (iii) $\text{Amb}(\mathcal{A})$ est infini, et $\text{Amb}(\mathcal{A}) \neq L(\mathcal{A})$.

2 Test d'ambiguïté

Le but de cette partie est d'obtenir un algorithme qui teste si un automate est ambigu et de déterminer sa complexité asymptotique.

Dans cette partie, \mathcal{A} est un automate (Q, T, I, F) tel que $L(\mathcal{A}) \neq \emptyset$.

2.1 Une construction utile

En utilisant \mathcal{A} , on définit l'automate $\widehat{\mathcal{A}} = (\widehat{Q}, \widehat{T}, \widehat{I}, \widehat{F})$ comme suit :

- $\widehat{Q} = Q \times Q \times \{0, 1\}$;
- $\widehat{I} = \{(i, i, 0) \mid i \in I\} \cup \{(i, i', 1) \mid i \in I, i' \in I, i \neq i'\}$;
- $\widehat{F} = \{(f, f', 1) \mid f \in F, f' \in F\}$;
- $\widehat{T} = T_1 \cup T_2 \cup T_3$ avec :
 - $T_1 = \{((s, s, 0), a, (t, t, 0)) \mid (s, a, t) \in T\}$;
 - $T_2 = \{((s, s, 0), a, (t, t', 1)) \mid (s, a, t) \in T, (s, a, t') \in T, t \neq t'\}$;
 - $T_3 = \{((s, s', 1), a, (t, t', 1)) \mid (s, a, t) \in T, (s', a, t') \in T\}$.

Question 2.1 Soit \mathcal{A}_1 le premier automate de la figure Figure 1. Construisez l'automate $\widehat{\mathcal{A}}_1$. Il est inutile de faire figurer les états qui ne sont pas accessibles à partir d'un état initial. Donnez, sans justification, le langage $L(\widehat{\mathcal{A}}_1)$.

Question 2.2 Soient $w \in \Sigma^*$ un mot et $\rho = (q_0, q'_0, b_0) \dots (q_n, q'_n, b_n)$ un calcul de $\widehat{\mathcal{A}}$ sur w . On pose $\mu = q_0 \dots q_n$ et $\mu' = q'_0, \dots, q'_n$.

- (i) Montrez que μ et μ' sont des calculs de \mathcal{A} sur w .
- (ii) Montrez que $b_n = 0$ si et seulement si $\mu = \mu'$.
- (iii) Montrez que, si ρ est acceptant, alors μ et μ' sont acceptants.
- (iv) Montrez qu'il existe un calcul ρ tel que μ et μ' sont acceptants mais ρ ne l'est pas.

Question 2.3 Montrez que $L(\widehat{\mathcal{A}}) = \text{Amb}(\mathcal{A})$.

2.2 L'algorithme

Pour deux fonctions $f, g : \mathbb{N} \rightarrow \mathbb{N}$, on dit que g est une borne asymptotique de f , et on le note par $f \in \mathcal{O}(g)$, s'il existe deux constantes strictement positives $n_0 \in \mathbb{N}$ et $c \in \mathbb{N}$ telles que pour tout $n \geq n_0$, $f(n) \leq c \times g(n)$. Cette définition se généralise naturellement à des fonctions f et g avec plusieurs paramètres.

Question 2.4 Pour chaque ensemble \widehat{Q} , \widehat{T} , \widehat{I} et \widehat{F} , donnez une borne asymptotique au nombre d'éléments qu'il contient en fonction des tailles de Q , T , I et F .

Question 2.5 Donnez un algorithme qui a comme entrée \mathcal{A} et comme sortie $\widehat{\mathcal{A}}$ en précisant :

- (i) quelle structure de données classique (matrice d'adjacence ou liste d'adjacence) est utilisée pour représenter \mathcal{A} et $\widehat{\mathcal{A}}$,
- (ii) une borne asymptotique de la complexité en temps d'exécution de cet algorithme en fonction de la somme des tailles des ensembles Q , T , I et F .

Question 2.6 Décrivez une méthode permettant de tester si \mathcal{A} est ambigu en utilisant l'automate $\widehat{\mathcal{A}}$. Donnez une borne asymptotique de la complexité en temps de cette méthode en fonction de la somme des tailles des ensembles Q , T , I et F .

2.3 Généralisation

Soit k un entier strictement positif.

Question 2.7 Soit $w \in L(\mathcal{A})$ un mot de longueur k .

- (i) Donnez, en fonction de k et $|Q|$, une borne supérieure sur le degré d'ambiguïté de w dans \mathcal{A} .
- (ii) Donnez un automate \mathcal{A} et un mot de longueur k pour lesquels la borne supérieure indiquée ci-dessus est atteinte.

Question 2.8 On pose $\text{Amb}_{\geq k}(\mathcal{A}) = \{w \in L(\mathcal{A}) \mid d_{\mathcal{A}}(w) \geq k\}$. Montrez que $\text{Amb}_{\geq k}(\mathcal{A})$ est régulier.

Question 2.9 On pose $\text{Amb}_k(\mathcal{A}) = \{w \in L(\mathcal{A}) \mid d_{\mathcal{A}}(w) = k\}$. Montrez que $\text{Amb}_k(\mathcal{A})$ est régulier.

3 Concision des automates non ambigus

Le but de cette partie est de prouver que les automates non ambigus peuvent être exponentiellement plus concis que leurs équivalents déterministes et complets.

Dans toute cette partie, on fixe l'alphabet $\Sigma = \{a, b\}$. Pour $n \geq 1$ un entier, on pose $L_n = \{w_1 \cdot b \cdot w_2 \mid w_1, w_2 \in \Sigma^*, |w_2| = n - 1\}$, le langage des mots dont la n -ième lettre en partant de la fin est un b .

Question 3.1 *Donnez un automate non ambigu acceptant L_3 , puis donnez un automate déterministe et complet acceptant L_3 .*

Question 3.2 *Montrez que pour tout $n \geq 1$ entier, il existe un automate non ambigu \mathcal{A} avec $n + 1$ états tel que $L(\mathcal{A}) = L_n$.*

Question 3.3 *Montrez que pour tout $n \geq 1$ entier, il existe un automate déterministe et complet \mathcal{B} avec 2^n états tel que $L(\mathcal{B}) = L_n$.*

On veut à présent prouver que tout automate déterministe et complet acceptant L_n a au moins 2^n états.

Question 3.4 *Soit \mathcal{B} un automate déterministe et complet. Soit $w \in \Sigma^*$. Montrez qu'il existe un unique calcul, non nécessairement acceptant, de \mathcal{B} sur w .*

Soit \mathcal{B} un automate déterministe et complet et $w \in \Sigma^*$. On note q_w l'état atteint par l'unique calcul de \mathcal{B} sur w , c'est-à-dire l'état q_m tel que $q_0 \dots q_m$ est un calcul de \mathcal{B} sur w .

Question 3.5 *Soit $n \in \mathbb{N}$ et \mathcal{B} un automate déterministe et complet reconnaissant L_n . Soient w et w' deux mots de Σ^* de longueur n . Montrez que $q_w = q_{w'}$ si et seulement si $w = w'$.*

Question 3.6 *Soit $n \in \mathbb{N}$. Montrez que tout automate déterministe et complet reconnaissant L_n a au moins 2^n états.*

4 Concision des automates ambigus

Le but de cette partie est de prouver que les automates ambigus peuvent être exponentiellement plus concis que leurs équivalents non ambigus.

Pour $n \in \mathbb{N}$, on pose Σ_n un alphabet à n lettres et K_n le langage des mots sur Σ_n^* dont au moins une lettre apparaît au moins deux fois, c'est-à-dire :

$$K_n = \{w_1 \cdot x \cdot w_2 \cdot x \cdot w_3 \mid w_1, w_2, w_3 \in \Sigma_n^*, x \in \Sigma_n\}$$

Question 4.1 Pour $\Sigma_3 = \{a, b, c\}$,

- (i) donnez un automate acceptant K_3 et ayant au plus 5 états,
- (ii) donnez un automate non ambigu acceptant K_3 et ayant au plus 9 états.

Question 4.2 Montrez que pour tout $n \in \mathbb{N}$, il existe un automate avec au plus $n + 2$ états acceptant K_n .

Question 4.3 Montrez que pour tout $n \in \mathbb{N}$, il existe un automate non ambigu avec au plus $2^n + 1$ états acceptant K_n .

On veut à présent prouver que tout automate non ambigu acceptant K_n a au moins $2^n + 1$ états. Jusqu'à la fin de cette partie, on fixe un entier $n \geq 2$ et un automate non ambigu $\mathcal{A} = (Q, T, I, F)$ acceptant K_n .

Soient u et v deux mots de Σ_n^* . Lorsque $u \cdot v \in K_n$, on note $q_{u,v}$ l'état de \mathcal{A} atteint après avoir lu u lors de l'unique calcul acceptant de \mathcal{A} sur $u \cdot v$. Autrement dit, soit $\ell = |u|$ et $q_0 \dots q_m$ l'unique calcul acceptant de \mathcal{A} sur $u \cdot v$, alors $q_{u,v} = q_\ell$.

Question 4.4 Soient u, u' et v, v' quatre mots de Σ_n^* tels que $u \cdot v \in K_n$ et $u' \cdot v' \in K_n$. On suppose que $q_{u,v} = q_{u',v'}$.

- (i) Montrez que $u \cdot v' \in K_n$ et $u' \cdot v \in K_n$.
- (ii) Montrez que $q_{u,v} = q_{u,v'} = q_{u',v} = q_{u',v'}$.

L'objectif à présent est de prouver que ces contraintes garantissent que \mathcal{A} a au moins 2^n états.

Soit $s = (s_i)$ une suite de mots et α une lettre. On note $s \cdot \alpha$ la suite $(s_i \cdot \alpha)$, c'est-à-dire la suite obtenue en ajoutant α à la fin de chaque mot de s .

On fixe un ordre total $<$ sur Σ_n , et on note $\Sigma_n = \{\alpha_1, \dots, \alpha_n\}$ avec pour tout $1 \leq i < j \leq n$, $\alpha_i < \alpha_j$. On construit une famille s^0, \dots, s^n de suites de mots de Σ_n comme suit :

- s^0 est la suite contenant uniquement ε ;

— Pour $0 < i \leq n$, $s^i = s^{i-1}, s^{i-1} \cdot \alpha_i$, la suite constituée de s^{i-1} , suivie de la suite $s^{i-1} \cdot \alpha_i$.

Finalement, on pose $s = s^n$. Par exemple, pour $n = 2$, $\Sigma_2 = \{a, b\}$ et $a < b$, la suite s contient 4 éléments : $s_0 = \varepsilon$, $s_1 = a$, $s_2 = b$, $s_3 = ab$.

Finalement, on remarque que s contient 2^n éléments et on pose M_n la matrice de dimension $2^n \times 2^n$ à coefficients en \mathbb{Z} définie par :

$$M_n[i, j] = \begin{cases} 1 & \text{si } s_i \text{ et } s_j \text{ ont une lettre en commun} \\ 0 & \text{sinon} \end{cases}$$

Par exemple, M_2 est donnée ci-dessous :

$$M_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

Question 4.5 Montrez que M_n peut s'écrire sous la forme :

$$M_n = \begin{pmatrix} M_{n-1} & M_{n-1} \\ M_{n-1} & \mathbf{1}_{n-1} \end{pmatrix}$$

où, pour $n \geq 1$, $\mathbf{1}_n$ est la matrice carrée de dimension $2^n \times 2^n$ dont tous les coefficients valent 1.

Question 4.6 Soient $i < 2^n$ et $j < 2^n$. Montrez que $M_n[i, j] = 1$ si et seulement si $s_i \cdot s_j \in K_n$.

À chaque état q de \mathcal{A} , on associe le vecteur colonne v_q de dimension 2^n défini par :

$$v_q[i] = \begin{cases} 1 & \text{s'il existe } j < 2^n \text{ tel que } q = q_{s_i, s_j} \\ 0 & \text{sinon} \end{cases}$$

Question 4.7 Montrez que l'ensemble de vecteurs $(v_q)_{q \in Q}$ est une famille génératrice du sous-espace vectoriel de \mathbb{Z}^n engendré par les vecteurs colonnes de M_n .

Indication : Soit u_j la j -ème colonne de M_n . On pourra montrer que u_j est une combinaison linéaire de vecteurs de $(v_q)_{q \in Q}$ en prouvant que :

$$u_j = \sum_{\substack{q \in Q \\ \exists i, q = q_{s_i, s_j}}} v_q$$

Question 4.8 Montrez que M_n est de rang $2^n - 1$.

Question 4.9 En déduire que \mathcal{A} contient au moins $2^n - 1$ états q tels que $v_q \neq 0$.

Question 4.10 Montrez que si $v_q \neq 0$, alors $q \notin I$ et $q \notin F$.

Question 4.11 Conclure que \mathcal{A} a au moins $2^n + 1$ états.

A2023 – INFO MPI I

**ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.**

Concours Mines-Télécom

CONCOURS 2023**PREMIÈRE ÉPREUVE D'INFORMATIQUE**

Durée de l'épreuve : 3 heures

L'usage de la calculatrice et de tout dispositif électronique est interdit.

*Les candidats sont priés de mentionner de façon apparente
sur la première page de la copie :*

INFORMATIQUE I - MPI

L'énoncé de cette épreuve comporte 9 pages de texte.

Cette épreuve concerne uniquement les candidats de la filière MPI.

*Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé,
il le signale sur sa copie et poursuit sa composition en expliquant les raisons
des initiatives qu'il est amené à prendre.*

Les sujets sont la propriété du GIP CCMP. Ils sont publiés sous les termes de la licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Pas de Modification 3.0 France. Tout autre usage est soumis à une autorisation préalable du Concours commun Mines Ponts.



Préliminaires

Présentation du sujet

L'épreuve est composée d'un problème unique comportant 32 questions divisées en trois sections. L'objectif du problème est de construire des *listes à accès direct* : une liste à accès direct est un type de donnée abstrait qui permet, d'une part, d'empiler et de dépiler efficacement un élément en tête de liste et, d'autre part, d'accéder efficacement au k^{e} élément de la liste, pour n'importe quel indice k valide.

Dans la première section (page 1), nous étudions un système de numération : la représentation binaire gauche des entiers naturels. Dans la deuxième section (page 4), nous étudions les arbres binaires parfaits. Dans la troisième section (page 6), nous implémentons le type liste à accès direct par la structure de données concrète *liste gauche*, que nous construisons en exploitant les résultats obtenus dans les sections précédentes.

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

Travail attendu

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat. Des rappels de programmation sont faits en annexe et peuvent être utilisés directement.

Selon les questions, il faudra coder des fonctions à l'aide du langage de programmation C exclusivement, en reprenant le prototype de fonction fourni par le sujet, ou en pseudo-code (c-à-d. dans une syntaxe souple mais conforme aux possibilités offertes par le langage C). Il est inutile de rappeler que les entêtes `<assert.h>`, `<stdbool.h>`, etc. doivent être inclus.

Quand l'énoncé demande de coder une fonction, sauf indication explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites. On suppose que le type `int` n'est jamais sujet à des débordements.

Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle. Lorsqu'une réponse en pseudo-code est permise, seule la logique de programmation est évaluée, même dans le cas où un code en C a été fourni en guise de réponse.

1. Représentation binaire gauche des entiers naturels

1.1. Mise en place

Soient m un entier naturel et N un entier naturel non nul. Il est classique d'appeler *représentation binaire standard de l'entier m sur N chiffres*, ou plus simplement *représentation standard*, toute suite finie $b = (b_n)_{0 \leq n < N}$ de longueur N telle que, pour tout indice n compris entre 0 et $N - 1$, le chiffre b_n appartient à $\{0, 1\}$ et l'égalité suivante est vérifiée

$$m = \sum_{n=0}^{N-1} b_n 2^n.$$

Première épreuve d'informatique MPI 2023

Définition : Nous appelons *représentation binaire gauche de l'entier m sur N chiffres* toute suite finie $g = (g_n)_{0 \leq n < N}$ de longueur N telle que,

- (i) pour tout indice n compris entre 0 et $N - 1$, le chiffre g_n appartient à $\{0, 1, 2\}$,
- (ii) l'égalité suivante est satisfaite

$$m = \sum_{n=0}^{N-1} g_n (2^{n+1} - 1),$$

- (iii) le chiffre « 2 » n'apparaît qu'au plus une fois parmi les chiffres $(g_n)_{0 \leq n < N}$,
- (iv) s'il existe une position p tel que le chiffre g_p est 2, alors, pour tout indice n compris entre 0 et $p - 1$, le chiffre g_n est nul.

De manière plus courte, nous parlons simplement de *représentation gauche*.

La figure 1 ci-dessous donne une représentation standard et une représentation gauche sur quatre chiffres des seize premiers entiers. Conformément à l'usage habituel, nous écrivons toute représentation, qu'elle soit standard ou gauche, sous la forme d'un mot $b_{N-1} \cdots b_0$ ou $g_{N-1} \cdots g_0$ dans lequel les chiffres de poids faibles sont écrits à droite.

Entier	Repr. standard	Repr. gauche	Entier	Repr. standard	Repr. gauche
0	0000	0000	8	1000	0101
1	0001	0001	9	1001	0102
2	0010	0002	10	1010	0110
3	0011	0010	11	1011	0111
4	0100	0011	12	1100	0112
5	0101	0012	13	1101	0120
6	0110	0020	14	1110	0200
7	0111	0100	15	1111	1000

FIGURE 1 – Représentations standard et gauche des seize premiers entiers

- 1 – Soit c un entier. Donner la représentation standard de l'entier dont une représentation gauche est $10 \cdots 0$ (avec c chiffres nuls) en justifiant sommairement. Faire de même avec l'entier dont une représentation gauche est $20 \cdots 0$ (avec c chiffres nuls).
- 2 – Déterminer, en justifiant, le plus grand entier naturel M_N qui admet une représentation gauche sur N chiffres. Préciser la représentation gauche de cet entier.

Définition : Soit n_0 un indice compris entre 0 et $N - 1$. On dit que l'indice n_0 est la *position du chiffre de plus fort poids* d'une représentation $g_{N-1} \cdots g_0$ si l'indice n_0 est le plus petit entier tel que, pour tout indice $n > n_0$, le chiffre g_n est nul. On appelle le chiffre g_{n_0} le *chiffre de plus fort poids*.

- 3 – Soient N un entier naturel non nul, $g = g_{N-1} \cdots g_0$ et $h = h_{N-1} \cdots h_0$ deux représentations gauches d'un même entier m . Démontrer que les chiffres de plus fort poids de g et de h sont de même valeur et à la même position.
- 4 – Démontrer que tout entier appartenant à l'intervalle $\llbracket 0, M_N \rrbracket$, où l'entier M_N a été introduit à la question 2, ne possède au plus qu'une seule représentation gauche sur N chiffres.

Première épreuve d'informatique MPI 2023

Indication C : L'entier N est déclaré comme constante globale. Nous utilisons la structure C déclarée comme suit pour écrire la représentation gauche sur N chiffres d'un entier $g_{N-1} \cdots g_0$:

```

1.  const int N = 8;
2.
3.  struct RepGauche {
4.      int position;
5.      bool chiffres[N];
6.  };
7.  typedef struct RepGauche rg;

```

Le champ `position` repère la position éventuelle du chiffre 2 ; il vaut -1 au cas où le chiffre 2 n'apparaît pas. Pour tout indice n compris entre 0 et $N - 1$, la n^{e} case du champ `chiffres` vaut `true` si le chiffre g_n est non-nul et vaut `false` sinon.

Par exemple, les entiers 15 et 21 ont pour représentation gauche les variables `entier_15` et `entier_21` suivantes :

```

8.  rg entier_15 = { .position = -1,
9.                  .chiffres = { 0, 0, 0, 1, 0, 0, 0, 0 } };
10. rg entier_21 = { .position = 1,
11.                   .chiffres = { 0, 1, 0, 1, 0, 0, 0, 0 } };

```

5 – Formaliser sous la forme d'un ou de plusieurs invariants le fait qu'une valeur C de type `rg` est la représentation gauche d'un entier.

6 – Écrire une fonction C `int rg_to_int(rg g)`, qui renvoie l'entier dont g est la représentation gauche. On supposera que l'invariant de la question 5 est satisfait.

1.2. Incrémentation et décrémentation

Nous proposons l'algorithme suivant :

ALGORITHME MYSTÈRE :

Entrée : Représentation gauche $g = g_{N-1} \cdots g_0$ d'un certain entier m .

Effet :

- Si aucun des chiffres $(g_n)_{0 \leq n < N}$ ne vaut 2, changer le chiffre g_0 en $g_0 + 1$.
- Sinon, en notant p la position du chiffre 2, changer le chiffre g_p en 0 et le chiffre g_{p+1} en $g_{p+1} + 1$.

Nous notons m' l'entier dont la représentation gauche est g après exécution de l'algorithme.

FIGURE 2 – Un algorithme

7 – Vérifier que l'invariant de la question 5 n'est pas rompu par l'algorithme mystère (cf. figure 2). Avec les notations m et m' de la figure 2, caractériser, en fonction de l'entier m , la valeur de l'entier m' .

- 8 – Écrire une fonction C **bool rg_incr(rg *s)** dont la spécification suit :
Précondition : La variable *s* est un pointeur vers la représentation gauche d'un entier *m*.
Effet : La valeur pointée par *s* est modifiée afin de représenter l'entier *m + 1*.
Valeur de retour : Booléen true si l'incrémentation de *m* peut avoir lieu et false si un débordement se produit car *m + 1* n'est pas représentable sur le même nombre de chiffres.
- 9 – Calculer la complexité en temps dans le pire des cas de la fonction rg_incr en fonction de *N*. Comparer avec la complexité de la même opération sur la représentation standard.
- 10 – Écrire une fonction C **bool rg_decr(rg *s)** dont la spécification suit :
Précondition : Le pointeur *s* désigne la représentation gauche d'un entier *m*.
Effet : La valeur pointée par *s* est modifiée afin de représenter l'entier *m - 1*.
Valeur de retour : Booléen true si la décrémentation de *m* peut avoir lieu et false si un débordement se produit car *m* est nul.
 Il est recommandé d'expliquer son intention avant de donner son code.

2. Arbres binaires parfaits

2.1. Opérations sur les arbres binaires

Indication C : Nous représentons des arbres binaires à valeurs entières au moyen du type C **arb** suivant, qui est un pointeur vers une structure contenant la valeur entière dans le champ **valeur** et les deux fils dans les champs **fils_g** et **fils_d**. L'arbre vide se représente par le pointeur **NULL**.

```

12.  typedef struct Noeud *arb;
13.  struct Noeud {
14.      int valeur;
15.      arb fils_g;
16.      arb fils_d;
17.  };

```

L'arbre vide est, par convention, de hauteur -1 .

- 11 – Écrire une fonction C **int hauteur(arb a)** qui calcule la hauteur de l'arbre *a*.
- 12 – Écrire une fonction C **arb noeud(int v, arb ag, arb ad)** qui construit un arbre dont la racine a pour étiquette *v*, dont le fils gauche est *ag* et le fils droit est *ad*. Dans cette question, il est demandé de se défendre, par le truchement d'une assertion, de toute erreur liée à un échec d'allocation dynamique de mémoire.

Première épreuve d'informatique MPI 2023

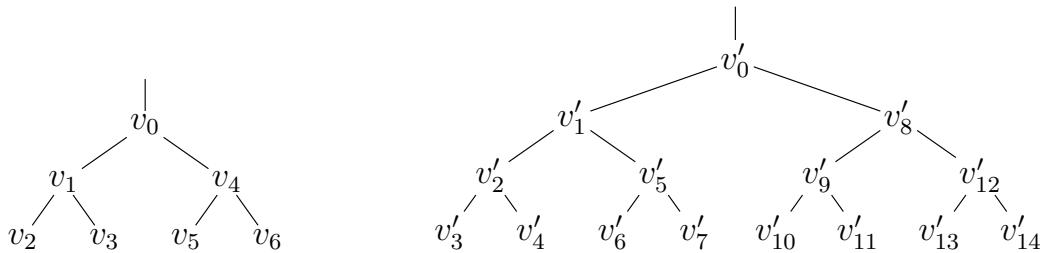


FIGURE 3 – Arbres binaires parfaits de hauteur 2 et de hauteur 3.

2.2. Arbres parfaits

Définition : Un *arbre binaire parfait*, ou simplement *arbre parfait*, est un arbre binaire dont tous les noeuds internes ont exactement deux fils, dont toutes les feuilles sont à la même profondeur et dont les noeuds sont étiquetés par des valeurs entières.

Des exemples d'arbres binaires parfaits sont donnés figure 3.

- 13 – Démontrer que tout arbre binaire parfait de hauteur n possède $2^{n+1} - 1$ noeuds.
- 14 – Écrire une fonction C **bool est_perfait(arb a, int n)** dont la spécification suit :
Précondition : Le pointeur a désigne la racine d'un arbre binaire (dont les noeuds sont bien tous distincts).
Valeur de retour : Booléen true si l'arbre pointé est parfait de hauteur n et false sinon.
- 15 – Calculer la complexité en temps dans le pire des cas de l'exécution de **est_perfait(a, n)** en fonction de l'entier n .

2.3. Opérations sur les arbres parfaits

Définition : Soient S une structure de données qui permet de stocker une collection d'entiers et t le cardinal de S . Nous disons que la structure de données S est à *accès direct* s'il existe une manière systématique de numérotter chaque élément de S entre 0 et $t - 1$ et s'il existe deux primitives, *acces* et *modif*, qui permettent respectivement de consulter et de modifier n'importe quel élément de S en temps logarithmique en t à partir seulement de son numéro.

Nous souhaitons vérifier qu'un arbre parfait est une structure de données à accès direct. Nous numérotions les éléments d'un arbre parfait en utilisant l'ordre préfixe de gauche à droite de l'arbre (comme illustré dans la figure 3).

- 16 – Écrire une fonction C **arb arb_trouve(arb a, int n, int k)** ainsi spécifiée :
Précondition : Le pointeur a désigne la racine d'un arbre binaire parfait de hauteur n . L'entier k satisfait les inégalités $0 \leq k < 2^{n+1} - 1$.
Valeur de retour : Pointeur vers le k^{e} noeud de l'arbre dans l'ordre préfixe.

Il est rappelé que la racine d'un arbre est à profondeur 0. Nous donnons la formule de sommation, valable pour tout réel $a \neq 1$ et tout entier t ,

$$\sum_{k=0}^t k \cdot a^k = \frac{((a-1)t-1) a^{t+1} + a}{(a-1)^2}.$$

- 17 – Nous appelons P la profondeur d'un nœud choisi aléatoirement et uniformément parmi les $2^{n+1} - 1$ nœuds d'un arbre binaire parfait de hauteur n . Calculer l'espérance $\mathbb{E}(P)$ de la variable aléatoire P .
- 18 – Déterminer la complexité moyenne en temps de l'instruction `arb_trouve(a, n, k)` lorsque la hauteur n et l'arbre a sont fixés et le numéro k est choisi aléatoirement et uniformément dans l'intervalle $\llbracket 0, 2^{n+1} - 2 \rrbracket$.
- 19 – Écrire une fonction C `int arb_acces(arb a, int n, int k)` qui renvoie la k^{e} valeur de l'arbre a de hauteur n ainsi qu'une fonction C `void arb_modif(arb a, int n, int k, int v)` qui remplace la k^{e} valeur de l'arbre a de hauteur n par la valeur v . Dire finalement si la structure de données arbre binaire parfait est à accès direct.

3. Listes gauches

Les arbres binaires parfaits seuls ne permettent de représenter que des collections d'entiers dont le cardinal est de la forme $2^{n+1} - 1$ où n est entier naturel. Afin de représenter des collections de cardinal quelconque, nous introduisons des suites d'arbres parfaits, aux hauteurs strictement croissantes et savamment choisies.

Définition : Nous appelons *liste binaire gauche de cardinal m sur N arbres* la structure de données constituée de $N + 1$ arbres binaires parfaits $((a_n)_{0 \leq n < N}, e)$ comme suit. Nous décomposons l'entier m selon sa représentation binaire gauche sur N chiffres : $g_{N-1} \dots g_0$. Pour tout indice n compris entre 0 et $N - 1$, si le chiffre g_n n'est pas nul, l'arbre binaire parfait a_n est un arbre de hauteur n , sinon il s'agit de l'arbre vide. Si le chiffre 2 apparaît parmi les chiffres de la représentation gauche de m et si l'indice p est sa position, alors l'arbre e est un arbre binaire parfait de hauteur p , sinon l'arbre e est l'arbre vide. De manière plus courte, nous parlons simplement de *liste gauche*.

Une liste binaire gauche de cardinal m sur N arbres permet de stocker une collection de m éléments entiers. Il apparaît qu'avec N arbres, une liste binaire gauche peut stocker jusqu'à M_N entiers (où l'entier M_N a été introduit à la question 2) et que m est inférieur à M_N .

Indication C : Nous adoptons le type C suivant

```

18. struct ListeGauche {
19.     int hauteur_e;
20.     arb extra;
21.     int nb_arbres;
22.     arb *arbres;
23. };
24. typedef struct ListeGauche lg;
```

Le champ `hauteur_e` contient la hauteur p de l'arbre exceptionnel e . Le champ `extra` désigne la racine de l'arbre exceptionnel e . Le champ `nb_arbres` contient l'entier N . Enfin, le champ `arbres` désigne un tableau de N pointeurs vers les arbres $(a_n)_{0 \leq n < N}$ qui a été alloué dynamiquement.

3.1. Opérations simples sur les listes gauches

- 20 – Écrire une fonction C **lg** **lg_init**(**int** N) qui renvoie une liste gauche de cardinal nul sur N arbres.
- 21 – Écrire une fonction C **int** **lg_card**(**lg** ℓ) qui calcule le cardinal m de la liste gauche ℓ .

Définition : Afin de numérotter l'ensemble des éléments d'une liste gauche $\ell = ((a_n)_{0 \leq n < N}, e)$, nous parcourons d'abord les éléments de l'arbre exceptionnel e dans l'ordre préfixe, puis nous parcourons les éléments des arbres a_0, a_1, \dots, a_{N-1} dans l'ordre préfixe. Les numéros sont attribués aux valeurs rencontrées par ordre de première rencontre.

- 22 – Écrire une fonction C **arb** **lg_trouve**(**lg** ℓ , **int** k) qui renvoie le k^{e} nœud de la liste gauche ℓ . On supposera que k est un indice valide ($0 \leq k < m$).
- 23 – Calculer la complexité en temps dans le pire des cas de **lg_trouve** en fonction de la capacité maximale M_N de la liste gauche ℓ .
- 24 – Écrire une fonction C **int** **lg_acces**(**lg** ℓ , **int** k) qui renvoie la k^{e} valeur de la liste gauche ℓ ainsi qu'une fonction C **void** **lg_modif**(**lg** ℓ , **int** k , **int** v) qui remplace la k^{e} valeur de la liste gauche ℓ par la valeur v .

3.2. Ajout et suppression en tête de liste gauche

- 25 – Soient v une valeur entière et $\ell = ((a_n)_{0 \leq n < N}, e)$ une liste gauche de cardinal m , avec $m < M_N$, qui contient les éléments v_1, \dots, v_m dans cet ordre. Décrire, en fonction de la liste gauche ℓ , la liste gauche $\ell' = ((a'_n)_{0 \leq n < N}, e')$ de cardinal $m + 1$ dont les éléments sont v, v_1, \dots, v_m dans cet ordre. En déduire le principe d'une fonction C **bool** **lg_empile**(**int** v , **lg** ℓ) réalisant l'insertion de la valeur v en tête de la liste gauche ℓ où le booléen résultat vaut **true** si l'ajout a eu lieu et vaut **false** si un débordement de capacité de la liste se produit. On ne demande pas le code complet de cette fonction.
- 26 – Déterminer la complexité en temps dans le pire des cas de la fonction **lg_empile**.
- 27 – Donner le principe d'une fonction C **bool** **lg_depile**(**int** $*w$, **lg** ℓ) réalisant le retrait de l'élément de tête de la liste gauche ℓ et son affectation à l'adresse w . Le booléen résultat vaut **true** si le retrait a eu lieu et vaut **false** si l'opération a échoué en raison d'une liste vide. On ne demande pas le code complet.
- 28 – Donner la complexité en temps dans le pire des cas de la fonction **lg_depile**.
- 29 – Discuter la possibilité d'obtenir une complexité plus faible à la question 28, quitte à modifier légèrement la définition du type **lg**.

- 30 – Soit N un entier et $\ell = ((a_n)_{0 \leq n < N}, e)$ une liste gauche. Discuter la possibilité de modifier en place et avec une faible complexité en temps la liste gauche ℓ de sorte que le nombre d'arbres N devienne $N + 1$ et que les mêmes éléments demeurent dans la liste.

3.3. Utilisation concurrente des listes gauches

Dans cette sous-section, on raisonne sur les algorithmes en supposant que les fils d'exécution peuvent s'entrelacer mais que les instructions d'un même fil s'exécutent dans l'ordre du programme. L'entête `#include <pthread.h>` a été déclarée ; la syntaxe de certaines fonctions s'y rattachant est rappelée en annexe.

- 31 – Deux fils d'exécution distincts exécutent la fonction `lg_empile` sur la même liste gauche. Montrer qu'une course critique advient de leur exécution concurrente et que la cohérence de ladite liste gauche n'est pas garantie, autrement dit que certains invariants qui caractérisent la bonne formation d'une liste gauche peuvent être violés à l'issue des exécutions.

Nous nous intéressons finalement au *problème des producteurs et des consommateurs* qui s'échangent des entiers au travers d'un tampon, ici constitué par une unique liste gauche ℓ à N arbres, l'entier N étant fixé à l'avance. Les producteurs écrivent dans la liste gauche ℓ en empilant un entier à la fois en tête, à condition que la liste gauche ℓ ne soit pas pleine ; les consommateurs vident la liste gauche ℓ en dépilant l'entier en tête de la liste, à condition que la liste gauche ℓ ne soit pas vide.

Un seul agent peut accéder au tampon à la fois. Lorsqu'un consommateur souhaite supprimer une donnée alors que le tampon est vide, il est mis en attente ; lorsqu'un producteur souhaite écrire une donnée alors que le tampon est plein, il est mis en attente.

- 32 – Décrire une solution au problème des producteurs et des consommateurs en s'appuyant sur un verrou et sur deux sémaphores. Détailler sous la forme de code C ou bien de pseudo-code ce que font les producteurs et les consommateurs.

* * *

*

Les listes binaires gauches, ou *skew binary lists*, ont été inventées par Eugene Myers en 1983.

A. Rappels de programmation en C

L'expression $1 \ll n$ représente le décalage de la valeur 1 sur n bits : elle a pour valeur l'entier 2^n .

Le type **pthread_t** désigne des fils d'exécution.

L'instruction `pthread_create(pthread_t *th_id, NULL, &ma_fonction, void *args)` crée un nouveau fil d'exécution qui appelle la fonction `ma_fonction` sur le ou les arguments désignés par `args` et qui s'exécute simultanément avec le fil d'exécution appelant.

L'instruction `pthread_join(th_id, NULL)` suspend l'exécution du fil d'exécution appelant jusqu'à ce que le fil d'exécution identifié par `th_id` achève son exécution.

Le type **pthread_mutex_t** désigne des verrous.

L'instruction `pthread_mutex_lock(&v)` verrouille le verrou `v`.

L'instruction `pthread_mutex_unlock(&v)` déverrouille le verrou `v`.

Le type **sem_t** désigne des sémaphores.

L'instruction `sem_init(&s, 0, v)` initialise le sémaphore `s` à la valeur `v` (avec $v \geq 0$).

L'instruction `sem_wait(&s)` décrémente le compteur du sémaphore `s` : si le compteur est toujours positif, l'appel se termine ; sinon le fil d'exécution appelant est bloqué.

L'instruction `sem_post(&s)` incrémenté le compteur du sémaphore `s` et, si le compteur redevient strictement positif, réveille un fil d'exécution bloqué sur `s`.

FIN DE L'ÉPREUVE

A2023 – INFO MPI II

**ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.**

Concours Mines-Télécom

CONCOURS 2023
DEUXIÈME ÉPREUVE D'INFORMATIQUE

Durée de l'épreuve : 4 heures

L'usage de la calculatrice et de tout dispositif électronique est interdit.

*Les candidats sont priés de mentionner de façon apparente
sur la première page de la copie :*

INFORMATIQUE II - MPI

L'énoncé de cette épreuve comporte 12 pages de texte.

Cette épreuve concerne uniquement les candidats de la filière MPI.

*Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé,
il le signale sur sa copie et poursuit sa composition en expliquant les raisons
des initiatives qu'il est amené à prendre.*

Les sujets sont la propriété du GIP CCMP. Ils sont publiés sous les termes de la licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Pas de Modification 3.0 France. Tout autre usage est soumis à une autorisation préalable du Concours commun Mines Ponts.



Préliminaires

L'épreuve est composée d'un problème unique, comportant 38 questions. Le problème est divisé en quatre sections qui peuvent être traitées séparément, à condition de lire toutes les définitions de la section 1. Dans la première section (page 1), nous introduisons la *complexité de Kolmogoroff* et étudions des propriétés de calculabilité. Dans la deuxième section (page 3), nous estimons la complexité de Kolmogoroff à l'aide du codage de Huffman. La troisième section (page 4) contient des prolégomènes pour la section suivante. Dans la quatrième section (page 5), nous nous appuyons sur un modèle de calcul épuré, introduit à travers plusieurs langages formels, afin toujours d'estimer la complexité de Kolmogoroff.

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractère différentes désigne la même entité, mais du point de vue mathématique pour la police en italique (par exemple n , \mathcal{D} ou π) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple n , d ou π).

Travail attendu

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat.

Il faudra coder des fonctions à l'aide du langage de programmation OCaml exclusivement, en reprenant l'en-tête de fonctions fourni par le sujet, sans s'obliger à recopier la déclaration des types. Il est permis d'utiliser la totalité du langage OCaml mais il est recommandé de s'en tenir aux fonctions les plus courantes afin de rester compréhensible. Des rappels ponctuels de documentation du langage OCaml peuvent être proposés à titre d'aide. Quand l'énoncé demande de coder une fonction, sauf demande explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites.

Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle.

1 Complexité de Kolmogoroff

Nous notons Σ l'ensemble ordonné des 256 caractères ASCII usuels et Σ^* l'ensemble des chaînes de caractères. Pour toute chaîne de caractères $x \in \Sigma^*$, la longueur de x , notée $|x|$, est le nombre de caractères qui la composent. Par exemple, la longueur de la chaîne "abac" est 4. Le nombre d'occurrences d'un symbole $\sigma \in \Sigma$ dans une chaîne de caractères $x \in \Sigma^*$ est noté $|x|_\sigma$. Par exemple, $|\text{abac}|_a = 2$.

Dans l'ensemble du sujet, nous nous appuyons sur une *machine universelle*, c'est-à-dire une fonction OCaml `eval`, de type `string → string`, telle que :

- si la chaîne x , de type `string`, est le code source d'une expression OCaml y de type `string` et si l'exécution du code x se termine sans erreur, alors `eval x` se termine et a pour valeur de retour la valeur de y ;
- sinon, `eval x` ne se termine pas.

Les exécutions de `eval` ont lieu sur une machine idéale dont la mémoire est infinie et qui est capable de gérer des types natifs de taille quelconque.

Définition : Pour toute chaîne de caractères $y \in \Sigma^*$, nous disons que la chaîne de caractères $x \in \Sigma^*$ est une *description* de la chaîne y si le calcul `eval x` se termine et renvoie la chaîne y . Nous appelons *complexité de Kolmogoroff* de y et notons $K(y)$ la longueur de la plus courte chaîne de caractères $x \in \Sigma^*$ qui décrit y .

L'objet de ce sujet est d'étudier des propriétés et diverses majorations de la complexité de Kolmogoroff. Dans toutes nos illustrations, nous nous concentrerons sur la description de la chaîne de caractères

$$y_0 = "1000000\dots" \in \Sigma^*,$$

qui correspond à l'entier $10^{(10^{10})}$ écrit en base 10 et que nous fixons une fois pour toutes.

1.1 Un premier exemple

- 1 – Proposer une première majoration de la complexité de Kolmogoroff $K(y_0)$, qui s'appuie sur la chaîne de caractères $x_0 = y_0 = "1000000\dots"$ comme description de y_0 .

Indication OCaml : Il est rappelé que la fonction `string_of_int` convertit un entier en une chaîne de caractères.

- 2 – Soient n un entier naturel et n' la partie entière de $\frac{n}{2}$. Exprimer la quantité 10^n en fonction de $10^{n'}$. Compléter le code OCaml suivant en utilisant une stratégie « diviser pour régner » :

```
let exp10 n = (* Calcul de  $10^n$  à écrire *)
  in string_of_int (exp10 (exp10 10))
```

afin d'en faire une description de la chaîne de caractères $y_0 = "1000000\dots"$. En déduire une nouvelle borne grossière (à 10^2 près) de la complexité de Kolmogoroff $K(y_0)$, significativement meilleure que celle de la question 1.

- 3 – Décrire quelle ou quelles difficultés adviendraient si l'on exécutait le code de la question 2 sur une machine réelle.

1.2 Quelques propriétés

Indication OCaml : L'expression `String.make (n : int) (sigma : char) : string` désigne la chaîne de caractères répétant n fois le caractère $\sigma \in \Sigma$. Pour tout entier n compris entre 0 et 255, `Char.chr (n : int) : char` désigne le n^{e} caractère dans la numérotation ASCII.

- 4 – Présenter une bijection $\varphi : \mathbb{N} \rightarrow \Sigma^*$ entre l'ensemble des entiers naturels et l'ensemble de chaînes de caractères. En écrire le code sous la forme d'une fonction OCaml `phi (n : int) : string`.

Seconde épreuve d'informatique MPI 2023

Nous prétendons avoir écrit une fonction OCaml `kolmogoroff` ($y : \text{string}$) : int qui calcule la complexité de Kolmogoroff $K(y)$.

- 5 – Écrire une fonction OCaml `psi` ($m : \text{int}$) : int dont la valeur de retour est l'entier

$$\psi(m) = \min \{n \in \mathbb{N}; K(\varphi(n)) \geq m\}$$

où $\varphi : \mathbb{N} \rightarrow \Sigma^*$ est la bijection définie à la question 4 et qui utilise la fonction `kolmogoroff`.

- 6 – Établir d'une part que, pour tout entier naturel m , on a

$$K(\varphi(\psi(m))) \geq m$$

et d'autre part que l'on a

$$K(\varphi(\psi(m))) = O(\log m).$$

Discuter l'existence de la fonction OCaml `kolmogoroff`.

Définition : Nous appelons *décompresseur* toute fonction OCaml $d : \text{string} \rightarrow \text{string}$. Pour toute chaîne de caractères $y \in \Sigma^*$ et pour tout décompresseur \mathcal{D} (noté informatiquement `d`), nous disons que la chaîne de caractères z est une *description* de la chaîne y par rapport à \mathcal{D} si le calcul `eval (d z)` se termine sans erreur et a pour valeur de retour y . Nous appelons *complexité de Kolmogoroff par rapport à \mathcal{D}* de la chaîne y , et notons $K_{\mathcal{D}}(y)$, la longueur de la plus courte chaîne de caractères $z \in \Sigma^*$ qui décrit y par rapport à \mathcal{D} .

- 7 – Dire comment se nomme en informatique un programme qui transforme un code source dans un certain langage de programmation en un code équivalent dans un second langage.

- 8 – Montrer que pour tout décompresseur \mathcal{D} , il existe une constante entière $c_{\mathcal{D}}$ telle que, pour toute chaîne de caractères y , nous avons :

$$K(y) \leq K_{\mathcal{D}}(y) + c_{\mathcal{D}}.$$

2 Estimation de la complexité grâce au décompresseur de Huffman

Nous fixons dans cette section la chaîne de caractères $x_0 \in \Sigma^*$ suivante

```
"let rec t e=if e=0 then 1 else let n=e-1 in 10*t n in let n=t 10 in t n".
```

La chaîne x_0 contient 71 caractères, l'espace étant un caractère et les guillemets ne faisant pas partie de la chaîne.

- 9 – Inférer le type OCaml de l'expression dénotée par la chaîne de caractères x_0 .

Seconde épreuve d'informatique MPI 2023

- 10 – Déterminer la valeur de l'évaluation de x_0 en tant que code source OCaml.
- 11 – Signaler une ou plusieurs caractéristiques du code source x_0 qui rend la lecture de ce code impénétrable par un humain.

Indication OCaml : Nous rappelons le détail de quelques fonctions du module OCaml Hashtbl permettant de manipuler des dictionnaires (ou tableaux associatifs) mutables.

- Hashtbl.create ($n : \text{int}$) : ($'a, 'b$) Hashtbl.t crée un dictionnaire vide de taille initiale n .
- Hashtbl.add ($d : ('a, 'b)$ Hashtbl.t) ($k : 'a$) ($v : 'b$) : unit ajoute une association entre la clé k et la valeur v au dictionnaire d .
- Hashtbl.find_opt ($d : ('a, 'b)$ Hashtbl.t) ($k : 'a$) : $'b$ option vaut Some v si le dictionnaire d possède une association entre la clé k et la valeur v et vaut None sinon.

- 12 – Écrire une fonction OCaml count ($x : \text{string}$) : (char, int) Hashtbl.t dont la valeur de retour est un dictionnaire qui associe chaque caractère $\sigma \in \Sigma$ présent dans la chaîne x à son nombre d'occurrences $|x|_\sigma$.

Nous exécutons count x_0 et obtenons le dictionnaire suivant :

'n'	't'	'i'	'l'	'1'	'c'	'f'	'h'	'r'	's'	'-'	'*'	'0'	'='	'e'	' '
8	8	4	4	4	1	1	1	1	1	1	1	3	4	10	19

Nous appelons z_0 la chaîne de bits correspondant à la compression de la chaîne x_0 par l'algorithme de Huffman.

- 13 – Dessiner un arbre de Huffman associé à la chaîne de caractères x_0 . Il est recommandé de placer les feuilles de gauche à droite comme dans le tableau ci-dessus.

Nous notons \mathcal{H} le décompresseur qui utilise l'arbre de Huffman de la question 13 pour transformer une chaîne de bits $z \in \{0, 1\}^*$ en un mot $x \in \Sigma^*$ et renvoie la chaîne de caractères "string_of_int (x)".

- 14 – Calculer la longueur $|z_0|$ de la chaîne obtenue après compression de x_0 . En déduire que la complexité de Kolmogoroff de $y_0 = 10000\dots$ par rapport au décompresseur \mathcal{H} vérifie

$$K_{\mathcal{H}}(y_0) \leq 239.$$

3 Interlude

Nous souhaitons écrire une fonction new_string : unit \rightarrow string ainsi spécifiée : à chaque appel, une chaîne de caractères inédite est produite. Nous offrons trois propositions de code.

Seconde épreuve d'informatique MPI 2023

```

let inc s = s := "a" ^ (!s); !s      (* Commun aux 3 propositions *)

let new_string1 =                      (* Proposition 1 *)
  let seed1 = ref "#" in
  fun () -> inc seed1

let new_string2 () =                  (* Proposition 2 *)
  let seed2 = ref "#" in
  inc seed2

let seed3 = ref "#"                  (* Proposition 3 *)
let new_string3 () =
  inc seed3

```

- 15 – Analyser la portée de la variable `seed1`, respectivement `seed2` et `seed3`, dans la fonction `new_string1`, respectivement `new_string2` et `new_string3`.
- 16 – Déduire de la question 15 laquelle des trois fonctions `new_string1`, `new_string2` et `new_string3` ne respecte pas la spécification. Écrire un test qui permet de discriminer la fonction erronée.
- 17 – Déduire de la question 15 laquelle des deux fonctions correctes restantes est plus propice à des erreurs de manipulation. Expliquer.

4 Estimation de la complexité grâce au décompresseur de De Bruijn

Nous nous avisons que la complexité de Kolmogoroff est influencée par la verbosité et l'expressivité du langage de programmation. Dans cette section, nous tentons de réduire l'encombrement ou la facilité d'écriture dus à la syntaxe du langage OCaml en introduisant un modèle de calcul nouveau et épuré.

4.1 Construction syntaxique d'un langage

Nous présentons systématiquement les *grammaires* sous la forme d'un quadruplet (N, Γ, S, Π) où N désigne l'alphabet des symboles non terminaux, Γ l'alphabet des symboles terminaux, $S \in N$ le symbole initial et Π l'ensemble des règles de production, de la forme $X \rightarrow \gamma$ avec $X \in N$ et $\gamma \in (N \cup \Gamma)^*$.

Une *dérivation immédiate* se note $\alpha \Rightarrow \beta$, avec $(\alpha, \beta) \in ((N \cup \Gamma)^*)^2$, et indique qu'il existe une règle de production $X \rightarrow \gamma$ de Π et des décompositions $\alpha = \alpha_1 X \alpha_2$ et $\beta = \alpha_1 \gamma \alpha_2$, avec $(\alpha_1, \alpha_2) \in ((N \cup \Gamma)^*)^2$. Une *dérivation* se note \Rightarrow^* et indique l'existence d'une suite finie, éventuellement vide, de dérivations immédiates. Enfin, le *langage engendré par une grammaire \mathcal{G}* se note $L(\mathcal{G})$ et désigne l'ensemble des mots de Γ^* qui dérivent du symbole initial S .

Seconde épreuve d'informatique MPI 2023

Selon les questions, nous représentons les mots de Γ par le type `char list` ou le type `string`.

Soit \mathcal{G}_0 la grammaire $(\{V\}, \{a, b, \#\}, V, \Pi_0)$ dont les règles de production sont

$$\Pi_0 : \quad V \rightarrow aV \mid bV \mid \#.$$

- 18 – Exhiber une expression régulière qui dénote le langage $L(\mathcal{G}_0)$.
- 19 – Dessiner l'automate de Glushkov associé à l'expression régulière de la question 18.
- 20 – Écrire une fonction OCaml `parseV (w : char list) : string * char list` dont la spécification suit :

Pré-condition : Il existe une décomposition du mot w en $w = vs$ avec $v \in L(\mathcal{G}_0)$ et $s \in \Sigma^*$.
Valeur de retour : Couple (v, s) où le préfixe v est représenté par le type `string` et s est le suffixe restant.
Effet : Une exception `SyntaxError` est levée si la pré-condition n'est pas satisfaite.

Soit \mathcal{G}_1 la grammaire $(\{T\}, \{(, ,)\}, T, \Pi_1)$ dont les règles de production sont

$$\Pi_1 : \quad T \rightarrow (TT) \mid _.$$

- 21 – Montrer que le langage $L(\mathcal{G}_1)$ est sans préfixe, c'est-à-dire qu'il n'existe pas deux mots non vides w et w' dans $L(\mathcal{G}_1)$ tels que w soit un préfixe strict de w' . On pourra, par exemple, raisonner par induction structurelle sur le nombre de parenthèses ouvrantes et fermantes qui se trouvent dans un préfixe d'un mot de $L(\mathcal{G}_1)$.

- 22 – Montrer que la grammaire \mathcal{G}_1 n'est pas ambiguë, c'est-à-dire que, pour tout mot du langage $L(\mathcal{G}_1)$, il n'existe qu'un seul arbre d'analyse (parfois aussi appelé arbre de dérivation) associé.

Soit \mathcal{G}_2 la grammaire $(\{T\}, \{\text{var}, (,), \rightarrow\}, T, \Pi_2)$ dont les règles de production sont

$$\Pi_2 : \quad T \rightarrow \text{var} \mid (TT) \mid \text{var} \rightarrow T.$$

- 23 – Montrer que la grammaire \mathcal{G}_2 n'est pas ambiguë.

Soit finalement \mathcal{G} la grammaire $(\{T, V\}, \{a, b, \#, (,), \rightarrow\}, T, \Pi)$ dont les règles de production sont

$$\Pi : \quad \begin{cases} T \rightarrow V \mid (TT) \mid V \rightarrow T \\ V \rightarrow aV \mid bV \mid \# \end{cases}$$

Nous admettons que la grammaire \mathcal{G} n'est pas ambiguë. Nous appelons *variable* les mots de $L(\mathcal{G}_0)$. Informellement, la grammaire \mathcal{G} engendre un langage $L(\mathcal{G})$ qui permet de parler de *variables*, d'*applications* d'une expression à une autre et de *fonctions*. Il nous reste à en construire une sémantique, ce qui sera fait en section 4.3.

Seconde épreuve d'informatique MPI 2023

Afin de représenter l'*arbre d'analyse* d'un mot du langage $L(\mathcal{G})$, nous introduisons le type ada par la déclaration suivante.

```
type ada = V of string | A of (ada * ada) | F of (string * ada)
```

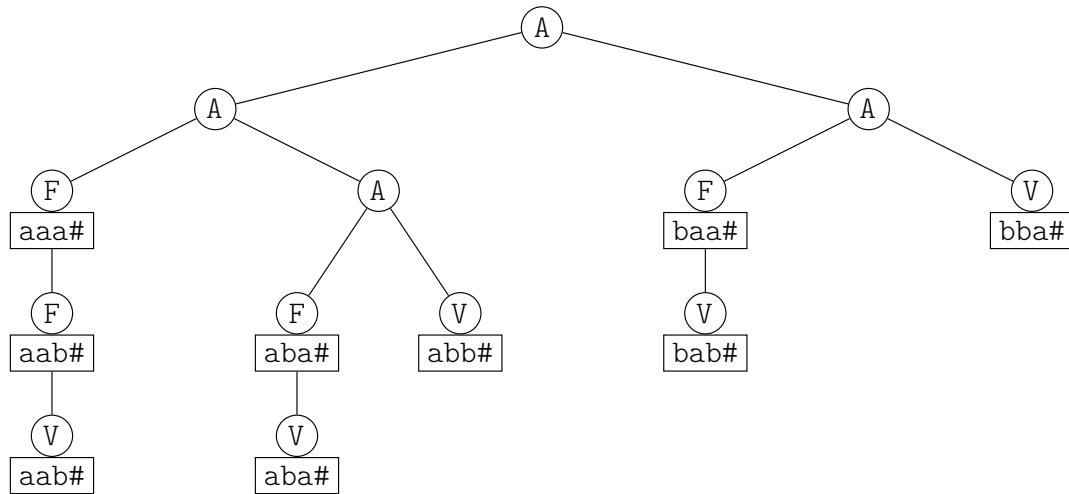
Nous notons \mathcal{A} l'*ensemble des arbres d'analyse*.

Le constructeur OCaml V représente les dérivations immédiates de règle $T \rightarrow V$ et introduit les feuilles ; le constructeur OCaml A représente les dérivations immédiates de règle $T \rightarrow (TT)$ et introduit des nœuds internes d'arité 2 ; le constructeur OCaml F représente les dérivations immédiates de règle $V \rightarrow T$ et introduit des nœuds internes d'arité 1. Les dérivations à partir du symbole non terminal V sont directement représentées par une valeur OCaml de type string.

Par exemple, le mot

$$((aaa\# \rightarrow aab\# \rightarrow aab\#(aba\# \rightarrow aba\#abb\#))(baa\# \rightarrow bab\#bba\#)) \in L(\mathcal{G})$$

admet pour arbre d'analyse :



- 24 – Écrire une fonction OCaml parseT ($w : \text{char list}$) : ada * char list dont la spécification suit :

Pré-condition : Il existe une décomposition du mot w en $w = ps$ avec $p \in L(\mathcal{G})$ et $s \in \Sigma^*$, dans laquelle le préfixe p est de longueur maximale.

Valeur de retour : Couple (a, s) où a est l'arbre d'analyse de p et s est le suffixe restant.

Effet : Une exception SyntaxError est levée si la pré-condition n'est pas satisfait.

Indication OCaml : Nous supposons définie une fonction `charlist_of_string : string -> char list` qui transforme une chaîne de caractères en une liste de caractères.

- 25 – Écrire une fonction OCaml parse ($w : \text{string}$) : ada dont la valeur de retour est l'unique arbre d'analyse de w quand $w \in L(\mathcal{G})$ et qui lève une exception SyntaxError sinon.

Seconde épreuve d'informatique MPI 2023

Définition : Pour tout entier naturel $n \in \mathbb{N}$, nous définissons le mot

$$[n] = b\# \rightarrow a\# \rightarrow (b\#(b\#\dots(b\#(b\#a\#))\dots)) \in L(\mathcal{G})$$

où la variable $b\#$ apparaît n fois à droite des deux symboles \rightarrow .

- 26 – Indiquer s'il existe un automate fini capable de reconnaître le langage $\{[n]; n \in \mathbb{N}\}$ et justifier la réponse.

Définition : Plus généralement, nous disons qu'un mot w de $L(\mathcal{G})$ *incarne* un entier naturel $n \in \mathbb{N}$ s'il existe deux variables distinctes v_1 et v_2 dans $L(\mathcal{G}_0)$ telles que w est de la forme

$$w = v_2 \rightarrow v_1 \rightarrow (v_2(v_2 \dots (v_2(v_2v_1)) \dots)) \in L(\mathcal{G})$$

où la variable v_2 apparaît n fois à droite des deux symboles \rightarrow .

- 27 – Écrire une fonction OCaml `int_of_ada` (`a : ada`) : `int` dont la valeur de retour est l'entier naturel n incarné par a . Si un tel entier n'existe pas, une exception `SyntaxError` est levée.

4.2 Réécriture des variables et sérialisation

- 28 – Nommer une structure de donnée concrète efficace qui réalise le type de donnée abstrait « ensemble » lorsqu'il existe une relation d'ordre entre les objets à ranger.

Indication OCaml : Nous supposons définis un module OCaml `StringSet` permettant de construire des ensembles de chaînes de caractères persistants, de type `StringSet.t`, et des fonctions :

- `StringSet.mem` : `string` \rightarrow `StringSet.t` \rightarrow `bool` qui teste l'appartenance d'un élément à un ensemble.
- `StringSet.remove` : `string` \rightarrow `StringSet.t` \rightarrow `StringSet.t` qui retourne un ensemble privé d'un élément.
- `StringSet.singleton` : `string` \rightarrow `StringSet.t` qui construit un singleton à partir d'un élément.
- `StringSet.union` : `StringSet.t` \rightarrow `StringSet.t` \rightarrow `StringSet.t` qui construit l'union de deux ensembles.

Définition : L'ensemble des *variables libres* $VL(a) \subseteq \Sigma^*$ d'un arbre d'analyse $a \in \mathcal{A}$ est défini par induction structurelle avec les règles d'inférence suivantes :

- si l'arbre d'analyse a est de la forme `V v`, où $v \in L(\mathcal{G}_0)$, alors $VL(a)$ est le singleton $\{v\}$;
- si l'arbre d'analyse a est de la forme `A (a1, a2)`, où a_1 et a_2 sont deux arbres d'analyse, alors $VL(a)$ est la réunion $VL(a_1) \cup VL(a_2)$;
- si l'arbre d'analyse a est de la forme `F (v, a1)`, où $v \in L(\mathcal{G}_0)$ et a_1 est un arbre d'analyse, alors $VL(a)$ est l'ensemble $VL(a_1) \setminus \{v\}$.

- 29 – Écrire une fonction OCaml `free_vars` (`a : ada`) : `StringSet.t` dont la valeur de retour est l'ensemble des variables libres de l'arbre d'analyse a .

Seconde épreuve d'informatique MPI 2023

Définition : Un arbre d'analyse est dit *clos* s'il ne contient pas de variables libres ; de même, un mot de $L(\mathcal{G})$ est dit *clos* si son arbre d'analyse est clos.

Dans un arbre d'analyse clos $a \in \mathcal{A}$, pour toute chaîne de caractères $v \in L(\mathcal{G}_0)$, si la construction OCaml $V\ v$ apparaît comme feuille de l'arbre a , alors il existe un nœud interne de la forme $F\ (v, _)$ parmi les descendants de $V\ v$ qui coïncide avec l'« introduction » de la variable v .

Afin de ne plus s'encombrer avec des variables nommées par une chaîne de caractères, nous adoptons une nouvelle représentation des mots du langage $L(\mathcal{G})$ sous forme d'un arbre, appelé *terme de De Bruijn*.

Définition : Un terme de De Bruijn s'obtient à partir d'un arbre d'analyse $a \in \mathcal{A}$ en remplaçant toute feuille de l'arbre a , disons $V\ v$ avec $v \in L(\mathcal{G}_0)$, par une feuille étiquetée par l'entier $u = 1 + \ell$, où $\ell \in \mathbb{N}$ est le nombre de noeuds internes de la forme $F\ (v', _)$ avec $v' \in L(\mathcal{G}_0) \setminus \{v\}$, qui existent entre ladite feuille $V\ v$ et le plus proche nœud interne de la forme $F\ (v, _)$ parmi ses descendants dans l'arbre d'analyse a .

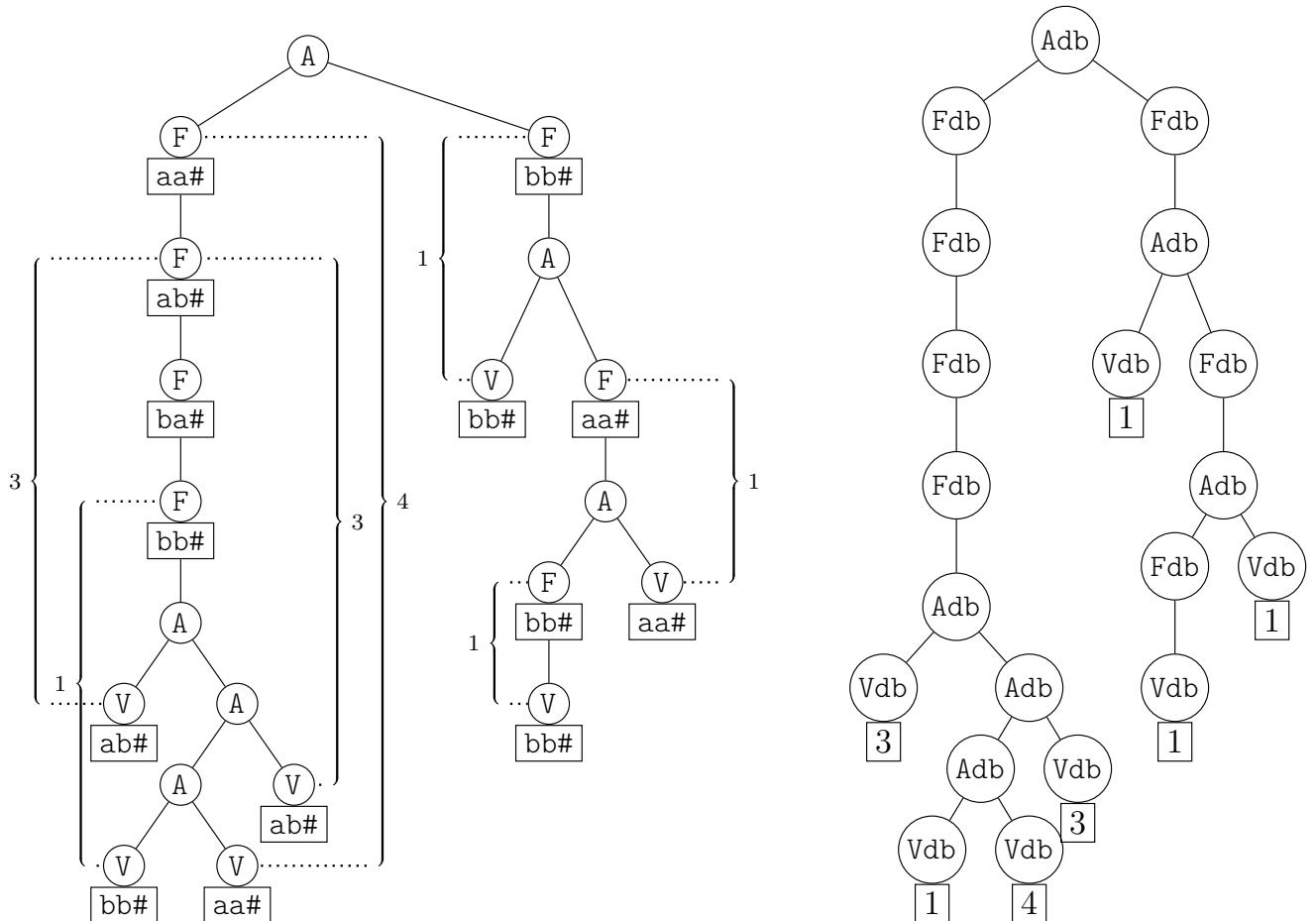
Nous déclarons un nouveau type

```
type tdb = Vdb of int | Adb of tdb * tdb | Fdb of tdb
```

Nous notons \mathcal{T} l'*ensemble des termes de De Bruijn*.

Voici un exemple de construction d'un terme de De Bruijn (à droite) à partir d'un arbre d'analyse (à gauche) du mot

(aa#->ab#->ba#->bb#->(ab#((bb#aa#)ab#))bb#->(bb#aa#->(bb#->bb#aa#))) :



- 30 – Dessiner le terme de De Bruijn qui représente le mot $[n]$.
- 31 – Écrire une fonction OCaml ada_of_tdb ($t : \text{tdb}$) : ada dont la valeur de retour est un arbre d'analyse clos associé au terme de De Bruijn t .

Définition : Le *codage binaire* des termes de De Bruijn est l'application $t \in \mathcal{T} \mapsto \hat{t} \in \{0, 1\}^*$ définie par induction structurelle avec les règles d'inférence suivantes sur l'ensemble des termes de De Bruijn :

- si le terme $t \in \mathcal{T}$ est de la forme Vdb u , avec $u \in \mathbb{N}^*$, alors \hat{t} est la chaîne de caractères $11..10$ (avec le symbole 1 répété u fois).
- si le terme $t \in \mathcal{T}$ est de la forme Adb (t_1, t_2) , t_1 et t_2 étant deux termes de De Bruijn, alors \hat{t} est la chaîne de caractères $01\hat{t}_1\hat{t}_2$.
- si le terme $t \in \mathcal{T}$ est de la forme Fdb t_1 , où t_1 est un terme de De Bruijn, alors \hat{t} est la chaîne de caractères $00\hat{t}_1$.

- 32 – Soient n un entier naturel et t le terme de De Bruijn associé au mot $[n]$ et obtenu à la question 30. Calculer la longueur $|\hat{t}|$ de la chaîne de caractères qui encode t .
- 33 – Vérifier que le codage binaire des termes de De Bruijn est une application injective.

Nous utilisons le type `string` avec les caractères '0' et '1' afin de représenter des codages binaires de termes de De Bruijn.

- 34 – Écrire une fonction OCaml decode ($z : \text{string}$) : tdb dont la valeur de retour est l'unique terme de De Bruijn t tel que $\hat{t} = z$ si un tel terme t existe et qui lève une exception `SyntaxError` sinon.

Il est possible de s'appuyer sur la fonction `charlist_of_string` précédemment présentée.

4.3 Interpréteur et décompresseur de De Bruijn

Dans cette sous-section, nous construisons un interpréteur, c'est-à-dire une procédure qui évalue les expressions appartenant au langage $L(\mathcal{G})$ et en déduisons une nouvelle description du mot $y_0 = "1000000..."$ relative à un décompresseur approprié.

Naïvement, lorsque nous rencontrons un sous-mot de la forme $w = (v \rightarrow w_b w_a)$ avec $v \in L(\mathcal{G}_0)$ et $(w_a, w_b) \in (L(\mathcal{G}))^2$, nous aimerais que w soit équivalent à un mot tiré de w_b dont les occurrences de la variable v ont été remplacées par w_a . Autrement dit, lorsqu'un arbre d'analyse est de la forme A (F (v, b), a), nous voudrions construire un nouvel arbre à partir de $b \in \mathcal{A}$ et dans lequel les apparitions de $v \in L(\mathcal{G}_0)$ sont devenues des sous-arbres $a \in \mathcal{A}$.

Seconde épreuve d'informatique MPI 2023

Définition : Soient $(a, b) \in (\mathcal{A})^2$ deux arbres d'analyse et $v \in L(\mathcal{G}_0)$ une variable. La *substitution de la variable v par l'arbre a dans l'arbre b* est l'application $[v \leftarrow a] : \mathcal{A} \rightarrow \mathcal{A}$ définie par induction structurelle avec les règles d'inférence suivantes :

- Si l'arbre $b \in \mathcal{A}$ est de la forme $V\ v_1$, avec $v_1 \in L(\mathcal{G}_0)$, alors

$$[v \leftarrow a](b) = \begin{cases} a & \text{si } v = v_1 \\ b & \text{si } v \neq v_1. \end{cases}$$

- Si l'arbre $b \in \mathcal{A}$ est de la forme $A\ (b_1, b_2)$, où $b_1 \in \mathcal{A}$ et $b_2 \in \mathcal{A}$ sont deux arbres d'analyse, alors

$$[v \leftarrow a](b) = A\ (b_1', b_2')$$

où $b_1' = [v \leftarrow a](b_1)$ et $b_2' = [v \leftarrow a](b_2)$.

- Si l'arbre $b \in \mathcal{A}$ est de la forme $F\ (v_1, b_1)$, avec $v_1 \in L(\mathcal{G}_0)$ et $b_1 \in \mathcal{A}$, alors

$$[v \leftarrow a](b) = \begin{cases} b & \text{si } v = v_1 \\ F\ (v_1, b_1') & \text{avec } b_1' = [v \leftarrow a](b_1) \quad \text{si } v \neq v_1 \text{ et } v_1 \notin VL(a) \\ F\ (v_1', b_1') & \text{avec } b_1' = [v \leftarrow a]([v_1 \leftarrow v_1'](b_1)) \quad \text{sinon} \end{cases}$$

où $v_1' \in L(\mathcal{G}_0)$ est une variable inédite qui n'appartient pas à $VL(a) \cup VL(b) \cup \{v, v_1\}$.

Nous supposons déjà programmée une fonction `new_string : unit -> string` inspirée de la section 3 qui permet, si besoin, d'engendrer une variable de $L(\mathcal{G}_0)$ jamais encore utilisée.

- 35 – Écrire une fonction OCaml `substitute (v : string) (by_a : ada) (in_b : ada) : ada` qui substitue la variable v par l'arbre a dans l'arbre b .

Définition : La *réduction en un pas* est l'application $\triangleright : \mathcal{A} \rightarrow \mathcal{A}$ définie par induction structurelle avec les règles d'inférence suivantes :

- Si l'arbre $a \in \mathcal{A}$ est de la forme $V\ v$, où $v \in L(\mathcal{G}_0)$, alors la valeur $\triangleright(a)$ n'est pas définie.
- Si l'arbre $a \in \mathcal{A}$ est de la forme $A\ (a_1, a_2)$, où $a_1 \in \mathcal{A}$ et $a_2 \in \mathcal{A}$ sont deux arbres d'analyse, alors

$$\triangleright(a) = \begin{cases} A\ (a_1, a_2') & \text{avec } a_2' = \triangleright(a_2) \quad \text{si } a_1 \text{ est de la forme } V\ v \\ a' & \text{avec } a' = [v \leftarrow a_2](a_{11}) \quad \text{si } a_1 \text{ est de la forme } F\ (v, a_{11}) \\ A\ (a_1', a_2) & \text{avec } a_1' = \triangleright(a_1) \quad \text{sinon.} \end{cases}$$

- Si l'arbre $a \in \mathcal{A}$ est de la forme $F\ (v, a_1)$, où $v \in L(\mathcal{G}_0)$ et $a_1 \in \mathcal{A}$, alors

$$\triangleright(a) = F\ (v, a_1') \text{ où } a_1' = \triangleright(a_1).$$

- 36 – Écrire une fonction OCaml `reduce_one_step (a : ada) : ada` qui implémente la réduction en un pas \triangleright . Lorsque `reduce_one_step` rencontre un cas non défini, une exception `NoReduction` est levée.

- 37 – Écrire une fonction OCaml `interpret (a : ada) : ada` qui applique répétitivement la réduction en un pas \triangleright à l'arbre d'analyse a jusqu'à ce que la réduction ne soit plus définie. La valeur de retour est le dernier arbre d'analyse rencontré.

Seconde épreuve d'informatique MPI 2023

Nous admettons que, pour tout entier naturel non nul n , si π est le mot de $L(\mathcal{G})$

$$\pi = (\text{a}\# \rightarrow ((\text{a}\#\text{a}\#)\text{a}\#) \lceil n \rceil),$$

l'expression `interpret (parse pi)` produit une incarnation de l'entier $n^{(n^n)}$. Nous notons \mathcal{B} le décompresseur défini par

```
let decompB (z : string) : string =
  "string_of_int ∘ int_of_ada ∘ interpret ∘ ada_of_tdb ∘ decode" ^ z ^ ")")"
```

- 38 – Proposer une méthode pour obtenir une chaîne de caractères z_0 , formée uniquement de 0 et de 1, telle que `decompB z0` a pour valeur de retour la chaîne de caractères $y_0 = "1000000..."$ et en calculer la longueur. En déduire que la complexité de Kolmogoroff de y_0 par rapport au décompresseur \mathcal{B} vérifie

$$K_{\mathcal{B}}(y_0) \leq 70.$$

FIN DE L'ÉPREUVE

A2023 – INFO MP

**ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.**

Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle International).

CONCOURS 2023**ÉPREUVE D'INFORMATIQUE MP**

Durée de l'épreuve : 3 heures

L'usage de la calculatrice ou de tout dispositif électronique est interdit.

Cette épreuve concerne uniquement les candidats de la filière MP.

*Les candidats sont priés de mentionner de façon apparente
sur la première page de la copie :*

INFORMATIQUE - MP

L'énoncé de cette épreuve comporte 11 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Préliminaires

L'épreuve est composée d'un problème unique, comportant 27 questions. Après cette section de préliminaires, qui présente le *jeu du hanjie*, le problème est divisé en trois sections qui peuvent être traitées séparément, à condition d'avoir lu les définitions introduites jusqu'à la question traitée. Dans la première section (page 2), nous résolvons le jeu par des raisonnements logiques. Dans la deuxième section (page 3), nous modélisons le jeu et le résolvons par une stratégie algorithmique de retour sur trace. Dans la troisième section (page 6), nous appliquons une méthode de parcours de graphe en théorie des automates pour accélérer la résolution du jeu.

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractères différentes désigne 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 espace fixe (par exemple `n`).

Des rappels de logique et des extraits du manuel de documentation de OCaml sont reproduits en annexe. Ces derniers portent sur le module `Array` et le module `Hashtbl`.

Travail attendu

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat.

Il faudra coder des fonctions à l'aide du langage de programmation OCaml, en reprenant l'en-tête de fonction fourni par le sujet, sans s'obliger à recopier la déclaration des types. Quand l'énoncé demande de coder une fonction, sauf demande explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de vérifier que des préconditions sont satisfaites.

Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle.

Description du jeu du hanjie

Le *hanjie* est un jeu de réflexion à l'intersection de l'art des pixels et de la tomographie discrète, technique d'imagerie courante en médecine, en géophysique ou en science des matériaux entre autres domaines.

Il consiste à retrouver une image par le noircissement de certaines cases d'une grille rectangulaire sur la base d'indications laissées sur les côtés de la grille. Pour chaque rangée, qu'elle soit horizontale ou verticale, le joueur dispose d'une suite d'entiers non nuls t_1, t_2, t_3 , etc. qui indiquent que la rangée contient une série de t_1 cases noires consécutives, suivie plus loin d'une série de t_2 cases noires consécutives, et ainsi de suite. Un nombre quelconque de cases blanches peut se trouver en tête ou en queue de rangée ; au moins une case blanche sépare deux séries de cases noires.

Épreuve d'informatique d'option MP 2023

Voici un exemple, que nous résolvons à la main. Une grille vide est fournie ci-contre. Elle est de dimension 5×7 . Nous marquons les cases par des symboles « ? » tant que nous ignorons leur couleur.

Nous comptons les rangées à partir de 0, de gauche à droite pour les colonnes et du haut vers le bas pour les lignes.

[4, 2]	?	?	?	?	?	?	?
[1, 1, 2]	?	?	?	?	?	?	?
[1, 2, 1]	?	?	?	?	?	?	?
[1, 1]	?	?	?	?	?	?	?
[6]	?	?	?	?	?	?	?

La colonne 0 et la colonne 5 sont de longueur 5. Or l'indication est [5] dans les deux cas. Elles doivent donc chacune contenir 5 blocs noirs consécutifs. Nous les noircissons en totalité.

[5]	?	?	?	?	?	?	?
[1, 1, 2]	?	?	?	?	?	?	?
[1, 2, 1]	?	?	?	?	?	?	?
[1, 1]	?	?	?	?	?	?	?
[6]	?	?	?	?	?	?	?

Dans la ligne 0, il n'y a qu'une seule manière de placer un bloc de 4 cases noires puis 2 cases noires. De même, dans la ligne 2, il n'y a qu'une seule manière de positionner le bloc de longueur 2 : il doit se trouver au milieu entre les deux blocs déjà isolés. Dans la ligne 3, nous avons déjà placé deux cases noires : les autres sont donc toutes blanches. Enfin, dans la ligne 4, il n'y a plus qu'une seule manière de placer un bloc de 6 cases noires.

[5]	?	?	?	?	?	?	?
[1, 1, 2]	?	?	?	?	?	?	?
[1, 2, 1]	?	?	?	?	?	?	?
[1, 1]	?	?	?	?	?	?	?
[6]	?	?	?	?	?	?	?

[5]	?	?	?	?	?	?	?
[1, 1, 2]	?	?	?	?	?	?	?
[1, 2, 1]	?	?	?	?	?	?	?
[1, 1]	?	?	?	?	?	?	?
[6]	?	?	?	?	?	?	?

Enfin, en reprenant les indications des colonnes, nous voyons que dans les colonnes 1, 2 et 4, la dernière case inconnue est blanche. Dans la colonne 3 et 6 en revanche, nous devons noircir la dernière case inconnue. Nous avons obtenu une solution de notre hanjie. Elle est unique.

1. Hanjie et calcul de vérité

Dans cette section, nous raisonnons avec la logique propositionnelle pour déterminer la couleur de certaines cases. Nous nous concentrerons sur le hanjie h_0 défini par

[2]	[1]	[1]
[2]		
[1, 1]		

et dont une solution est la suivante :

[2]	[1]	[1]
[2]		
[1, 1]		

- 1 – Établir, par raisonnement en langue française, que la solution du hanjie h_0 est unique.

Nous introduisons six variables booléennes, nommées x_0, x_1, \dots, x_5 et correspondant aux cases ci-contre. Nous associons la valeur de vérité V (vrai) à la couleur noire et F (faux) à la couleur blanche.

x_0	x_1	x_2
x_3	x_4	x_5

 Épreuve d'informatique d'option MP 2023

Soit L_0 le prédicat : « l'indication de la ligne zéro du hanjie h_0 est satisfaite ».

- 2 – Dresser la table de vérité du prédicat L_0 portant sur les variables x_0, x_1, x_2 . En déduire une formule de logique φ sous forme normale conjonctive qui décrit le prédicat L_0 .

Soit C_1 le prédicat : « l'indication de la colonne du milieu du hanjie h_0 est satisfaite ».

- 3 – Dresser la table de vérité du prédicat C_1 portant sur les variables x_1, x_4 . En déduire une formule de logique ψ sous forme normale conjonctive qui décrit le prédicat C_1 .

Les règles d'inférence de la déduction naturelle sont rappelées dans l'annexe B.

- 4 – Construire un arbre de preuve qui démontre le séquent $\varphi \vdash x_1$ à partir des règles d'inférence de la déduction naturelle.

- 5 – Construire de même un arbre de preuve qui démontre le séquent $\psi, x_1 \vdash \neg x_4$.

Nous notons ψ' la formule de logique obtenue à partir de ψ en remplaçant la variable x_1 par x_2 et la variable x_4 par x_5 .

- 6 – Démontrer qu'il n'existe pas d'arbre de preuve qui démontre la formule $\varphi \wedge \psi' \rightarrow \neg x_2$.

2. Cadre de résolution systématique du hanjie

2.1. Le hanjie

Nous fixons un alphabet $\mathcal{C} = \{N, B\}$ et notons $\bar{\mathcal{C}}$ l'alphabet complété $\bar{\mathcal{C}} = \{N, B, I\}$. Les symboles **N**, **B** et **I** désignent respectivement les couleurs **Noir**, **Blanc** et **Inconnu**. Nous déclarons en OCaml :

```
type couleur = N | B | I
```

- 7 – Écrire une fonction OCaml `est_connu (c:couleur) : bool` dont la valeur de retour est le booléen **V** (vrai) si $c \in \mathcal{C} = \{N, B\}$ et le booléen **F** (faux) si c égale la couleur **I**.

Pour l'ensemble du sujet, nous fixons deux entiers naturels non nuls m et n qui désignent respectivement un nombre de lignes et un nombre de colonnes.

Définition : Une *présolution* d'un hanjie est un tableau de dimension $m \times n$ à valeurs dans l'ensemble $\bar{\mathcal{C}}$.

Les figures de l'introduction (en page 2) sont des exemples de présolutions.

Épreuve d'informatique d'option MP 2023

Indication OCaml : Nous déclarons des constantes globales, le type et le constructeur suivants :

```

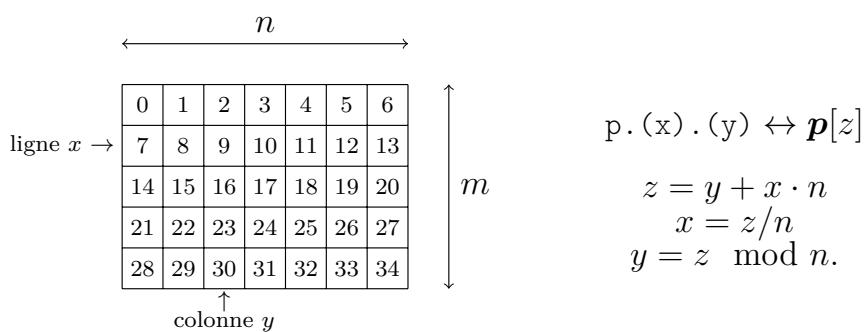
let m = 5
let n = 7
type presolution = couleur array array

let presolution_init () : presolution = (* cree une presolution *)
    Array.make_matrix m n I           (* toute égale à I *)

```

Lorsqu'un tableau \mathbf{p} est de dimension $m \times n$, nous numérotions ses cases ligne après ligne avec les entiers compris entre 0 et $m \cdot n - 1$. La case en ligne x et colonne y a pour numéro $z = y + x \cdot n$. Nous notons alors $\mathbf{p}[z]$ la valeur de \mathbf{p} en position z .

Dans notre exemple, nous aurions les numéros :



□ 8 – Écrire en langage OCaml un accesseur get (p:presolution) (z:int) : couleur dont la valeur de retour est la couleur $\mathbf{p}[z]$.

□ 9 – Écrire en langage OCaml un transformateur set (p:presolution) (z:int) (c:couleur) : presolution dont la valeur de retour \mathbf{p}' est une copie profonde de \mathbf{p} avec $\mathbf{p}'[z] = c$, c'est-à-dire une copie n'ayant aucun lien avec la présolution initiale.

Dans tout le sujet, on s'astreint à manipuler le type presolution exclusivement en employant les fonctions presolution_init, set et get. De la sorte, on pourra considérer que le type presolution est immuable.

□ 10 – Définir le terme *immuable* et citer un avantage à utiliser des variables immuables.

La *ligne* x d'un tableau, avec $0 \leq x < m$, désigne l'ensemble des positions de numéro $x \cdot n$, $x \cdot n + 1$, ..., $(x + 1) \cdot n - 1$ (lire n numéros au total). La *colonne* y , avec $0 \leq y < n$, désigne l'ensemble des positions de numéro y , $y + n$, ..., $y + (m - 1) \cdot n$ (lire m numéros au total). Nous utilisons le terme *rangée* pour parler indifféremment d'une ligne ou d'une colonne.

Définition : Nous disons qu'une rangée d'une présolution est *complète* si elle est à valeurs dans \mathcal{C} .

□ 11 – Écrire une fonction OCaml est_complete_lig (p:presolution) (x:int) : bool qui teste si la ligne x de la présolution \mathbf{p} est complète.

Nous supposons écrite de même une fonction est_complete_col (p:presolution) (y:int) : bool, qui teste si la colonne y de la présolution \mathbf{p} est complète.

 Épreuve d'informatique d'option MP 2023

Définition : Si une rangée est complète, nous appelons *trace* la suite des longueurs des sous-suites maximales de termes consécutifs égaux à Noir. Par exemple, la trace de la rangée

$$B, \underbrace{N, N}_1, B, \underbrace{N}_1, B, \underbrace{N, N, N}_3, B, \underbrace{N}_1$$

est $[2; 1; 3; 1]$.

- 12 – Écrire une fonction OCaml `trace_lig (p:presolution) (x:int) : int list` dont la valeur de retour est la trace de la ligne x de la présolution p .

Nous supposons écrite de la même manière une fonction `trace_col (p:presolution) (y:int) : int list`, dont la valeur de retour est la trace de la colonne y de la présolution p .

Définition : Le terme *indication* désigne toute suite finie d'entiers naturels non nuls. Nous appelons *hanjie de dimension* $m \times n$ la donnée d'un tableau d'indications \mathbf{ind}_{lig} , de longueur m , et d'un tableau d'indications \mathbf{ind}_{col} , de longueur n .

Indication OCaml : Nous déclarons :

```
type hanjie = { ind_lig : int list array;
                 ind_col : int list array }
```

Définition : Une *solution* d'un hanjie $h = (\mathbf{ind}_{lig}, \mathbf{ind}_{col})$ de dimension $m \times n$ est un tableau p de même dimension $m \times n$ et à valeurs dans l'ensemble \mathcal{C} tel que le tableau des traces des lignes de p égale \mathbf{ind}_{lig} et le tableau des traces des colonnes de p égale \mathbf{ind}_{col} .

Par exemple, le hanjie étudié en introduction et déclaré par :

```
let h_escargot = {ind_lig = [| [4;2]; [1;1;2]; [1;2;1]; [1;1]; [6] |];
                  ind_col = [| [5]; [1;1]; [1;1;1]; [3;1]; [1] ; [5];
                             [2] |] }
```

admet comme solution la valeur OCaml :

```
let p_escargot = [| [|N; N; N; N; B; N; N|];
                   [|N; B; B; N; B; N; N|];
                   [|N; B; N; N; B; N; B|];
                   [|N; B; B; B; B; N; B|];
                   [|N; N; N; N; N; N; B|] |]
```

- 13 – Écrire une fonction OCaml `est_admissible (h:hanjie) (p:presolution) : bool` dont la valeur de retour est le booléen V (vrai) si et seulement si, pour toute rangée complète de p , la trace égale l'indication du hanjie. Il n'est pas nécessaire de contrôler que la présolution et le hanjie ont même dimension.

2.2. Recherche de solutions

Définition : Nous disons qu'une présolution p' étend une présolution p , si pour tout numéro z avec $p'[z] \in \mathcal{C}$, les couleurs $p'[z]$ et $p[z]$ sont égales.

Par exemple, la présolution $p' = \boxed{B \ N \ I \ N \ B}$ étend $p = \boxed{B \ I \ I \ N \ I}$ puisque deux cases sont passées de la couleur I à une couleur connue N ou B et les autres sont restées inchangées.

 Épreuve d'informatique d'option MP 2023

Indication OCaml : Le type paramétré `'a option` permet de distinguer l'absence d'une valeur définie, avec le constructeur `None`, de la présence d'une valeur v de type `'a`, avec la construction `Some v`. Il est défini par la déclaration :

```
type 'a option = None | Some of 'a
```

- 14 – Écrire une fonction OCaml `etend_trivial (h:hanjie) (p:presolution) (z:int) (c:couleur)` : `presolution option` qui, si la copie \mathbf{p}' de \mathbf{p} avec $\mathbf{p}'[z] = c$ est une extension de \mathbf{p} et est admissible au sens de la question 13, renvoie `Some p'` et sinon renvoie `None`.

Plus généralement, nous appelons *extenseur* toute fonction OCaml `etend (h:hanjie) (p:presolution) (z:int) (c:couleur)` : `presolution option` ainsi spécifiée :

Précondition : Le numéro z est valide ($0 \leq z < m \cdot n$). La couleur c appartient à \mathcal{C} .

Postcondition : Si la valeur de retour vaut `Some p'`, alors

- (i) \mathbf{p}' est une extension admissible de \mathbf{p} avec $\mathbf{p}'[z] = c$
- (ii) et toute présolution complète \mathbf{p}'' qui satisfait (i) est une extension de \mathbf{p}' .

Sinon, si la valeur de retour est `None`, alors il n'existe pas de présolution \mathbf{p}' complète vérifiant (i).

Nous définissons le type

```
type extenseur = hanjie -> presolution ->
                  int -> couleur -> presolution option
```

Nous souhaitons implémenter une recherche de solutions par une stratégie de *retour sur trace* dans laquelle les cases d'une présolution sont considérées par numéro croissant.

- 15 – Compléter le code suivant afin que, si \mathbf{p} est une présolution dont au moins les $z - 1$ premières cases appartiennent à \mathcal{C} , alors `explore p z` renvoie si possible une solution qui étend \mathbf{p} à partir du numéro z à l'aide d'itérations sur l'extenseur `ext` et sinon renvoie la valeur `None` :

```
let resout (h:hanjie) (ext:extenseur) : presolution option =
  let p0 = presolution_init () in
  let rec explore (p:presolution) (z:int) : presolution option =
    (* A COMPLÉTER *)
    in
    explore p0 0
```

3. Résolution autonome de lignes et extenseur

La stratégie d'extension de la question 14 est très naïve. Elle ignore les déductions intermédiaires qui pourraient être faites grâce à une seule indication à partir d'une rangée partiellement complétée.

Soit $w \in \overline{\mathcal{C}}^n$ le mot formé des inscriptions dans une rangée dont l'indication est $\tau = [t_1; t_2; \dots; t_s]$ avec $s \geq 1$. Nous notons $[w]_\tau$ l'ensemble des mots de \mathcal{C}^n de trace τ qui étendent w . Lorsque l'ensemble $[w]_\tau$ n'est pas vide, nous posons, pour tout indice i compris entre 0 et $n - 1$, l'ensemble des couleurs

$$E_i = \{z_i \in \mathcal{C}; z = z_0 \dots z_{n-1} \in [w]_\tau\}.$$

 Épreuve d'informatique d'option MP 2023

Enfin nous posons, pour tout indice i compris entre 0 et $n - 1$,

$$x_i = \begin{cases} \text{B} & \text{si } E_i = \{\text{B}\} \\ \text{N} & \text{si } E_i = \{\text{N}\} \\ \text{I} & \text{si } E_i = \{\text{B}, \text{N}\}. \end{cases}$$

Définition : Nous appelons *plus grande extension commune du mot w par rapport à l'indication τ* le mot $x = x_0 \cdots x_{n-1} \in \bar{\mathcal{C}}^*$, lorsqu'il est défini.

Par exemple, si nous rencontrons la ligne

N	I	I	I	I	I	I	N	I
---	---	---	---	---	---	---	---	---

 avec l'indication $[1, 2, 1]$, il y a deux extensions complètes possibles :

N	B	N	N	B	B	N	B
---	---	---	---	---	---	---	---

 ou

N	B	B	N	N	B	N	B
---	---	---	---	---	---	---	---

. La plus grande extension commune est dans ce cas

N	B	I	N	I	B	N	B
---	---	---	---	---	---	---	---

.

- 16 – Dans cette question uniquement, nous supposons que n est de la forme $n = 3s - 1$, où s est un entier naturel non nul, que l'indication τ est égale à $[1; 1; \dots; 1]$ avec s occurrences de 1 et que w vaut I^n . Compter le nombre de mots du langage $[w]_\tau$.

Nous notons L_τ le langage des mots de \mathcal{C}^* de trace $\tau = [t_1; t_2; \dots; t_s]$ quelconque. Nous supposons toujours que l'on a $s \geq 1$.

- 17 – Proposer une expression régulière e qui dénote le langage L_τ . Aucune justification n'est attendue.

- 18 – Dessiner un automate fini déterministe incomplet à $s + \sum_{i=1}^s t_i$ états qui reconnaît le langage L_τ . Aucune justification n'est attendue.

- 19 – Dire combien d'états l'automate de Glushkov qui dérive de l'expression régulière e de la question 17 possède et si cet automate coïncide avec celui dessiné à la question 18.

Dans ce qui suit, nous nous intéressons à des *automates finis déterministes incomplets* qui ne possèdent qu'un seul état final et dont l'alphabet est toujours \mathcal{C} . Nous numérotions systématiquement les états de sorte que l'état initial soit 0 et l'unique état final soit $r - 1$, où r est le nombre total d'états. Le terme *transition* désigne un triplet (q, c, q') où q et q' sont des états (avec $0 \leq q, q' < r$) et c est une couleur (avec $c \in \mathcal{C}$). Nous représentons la fonction de transition par un tableau de longueur r ; la case q du tableau contient un dictionnaire qui, quand il existe une transition (q, c, q') , associe la couleur c à l'état q' . Nous déclarons :

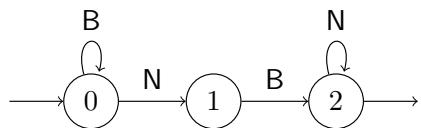
```
type automate = { r : int;
                  transitions : (couleur, int) Hashtbl.t array}
```

Définition : Le *déploiement* d'un automate \mathcal{A} ayant r états par un mot $w = w_0 w_1 \cdots w_{n-1} \in \bar{\mathcal{C}}^*$ de longueur n est un nouvel automate, noté $\mathcal{A} \bowtie w$, qui possède $r \cdot (n + 1)$ états. Pour toute transition (q, c, q') dans l'automate \mathcal{A} et pour tout indice i compris entre 0 et $n - 1$,

 Épreuve d'informatique d'option MP 2023

pour $w_i \in \mathcal{C}$ et $c = w_i$ ou encore pour $w_i = \text{I}$, il y a dans l'automate $\mathcal{A} \bowtie \mathbf{w}$ une transition $(i \cdot r + q, c, (i + 1) \cdot r + q')$.

- 20 – Dans cette question uniquement, on considère l'exemple avec $n = 4$, $\mathbf{w}_0 = \text{INBI} \in \overline{\mathcal{C}}^*$ et où \mathcal{A}_0 est l'automate défini par



Dessiner le déploiement $\mathcal{A}_0 \bowtie \mathbf{w}_0$. Marquer les états accessibles. Marquer les états co-accessibles, c'est-à-dire les états à partir desquels il existe un chemin vers l'état final.

- 21 – Soient \mathcal{A} un automate et $\mathbf{w} \in \mathcal{C}^*$ un mot. Écrire une fonction OCaml `accessible (a:automate) (w:couleur array) : bool array` dont la valeur de retour \mathbf{b} est un tableau de booléens tels que $\mathbf{b}[q]$ est vrai si et seulement si l'état q de $\mathcal{A} \bowtie \mathbf{w}$ est accessible.
- 22 – Calculer la complexité en temps de la fonction `accessible` définie à la question 21.
- 23 – Soient \mathcal{A} un automate à r états et $\mathbf{w} \in \mathcal{C}^*$ un mot de longueur n . Écrire une fonction `coaccessible (a:automate) (w:couleur array) : bool array` dont la valeur de retour \mathbf{b} est un tableau de booléens tels que $\mathbf{b}[q]$ est vrai si et seulement si l'état q de $\mathcal{A} \bowtie \mathbf{w}$ est co-accessible.

Définition : Soient \mathcal{A} un automate à r états et $\mathbf{w} \in \overline{\mathcal{C}}^*$ un mot de longueur n . Nous supposons qu'il existe un mot de longueur n reconnu par l'automate $\mathcal{A} \bowtie \mathbf{w}$. Nous rappelons que l'automate émondé de $\mathcal{A} \bowtie \mathbf{w}$ est la copie de l'automate duquel ont été retirées toutes les transitions depuis ou vers des états qui ne sont pas à la fois accessibles et co-accessibles (nous ne chassons pas les états inutiles et conservons la numérotation des états). Nous notons $\widehat{\Delta}$ l'ensemble des transitions restantes dans l'automate émondé. Nous notons, pour tout indice i compris entre 0 et $n - 1$, l'ensemble des étiquettes

$$H_i = \left\{ c \in \mathcal{C}; (q, c, q') \in \widehat{\Delta} \cap \llbracket i \cdot r, (i + 1) \cdot r - 1 \rrbracket \times \mathcal{C} \times \llbracket (i + 1) \cdot r, (i + 2) \cdot r - 1 \rrbracket \right\}.$$

Nous posons, pour tout entier i compris entre 0 et $n - 1$,

$$y_i = \begin{cases} \text{B} & \text{si } H_i = \{\text{B}\} \\ \text{N} & \text{si } H_i = \{\text{N}\} \\ \text{I} & \text{si } H_i = \{\text{B}, \text{N}\}. \end{cases}$$

Nous appelons *mot projeté du déploiement* $\mathcal{A} \bowtie \mathbf{w}$ le mot $\mathbf{y} = y_0 y_1 \dots y_{n-1} \in \overline{\mathcal{C}}^*$.

- 24 – Avec les notations du paragraphe qui précède et lorsque l'automate \mathcal{A} est l'automate dessiné à la question 18, vérifier que le mot projeté $\mathbf{y} \in \overline{\mathcal{C}}^*$ est la plus grande extension commune du mot $\mathbf{w} \in \overline{\mathcal{C}}^*$ par rapport à l'indication τ (cf. définition p. 7).

Épreuve d'informatique d'option MP 2023

- 25 – Écrire une fonction projete (a:automate) (w:couleur array) : couleur array dont la valeur de retour est le mot projeté du déploiement $\mathcal{A} \bowtie w$.

Nous rappelons la *formule de Stirling* :

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n.$$

- 26 – Comparer la complexité en temps de la fonction projete (question 25) avec un calcul direct de la plus grande extension commune par énumération de l'ensemble $[w]_\tau$. Argumenter en s'appuyant sur la question 16.

Nous nous donnons une fonction OCaml pgec_lig (h:hanjie) (p:presolution) (x:int) : presolution option, et respectivement pgec_col (h:hanjie) (p:presolution) (y:int) : presolution option, qui étend la ligne x , respectivement la colonne y , par la plus grande extension commune à l'image de la question 25 ou bien renvoie None si la plus grande extension commune n'est pas définie.

- 27 – Écrire un extenseur etend_nontrivial (h:hanjie) (p:presolution) (z:int) (c:couleur) : presolution option qui modifie la couleur de p au numéro z et applique les fonctions pgec_lig et pgec_col aussi longtemps que cela permet de faire progresser la présolution. Il est demandé de détailler la stratégie de l'extenseur avant d'en fournir le code.

A. Annexe : aide à la programmation en OCaml

Opérations sur les tableaux : Le module Array offre les fonctions suivantes :

- `length : 'a array → int`
Return the length (number of elements) of the given array.
- `make : int → 'a → 'a array`
`Array.make n x` returns a fresh array of length n , initialized with x . All the elements of this new array are initially physically equal to x (in the sense of the `=` predicate). Consequently, if x is mutable, it is shared among all elements of the array, and modifying x through one of the array entries will modify all other entries at the same time.
- `make_matrix : int → int → 'a → 'a array array`
`Array.make_matrix dimx dimy e` returns a two-dimensional array (an array of arrays) with first dimension $dimx$ and second dimension $dimy$. All the elements of this new matrix are initially physically equal to e . The element (x, y) of a matrix m is accessed with the notation `m.(x).(y)`.
- `init : int → (int → 'a) → 'a array`
`Array.init n f` returns a fresh array of length n , with element number i initialized to the result of $f(i)$. In other terms, `init n f` tabulates the results of f applied to the integers 0 to $n - 1$.
- `copy : 'a array → 'a array`
`Array.copy a` returns a copy of a , that is, a fresh array containing the same elements as a .
- `mem : 'a → 'a array → bool`
`mem a l` is true if and only if a is structurally equal to an element of l (i.e. there is an x in l such that `compare a x = 0`).
- `for_all : ('a → bool) → 'a array → bool`
`Array.for_all f [|a1; ...; an|]` checks if all elements of the array satisfy the predicate f . That is, it returns `(f a1) && (f a2) && ... && (f an)`.
- `exists : ('a → bool) → 'a array → bool`
`Array.exists f [|a1; ...; an|]` checks if at least one element of the array satisfies the predicate f . That is, it returns `(f a1) || (f a2) || ... || (f an)`.
- `map : ('a → 'b) → 'a array → 'b array`
`Array.map f a` applies function f to all the elements of a , and builds an array with the results returned by $f : [| f a.(0); f a.(1); ...; f a.(length a - 1) |]$.
- `iter : ('a → unit) → 'a array → unit`
`Array.iter f a` applies function f in turn to all the elements of a . It is equivalent to `f a.(0); f a.(1); ...; f a.(length a - 1); ()`.

D'après <https://v2.ocaml.org/api/Array.html>

Opérations sur les tables de hachage : Le module Hashtbl offre les fonctions suivantes :

- `('a, 'b) Hashtbl.t`
The type of hash tables from type ' a to type ' b .
- `create : int → ('a, 'b) t`
`Hashtbl.create n` creates a new, empty hash table, with initial size n . For best results, n should be on the order of the expected number of elements that will be in the table. The table grows as needed, so n is just an initial guess.
- `add : ('a, 'b) t → 'a → 'b → unit`
`Hashtbl.add tbl key data` adds a binding of key to $data$ in table tbl .
- `remove : ('a, 'b) t → 'a → unit`
`Hashtbl.remove tbl x` removes the current binding of x in tbl , restoring the previous binding if it exists. It does nothing if x is not bound in tbl .
- `mem : ('a, 'b) t → 'a → bool`
`Hashtbl.mem tbl x` checks if x is bound in tbl .
- `iter : ('a → 'b → unit) → ('a, 'b) t → unit`
`Hashtbl.iter f tbl` applies f to all bindings in table tbl . f receives the key as first argument, and the associated value as second argument. Each binding is presented exactly once to f .
- `find_opt : ('a, 'b) t → 'a → 'b option`
`Hashtbl.find_opt tbl x` returns the current binding of x in tbl , or `None` if no such binding exists.

D'après <https://v2.ocaml.org/api/Hashtbl.html>

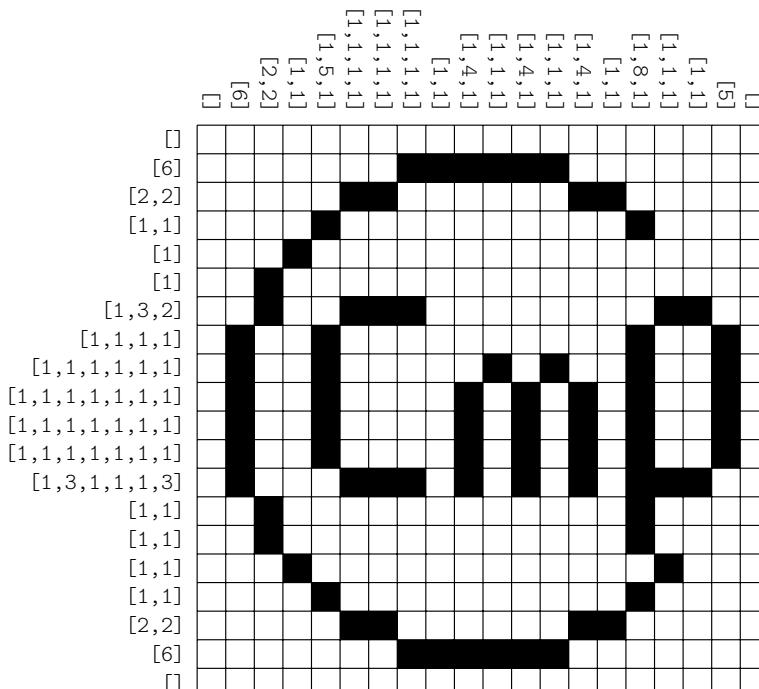
Épreuve d'informatique d'option MP 2023

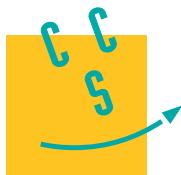
B. Annexe : règles de la déduction naturelle

Dans les tableaux suivants, la lettre Δ désigne un ensemble de formules de logique ; les lettres A , B et C désignent des formules de logique.

Axiome	
$\frac{}{\Delta, A \vdash A}$ (ax)	

	Introduction	Élimination
\rightarrow	$\frac{\Delta, A \vdash B}{\Delta \vdash A \rightarrow B}$ (\rightarrow i)	$\frac{\Delta \vdash A \quad \Delta \vdash A \rightarrow B}{\Delta \vdash B}$ (\rightarrow e)
\wedge	$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \wedge B}$ (\wedge i)	$\frac{\Delta \vdash A \wedge B}{\Delta \vdash A}$ (\wedge e) $\frac{\Delta \vdash A \wedge B}{\Delta \vdash B}$ (\wedge e)
\vee	$\frac{\Delta \vdash A}{\Delta \vdash A \vee B}$ (\vee i) $\frac{\Delta \vdash B}{\Delta \vdash A \vee B}$ (\vee i)	$\frac{\Delta \vdash A \vee B \quad \Delta, A \vdash C \quad \Delta, B \vdash C}{\Delta \vdash C}$ (\vee e)
\neg	$\frac{\Delta, A \vdash B \quad \Delta, A \vdash \neg B}{\Delta \vdash \neg A}$ (\neg i)	$\frac{\Delta \vdash A \quad \Delta \vdash \neg A}{\Delta \vdash B}$ (\neg e)





Informatique

MPI

2023

CONCOURS CENTRALE-SUPÉLEC

4 heures

Calculatrice autorisée

Le sujet est composé de deux parties indépendantes, la première utilise le langage C et la seconde le langage OCaml. Toutes deux traitent de graphes.

La première se concentre sur un problème classique, le problème du voyageur de commerce. Elle étudie d'abord sa complexité, en le comparant notamment au problème très proche du cycle et du chemin hamiltonien, puis en le réduisant à 3-SAT, et en donnant un résultat sur l'approximabilité du problème. On propose ensuite une heuristique, l'algorithme de Christofides, qui repose sur différentes notions : arbres couvrants, couplages, circuit eulérien. On montrera que, dans une variante du problème où les distances vérifient l'inégalité triangulaire, l'algorithme de Christofides est une approximation de facteur $3/2$. Dans cette première partie, les graphes seront représentés par des matrices d'adjacence en C.

La deuxième partie s'intéresse à des processus d'édition sur des arbres non racinés dont les feuilles sont étiquetées, qui sont notamment utilisés en bio-informatique pour représenter l'évolution des espèces. On étudiera en particulier l'espace induit sur ces arbres par ces processus d'édition et on représentera cet espace par un graphe. On montrera notamment que le graphe induit par l'un de ces processus d'édition possède un cycle hamiltonien. On utilisera pour cette partie le langage OCaml pour implémenter des arbres par une définition récursive et des graphes par liste d'adjacence.

I Problème du voyageur de commerce

On considère des graphes non orientés $G(V, E)$ où V est l'ensemble des sommets et E l'ensemble des arêtes. On notera $n = |V|$ le nombre de sommets.

Un *chemin* est une suite de sommets reliés par des arêtes. On dit qu'un chemin *passe* par un sommet si ce sommet appartient au chemin. Un *circuit* est un chemin qui commence et se termine au même sommet. Un *chemin hamiltonien* est un chemin qui passe une et une seule fois par chaque sommet du graphe. Un *circuit hamiltonien* est un circuit qui passe par chaque sommet une et une seule fois.

Le *problème du voyageur de commerce* consiste, étant donnée une liste de villes toutes reliées entre elles, à trouver le circuit le plus court qui passe une et une seule fois par chacune des villes.

Plus formellement, on considère un graphe complet non orienté, dont les arêtes sont étiquetées avec des nombres entiers strictement positifs, appelés *poids*, et on cherche le circuit passant par chacun des sommets du graphe qui minimise la somme des poids des arêtes. On appellera *poids d'un circuit* la somme des poids des arêtes empruntées par ce circuit. Une solution au problème du voyageur de commerce est un circuit hamiltonien de poids minimal.

Dans cette partie, on représente les graphes en C par des matrices d'adjacence. Le poids d'une arête est représenté par un `int`, une arête absente étant représentée par un 0. On représente un graphe par une structure de données avec deux attributs : son nombre de sommets `V` et un pointeur `adj` vers sa matrice d'adjacence de taille `V × V`. Les sommets sont associés aux entiers de 0 à `V-1`.

```
struct Graphe {
    int V;
    int* adj;
};
```

Pour accéder au poids de l'arête entre le sommet `i` et le sommet `j` du graphe `G`, on pourra utiliser l'expression `G.adj[i * G.V + j]`.

On pourra par la suite utiliser les deux fonctions suivantes, qui sont supposées travailler en temps constant :

```
struct Graphe alloue_graphe(int V) {
    int* adj = malloc(V * V * sizeof(int));
    struct Graphe g = { .V = V, .adj = adj };
    return g;
}

void libere_graphe(struct Graphe g) {
    free(g.adj);
}
```

La fonction `alloue_graphe` prend comme argument un nombre V et renvoie un graphe qui possède V sommets et dont la matrice d'adjacence est allouée sur le tas, grâce à la fonction `malloc`. La fonction `libere_graphe` libère la mémoire du tas occupée par la matrice d'adjacence d'un graphe dont on n'a plus besoin.

On définit également une structure `Chemin` qui représente un chemin par un attribut `longueur` et un attribut `l_sommets`, pointeur vers un tableau à `longueur` éléments. Si C est un chemin et k un entier naturel strictement plus petit que $C.\text{longueur}$, alors $C.\text{l_sommets}[k]$ est le k -ième sommet du chemin C . De même que pour les graphes, on définit aussi des fonctions permettant d'allouer un chemin et de libérer un chemin dont on n'a plus besoin, et qui sont supposées travailler en temps constant.

```
struct Chemin {
    int longueur;
    int* l_sommets;
};

struct Chemin alloue_chemin(int longueur) {
    int* l_sommets = malloc(longueur * sizeof(int));
    struct Chemin c = { .longueur = longueur, .l_sommets = l_sommets };
    return c;
}

void libere_chemin(struct Chemin c) {
    free(c.l_sommets);
}
```

I.A – Prise en main du problème et appartenance à NP

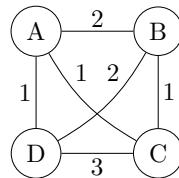


Figure 1 Exemple d'instance du problème du voyageur de commerce

Q 1. Donner une solution du problème du voyageur de commerce sur l'exemple de la figure 1 en précisant le poids du circuit trouvé.

Q 2. Donner le nombre de circuits hamiltoniens sur un graphe complet de n sommets. Donner un exemple de pondération pour que chacun de ces circuits ait un poids minimal pour le problème du voyageur de commerce.

Q 3. Écrire une fonction

```
int poids_chemin(struct Graphe g, struct Chemin c);
```

qui prend en arguments un graphe et un chemin et renvoie le poids de ce chemin. Donner la complexité de cette fonction.

Q 4. Le problème du voyageur de commerce est un problème d'optimisation ; à l'aide d'un seuil, transformer ce problème en un problème de décision. Montrer que ce nouveau problème, que nous appellerons par la suite « problème de décision du voyageur de commerce », appartient à la classe de complexité NP.

I.B – Étude de la complexité

Le *problème du chemin hamiltonien* consiste, étant donné un graphe non orienté et deux sommets a et b , à déterminer s'il existe un chemin hamiltonien commençant en a et finissant en b . Le *problème du chemin hamiltonien orienté* consiste, étant donné un graphe orienté et deux sommets a et b , à déterminer s'il existe un chemin hamiltonien commençant en a et finissant en b . Le *problème du circuit hamiltonien* consiste, étant donné un graphe non orienté, à déterminer s'il existe un circuit hamiltonien dans ce graphe.

I.B.1) NP-complétude

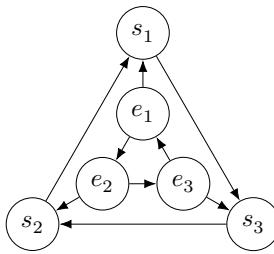
Q 5. Montrer que le problème du chemin hamiltonien se réduit au problème du circuit hamiltonien.

Q 6. Montrer que le problème du circuit hamiltonien se réduit au problème de décision du voyageur de commerce.

Q 7. Montrer que le problème du chemin hamiltonien orienté se réduit au problème du chemin hamiltonien.

Le problème 3-SAT consiste à déterminer la satisfiabilité d'une formule logique sous forme normale conjonctive avec exactement 3 littéraux : pour n clauses C_i et m variables y_k , déterminer s'il existe une valuation des y_k qui permette de rendre vraie la formule $C_1 \wedge C_2 \wedge \dots \wedge C_n$ avec $C_i = x_{i,1} \vee x_{i,2} \vee x_{i,3}$ sachant que, pour chaque i et chaque p , il existe un k tel que $x_{i,p} = y_k$ ou $x_{i,p} = \neg y_k$. On admet que le problème 3-SAT est NP-complet.

On va maintenant montrer que 3-SAT se réduit au problème du chemin hamiltonien orienté.

**Figure 2** Graphe A

Q 8. On considère le graphe A (figure 2). Montrer qu'il existe un chemin entrant par e_1 et passant par tous les sommets pour ressortir en s_1 . Puis qu'il existe un chemin entrant par e_1 et sortant par s_1 et un chemin entrant par e_2 et sortant par s_2 , tels que chaque sommet soit visité par un et un seul des deux chemins. Même question, mais avec trois chemins, pour e_1, e_2, e_3 et s_1, s_2, s_3 .

On se donne une instance du problème 3-SAT, pour n clauses C_i et m variables $y_k : C_1 \wedge C_2 \wedge \dots \wedge C_n$ avec $C_i = x_{i,1} \vee x_{i,2} \vee x_{i,3}$ et $x_{i,p} = y_k$ ou $x_{i,p} = \neg y_k$. On veut donc savoir s'il existe une valuation des y_k pour que la formule soit vraie.

On construit alors le graphe orienté G de la manière suivante :

- pour chaque variable y_k on crée un sommet v_k ;
- on ajoute un sommet supplémentaire v_{m+1} ;
- pour chaque clause C_i on ajoute une copie du graphe A , notée A_i ;
- pour chaque variable y_k , on note $C_{k_1}, \dots, C_{k_\ell}$ les clauses dans lesquelles y_k apparaît en positif. On relie alors v_k à A_{k_1} par un arc allant de v_k vers e_p dans A_{k_1} lorsque y_k est en position p dans C_{k_1} c'est-à-dire lorsque $C_{k_1} = x_{k_1,1} \vee x_{k_1,2} \vee x_{k_1,3}$ et $x_{k_1,p} = y_k$. On relie ensuite la sortie s_p de A_{k_1} à l'entrée de A_{k_2} correspondant à la position de y_k dans C_{k_2} et ainsi de suite jusqu'au dernier dont on relie la sortie à v_{k+1} . On appelle G_k^+ le sous-graphe constitué du sommet v_k , des graphes $A_{k_1}, A_{k_2}, \dots, A_{k_\ell}$ et du sommet v_{k+1} , ainsi que des arcs que l'on vient d'ajouter entre eux en considérant y_k et les clauses dans lesquelles il apparaît positivement ;
- on crée de même des arcs pour chaque variable y_k et chaque clause dans laquelle y_k apparaît en négatif. On note G_k^- , le sous-graphe correspondant entre v_k et v_{k+1} .

Q 9. Montrer que pour toute valuation de la formule il existe un chemin hamiltonien orienté de v_1 à v_{m+1} dans le graphe G .

Q 10. Montrer, en une dizaine de lignes au maximum, que pour chaque chemin hamiltonien orienté de v_1 à v_{m+1} il existe bien une valuation.

Q 11. En déduire que le problème du circuit hamiltonien et le problème de décision du voyageur de commerce sont NP-complets.

I.B.2) Approximation

Soit $\varepsilon > 0$, on va montrer que, si $P \neq NP$, il n'existe pas de $1 + \varepsilon$ approximation pour le problème du voyageur de commerce. Un algorithme est une $1 + \varepsilon$ approximation à un problème d'optimisation d'un poids p lorsque la solution proposée de poids p se compare toujours à la solution optimale p^* par $p < (1 + \varepsilon)p^*$.

Soit G un graphe non orienté à n sommets, on considère le graphe complet G' , de mêmes sommets que G , avec des poids aux arêtes obtenus en donnant un poids de 1 aux arêtes de G et un poids de $n(1 + \varepsilon) + 1$ aux arêtes qui ne sont pas dans G .

Q 12. Montrer que si G possède un circuit hamiltonien, alors G' possède un circuit de poids n .

Q 13. Montrer que si G ne possède pas de circuit hamiltonien alors toute solution pour l'instance de voyageur de commerce est de poids au moins $n(2 + \varepsilon)$.

Q 14. En déduire que, si $P \neq NP$, il n'existe pas de $1 + \varepsilon$ approximation au problème du voyageur de commerce.

I.C – Algorithme de Christofides

On va proposer une heuristique pour le problème du voyageur de commerce, l'algorithme de Christofides, et on va montrer que, sous certaines conditions sur le graphe en entrée, cette heuristique constitue un algorithme d'approximation. L'algorithme prend en argument un graphe G et procède comme suit :

- calculer un arbre couvrant de poids minimal T de G ;
- en notant I l'ensemble des sommets de degré impair dans T , calculer un couplage parfait M de poids minimum dans le sous-graphe de G induit par les sommets de I , $G|_I$;
- construire H le multigraphe ayant pour sommet les sommets de G et comme arêtes les arêtes de M et celles de T ;

- trouver un cycle eulérien dans H ;
 - transformer le cycle eulérien en circuit hamiltonien en supprimant les éventuels sommets vus plusieurs fois.
- Dans la suite, on étudie plus précisément certaines étapes de cet algorithme, avant de proposer une implémentation de cet algorithme.

I.C.1) Arbre couvrant

Un arbre couvrant est un sous-graphe connexe sans cycle d'un graphe avec les mêmes sommets. On appelle poids de l'arbre la somme des poids des arêtes de cet arbre. On rappelle que l'algorithme de Kruskal est un algorithme glouton qui vise à construire un arbre couvrant de poids minimal en considérant les arêtes par poids croissant et en ajoutant chaque arête si elle ne crée pas de cycle.

Pour cela, on va représenter une arête par une structure de données avec trois attributs : $s1$ et $s2$ donnent les sommets reliés par l'arête et p son poids :

```
struct Arete {
    int s1;
    int s2;
    int p;
};
```

Q 15. Écrire une fonction

```
struct Arete* liste_aretes(struct Graphe g);
```

qui prend en argument un graphe complet g et alloue et renvoie un tableau contenant les $\frac{g.V \times (g.V-1)}{2}$ arêtes du graphe.

On dispose d'une fonction

```
void tri_aretes(struct Arete a[], int k);
```

qui prend en arguments un tableau d'arêtes et sa longueur et trie le tableau par ordre croissant de poids. La complexité de cette fonction est en $\mathcal{O}(k \ln(k))$.

Q 16. Écrire une fonction

```
struct Graphe kruskal(struct Graphe g);
```

implémentant l'algorithme de Kruskal qui renvoie un graphe représentant l'arbre couvrant de poids minimal du graphe donné en argument. On mettra des 0 lorsque l'arête est absente et des 1 lorsqu'elle est présente. Donner la complexité de la fonction `kruskal`.

Q 17. Montrer la correction de cet algorithme, c'est-à-dire l'optimalité de la solution proposée pour le problème d'arbre couvrant de poids minimal.

I.C.2) Couplage

On appelle couplage d'un graphe un ensemble d'arêtes qui n'ont pas de sommets en commun. Un couplage est parfait si tous les sommets du graphe appartiennent à une arête du couplage.

Q 18. Écrire une fonction

```
int degre(struct Graphe g, int i);
```

qui prend en arguments un graphe et l'indice d'un sommet et renvoie le degré de ce sommet.

Q 19. Écrire une fonction

```
int* sommets_imparis(struct Graphe g, int* nb_sommets);
```

qui prend en arguments un graphe g et un pointeur vers un entier. Cette fonction alloue sur le tas un tableau, le remplit avec les numéros des sommets de degré impair et le renvoie. Par ailleurs, elle renseigne l'entier pointé par `nb_sommets` avec le nombre des sommets de degré impair.

Q 20. Montrer l'existence d'un couplage parfait de poids minimal dans $G|_I$.

On dispose des deux fonctions suivantes :

```
struct Graphe graphe_induit(struct Graphe g, int nb_sommets, int* liste_sommets);
struct Graphe couplage(struct Graphe g);
```

La fonction `graphe_induit` renvoie le graphe induit dans un graphe g donné par un nombre et un tableau de sommets comme ceux de la question 19.

La fonction `couplage` renvoie un couplage parfait de poids minimal s'il existe, sous la forme d'un graphe représentant ce couplage, avec des 0 lorsque l'arête est absente et 1 lorsque l'arête est présente.

On supposera ces deux fonctions de complexité polynomiale.

I.C.3) Cycle eulérien

On appelle cycle eulérien un circuit qui passe par chaque arête une et une unique fois. On admet qu'un graphe possède un cycle eulérien si et seulement les degrés des sommets de ce graphe sont pairs et que le graphe est connexe. Un multigraphe est un graphe dans lequel il peut exister plusieurs arêtes reliant un même couple de sommets.

Q 21. Montrer que le multigraphe H , défini dans l'introduction de la sous-partie I.C, possède un cycle eulérien.

On dispose des trois fonctions suivantes :

```
struct Multigraphe multigraphe(struct Graphe g1, struct Graphe g2);
struct Chemin eulerien(struct Multigraphe h);
void libere_multigraphe(struct Multigraphe h);
```

La fonction `multigraphe` prend en arguments deux graphes sur les mêmes sommets et renvoie le multigraphe obtenu en considérant les arêtes des deux graphes. La fonction `eulerien` renvoie un circuit eulérien d'un multigraphe sous la forme d'un chemin. La fonction `libere_multigraphe` libère la mémoire du tas utilisée par un multigraphe. Toutes trois sont supposées de complexité polynomiale.

Q 22. Écrire une fonction

```
struct Chemin euler_to_hamilton(struct Chemin c);
```

qui transforme un cycle eulérien c du multigraphe H en un circuit hamiltonien du graphe G en supprimant les doublons, et renvoie le chemin représentant la suite des sommets.

I.C.4) Implémentation

Q 23. Sur l'exemple de la figure 1, réaliser les différentes étapes de l'algorithme. On ne demande pas de détailler les étapes pour trouver un couplage et un cycle eulérien.

Q 24. Écrire une fonction

```
struct Chemin christofides(struct Graphe g);
```

qui implémente l'algorithme de Christofides, en prenant soin de libérer la mémoire allouée sur le tas qui n'est plus utilisée.

Q 25. Justifier que la fonction `christofides` renvoie bien un circuit hamiltonien.

Q 26. Montrer que la fonction `christofides` est de complexité polynomiale.

I.C.5) Preuve de l'approximation

On va maintenant montrer que cet algorithme est une $3/2$ -approximation pour le problème du voyageur de commerce, dans le cas où les poids des arêtes vérifient l'inégalité triangulaire, c'est-à-dire que pour tout sommet u, v, w les poids des arêtes vérifient, en notant $c_{i,j}$ le poids de l'arête entre des sommets i et j : $c_{u,v} \leq c_{u,w} + c_{w,v}$.

On note U une solution optimale et $c(U)$ son poids.

Q 27. Montrer que $c(T) \leq c(U)$, où $c(T)$ est le coût de l'arbre couvrant minimal.

Q 28. Montrer que $c(M) \leq 0.5c(U)$ où $c(M)$ est le coût du couplage parfait minimal.

Q 29. Montrer que la solution construite par l'algorithme est une $3/2$ -approximation de U .

Q 30. Sans la supposition de l'inégalité triangulaire, cette solution est-elle toujours une $3/2$ -approximation ? Proposer un contre-exemple ou une justification.

II Espaces d'arbres

Dans cette partie on considère des arbres binaires non racinés avec des feuilles étiquetées. Un *arbre binaire non raciné* est un graphe connexe sans cycle dont les noeuds sont de degrés 1 ou 3. Un *arbre binaire raciné* est un graphe connexe sans cycle dont les noeuds sont de degrés 1 ou 3 sauf exactement un noeud qui est de degré 2. Les noeuds de degré 1 sont appelés des *feuilles*, les noeuds de degré 2 ou 3 des *noeuds internes* et l'unique noeud de degré 2 d'un arbre binaire raciné est appelé sa *racine*. Un graphe réduit à un unique noeud est considéré comme un arbre raciné dont le noeud est à la fois feuille et racine. Les arêtes d'un arbre sont appelées des *branches*, les branches reliant des sommets de degré 2 ou 3 *branches internes*. On étiquette ensuite les feuilles d'un arbre à n feuilles par des entiers distincts entre 1 et n . On s'intéresse à des opérations d'édition qui transforment un arbre à n feuilles en un second arbre à n feuilles.

La première opération, notée SPR, pour *Subtree Prune and Regraft* (ou découpe et greffe d'un sous-arbre), prend en entrée 2 branches e et f de l'arbre. La branche e est supprimée, créant deux arbres racinés par les sommets à chaque extrémité de e , notons ces arbres T_1 (qui contient la branche f) et T_2 (qui ne la contient pas). On divise ensuite la branche f pour créer un nouveau noeud v de degré 2. Le sous-arbre raciné T_2 est alors rattaché par sa racine à v par une nouvelle branche ; v devient alors de degré 3. On obtient alors un arbre raciné, avec une racine de degré 2, que l'on supprime en reliant directement ses deux voisins pour obtenir un arbre non raciné.

On s'intéresse également à une autre opération, notée NNI, pour *Nearest Neighbor Interchange* (ou échange entre plus proches voisins). Cette opération prend une branche interne de l'arbre e et échange entre eux deux des quatre sous-arbres incidents à cette branche. C'est-à-dire, en notant a et b les extrémités de la branche interne e , a a deux sommets voisins qui ne sont pas b , a_1 et a_2 , et de même pour b , b_1 et b_2 . On choisit alors un des a_i et un des b_j à échanger, par exemple a_1 et b_1 , on supprime l'arête reliant a à a_1 et b à b_1 et on relie a à b_1 et b à a_1 .

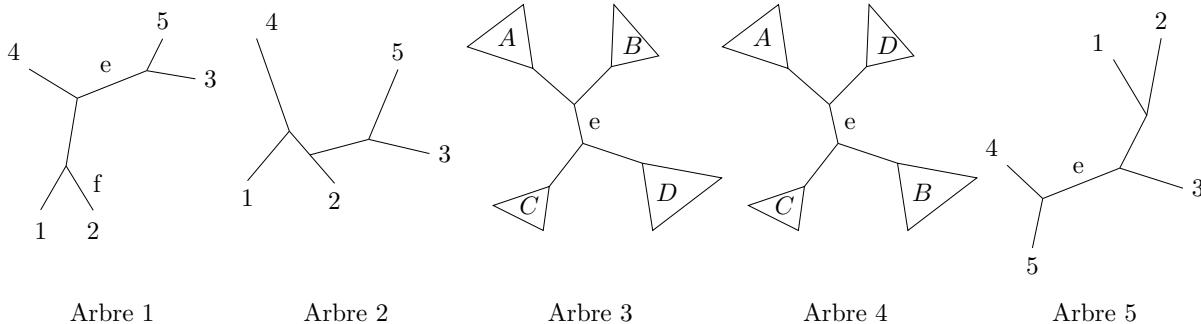


Figure 3 Exemples d'arbres non racinés

La figure 3 présente quelques exemples d'arbres non racinés :

- l'arbre 1 est un arbre binaire non raciné avec ses feuilles étiquetées ;
- l'arbre 2 est obtenu à partir de l'arbre 1 par un SPR en coupant la branche e et en la greffant sur la branche f ;
- dans l'arbre 3, où les triangles représentent des sous-arbres racinés, la branche interne e sépare 4 sous-arbres racinés A, B, C et D ;
- l'arbre 4 est obtenu à partir de l'arbre 3 par un mouvement NNI sur la branche e en échangeant les sous-arbres B et D ;
- l'arbre 5 est obtenu à partir de l'arbre 1 par un NNI sur la branche e en échangeant le sous-arbre contenant la feuille 5 et le sous-arbre contenant les feuilles 1 et 2.

On étudie l'espace des arbres binaires non racinés étiquetés de n feuilles $B(n)$ suivant ces deux processus d'édition. On définit $G_{\text{NNI}}(n)$ le graphe dont les sommets sont les arbres de $B(n)$ et dans lequel une arête relie deux arbres si l'on peut passer de l'un à l'autre par un mouvement NNI. On définit de même $G_{\text{SPR}}(n)$ avec les mouvements SPR. On va notamment s'intéresser à la structure de $G_{\text{NNI}}(5)$, écrire un programme permettant de construire $G_{\text{NNI}}(n)$ et montrer que $G_{\text{SPR}}(n)$ est hamiltonien, c'est-à-dire qu'il possède un circuit hamiltonien.

II.A – Prise en main

Q 31. Montrer que l'on peut passer de l'arbre 1 à l'arbre 2 par une opération NNI. Dessiner tous les voisins de l'arbre 1 dans $G_{\text{NNI}}(5)$.

Q 32. Donner le nombre de branches et de noeuds d'un arbre de $B(n)$, pour $n \geq 2$.

Q 33. On note $RB(n)$ l'ensemble des arbres binaires racinés à n feuilles étiquetées. Montrer que $|RB(n)| = (2n-3)|B(n)|$ et que $|B(n)| = |RB(n-1)|$. En déduire le nombre d'arbres binaires $B(n)$ en fonction de n .

Q 34. Tracer $G_{\text{NNI}}(4)$.

Q 35. Combien de voisins un arbre de $B(n)$ a-t-il dans $G_{\text{NNI}}(n)$?

On appelle arbre chenille un arbre binaire non raciné dans lequel il existe un chemin passant une et une seule fois par chaque noeud interne.

Q 36. Montrer que l'on peut passer de tout arbre de $B(n)$ à un arbre chenille, en effectuant seulement des mouvements NNI. En déduire que $G_{\text{NNI}}(n)$ est connexe.

Q 37. Montrer que tout mouvement SPR peut se décomposer en mouvements NNI, et que tous les mouvements NNI sont des mouvements SPR. En déduire que $G_{\text{SPR}}(n)$ est connexe.

II.B – Étude de $G_{\text{NNI}}(5)$

Q 38. Combien de sommets possède $G_{\text{NNI}}(5)$ et quel est le degré de ces sommets ?

Q 39. Montrer que tout arbre de $G_{\text{NNI}}(5)$ fait partie de cycles de longueur 5, de deux cycles de longueurs 3, et que pour tout arbre deux arbres sont à distance 3 de cet arbre.

Q 40. Tracer $G_{\text{NNI}}(5)$.

II.C – Construction de $G_{\text{NNI}}(n)$

On va écrire en OCaml un programme qui construit $G_{\text{NNI}}(n)$. Pour représenter les arbres binaires (racinés ou non), nous allons utiliser la structure de données suivante :

```
type arbre = Feuille of int
           | Noeud of arbre * arbre ;;
```

Un arbre raciné est alors représenté par sa racine, qui peut être soit une feuille soit un nœud interne ayant un sous-arbre gauche et un sous-arbre droit.

Un arbre non raciné à n feuilles est représenté à partir d'une racine fictive ajoutée entre la feuille d'étiquette n et le nœud auquel elle est reliée.

Enfin, pour assurer l'unicité de la représentation informatique d'un arbre, et ainsi simplifier les programmes, on adopte la convention, pour chaque nœud interne, que la plus grande des étiquettes du sous-arbre droit est supérieure aux étiquettes du sous-arbre gauche.

Ainsi, l'arbre 1 de la figure 3 est représenté par :

```
Noeud (Noeud (Feuille 3,
           Noeud (Noeud (Feuille 1,
                           Feuille 2),
                           Feuille 4)),
           Feuille 5);;
```

On représente les graphes par liste d'adjacence, plus précisément par une liste de couples dont le premier élément est le nom d'un sommet et le second la liste d'adjacence de ce sommet.

```
type graphe = (arbre * arbre list) list;;
```

Q 41. Écrire une fonction `feuilles` qui prend en argument un arbre et renvoie la liste des étiquettes de ses feuilles.

Q 42. Écrire une fonction `degres` qui prend en argument un graphe implémenté par liste d'adjacence et renvoie la liste des degrés de ces noeuds.

Q 43. Écrire une fonction `egaux` qui teste si deux arbres sont égaux, au sens des arbres binaires non racinés étiquetés. Justifier que cette fonction est correcte.

Q 44. Étant donnés une liste d'arbres et un arbre, écrire une fonction `appartient` qui teste si l'arbre fait partie de la liste.

Q 45. En remarquant que parmi les 4 sous-arbres à échanger pour un mouvement de NNI autour d'une branche interne, on peut en choisir un qui restera fixe, écrire une fonction `voisinsNNI` qui prend en argument un arbre et renvoie tous les arbres que l'on peut obtenir à partir de celui-ci par un mouvement NNI.

Q 46. Écrire une fonction `chenille` qui prend en argument un entier n et renvoie l'arbre chenille dans lequel les feuilles sont rangées de 1 à n .

Q 47. Écrire une fonction `insere` qui prend en arguments un arbre et un graphe et ajoute l'arbre à la liste des sommets du graphe.

Q 48. Écrire une fonction `relie` qui prend en arguments un graphe et deux arbres et qui rajoute au graphe une arête reliant les deux sommets, si elle n'est pas déjà présente.

Q 49. Écrire une fonction `grapheNNI` qui prend en argument un entier n et renvoie $G_{\text{NNI}}(n)$, en implémentant le graphe avec des listes d'adjacences. Justifier la correction et la terminaison de votre programme.

II.D – $G_{\text{SPR}}(n)$ est hamiltonien

On va démontrer par récurrence que $G_{\text{SPR}}(n)$ est hamiltonien, c'est-à-dire qu'il possède un circuit hamiltonien, comme défini en première partie.

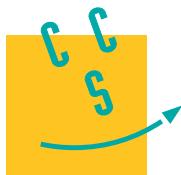
On note S_i l'ensemble des arbres de taille $n + 1$ obtenu à partir d'un arbre t_i de taille n en ajoutant une feuille étiquetée $n + 1$ à une des branches de t_i . Pour ajouter une feuille à une branche, on supprime la branche et on crée un nouveau nœud, relié à la nouvelle feuille et aux deux nœuds qui étaient reliés par la branche.

Q 50. Montrer que les sous-graphes $G_{\text{SPR}}(n + 1)|_{S_i}$, induits dans $G_{\text{SPR}}(n + 1)$ par les sommets de S_i sont complets.

Q 51. Soient t_i et t_j deux sommets de $G_{\text{SPR}}(n)$ reliés entre eux, montrer qu'il existe des arêtes entre S_i et S_j dans $G_{\text{SPR}}(n + 1)$.

Q 52. Démontrer par récurrence que $G_{\text{SPR}}(n)$ est hamiltonien.

• • • FIN • • •



Option informatique

MP

2023

CONCOURS CENTRALE-SUPÉLEC

4 heures

Calculatrice autorisée

Ce sujet comporte quatre parties.

La partie I est totalement indépendante des suivantes et étudie des transformations sur des langages.

Les autres parties s'intéressent à des structures de données représentant des ensembles d'entiers naturels E contenus dans $\llbracket 0, N-1 \rrbracket$, « denses » au sens où $|E|$ est du même ordre de grandeur que N . On cherche à optimiser le temps d'exécution des opérations de test d'appartenance, d'insertion ou de suppression d'un élément, de calcul du minimum, de calcul de l'élément de E immédiatement supérieur à x (qu'on appellera successeur de x dans E , même si x n'appartient pas à E) ou d'opérations ensemblistes telles que l'union.

Dans la partie II, on étudie des structures ordinaires dont on identifie le défaut. Dans la partie III, on étudie une structure simple d'arbre complet modélisant des parties de $\llbracket 0, 2^p - 1 \rrbracket$ et dans la partie IV, on implémente des arbres de van Emde Boas qui, sur des parties E de $\llbracket 0, 2^{2^p} - 1 \rrbracket$, donnent des opérations en temps quasi-constant.

I Langages et automates

Dans cette partie, on s'intéresse à des transformations sur des langages définis sur un alphabet à deux lettres $X = \{a, b\}$. On note $L_1 | L_2$ la réunion de deux langages L_1 et L_2 et ε le mot vide (ou le langage réduit au mot vide, suivant le contexte). Les automates considérés sont des quadruplets (Q, I, F, Δ) où Q est l'ensemble des états de l'automate, I l'ensemble de ses états initiaux, F l'ensemble de ses états finals et $\Delta \subset Q \times X \times Q$ l'ensemble de ses transitions.

Soit L et K deux langages. On définit le langage noté $L \triangleright K$ par

$$L \triangleright K = \left\{ uvw \mid (u, v, w) \in (X^*)^3, uw \in K, v \in L \right\}$$

Q 1. Déterminer les langages suivants :

$$a^* \triangleright b^* \quad (ba)^* \triangleright (ab)^* \quad a^* \triangleright \left\{ a^p b a^q \mid (p, q) \in \mathbb{N}^2, p \leq q \right\}$$

Q 2. Soit L , K_1 et K_2 trois langages. Exprimer à l'aide de $L \triangleright K_1$, $L \triangleright K_2$, K_1 et K_2 les langages suivants :

$$L \triangleright (K_1 | K_2) \quad L \triangleright (K_1 \cdot K_2) \quad L \triangleright (K_1^*)$$

On ne justifiera que la formule portant sur l'étoile de Kleene.

Q 3. Montrer que si L et K sont deux langages réguliers, $L \triangleright K$ est régulier.

Q 4. Donner l'exemple d'un langage régulier L et d'un langage non régulier K tel que $L \triangleright K$ soit régulier. On justifiera en détail que le langage K proposé n'est pas régulier.

Q 5. À l'aide d'automates reconnaissant des langages réguliers L et K et d' ε -transitions, construire un automate à ε -transitions reconnaissant $L \triangleright K$. On expliquera la construction puis on formalisera l'automate correspondant. Il est inutile de justifier qu'il convient.

On définit sur X^* l'application σ par :

$$\begin{cases} \sigma(\varepsilon) = \varepsilon \\ \forall u \in X^*, \forall x \in X, \quad \sigma(ux) = xu \end{cases}$$

On note alors pour tout langage L , \widehat{L} le langage défini par :

$$\widehat{L} = \bigcup_{k=0}^{+\infty} \sigma^k(L)$$

Q 6. Pour chacun des langages L suivants, déterminer $\sigma(L)$ puis \widehat{L} :

$$L_1 = (ab)^* \quad L_2 = a^*(ba|b)$$

Q 7. Soit L un langage régulier et $\mathcal{A} = (Q, I, F, \Delta)$ un automate le reconnaissant. Pour $(q, q') \in Q^2$, on note $L_{q,q'}$ l'ensemble des mots qui étiquettent un chemin de l'état q à l'état q' dans \mathcal{A} . Exprimer $\sigma(L)$ à l'aide de langages $L_{q,q'}$ et du langage X . On portera une attention particulière au cas du mot vide et on justifiera la formule proposée.

- Q 8.** Montrer que si L est régulier, $\sigma(L)$ est régulier.
Q 9. Montrer que si L n'est pas régulier, $\sigma(L)$ ne l'est pas non plus.
Q 10. Montrer que si L est régulier, \widehat{L} est régulier. Étudier la réciproque.

II Représentations classiques d'ensembles

Dans cette partie, on implémente des ensembles par des structures connues. On note $|E|$ le cardinal d'un ensemble E .

II.A – Avec une liste triée

Q 11. Dans cette question uniquement, on implémente un ensemble d'entiers positifs par la liste de ses éléments, rangés **dans l'ordre croissant**. Écrire une fonction `succ_list` de signature `int list -> int -> int` prenant en arguments une liste d'entiers distincts dans l'ordre croissant et un entier x et renvoyant le successeur de x dans la liste, c'est-à-dire le plus petit entier strictement supérieur à x de la liste (-1 si cela n'existe pas). Donner sa complexité dans le pire cas.

II.B – Avec un vecteur trié

Soit N un entier naturel strictement positif, fixé pour toute cette partie. On choisit de représenter un ensemble d'entiers E de cardinal $n \leq N$ par un tableau t de taille $N + 1$ dont la case d'indice 0 indique le nombre n d'éléments de E et les cases d'indices 1 à n contiennent les éléments de E rangés dans l'ordre croissant, les autres cases étant non significatives. Par exemple, le tableau `[|3;2;5;7;9;1;14|]` représente l'ensemble à 3 éléments $\{2, 5, 7\}$.

Q 12. Pour une telle implémentation d'un ensemble E , décrire brièvement des méthodes permettant de réaliser chacune des opérations ci-dessous (on ne demande pas d'écrire des programmes) et donner leurs complexités dans le pire cas :

- déterminer le maximum de E ;
- tester l'appartenance d'un élément x à E ;
- ajouter un élément x dans E (on suppose la taille du tableau suffisante et que x n'appartient pas à E).

Q 13. Par une méthode dichotomique, écrire une fonction `succ_vect` de signature `int array -> int -> int` prenant en arguments un tableau t codant un ensemble E comme ci-dessus et un entier x et renvoyant le successeur de x dans E (-1 si cela n'existe pas).

Q 14. Calculer la complexité dans le pire cas de la fonction `succ_vect` en fonction de n .

Q 15. Écrire une fonction `union_vect` de signature `int array -> int array -> int array` prenant en arguments deux tableaux t_1 et t_2 , de taille N , codant deux ensembles E_1 et E_2 et renvoyant le tableau correspondant à $E_1 \cup E_2$. On supposera que $|E_1 \cup E_2| \leq N$.

II.C – Avec un arbre binaire de recherche

On choisit maintenant de représenter un ensemble d'entiers à l'aide d'un arbre binaire de recherche, les noeuds étant étiquetés par les éléments de E . On définit le type

```
type abr = Nil | Noeud of int * abr * abr
```

Pour un tel arbre, pour **tout** noeud x , les étiquettes de tous les noeuds du sous-arbre gauche de x , s'il y en a, doivent être inférieures à l'étiquette de x et les étiquettes de tous les noeuds du sous-arbre droit de x , s'il y en a, doivent être supérieures à l'étiquette de x .

Par exemple, on peut représenter l'ensemble $\{2, 3, 5, 8, 13\}$ par l'arbre figure 1.

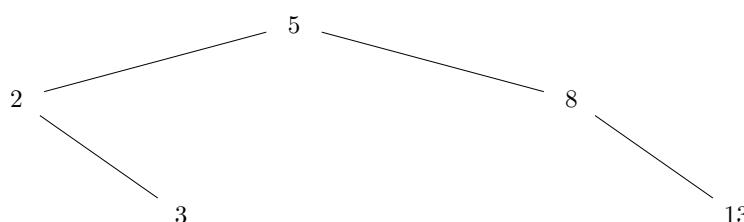


Figure 1 Un arbre binaire de recherche pour l'ensemble $\{2, 3, 5, 8, 13\}$

Q 16. Écrire une fonction `min_abr` de signature `abr -> int` prenant en argument un arbre binaire de recherche représentant un ensemble E et renvoyant son étiquette minimale (-1 si l'ensemble est vide).

Q 17. Écrire une fonction récursive `partitionne_abr` de signature `abr -> int -> (bool * abr * abr)` prenant en arguments un arbre binaire de recherche représentant un ensemble E et un entier x et renvoyant un

triplet (b, ag, ad) où b vaut `true` si x appartient à E et `false` sinon, ag est un arbre binaire de recherche codant les éléments de E strictement plus petits que x et ad un arbre binaire de recherche codant les éléments de E strictement plus grands que x .

Q 18. Écrire une fonction `insertion_abr` de signature `abr -> int -> abr` prenant en arguments un arbre binaire de recherche représentant un ensemble E et un entier x et renvoyant un arbre binaire de recherche associé à l'ensemble $E \cup \{x\}$ et de racine étiquetée par x . Calculer sa complexité dans le pire cas en fonction de l'arbre reçu puis en fonction de E .

Q 19. Écrire une fonction `union_abr` de signature `abr -> abr -> abr` prenant en arguments deux arbres binaires de recherche représentant deux ensembles E_1 et E_2 et renvoyant un arbre binaire de recherche associé à l'ensemble $E_1 \cup E_2$. On expliquera brièvement la méthode choisie.

III Représentation par arbres binaires complets

On considère dans cette partie des arbres binaires complets dont les noeuds sont étiquetés par des booléens. On rappelle que la profondeur d'un noeud est égale à la longueur du chemin reliant la racine à ce noeud et la hauteur d'un arbre est la plus grande profondeur de ses feuilles (la racine est donc à la profondeur 0). Les noeuds seront numérotés à partir de la racine qui porte le numéro 1 dans l'ordre d'un parcours en largeur (de gauche à droite à chaque profondeur).

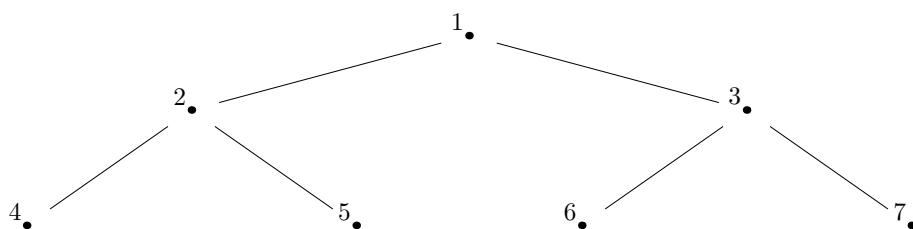


Figure 2 Numérotation des noeuds

Q 20. Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, quel numéro peut avoir un noeud à la profondeur $k \in \llbracket 0, p \rrbracket$? Combien le sous-arbre dont la racine a le numéro i a-t-il de feuilles?

Q 21. Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, pour le noeud numéro i , donner, en les justifiant, les numéros de son fils gauche, de son fils droit et de son père.

Informatiquement, un tel arbre complet de hauteur p sera implémenté par un tableau de booléens de taille 2^{p+1} , la case d'indice $i \in \llbracket 1, 2^{p+1} - 1 \rrbracket$ contenant l'étiquette du noeud numéroté i . La case d'indice 0 n'est pas utilisée. On notera que dans tous les programmes de cette partie, on peut avoir accès à la valeur 2^p via la taille du tableau.

Soit $p \in \mathbb{N}$. On choisit de coder un ensemble $E \subset \llbracket 0, 2^p - 1 \rrbracket$ par un arbre binaire complet de hauteur p selon les règles suivantes :

- chaque feuille numérotée i est étiquetée par `true` si et seulement si $i - 2^p \in E$;
- chaque noeud interne est étiqueté par le « ou logique » des étiquettes de ses deux fils.

```
let ensemble = bool array ;;
```

Par exemple, l'ensemble $\{0, 1, 7\}$ est représenté par l'arbre représenté figure 3.

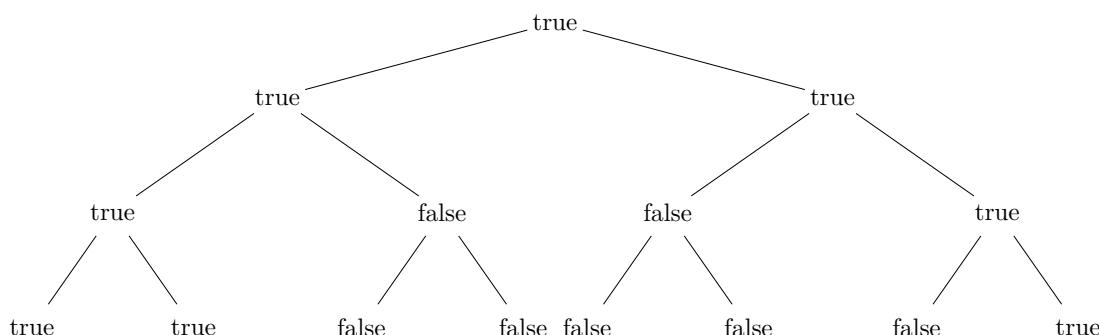


Figure 3 Arbre associé à l'ensemble $\{0, 1, 7\}$ pour $p = 3$

Q 22. Écrire une fonction `appartient` de signature `ensemble -> int -> bool` qui détermine si un entier quelconque appartient ou non à un ensemble donné. Calculer sa complexité.

Q 23. Écrire une fonction `fabrique` de signature `int list -> int -> ensemble` qui prend en arguments une liste d'entiers positifs distincts ℓ et une valeur pour 2^p et renvoie l'arbre associé à l'ensemble E dont les éléments sont ceux de la liste. On supposera que tous les éléments de ℓ appartiennent à $\llbracket 0, 2^p - 1 \rrbracket$. Cette fonction devra s'exécuter en $\mathcal{O}(2^p)$.

Q 24. Écrire une fonction `insere` de signature `ensemble -> int -> unit` qui ajoute un entier k à un ensemble E . On suppose k compatible avec la valeur de p associée à E . Cette fonction devra s'exécuter en $\mathcal{O}(1)$ dans le meilleur cas.

Q 25. Écrire une fonction `supprime` de signature `ensemble -> int -> unit` qui retire un entier k d'un ensemble E . On suppose k compatible avec la valeur de p associée à E . Calculer la complexité de cette fonction dans le pire cas.

Q 26. Écrire une fonction `minlocal` de signature `ensemble -> int -> int` qui cherche l'élément de E minimal parmi ceux codés dans le sous-arbre de racine numérotée i dans l'arbre associé à E . Si un tel élément n'existe pas, cette fonction devra renvoyer -1 . Calculer la complexité de cette fonction en fonction de p et i .

On veut maintenant écrire une fonction qui calcule le successeur d'un entier x dans un ensemble E pour une telle structure. On propose l'algorithme suivant, pour $x \in \llbracket 0, 2^p - 1 \rrbracket$:

- on part de la case numéro i codant l'entier x dans E ;
- tant que i n'est pas le noeud le plus à droite à sa profondeur et que la case $i + 1$ vaut `false`, on remplace i par son père ;
- on renvoie l'élément minimum du sous-arbre de racine $i + 1$ ou -1 si i était totalement à droite.

Q 27. Prouver l'algorithme décrit ci-dessus.

Q 28. Écrire une fonction `successeur` de signature `ensemble -> int -> int` prenant en arguments l'ensemble E et un entier x positif et renvoyant son successeur dans E (-1 si cela n'existe pas).

Q 29. Montrer que si $x \in E$ admet bien un successeur dans E , il existe une constante $K > 0$ indépendante de E et p telle que la complexité de `successeur e x` soit majorée par $K(\log_2(successeur(x) - x) + 2)$. On admet que le même type de justification montre que si x est le maximum de E , la complexité de `successeur e x` est majorée par $K(\log_2(2^p - x) + 2)$.

Q 30. En utilisant la fonction `successeur`, écrire une fonction `cardinal` de signature `ensemble -> int` prenant en argument un ensemble et renvoyant son cardinal.

Q 31. Déterminer la complexité de la fonction `cardinal` en fonction de p et $n = |E|$. On rappelle que la fonction \log_2 est concave.

Q 32. Quels sont les intérêts et inconvénients d'une telle structure ? Dans quels cas peut-elle s'avérer plus intéressante que des structures connues ?

IV Arbres de van Emde Boas

Soit p un entier positif et $N = 2^{2^p}$. On supposera que tous les entiers manipulés restent représentables par la structure d'entier OCaml ordinaire. On considère le type de structure suivant (appelé arbre *veb* par suite) implémentant un ensemble E d'entiers positifs strictement inférieurs à N :

```
type veb = {mutable mini : int; mutable maxi : int; table : veb array};;
```

Les champs mutables d'un arbre *veb* sont modifiables : par exemple, pour un arbre *veb* noté `v`, `v.mini <- 1` change la valeur du champ `v.mini`.

Le codage d'un ensemble $E \subset \llbracket 0, N - 1 \rrbracket$ par un arbre *veb* dit d'ordre $N = 2^{2^p}$ suit les règles suivantes :

- Les champs `mini` et `maxi` représentent toujours la valeur minimale et maximale de E . Ils sont mis arbitrairement à -1 si l'ensemble est vide.
- Si $N = 2$, i.e. si l'arbre doit coder une partie de $\llbracket 0, 1 \rrbracket$, le champ `table` est un tableau vide (i.e. `[[]]`), les champs `mini` et `maxi` suffisant à coder E . **On veillera à ce que les fonctions demandées traitent correctement ce cas particulier.**
- Pour $N > 2$, on note $\widehat{E} = E \setminus \{\min(E)\}$ (où $\min(E)$ désigne le minimum de E). On décompose \widehat{E} en $\sqrt{N} = 2^{2^{p-1}}$ ensembles E_k définis par

$$\forall k \in \llbracket 0, \sqrt{N} - 1 \rrbracket, \quad E_k = \left\{ x - k\sqrt{N} \mid x \in \widehat{E} \cap \llbracket k\sqrt{N}, (k+1)\sqrt{N} - 1 \rrbracket \right\}$$

le champ `table` est alors un tableau de $\sqrt{N} + 1$ arbres *veb* d'ordre \sqrt{N} :

- pour $k \in \llbracket 0, \sqrt{N} - 1 \rrbracket$, l'arbre *veb* stocké dans la case `table.(k)` code l'ensemble E_k ;
- l'arbre *veb* stocké dans la case `table.(\sqrt{N})` code l'ensemble $R = \left\{ k \in \llbracket 0, \sqrt{N} - 1 \rrbracket \mid E_k \neq \emptyset \right\}$.

Par exemple, considérons l'arbre *veb* **ex** codant l'ensemble $E = \{2, 13, 14, 15\}$ avec $p = 2$ i.e. $N = 16$. On a $\widehat{E} = \{13, 14, 15\}$ puisque $\min(E) = 2$ et donc les cinq cases du tableau **ex.table** correspondent respectivement aux ensembles

$$E_0 = \emptyset \quad E_1 = \emptyset \quad E_2 = \emptyset \quad E_3 = \{13 - 3\sqrt{16}, 14 - 3\sqrt{16}, 15 - 3\sqrt{16}\} = \{1, 2, 3\} \quad R = \{3\}$$

On obtient la structure représentée figure 4.

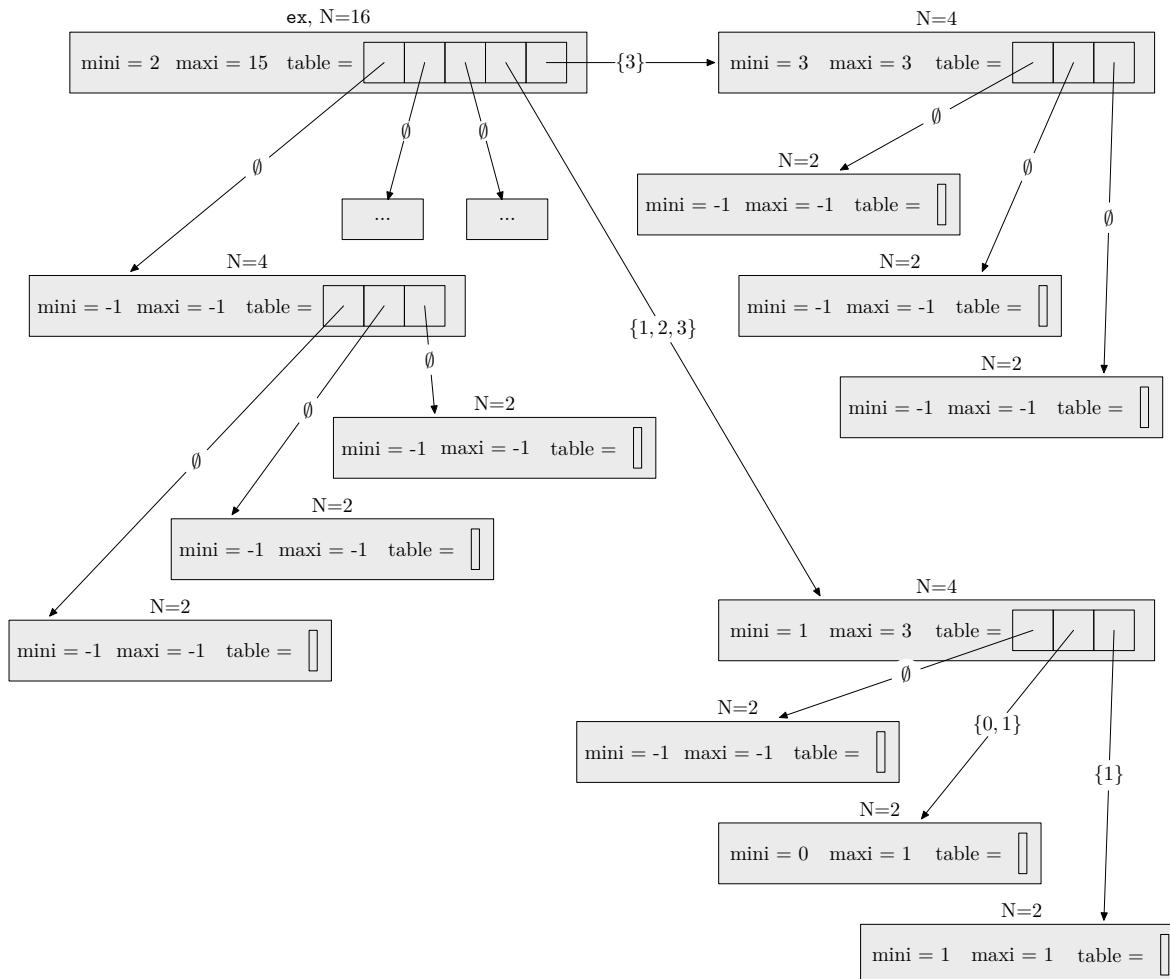


Figure 4 L'arbre *veb* **ex** associé à l'ensemble $\{2, 13, 14, 15\}$ avec $p = 2$

Q 33. On veut coder l'ensemble $\{0, 2, 3, 5, 7, 13, 14\}$ par un arbre *veb* d'ordre $N = 16$, noté **ex_veb**. Indiquer quel ensemble code chaque arbre *veb* stocké dans **ex_veb.table** puis préciser **ex_veb.table**.(3).

Q 34. Écrire une fonction **creer_veb** de signature **int -> veb** prenant en argument un entier p et créant un arbre *veb* d'ordre $N = 2^{2p}$ implémentant l'ensemble vide.

Q 35. Résoudre en fonction de q la récurrence $C(q) = C(\sqrt{q}) + \mathcal{O}(1)$ dans le cas où $q = 2^{2^s}$.

Q 36. Écrire une fonction **appartient_veb** de signature **veb -> int -> bool** qui teste l'appartenance d'un entier x quelconque à un ensemble E codé par un arbre *veb*. Calculer sa complexité dans le pire cas.

Q 37. Écrire une fonction **successeur_veb** de signature **veb -> int -> int** qui calcule le successeur d'un entier x quelconque dans E (-1 s'il n'y en a pas). Calculer sa complexité dans le pire cas.

Q 38. Écrire une fonction **insertion_veb** de signature **veb -> int -> unit** qui insère un entier x dans un arbre *veb*. On suppose que $x \in \llbracket 0, N-1 \rrbracket$. Cette fonction devra avoir une complexité en $\mathcal{O}(\log_2(\log_2(N)))$.

Q 39. Soit $M(N)$ la totalité de l'espace mémoire nécessaire pour stocker un ensemble par un arbre *veb* d'ordre N avec $N = 2^{2p}$. Donner la relation de récurrence vérifiée par $M(N)$ puis déterminer l'ordre de grandeur de $M(N)$.

• • • FIN • • •

SESSION 2023

MPI5IN



ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE MPI

INFORMATIQUE

Durée : 4 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.
- Ne pas utiliser de correcteur.
- Écrire le mot FIN à la fin de votre composition.

Les calculatrices sont interdites.

Le sujet est composé de trois parties, toutes indépendantes.

Partie I - Palindromes

En langue française, "ressasser" est le mot palindrome le plus long, tandis qu'il semble que "saippua-kauppias" soit le plus long mot palindrome au monde, désignant un marchand de savon en Finlande. L'objet de cette partie est de compter le nombre de palindromes présents dans un mot donné.

Soit Σ un alphabet fini contenant au moins deux lettres. On note $u = u_0 \cdots u_{n-1}$ un *mot* sur Σ , composé de n lettres $u_i \in \Sigma, i \in \llbracket 0, n-1 \rrbracket$. La longueur de u est notée $|u|$. Pour $0 \leq i < j \leq n$, on note $u[i, j]$ le mot $u_i \cdots u_{j-1}$. L'ensemble des mots construits sur Σ et contenant le mot vide ϵ est noté Σ^* .

Définition 1 (Miroir)

Soit $u \in \Sigma^*$. Le *miroir* de $u = u_0 \cdots u_{n-1}$, noté \bar{u} , est le mot $\bar{u} = u_{n-1} \cdots u_0$. Par convention $\bar{\epsilon} = \epsilon$.

Définition 2 (Palindrome)

$u \in \Sigma^*$ est un *palindrome* si et seulement si $u = \bar{u}$.

Par convention, le mot vide ϵ n'est pas considéré comme un palindrome.

On dira qu'un palindrome u est *pair* (respectivement *impair*) lorsque sa longueur $|u|$ est paire (resp. impaire).

On recherche donc dans cette partie le nombre de palindromes facteurs d'un mot $u \in \Sigma^*$ (comptés avec les multiplicités éventuelles), soit le cardinal de l'ensemble $\{(i, j), 0 \leq i < j \leq |u|, u[i, j] = \overline{u[i, j]}\}$. Dit autrement, on recherche le nombre de palindromes contenus dans u .

Q1. Si $\Sigma = \{a, b\}$, donner le nombre de palindromes contenus dans le mot $u = babb$.

En parcourant naïvement les lettres d'un mot u donné, on peut proposer un algorithme en pseudo-code permettant de compter le nombre palindromes de u (**algorithme 1**).

Algorithme 1 - Décompte naïf du nombre de palindromes contenus dans un mot donné.

Entrées : Un mot u .

Sorties : Le nombre de palindromes contenus dans u .

début

```

nb = 0
pour i=0 à |u|-1 faire
    pour j=0 à |u|-1 faire
        estPalindrome=True
        pour k=i à j - 1 faire
            si u[i] ≠ u[j - k - 1] alors
                estPalindrome=False
            Sortir du pour k
        si estPalindrome alors
            nb = nb+1
    retourner nb

```

Q2. Évaluer la complexité au pire des cas de l'**algorithme 1** en fonction de $|u|$.

On souhaite bien sûr améliorer cette première idée. Pour ce faire, on utilise tout d'abord le paradigme de la programmation dynamique.

Pour $u \in \Sigma^*$, on définit un tableau de booléens P de taille $(|u| + 1) \times (|u| + 1)$, $P[i][j]$ étant vrai si $u[i, j]$ est un palindrome. On a donc pour tout $i \in \llbracket 0, |u| - 1 \rrbracket$, $P[i][i + 1] = True$.

Q3. Soit $u[i, j]$ un mot. À quelles conditions sur u_i, u_{j-1} et $u[i + 1, j - 1]$ le mot $u[i, j]$ est-il un palindrome ?

Q4. En déduire une relation de récurrence vérifiée par les coefficients de P .

Q5. Écrire un algorithme de programmation dynamique en pseudo-code résolvant le problème.
Évaluer sa complexité.

On insère maintenant entre chaque paire de lettres de u , ainsi qu'au début et à la fin du mot, un symbole spécial noté $\#$. On appelle ce nouveau mot $u^\#$. Ainsi, le mot $u = abba$ se transforme en $u^\# = \#a\#b\#b\#a\#$.

- Q6.** Montrer que les palindromes de $u^\#$, s'ils existent, sont tous impairs.
- Q7.** Soit v un palindrome de longueur $2k + 1$ de u , $k \in \mathbb{N}$. On construit le mot $u^\#$. Donner les deux palindromes correspondants à v dans $u^\#$.
- Q8.** Même question si v est un palindrome de longueur $2k$ de u .
- Q9.** En déduire une stratégie de recherche de tous les palindromes de u .

On voit que les palindromes impairs sont importants. On va donc construire un algorithme de recherche de ce type de palindrome.

Définition 3 (Rayon d'un palindrome)

Soient $u \in \Sigma^*$ et $i \in \llbracket 0, |u| - 1 \rrbracket$. On dit qu'il existe un *palindrome centré en i de rayon $\rho > 0$* si :

- (i). $i - \rho \geq 0$,
- (ii). $i + \rho + 1 \leq |u|$,
- (iii). $u[i - \rho, i + \rho + 1]$ est un palindrome.

Le *rayon maximal* $\hat{\rho}_i$ d'un palindrome centré en i est le plus grand rayon d'un palindrome centré en i .

Par exemple, si $u = abbababab$ et $i = 4$, alors b est un palindrome centré en 4 de rayon 0, aba est un palindrome centré en 4 de rayon 1, $babab$ est un palindrome centré en 4 de rayon 2 et c'est le plus grand, donc $\hat{\rho}_4 = 2$.

- Q10.** En remarquant qu'un palindrome impair est centré sur une lettre (ou un symbole spécial dans le cas de $u^\#$), proposer un algorithme en pseudo-code permettant de compter le nombre de palindromes impairs d'un mot u . Évaluer sa complexité.
- Q11.** Soient $u \in \Sigma^*$, $i \in \llbracket 0, |u| - 1 \rrbracket$ et $0 < j \in \llbracket 1, \hat{\rho}_i \rrbracket$. En exploitant les différentes symétries, montrer qu'il existe un palindrome centré en $i+j$ de rayon $\min(\hat{\rho}_i - j, \hat{\rho}_{i-j})$. En déduire $\hat{\rho}_{i+j} \geq \min(\hat{\rho}_i - j, \hat{\rho}_{i-j})$. Préciser à quelle condition il y a égalité.

En utilisant cette remarque, on développe un algorithme, dit de Manacher, qui construit un tableau d'entiers T permettant de compter le nombre de palindromes d'un mot u . Plus précisément, pour chaque position $i \in \llbracket 1, |u| - 1 \rrbracket$, $T[i]$ indique le rayon maximal $\hat{\rho}_i$, donc tel que la sous-chaîne de $i - \rho_i$ à $i - \rho_i + 1$ est un palindrome. L'algorithme 2 consiste à incrémenter $T[i]$ jusqu'à trouver le plus grand palindrome $u[i - T[i], i + T[i] + 1]$ centré en i .

Algorithme 2 - Algorithme de Manacher

Entrées : Un mot u

Sorties : Un tableau T .

début

```

Initialiser un tableau  $T$  de  $|u|$  cases, initialisées à 0
 $k=0$ 
pour  $i = 1$  à  $|u| - 1$  faire
     $j = i - k$ 
    si  $T[k] \geq j$  alors
         $T[i] = \min(T[k - j], T[k] - j)$ 
    tant que  $(i - (T[i] + 1)) \geq 0$  ET  $(i + T[i] + 1) < |u|$  ET  $(u[i - (T[i] + 1)] = u[i + T[i] + 1])$  faire
         $T[i] = T[i] + 1$ 
         $k = i$ 
    retourner  $T$ 

```

- Q12.** Comment trouver à partir de cet algorithme le nombre de palindromes de u ?

- Q13.** Quelle est la complexité de cet algorithme ? Justifier.

Partie II - Traversée de rivière

Cette partie comporte des questions nécessitant un **code OCaml**.

Dans une vallée des Alpes, un passage à gué fait de cailloux permet de traverser la rivière. Deux groupes de randonneurs arrivent simultanément sur les berges gauche et droite de cette rivière et veulent la traverser. Le chemin étant très étroit, une seule personne peut se trouver sur chaque caillou de ce chemin (**figure 1**). Un randonneur sur la berge de gauche peut avancer d'un caillou (vers la droite sur la **figure 1**) et sauter par dessus le randonneur devant lui (un caillou à droite) si le caillou où il atterrit est libre. De même, chaque randonneur de la berge de droite peut avancer d'un caillou (vers la gauche sur la **figure 1**) et sauter par dessus le randonneur devant lui, dans la mesure où le caillou sur lequel il atterrit est libre. Une fois engagés, les randonneurs ne peuvent pas faire marche arrière. De plus, pour simplifier, on suppose qu'une fois tous les randonneurs sur le chemin, il ne reste qu'un caillou de libre.



Figure 1 - Les randonneurs et le chemin de cailloux

Le chemin de cailloux est défini par un tableau d'entiers :

```
type chemin_caillou = int array
```

Dans ce tableau, un randonneur venant de la berge de gauche est représenté par un 1, un randonneur issu de la berge de droite par un 2 et un caillou libre par un 0.

- Q14.** Écrire une fonction de signature caillou_vide : `chemin_caillou -> int` qui détermine la position du caillou inoccupé.
- Q15.** Écrire une fonction de signature echange : `chemin_caillou -> int -> int -> chemin_caillou` qui permute les valeurs codées sur deux cailloux. Le tableau d'entiers initial représentant le chemin n'est pas modifié. On pourra utiliser ici la fonction `copy` du module `Array`.
- Q16.** Écrire une fonction de signature randonneurG_avance : `chemin_caillou -> bool` qui teste si, parmi les randonneurs venant de la berge de gauche, il en existe un qui puisse avancer (vers la droite).
- Q17.** Écrire une fonction de signature randonneurG_saute : `chemin_caillou -> bool` qui teste si, parmi les randonneurs venant de la berge de gauche, il en existe un qui puisse sauter (vers la droite) au-dessus d'un randonneur.

On supposera dans la suite les fonctions de signature `randonneurD_avance : chemin_caillou -> bool` et `randonneurD_saute : chemin_caillou -> bool` écrites de manière similaire pour les randonneurs venant de la berge de droite.

Q18. Écrire une fonction de signature `mouvement_chemin : chemin_caillou -> chemin_caillou list` qui, en fonction de l'état du chemin, calcule l'état suivant après les opérations suivantes (si elles sont permises) :

- (i). déplacement d'un randonneur venant de la berge de gauche,
- (ii). déplacement d'un randonneur venant de la berge de droite,
- (iii). saut d'un randonneur venant de la berge de gauche,
- (iv). saut d'un randonneur venant de la berge de droite.

On donne la syntaxe OCaml pour créer une liste de N entiers $i : \text{List.init } N (\text{fun } x \rightarrow i)$.

Par exemple, `List.init 5 (fun x->2);;` renvoie `[2; 2; 2; 2; 2]`.

Q19. Écrire une fonction de signature `passage : int -> int`, utilisant la question précédente, telle que l'appel `passage nG nD` résout le problème de passage de nG randonneurs venant de la berge de gauche et nD randonneurs venant de la berge de droite. Par exemple, `passage 3 2` permet de passer de `[1 ; 1 ; 0 ; 2 ; 2]` à `[2 ; 2 ; 0 ; 1 ; 1]`.

Partie III - Calcul d'une coupe minimum d'un graphe

Un agent secret a pour mission de perturber le réseau de communications ennemi en coupant certains fils dans le réseau. Ayant peu de moyens, l'agent a pour consigne de couper le moins de fils possibles pour accomplir cette tâche. Le réseau de communications est modélisé à l'aide d'un *multigraphe* non orienté $G = (S, A)$, où S est l'ensemble des sommets du graphe et A l'ensemble de ses arêtes. Résoudre le problème de l'agent secret, c'est rechercher une *coupé minimum* dans G .

Définition 4 (Multigraphe)

Un multigraphe est un graphe dans lequel des couples de mêmes sommets peuvent être reliés par plus d'une arête.

Dans toute la suite, on considérera les multigraphes sans arêtes du type (s, s) (boucles).

Définition 5 (Coupe)

Une coupe d'un multigraphe non orienté $G = (S, A)$ est une partition non triviale (S_1, S_2) de S , c'est-à-dire telle que $S_1 \cup S_2 = S$, $S_1 \cap S_2 = \emptyset$ et $S_1, S_2 \neq \emptyset$. On dira que les arêtes reliant S_1 à S_2 sont les arêtes de la coupe.

La *taille* d'une coupe (S_1, S_2) est définie par $|(S_1, S_2)| = |\{(s, t) \in A, s \in S_1, t \in S_2\}|$. Autrement dit, c'est le nombre d'arêtes de G qui relient S_1 et S_2 .

Définition 6 (Coupe minimum)

Une coupe minimum est une coupe de taille minimale.

Pour trouver une coupe minimum d'un multigraphe $G = (S, A)$, on pourrait énumérer l'ensemble des coupes possibles (il y en a $2^{|S|} \dots$), ou encore utiliser un algorithme de flot (le plus efficace étant en $\mathcal{O}(|S|^3)$). Nous proposons dans la suite un algorithme probabiliste, l'algorithme de Karger, basé sur la notion de contraction d'arête.

III.1 - Contraction d'arête

Soient $G = (S, A)$ un multigraphe et $a = (s, t) \in A$ une arête de G de sommets s et t . *Contracter* l'arête a consiste à :

- (i). créer un nouveau sommet $u = (st)$ dans S . u est un *supersommet* du nouveau graphe,
- (ii). pour toute arête (r, s) ou (r, t) , $r \in S$, ajouter une arête (r, u) à A . Dans le cas où (r, s) et (r, t) existent, deux arêtes reliant r à u sont créées.
- (iii). Supprimer de A toutes les arêtes ayant s ou t comme extrémité.
- (iv). Supprimer s et t de S .

Un supersommet peut donc être vu comme un ensemble de deux sommets du graphe initial, obtenu après une contraction d'arête.

Le multigraphe obtenu est appelé *graphe contracté* et est noté G/st . Si plusieurs arêtes relient s à t dans G , contracter l'une d'entre elles supprime toutes les autres du graphe G/st . Dans toute la suite, on notera G_0 le graphe initial, avant transformations par contractions.

On considère le graphe G_0 de la **figure 2**.

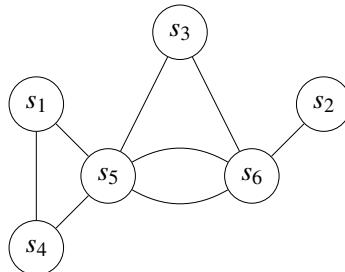


Figure 2 - Graphe G_0 exemple

Q20. Donner le graphe obtenu par contraction d'une arête (s_5, s_6) .

Après plusieurs contractions, un supersommet u du graphe résultant contient un sous-ensemble $S_u \subseteq S$ de sommets de G .

- Q21.** Soit u un supersommet et $s, t \in S_u$. Montrer qu'il existe un chemin Γ entre s et t dans G_0 tel que chaque arête de Γ a été contractée.
- Q22.** Montrer qu'en contractant une arête s, t dans un multigraphe G , la taille d'une coupe minimum du graphe contracté G/st est au moins égale à la taille d'une coupe minimum de G .
- Q23.** Montrer que la taille d'une coupe minimum de G/st est strictement plus grande que celle d'une coupe minimum de G si et seulement si l'arête (s, t) est une arête de toutes les coupes minimum de G .

III.2 - Premier algorithme

On construit un premier algorithme qui essaye de trouver une coupe minimum en contractant aléatoirement une arête de G , jusqu'à ce qu'il ne reste que deux sommets (**algorithme 3**).

Algorithme 3 - Algorithme de Karger.

Karger(G, n)

Entrées : $G = (S, A)$ multigraphe non orienté, $n \in \mathbb{N}$

Sorties : Une coupe minimum de G .

début

pour $i = |S|$ à n en décrémentant de 1 faire

Tirer aléatoirement (loi uniforme) une arête $a = (s, t) \in A$

$G = G/st$

// Appel

Karger($G, 2$)

- Q24.** Appliquer l'**algorithme 3** au graphe de la **figure 2**, en contractant successivement (s_5, s_6) , $((s_5s_6), s_2)$, (s_1, s_4) et $((s_5s_6s_2), s_3)$. Chaque graphe contracté sera représenté. La coupe calculée est-elle une coupe minimum ?

D'après la **Q23**, tant que l'on ne contracte pas une arête faisant partie de toutes les coupes minimum de G , alors l'**algorithme 3** trouvera une coupe minimum. On cherche alors la probabilité de ne jamais contracter une telle arête.

Q25. Soit $d(s)$ le degré d'un sommet $s \in S$. Montrer que $\sum_{s \in S} d(s) = 2|A|$ (lemme des poignées de main).

Q26. En déduire qu'une coupe minimum a une taille d'au plus $2|A|/|S|$.

Soit $C = (S_1, S_2)$ une coupe minimum de G . L'objet des questions **Q27** et **Q28** est de minorer la probabilité que l'**algorithme 3** renvoie C . Pour cela, on remarque que cet algorithme ne renvoie pas C si et seulement si lors d'une itération on choisit une arête qui traverse C , c'est-à-dire telle qu'un sommet est dans S_1 et l'autre dans S_2 .

Q27. Quelle est la probabilité maximum de choisir une arête qui traverse C ?

Q28. En déduire que la probabilité $P_{|S|}$ que l'**algorithme 3** renvoie C est supérieure ou égale à $\frac{2}{|S|(|S|-1)}$.

Q29. Déduire de la question précédente qu'un multigraphe non orienté $G = (S, A)$ possède au plus $\frac{|S|(|S|-1)}{2}$ coupes minimum.

III.3 - Deuxième algorithme

Le résultat précédent n'est pas satisfaisant d'un point de vue complexité. On peut cependant l'améliorer facilement en utilisant une technique d'*amplification* : on itère l'**algorithme 3** plusieurs fois et on retourne la valeur minimale obtenue (**algorithme 4**).

Algorithme 4 - Algorithme de Karger amplifié.

KargerAmplifie(G, N)

Entrées : $G = (S, A)$ multigraphe non orienté, $N \in \mathbb{N}$.

Sorties : Une coupe minimum de G .

début

$t = \infty$

pour $i = 1$ à N **faire**

$X = \text{Karger}(G, 2)$

si $|X| < t$ **alors**

$t = |X|$

$C = X$

retourner C

Q30. Montrer que la probabilité maximale que l'**algorithme 4** ne trouve pas une coupe minimum est égale à $\left(1 - \frac{2}{|S|(|S|-1)}\right)^N$.

Q31. On rappelle que pour tout $x \in \mathbb{R}$, $1 + x \leq e^x$. Montrer que la probabilité que l'**algorithme 4** renvoie une coupe minimum est supérieure à $1 - e^{2N/|S|(|S|-1)}$. En déduire, pour $c > 0$ fixé, la plus petite valeur de N telle que cette probabilité soit supérieure à $1 - \frac{1}{N^c}$.

Q32. Les algorithmes proposés dans cette partie sont des algorithmes probabilistes. Sont-ils de type Monte Carlo ou Las Vegas ? Justifier la réponse.

III.4 - Implémentation en langage C

Cette partie comporte des questions de programmation qui seront abordées en utilisant **exclusivement le langage C**. Les codes seront commentés de manière pertinente.

On suppose qu'un graphe est codé à l'aide de l'ensemble de ses arêtes, chaque arête étant définie par ses deux sommets, représentés par des entiers.

- Q33.** Définir en C des types structurés Arete et Graphe. Pour ce dernier, on s'attachera à avoir un accès direct au nombre de sommets et au nombre d'arêtes. On rappelle la définition d'un type structuré par `struct nom_s {type1 champ1; ...; typen champn;}` et ensuite `typedef struct nom_s nom;`.

Pour tout u supersommet, les S_u forment une partition de S . Pour coder ces sous-ensembles, on a donc naturellement recours à la structure de données Unir & Trouver (Union-Find), qui va permettre de trouver rapidement à quel sous-ensemble S_u un sommet $s \in S$ appartient (Trouver) et de fusionner deux supersommets S_u et S_v déjà construits (Unir).

On suppose donc disposer :

- (i). d'un type `subset`, décrivant un supersommet d'un graphe contracté.

```
struct subset
{
    int parent; // recherche par compression de chemin pour Trouver
    int rang;   // Union par rang.
};
typedef struct subset subset;
```

- (ii). d'une fonction de prototype `int Trouver(subset subsets[], int s)`, qui retourne le numéro du supersommet dans lequel le sommet s se trouve (par compression de chemin),
- (iii). d'une fonction de prototype `void Unir(subset subsets[], int Su, int Sv)` qui fusionne les deux supersommets S_u et S_v (union par rang) et maintient à jour la liste des supersommets.

Il n'est pas demandé d'écrire ces deux dernières fonctions.

- Q34.** Écrire une fonction de prototype `int contracteArete(Graphe G, subset subsets[], Arete a)` qui contracte l'arête a . On veillera à ne pas contracter l'arête si ses deux extrémités sont dans le même supersommet. La fonction renvoie 0 si aucune arête est contractée et -1 sinon.
- Q35.** Écrire une fonction de prototype `int compteAretesCoupe(Graphe G, subset subsets[])` qui compte le nombre d'arêtes du graphe contracté final G résultant de l'**algorithme 3**. `subsets` est la partition des sommets du graphe initial S calculée par l'**algorithme 3**.
- Q36.** En déduire une fonction de prototype `int kargerMinCut(Graphe G0)` qui renvoie la taille de la coupe calculée par l'**algorithme 3** sur le graphe G_0 . Les supersommets initiaux sont les $|S|$ sommets de G_0 , chacun ayant un rang nul et un numéro parent égal au numéro du sommet. Pour effectuer un tirage aléatoire uniforme, on utilisera la fonction `int rand()` de la bibliothèque `stdlib.h` qui renvoie un nombre aléatoire entre 0 et `RAND_MAX` ≥ 32767. `RAND_MAX` est une constante définie dans `stdlib.h`.
- Q37.** Donner la complexité au pire des cas de cet algorithme.

FIN

SESSION 2023



MP7IN

ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE MP

INFORMATIQUE

Durée : 4 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.
- Ne pas utiliser de correcteur.
- Écrire le mot FIN à la fin de votre composition.

Les calculatrices sont interdites.

Le sujet est composé de trois parties indépendantes.

Partie I - Programmation en OCaml : sélection du $(k + 1)^{\text{e}}$ plus petit élément

La sélection du $(k + 1)^{\text{e}}$ plus petit élément d'une liste d'entiers L , non nécessairement triée, consiste à trouver le $(k + 1)^{\text{e}}$ élément de la liste obtenue en triant L dans l'ordre croissant.

Par exemple, si $L = [9; 1; 2; 4; 7; 8]$ le 3^{e} plus petit élément de L est 4. On pourra remarquer que si la liste L est triée dans l'ordre croissant, le $(k + 1)^{\text{e}}$ plus petit élément est l'élément de rang k dans L .

On présente un algorithme permettant de résoudre ce problème de sélection avec une complexité temporelle linéaire dans le pire cas. Celui-ci est basé sur le principe de "diviser pour régner" et sur le choix d'un bon pivot pour partager la liste en deux sous-listes.

Dans cette partie, les fonctions demandées sont à écrire en OCaml et ne doivent faire intervenir aucun trait impératif du langage (références, tableaux ou autres champs mutables ou exception par exemple).

Étant donné un réel a , on note $\lfloor a \rfloor$ le plus grand entier inférieur ou égal à a .

I.1 - Fonctions utiles

Dans cette section, on écrit des fonctions auxiliaires qui sont utiles pour la fonction principale.

Q1. Écrire une fonction récursive de signature :

```
longueur : 'a list -> int
```

et telle que `longueur l` est la longueur de la liste `l`.

Q2. Écrire une fonction récursive de signature :

```
insertion : 'a list -> 'a -> 'a list
```

et telle que `insertion l a` est la liste triée dans l'ordre croissant obtenue en ajoutant l'élément `a` dans la liste croissante `l`.

Q3. En déduire une fonction récursive de signature :

```
tri_insertion : 'a list -> 'a list
```

et telle que `tri_insertion l` est la liste obtenue en triant `l` dans l'ordre croissant.

Q4. Écrire une fonction récursive de signature :

```
selection_n : 'a list -> int -> 'a
```

et telle que `selection_n l n` est l'élément de rang `n` de la liste `l`.

Par exemple, `selection_n [4;2;6;4;1;15] 3` est égal à 4.

Q5. Écrire une fonction récursive de signature :

```
paquets_de_cinq : 'a list -> 'a list list
```

et telle que `paquets_de_cinq l` est une liste de listes obtenue en regroupant les éléments de la liste `l` par paquets de cinq sauf éventuellement le dernier paquet qui est non vide et qui contient au plus cinq éléments. Par exemple :

- `paquets_de_cinq []` est égal à `[]`,
- `paquets_de_cinq [2;1;2;1;3]` est égal à `[[2;1;2;1;3]]`,
- `paquets_de_cinq [3;4;2;1;5;6;3]` est égal à `[[3;4;2;1;5];[6;3]]`.

Q6. Écrire une fonction récursive de signature :

```
medians : 'a list list -> 'a list
```

et telle que `medians l` est la liste `m` obtenue en prenant dans chaque liste `lk` apparaissant dans la liste de listes `l` l'élément médian de `lk`. On convient que pour une liste `A` dont les éléments sont exactement $a_0 \leq a_1 \leq \dots \leq a_{n-1}$, l'élément médian désigne $a_{\lfloor \frac{n}{2} \rfloor}$.

Dans le cas où la liste `L` n'est pas triée, l'élément médian désigne l'élément médian de la liste obtenue en triant `L` par ordre croissant. Par exemple :

```
medians [[3;1;5;3;2];[4;3;1];[1;3];[5;1;2;4]] est égal à [3;3;1;2].
```

Q7. Écrire une fonction de signature :

```
partage : 'a -> 'a list -> 'a list * 'a list * int * int
```

telle que `partage p l` est un quadruplet `11,12,n1,n2` où `11` est la liste des éléments de `l` plus petit que `p`, `12` est la liste des éléments de `l` strictement plus grand que `p`, `n1` et `n2` sont respectivement les longueurs de `11` et `12`.

I.2 - La fonction de sélection et sa complexité

On détaillera la fonction de sélection :

Q8. Écrire une fonction récursive de signature :

```
selection : 'a list -> int -> 'a
```

telle que `selection l k` est le $(k + 1)^{\text{e}}$ plus petit élément de la liste `l`. L'écriture de la fonction sera une traduction en OCaml de l'[Algorithme 1](#) présenté en page 4.

On cherche à déterminer la complexité en nombre de comparaisons de la fonction `selection`. Pour tout $n \in \mathbb{N}$, on note $T(n)$ le nombre maximum de comparaisons entre éléments lors d'une sélection d'un élément quelconque dans des listes `L` **sans répétition** de taille n .

En analysant l'[Algorithme 1](#), il est possible de démontrer que :

$$\forall n \geq 55, T(n) \leq T\left(\left\lfloor \frac{n+4}{5} \right\rfloor\right) + T\left(\left\lfloor \frac{8n}{11} \right\rfloor\right) + 4n. \quad (\text{I})$$

Q9. En admettant la proposition (I), montrer que pour tout entier n supérieur à 1, on a :

$$T(n) \leq (200 + T(55))n.$$

Pour l'initialisation, on pourra remarquer que T est une fonction croissante.

Algorithme 1 - Sélection du $(k + 1)^{\text{e}}$ plus petit élément

```

1 SELECTION L k :
  /* L est une liste, k est un entier positif */ 
2 début
3   n ← LONGUEUR L
4   si n ≤ 5 alors
5     M ← (TRI_INSERTION L)
6     retourner l'élément de rang k de M
7   fin
8   sinon
9     L_Cinq ← PAQUETS_DE_CINQ L
10    M ← MEDIANES L_Cinq
11    pivot ← SELECTION M ((n + 4) // 5) // 2
        /* L'opérateur // désigne le quotient d'entiers. Le rang ((n + 4) // 5) // 2
           correspond au rang du médian de la liste M */
12    L1, L2, n1, n2 ← PARTAGE pivot L
13    si k < n1 alors
14      retourner SELECTION L1 k
15    fin
16    sinon
17      retourner SELECTION L2 (k - n1)
18    fin
19  fin
20 fin

```

Partie II - Recherche d'une clique de célébrités

II.1 - Définitions et propriétés

Définition 1 (Graphe). On appelle **graphe** un couple $G = (S, A)$ où S est un ensemble fini appelé ensemble des sommets et A est une partie de $S \times S$, appelée ensemble des arêtes.

On pourra remarquer que, dans cette définition de graphe, les éléments de la forme (s, s) où $s \in S$ sont des arêtes possibles.

Définition 2 (Clique). Soit $G = (S, A)$ un graphe. Soit S' une partie de S . On dit que S' est une **clique** si :

$$\forall (s_1, s_2) \in S' \times S', (s_1, s_2) \in A.$$

Définition 3 (Clique de célébrités, célébrité). Soient $G = (S, A)$ un graphe et C une partie de S . On dit que C est une **clique de célébrités** si C est une clique et :

$$\forall (c, s) \in C \times S, ((s, c) \in A) \wedge ((c, s) \in A \implies s \in C).$$

Un élément de l'ensemble C est alors appelé **célébrité**.

Le terme "célébrité" provient de l'interprétation suivante : l'ensemble des sommets correspond à un ensemble de personnes et une arête (s, c) représente le fait que s connaît c . Ainsi, une célébrité est connue de tous et elle connaît uniquement les autres célébrités.

Q10. Dans cette question, on pose $S = \{0, 1, 2, \dots, 6\}$. Pour chacun des graphes suivants, préciser s'ils contiennent une clique de célébrités non vide. Dans le cas où il y en a une, l'expliciter.

1. $G_1 = (S, A_1)$ avec $A_1 = \{(1, 2), (1, 3), (1, 5), (2, 6)\}$.
2. $G_2 = (S, A_2)$ avec

$$A_2 = \left\{ \begin{array}{l} (0, 3), (0, 5), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3), \\ (4, 1), (4, 3), (4, 5), (5, 1), (5, 3), (6, 1), (6, 3) \end{array} \right\}.$$

Q11. Soit $G = (S, A)$ un graphe quelconque. Montrer que s'il existe une clique de célébrités non vide C dans G , alors celle-ci est unique.

Dans la suite, on note C_G l'unique clique de célébrités non vide du graphe G . Dans le cas où celle-ci n'existe pas, C_G désigne alors l'ensemble vide qui est noté \emptyset .

Q12. Soient $G = (S, A)$ un graphe et p un sommet de G . On note $G' = (S \setminus \{p\}, A \cap (S \setminus \{p\}) \times S \setminus \{p\}))$. Montrer les propositions suivantes :

- a) Montrer que si $C_{G'}$ est égal à l'ensemble vide, alors $C_G \in \{\emptyset, \{p\}\}$.
- b) Montrer que si $C_G \setminus \{p\} \neq \emptyset$, alors $C_{G'} = C_G \setminus \{p\}$.
- c) On suppose que $C_{G'}$ n'est pas l'ensemble vide et on fixe c' un élément de $C_{G'}$.
 - i) Montrer que si (p, c') n'est pas un élément de A , alors $C_G \in \{\emptyset, \{p\}\}$.
 - ii) Montrer que si (c', p) n'est pas un élément de A , alors $C_G \in \{\emptyset, C_{G'}\}$.
 - iii) Montrer que si (p, c') et (c', p) sont des éléments de A , alors $C_G \in \{\emptyset, \{p\} \cup C_{G'}\}$.

II.2 - Algorithmique et programmation en Python (Informatique Commune)

Dans la suite, l'ensemble des sommets est de la forme $\{0, 1, \dots, n-1\}$ où n est un entier supérieur à 1 et un graphe $G = (S, A)$ est représenté en Python par sa liste d'adjacence que l'on note L_G et qui est définie par :

$$[[j \mid j \in S \text{ et } (i, j) \in A] \mid i \in S].$$

Par exemple, si $G = (\{0, 1, 2, 3\}, \{(0, 1), (3, 2), (3, 1), (1, 2)\})$, alors $L_G = [[1], [2], [], [1, 2]]$.

On pourra remarquer que si l'ensemble des sommets d'un graphe G est égal à $\{0, 1, \dots, n-1\}$, alors la longueur de la liste L_G est égale à n .

Q13. Écrire une fonction Python `est_clique(L, R)` prenant en argument une liste L qui est une liste d'adjacence d'un graphe $G = (S, A)$ et une liste R sans répétition d'éléments de S et qui renvoie `True` si l'ensemble des éléments de R constitue une clique de G et `False` sinon.

Q14. On considère le graphe G ayant comme liste d'adjacence :

$$L_G = [[1, 3, 5], [0, 2], [4, 6], [2, 4, 5, 6], [2], [2, 3, 4], [2, 4, 6]].$$

Décrire l'évolution de la variable C à chaque étape de l'`Algorithm 2` décrit en page 6.

Q15. Écrire une fonction Python `Clique_possible_C(G)` prenant en argument une liste G représentant un graphe et qui renvoie la liste C construite à l'aide de l'`Algorithm 2`.

Q16. Montrer par récurrence sur le nombre de sommets que si G est un graphe où C_G est non vide, alors `Clique_possible_C(G)` est égale à C_G .

Algorithme 2 - Construction d'une clique de célébrités possibles

```

1 CLIQUE_POSSIBLE_C G :
2 début
3   |   C ← []
4   |   S ← [0, 1, ..., n − 1]
5   |   /* n est le nombre de sommet de G */
6   |   pour chaque s élément de S faire
7       |   |   si C est vide alors
8           |   |   |   Ajouter s dans C
9       |   |   fin
10      |   |   sinon
11          |   |   |   c ← premier élément de C
12          |   |   |   t ← FAUX
13          |   |   |   /* t permet de vérifier si on a effectué certaines instructions */
14          |   |   |   si (s,c) n'est pas une arête de G alors
15              |   |   |   |   C ← [s]
16              |   |   |   |   t ← VRAI
17              |   |   |   fin
18              |   |   |   si (c,s) n'est pas une arête de G alors
19                  |   |   |   |   C ← C
20                  |   |   |   |   t ← VRAI
21                  |   |   |   fin
22                  |   |   |   si t = FAUX alors
23                      |   |   |   |   Ajouter s à la fin de liste C
24                      |   |   |   fin
25      |   |   fin
26  fin
27  retourner C

```

Partie III - Étude d'une famille d'automates

Dans cette partie, l'alphabet Σ désigne l'ensemble $\{0, 1\}$, le symbole ε désigne le mot vide et on rappelle que Σ^* désigne l'ensemble des mots sur l'alphabet Σ .

Étant donné un mot w , on rappelle que $|w|$ désigne la longueur du mot w et l'indexation des lettres de w commence par 0. La première lettre de w est donc w_0 .

La notation $\text{Card}(E)$ désigne le cardinal d'un ensemble E .

Étant donnés un entier n et un entier non nul m , la notation $n \bmod m$ désigne le reste de la division euclidienne de n par m .

III.1 - Définitions

Définition 4 (Automate déterministe). Un **Automate déterministe** est un quintuplet $A = (Q, \Sigma, \delta, q_0, F)$ avec :

- Q un ensemble fini non vide appelé ensemble des états,
- Σ est un ensemble fini appelé alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ une application appelée application de transition,
- q_0 un élément de Q appelé état initial,
- F une partie de Q appelée ensemble des états finaux.

Définition 5 (Application de transition étendue aux mots). Soit $A = (Q, \Sigma, \delta, q_0, F)$ un automate déterministe.

On définit de manière récursive $\delta^* : Q \times \Sigma^* \rightarrow Q$ par :

$$\begin{aligned} \forall q \in Q, \quad \delta^*(q, \varepsilon) &= q, \\ \forall q \in Q, \forall a \in \Sigma, \forall w \in \Sigma^*, \quad \delta^*(q, aw) &= \delta^*(\delta(q, a), w). \end{aligned}$$

Définition 6 (Reconnaissance d'un mot par un automate). Soient $w = w_0 w_1 \dots w_n$ un mot sur un alphabet Σ et $A = (Q, \Sigma, \delta, q_0, F)$. On dit que w est **reconnu** par l'automate A si $\delta^*(q_0, w) \in F$.

Définition 7 (Automate $A_{k,p}$, fonction indicatrice $L_{k,p}$). Soient p et k deux entiers vérifiant $0 \leq k \leq p-1$. L'automate $A_{k,p}$ est défini par :

- $Q = \{0, 1, \dots, p-1\} \times \{0, 1\}$,
- $\Sigma = \{0, 1\}$,
- $\forall (c, e) \in Q, \delta((c, e), 0) = ((c+1) \bmod p, e)$,
- $\forall (c, e) \in Q, \delta((c, e), 1) = \begin{cases} ((c+1) \bmod p, 1-e) & \text{si } c = k \bmod p, \\ ((c+1) \bmod p, e) & \text{sinon.} \end{cases}$
- $q_0 = (0, 0)$,
- $F = \{0, 1, \dots, p-1\} \times \{1\}$.

On note $L_{k,p}$ la fonction indicatrice de l'ensemble des mots reconnus par $A_{k,p}$. Soit autrement :

$$\forall u \in \Sigma^*, L_{k,p}(u) = \begin{cases} 1 & \text{si } A_{k,p} \text{ reconnaît } u \\ 0 & \text{sinon.} \end{cases}$$

III.2 - Exemples et propriétés élémentaires des $A_{k,p}$

Q17. Soit $w \in \Sigma^*$. Expliciter sans démonstration l'état $\delta^*(q_0, w)$, la lecture du mot étant effectuée dans l'automate $A_{1,3}$. On pourra s'aider d'une représentation graphique de l'automate.

Dans les questions **Q18 à Q22**, p et k désignent des entiers tels que $p > 2$ et $k \in \{0, 1, \dots, p - 1\}$.

Q18. Soit $w \in \Sigma^*$. Expliciter l'état $\delta^*(q_0, w)$, la lecture du mot étant effectuée dans l'automate $A_{k,p}$.
On ne demande pas de démonstration.

Un corollaire direct du résultat de la question **Q18** est que l'ensemble des mots reconnus par l'automate $A_{k,p}$ est égal à :

$$\{w \in \Sigma^* \mid \text{Card}(\{m \in \mathbb{N} \mid pm + k \leq |w| - 1 \text{ et } w_{pm+k} = 1\}) \text{ est impair}\}.$$

Dans la suite du problème, on admet ce résultat.

Q19. Soit w un mot reconnu par un automate $A_{k,p}$. Montrer que w est reconnu par au moins un autre automate parmi $A_{0,2}, A_{1,2}, A_{l,p}$ avec $l \neq k$.

Définition 8 (Ou exclusif étendu aux mots binaires). On rappelle que le **Ou exclusif** qu'on note \oplus est une opération définie sur $\{0, 1\}$ par :

$$0 \oplus 0 = 1 \oplus 1 = 0 \text{ et } 0 \oplus 1 = 1 \oplus 0 = 1.$$

Soient $n \in \mathbb{N}$, u et v deux éléments de $\{0, 1\}^n$. On définit le **Ou exclusif** de u et v , noté $u \oplus v$, le mot de longueur n défini par :

$$\forall i \in \{0, 1, \dots, n - 1\}, (u \oplus v)_i = u_i \oplus v_i.$$

Q20. Soient u et v deux mots de Σ^* de même longueur. Montrer que :

$$L_{k,p}(u \oplus v) = L_{k,p}(u) \oplus L_{k,p}(v).$$

Q21. Soit w un mot binaire vérifiant :

$$L_{0,2}(w) = L_{1,2}(w) = 0 \text{ et } \forall k \in \{1, 2, \dots, p - 1\}, L_{k,p}(w) = 0.$$

- a) Montrer que $L_{0,p}(w) = 0$.
- b) En déduire que pour tout mot $w' \in 0^* \cdot w$, on a :

$$L_{0,2}(w') = L_{1,2}(w') = 0 \text{ et } \forall k \in \{1, 2, \dots, p - 1\}, L_{k,p}(w') = 0.$$

Q22. Montrer que pour tout $w \in \Sigma^*$ et $w' \in w \cdot 0^*$, on a $L_{k,p}(w) = L_{k,p}(w')$.

Remarque. Ces égalités permettent la construction d'une relation d'équivalence sur les mots qui est utilisée pour montrer que deux mots de longueur N peuvent être séparés par un automate de la forme $A_{k,p}$ ayant $O(\sqrt{N} \ln(N))$ états.

FIN

Épreuve disciplinaire

Le sujet est constitué de 2 problèmes qui peuvent être traités de manière indépendante.

Notes de programmation : Vous disposez, pour répondre aux questions de ce sujet, des fonctions Python de manipulation de listes ou de matrices suivantes :

- On peut créer une liste de taille n remplie avec la valeur x avec `li = [x] * n`.
- On peut obtenir la taille d'une liste `li` avec `len(li)`.
- Si `li` est une liste de n éléments, on peut accéder au k -ème élément (pour $0 \leq k < \text{len}(li)$) avec `li[k]`. On peut définir sa valeur avec `li[k] = x`.
- On peut ajouter un élément x dans une liste `li` à l'aide de `li.append(x)`, et on considérera qu'il s'agit d'une opération élémentaire.
- Les matrices sont des listes de listes, chaque sous-liste étant considérée comme une ligne de la matrice. Si `mat` est une matrice, elle possède `len(mat)` lignes et `len(mat[0])` colonnes.
- On peut créer une matrice de n lignes et p colonnes, dont toutes les cases contiennent x avec `mat = [[x for j in range(p)] for i in range(n)]`.
- On accède (resp. modifie) l'élément de `mat` dans la i -ème ligne et j -ème colonne avec `mat[i][j]` (resp. `mat[i][j] = x`).
- On peut concaténer deux listes en utilisant l'opération `li1 + li2`. On utilisera aussi cette opération dans des expressions mathématiques.
- `li[a:b]` désigne la liste des éléments d'indice compris entre a et $b - 1$ dans `li`. On utilisera aussi cette opération dans des expressions mathématiques.

Les autres fonctions sur les listes (`sort`, `index`, `max`, etc.) sont interdites à moins de les réécrire explicitement. L'opérateur `in` d'appartenance à une liste est interdit, mais on peut utiliser ce mot-clé dans les autres contextes (par exemple dans une boucle `for`).

Complexité : Par *complexité* d'un algorithme, on entend le nombre d'opérations élémentaires nécessaires à l'exécution de cet algorithme dans le pire cas. Lorsque cette complexité dépend d'un ou plusieurs paramètres k_0, \dots, k_{r-1} , on dit que la complexité est $O(f(k_0, \dots, k_{r-1}))$ s'il existe une constante $C > 0$ telle que, pour toutes les valeurs k_0, \dots, k_{r-1} suffisamment grandes, ce nombre d'opérations élémentaires est majoré par $C \times f(k_0, \dots, k_{r-1})$.

Problème 1 Programmation en Python

Ce problème porte sur l'écriture de programmes à l'aide du langage Python.

Partie I Programmes divers

► **Question 1** Écrire une fonction `fibonacci(n)` qui prend en argument un entier n supérieur ou égal à 2 et renvoie la liste des n premiers termes de la suite de Fibonacci $(F_n)_{n \in \mathbb{N}}$ définie par $F_0 = 0$ et $F_1 = 1$ et $\forall n \geq 2, F_n = F_{n-1} + F_{n-2}$ (chaque terme est la somme des deux précédents).

► **Question 2** Écrire une fonction `indice_min(li)` qui prend en argument une liste d'entiers `li` et renvoie l'indice d'un de ses minimums.

► **Question 3** Que renverra `indice_min([1, 0, 2, 0])` avec votre programme ?

► **Question 4** Écrire une fonction `zero_sur_lignes(mat)` qui prend en argument une matrice (c'est-à-dire une liste de listes) contenant des 0 et des 1, renvoie `True` lorsque chaque ligne contient au moins un zéro, et `False` sinon.

Note : un programme efficace sera valorisé.

► **Question 5** Écrire une fonction `lettre_majoritaire(ch)` qui prend en argument une chaîne de caractères non vide et renvoie le caractère qui apparaît le plus fréquemment. Ainsi, `lettre_majoritaire('abcdedde')` devrait renvoyer '`d`'.

Note : l'utilisation efficace d'un dictionnaire sera valorisée. On pourra alors utiliser l'opérateur `in`.

Partie II Saut de valeur maximale

A. Introduction

Dans une liste de flottants `li`, on appelle *saut* un couple (i, j) avec $0 \leq i \leq j < \text{len}(li)$, et la *valeur* d'un saut est la valeur $li[j] - li[i]$. On va ici programmer plusieurs manières de trouver un saut de valeur maximale dans une liste. Par exemple, dans la liste `[2.0, 0.2, 3.0, 5.3, 2.0]`, un tel saut est $(1, 3)$ (car 0.2 et 5.3 sont aux indices 1 et 3 respectivement).

► **Question 6** Écrire une fonction `valeur(li, saut)` qui prend en argument une liste et un saut et renvoie la valeur du saut.

► **Question 7** Donner un exemple de liste avec exactement deux sauts de valeur maximale et préciser ces sauts.

► **Question 8** À l'aide d'un contre-exemple, montrer qu'on ne peut pas se contenter de chercher le minimum et le maximum d'une liste pour trouver un saut de valeur maximale.

► **Question 9** Écrire une fonction `saut_max_naif(li)` qui renvoie un saut de valeur maximale en testant tous les couples (i, j) tels que $0 \leq i \leq j < \text{len}(li)$.

B. Programmation dynamique

On décrit ici un algorithme utilisant le paradigme de la programmation dynamique pour résoudre ce problème : pour chaque k entre 1 et `len(li)`, on va calculer m_k l'indice du minimum de `li[0:k]`, et le couple (i_k, j_k) un saut de valeur maximale dans `li[0:k]`. Ainsi, on aura $m_1 = i_1 = j_1 = 0$ car `li[0:1]` ne comporte qu'un seul élément.

► **Question 10** Pour $k < \text{len}(\text{li})$, expliquer comment on peut calculer efficacement m_{k+1} à partir de m_k et des valeurs dans `li`.

► **Question 11** Justifier que la relation suivante est correcte.

$$(i_{k+1}, j_{k+1}) = \begin{cases} (i_k, j_k) \text{ si } \text{li}[k] - \text{li}[m_k] < \text{li}[j_k] - \text{li}[i_k] \\ (m_k, k) \text{ sinon} \end{cases}$$

► **Question 12** Écrire une fonction `saut_max_dynamique(li)` qui renvoie un saut de valeur maximale en utilisant la relation de la question 11.

► **Question 13** Déterminer la complexité de votre programme dans le pire cas, puis comparer cette complexité avec celle du programme donné en question 9.

C. Méthode diviser pour régner

On propose une dernière méthode pour le calcul d'un saut de valeur maximale utilisant le paradigme diviser pour régner. On notera $\lfloor x \rfloor$ la partie entière d'un nombre x quelconque.

Si une liste `li` comporte $n \geq 2$ éléments, on souhaite calculer :

- (i_g, j_g) un saut de valeur maximale lorsque $j_g < \lfloor n/2 \rfloor$,
- (i_d, j_d) un saut de valeur maximale lorsque $i_d \geq \lfloor n/2 \rfloor$,
- et (i_m, j_m) un saut de valeur maximale lorsque $i_m < \lfloor n/2 \rfloor \leq j_m$.

► **Question 14** Justifier qu'un saut de valeur maximale de `li` est nécessairement un des trois ci-dessus.

► **Question 15** Avec les notations précédentes, justifier que i_m est nécessairement l'indice d'une valeur minimale dans la moitié gauche de `li` (on admettra que, de même, j_m est nécessairement l'indice d'une valeur maximale dans la moitié droite de `li`).

► **Question 16** Écrire une fonction récursive `saut_max_aux(li, a, b)` qui prend en argument une liste `li` et deux indices $a < b$ de cette liste et renvoie le quadruplet constitué d'un saut de valeur maximale dans `li` entre les indices a et $b - 1$ (deux valeurs), des indices d'un minimum et d'un maximum de `li` entre les indices a et $b - 1$ (deux valeurs).

Ainsi, `saut_max_aux([2.0, 5.0, 3.0, 4.0, 6.0, 1.0], 2, 6)` doit renvoyer $(2, 4, 5, 4)$, car entre les indices 2 (inclus) et 6 (exclus), le saut de valeur maximale est $(2, 4)$ (de valeur 3.0), le minimum est 1.0 et le maximum est 6.0.

Note : l'efficacité de l'algorithme proposé sera valorisée.

- **Question 17** En déduire une fonction `saut_max(li)` qui renvoie un saut de valeur maximale d'une liste `li`.

Problème 2 Résolution de logigrammes

Partie III Cases, blocs et indications

A. Indications

On considère une ligne `li` constituée de cases qui peuvent être blanches ou noires. On notera $|li|$ la taille d'une telle ligne.

Les lignes sont représentées en Python comme des listes contenant des 0 (pour indiquer une case blanche) ou des 1 (pour indiquer une case noire).

Un *bloc* dans cette ligne est une succession maximale de cases noires, c'est-à-dire qu'un bloc est toujours précédé et suivi par une case blanche, ou une extrémité de la ligne. La *longueur* d'un bloc est le nombre de cases noires qu'il contient. La *valeur* d'une ligne `li` est le nombre total de cases noires qu'elle contient, que l'on notera $v(li)$. L'*indication* de cette ligne est la liste ordonnée de gauche à droite des longueurs de tous les blocs de la ligne, comme illustré sur la figure 1. On notera cette indication $\chi(li)$, et on appellera *valeur d'une indication* (notée également $v(\chi(li))$) la somme de ses éléments.

Dans toute cette partie, on notera $|\chi(li)|$ le nombre d'éléments dans son indication.



```
ligne_ex = [1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0]
indication_ex = [3, 5, 1]
```

Figure 1: exemple d'une ligne et son indication. La ligne est de *taille* 15 et de *valeur* 9.

- **Question 18** Donner un exemple d'une ligne distincte de `ligne_ex` (donné en figure 1) mais ayant la même taille et la même indication.

- **Question 19** Écrire une fonction `valeur_ligne(li)` qui prend en argument une ligne `li` et renvoie sa valeur $v(li)$ comme définie ci-dessus.

- **Question 20** Écrire une fonction `indication(li)` qui prend en argument une ligne `li` et renvoie son indication $\chi(li)$. La fonction proposée devra avoir une complexité en $O(|li|)$.

- **Question 21** Justifier que pour toute ligne `li`, on a $v(li) \leq |li| + 1 - |\chi(li)|$, et donner un exemple où il y a égalité.

B. Lignes réduites

Dans cette sous-partie, on fixe une longueur n et une indication \mathcal{I} , et on souhaite trouver toutes les lignes de taille n ayant cette indication.

Étant donnée une ligne li telle que $\chi(li) = \mathcal{I}$, on construit la *ligne réduite* $\phi(li)$ de la manière suivante :

- on réduit la taille de chaque bloc de li à 1 en supprimant autant de cases noires qu'il le faut,
- puis, dans chaque intervalle de cases blanches entre deux blocs, on supprime exactement une case blanche. On ne modifie pas les intervalles de cases blanches aux extrémités.

► **Question 22** Donner la ligne réduite $\phi(ligne_ex)$ de la ligne de la figure 1.

► **Question 23** Écrire une fonction `ligne_reduite(li)` qui prend en argument une ligne li et renvoie la ligne réduite $\phi(li)$.

► **Question 24** Pour une ligne li quelconque, exprimer la taille de la ligne réduite $|\phi(li)|$ en fonction de $|li|$ et de $v(li)$. Vous justifierez votre réponse.

► **Question 25** Soient k un entier, \mathcal{I} une indication, et li_r une ligne de taille k possédant exactement $|\mathcal{I}|$ cases noires. Montrer que li_r est la ligne réduite d'une ligne de taille $k + v(\mathcal{I}) - 1$ et d'indication \mathcal{I} .

► **Question 26** Écrire une fonction `reconstruire_ligne(li_r, indic)` qui prend en argument une ligne réduite li_r et une indication `indic`, et renvoie une ligne dont la ligne réduite est li_r et dont l'indication est `indic`, lorsqu'une telle ligne existe. Si aucune ligne ne convient, aucun comportement particulier n'est exigé.

► **Question 27** Soient deux lignes l_1 et l_2 de même taille ($|l_1| = |l_2|$), même indication ($\chi(l_1) = \chi(l_2)$), et même ligne réduite ($\phi(l_1) = \phi(l_2)$). Montrer que $l_1 = l_2$.

► **Question 28** Soient n un entier et \mathcal{I} une indication. On note E l'ensemble des lignes de taille n et d'indication \mathcal{I} . Déduire des questions précédentes que la fonction ϕ réalise une bijection entre E et un autre ensemble que l'on précisera, et en conclure que E est de cardinal $\binom{n - v(\mathcal{I}) + 1}{|\mathcal{I}|}$, où $\binom{n}{p}$ désigne le nombre de combinaisons de p éléments parmi n (à savoir $\frac{n!}{p!(n-p)!}$).

C. Énumération de lignes

Soient \mathcal{I} une indication et k un entier. On souhaite énumérer toutes les lignes réduites de taille k compatibles avec \mathcal{I} , c'est-à-dire toutes les listes de taille k , contenant exactement $|\mathcal{I}|$ cases noires et $k - |\mathcal{I}|$ cases blanches. On admettra qu'il est possible d'énumérer toutes ces lignes réduites en commençant par celle dont les 0 sont à gauche (c'est-à-dire qu'elle est de la forme $[0, \dots, 0, 1, \dots, 1]$) puis en passant d'une ligne réduite à la suivante à l'aide de la fonction `next(li)` donnée ci-dessous.

Note : on ne demande pas de prouver que cette fonction est correcte.

```
def next(li):
    """Renvoie la ligne réduite suivant li dans l'ordre
    d'énumération des lignes réduites."""
    k = len(li)
    for i in range(k-1, 0, -1):
        if li[i] == 1 and li[i-1] == 0:
            return li[0:i-1] + [1, 0] + li[k-1:i:-1]
    return []
```

► **Question 29** Que renvoie `next([0, 0, 1, 1, 1])` ? Que renvoie `next([1, 1, 1, 0, 0])` ?

► **Question 30** Compléter la fonction `liste_lignes_reduites(nb_noires, k)`, donnée ci-dessous, qui renvoie la liste des lignes réduites de taille k ayant `nb_noires` cases noires.

Note : vous écrirez les lignes 6, 7 et 8 de la fonction sur votre copie.

```
1. def liste_lignes_reduites(nb_noires, k):
2.     # Première ligne réduite
3.     li = [0] * (k-nb_noires) + [1] * nb_noires
4.     # Liste des lignes réduites
5.     res = []
6.     while ...:
7.         res.append(...)
8.         li = ...
9.     return res
```

► **Question 31** Écrire une fonction `liste_lignes(indic, n)` qui renvoie la liste des lignes de longueur n et d'indication `indic`.

Partie IV Jeu du logigramme

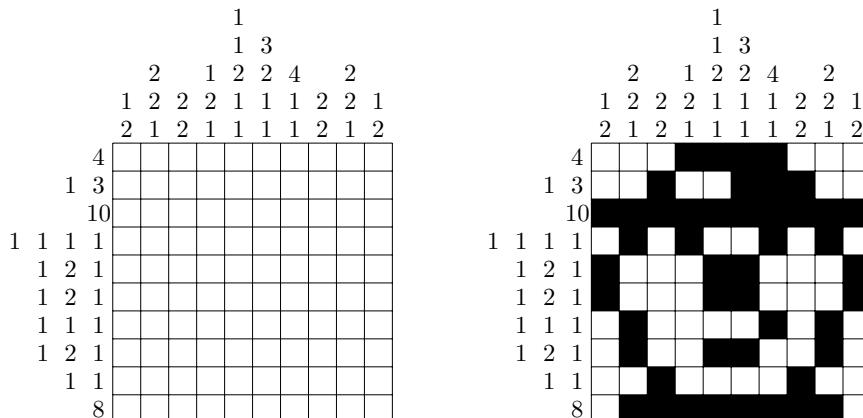


Figure 2: Exemple de *puzzle* et sa résolution pour le jeu du logigramme.

Un *puzzle* est la donnée de deux listes dont les éléments sont des indications (on rappelle qu'une indication est une liste d'entiers ; autrement dit un *puzzle* est un couple de listes de listes d'entiers).

Si on note n et p le nombre d'indications dans chaque liste respectivement, le jeu consiste à remplir une matrice de n lignes et p colonnes avec des cases noires ou blanches, de telle sorte que chaque ligne et chaque colonne correspondent à l'indication associée (voir figure 2).

En Python, les indications sur les lignes `indic_lignes` seront données sous forme d'une liste des indications sur les lignes, lues de haut en bas. De même pour `indic_colonnes` pour les colonnes,

lues de gauche à droite. Le puzzle est donné sous forme du tuple (`indic_lignes`, `indic_colonnes`) comme sur l'exemple de la figure 3.

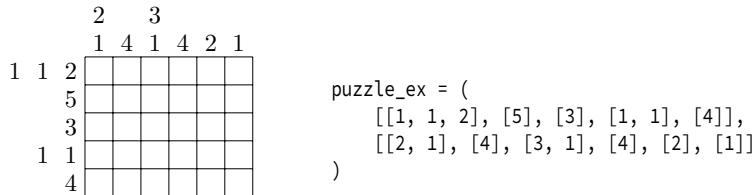


Figure 3: Exemple du puzzle `puzzle_ex` et sa représentation en Python.

- ▶ **Question 32** Donner un exemple de puzzle avec $n = p = 2$ qui n'admette aucune solution.
- ▶ **Question 33** Donner un exemple de puzzle avec $n = p = 2$ qui admette plusieurs solutions, et donner toutes ses solutions.

Une grille de solution sera représentée par une matrice à n lignes et p colonnes, et on inscrit dans chaque case :

- un 0 si la case est blanche ;
- un 1 si la case est noire ;
- et -1 enfin si la couleur de la case n'est pas encore déterminée.

▶ **Question 34** Écrire une fonction `grille_vide(puzzle)` qui prend en argument un puzzle et renvoie une grille ayant la bonne taille, mais ne contenant que des -1 (toutes les cases sont indéterminées).

▶ **Question 35** Écrire une fonction `nb_cases_ind(grille)` qui prend en argument une grille et renvoie le nombre de cases indéterminées dans la grille.

▶ **Question 36** Écrire une fonction `est_valide(puzzle, grille)` qui prend en entrée un puzzle et une grille de dimension adaptée dont toutes les cases sont déterminées, et renvoie `True` si la grille est une solution du puzzle, et `False` sinon.

▶ **Question 37** Donner la complexité de la fonction `est_valide` en fonction de n et p , les dimensions de la grille.

Partie V Remplissage

On suppose qu'on dispose d'une grille partiellement remplie, c'est-à-dire de la bonne dimension et pouvant contenir des cases blanches, des cases noires, et des cases indéterminées. L'objectif de cette partie sera de compléter au maximum la grille en utilisant la technique suivante : on regarde toutes les manières de compléter une ligne (respectivement une colonne), et si une case est toujours coloriée de la même manière, on complète la grille en fixant cette couleur dans cette case. On rappelle qu'on peut générer la liste des lignes d'une longueur donnée et pour une indication donnée avec la fonction `liste_lignes(indic, n)` de la question 31.

► **Question 38** Écrire une fonction `liste_lignes_compatibles(indic, ligne_partielle)` qui prend en arguments une indication, une ligne partiellement remplie d'une grille, et renvoie la liste de toutes les lignes ayant comme indication `indic` et compatibles avec les cases déjà remplies dans `ligne_partielle`.

Ainsi, on devrait obtenir les valeurs suivantes, comme sur la figure 4.

```
>>> liste_lignes_compatibles([2, 2], [-1]*6)
[[1, 1, 0, 1, 1, 0],
 [1, 1, 0, 0, 1, 1],
 [0, 1, 1, 0, 1, 1]]
>>> liste_lignes_compatibles([2, 2], [1, -1, 0, -1, -1, -1])
[[1, 1, 0, 1, 1, 0],
 [1, 1, 0, 0, 1, 1]]
```

Ligne partielle	<table border="1"><tr><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr></table>	?	?	?	?	?	?	<table border="1"><tr><td>██████</td><td>?</td><td>██████</td><td>?</td><td>?</td><td>?</td></tr></table>	██████	?	██████	?	?	?																								
?	?	?	?	?	?																																	
██████	?	██████	?	?	?																																	
Lignes possibles	<table border="1"><tr><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td></tr><tr><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td></tr><tr><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td></tr></table>	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	<table border="1"><tr><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td></tr><tr><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td></tr><tr><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td><td>██████</td></tr></table>	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████	██████
██████	██████	██████	██████	██████	██████																																	
██████	██████	██████	██████	██████	██████																																	
██████	██████	██████	██████	██████	██████																																	
██████	██████	██████	██████	██████	██████																																	
██████	██████	██████	██████	██████	██████																																	
██████	██████	██████	██████	██████	██████																																	

Figure 4: Liste des lignes compatibles avec l'indication `[2, 2]` et une ligne partielle. Les points d'interrogation sont les cases indéterminées.

► **Question 39** Écrire une fonction `remplir_ligne(indic, ligne)` qui prend en argument une indication, une ligne partiellement remplie, et renvoie une copie de `ligne` complétée au maximum, comme dans l'exemple ci-dessous :

```
>>> remplir_ligne([2, 2], [-1]*6)
[-1, 1, -1, -1, 1, -1]
>>> remplir_ligne([2, 2], [1, -1, 0, -1, -1, -1])
[1, 1, 0, -1, 1, -1]
```

► **Question 40** Écrire une fonction `completer_lignes(puzzle, grille)` qui applique cette méthode sur chaque ligne et modifie `grille` en fonction. Cette fonction ne renvoie rien.

► **Question 41** On exécute le programme suivant (`puzzle_ex` est donné dans la figure 3) :

```
grille = grille_vide(puzzle_ex)
completer_lignes(puzzle_ex, grille)
```

Donner alors la valeur de `grille`.

► **Question 42** On suppose qu'on dispose d'une fonction `completer_colonnes(puzzle, grille)` au fonctionnement similaire à la fonction `completer_lignes`. Écrire une fonction `completer(puzzle, grille)` qui applique ces deux fonctions sur la grille, jusqu'à ce que la grille ne soit plus modifiée.

Partie VI Retour sur trace

On s'intéresse dans cette partie à un algorithme plus général permettant de résoudre n'importe quelle grille.

► **Question 43** Que donnerait la fonction `completer` sur le puzzle `([[1, 1], [2]], [[1], [1], [1], [1]])` et avec une grille vide ? Donner sans justification l'unique solution de ce puzzle.

► **Question 44** La méthode du retour sur trace (ou *backtracking*) consiste, quand la méthode précédente (implémentée par la fonction `completer`) ne permet pas de résoudre une grille, à faire une hypothèse sur une case indéterminée de la grille : on essaye de terminer la résolution en supposant cette case blanche, puis en supposant cette case noire. Il est possible que cette hypothèse ne suffise pas, on formulera alors une autre hypothèse sur une autre case. Il est possible qu'une suite d'hypothèses aboutisse à une impasse, ce qui se traduira par la fonction `remplir_ligne` qui renvoie une liste vide. Dans ce cas, il convient d'abandonner cette suite d'hypothèses et d'en tester une autre.

Programmer la méthode du retour sur trace pour ce problème.

Note 1 : cette question est relativement ouverte, les réponses partielles seront valorisées si elles sont pertinentes.

Note 2 : l'algorithme décrit dans la partie V n'est pas le plus efficace : la génération des lignes possibles est ici exponentielle alors qu'on peut déterminer les cases nécessairement noires ou blanches avec une complexité cubique en la taille de la ligne. Aucune méthode n'est en revanche connue pour résoudre les puzzles en temps polynomial, ce problème fait partie des problèmes NP-complets.

Épreuve disciplinaire appliquée

Préambule : cette épreuve est constituée de deux parties A et B indépendantes. Les réponses aux questions doivent être précises et rédigées avec soin.

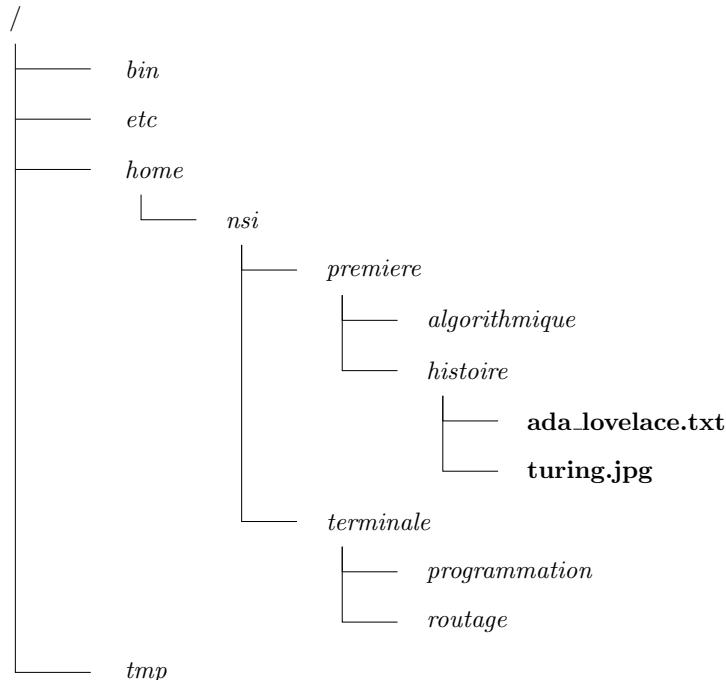
Certaines questions demandent des productions d'enseignement à destination des élèves (cours, exercices, projets, activités sur machines, activités débranchées, etc.). Ces questions devront être traitées avec la plus grande attention.

Partie A - Systèmes d'exploitation et processus

Dans cette partie, on se propose d'étudier les systèmes d'exploitation et les processus, notions qui font partie des programmes de NSI de première et de terminale. Elle est composée de deux sous-parties, l'une sur les systèmes d'exploitation et l'autre sur les processus et leur ordonnancement.

1 Systèmes d'exploitation

1. Donner la définition d'un système d'exploitation et préciser trois de ses principales fonctionnalités.
2. Donner un exemple de système d'exploitation propriétaire et un exemple de système d'exploitation libre.
3. Donner deux caractéristiques principales des systèmes Unix récents (compatibles avec le standard POSIX).
4. Dans un système Unix, un utilisateur se voit attribuer un UID et un GID. Expliquer à quoi cela correspond.
5. Préciser à quoi correspond le super-utilisateur d'un système d'exploitation de type Unix. Donner son UID et son GID utilisés par convention dans les systèmes Unix.
6. Dans un système d'exploitation de type Unix, on considère l'arborescence des fichiers suivante dans laquelle les noms de répertoires sont en italique et ceux des fichiers sont en gras :



On se place dans un terminal affichant une invite de commande et on souhaite explorer et modifier, en lignes de commande, les répertoires et fichiers présents.

On suppose que le répertoire de travail est `/home/nsi/terminale`. Pour cette question, on considère que les commandes sont exécutées par l'utilisateur `nsi` et que ce dernier est propriétaire de tous les fichiers et répertoires de l'arborescence `/home/nsi`.

- (a) Donner la commande permettant d'afficher le répertoire de travail.
- (b) Donner l'affichage correspondant à l'utilisation de la commande `ls`.
- (c) Donner la commande qui permet de changer le répertoire de travail pour que ce soit le répertoire *histoire*.

Pour la suite de l'exercice, le répertoire de travail est donc `/home/nsi/premiere/histoire`. Le résultat de la commande `ls -l` est le suivant :

```
-r--r---- 1 nsi spe 10133 sep. 7 10:07 ada_lovelace.txt
-rw-r---- 1 nsi spe 124158 sep. 5 18:33 turing.jpg
```

- (d) Préciser les droits que l'utilisateur `nsi` possède et ceux qu'il ne possède pas sur le fichier `ada_lovelace.txt`.
- (e) Écrire la commande permettant de modifier les droits (et ce, récursivement) de l'ensemble des fichiers et répertoires contenus dans le répertoire *histoire*, de telle sorte que l'utilisateur `nsi` ait tous les droits et que tous les autres utilisateurs du système n'en aient aucun.

On supposera dans la suite que l'utilisateur `nsi` a tous les droits dans le répertoire *histoire*.

- (f) Écrire la commande qui permet de créer dans le répertoire de travail un répertoire nommé *expose*.
- (g) Écrire la commande qui permet de supprimer le fichier `turing.jpg`.
- (h) Écrire la commande permettant de déplacer le fichier `ada_lovelace.txt` dans le répertoire *expose*.

7. On suppose dans cette question qu'on dispose d'une salle de TP équipée de machines dont le système d'exploitation est de type Unix.

L'identifiant de l'administrateur et le mot de passe associé sont identiques pour toutes les machines :

identifiant administrateur : adm

mot de passe administrateur : Turing

On souhaite, dans cette question, proposer un exercice à destination d'élèves de première NSI leur permettant d'écrire un script Shell dont le but est d'éteindre tous les ordinateurs de la salle informatique à partir d'une machine extérieure à cette salle mais interconnectée aux ordinateurs de la salle informatique grâce à un réseau informatique. On suppose que la connexion se fait en SSH et que celle-ci peut se faire sans mot de passe car la clé publique de l'utilisateur qui va exécuter le script a déjà été diffusée aux machines à éteindre.

- (a) Indiquer quelles sont les entrées dont aura besoin le script Shell.
- (b) Déterminer les actions que devra comprendre le script Shell envisagé.
- (c) Pour chaque action, préciser la ou les commandes Unix correspondantes.
- (d) Proposer un énoncé progressif à destination d'élèves de première NSI leur permettant d'aboutir à la construction de ce script.
- (e) Proposer une correction à cet énoncé.

8. Votre classe se montre très intéressée par les systèmes d'exploitation. Vous décidez d'aller un peu plus loin sur le sujet par rapport à ce qui est préconisé dans le programme.

- (a) Écrire l'énoncé d'un exercice que vous proposeriez à vos élèves afin qu'elles et ils apprennent à utiliser les caractères spéciaux « ? » et « * » dans les commandes Unix.
- (b) Indiquer à vos élèves quelles sont les trois entrées-sorties standards.
- (c) Donner à vos élèves un exemple de commande qui modifie une de ces entrées-sorties standards.
- (d) Donner une commande qui illustre la redirection entre commandes. Expliquer le fonctionnement de cette commande tel que vous le ferez avec vos élèves.

2 Processus

On trouvera dans l'**annexe 5.1** le programme de terminale NSI sur les processus. Dans cette section, on suppose qu'on travaille avec un système Unix conforme au standard POSIX.

2.1 Définitions

9. Donner la définition d'un processus.

10. Un processus est caractérisé par :

- un espace mémoire,
- un ensemble de ressources.

Préciser le sens de ces deux notions.

11. Indiquer la différence fondamentale en terme de gestion de la mémoire entre un processus et un processus léger (thread en anglais).

Dans toute la suite de cette partie, on ne considérera que des processus et aucun processus léger. Sauf mention explicite du contraire, les machines utilisées dans cet exercice ne disposent que d'une seule unité arithmétique et logique (un seul processeur). Ainsi un seul processus peut être exécuté à la fois.

12. Un processus ne peut être lancé que par un autre processus. Donner les noms généralement associés aux deux processus.
13. Donner le nom du seul processus qui ne suit pas cette règle.
14. À un instant t , un processus est caractérisé par son état qui peut être, entre autres, :
- Prêt ;
 - Bloqué (aussi appelé En attente) ;
 - Élu (aussi appelé En exécution).

Rappeler en quoi consiste chacun de ces trois états.

15. Proposer une représentation graphique que vous proposeriez à vos élèves pour illustrer les interactions entre ces différents états.

2.2 Visualisation des processus sur un système d'exploitation de type Unix

16. La copie d'écran donnée en figure 1 est le résultat de la commande `top` lancée dans un terminal.

PID UTIL.	PR S	%CPU	%MEM	TEMPS+ COM.	PPID
1562 nsi	20 S	16,9	1,6	0:27.69 Xorg	1560
1700 nsi	20 R	14,3	6,0	0:35.77 gnome-shell	1472
2667 nsi	20 S	11,3	1,8	0:03.99 kazam	1700
2710 nsi	20 S	4,0	1,4	0:03.84 thonny	1700
2301 nsi	20 S	0,7	1,4	0:02.39 gnome-terminal-	1472
2974 nsi	20 S	0,7	2,6	0:00.77 WebExtensions	2662
17 root	20 I	0,3	0,0	0:00.86 kworker/0:1-events	2
50 root	20 I	0,3	0,0	0:00.96 kworker/2:1-events	2
129 root	20 I	0,3	0,0	0:00.82 kworker/1:2-events	2
402 root	0 I	0,3	0,0	0:00.78 kworker/u17:1-i915_flip	2
429 root	20 I	0,3	0,0	0:00.62 kworker/3:3-events	2
2160 nsi	20 S	0,3	2,9	0:08.01 okular	1472
2178 nsi	20 S	0,3	1,1	0:00.36 kglobaccel5	1472
2754 nsi	20 R	0,3	0,1	0:00.19 top	2745
1 root	20 S	0,0	0,3	0:02.02 systemd	0
2 root	20 S	0,0	0,0	0:00.00 kthreadd	0
3 root	0 I	0,0	0,0	0:00.00 rcu_gp	2

FIGURE 1 – Commande `top`

- (a) Préciser le rôle de la commande `top`.
- (b) Donner la commande à utiliser pour afficher le manuel de la commande `top` dans un terminal affichant une invite de commande.
- (c) En s'aidant des extraits du manuel de la commande `top` donnés en **annexe 5.2**, préciser en quoi consiste chacun des descripteurs se trouvant sur la ligne surlignée en blanc commençant par les termes PID UTIL.
17. Proposer une activité à destination des élèves de terminale permettant d'illustrer le principe d'arborescence des processus en utilisant le résultat de la commande `ps -ef` lancée dans un terminal. Un exemple de résultat de cette commande est donné en figure 2.

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	oct.24	?	00:00:22	/sbin/init splash
root	2	0	0	oct.24	?	00:00:00	[kthreadd]
root	3	2	0	oct.24	?	00:00:00	[rcu_gp]
root	4	2	0	oct.24	?	00:00:00	[rcu_par_gp]
root	5	2	0	oct.24	?	00:00:00	[netns]
root	7	2	0	oct.24	?	00:00:00	[kworker/0:0H-events_highpri]
root	9	2	0	oct.24	?	00:00:11	[kworker/0:1H-kblockd]
root	10	2	0	oct.24	?	00:00:00	[mm_percpu_wq]
root	11	2	0	oct.24	?	00:00:00	[rcu_tasks_rude_]
root	12	2	0	oct.24	?	00:00:00	[rcu_tasks_trace]
root	13	2	0	oct.24	?	00:00:02	[ksoftirqd/0]
root	14	2	0	oct.24	?	00:00:54	[rcu_sched]
root	15	2	0	oct.24	?	00:00:01	[migration/0]
root	16	2	0	oct.24	?	00:00:00	[idle_inject/0]
root	18	2	0	oct.24	?	00:00:00	[cpuhp/0]
root	19	2	0	oct.24	?	00:00:00	[cpuhp/1]
root	20	2	0	oct.24	?	00:00:00	[idle_inject/1]
<hr/>							
nsi	59382	1347	0	14:42	?	00:00:02	eog /home/nsi/Images/top2.pn
root	59413	2	0	14:43	?	00:00:00	[kworker/u16:0-iwlwifi]
root	59424	2	0	14:44	?	00:00:00	[kworker/u17:2-rb_allocator]
root	59425	2	0	14:45	?	00:00:00	[kworker/0:0-events]
root	59489	2	0	14:45	?	00:00:00	[kworker/3:2-events]
root	59606	2	0	14:46	?	00:00:00	[kworker/1:0-events]
root	59683	2	0	14:47	?	00:00:00	[kworker/2:0-events]
root	59817	2	0	14:50	?	00:00:00	[kworker/0:1-events]
root	59819	2	0	14:50	?	00:00:00	[kworker/u17:0]
root	59826	2	0	14:50	?	00:00:00	[kworker/u16:3-iwlwifi]
nsi	59864	1347	2	14:50	?	00:00:01	/usr/libexec/gnome-terminal-
nsi	59872	59864	0	14:50	pts/1	00:00:00	bash
nsi	59879	59872	9	14:50	pts/1	00:00:03	gimp
nsi	59896	1347	1	14:51	?	00:00:00	/usr/bin/gjs /usr/share/org.
nsi	59897	1347	0	14:51	?	00:00:00	/usr/libexec/gnome-control-c
nsi	59922	1588	8	14:51	?	00:00:01	/usr/bin/python3 /usr/bin/th
nsi	59935	59879	2	14:51	pts/1	00:00:00	/usr/lib/gimp/2.0/plug-ins/s
nsi	59950	59864	0	14:51	pts/2	00:00:00	bash
nsi	59958	59922	4	14:51	?	00:00:00	/usr/bin/python3 -u -B /usr/
nsi	59963	59950	0	14:51	pts/2	00:00:00	ps -ef

FIGURE 2 – Résultat de la commande `ps -ef`

2.3 Ordonnancement

L'ordonnancement est l'action qui consiste pour une machine à choisir, au début de chaque unité de temps d'exécution, le processus qui va être exécuté par la machine parmi l'ensemble des différents processus « prêts ». L'ordonnancement est réalisé par l'ordonnanceur du système d'exploitation.

Dans la suite de l'exercice sur le problème d'ordonnancement, on suppose qu'un processus peut être caractérisé par :

- sa **durée d'exécution** exprimée en unités de temps d'exécution (nombre entier) ;
- son **instant d'arrivée** correspondant à l'instant où le processus est déclaré comme « prêt » pour la première fois ;
- sa **position dans la file d'attente** de l'ordonnanceur exprimée par un entier, 1 correspondant à la première position.

Définition 1 (Temps d'attente et temps de réponse).

Deux grandeurs peuvent être calculées sur les processus :

- le temps d'attente qui correspond à la durée écoulée entre l'instant d'arrivée et l'instant de début d'exécution du processus ;
- le temps de réponse qui correspond à la durée écoulée entre l'instant d'arrivée et l'instant de fin d'exécution du processus.

18. Parmi les trois paramètres choisis pour caractériser un processus (durée d'exécution, instant d'arrivée, position dans la file d'attente), indiquer le paramètre qui est *a priori* inconnu de la plupart des systèmes d'exploitation.
19. Proposer un exemple permettant de comprendre l'intérêt d'un ordonnancement intelligent des processus afin d'optimiser leur temps de réponse en s'inspirant de l'ordonnancement de tâches de la vie courante.

2.3.1 Ordonnancement par ordre de soumission

Dans l'ordonnancement par ordre de soumission (premier arrivé, premier servi), les processus « prêts » sont choisis selon l'ordre dans lequel ils arrivent dans la file d'attente de l'ordonnanceur. Le processus choisi s'exécute, soit jusqu'à ce qu'il

soit terminé, soit jusqu'à ce qu'il se bloque de lui-même car il lui manque une ressource. Il reste bloqué tant qu'il n'a pas obtenu cette ressource.

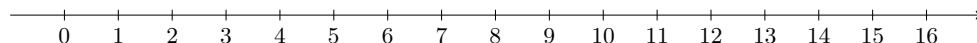
On donne ci-dessous un exercice à destination d'élèves de terminale NSI.

Exercice 1 (Mise en œuvre de l'ordonnancement par ordre de soumission sur trois processus).

On considère trois processus P1, P2 et P3 soumis à l'ordonnanceur dont les caractéristiques sont les suivantes :

Nom du processus	Durée d'exécution	Instant d'arrivée
P1	8	0
P2	2	5
P3	3	7

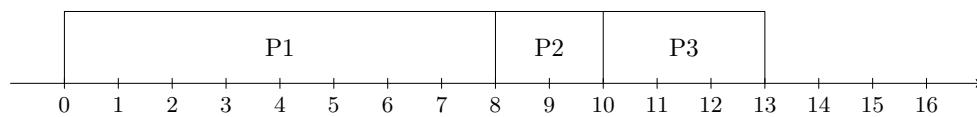
1. Tracer le chronogramme d'exécution de ces trois processus. On supposera que les processus ne se bloquent pas et qu'une unité de temps d'exécution d'un processus correspond à une unité de temps de l'échelle temporelle donnée ci-dessous.



2. Déterminer les temps d'attente et de réponse des trois processus P1, P2 et P3.

20. Voici la réponse attendue par l'enseignant :

1.

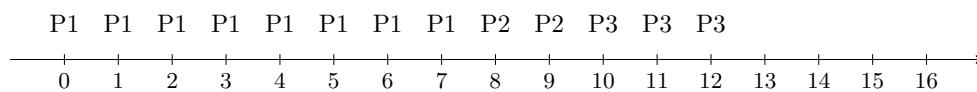


2.

temps d'attente de P1 : 0 temps de réponse de P1 : 8
 temps d'attente de P2 : 3 temps de réponse de P2 : 5
 temps d'attente de P3 : 3 temps de réponse de P3 : 6

Voici la proposition d'un élève :

1.



2.

temps d'attente de P1 : 0 temps de réponse de P1 : 8
 temps d'attente de P2 : 3 temps de réponse de P2 : 4
 temps d'attente de P3 : 3 temps de réponse de P3 : 5

Commenter la justesse de sa réponse.

21. Proposer une correction pour cet élève en expliquant pourquoi la représentation choisie par l'enseignant est plus judicieuse que celle proposée par l'élève.

2.3.2 Autres ordonnancements

22. Sur le même principe que l'exercice précédent, proposer une activité à destination d'élèves de terminale NSI permettant d'illustrer les trois différents algorithmes d'ordonnancement décrits ci-dessous.

On supposera que, dans cet exercice, les configurations des processus seront les mêmes pour les différents ordonnancements et que les processus ne se bloquent pas à cause d'un manque de ressource.

On donnera également la correction de cette activité.

L'ordonnancement par tourniquet : Le processus élu s'exécute soit pendant un temps donné Q prédéfini appelé quantum (il retourne alors à l'état « prêt » et réintègre la file d'attente de l'ordonnanceur en dernière position), soit jusqu'à ce qu'il soit terminé (durée d'exécution restante inférieure à Q), soit jusqu'à ce qu'il se bloque de lui-même car il lui manque une ressource.

L'ordonnancement par priorité non préemptive : À chaque processus est associé un **numéro de priorité** : le numéro de priorité d'un processus est un entier positif d'autant plus petit que la priorité est grande.

Lorsqu'il doit choisir un processus à élire, l'ordonnanceur choisit celui qui a la priorité la plus grande dans la file d'attente et l'exécute, soit jusqu'à ce qu'il soit terminé, soit jusqu'à ce qu'il se bloque de lui-même car il lui manque une ressource.

L'ordonnancement par priorité préemptive : À chaque processus est associé un **numéro de priorité** : le numéro de priorité d'un processus est un entier positif d'autant plus petit que la priorité est grande.

Au début de chaque unité de temps d'exécution, l'ordonnanceur choisit le processus qui a la priorité la plus grande et l'exécute, ce qui peut provoquer la suspension d'un autre processus en cours de traitement, lequel reprendra lorsqu'il sera le plus prioritaire parmi la file d'attente de l'ordonnanceur.

2.3.3 Prise en compte des ressources

On donne ci-dessous un exercice à destination d'élèves de terminale NSI.

Exercice 2 (Utilisation d'une ressource par plusieurs processus).

On considère trois processus P1, P2 et P3 qui vont être ordonnancés selon la politique du tourniquet avec un quantum Q de 2.

À l'instant initial, ces trois processus ont différentes caractéristiques données ci-dessous :

Nom du processus	Durée d'exécution	Position dans la file d'attente
P1	7	2
P2	6	1
P3	11	3

On a de plus les contraintes suivantes :

- Lors de son exécution, le processus P1 a besoin de la ressource R1 du début de sa 3^e unité de temps d'exécution jusqu'à la fin de sa 7^e unité de temps d'exécution.
 - Lors de son exécution, le processus P2 a besoin de la ressource R1 du début de sa 2^e unité de temps d'exécution jusqu'à la fin de sa 5^e unité de temps d'exécution.
 - Lors de son exécution, le processus P3 a besoin de la ressource R1 du début de sa 5^e unité de temps d'exécution jusqu'à la fin de sa 9^e unité de temps d'exécution.
1. Tracer le chronogramme d'exécution de ces trois processus
 2. Tracer le chronogramme d'allocation de la ressource R1.
 3. Donner les temps d'attente et de réponse des trois processus P1, P2 et P3.

23. Proposer une correction de cet exercice permettant de comprendre ce qu'il se passe par une lecture visuelle.

24. Rappeler en quoi consiste le risque d'interblocage.

25. Proposer une activité débranchée permettant de l'illustrer.

2.4 Synthèse

26. Proposer un plan de cours détaillé concernant la gestion des processus et des ressources par un système d'exploitation pour une classe de terminale NSI. Ce plan de cours doit détailler les notions qui seront abordées, la chronologie d'enchaînement de ces notions ainsi qu'une indication sur le temps prévu sur chacune de ces notions. Les éventuels pré-requis ainsi que le type d'activités envisagées seront précisés. Enfin, les objectifs d'apprentissage seront listés.

2.5 Projet : simulation d'un ordonnanceur

Un groupe de trois élèves en classe de terminale NSI souhaite simuler un ordonnanceur en Python permettant d'implémenter les différents ordonnancements vus en cours (et abordés dans la section 2.3 de ce sujet). Ils proposent d'implémenter les processus en programmation orientée objet et souhaitent utiliser une classe File pour implémenter la file d'attente de l'ordonnanceur.

27. (a) Une de vos collègues a proposé une implémentation possible d'un processus via la classe suivante :

```

class Processus:
    PRET = 0
    ELU = 1
    BLOQUE = 2
    def __init__(self, nom, priorite, duree_execution, instant_arrivee):
        self.nom = nom
        self.priorite = priorite
        self.duree_execution = duree_execution
        self.instant_arrivee = instant_arrivee
        self.etat = Processus.PRET

```

Commenter cette classe tel que vous le feriez à vos élèves.

- (b) Proposer une implémentation de la classe File utilisant la classe standard **list** dont quelques méthodes sont données en **annexe 5.3**. Proposer un exemple d'utilisation de la classe File définie.
28. Les élèves rencontrent des difficultés en voulant programmer l'ordonnancement par priorité préemptive. Après quelques recherches, ils découvrent l'existence d'une classe FileDePriorite sur un forum en ligne.
- (a) Proposer une interface pour cette classe.
- (b) Proposer une implémentation possible de cette classe utilisant la classe File définie précédemment et la classe standard **dict** dont quelques méthodes sont données en **annexe 5.4**. Proposer un exemple d'utilisation de la classe FileDePriorite définie.
29. Identifier les points d'étapes que vous proposeriez à ce groupe d'élèves pour suivre le projet.
30. On rappelle ci-dessous les compétences constitutives de la pensée informatique proposées dans les programmes de première et terminale NSI :
- analyser et modéliser un problème en termes de flux et de traitement d'informations ;
 - décomposer un problème en sous-problèmes ;
 - reconnaître des situations déjà analysées et réutiliser des solutions ;
 - concevoir des solutions algorithmiques ;
 - traduire un algorithme dans un langage de programmation, en spécifier les interfaces et les interactions, comprendre et réutiliser des codes sources existants, développer des processus de mise au point et de validation de programmes ;
 - mobiliser les concepts et les technologies utiles pour assurer les fonctions d'acquisition, de mémorisation, de traitement et de diffusion des informations ;
 - développer des capacités d'abstraction et de généralisation.
- On rappelle ci-après les compétences transversales qui peuvent être développées en NSI :
- faire preuve d'autonomie, d'initiative et de créativité ;
 - présenter un problème ou sa solution, développer une argumentation dans le cadre d'un débat ;
 - coopérer au sein d'une équipe dans le cadre d'un projet ;
 - rechercher de l'information, partager des ressources ;
 - faire un usage responsable et critique de l'informatique.

Proposer une grille d'évaluation pour évaluer le travail en projet de ce groupe d'élèves. Vous expliquerez en quoi le projet fait appel à certaines des compétences listées ci-dessus.

Partie B - Gestion de données

Dans ce problème, on se propose d'étudier la gestion des données à travers le continuum des programmes de SNT, première NSI et terminale NSI.

Dans une première partie, on étudie la façon dont sont organisées les données. Dans une seconde partie, on étudie le traitement de ces données.

3 Organisation des données

31. Proposer une activité sans ordinateur à destination d'une classe de SNT permettant d'introduire les concepts de donnée, de descripteur et de collection et conduisant à la notion de données structurées. On précisera le temps prévu pour cette activité.
32. Définir ce que sont les métadonnées d'un fichier et en donner deux exemples.
33. Donner deux exemples de format de fichiers contenant des données structurées.
34. (a) Définir ce qu'est l'indexation de données.
 (b) Proposer une activité sans ordinateur à destination d'une classe de SNT permettant d'expliquer l'indexation de données. On précisera le temps prévu pour cette activité.
35. Pour illustrer l'impact des centres de données (datacenters) sur les pratiques humaines, et pour préparer vos élèves au grand oral, vous décidez d'organiser un débat dans votre cours de SNT en proposant à vos élèves l'exercice suivant :

Exercice 3. Certains magasins proposent de remplacer le ticket de caisse papier par un ticket dématérialisé, envoyé par mail ou sur une application dédiée.
 Êtes-vous pour ou contre cette démarche ? Argumentez votre réponse.

- (a) Donner un argument pour et un argument contre que pourraient proposer vos élèves pour défendre chacun des deux points de vue.
 - (b) On rappelle ci-après les cinq compétences évaluées lors du Grand Oral :
 - qualité de la prise de parole en continu ;
 - qualité orale ;
 - qualité des connaissances ;
 - qualité et construction de l'argumentation ;
 - qualité de l'interaction.
- Dans l'optique de préparer vos élèves à cette épreuve, vous en interrogez deux pour débattre à l'oral sur cet exercice, chacun défendant un point de vue qui lui est imposé. Proposer une grille d'évaluation avec un barème précis s'appuyant sur ces cinq compétences.
36. En classe de première NSI, le programme demande de savoir importer une table depuis un fichier texte tabulé ou un fichier CSV (voir **annexe 5.5**).
 - (a) En Python, recommanderiez-vous l'usage d'une bibliothèque de ce langage à cette fin ? Pourquoi, et si oui, laquelle ?
 - (b) Dans le programme de première NSI, dans la partie traitement de données en tables, il est précisé : « Est utilisé un tableau doublement indexé ou un tableau de p-uplets qui partagent les mêmes descripteurs » (voir **annexe 5.5**). On précise que les p-uplets en question sont des p-uplets nommés et que, d'après le programme de première NSI, « en Python, les p-uplets nommés sont implémentés par des dictionnaires ». Donner un avantage et un inconvénient d'utiliser un tableau doublement indexé et un avantage et un inconvénient d'utiliser un tableau de p-uplets nommés qui partagent les mêmes descripteurs pour importer une table en Python.
 37. (a) En terminale NSI, un élève a proposé une organisation des données dans une seule table (voir figure 3). Cette table lui permet de recueillir, entre autres, la date, l'heure et la salle de diffusion de films d'un complexe cinématographique.
 Donner trois arguments pour lui faire comprendre les inconvénients d'une telle organisation.

Titre	Réalisateur	Année	Durée	Salle	Heure	Jour
De battre mon cœur s'est arrêté	Jacques Audiard	2005	107	4	14 :40	2021-06-12
Le Cercle des poètes disparus	Peter Weir	1989	128	5	20 :30	2021-06-12
...

FIGURE 3 – Proposition d'un élève

- (b) Proposer une autre structure pour cette base de données permettant de pallier ces inconvénients.
38. (a) Le modèle relationnel est au programme de terminale NSI (voir **annexe 5.6**).
 Proposer un plan de cours succinct sur ce sujet, tel que vous le présenteriez à des élèves de terminale NSI.
 (b) Proposer une activité permettant d'introduire la notion de contrainte d'intégrité.
39. Définir, tel que vous le feriez à une classe de terminale NSI, ce qu'est un SGBD et son rôle.

4 Traitement des données

40. En SNT, le programme donne l'exemple d'activité suivante : << Télécharger des données ouvertes (sous forme d'un fichier au format CSV avec les métadonnées associées), observer les différences de traitements possibles selon le logiciel choisi pour lire un fichier : programme Python, tableur, éditeur de textes ou encore outils spécialisés en ligne. >>
- Proposer deux sites sur lesquels on peut télécharger des données ouvertes.
 - Citer un avantage et un inconvénient pour chacune des quatre propositions (programme Python, tableur, éditeur de textes ou outils spécialisés en ligne) permettant de lire le fichier CSV.
 - Choisir deux logiciels puis proposer une activité permettant d'observer les différences de traitements possibles selon le logiciel choisi pour lire le fichier.

41. En classe de première NSI, vous décidez de faire travailler les élèves sur trois fichiers CSV, contenant respectivement des données sur les régions françaises, les départements français et les communes françaises.

Vous avez utilisé trois tableaux de dictionnaires qui partagent les mêmes clés pour importer trois tables depuis ces trois fichiers CSV.

Pour se donner une idée :

- un élément du tableau REGION est
`{"CodeRegion": 84, "NomRegion": "Auvergne-Rhône-Alpes", "CodePref": 69123, "CodeCR": "69123"}`
- un élément du tableau DEPARTEMENT est
`{"NumDep": 1, "CodeRegion": 84, "CodePref": 1053, "NomDep": "Ain"}`
- un élément du tableau COMMUNE est
`{"NumDep": 1, "NomVille": "Nantua", "CodePostal": "01460", "CodeInsee": "01269", "Population": 3713}`

Les valeurs des clés "CodePref" et "CodeCR" des éléments des tableaux REGION et DEPARTEMENT correspondent aux valeurs de la clé "CodeInsee" des éléments du tableau COMMUNE.

À partir des trois tableaux REGION, DEPARTEMENT et COMMUNE, vous proposez à vos élèves de première NSI l'exercice suivant :

Exercice

- 1) Écrire une fonction Python renvoyant un tableau contenant le nom des communes de plus de 10 000 habitants.
 - 2) Écrire une fonction Python renvoyant le nombre de départements ayant une commune dont le nom est "Saint-Michel".
 - 3) Écrire une fonction Python renvoyant un tableau de p -uplets contenant le nom des régions et le nom de leurs préfectures de régions lorsque la préfecture de région a plus de 250 000 habitants.
- Identifier pour chaque question de l'exercice une difficulté algorithmique à laquelle on peut s'attendre de la part d'un élève de première.
 - Pour chacune des difficultés identifiées dans la question précédente, proposer une activité de remédiation permettant de la surmonter.
 - Proposer deux corrections de cet exercice :
 - l'une utilisant des boucles ;
 - l'autre utilisant uniquement des tableaux définis par compréhension.
42. On dispose d'une base de données pour gérer les locations de VTT électriques d'un magasin de sport. Le schéma relationnel de cette base de données est :
- ```
VTT(num_vtt, marque, prix_achat, date_achat)
CLIENT(num_client, nom, prenom, adresse_mail)
LOCATION(id_loc, num_VTT#, num_client#, date_location, date_retour)
```
- date\_retour peut être égal à NULL si le VTT est en cours de location.
- Vous utilisez cette base de données pour construire un exercice permettant d'évaluer l'ensemble des notions sur le langage SQL au programme de terminale NSI (voir **annexe 5.6**).
- Dans la convention choisie par le concepteur du schéma relationnel, préciser à quoi correspondent les descripteurs soulignés et les descripteurs suivis d'un #.
  - Rédiger l'énoncé de cet exercice à partir de cette base de données. Les questions devront être progressives, non redondantes et respecter le programme de terminale NSI.  
 Pour chaque question, déterminer ce qu'elle évalue.
  - Proposer une correction de cet exercice.
  - Proposer une grille d'évaluation de cet exercice qui indique avec précision les éléments évalués.

## 5 Annexes

### 5.1 Programme de terminale NSI sur les processus

#### Architectures matérielles, systèmes d'exploitation et réseaux

La réduction de taille des éléments des circuits électroniques a conduit à l'avènement de systèmes sur puce (*SoCs* pour *Systems on Chips* en anglais) qui regroupent dans un seul circuit nombre de fonctions autrefois effectuées par des circuits séparés assemblés sur une carte électronique. Un tel système sur puce est conçu et mis au point de façon logicielle, ses briques électroniques sont accessibles par des API, comme pour les bibliothèques logicielles.

Toute machine est dotée d'un système d'exploitation qui a pour fonction de charger les programmes depuis la mémoire de masse et de lancer leur exécution en leur créant des processus, de gérer l'ensemble des ressources, de traiter les interruptions ainsi que les entrées-sorties et enfin d'assurer la sécurité globale du système.

Dans un réseau, les routeurs jouent un rôle essentiel dans la transmission des paquets sur Internet : les paquets sont routés individuellement par des algorithmes. Les pertes logiques peuvent être compensées par des protocoles reposant sur des accusés de réception ou des demandes de renvoi, comme TCP.

La protection des données sensibles échangées est au cœur d'Internet. Les notions de chiffrement et de déchiffrement de paquets pour les communications sécurisées sont explicitées.

| Contenus                                                               | Capacités attendues                                                                                                                                                    | Commentaires                                                                                                                                                                                                       |
|------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Composants intégrés d'un système sur puce.                             | Identifier les principaux composants sur un schéma de circuit et les avantages de leur intégration en termes de vitesse et de consommation.                            | Le circuit d'un téléphone peut être pris comme un exemple : microprocesseurs, mémoires locales, interfaces radio et filaires, gestion d'énergie, contrôleurs vidéo, accélérateur graphique, réseaux sur puce, etc. |
| Gestion des processus et des ressources par un système d'exploitation. | Décrire la création d'un processus, l'ordonnancement de plusieurs processus par le système.<br><br>Mettre en évidence le risque de l'interblocage ( <i>deadlock</i> ). | À l'aide d'outils standard, il s'agit d'observer les processus actifs ou en attente sur une machine.<br><br>Une présentation débranchée de l'interblocage peut être proposée.                                      |

## 5.2 Extraits du manuel de la commande top

```

1. %CPU -- CPU Usage
The task's share of the elapsed CPU time since the last screen update, expressed as a percentage of total CPU time.

In a true SMP environment, if a process is multi-threaded and top is not operating in Threads mode, amounts greater than 100% may be reported. You toggle Threads mode with the 'H' interactive command.

Also for multi-processor environments, if Irix mode is Off, top will operate in Solaris mode where a task's cpu usage will be divided by the total number of CPUs. You toggle Irix/Solaris modes with the 'I' interactive command.

Note: When running in forest view mode ('V') with children collapsed ('v'), this field will also include the CPU time of those unseen children. See topic 4c. TASK AREA Commands, CONTENT for more information regarding the 'V' and 'v' toggles.

2. %MEM -- Memory Usage (RES)
A task's currently resident share of available physical memory.

See `OVERVIEW, Linux Memory Types' for additional details.

```

```

19. PID -- Process Id
The task's unique process ID, which periodically wraps, though never restarting at zero. In kernel terms, it is a dispatchable entity defined by a task_struct.

This value may also be used as: a process group ID (see PGRP); a session ID for the session leader (see SID); a thread group ID for the thread group leader (see Tgid); and a TTY process group ID for the process group leader (see TPGID).

20. PPID -- Parent Process Id
The process ID (pid) of a task's parent.

21. PR -- Priority
The scheduling priority of the task. If you see 'rt' in this field, it means the task is running under real time scheduling priority.

Under linux, real time priority is somewhat misleading since traditionally the operating system was not preemptible. And while the 2.6 kernel can be made mostly preemptible, it is not always so.

```

```

29. S -- Process Status
The status of the task which can be one of:
D = uninterruptible sleep
I = idle
R = running
S = sleeping
T = stopped by job control signal
t = stopped by debugger during trace
Z = zombie

Tasks shown as running should be more properly thought of as ready to run -- their task_struct is simply represented on the Linux run-queue. Even without a true SMP machine, you may see numerous tasks in this state depending on top's delay interval and nice value.

```

```
38. TIME -- CPU Time
 Total CPU time the task has used since it started. When Cumulative mode is On, each process is listed with the cpu time that it and its dead children have used. You toggle Cumulative mode with 'S', which is both a command-line option and an interactive command. See the 'S' interactive command for additional information regarding this mode.

39. TIME+ -- CPU Time, hundredths
 The same as TIME, but reflecting more granularity through hundredths of a second.
```

```
6. COMMAND -- Command Name or Command Line
 Display the command line used to start a task or the name of the associated program. You toggle between command line and name with 'c', which is both a command-line option and an interactive command.

 When you've chosen to display command lines, processes without a command line (like kernel threads) will be shown with only the program name in brackets, as in this example:
 [kthreadd]

 This field may also be impacted by the forest view display mode. See the 'V' interactive command for additional information regarding that mode.

 Note: The COMMAND field, unlike most columns, is not fixed-width. When displayed, it plus any other variable width columns will be allocated all remaining screen width (up to the maximum 512 characters). Even so, such variable width fields could still suffer truncation. This is especially true for this field when command lines are being displayed (the 'c' interactive command.) See topic 5c. SCROLLING a Window for additional information on accessing any truncated data.
```

```
44. USER -- User Name
 The effective user name of the task's owner.
```

### 5.3 Extraits de l'aide Python pour les objets de type list

```
>>> help(len)
Help on built-in function len in module builtins:

len(obj, /)
 Return the number of items in a container.

>>> help(list)
Help on class list in module builtins:

class list(object)
| list(iterable=(), /)
|
| Built-in mutable sequence.
|
| If no argument is given, the constructor creates a new empty list.
| The argument must be an iterable if specified.
|
| Methods defined here:
|
| __len__(self, /)
| Return len(self).
|
| append(self, object, /)
| Append object to the end of the list.
|
| clear(self, /)
| Remove all items from list.
|
| copy(self, /)
| Return a shallow copy of the list.
|
| index(self, value, start=0, stop=9223372036854775807, /)
| Return first index of value.
|
| Raises ValueError if the value is not present.
|
| insert(self, index, object, /)
| Insert object before index.
|
| pop(self, index=-1, /)
| Remove and return item at index (default last).
|
| Raises IndexError if list is empty or index is out of range.
|
| remove(self, value, /)
| Remove first occurrence of value.
|
| Raises ValueError if the value is not present.
|
| reverse(self, /)
| Reverse *IN PLACE*.
|
| sort(self, /, *, key=None, reverse=False)
| Sort the list in ascending order and return None.
|
| The sort is in-place (i.e. the list itself is modified) and stable (i.e.
| the order of two equal elements is maintained).
|
| -----
| Data and other attributes defined here:
|
| __hash__ = None
```

## 5.4 Extraits de l'aide Python pour les objets de type dict

```
>>> help(min)
Help on built-in function min in module builtins:

min(...)
 min(iterable, *, default=obj, key=func) -> value
 min(arg1, arg2, *args, *, key=func) -> value

 With a single iterable argument, return its smallest item. The
 default keyword-only argument specifies an object to return if
 the provided iterable is empty.
 With two or more arguments, return the smallest argument.

>>> help(dict)
Help on class dict in module builtins:

class dict(object)
| dict() -> new empty dictionary
| dict(mapping) -> new dictionary initialized from a mapping object's
| (key, value) pairs
| dict(iterable) -> new dictionary initialized as if via:
| d = {}
| for k, v in iterable:
| d[k] = v
|
| Methods defined here:
|
| __contains__(self, key, /)
| True if the dictionary has the specified key, else False.
|
| __delitem__(self, key, /)
| Delete self[key].
|
| __init__(self, /, *, **kwargs)
| Initialize self. See help(type(self)) for accurate signature.
|
| __iter__(self, /)
| Implement iter(self).
|
| __len__(self, /)
| Return len(self).
|
| clear(...)
| D.clear() -> None. Remove all items from D.
|
| copy(...)
| D.copy() -> a shallow copy of D
|
| items(...)
| D.items() -> a set-like object providing a view on D's items
|
| keys(...)
| D.keys() -> a set-like object providing a view on D's keys
|
| pop(...)
| D.pop(k[,d]) -> v, remove specified key and return the corresponding value.
| If key is not found, d is returned if given, otherwise KeyError is raised
|
| -----
| Data and other attributes defined here:
|
| __hash__ = None
```

## 5.5 Programme de première NSI sur les données en table

### Traitement de données en tables

Les données organisées en table correspondent à une liste de p-uplets nommés qui partagent les mêmes descripteurs. La mobilisation de ce type de structure de données permet de préparer les élèves à aborder la notion de base de données qui ne sera présentée qu'en classe terminale. Il s'agit d'utiliser un tableau doublement indexé ou un tableau de p-uplets, dans un langage de programmation ordinaire et non dans un système de gestion de bases de données.

| Contenus                 | Capacités attendues                                                                            | Commentaires                                                                                             |
|--------------------------|------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| Indexation de tables     | Importer une table depuis un fichier texte tabulé ou un fichier CSV.                           | Est utilisé un tableau doublement indexé ou un tableau de p-uplets qui partagent les mêmes descripteurs. |
| Recherche dans une table | Rechercher les lignes d'une table vérifiant des critères exprimés en logique propositionnelle. | La recherche de doublons, les tests de cohérence d'une table sont présentés.                             |
| Tri d'une table          | Trier une table suivant une colonne.                                                           | Une fonction de tri intégrée au système ou à une bibliothèque peut être utilisée.                        |
| Fusion de tables         | Construire une nouvelle table en combinant les données de deux tables.                         | La notion de domaine de valeurs est mise en évidence.                                                    |

## 5.6 Programme de terminale NSI sur les bases de données

### Bases de données

Le développement des traitements informatiques nécessite la manipulation de données de plus en plus nombreuses. Leur organisation et leur stockage constituent un enjeu essentiel de performance.

Le recours aux bases de données relationnelles est aujourd'hui une solution très répandue. Ces bases de données permettent d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses utilisées simultanément par différents programmes ou différents utilisateurs. Cela est impossible avec les représentations tabulaires étudiées en classe de première.

Des systèmes de gestion de bases de données (SGBD) de très grande taille (de l'ordre du pétaoctet) sont au centre de nombreux dispositifs de collecte, de stockage et de production d'informations.

L'accès aux données d'une base de données relationnelle s'effectue grâce à des requêtes d'interrogation et de mise à jour qui peuvent par exemple être rédigées dans le langage SQL (*Structured Query Language*). Les traitements peuvent conjuguer le recours au langage SQL et à un langage de programmation.

Il convient de sensibiliser les élèves à un usage critique et responsable des données.

| Contenus                                                                                             | Capacités attendues                                                                                                                                                                                                                                 | Commentaires                                                                                                                                                                                                                                                                                         |
|------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Modèle relationnel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel. | Identifier les concepts définissant le modèle relationnel.                                                                                                                                                                                          | Ces concepts permettent d'exprimer les contraintes d'intégrité (domaine, relation et référence).                                                                                                                                                                                                     |
| Base de données relationnelle.                                                                       | Savoir distinguer la structure d'une base de données de son contenu.<br>Repérer des anomalies dans le schéma d'une base de données.                                                                                                                 | La structure est un ensemble de schémas relationnels qui respecte les contraintes du modèle relationnel.<br>Les anomalies peuvent être des redondances de données ou des anomalies d'insertion, de suppression, de mise à jour.<br>On priviliege la manipulation de données nombreuses et réalistes. |
| Système de gestion de bases de données relationnelles.                                               | Identifier les services rendus par un système de gestion de bases de données relationnelles : persistance des données, gestion des accès concurrents, efficacité de traitement des requêtes, sécurisation des accès.                                | Il s'agit de comprendre le rôle et les enjeux des différents services sans en détailler le fonctionnement.                                                                                                                                                                                           |
| Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données.                      | Identifier les composants d'une requête.<br>Construire des requêtes d'interrogation à l'aide des clauses du langage SQL : SELECT, FROM, WHERE, JOIN.<br>Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE. | On peut utiliser DISTINCT, ORDER BY ou les fonctions d'agrégation sans utiliser les clauses GROUP BY et HAVING.                                                                                                                                                                                      |

**EAE INF 1****SESSION 2023****AGREGATION  
CONCOURS EXTERNE****Section : INFORMATIQUE****COMPOSITION EN INFORMATIQUE**

Durée : 5 heures

*L'usage de tout ouvrage de référence, de tout dictionnaire et de tout matériel électronique (y compris la calculatrice) est rigoureusement interdit.*

*Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.*

*Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.*

**NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.  
Le fait de rendre une copie blanche est éliminatoire.**

Tournez la page S.V.P.

A

**INFORMATION AUX CANDIDATS**

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

Concours

**EAE**

Section/option

**6900A**

Epreuve

**101**

Matière

**9499**





**Dépendances.** Ce sujet contient trois parties indépendantes qui doivent être traitées toutes les trois. On veillera à bien indiquer sur la copie les changements de partie.

**Attendus.** Il est attendu des candidates et des candidats des réponses construites. Ils seront aussi évalués sur la précision, le soin et la clarté de la rédaction.

---

## Partie I. Provenance en bases de données

---

Cette partie vise à introduire une notion d'explication pour l'évaluation d'une requête sur une base de données. Un exemple de base de données est présenté, puis la notion de provenance booléenne pour les requêtes, et celle de formule booléenne pour la représentation de la provenance. Pour finir, on s'intéresse à comment calculer la provenance à partir d'une requête exprimée en algèbre relationnelle puis en SQL.

### 1 Exemple de base de données

#### 1.1 Présentation du schéma de la base de données

Voici une base de données d'une société organisant des croisières et son schéma.

```
Croisiere(cid : string, depart : string, arrivee : string, distance : integer)
Bateau(bid : string, type : string, autonomie : integer)
Navigable(bid : string, eid : string)
Employe(eid : string, nom : string, age : integer)
```

La relation *Employe* contient tous les employés de la compagnie dont les marins pilotant les bateaux. La relation *Navigable* indique pour chaque bateau quels sont les marins pouvant piloter ce bateau. Dans la relation *Navigable*, l'attribut *eid* est une clef étrangère provenant de la relation *Employe* et l'attribut *bid* est une clef étrangère provenant de la relation *Bateau*. Un marin est un employé qui peut piloter au moins un bateau. Les clefs primaires d'une relation sont les attributs qui sont soulignés. Les distances et les autonomies sont exprimées en noeuds.

**Question 1.1.** Écrire en SQL les commandes pour créer le schéma présenté dans ce paragraphe.

## 1.2 Requêtes

Certaines requêtes sont à écrire en SQL seulement et d'autres en SQL *et* algèbre relationnelle.

**Question 1.2.** Écrire la requête suivante en SQL et en algèbre relationnelle : donner les marins qui peuvent piloter des catamarans. Renvoyer les identifiants de ces marins.

**Question 1.3.** Écrire la requête suivante en SQL et en algèbre relationnelle : donner les identifiants des bateaux et les identifiants de croisières tels que les bateaux peuvent effectuer les croisières sans réapprovisionnement.

**Question 1.4.** Écrire la requête suivante en SQL et en algèbre relationnelle : donner l'identifiant des bateaux pouvant être pilotés par tous les marins d'un âge strictement supérieur à 40 ans.

**Question 1.5.** Écrire la requête suivante en SQL et en algèbre relationnelle : donner l'identifiant des marins qui peuvent piloter des bateaux pouvant parcourir strictement plus de 3000 nœuds mais qui ne peuvent pas piloter des catamarans.

**Question 1.6.** Écrire la requête suivante en SQL et en algèbre relationnelle : donner l'identifiant des employés les plus âgés.

**Question 1.7.** Écrire la requête suivante en SQL : donner le nombre de bateaux dans la base de données.

**Question 1.8.** Écrire la requête suivante en SQL : pour chaque bateau, renvoyer son identifiant et le nombre de marins qui peuvent le piloter.

**Question 1.9.** Écrire la requête suivante en SQL : donner l'âge moyen des marins.

## 2 Introduction à la provenance

Dans le cadre des bases de données, il est intéressant de pouvoir expliquer les réponses de l'évaluation des requêtes sur une base de données. Pour expliquer les réponses, il existe plusieurs méthodes dont la provenance booléenne.

### 2.1 Requêtes booléennes

**Définition 1** Une requête booléenne est une requête qui renvoie un enregistrement fixe, en général un 0-uplet, si la requête est vraie et qui ne renvoie rien sinon.

Pour information, il est possible d'écrire une requête qui renvoie un 0-uplet en ne mettant rien dans la partie SELECT. Ainsi la requête SELECT FROM R renvoie autant de fois le 0-uplet qu'il y a d'enregistrements dans la relation R.

**Question 1.10.** Soit  $Q$  une requête. Indiquer comment transformer  $Q$  en une requête booléenne  $Q'$  telle que  $Q'$  renvoie un enregistrement constant (par exemple 1) sur la base de donnée  $D$  si et seulement si  $Q$  renvoie au moins un enregistrement lorsqu'évaluée sur  $D$ . Expliquer cette transformation dans le cadre de SQL et de l'algèbre relationnelle. Appliquer votre méthode pour la requête de la question 1.2 à ses formulations en SQL et algèbre relationnelle.

**Définition 2** Soit  $Q$  une requête booléenne et soit  $D$  une base de données. La provenance booléenne de  $Q$  évaluée  $D$  et dénotée  $\text{Pr}(Q, D)$  est l'ensemble des sous-bases de données de  $D$  qui satisfont la requête  $Q$ .

**Exemple 1** Nous prenons la base de données avec le schéma suivant : il y a une relation  $R$  qui a deux attributs  $A$  et  $B$ . L'attribut  $A$  est de type VARCHAR et  $B$  de type INTEGER. Considérons la requête  $Q_1$  renvoyant le 0-uplet s'il existe un enregistrement ayant la valeur 3 pour l'attribut  $B$ . La base de donnée est égale à  $\{R(a, 4); R(b, 3); R(c, 3)\}$ . La provenance de l'évaluation de la requête sur cette base de donnée est égale à  $\{\{R(b, 3)\}; \{R(c, 3)\}; \{R(b, 3), R(c, 3)\}; \{R(b, 3), R(a, 4)\}; \{R(c, 3), R(a, 4)\}; \{R(b, 3), R(c, 3), R(a, 4)\}\}$ .

## 2.2 Provenance pour des requêtes non booléennes

Il est possible de définir la notion de provenance pour des requêtes non booléennes. C'est une généralisation de la provenance pour les requêtes booléennes. Soit  $f$  un enregistrement appartenant à la réponse de  $Q$  appliquée à la base de donnée  $D$ . La provenance booléenne de  $f$  est l'ensemble des sous-bases de données pour lesquelles  $f$  appartient aux réponses de  $D$ .

**Exemple 2** Nous prenons le schéma et la base de données de l'exemple 1. La requête  $Q_2$  renvoie l'attribut  $A$  des enregistrements ayant leur valeur de l'attribut  $B$  égale à 3. L'enregistrement  $b$  est une réponse de  $Q_2$  évaluée sur  $D$ . Sa provenance est égale à  $\{\{R(b, 3)\}; \{R(b, 3), R(c, 3)\}; \{R(b, 3), R(a, 4)\}; \{R(b, 3), R(c, 3), R(a, 4)\}\}$ . L'autre réponse de  $Q_2$  évaluée sur  $D$  est  $c$  et sa provenance est  $\{\{R(c, 3)\}; \{R(b, 3), R(c, 3)\}; \{R(c, 3), R(a, 4)\}; \{R(b, 3), R(c, 3), R(a, 4)\}\}$ .

## 2.3 Calcul de la provenance pour notre exercice

**Base de donnée pour l'évaluation** Nous présentons une très petite base de données correspondant au schéma précédent.

- Croisiere(c1,NY,L,5000) ; Croisiere(c2,L,V,5)
- Bateau(b1,Catamaran,2000) ; Bateau(b2,Yacht,10000) ; Bateau(b3,Kayak,10)
- Navigable(b1,e2) ; Navigable (b2,e2) ; Navigable(b3,e1)
- Employe(e1,Bob,20) ; Employe (e2,Alice, 42)

**Question 1.11.** Évaluer les requêtes des questions 1.2 et 1.9 de la sous-section 1.2 sur la base de données exemple et indiquer la provenance associée pour chaque réponse. Vous pouvez utiliser les valeurs de la clef primaire pour représenter un enregistrement.

### 3 Représentation compacte d'un ensemble : formules booléennes

Représenter l'ensemble des ensembles de sous-instances satisfaisant une requête booléenne peut être verbeux. Nous présentons une méthode pour représenter ces ensembles de façon plus compacte.

Soit  $E$  un ensemble fini d'éléments. Il existe une méthode pour représenter les ensembles d'ensembles d'éléments efficacement. Soit  $X_E$ , un ensemble de variables en bijection avec  $E$ . Une valuation de  $X_E$  est une fonction de  $X_E$  dans  $\{\top, \perp\}$ . Il existe une bijection entre les valuations et les ensembles de  $2^E$  définie comme suit : soit  $P$  un sous-ensemble appartenant à  $2^E$  et  $\nu_P$  la valuation associée à  $P$  telle que, pour que chaque variable  $x_e$  et l'élément associé  $e$ , on a  $e$  appartient à  $P$  si et seulement si  $\nu_P(x_e)$  est égale à  $\top$ . Nous pouvons remarquer qu'un ensemble de valuations peut être décrit par une formule booléenne. En décrivant l'ensemble des valuations, une formule booléenne peut décrire un ensemble d'ensembles d'éléments.

Dans cet exercice, les formules booléennes n'utilisent que les opérateurs  $\wedge, \vee, \neg$ .

**Question 1.12.** Soit  $E$  un ensemble  $\{e_1, e_2\}$ . On utilise les variables  $x_1$  et  $x_2$ .

1. Écrire une formule booléenne qui représente l'ensemble  $\{\{e_1, e_2\}\}$ .
2. Écrire une formule booléenne qui représente l'ensemble  $\{\{e_1\}, \{e_2\}\}$ .

Il est intéressant de noter que l'utilisation d'une formule booléenne peut être bien plus compacte qu'une représentation d'un ensemble d'ensembles. Plus précisément, il peut y avoir un gain exponentiel dans la représentation d'un ensemble d'ensembles avec une formule booléenne. La taille d'un ensemble est la somme des tailles de ces éléments. Pour un élément simple, c'est-à-dire dans  $E$ , la taille est égale à 1. Dans le cadre d'un ensemble d'ensembles d'éléments simples, la taille est égale à la somme des tailles de chaque ensemble contenu dans l'ensemble d'ensembles. Pour une formule booléenne, sa taille est égale à la taille du nombre de symboles utilisés pour la décrire : les symboles sont  $\wedge, \vee, \neg$ , les parenthèses ainsi que les variables (chaque variable compte pour un).

**Question 1.13.** Soit  $E = \{e_1, \dots, e_n\} \cup \{f_1, \dots, f_n\}$ , un ensemble d'éléments de taille  $2 \cdot n$ . Soit  $\phi$ , l'ensemble des ensembles tel que, quel que soit  $i$  entre 1 et  $n$ , on a  $e_i$  ou  $f_i$  qui appartient à l'ensemble mais pas les deux. Démontrer que  $\phi$  est un ensemble de taille exponentielle en  $n$ . Exhiber une formule booléenne de taille linéaire en  $n$  qui décrit  $\phi$ . Justifier formellement la taille de la formule ainsi que sa correction.

#### 3.1 Utilisation des formules booléennes pour représenter la provenance booléenne

En utilisant le enregistrement que nous pouvons décrire les ensembles d'ensembles par des formules booléennes, nous pouvons ainsi les utiliser dans le cadre de la provenance booléenne de bases de données.

**Question 1.14.** Exprimer une formule booléenne décrivant la provenance booléenne de la requête de la question 1.2 appliquée à la base de données présentée dans 2.3.

## 4 Calcul automatique de la provenance

Le but de cette partie est de proposer plusieurs méthodes pour calculer une formule booléenne représentant la provenance booléenne d'un enregistrement.

### 4.1 Calcul automatique de la provenance au travers de l'algèbre relationnelle

Le but de cette partie est d'utiliser l'algèbre relationnelle pour calculer la provenance. Pour cela, nous associons à chaque enregistrement  $f$  d'un résultat d'une sous-requête  $Q$  évaluée sur la base de données  $D$  une formule qui est notée  $\text{Pr}(f, Q, D)$ . Nous devons associer à chaque enregistrement une variable qui sera le  $n$ -uplet constitué des valeurs des clefs primaires des relations.

Tout d'abord, nous expliquons comment décrire la provenance d'une réponse à partir de la provenance du produit cartésien, de la projection et de la sélection.

- Pour le produit cartésien, la provenance d'un enregistrement dans la réponse est la conjonction de la provenance des deux enregistrements produisant ce tuple.
- La provenance d'un enregistrement provenant d'une sélection est la provenance de l'enregistrement sélectionné.
- La provenance d'un enregistrement  $f$  obtenu par la projection d'un ensemble d'attributs est la somme des provenances des enregistrements donnant  $f$  après projection.
- La provenance d'un enregistrement  $f$  d'une requête de la forme  $Q_1 - Q_2$  est égale à  $\text{Pr}(f, Q_1) \wedge \neg \text{Pr}(f, Q_2)$ .

**Exemple 3** Nous prenons l'exemple 2. La formule algébrique de la requête est  $\Pi_{R.A}(\sigma_{R.B=3}(D))$ . La provenance  $\text{Pr}(b, Q, D)$  est égale à  $R(b, 3)$ .

**Question 1.15.** Décrire avec les détails les calculs de provenance associés aux expressions algébriques de vos requêtes en algèbre relationnelle des questions 1.3 et 1.6.

### 4.2 Monotonie de la requête et monotonie de la provenance

Dans le cadre des requêtes, il est possible de définir une notion de requête croissante. Pour cela, nous définissons la notion d'ordre sur les instances. Une base de données  $D$  est inférieure à une autre base de données  $D'$  si  $D$  est incluse dans  $D'$ . Une requête  $Q$  est croissante si quels que soient  $D$  et  $D'$  telles que  $D \leq D'$ , on a  $Q(D) \leq Q(D')$ .

**Question 1.16.** Indiquer en justifiant quelles sont les requêtes qui sont croissantes et celles qui ne sont pas dans les requêtes de 1.2 à 1.9.

Nous définissons également une notion d'ordre partiel sur les valuations d'un ensemble de variables  $X$ . Soit  $\nu_1$  et  $\nu_2$  deux valuations d' $X$ . La valuation  $\nu_1$  est plus petite que la valuation

$\nu_2$  si pour chaque variable  $x$  dans  $X$ , si  $\nu_1(x)$  est égale à  $\top$  alors  $\nu_2(x)$  également. Une formule booléenne  $\varphi$  est croissante si, pour toute paire de valuations  $\nu_1$  et  $\nu_2$  telles que  $\nu_1$  est plus petite que  $\nu_2$ , si  $\nu_1$  satisfait  $\varphi$  alors  $\nu_2$  satisfait  $\varphi$ .

**Question 1.17.** Soit  $Q$  une requête booléenne croissante et  $D$  une base de données. Démontrer que la provenance de l'évaluation de  $Q$  sur  $D$  est croissante.

### 4.3 Calcul de la provenance en SQL

Dans cette partie, nous souhaitons réécrire les requêtes afin de renvoyer une chaîne de caractères représentant une formule booléenne codant la provenance des réponses de la requête évaluée sur la base de données. L'opérateur  $\wedge$  est représenté par le caractère \* et l'opérateur  $\vee$  est représenté par le caractère U. Par exemple la formule  $x \wedge y$  aura comme représentation la chaîne de caractères X \* Y. De même, la formule  $x \vee y$  aura comme représentation la chaîne de caractères X U Y.

Comme vu précédemment, nous devons associer à chaque enregistrement une variable qui sera construite à partir des valeurs des clefs primaires des relations. Dans la suite de cette sous-section, nous supposons que les attributs composant les clés primaires des relations sont de type VARCHAR. La variable est obtenue en concaténant les valeurs des différents attributs de la clef primaire. Cette valeur sera stockée dans un nouvel attribut dénoté VAR.

Pour cet exercice, nous utiliserons plusieurs fonctions qui sont admises.

- La fonction ADDOR prend deux chaînes de caractères CH1 et CH2 et renvoie (CH1 U CH2).
- La fonction ADDAND prend deux chaînes de caractères CH1 et CH2 et renvoie (CH1 \* CH2).
- La fonction AGGREG-OR prend un ensemble de chaînes de caractères et renvoie une chaîne de caractères représentant la disjonction des formules associées.
- La fonction AGGREG-AND prend un ensemble de chaînes de caractères et renvoie une chaîne de caractères représentant la conjonction des formules associées.

**Question 1.18.** Créer les commandes SQL qui permettent d'ajouter la colonne VAR à la relation *Croisiere* du schéma de la sous-section 1.2. Créer la requête de mises à jour pour remplir cette colonne dans la base de données.

**Exemple 4** Nous reprenons la requête  $Q_2$  égale à SELECT R.A FROM R WHERE R.B = 3. Cette requête est réécrite en la requête SELECT R.A AGGREG-OR(R.VAR) AS PROVENANCE FROM R WHERE R.B = 3 GROUP BY R.A. La chaîne de caractères de l'attribut provenance donne une représentation d'une formule booléenne de la provenance de chaque réponse.

**Question 1.19.** Réécrire les requêtes des questions 1.2, 1.6 et 1.9 qui renvoient, en plus des réponses, leur provenance avec le schéma modifié pour toutes les relations, i.e. toutes les relations ont un attribut supplémentaire VAR.

---

## Partie II. Ponts d'un graphe

---

### 1 Ponts et blocs dans un graphe non orienté

Dans un graphe non orienté, un **pont** est une arête dont la suppression fait croître le nombre de composantes connexes. Lorsque l'on retire tous les ponts d'un graphe, les composantes connexes restantes sont appelées **blocs**. Par exemple, le graphe représenté figure 1 possède 4 ponts, reliant les paires de sommets  $(1, 3)$ ,  $(3, 4)$ ,  $(6, 7)$  et  $(11, 15)$  et 6 blocs, comme illustré figure 2.

Dans ce problème, nous allons nous intéresser à un algorithme permettant de déterminer les blocs d'un graphe non orienté décrit *en ligne*, c'est-à-dire en ajoutant les arêtes une par une. On suppose que l'on connaît à l'avance le nombre  $n$  de sommets du graphe, et que ceux-ci sont numérotés de 0 à  $n - 1$ .

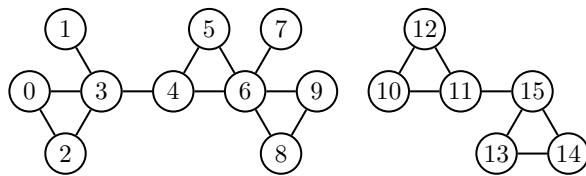


FIGURE 1 – Exemple de graphe.

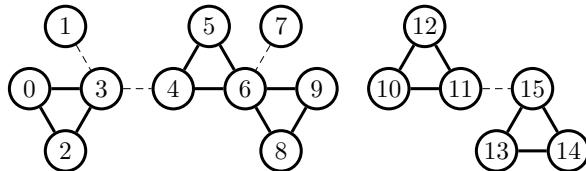


FIGURE 2 – Blocs en gras et ponts en hachuré.

La programmation s'effectuera en OCaml. Le candidat peut, s'il en éprouve le besoin, définir des fonctions auxiliaires pour mieux structurer son code. Il devra alors en préciser le rôle.

#### Quelques rappels sur le langage OCaml

Une liste est construite à partir de la liste vide `[]` et de la construction `x :: ℓ` qui renvoie une nouvelle liste dont la tête est l'élément `x` et dont la queue est la liste `ℓ`. L'appel de `List.rev ℓ` renvoie une nouvelle liste, formée des éléments de la liste `ℓ` en ordre inverse.

On peut créer des tableaux avec les fonctions `Array.make`, `Array.init` et `Array.of_list`.

- L'appel de `Array.make n x` crée un tableau de taille `n` dont toutes les cases contiennent la valeur `x`.
- L'appel de `Array.init n f` crée un tableau de taille `n` dans lequel la valeur de la case d'indice `i` est égale à `f(i)`.
- 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  $t$ , on accède à l'élément d'indice  $i$  avec  $t.(i)$  et on le modifie avec  $t.(i) \leftarrow v$ .

On mentionne enfin le type polymorphe `'a option` défini par :

```
type 'a option = None | Some of 'a
```

Un élément de la forme `Some x` correspond à la présence d'une valeur  $x$  de type `'a`, et un élément de la forme `None` correspond à une absence de valeur.

## 1.1 Représentation sous forme de forêt

Nous allons représenter le graphe sous forme d'une forêt à l'aide d'une structure de type *union-find* et que nous appellerons *buf* (pour *block union-find*). Cette structure consiste en trois tableaux de taille le nombre  $n$  de sommets du graphe :

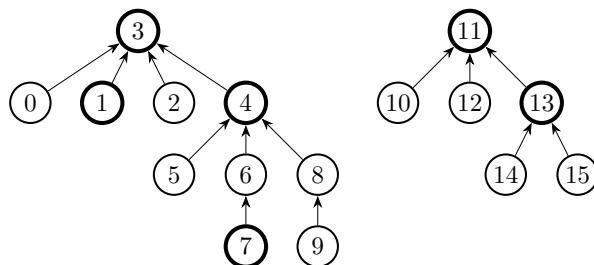
```
type buf = { parent : int array; repr : bool array; rang : int array; }
```

Chaque bloc a un unique *représentant* (défini de façon arbitraire), indiqué par le tableau `repr`. Chaque sommet a un *parent*, représenté par le tableau du même nom et qui a deux fonctions :

- si un sommet  $s$  est le représentant d'un bloc, notons-le  $b$ , alors soit  $\text{parent}.(s) = s$ , auquel cas  $s$  est racine d'un des arbres de la forêt, soit il indique un sommet d'un bloc  $b'$  tel qu'il existe un pont entre un sommet de  $b$  et un sommet de  $b'$ ;
- sinon, en suivant les parents successifs à partir d'un sommet  $s$ , on arrive au représentant de son bloc.

Enfin, le tableau `rang` indique, pour chaque représentant de bloc, une mesure de la taille de ce bloc. Cette valeur interviendra uniquement à la question 2.10 lors de l'implémentation de l'opération de fusion entre blocs.

Ainsi, tous les blocs sont représentés à la manière d'une structure de type *union-find*, à la différence que les différents blocs forment une forêt. On donne en figure 3 une représentation du graphe exemple de la figure 1 comme un élément de type `buf`, ainsi que son illustration graphique.



```
{ parent = [|3; 3; 3; 3; 3; 4; 4; 6; 4; 8; 11; 11; 11; 11; 13; 13|];
 repr = [|false; true; false; true; true; false; false; true; false;
 false; false; true; false; true; false; false|];
 rang = [|0; 0; 0; 1; 2; 0; 0; 0; 0; 0; 0; 0; 1; 0; 1; 0|] }
```

FIGURE 3 – Structure *buf* représentant le graphe exemple.

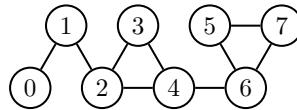


FIGURE 4 – Un autre graphe.

**Question 2.1.** Indiquer les blocs et les ponts du graphe représenté figure 4.

**Question 2.2.** Représenter graphiquement une structure *buf* correspondant au graphe de la figure 4.

**Question 2.3.** Écrire une fonction `init : int -> buf` qui, étant donné un entier *n*, renvoie une structure *buf* décrivant un graphe de *n* sommets sans aucune arête : chaque sommet donne lieu à un bloc de rang 0 dont il est le représentant.

**Question 2.4.** Écrire une fonction `find : buf -> int -> int` qui renvoie le représentant dans la structure *buf* du sommet passé en argument. On mettra en œuvre la *compression de chemin* : on modifie les parents de tous les sommets croisés sur le chemin entre le sommet de départ et son représentant, chaque sommet ayant pour nouveau parent leur représentant commun.

**Question 2.5.** Écrire une fonction `blocs : buf -> int list list` qui renvoie la liste des blocs correspondant à la structure passée en argument. L'ordre des blocs ainsi que l'ordre des sommets à l'intérieur d'un bloc n'est pas contraint. Ainsi, avec le graphe de la figure 1 (représenté par la forêt de la figure 3), on *peut* avoir :

```
[[15; 13; 14]; [12; 11; 10]; [7]; [9; 8; 6; 5; 4]; [3; 2; 0]; [1]]
```

On utilisera systématiquement la fonction `find` pour déterminer le représentant d'un sommet.

## 1.2 Ajout d'arêtes

Nous allons maintenant étudier l'effet de l'ajout d'une arête sur notre structure. Plusieurs cas sont possibles, suivant que les extrémités de l'arête ajoutée appartiennent à un même bloc, à des blocs distincts d'une même composante connexe, ou à des blocs de composantes connexes distinctes. Notons que si l'on ajoute une arête entre les sommets d'un même bloc, aucune modification n'est nécessaire. Pour traiter les deux autres cas, nous allons tout d'abord écrire quelques fonctions utilitaires avant d'implémenter la fonction d'ajout d'arête proprement dite.

Dans la suite, on appelle *chaîne de représentants* une suite finie non vide  $(s_0, \dots, s_p)$  de représentants telle que pour tout  $i \in \{0, \dots, p-1\}$ , le sommet  $s_i$  est le représentant du parent de  $s_{i+1}$ . En particulier, les représentants apparaissent de la gauche vers la droite par profondeur croissante. De façon naturelle, une telle chaîne sera représentée par une liste.

**Question 2.6.** Écrire une fonction `chaine_racine : buf -> int -> int list` qui, étant donné un sommet *s* du graphe, renvoie la chaîne des représentants reliant le représentant de *s* à la racine de l'arbre correspondant.

Ainsi, avec l'exemple précédent, la chaîne des représentants pour le sommet 5 est la liste [3; 4]. De même, pour le sommet 7 (qui est lui-même un représentant), on doit obtenir [3; 4; 7].

### 1.2.1 Arêtes entre des sommets de composantes connexes distinctes

Lorsque l'on ajoute une arête entre deux sommets appartenant à des composantes connexes distinctes, cette nouvelle arête est un pont. En terme de structure *buf*, si l'on note  $u$  et  $v$  les extrémités de l'arête ajoutée, on considère la chaîne des représentants allant de celui de l'un des sommets (supposons que l'on utilise  $u$ ) vers la racine correspondante et on « retourne » toutes les flèches de la chaîne, faisant du représentant de  $u$  la nouvelle racine de son arbre. On change alors le parent du représentant de  $u$  pour le faire pointer vers celui de  $v$ .

La figure 5 illustre le résultat de l'ajout d'une arête entre les sommets 5 et 14.

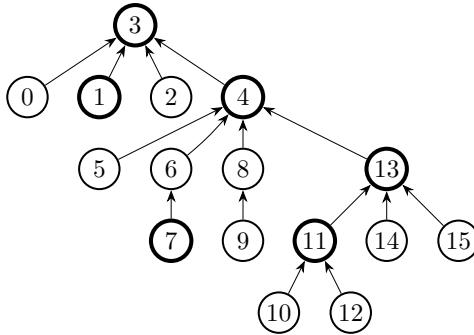


FIGURE 5 – Exemple d'ajout d'arête entre deux composantes connexes distinctes, entre les sommets 5 et 14.

**Question 2.7.** Représenter la forêt obtenue après avoir ajouté dans la forêt représentée en figure 3 une arête entre les sommets 7 et 12. On supposera que c'est le chemin issu du sommet 7 qui est retourné.

**Question 2.8.** Écrire une fonction `retourner_chaine : buf -> int list -> unit` qui implémente le retournement de chaîne présenté ci-dessus, sans effectuer le changement de parent de la nouvelle racine. On supposera que la liste passée en argument est une chaîne de représentants, de premier élément la racine d'un arbre.

### 1.2.2 Arêtes entre blocs distincts d'une même composante connexe

L'ajout d'une arête entre deux blocs d'une même composante connexe va entraîner la fusion de ces deux blocs ainsi que de tous les blocs compris entre les deux. Par exemple, comme illustré sur la figure 6, l'ajout d'une arête entre les sommets 1 et 15 à partir du graphe de la figure 5 conduit à la fusion des blocs de représentants 1, 3, 4 et 13.

**Question 2.9.** Représenter une forêt que l'on peut obtenir après avoir ajouté une arête entre les sommets 7 et 12 dans la forêt représentée en figure 5.

**Question 2.10.** Écrire une fonction `union : buf -> int -> int -> unit` qui effectue l'union des blocs dont les représentants sont passés en argument. Concrètement, le parent du représentant du bloc de plus petit rang deviendra le représentant du bloc de plus grand rang. En cas d'égalité des rangs, le choix se fera de façon arbitraire, et le rang du bloc résultant augmentera de 1. Le rang d'un sommet qui n'est pas un représentant ne joue aucun rôle. On ne se souciera pas, pour le moment, de la valeur du parent du bloc obtenu.

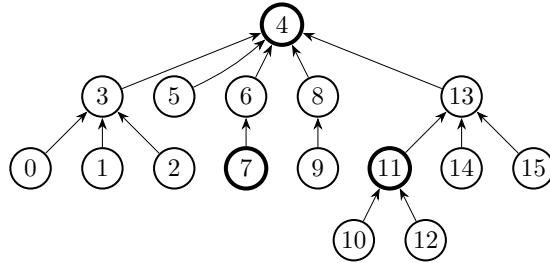


FIGURE 6 – Exemple d'ajout d'arête entre deux sommets d'une même composante connexe.

**Question 2.11.** Écrire une fonction `fusion_chaine : buf -> int list -> unit` qui effectue la fusion de tous les blocs dont les représentants sont dans la chaîne de représentants passée en argument. On portera une attention particulière, une fois la fusion de la chaîne effectuée, à la valeur du parent du représentant du bloc obtenu.

### 1.2.3 Fonction d'ajout

**Question 2.12.** Écrire une fonction `ajout : buf -> int -> int -> unit` qui implémente l'ajout d'un arête entre les deux sommets  $u$  et  $v$  passés en argument. En notant leurs représentants  $r_u$  et  $r_v$ , si ceux-ci sont différents, on distingue (à partir des chaînes de ces représentants vers leurs racines respectives) les cas où ceux-ci appartiennent à des composantes connexes distinctes ou à la même composante connexe. Dans le premier cas, on retourne la chaîne d'un des représentants jusqu'à la racine puis on lui attribue comme parent l'autre représentant. Dans le second, on fusionne les blocs des chaînes reliant  $r_u$  et  $r_v$  à leur plus proche ancêtre commun qui est l'élément commun le plus profond de leurs chaînes respectives vers leur racine.

## 1.3 Liste des ponts

On désire modifier la structure `buf` afin de pouvoir obtenir, en plus des blocs, la liste des ponts du graphe modélisé.

**Question 2.13.** Décrire une telle modification, en indiquant précisément les modifications à apporter à la structure `buf` et au code des diverses fonctions pour la manipuler, ainsi que l'implémentation d'une nouvelle fonction

```
ponts : buf -> (int * int) list.
```

Idéalement, les modifications des fonctions précédentes se traduiront par un surcoût de complexité constante et la fonction `ponts` sera de complexité linéaire en le nombre de sommets du graphe.

---

## Partie III. Architecture des ordinateurs

---

### 1 Algèbre de Boole et circuits combinatoires

**Question 3.1.** Dans cette question,  $x$  et  $y$  sont des valeurs dans  $\{0, 1\}$ . On considère l'opérateur booléen NOR défini par  $\text{NOR}(x, y) = \overline{x + y}$ . L'opérateur « ou » considéré ici n'est pas exclusif, soit  $1 + 1 = 1$ . La porte associée est représentée par la figure 1.

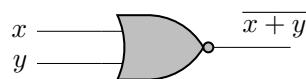


FIGURE 1 – Représentation d'une porte NOR.

- a) Donner la table de vérité de l'opérateur NOR.
- b) Exprimer les opérateurs de base  $\text{NOT}(x) = \overline{x}$ ,  $\text{AND}(x, y) = x.y$  et  $\text{OR}(x, y) = x + y$  avec uniquement des opérateurs binaires NOR. En déduire leur réalisation avec des portes NOR à deux entrées.
- c) Est-ce que l'opérateur NOR est associatif ? Justifier votre réponse.
- d) Calculer  $A(x, y, z) = x.y.z$  avec des opérateurs NOR binaires. Pour simplifier la formule, la négation des paramètres  $\overline{x}$ ,  $\overline{y}$  et  $\overline{z}$  est acceptée.

**Question 3.2.** Soient les fonctions booléennes  $f(x, y, z, t) = x + \overline{x}.\overline{y}.\overline{z}.\overline{t}$ ,  $g(x, y, z, t) = \overline{t}.(z + \overline{x}.y)$  et  $h(x, y, z) = x.\overline{y} + \overline{x}.y.\overline{z}$ .

- a) Démontrer que  $x + \overline{x}.y = x + y$ . En déduire une (petite) simplification de  $f$ .
- b) Exprimer les fonctions  $f$ ,  $g$  et  $h$  uniquement avec des opérateurs binaires NOR. On peut utiliser la fonction  $A$  et la négation des paramètres pour alléger l'expression des résultats.

### 2 Bascules RS et D

**Question 3.3.** On considère dans cette question la bascule RS réalisée avec des portes NOR représentée par la figure 2.

- a) Quelles sont les valeurs possibles des sorties  $Q_1$  et  $Q_2$  quand les entrées  $R$  et  $S$  sont stables à 0 ( $R = S = 0$ ) ? On appelle cet état  $E_1$ . Justifier votre réponse.
- b) Que se passe-t-il quand  $R$  passe à 1 à partir de l'état  $E_1$  ? On appelle cet état  $E_2$ . Justifier votre réponse.
- c) Que se passe-t-il quand à partir de l'état  $E_2$ ,  $S$  passe à 1 ? Justifier votre réponse.

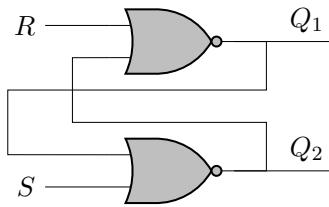


FIGURE 2 – Bascule RS réalisée avec des portes NOR.

- d) En déduire la table de vérité de la bascule RS. Ici, les nouvelles valeurs de sortie sont notées  $Q_1$  et  $Q_2$  et sont dépendantes de  $R$ ,  $S$  et des valeurs de sortie précédentes  $Q'_1$  et  $Q'_2$ ;
- e) À partir de la table de vérité de la bascule RS, montrer que  $Q_1 = \overline{R}(Q'_1 + S)$  et  $Q_2 = \overline{S}(Q'_2 + R)$ ;
- f) On suppose qu'au démarrage  $R = S = 0$  et que les entrées varient de sorte qu'à tout instant,  $R.S = 0$ . Quelle est la relation entre  $Q_1$  et  $Q_2$ ? Justifier votre réponse.

**Question 3.4.** On souhaite maintenant réaliser une bascule D à front descendant à partir de 3 bascules RS. Pour cela, on considère le schéma présenté par la figure 3. Le signal  $D$  est la donnée,  $H$  un signal d'horloge,  $Q$  et  $\overline{Q}$  les signaux de sortie.

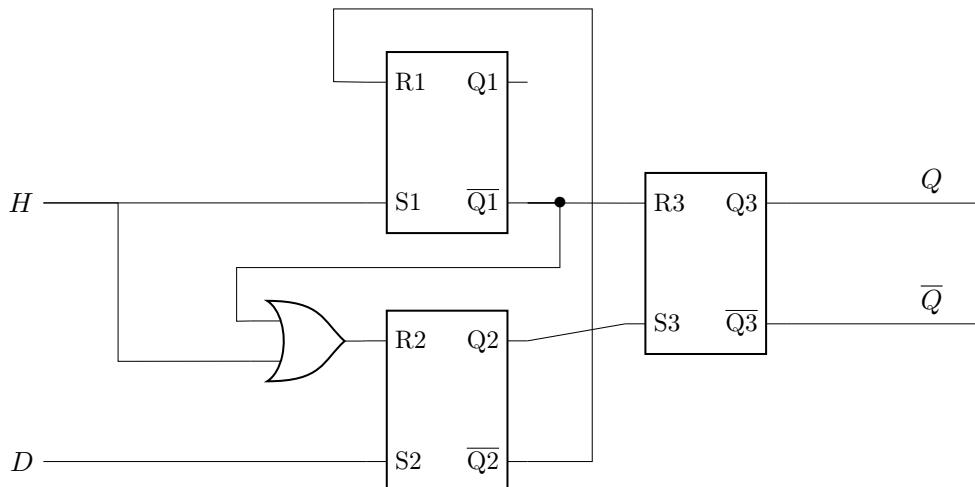


FIGURE 3 – Réalisation d'une bascule D à front descendant avec 3 bascules RS. La porte représentée est une porte OR.

- a) Donner les équations vérifiées par les sorties des 3 bascules  $\overline{Q_1}$ ,  $Q_2$ ,  $\overline{Q_2}$ ,  $Q_3$  et  $\overline{Q_3}$ ;
- b) Donner les équations vérifiées par les entrées des 3 bascules  $R_1$ ,  $S_1$ ,  $R_2$ ,  $S_2$ ,  $R_3$  et  $S_3$ .

**Question 3.5.** On suppose dans cette question que la valeur de  $D$  est stable quand le signal de l'horloge passe de 1 à 0 (au moment du front descendant). On souhaite démontrer les deux propriétés suivantes :

**P1** La valeur de la sortie  $\overline{Q}$  est celle de  $\overline{D}$  au moment du dernier front descendant.

**P2** La valeur de sortie  $\overline{Q}$  est stable entre deux fronts descendants.

Soit  $t$ , un instant qui coïncide avec un front descendant. On suppose dans cette question que  $D = 0$  à  $t$ .

- On considère dans un premier temps l'état du système à  $H = 1$  juste avant le front descendant à  $t$ . Montrer que les sorties  $Q_3$  et  $\overline{Q_3}$  sont stables (ne varient pas) ;
- On suppose maintenant qu'à  $t$ ,  $D = 0$ . Démontrer qu'au moment où  $H$  passe de 1 à 0,  $Q_3 = 0$  et  $\overline{Q_3} = 1$
- Démontrer que les sorties  $Q_3$  et  $\overline{Q_3}$  restent inchangées tant que  $H$  reste à 0 ;
- Que se passe-t-il quand  $H$  repasse à 1 ?

#### Question 3.6.

- Reprendre le raisonnement de la question précédente en le modifiant pour traiter le cas  $D = 1$  ;
- Conclure dans le cas général ( $D \in \{0, 1\}$ ) que les deux propriétés **P1** et **P2** sont vérifiées, en justifiant votre réponse.

### 3 Synthèse d'un automate de Moore

On considère dans cette partie un automate de Moore  $\mathcal{A}$  de 5 états représenté par la figure 4. Les noeuds sont étiquetés par l'état et la sortie associée.

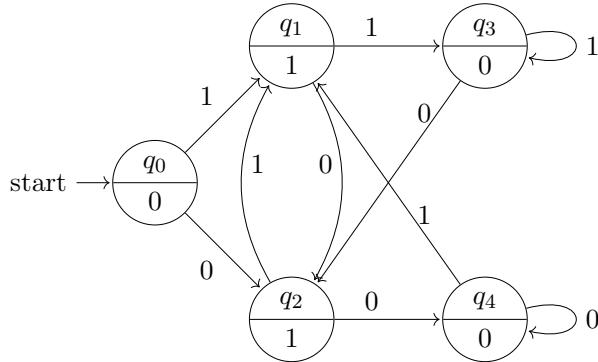


FIGURE 4 – Un automate de Moore  $\mathcal{A}$ .

#### Question 3.7.

- Donner la sortie de l'automate de Moore  $\mathcal{A}$  pour l'entrée  $e = 1000110111$  et la liste des états visités.
- Décrire sans justification à quoi correspond la sortie de cet automate.

**Question 3.8.** On souhaite synthétiser cet automate par une machine de Moore. Les états seront stockés à l'aide de bascule D. On suppose que les états sont codés par des entiers correspondant à leur numérotation, soit l'état  $q_0$  est représenté par l'entier 0, l'état  $q_1$  par l'entier 1, etc.

- a) Quel est le nombre  $\beta$  de bascules nécessaires pour mémoriser les états de l'automate ? Justifier votre réponse.

Par la suite, on note  $Q_0, \dots, Q_{\beta-1}$  les  $\beta$  valeurs booléennes qui permettent de coder les états de l'automate de sorte que l'entier  $i \in \{0, \dots, 5\}$  associé à l'état  $q_i$  vérifie  $i = \sum_{j=0}^{\beta-1} 2^j Q_j$  ;

- b) Donner le tableau de Karnaugh de la fonction qui associe selon les valeurs  $Q_j, j \in \{0, \dots, \beta-1\}$  la valeur de sortie  $S$  de l'automate. En déduire l'expression booléenne de  $S$  ;  
 c) Donner l'expression de  $S$  uniquement avec des opérateurs binaires NOR et les valeurs  $Q_j$  et  $\bar{Q}_j$  pour  $j \in \{0, \dots, \beta-1\}$ .

**Question 3.9.**

- a) Donner les tables de Karnaugh qui permettent de calculer l'état futur de la machine de Moore en fonction de l'état présent exprimé par les valeurs  $Q_j, j \in \{0, \dots, \beta-1\}$  et la valeur  $E$  de l'entrée. On note  $D_j, j \in \{0, \dots, \beta-1\}$  les valeurs de signaux booléens correspondant à l'état futur ;  
 b) En déduire les expressions booléennes des valeurs  $D_j, j \in \{0, \dots, \beta-1\}$  en fonction des valeurs  $Q_j, j \in \{0, \dots, \beta-1\}$  et de la valeur  $E$  de l'entrée ;  
 c) Donner les expressions de  $D_j, j \in \{0, \dots, \beta-1\}$  uniquement avec des opérateurs binaires NOR et les valeurs  $E, \bar{E}$  et  $Q_j$  et  $\bar{Q}_j$  pour  $j \in \{0, \dots, \beta-1\}$ .

\*   \*

\*



EAE INF 2

SESSION 2023

## AGREGATION CONCOURS EXTERNE

### Section : INFORMATIQUE

#### ÉTUDE D'UN PROBLÈME INFORMATIQUE

Durée : 6 heures

*L'usage de tout ouvrage de référence, de tout dictionnaire et de tout matériel électronique (y compris la calculatrice) est rigoureusement interdit.*

*Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.*

*Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.*

**NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.  
Le fait de rendre une copie blanche est éliminatoire.**

**Tournez la page S.V.P.**

(A)

**INFORMATION AUX CANDIDATS**

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

Concours

**EAE**

Section/option

**6200A**

Epreuve

**102**

Matière

**9423**





---

## Test d'égalité de langages rationnels

---

L'objet de ce problème est d'étudier comment on peut décider si deux expressions rationnelles  $\mathcal{E}$  et  $\mathcal{F}$  décrivent le même langage rationnel. Ce problème est intrinsèquement difficile : notamment si  $P \neq NP$  alors il n'y a pas d'algorithme polynomial pour le résoudre. Dans ce sujet, après des préliminaires algorithmiques, nous allons considérer que les expressions rationnelles sont encodées comme des arbres, et étudier un algorithme pour calculer un automate non-déterministe qui reconnaît le même langage. Si on déterminise les automates qui correspondent à  $\mathcal{E}$  et  $\mathcal{F}$ , on se ramène donc à tester si deux automates déterministes reconnaissent le même langage, ce qui fait l'objet de la dernière partie.

**Attendus.** Il est attendu des candidates et des candidats des réponses construites. Ils seront aussi évalués sur la précision, le soin et la clarté de la rédaction.

### Partie I. Préliminaires algorithmiques

Dans tout le sujet, le mot *complexité* désigne la complexité temporelle.

#### 1 Tableaux redimensionnables

Les *listes* PYTHON telles qu'elles sont implantées en CPYTHON (l'implantation de référence du langage PYTHON) sont en réalité ce que l'on appelle des *tableaux redimensionnables*, une structure de données souple et efficace, qui permet de stocker un nombre non borné de données, et en plus d'accéder au  $i$ -ème élément en temps constant.

Pour notre usage dans ce problème, un *tableau redimensionnable* est une structure de données qui supporte les opérations suivantes (sauf mention contraire, on n'utilisera pas les autres) :

- **initialisation** : crée un tableau redimensionnable vide ne contenant aucune donnée (en PYTHON, c'est l'instruction `t = []`) ;
- **longueur** : renvoie le nombre d'éléments stockés dans le tableau redimensionnable (en PYTHON, c'est l'instruction `len(t)`) ;
- **accès** : permet d'accéder en lecture et en écriture à l'élément en  $i$ -ème position dans un tableau redimensionnable ; les indices commencent à 0, donc il faut que  $i$  soit compris entre 0 et la longueur moins 1 (en PYTHON, c'est `t[i]`) .
- **ajout à la fin** : ajoute une donnée après la dernière donnée existante du tableau redimensionnable (en PYTHON, c'est l'instruction `t.append(x)`) ;

Pour notre analyse algorithmique, on considère les opérations suivantes sur les *tableaux* (pas les tableaux redimensionnables) et leurs complexités :

- allouer un tableau de taille  $n$  en temps  $\mathcal{O}(n)$  ;
- accéder au  $i$ -ème élément d'un tableau en temps  $\mathcal{O}(1)$  ;
- affecter une valeur au  $i$ -ème élément d'un tableau en temps  $\mathcal{O}(1)$ .

On souhaite planter des tableaux redimensionnables en utilisant des *enregistrements* (les **struct** du langage C) qui contiennent trois champs : **tableau** qui est un tableau, **capacite** qui est la taille allouée au tableau, **longueur** qui est un entier indiquant combien d'éléments sont stockés dans la structure : une **capacite** de 8 indique que **tableau** est alloué pour contenir 8 éléments, et si la **longueur** vaut 5, cela signifie que seuls les 5 premiers éléments, d'indices 0 à 4 inclus, sont dans la liste qui est représentée, voir Fig. 1.

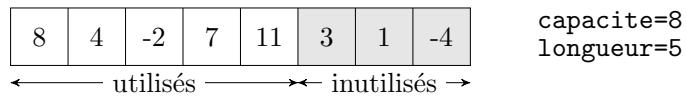


FIGURE 1 – Représentation de la liste abstraite  $[8, 4, -2, 7, 11]$  au moyen d'un tableau redimensionnable de capacité 8.

Les algorithmes pour les tableaux redimensionnables sont les suivants :

- **initialisation** : crée un enregistrement avec **capacite=1**, **longueur=0** et alloue une case pour **tableau**;
- **longueur** : renvoie **longueur**;
- **accès** : renvoie **tableau[i]** ;
- **ajout à la fin** : il y a deux cas selon que l'on ait la place pour un nouvel élément ou non
  - si **longueur < capacite**, on place le nouvel élément dans la case d'indice **longueur** du champs **tableau** et on incrémente **longueur** ;
  - si **longueur == capacite**, on alloue un nouveau **tableau** de **capacite** doublée, on recopie tous les éléments de l'ancien tableau dedans, puis on place le nouvel élément dans la case d'indice **longueur** du champs **tableau** et on incrémente **longueur**.

**Question 1.** Représenter, à chaque étape, l'enregistrement encodant un tableau redimensionnable que l'on initialise, puis auquel on ajoute successivement 3, -2, 4, 1 et -2 (6 étapes en tout avec l'initialisation).

**Question 2.** Parmi les opérations considérées, et dans le modèle pour les tableaux décrit plus haut, quelle est la seule opération à ne pas s'effectuer en temps constant ? Dans quels cas précisément, en fonction de la valeur de **longueur**, l'opération ne se fait pas en temps constant ? Quelle est sa complexité dans ces cas ?

**Question 3.** On initialise puis effectue  $n \geq 1$  insertions consécutives dans un tableau redimensionnable. On note  $I_n$  le nombre total d'écritures dans **tableau** qui ont été effectuées dans le processus (instructions du type **tableau[i] = ...**, en comptant celles effectuées lors des recopies). Soit  $k$  l'unique entier tel que  $2^{k-1} < n \leq 2^k$ . Montrer que  $I_1 = 1$  et que pour tout  $n \geq 2$  on a

$$I_n = n + \sum_{i=0}^{k-1} 2^i.$$

**Question 4.** En déduire qu'initialiser puis effectuer  $n \geq 1$  insertions consécutives dans un tableau redimensionnable se fait en temps  $\mathcal{O}(n)$ .

## 2 Tri lexicographique

Dans cette section on s'intéresse au problème de trier pour l'ordre lexicographique  $n$  listes contenant chacune  $k$  entiers de  $\{0, \dots, c-1\}$ , où  $n, k, c$  sont des entiers strictement positifs. Quand on est dans ce cadre, on peut trier plus efficacement qu'en utilisant un tri comme le tri fusion. Cet algorithme sera utilisé dans la section 6.

On considère que les listes sont implantées au moyen de tableaux redimensionnables, et que l'on peut ainsi ré-utiliser les complexités de la section précédente.

Par exemple, l'entrée de l'algorithme pour  $n = 5$ ,  $k = 3$  et  $c = 7$  pourrait être, en notation PYTHON : `L = [[2,5,1], [6,0,0], [1,2,3], [1,0,3], [4,2,1]]`.

**Question 5.** Quel est le résultat attendu après avoir trié `L` ci-dessus selon l'ordre lexicographique ?

**Question 6.** Écrire une fonction PYTHON `plus_petit_k_uple` qui prend en argument deux listes d'entiers `l1` et `l2` supposées être de même taille et renvoie `True` si et seulement si `l1` arrive avant `l2` dans l'ordre lexicographique, et `False` sinon. Cette fonction doit posséder une complexité linéaire en la taille commune des listes fournies en argument (sans avoir à la justifier).

**Question 7.** Si on utilise la fonction `plus_petit_k_uple` pour comparer deux listes d'entiers de taille  $k$ , quelle serait la complexité en fonction de  $n$  et de  $k$  d'utiliser le Tri Fusion (MERGESORT) pour résoudre le problème de trier selon l'ordre lexicographique une liste de  $n$  listes d'entiers de taille  $k$  ?

La solution ci-dessus n'utilise pas le fait que les valeurs sont toutes comprises entre 0 et  $c-1$ . On va pouvoir proposer une solution algorithmiquement plus performante en exploitant cette spécificité.

On considère l'algorithme `TRIPARCLE(L,j,c)`, où  $L$  est une liste de listes de notre problème et  $j \in \{0, \dots, k-1\}$ . Cet algorithme renvoie une liste  $K$  contenant une permutation de  $L$  où les éléments sont triés selon leur indice  $j$  : pour tous  $i, i'$  tels que  $0 \leq i < i' < n$ ,  $K[i][j] \leq K[i'][j]$ . On utilise pour cela le procédé suivant :

- Créer un tableau redimensionnable  $T$  contenant  $c$  tableaux redimensionnables initialement vides.
- Pour chaque élément (liste de longueur  $k$ )  $\ell$  de  $L$  dans l'ordre, ajouter  $\ell$  en fin du tableau redimensionnable  $T[\ell[j]]$ .
- Renvoyer la concaténation des tableaux redimensionnables  $T[0], T[1], \dots, T[c-1]$ .

**Question 8.** Que renvoie l'algorithme appliqué à `L = [[2,5,1], [6,0,0], [1,2,3], [1,0,3], [4,2,1]]` avec  $j = 1$  ?

**Question 9.** Écrire l'algorithme `TRIPARCLE(L,j,c)` en PYTHON.

**Question 10.** Montrer que l'algorithme  $\text{TRIPARCLE}(L, j, c)$  a pour complexité  $\mathcal{O}(c + n)$ .

**Question 11.** Montrer que l'algorithme  $\text{TRIPARCLE}(L, j, c)$  est stable : si on a deux indices  $i, i'$  avec  $0 \leq i < i' < n$  tels que  $L[i] \neq L[i']$  et que  $L[i][j] = L[i'][j]$ , alors dans le résultat  $K$ , l'élément  $L[i]$  est avant  $L[i']$ .

L'algorithme  $\text{TRILEXICOGRAPHIQUE}(L, c)$  consiste à trier successivement  $L$  avec  $\text{TRIPARCLE}(L, j, c)$  en faisant décroître  $j$  de  $k - 1$  à 0.

**Question 12.** Appliquer l'algorithme  $\text{TRILEXICOGRAPHIQUE}(L, 7)$  à  $L = [[2, 5, 1], [6, 0, 0], [1, 2, 3], [1, 0, 3], [4, 2, 1]]$ , en indiquant la liste obtenue à chaque itération de la boucle sur  $j$ .

**Question 13.** Montrer que l'algorithme  $\text{TRILEXICOGRAPHIQUE}(L)$  est une solution correcte au problème initial : il trie les éléments de  $L$  pour l'ordre lexicographique.

On a ainsi un algorithme de complexité  $\mathcal{O}(k(c + n))$  pour résoudre le problème de trier pour l'ordre lexicographique  $n$  listes de  $k$  entiers compris entre 0 et  $c - 1$ .

**Important :** Pour toute la suite, afin de simplifier les explications, on considérera que les listes de PYTHON ont les opérations **initialisation**, **longueur**, **accès** et **ajout à la fin** en temps  $\mathcal{O}(1)$ , sans spécifier à chaque fois qu'il s'agit en fait de tableaux redimensionnables et que l'ajout à la fin se fait en réalité en temps  $\mathcal{O}(1)$  amorti.

## Partie II. Des expressions rationnelles aux automates

Dans cette partie, nous explorons l'algorithme de Glushkov dont le but est de trouver un automate (a priori non déterministe) qui reconnaît le même langage qu'une expression rationnelle donnée.

### 3 Représenter des expressions rationnelles non vides

Soit un alphabet fini  $\Sigma$  et  $\varepsilon$  le mot vide sur cet alphabet.

Les *expressions rationnelles non vides* sur  $\Sigma$  sont définies inductivement par :

- l'ensemble d'assertions  $\{\varepsilon, a \mid a \in \Sigma\}$ ,
- l'ensemble de règles d'inférence {somme :  $(E_1, E_2) \mapsto (E_1) + (E_2)$ , étoile :  $E \mapsto (E)^*$ , produit :  $(E_1, E_2) \mapsto (E_1) \cdot (E_2)$ }.

On s'autorisera à ne pas mettre systématiquement des parenthèses, en prenant la convention de préséance suivante : l'étoile est prioritaire sur le produit qui est prioritaire sur la somme.

On s'autorisera également à ne pas écrire systématiquement le symbole  $\cdot$  pour le produit. Ainsi, l'expression  $((a) \cdot (b))^* + (((a) \cdot (a) + \varepsilon))$  pourra s'écrire  $(ab)^* + aa + \varepsilon$ .

**Par la suite on utilisera le terme *expression rationnelle* pour désigner une expression rationnelle non vide.**

À chaque expression rationnelle  $E$ , on peut associer sa longueur  $|E|$  et son langage  $\mathcal{L}(E)$  sur  $\Sigma$ , définis de manière inductive :

| expression             | longueur            | langage                                                      |
|------------------------|---------------------|--------------------------------------------------------------|
| $\varepsilon$          | 1                   | $\{\varepsilon\}$                                            |
| $a$ ( $a \in \Sigma$ ) | 1                   | $\{a\}$                                                      |
| $(E_1) + (E_2)$        | $1 +  E_1  +  E_2 $ | $\mathcal{L}(E_1) \cup \mathcal{L}(E_2)$                     |
| $(E_1) \cdot (E_2)$    | $1 +  E_1  +  E_2 $ | $\mathcal{L}(E_1) \cdot \mathcal{L}(E_2)$                    |
| $(E)^*$                | $1 +  E $           | $\mathcal{L}(E)^* = \bigcup_{i=0}^{\infty} \mathcal{L}(E)^i$ |

avec par convention  $L^0 = \{\varepsilon\}$  et  $L^i = L \cdot L^{i-1}$  pour tout langage  $L$  et tout entier  $i \geq 1$ . Sur les langages, on prend la règle de préséance suivante : l'étoile est prioritaire sur le produit qui est prioritaire sur l'union.

On peut représenter de telles expressions sous forme d'arbres syntaxiques (représentation naturelle à partir de la définition inductive) et construire à partir d'un tel arbre une liste de listes imbriquées pour les calculs en PYTHON :

| expression             | arbre                                                                      | PYTHON          |
|------------------------|----------------------------------------------------------------------------|-----------------|
| $\varepsilon$          | $\varepsilon$                                                              | [ '' ]          |
| $a$ ( $a \in \Sigma$ ) | $a$                                                                        | [ 'a' ]         |
| $(E_1) + (E_2)$        | $  \begin{array}{c}  + \\  / \quad \  \\ E_1 \quad E_2  \end{array}  $     | [ '+', E1, E2 ] |
| $(E_1) \cdot (E_2)$    | $  \begin{array}{c}  \cdot \\  / \quad \  \\ E_1 \quad E_2  \end{array}  $ | [ '.', E1, E2 ] |
| $(E)^*$                | $  \begin{array}{c}  * \\     \\ E  \end{array}  $                         | [ '*', E ]      |

Dans la suite, quand on parlera d'une expression rationnelle en PYTHON, elle sera nécessairement sous forme de liste de listes imbriquées, comme défini ci-dessus, sans qu'on ait besoin de le spécifier.

**Question 14.** Donner l'arbre syntaxique et la représentation PYTHON de l'expression rationnelle  $(a + b)^*a + (ab + \varepsilon)(c + \varepsilon)$ .

**Question 15.** Écrire une fonction PYTHON `expression` qui prend en argument une expression rationnelle et renvoie une représentation sous forme de chaîne de caractères de cette expression.

**Exemple.**

`expression(['+', ['*', ['.', ['a'], ['b']]], ['+', ['.', ['a'], ['a']], []])`  
s'évalue en '`((a).(b))*+((a).(a))+(_)`', où le caractère '`_`' représente le mot vide.

**Question 16.** Écrire une fonction PYTHON `contient_mot_vide` qui prend en argument une expression rationnelle et renvoie `True` si le mot vide appartient au langage associé à l'expression rationnelle, `False` sinon. Votre fonction doit avoir une complexité linéaire en la longueur de l'expression rationnelle (sans avoir à le justifier).

## 4 Automate de Glushkov

L'automate de Glushkov d'une expression rationnelle  $E$  est un automate particulier qui reconnaît le langage  $\mathcal{L}(E)$  associé à cette expression. Le but de cette section est de construire cet automate de manière efficace.

### 4.1 Rappels et propriétés

La première étape pour construire l'automate de Glushkov associé à une expression rationnelle  $E$  sur l'alphabet  $\Sigma$  est de *linéariser* cette expression, c'est-à-dire construire une nouvelle expression  $E_\ell$  sur un nouvel alphabet  $\Sigma_\ell$ , dont on déduit facilement  $E$  et dans laquelle chaque lettre possède au plus une occurrence. Une telle expression est qualifiée de *locale*.

On note  $\#E$  le nombre de lettres apparaissant dans l'expression  $E$  et on construit une nouvelle expression  $E_\ell$  sur l'alphabet  $\{a_i \mid a \in \Sigma, 1 \leq i \leq \#E\}$ , en adjoignant à chaque lettre qui apparaît dans l'expression  $E$  sa position : on numérote les lettres de  $E$  de gauche à droite en partant de 1, et on ajoute ce numéro en indice à chaque lettre. L'ensemble des lettres effectivement utilisées dans  $E_\ell$  est appelé *alphabet* de  $E_\ell$  et noté  $\Sigma_\ell$ .

Ainsi, à partir de l'expression  $E = (ab)^* + aa + \varepsilon$  sur l'alphabet  $\Sigma = \{a, b\}$ , on obtient l'expression linéarisée  $E_\ell = (a_1 b_2)^* + a_3 a_4 + \varepsilon$  sur l'alphabet  $\Sigma_\ell = \{a_1, b_2, a_3, a_4\}$ .

**Question 17.** Écrire une fonction `linearisation` qui prend en argument une expression rationnelle et renvoie l'expression obtenue en la linéarisant. Cette fonction doit avoir une complexité linéaire en la longueur de l'expression de départ (sans avoir à le justifier).

**Exemple.**

`linearisation(['+', ['*', ['.', ['a'], ['b']]], ['+', ['.', ['a'], ['a']], []])`  
s'évalue en '`[+] [ * [.] [a] [b] ] , [+] [.] [a] [a] ]`'.

Pour construire l'automate de Glushkov de  $E$  à partir de  $E_\ell$ , on définit les sous-ensembles et valeurs suivants :

- $\text{First}(E)$  est l'ensemble des lettres de  $\Sigma_\ell$  qui apparaissent au début d'un mot de  $\mathcal{L}(E_\ell)$  :

$$\text{First}(E) = \{x \in \Sigma_\ell \mid \exists u \in \Sigma_\ell^*, xu \in \mathcal{L}(E_\ell)\},$$

- $\text{Last}(E)$  est l'ensemble des lettres de  $\Sigma_\ell$  qui apparaissent à la fin d'un mot de  $\mathcal{L}(E_\ell)$  :

$$\text{Last}(E) = \{x \in \Sigma_\ell \mid \exists u \in \Sigma_\ell^*, ux \in \mathcal{L}(E_\ell)\},$$

- $\text{Null}(E)$  = vrai si  $\mathcal{L}(E_\ell)$  contient le mot vide, et faux sinon ;
- $\text{Follow}(E)$  est l'ensemble des facteurs de longueur 2 des mots de  $\mathcal{L}(E_\ell)$  :

$$\text{Follow}(E) = \{xy \in \Sigma_\ell^2 \mid \exists u, v \in \Sigma_\ell^*, ux y v \in \mathcal{L}(E_\ell)\}.$$

**Question 18.** Donner les valeurs de  $\text{First}(E)$ ,  $\text{Last}(E)$ ,  $\text{Null}(E)$  et  $\text{Follow}(E)$  pour l'expression  $E = (a + b)^*a + (ab + \varepsilon)(c + \varepsilon)$ .

On associe à l'expression  $E$  l'automate  $\mathcal{A}_\ell = (\Sigma_\ell, \Sigma_\ell \cup \{i\}, \delta_\ell, i, F)$  sur l'alphabet  $\Sigma_\ell$ , ayant pour ensemble d'états  $\Sigma_\ell \cup \{i\}$  (où  $i \notin \Sigma_\ell$ ) et pour état initial  $i$  défini par :

- pour toute lettre  $x \in \text{First}(E)$ , on a la transition  $\delta_\ell(i, x) = \{x\}$  de l'état  $i$  par la lettre  $x$ ,
- pour toute lettre  $x \in \Sigma_\ell$  et toute lettre  $y \in \Sigma_\ell$  telles que  $xy \in \text{Follow}(E)$ , on a la transition  $\delta_\ell(x, y) = \{y\}$  de l'état  $x$  par la lettre  $y$ ,
- l'ensemble des états finaux  $F = \text{Last}(E) \cup \{i\}$  si  $\text{Null}(E)$  est vrai, et  $F = \text{Last}(E)$  sinon.

**Question 19.** Montrer que l'automate  $\mathcal{A}_\ell$  reconnaît le langage décrit par l'expression  $E_\ell$ .

On déduit de  $\mathcal{A}_\ell$  un automate  $\mathcal{A}$  en supprimant les indices sur les transitions. Ainsi, la transition  $a_j \xrightarrow{a_k} a_k$  ( $a \in \Sigma, j, k \in \{1, \dots, \#E\}$ ) de  $\mathcal{A}_\ell$  devient la transition  $a_j \xrightarrow{a} a_k$  dans  $\mathcal{A}$ .

Cet automate est appelé *automate de Glushkov* associé à  $E$ .

**Question 20.** Justifier que dans l'automate  $\mathcal{A}$ , toutes les transitions qui arrivent dans un état portent la même étiquette.

**Question 21.** Montrer que l'automate  $\mathcal{A}$  reconnaît le langage décrit par l'expression  $E$ .

**Question 22.** Donner en la justifiant une comparaison entre le nombre d'états de l'automate  $\mathcal{A}$  ainsi obtenu et la taille de l'expression rationnelle  $E$ .

## 4.2 Construction de l'automate

Dans cette section, on s'intéresse à l'implantation de l'automate de Glushkov d'une expression rationnelle.

Pour toute expression rationnelle  $E$ , les ensembles  $\text{First}(E)$ ,  $\text{Last}(E)$  et  $\text{Follow}(E)$  sont finis par construction. On suppose qu'on dispose d'une classe `Set` qui nous servira à représenter et manipuler des ensembles finis, dont voici un extrait de la documentation et qu'on pourra supposer importée (on ne vous demande pas d'écrire le code de cette classe) :

```

class Set(builtins.object)
 | représentation d'un ensemble fini
 |
 | Methods defined here:
 |
 | disjoint_extend(self, other)
 | modifie l'objet courant en l'étendant avec le contenu de
 | l'objet passé en paramètre et renvoie l'objet courant ainsi modifié,
 | les deux objets doivent représenter des ensembles disjoints,
 | le résultat représente l'union des deux ensembles;
 | cette méthode s'exécute en temps linéaire en la taille de
 | l'ensemble fourni en argument.
 |
 | empty()
 | renvoie un ensemble vide;
 | cette méthode s'exécute en temps constant.
 |
 | extend(self, other)
 | modifie l'objet courant en l'étendant avec le contenu de
 | l'objet passé en paramètre et renvoie l'objet courant ainsi modifié,
 | le résultat représente l'union des deux ensembles (sans doublons);
 | cette méthode s'exécute en temps proportionnel au produit
 | de la taille de l'ensemble sur lequel la méthode est
 | invoquée et de la taille de l'ensemble fourni en argument.
 |
 | product(self, other)
 | renvoie le produit cartésien de deux langages,
 | le résultat est un ensemble de couples;
 | cette méthode s'exécute en temps proportionnel au produit
 | de la taille de l'ensemble sur lequel la méthode est
 | invoquée et de la taille de l'ensemble fourni en argument.
 |
 | singleton(e)
 | renvoie un singleton contenant l'élément fourni en argument;
 | cette méthode s'exécute en temps constant.

```

Une façon d'implanter une telle classe `Set` est d'utiliser un tableau redimensionnable. On obtient alors la complexité linéaire de la méthode `disjoint_extend` en suivant le même principe de redimensionnement que celui expliqué en section 1.

**Remarque :** On n'utilise pas le type `set` de PYTHON car on souhaite pouvoir exploiter la différence de complexité entre une union disjointe et une union qui a priori ne l'est pas, et on proposera une optimisation de la classe ci-dessus plus tard dans le sujet.

On donne le code suivant pour calculer First, Last, Null et Follow d'une expression rationnelle locale non vide :

```

def glushkov(exp):
 """renvoie first, last, null et follow de l'expression exp
 exp doit être locale et non vide
 first, last et follow sont des instances de Set, null est un booléen"""
 assert(len(exp) > 0 and len(exp) <= 3)
 if len(exp) == 1:
 if exp[0] == '':
 first, last, null, follow = Set.empty(), Set.empty(), True, Set.empty()
 else:
 first, last = Set.singleton(exp[0]), Set.singleton(exp[0])
 null, follow = False, Set.empty()
 elif len(exp) == 3:
 assert(exp[0] in ['+', '.'])
 first1, last1, null1, follow1 = glushkov(exp[1])
 first2, last2, null2, follow2 = glushkov(exp[2])
 if exp[0] == '+':
 first = first1.disjoint_extend(first2)
 last = last1.disjoint_extend(last2)
 null = null1 or null2
 follow = follow1.disjoint_extend(follow2)
 else:
 first = first1 if not null1 else first1.disjoint_extend(first2)
 last = last2 if not null2 else last2.disjoint_extend(last1)
 null = null1 and null2
 follow = follow1.disjoint_extend(follow2).disjoint_extend(last1.product(first2))
 else:
 assert(exp[0] == '*')
 first1, last1, null1, follow1 = glushkov(exp[1])
 first, last, null = first1, last1, True
 follow = follow1.extend(last1.product(first1))
 return first, last, null, follow

```

**Question 23.** Dérouler la fonction `glushkov` sur l'expression `['+', ['*', ['.', ['a1'], ['b2']]], ['.', ['a3'], ['a4']]]`.

**Question 24.** La fonction `glushkov` réalise un parcours de l'arbre syntaxique de l'expression. Comment est appelé ce parcours ?

**Question 25.** Justifier que les utilisations de `disjoint_extend` sont adéquates.

**Question 26.** Donner un exemple pour expliquer pourquoi la dernière extension du code (ligne 29) n'est pas une extension disjointe.

**Question 27.** Montrer la correction totale de la fonction `glushkov`.

On s'intéresse maintenant à la complexité temporelle de la fonction `glushkov`, en séparant l'analyse en fonction de ce qu'on calcule parmi First, Last, Null et Follow. On note  $n$  la longueur de l'expression fournie en argument.

**Question 28.** Montrer que la complexité temporelle du calcul de Null en suivant l'algorithme de la fonction `glushkov` est en  $\Theta(n)$ .

On admet que la complexité des calculs de First et Last en suivant l'algorithme de la fonction `glushkov` est linéaire en la longueur de l'expression de départ.

**Question 29.** Montrer que la complexité dans le pire des cas du calcul de Follow en suivant l'algorithme de la fonction `glushkov` est en  $\Omega(n^5)$  où  $n$  est la longueur de l'expression fournie en argument. On pourra par exemple considérer la famille d'expressions locales définie récursivement par  $F_1 = a_1^*$  et  $F_n = (F_{n-1} + a_n)^*$ . On rappelle que les complexités des méthodes de la classe `Set` sont données dans la documentation de cette même classe.

### 4.3 Amélioration de la complexité

L'extension non disjointe dans la fonction `glushkov` joue un rôle non négligeable dans sa complexité. Dans cette partie, nous allons voir comment nous ramener à une extension disjointe.

Soit une expression locale  $E$  de la forme  $F^*$ . Le calcul de `follow` repose sur la relation

$$\text{Follow}(E) = \text{Follow}(F^*) = \text{Follow}(F) \cup (\text{Last}(F) \times \text{First}(F)) ,$$

où l'union n'est pas nécessairement disjointe, comme on l'a vu à la question 26.

Or, on peut exprimer cette relation avec l'union disjointe suivante (en notant  $\sqcup$  l'opérateur d'union disjointe) :

$$\text{Follow}(E) = \text{Follow}(F^*) = \text{Follow}(F) \sqcup ((\text{Last}(F) \times \text{First}(F)) \setminus \text{Follow}(F)) .$$

Notons

$$\text{RFoll}(F) = (\text{Last}(F) \times \text{First}(F)) \setminus \text{Follow}(F) .$$

Le but des questions suivantes est de montrer que si  $F$  est une expression rationnelle locale, cet ensemble peut se calculer inductivement.

**Question 30.** Donner les valeurs de  $\text{RFoll}(\varepsilon)$ ,  $\text{RFoll}(a)$  pour une lettre  $a \in \Sigma$  et  $\text{RFoll}(E^*)$  pour une expression rationnelle locale  $E$ .

**Question 31.** Montrer que si  $E$  et  $F$  sont des expressions rationnelles locales, on a :

$$\begin{aligned} \text{RFoll}(E + F) &= \text{RFoll}(E) \sqcup \text{RFoll}(F) \sqcup \text{Last}(E) \times \text{First}(F) \sqcup \text{Last}(F) \times \text{First}(E) \\ \text{RFoll}(E \cdot F) &= \text{Last}(F) \times \text{First}(E) \sqcup \text{Null}(F) \cdot \text{RFoll}(E) \sqcup \text{Null}(E) \cdot \text{RFoll}(F) , \end{aligned}$$

où le produit  $b \cdot X$  d'un booléen  $b$  par un ensemble  $X$  est l'ensemble vide si  $b$  vaut faux, et l'ensemble  $X$  si  $b$  vaut vrai.

**Question 32.** Expliquer brièvement (sans écrire de code) comment on peut utiliser cette formule pour améliorer la complexité du calcul de Follow de manière efficace.

**Question 33.** Expliquer (sans écrire de code) comment on peut implanter des ensembles avec des listes chaînées de sorte que la complexité de l'extension disjointe soit constante et les complexités des autres opérations présentes dans la classe `Set` restent inchangées. Quelle serait la conséquence d'un tel choix sur la complexité de la fonction `glushkov` (en supposant qu'on a utilisé l'implantation suggérée à la question 32) ?

### Partie III. Test d'égalité de langages reconnus par automates

Pour tester si deux expressions  $\mathcal{E}$  et  $\mathcal{F}$  décrivent le même langage, on se propose de calculer leurs automates de Glushkov, qui sont ensuite déterminisés grâce à la construction classique par sous-ensembles. Il faut alors décider si deux automates déterministes reconnaissent le même langage. Nous verrons deux façons de procéder.

On travaillera uniquement sur des automates déterministes et complets. Soit  $\mathcal{A} = (\Sigma, Q, \delta, i_0, F)$  un tel automate où :

- $\Sigma$  est un alphabet fini non vide contenant les *lettres* ;
- $Q$  est un ensemble fini non vide contenant les *états* ;
- $\delta : Q \times \Sigma \rightarrow Q$  est l'application des *transitions* ;
- $i_0 \in Q$  est l'état initial ;
- $F \subseteq Q$  est l'ensemble des *états terminaux*.

Quand  $\delta(p, a) = q$  on dira qu'il y a une transition de  $p$  à  $q$  étiquetée par  $a$ , et on le notera également  $p \xrightarrow{a} q$ . La *taille* d'un automate est son nombre d'états, elle est notée  $\|\mathcal{A}\|$ . On a ainsi  $\|\mathcal{A}\| = |Q|$ . On notera  $\mathcal{L}(\mathcal{A})$  le langage reconnu par  $\mathcal{A}$ .

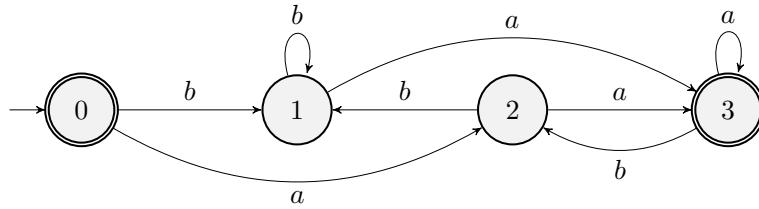
## 5 Partitions des états

On rappelle que si  $E$  est ensemble fini non vide, une *partition*  $\mathcal{P}$  de  $E$  est un ensemble  $\{E_0, \dots, E_{\ell-1}\}$  de sous-ensembles non vides de  $E$  dont l'union est  $E$  tout entier ( $\bigcup_{i=0}^{\ell-1} E_i = E$ ) et qui sont deux à deux disjoints : si  $i$  et  $j$  sont deux indices différents entre 0 et  $\ell-1$ , alors  $E_i \cap E_j = \emptyset$ . Chaque  $E_i$  est appelé une *part* de la partition. Par exemple,  $\mathcal{P} = \{\{0, 1, 4\}, \{2, 5\}, \{3\}\}$  est une partition de  $\{0, 1, 2, 3, 4, 5\}$  en trois parts.

Dans toute la suite on ne considérera que des partitions d'ensembles d'états d'automates déterministes et complets. Si  $\mathcal{A} = (\Sigma, Q, \delta, i_0, F)$  est un tel automate, et  $\mathcal{P}$  est une partition de  $Q$ , alors pour tout  $(p, q) \in Q^2$  on note  $p \sim_{\mathcal{P}} q$  le fait que  $p$  et  $q$  sont dans la même part de la partition  $\mathcal{P}$ . On définit le  $\Sigma$ -raffinement de  $\mathcal{P}$  comme étant l'unique partition  $\mathcal{P}'$  de  $Q$  telle que

$$\forall (p, q) \in Q^2, \quad p \sim_{\mathcal{P}'} q \iff \begin{cases} p \sim_{\mathcal{P}} q, \\ \text{et} \\ \forall a \in \Sigma, \delta(p, a) \sim_{\mathcal{P}} \delta(q, a). \end{cases} \quad (1)$$

**Question 34.** On considère l'automate ci-dessous et la partition  $\mathcal{P} = \{\{0, 1\}, \{2, 3\}\}$  de son ensemble d'états. Calculer le  $\Sigma$ -raffinement de  $\mathcal{P}$ .



On souhaite à présent calculer efficacement le  $\Sigma$ -raffinement d'une partition de l'ensemble d'états d'un automate. Pour simplifier les représentations en machine, on considérera que l'alphabet  $\Sigma$  est  $\{0, \dots, m - 1\}$  et que si l'automate possède  $n$  états, son ensemble d'états est  $\{0, \dots, n - 1\}$ . Un tel automate déterministe et complet sera encodé en PYTHON par un tuple `(m, n, delta, q0, F)`, où  $m$  est le nombre de lettres de l'alphabet,  $n$  est le nombre d'états de l'automate,  $q0$  est l'état initial (un entier entre 0 et  $n - 1$ ) et

- `delta` est une liste contenant  $n$  listes de longueur  $m$ , appelé la *table des transitions* ; pour tous états  $p, q \in \{0, \dots, n - 1\}$  et toute lettre  $a \in \{0, \dots, m - 1\}$  on a `delta[p][a] = q` si et seulement si  $p \xrightarrow{a} q$  (on rappelle que les automates de cette partie sont déterministes et complets, ce qui permet un tel encodage) ;
- `F` est une liste de  $n$  booléens avec, pour tout état  $p \in \{0, \dots, n - 1\}$ , `F[p] == True` si et seulement si  $p$  est terminal.

Les partitions de l'ensemble des états d'un automate sont représentées de la façon suivante. Si  $\mathcal{P}$  est une partition de  $\{0, \dots, n - 1\}$  qui contient  $c$  parts, on encodera  $\mathcal{P}$  par une liste `part` de longueur  $n$ , contenant des entiers de  $\{0, \dots, c - 1\}$ , et tel que pour tous états  $p, q \in \{0, \dots, n - 1\}$ ,  $p$  et  $q$  sont dans la même part de  $\mathcal{P}$  si et seulement si `part[p] == part[q]`. Autrement dit, on numérote les parts avec des nombres entre 0 et  $c - 1$ , et la liste `part` associe à chaque état le numéro de sa part. On remarque qu'un tel encodage n'est pas unique car on peut échanger les numéros des parts.

**Exemple.** Si  $n = 7$  et que la partition est  $\{\{0, 3, 4\}, \{1\}, \{2, 5, 6\}\}$ , on peut la représenter par `[0, 1, 2, 0, 0, 2, 2]` ou encore `[1, 2, 0, 1, 1, 0, 0]`.

**Question 35.** Écrire la fonction PYTHON `nombre_parts(part)` qui calcule le nombre de parts de la partition encodée par `part`, en temps  $\mathcal{O}(n)$ , où  $n$  est la taille de `part`. On ne demande pas de justifier la complexité.

On souhaite écrire la fonction PYTHON `raffine(A, part)` qui calcule et renvoie le  $\Sigma$ -raffinement de la partition encodée par une liste `part` de l'ensemble d'états d'un automate `A` donné également en argument en utilisant l'équation (1). Pour cela, on va associer à chaque état  $p$  la liste `s[p]` de longueur  $m + 2$  suivante :

$$s[p] = [part[p], part[\delta[p][0]], \dots, part[\delta[p][m-1]], p].$$

On encode donc dans `s[p]` les informations utiles pour l'équation (1) ; on y a ajouté `p` à la fin pour pouvoir directement retrouver l'état  $p$  à partir de `s[p]`.

**Question 36.** En utilisant le tri lexicographique de la section 2, écrire une implantation en PYTHON de la fonction `raffine(A, part)` en temps  $\mathcal{O}(nm)$  : elle doit renvoyer un encodage du  $\Sigma$ -raffinement de la partition encodée par `part`, où `A` est un encodage de l'automate déterministe et complet avec  $n$  états sur un alphabet de taille  $m$ . On ne demande pas de justifier la complexité.

## 6 Partition de Nerode

Soit  $\mathcal{A} = (\Sigma, Q, \delta, i_0, F)$  un automate déterministe et complet. Soit  $\bar{F} = Q \setminus F$  le complémentaire de  $F$  dans  $Q$ . La partition  $\mathcal{P}_0$  de  $Q$  est la partition telle que pour tout  $(p, q) \in Q^2$ ,  $p \sim_{\mathcal{P}_0} q$  si et seulement si  $(p, q) \in F^2$  ou  $(p, q) \in \bar{F}^2$  : autrement dit  $p$  et  $q$  sont dans la même part de  $\mathcal{P}_0$  quand ils sont soit tous les deux terminaux, soit tous les deux non-terminaux.

**Question 37.** Écrire une fonction PYTHON `calcule_partition0(A)` qui calcule et renvoie la partition  $\mathcal{P}_0$  des états de l'automate déterministe et complet encodé par `A`, en temps  $\mathcal{O}(n)$ . On ne demande pas de justifier la complexité.

Soit  $\mathcal{A}$  un automate déterministe et complet sur l'alphabet  $\Sigma$ . Pour tout entier  $i \geq 1$ , on définit récursivement la partition  $\mathcal{P}_i$  comme étant le  $\Sigma$ -raffinement de  $\mathcal{P}_{i-1}$ . Il s'agit donc d'itérer  $i$  fois le  $\Sigma$ -raffinement de  $\mathcal{P}_0$ . Pour simplifier les notations, pour tout  $i \in \mathbb{N}$  on notera  $\sim_i$  au lieu de  $\sim_{\mathcal{P}_i}$  dans la suite.

**Question 38.** Calculer  $\mathcal{P}_0$ ,  $\mathcal{P}_1$  et  $\mathcal{P}_2$  pour l'automate de la question 34.

Pour un alphabet  $\Sigma$  et un entier  $i \geq 0$ , on note  $\Sigma^{\leq i}$  l'ensemble des mots sur  $\Sigma$  dont la longueur est au plus  $i$  :  $\Sigma^{\leq i} = \{u \in \Sigma^* \mid |u| \leq i\}$ . Si  $\mathcal{A} = (\Sigma, Q, \delta, i_0, F)$  est un automate déterministe et complet, pour tout  $q \in Q$  on note  $\mathcal{A}_q = (\Sigma, Q, \delta, q, F)$ , l'automate obtenu en déplaçant l'état initial en  $q$ . On note  $\mathcal{L}_q$  le langage reconnu par  $\mathcal{A}_q$ . Ainsi, par exemple,  $\mathcal{L}_{i_0}$  est le langage  $\mathcal{L}(\mathcal{A})$  reconnu par  $\mathcal{A}$  car  $\mathcal{A} = \mathcal{A}_{i_0}$ .

**Question 39.** Soit  $\mathcal{A} = (\Sigma, Q, \delta, i_0, F)$  un automate déterministe et complet. Montrer que pour tout  $i \in \mathbb{N}$  et pour tout  $(p, q) \in Q^2$ ,  $p \sim_i q$  si et seulement si  $\mathcal{L}_p \cap \Sigma^{\leq i} = \mathcal{L}_q \cap \Sigma^{\leq i}$ .

Soit  $\mathcal{A} = (\Sigma, Q, \delta, i_0, F)$  un automate déterministe et complet. On définit la *partition de Nerode*  $\mathcal{N}_{\mathcal{A}}$  associée à  $\mathcal{A}$  comme étant l'unique partition de  $Q$  telle que pour tout  $(p, q) \in Q^2$ ,  $p$  et  $q$  sont dans la même part de  $\mathcal{N}_{\mathcal{A}}$  si et seulement si  $\mathcal{L}_p = \mathcal{L}_q$ . Pour simplifier les notations, on écrira  $p \sim q$  au lieu de  $p \sim_{\mathcal{N}_{\mathcal{A}}} q$  pour indiquer que  $p$  et  $q$  sont dans la même part de  $\mathcal{N}_{\mathcal{A}}$ . L'algorithme de Moore ci-dessous permet de calculer la partition de Nerode d'un automate encodé par `A` (on considère qu'appliquer la fonction `nombre_parts` à `None` renvoie 0).

```

def moore(A):
 """A est un automate déterministe et complet,
 qui respecte l'encodage du sujet. la fonction renvoie
 la partition de Nerode de A."""
 ancienne_part = None
 part = calcule_partition0(A)
 while nombre_parts(ancienne_part) != nombre_parts(part):
 ancienne_part = part
 part = raffine(A, part)
 return part

```

**Question 40.** Montrer que si l'automate possède  $n \geq 2$  états, il y a au plus  $n - 1$  itérations de la boucle `while`.

**Question 41.** Montrer la correction totale de l'algorithme de Moore.

**Question 42.** Montrer que l'algorithme de Moore a une complexité en  $\mathcal{O}(mn^2)$ , où  $n$  est le nombre d'états de l'automate et  $m$  le nombre de lettres de l'alphabet.

**Question 43.** Montrer que l'algorithme de Moore a une complexité en  $\Theta(mn^2)$ , où  $n$  est le nombre d'états de l'automate et  $m$  le nombre de lettres de l'alphabet.

## 7 Test d'égalité utilisant la partition de Nerode

Pour les deux questions suivantes, vous pouvez utiliser la définition de la partition de Nerode, ou bien le calcul qui en est fait par l'algorithme de Moore, qui est correct d'après la question 41.

**Question 44.** Soient  $p$  et  $q$  deux états de l'automate. Montrer que si  $p \sim q$  alors  $(p, q) \in F^2$  ou  $(p, q) \in \overline{F}^2$ .

**Question 45.** Soient  $p$  et  $q$  deux états de l'automate. Montrer que si  $p \sim q$  alors pour tout lettre  $a \in \Sigma$ , on a  $\delta(p, a) \sim \delta(q, a)$ .

Les deux propriétés des questions 44 et 45 permettent de définir l'*automate quotient* de  $\mathcal{A}$  par sa partition de Nerode qui est l'automate déterministe et complet  $\mathcal{A}/\sim = (\Sigma, \mathcal{N}_{\mathcal{A}}, \delta_{\sim}, Q_0, F_{\sim})$  dont l'ensemble d'états est l'ensemble des parts  $Q_0, Q_1, \dots, Q_{\ell-1}$  de  $\mathcal{N}_{\mathcal{A}}$ , où  $Q_0$  est la partie de  $i_0$  et :

- pour tout  $Q_i \in \mathcal{N}_{\mathcal{A}}$  et pour tout  $a \in \Sigma$ , on définit  $\delta_{\sim}(Q_i, a)$  comme étant l'unique partie  $Q_j$  de  $\mathcal{N}_{\mathcal{A}}$  telle que pour tout  $p \in Q_i$ ,  $\delta_{\sim}(p, a) \in Q_j$  (cela est garanti par la question 45) ;
- $F_{\sim}$  est l'ensemble des  $Q_i \in \mathcal{N}_{\mathcal{A}}$  composés uniquement d'états terminaux (la question 44 assure que les états d'une même partie sont tous terminaux ou tous non-terminaux).

**Question 46.** Calculer l'automate quotient de  $\mathcal{A}$  par sa partition de Nerode, où  $\mathcal{A}$  est l'automate de la question 34.

Pour finaliser le test d'équivalence, on utilise un résultat (admis) de théorie des automates : soient  $\mathcal{A} = (\Sigma, Q_{\mathcal{A}}, \delta_{\mathcal{A}}, i_{\mathcal{A}}, F_{\mathcal{A}})$  et  $\mathcal{B} = (\Sigma, Q_{\mathcal{B}}, \delta_{\mathcal{B}}, i_{\mathcal{B}}, F_{\mathcal{B}})$  deux automates déterministes et complets dont tous les états sont accessibles depuis leurs états initiaux. Alors  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$  si et seulement si  $\mathcal{A}/\sim$  et  $\mathcal{B}/\sim$  sont *isomorphes*, c'est-à-dire qu'il existe une bijection  $\phi$  de  $Q_{\mathcal{A}}$  dans  $Q_{\mathcal{B}}$  telle que

- $\phi(i_{\mathcal{A}}) = i_{\mathcal{B}}$ ,
- pour tout  $p \in Q_{\mathcal{A}}$  et tout  $a \in \Sigma$ ,  $\phi(\delta_{\mathcal{A}}(p, a)) = \delta_{\mathcal{B}}(\phi(p), a)$ ,
- $\phi(F_{\mathcal{A}}) = F_{\mathcal{B}}$ .

En particulier, ce résultat utilise le fait que  $\mathcal{A}$  et  $\mathcal{A}/\sim$  reconnaissent le même langage, ce qu'on ne vous demande pas de montrer.

**Question 47.** Sans écrire le programme, expliquer comment on peut en temps  $\mathcal{O}(|\Sigma|(\|\mathcal{A}\| + \|\mathcal{B}\|))$  tester si deux automates déterministes et complets  $\mathcal{A}$  et  $\mathcal{B}$ , dont tous les états sont accessibles depuis leurs états initiaux, sont isomorphes. On rappelle que  $\|\mathcal{A}\|$  est le nombre d'états de l'automate  $\mathcal{A}$  et  $|\Sigma|$  est le cardinal de l'alphabet.

En conclusion de cette section, on remarque qu'on peut en déduire un algorithme pour tester en temps  $\mathcal{O}(m n^2)$  si deux automates déterministes et complets  $\mathcal{A}$  et  $\mathcal{B}$  reconnaissent le même langage, où  $n = \max\{\|\mathcal{A}\|, \|\mathcal{B}\|\}$  et  $m = |\Sigma|$  :

- on enlève les états qui ne sont pas accessibles depuis les états initiaux dans  $\mathcal{A}$  et  $\mathcal{B}$  en les identifiants avec un parcours en profondeur ;
- on applique l'algorithme de Moore aux deux automates, et on calcule leur automates quotients par leurs partitions de Nerode respectives ;
- on teste si les deux automates quotients sont isomorphes.

## 8 Test d'égalité avec la structure “unir & trouver”

L'objet de cet section est de proposer un algorithme plus efficace pour déterminer si deux automates déterministes et complets  $\mathcal{A} = (\Sigma, Q_{\mathcal{A}}, \delta_{\mathcal{A}}, i_{\mathcal{A}}, F_{\mathcal{A}})$  et  $\mathcal{B} = (\Sigma, Q_{\mathcal{B}}, \delta_{\mathcal{B}}, i_{\mathcal{B}}, F_{\mathcal{B}})$ , définis sur le même alphabet  $\Sigma$ , reconnaissent le même langage.

L'algorithme étudié utilise également des partitions mais diffère fondamentalement des parties précédentes.

On travaille sur les deux automates simultanément, l'ensemble dont on considère les partitions est  $Q = Q_{\mathcal{A}} \cup Q_{\mathcal{B}}$ , l'union des ensembles d'états de  $\mathcal{A}$  et de  $\mathcal{B}$ , que l'on suppose disjoints. Si  $p \in Q$ , on note  $\mathcal{L}_p$  le langage reconnu en changeant l'état initial de l'automate qui contient  $p$  pour le placer en  $p$ .

Informellement, l'algorithme fonctionne de la façon suivante. Au début, chaque état est seul dans sa part. Ensuite, on fait l'union, autant que possible, des parts des états  $p$  et  $q$  qui

```

class Partition(builtins.object):
 Methods defined here:

 __init__(self, n)
 initialise une partition de {0,...,n-1}
 où chaque élément est seul dans sa part

 trouver(self, x)
 renvoie un identifiant unique de la part de x :
 x et y sont dans la même part si et seulement si
 trouver(x) == trouver(y)

 unir(self, x, y)
 modifie la partition en réalisant l'union des parts de x et de y

```

FIGURE 2 – Méthodes de la classe `Partition` implantant la structure `Unir` et `Trouver`.

doivent nécessairement vérifier  $\mathcal{L}_p = \mathcal{L}_q$  si  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$  : par exemple, la première union consiste à créer la part  $\{i_{\mathcal{A}}, i_{\mathcal{B}}\}$ , puisque si  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$  alors  $\mathcal{L}_{i_{\mathcal{A}}} = \mathcal{L}_{i_{\mathcal{B}}}$ . Toujours sous l'hypothèse que  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ , pour chaque  $a \in \Sigma$ , on doit avoir  $\mathcal{L}_p = \mathcal{L}_q$  quand  $p = \delta_{\mathcal{A}}(i_{\mathcal{A}}, a)$  et  $q = \delta_{\mathcal{B}}(i_{\mathcal{B}}, a)$ , il faut donc réaliser l'union des parts contenant  $\delta_{\mathcal{A}}(i_{\mathcal{A}}, a)$  et  $\delta_{\mathcal{B}}(i_{\mathcal{B}}, a)$  pour toute lettre  $a$ . L'algorithme utilise une pile, implantée par une liste PYTHON, pour garder la trace des paires d'états à considérer à chaque étape. L'algorithme a deux façons de s'arrêter :

- soit il doit unir les parts de  $p$  et  $q$  alors que seulement l'un de ces deux états est final, auquel cas il renvoie faux ;
- soit il n'y a plus de paire d'états à considérer, auquel cas il renvoie vrai.

Une implantation de l'algorithme est proposée à la Fig. 3 page 17 elle utilise la méthode `pop()` qui permet d'enlever le dernier élément d'une liste PYTHON et de le renvoyer, en temps constant. Elle utilise également une implantation de la structure de données “unir & trouver” (“union & find” en anglais), qui permet de travailler sur des partitions d'un ensemble fini non vide  $\{0, \dots, n-1\}$  comme décrit Fig. 2. L'implantation est réalisée de sorte que si on initialise une partition de taille  $n$  puis que l'on effectue  $t$  appels à la fonction `trouver` ou `unir`, la complexité totale en temps est en  $\mathcal{O}(t\alpha(n))$ , où  $\alpha$  est l'inverse de la fonction d'Ackermann. La fonction  $\alpha$  tend vers l'infini extrêmement lentement (on considère en général que  $\alpha(n) \leq 5$  pour les valeurs de  $n$  que l'on peut utiliser en pratique).

L'entrée de `test_egalite_langages` est constituée des deux automates déterministes et complets  $\mathcal{A}$  et  $\mathcal{B}$ , définis sur le même alphabet, encodés par  $A=(mA,nA,deltaA,iA,FA)$  et  $B=(mB,nB,deltaB,iB,FB)$ . Comme cette fonction doit travailler avec une partition de l'union de  $Q_{\mathcal{A}}$  et de  $Q_{\mathcal{B}}$  et que dans notre choix d'encodage  $Q_{\mathcal{A}} = \{0, \dots, n_{\mathcal{A}}-1\}$  et  $Q_{\mathcal{B}} = \{0, \dots, n_{\mathcal{B}}-1\}$  ne sont pas disjoints, on différencie les états de  $Q_{\mathcal{B}}$  en leur ajoutant  $n_{\mathcal{A}}$  : l'ensemble de la partition est  $\{0, \dots, n_{\mathcal{A}} + n_{\mathcal{B}}-1\}$ , et les entiers de  $\{0, \dots, n_{\mathcal{A}}-1\}$  représentent les états de  $\mathcal{A}$ , alors que les entiers de  $\{n_{\mathcal{A}}, \dots, n_{\mathcal{A}} + n_{\mathcal{B}}-1\}$  représentent les états de  $\mathcal{B}$ .

**Question 48.** Montrer qu'à tout moment lors de l'exécution de `test_egalite_langages`, pour tout couple d'états  $(p, q)$  de la liste `a_faire`, il existe un mot  $u \in \Sigma^*$  tel que  $\delta_{\mathcal{A}}(i_{\mathcal{A}}, u) = p$  et  $\delta_{\mathcal{B}}(i_{\mathcal{B}}, u) = q$ . En déduire que si le programme renvoie faux, les langages reconnus par  $\mathcal{A}$  et par  $\mathcal{B}$  sont différents.

```

def test_egalite_langages(A, B):
 """A et B sont des encodages d'automates déterministes et complets
 sur le même alphabet"""
 mA, nA, deltaA, iA, FA = A
 mB, nB, deltaB, iB, FB = B
 assert(mA == mB)
 partition = Partition(nA + nB)
 a_faire = [(iA, iB)]
 while len(a_faire)>0:
 p, q = a_faire.pop()
 if partition.trouver(p) != partition.trouver(q + nA):
 if FA[p] != FB[q]:
 return False
 else:
 partition.unir(p, q + nA)
 for a in range(mA):
 a_faire.append((deltaA[p][a], deltaB[q][a]))
 return True

```

FIGURE 3 – Programme PYTHON pour tester l'égalité des deux langages reconnus par les automates déterministes et complets  $\mathcal{A}$  et  $\mathcal{B}$ .

Dans la suite, pour tout état  $p \in Q_{\mathcal{A}} \cup Q_{\mathcal{B}}$  et toute lettre  $a \in \Sigma$ , on note  $\delta(p, a) = \delta_{\mathcal{A}}(p, a)$  si  $p \in Q_{\mathcal{A}}$  et  $\delta(p, a) = \delta_{\mathcal{B}}(p, a)$  si  $p \in Q_{\mathcal{B}}$ .

**Question 49.** On suppose que le programme `test_egalite_langages` renvoie vrai. On note  $\mathcal{P}_{\text{fin}}$  la partition obtenue à la fin de l'exécution du programme. Montrer la propriété suivante : à tout moment lors de l'exécution du programme, si deux états  $p$  et  $q$  sont dans la même part  $P$  de la partition courante `partition`, alors, à la fin de l'exécution,  $\delta(p, a)$  et  $\delta(q, a)$  sont dans la même part de  $\mathcal{P}_{\text{fin}}$ , pour toute lettre  $a \in \Sigma$ . On pourra utiliser une récurrence sur le cardinal de la part  $P$ .

**Question 50.** En déduire que si le programme renvoie vrai, les deux automates reconnaissent le même langage.

**Question 51.** Montrer que `test_egalite_langages` s'exécute en temps  $\mathcal{O}(m n \alpha(n))$ , où  $n = \|\mathcal{A}\| + \|\mathcal{B}\|$  et  $m = |\Sigma|$ .

\* \*  
\*

**EAE INF 3****SESSION 2023****AGREGATION  
CONCOURS EXTERNE****Section : INFORMATIQUE****ÉPREUVE SPÉCIFIQUE SELON L'OPTION CHOISIE :**

- **ÉTUDE DE CAS INFORMATIQUE**
- **FONDEMENTS DE L'INFORMATIQUE**

Durée : 6 heures

*L'usage de tout ouvrage de référence, de tout dictionnaire et de tout matériel électronique (y compris la calculatrice) est rigoureusement interdit.*

*Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.*

*Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.*

**NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.**

**Le fait de rendre une copie blanche est éliminatoire.**

**Tournez la page S.V.P.****A**

**INFORMATION AUX CANDIDATS**

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

► **Etude de cas informatique :**

| Concours | Section/option | Epreuve | Matière |
|----------|----------------|---------|---------|
| EAE      | 6200A          | 103     | 9424    |

► **Fondement de l'informatique :**

| Concours | Section/option | Epreuve | Matière |
|----------|----------------|---------|---------|
| EAE      | 6200A          | 103     | 9425    |

---

## Épreuve spécifique

---

|                                    |    |
|------------------------------------|----|
| Étude de cas informatique .....    | 3  |
| Fondements de l'informatique ..... | 16 |



---

## Étude de cas informatique

---

### Préliminaires

L'énoncé proposé s'inspire d'un projet concernant des livraisons par des véhicules. Chaque partie comprend la définition d'un certain nombre de problématiques à résoudre et présente des objectifs concrets, ainsi que la réflexion sur les moyens de les atteindre.

**Attendus.** Il est attendu des candidates et des candidats des réponses construites. Ils seront aussi évalués sur la précision, le soin et la clarté de la rédaction.

**Dépendances.** Ce sujet contient cinq parties. Les différentes parties et un grand nombre de leurs questions sont largement indépendantes. Il est possible d'aborder les différentes parties dans l'ordre qui vous conviendra le mieux mais en indiquant clairement quelle question est répondue.

### Partie I. Déterminer les tournées de véhicules

Dans cette partie, nous allons nous intéresser au problème de tournée de véhicules : un entrepôt stocke des produits qui doivent être livrés à des clients. Chaque client commande un nombre de produits (qui peuvent être différents). Les produits sont livrés par véhicule, par exemple un camion, et le but est de trouver des tournées de livraison qui minimisent la distance totale de kilomètres parcourus par les véhicules.

Ce problème est NP-complet (avec un véhicule seulement, c'est déjà le problème du voyageur de commerce) et nous allons donc nous intéresser à des heuristiques.

Nous allons utiliser la représentation via un graphe non-orienté  $G = (V, E)$  avec les sommets représentant les différentes implantations des clients et les arêtes représentent les chemins entre les clients. L'entrepôt se situe au sommet  $v_0$  et les  $n$  sommets restants représentent les clients. On considère que tous les sommets sont accessibles à partir de  $v_0$  (donc le graphe est connexe). De plus, les arêtes  $e_{ij}$  sont pondérées par  $d_{ij}$ , la distance entre  $v_i$  et  $v_j$ . Le nombre maximum de véhicules est  $n$ , ce qui serait la situation dans laquelle exactement un véhicule est associé à chaque destination de livraison (client). Chaque véhicule a une capacité maximale  $C$  de produits qu'il peut charger en nombre de palettes. Pour finir, chaque client  $v_i$ ,  $i \geq 1$ , a une demande  $w_i$  de produits. Un exemple est donné dans la figure 1 où on voit le graphe d'un réseau routier (figure 1(a) avec l'entrepôt au milieu et 4 clients) et sa représentation sous forme de matrice d'adjacence (figure 1(c)) ainsi que les demandes des clients (table 1(b) dans la figure 1).

Pour commencer, nous allons calculer les plus courtes distances entre tous les sommets pour un graphe donné. Ceci peut être fait avec l'**algorithme de Floyd-Warshall**. L'algorithme calcule, pour chaque paire de sommets, la distance minimale parmi tous les chemins entre ces

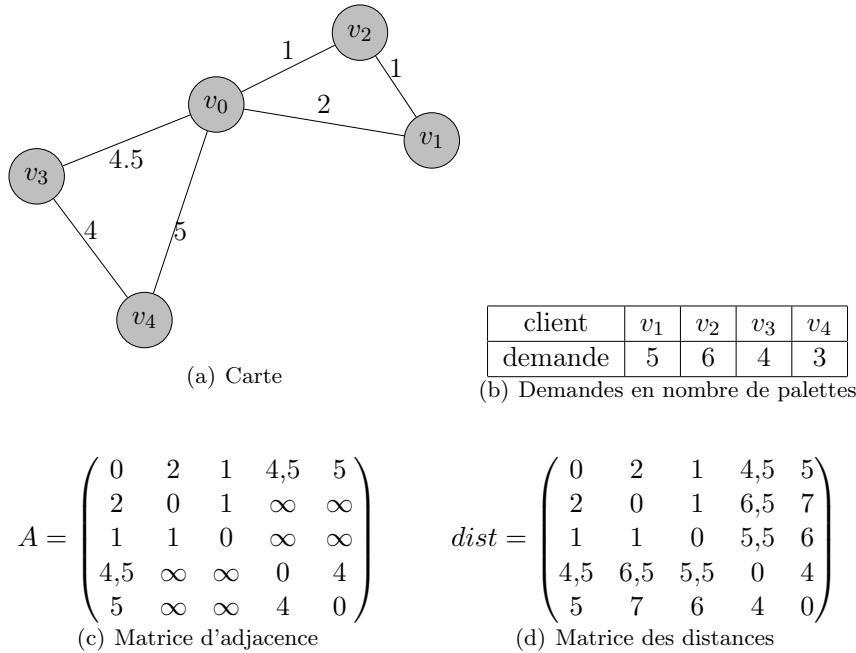


FIGURE 1 – Exemple d'un graphe et sa représentation sous forme de matrice d'adjacence et sous forme de matrice des distances.

deux sommets. Le pseudo-code est donné dans l'algorithme 1. La matrice des distances produite par l'algorithme 1 en prenant la matrice d'adjacence de la figure 1(c)) en entrée, est donnée dans la figure 1(d)).

**Question 1.** Proposer une implémentation en Python de l'algorithme Floyd-Warshall présenté dans l'algorithme 1. La fonction prend la matrice  $A$  d'adjacence pondérée d'un réseau routier et produit une matrice  $dist$  des plus courtes distances entre tous les sommets. On considère pouvoir utiliser `np.inf` comme infini.

```

 $dist \leftarrow A$
foreach vertex $z \in V$ do
 foreach vertex $x \in V$ do
 foreach vertex $y \in V$ do
 if $dist(x, z) \neq \infty \& dist(z, y) \neq \infty \& dist(x, z) + dist(z, y) < dist(x, y)$ then
 $dist(x, y) \leftarrow dist(x, z) + dist(z, y)$
return $dist$

```

**Algorithme 1 :** Floyd-Warshall.

Maintenant que nous savons créer la matrice des plus courtes distances à partir d'une matrice d'adjacence, dans la suite du sujet nous allons considérer directement des matrices de distances en entrée des algorithmes.

Nous allons aborder le problème des tournées des véhicules. Nous supposons que la demande de chaque client est inférieure à la capacité des véhicules, c'est-à-dire un client peut être livré en une fois par un véhicule.

Une première approche pour créer des tournées de véhicules peut être la suivante (**algorithme naïf**) : étant donnée la matrice des distances les plus courtes entre tous les points de la carte produit par l'algorithme Floyd-Warshall, on prend un premier véhicule et on lui affecte le client le plus proche. Puis, depuis ce client, on rajoute le client le plus proche de lui qui n'est pas encore affecté à la tournée si la capacité de chargement du véhicule n'est pas dépassée par cette affectation. On continue ainsi jusqu'à ce que la capacité restante du véhicule soit trop petite pour livrer en entier le client qu'on souhaite rajouter à la tournée. Dans ce cas, on commence une nouvelle tournée avec un autre véhicule selon le même principe jusqu'à ce que tous les clients soient affectés à des tournées. Chaque véhicule rentre à l'entrepôt à la fin de sa tournée.

**Question 2.** On considère la carte de la figure 1(a) et sa matrice des plus courtes distances (figure 1(d)). Chaque véhicule peut charger  $C=12$  palettes. Les demandes de palettes des clients sont données dans la table 1(b) de la figure 1.

1. Appliquer l'**algorithme naïf** expliqué au dessus et donner les tournées avec leurs coûts en termes de nombre de kilomètres pour l'ensemble des véhicules. Le coût d'une tournée inclut le retour du véhicule à l'entrepôt via le plus court chemin.
2. En considérant l'objectif de minimiser la distance totale des kilomètres parcourus par tous les véhicules, peut-on trouver une meilleure solution ? Si oui, quelle est-elle ?

**Question 3.** Est-ce que la suppression du client  $v_2$  change quelque chose pour la répartition des clients restants en tournées ? Expliquer.

**Question 4.** Implémenter en Python l'**algorithme naïf** via la fonction `AlgoNaif(dist, demandes, C)`. L'algorithme prend en entrée une matrice de distances entre tous les clients et l'entrepôt, les demandes des clients sous forme d'une liste ainsi que la capacité  $C$  (constante entière) des véhicules à disposition.

**Question 5.** Il est possible de trouver des situations dans lesquelles l'algorithme n'est pas optimal concernant notre critère de la distance totale des tournées. Donner deux configurations pour lesquelles l'algorithme se fait piéger. Un ou plusieurs dessins sont attendus.

Nous allons maintenant regarder une approche plus sophistiquée, l'**algorithme de Clarke et Wright (Savings algorithm)**, dont le principe est le suivant :

1. On commence avec  $n$  tournées :  $v_0 \rightarrow v_i \rightarrow v_0$ , pour tout  $i \geq 1$ , c'est-à-dire chaque client est livré par un véhicule différent.
2. Calculer ensuite les économies  $s(i, j)$  (*savings*) pour fusionner deux clients  $v_i$  et  $v_j$  en une même tournée :  $s(i, j) = dist(i, 0) + dist(0, j) - dist(i, j)$ , pour tout  $i, j \geq 1$  et  $i \neq j$  ;
3. Trier les économies par ordre décroissant ;

4. Commencer en tête de la liste (restante) des économies, fusionner les deux tournées associées avec l'économie (restante) la plus élevée, si :
  - (a) Les deux clients ne sont pas déjà dans la même tournée ;
  - (b) Aucun des deux clients n'est à l'intérieur de sa tournée : Les deux clients sont connectés directement à l'entrepôt dans leur tournée respective et donc sont livrés en premier ou en dernier dans leur tournée respective ;
  - (c) La somme des demandes des deux tournées à fusionner ne dépasse pas la capacité maximum du véhicule.
5. Répéter l'étape 4 jusqu'à ce que plus aucune économie ne puisse être faite.

Nous allons utiliser une classe `Tournee` qui représente la tournée d'un véhicule et qui propose les méthodes suivantes :

- `distance(self, distance)` : qui renvoie la longueur actuelle de la tournée,
- `fusionnable(self, autreTournee, clients, capacite)` : teste si la tournée peut être fusionnée avec `autreTournee`. Elle teste notamment les différents cas du point (2) de l'**algorithme de Clarke et Wright**.
- `fusion(self, autreTournee, clients)` : qui prend en entrée la tournée avec laquelle elle doit faire la fusion et une liste de clients par lesquels la tournée fusionnée doit être reliée.
- `affiche(self)` : affiche la tournée avec la liste des différents clients à parcourir et le chargement total de cette tournée.

Le listing 1 donne des exemples d'appel.

Listing 1 – Code d'utilisation de la classe `Tournee`.

```
tournee1.affiche() # tournee 0 2 3 4 0 chargement 8
tournee2 = Tournee(1,3) #client 1 demande 3 palettes
tournee1.fusionnable(tournee2, [3,1], 12) #False
tournee1.fusionnable(tournee2, [2,1], 12) #True
tournee1.fusion(tournee2, [2,1])
tournee1.affiche() # tournee 0 4 3 2 1 0, chargement 11
le parcours 0-4-3-2-1-0 est équivalent à 0-1-2-3-4-0
```

**Question 6.** Implémenter la classe `Tournee` en Python.

**Question 7.** En utilisant la classe `Tournee`, implémenter en Python l'**algorithme de Clarke et Wright** via la fonction `ClarkeWright(dist, demandes, C)`. L'algorithme prend en entrée la matrice des distances des plus courts chemins, les demandes des clients et la capacité des véhicules. Il renvoie la liste des tournées ainsi que la longueur totale des tournées.

**Question 8.** Il peut être nécessaire de livrer certaines clients en urgence. Discuter (sans faire l'implémentation dans le code) des modifications de l'algorithme précédent nécessaires pour prendre en compte des priorités dans la liste des clients.

La version actuelle du problème de tournée de véhicules fait abstraction d'un certain nombre de contraintes supplémentaires qui font partie du cas réel, telles que le temps de conduite des conducteurs, la date de contrôle technique des véhicules, leur capacité ou leur plaque d'immatriculation.

On veut maintenant tenir compte de la différence de caractéristiques des différents véhicules. On considère avoir une classe `Vehicule` dont une instance sera associée à chaque tournée. Un véhicule sera associé à une tournée lors de la création de cette dernière. Dans les trois questions suivantes, on se limitera à considérer qu'un véhicule ne possède comme attributs que sa capacité en nombre de palettes (un entier) et une plaque d'immatriculation (une chaîne de caractères). On veut modifier la classe `Tournee` pour intégrer les véhicules, sans modifier la liste des méthodes de cette classe.

**Question 9.** Est-il nécessaire de modifier les arguments des méthodes de `Tournee`? Si oui, donner et expliciter les prototypes des méthodes ainsi modifiées ou ajoutées, *i.e.* donnez leur nom, leurs arguments, ainsi que la sémantique des arguments ajoutés, enlevés ou modifiés.

**Question 10.** Est-il nécessaire de modifier le code des méthodes de `Tournee`? Si oui, lesquelles (sans donner le code ainsi modifié)?

Certaines informations concernant les véhicules (la plaque d'immatriculation par exemple) sont utiles pour l'exécution des tournées, mais ne sont pas utile dans les algorithmes précédents. Pour que l'équipe de développement en charge de programmer `Tournee` et celle en charge de programmer `Vehicule` travaillent efficacement, on définit une classe `Transport` qui se limite à garantir l'accès à la capacité. Les informations plus précises sur chaque véhicule resteront quand à elles dans `Vehicule`.

**Question 11.** En se limitant à la capacité (`capacite`) et au numéro d'immatriculation (`immatriculation`), proposer les classes `Transport` et `Vehicule` en vous limitant à leur définition (ligne commençant par `class`) et à leur constructeur.

## Partie II. Prise en compte du réseau

Lors de la tournée elle-même, les véhicules sont en communication constante avec le service central de coordination de la flotte. Cette communication réseau utilise des applications présentes dans chaque véhicule qui envoient régulièrement des données (vitesse, position...) au service central. Les communications entre les véhicules et le serveur peuvent être compliquées : zones de non couverture, tunnels, conditions météo...

**Question 12.** On suppose le cas où un véhicule se trouve bloqué dans un tunnel empêchant la communication. Que se passe-t-il (couche transport du point de vue de l'application embarquée) si on suppose que le véhicule veut envoyer une information au serveur en tentant d'établir une communication basée sur TCP ?

Un blocage long dans une zone non connectée peut être lié à un embouteillage. Dans ce cas-là, la densité de véhicules peut permettre de transmettre des informations en utilisant un principe de communication inter-véhicules. On suppose pour les questions suivantes que tous les véhicules sont équipés d'un système de communication local sans fil compatible.

On teste un premier protocole de routage. Quand le véhicule veut envoyer une donnée au serveur, il l'envoie (**broadcast**) à tous les véhicules autour de lui. Ces derniers font de même sauf s'ils sont capables de contacter l'extérieur (véhicule de *bordure*). Dans ce cas-là, ils envoient le message au serveur.

**Question 13.** Ce protocole a un défaut fondamental, quel est-il ? Comment le corriger ?

Une fois ce problème corrigé, on se pose la question de la réception au niveau du serveur. On suppose avoir utilisé le protocole TCP entre les véhicules de *bordure* et le serveur. Si plusieurs véhicules sont en bordure, plusieurs messages applicatifs vont donc être envoyés vers le serveur pour la même donnée.

**Question 14.** Est-il nécessaire, au niveau de l'application recevant les données sur le serveur, de gérer ces multiples messages contenant des données identiques, ou est-ce déjà géré au niveau TCP ? Justifier votre réponse.

Lorsque la densité de véhicules est forte (comme lors d'un embouteillage), la distance entre les véhicules est faible par rapport à la portée de la communication sans fil. Lors d'un embouteillage on peut avoir plusieurs dizaines de véhicules à portée de communication directe. Dans un premier temps, on considère qu'un seul véhicule tente d'envoyer un message vers le serveur extérieur.

**Question 15.** Dans le cadre où un seul véhicule veut envoyer un message, tous les messages intermédiaires entre le véhicule initial et les véhicules de *bordure* sont-ils nécessaires ? Justifier.

**Question 16.** Lors d'une communication sans fil, il est possible d'obtenir au niveau applicatif la qualité du signal portant un message. Plus la distance entre l'émetteur et le récepteur est grande, plus le message est faible. Proposez un algorithme de retransmission de message permettant d'optimiser le nombre de messages utilisés.

**Question 17.** Discuter l'impact de cet algorithme en terme de nombre de messages et de latence. On pourra fixer par exemple le nombre de véhicule à distance de communication sans fil comme étant une constante.

Après des tests dans les situations d'embouteillages dans des tunnels, on réalise que plusieurs véhicules peuvent vouloir transmettre de l'information vers le serveur extérieur. Le nombre de messages émis pour être retransmis entre les véhicules est proportionnel au nombre de véhicules.

Le premier problème est lié à la nature des communications sans-fil. Si deux véhicules tentent de communiquer en même temps, les deux messages vont être brouillés.

**Question 18.** À quel couche du modèle OSI se pose ce premier problème ? Proposer une solution pour régler ce problème.

Le second problème est lié au nombre de messages de contenus différents.

**Question 19.** Tous les messages sont retransmis de proche en proche vers la bordure. Comment réduire le nombre de messages lorsqu'un grand nombre de véhicules tentent chacun d'envoyer indépendamment des données vers un serveur extérieur ?

### Partie III. Archivage de la position, de la vitesse

Pour respecter la loi, les véhicules professionnels doivent conserver des informations légales, principalement la vitesse, dans un tachygraphe. On décide de mettre en place un tel tachygraphe numérique. On profite de sa mise en place pour ajouter une fonctionnalité. Les données sont envoyées au serveur en temps réel quand le véhicule est joignable, mais les données sont stockées localement dans le cas contraire. Pour garantir les aspects réglementaires (non modification des données par le personnel de l'entreprise), on utilise une machine virtuelle qui peut être déplacée entre le serveur et le véhicule lorsque la qualité de la communication entre ces deux éléments se dégrade. Chaque véhicule est suivi par une application tachygraphe qui se trouve dans une machine virtuelle spécifique à ce véhicule.

On considère que le serveur et l'ordinateur de bord sont d'architectures différentes (par exemple d'architecture x86 pour les serveurs et de type ARM pour les véhicules).

**Question 20.** Est-il possible de migrer l'application tachygraphe entre le serveur et le véhicule ? Si oui, expliciter l'impact sur les performances.

Cette application de tachygraphe permet de centraliser l'information pour plusieurs usages : archivage légal, mais aussi information du centre de coordination des véhicules, affichage pour le conducteur... Comme l'accès et la modification des données est complexe, elle peut prendre du temps. Aussi, il faut gérer en interne la synchronisation d'accès aux données pour plusieurs processus fournissant les services pour ces différents usages.

On suppose avoir deux fonctions `lecture(...)` et `écriture(...)`, déjà implémentées, permettant respectivement de lire et d'écrire ces données qui peuvent prendre du temps. On veut avoir les propriétés suivantes :

1. Plusieurs lectures peuvent avoir lieu en même temps,
2. Les écritures sont exclusives (sans lecture ni écriture en même temps),
3. En cas de choix, priorité aux lecteurs.

Un premier test est le suivant :

Listing 2 – code de la classe `Acces`.

```
import threading

class Acces:
 def __init__(self):
 self.mutex = threading.Semaphore(value = 1)

 def debut_lecture():
 self.mutex.acquire() # P
 def fin_lecture():
 self.mutex.release() # V

 def debut_ecriture():
 self.mutex.acquire()
 def fin_ecriture():
 self.mutex.release()
```

Avec une utilisation type (en considérant la variable `synchro` de classe `Acces`) :

Listing 3 – code d'utilisation de la classe `Acces`.

```
Pour une lecture
...
synchro.debut_lecture()
valeur = lecture(...)
synchro.fin_lecture()
...

Pour une écriture
...
synchro.debut_ecriture()
écriture(...)
synchro.fin_ecriture()
...
```

**Question 21.** Pour chacune des trois propriétés, expliquer pourquoi elle est respectée ou pourquoi elle ne l'est pas.

**Question 22.** Proposer (en Python) une classe `Acces_v2` permettant d'implémenter l'ensemble des propriétés souhaitées en conservant les mêmes méthodes.

**Question 23.** Donner un exemple de famine dans le cadre des propriétés souhaitées.

## Partie IV. Gestion de tournées

Voici en figure 2 le schéma relationnel d'une base de données qui permet à l'entreprise de faire la gestion de ses tournées. La spécification des domaines est donnée dans le listing 4.

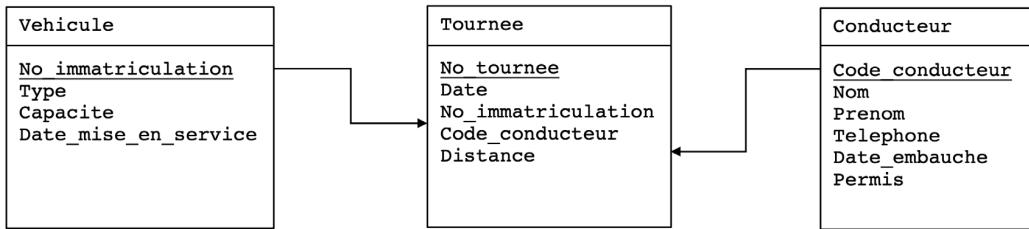


FIGURE 2 – Schéma relationnel de la base de données **Gestion de tournées**.

Listing 4 – Spécification de la base de données.

```

VEHICULE(*No_immatriculation(char(10)), Type(text), Capacite(float),
Date_mise_en_service(date))

TOURNEE(*No_tournee(int), Date(date), No_immatriculation(char(10)),
Code_conducteur(char(8)), Distance(float))

CONDUCTEUR(*Code_conducteur(char(8)), Nom(char(30)),
Prenom(char(40)), Telephone(texte), Date_embauche(date),
Permis(texte))

```

**Question 24.** Écrire une requête en SQL qui permet d'afficher combien de conducteurs ont un permis C1E (élément **Permis** dans la table précédente).

**Question 25.** Écrire une requête en SQL qui affiche la distance totale parcourue par John Doe en novembre 2021. On rappelle que les dates sont exprimées par défaut sous la forme YYYY-MM-DD dans les requêtes.

**Question 26.** Écrire une requête en SQL qui permet de calculer la capacité cumulée des véhicules selon le permis de leur conducteur.

Les différents véhicules nécessitent des permis de conduire différents. En effet, selon leur poids total autorisé en charge (PTAC), ou selon la présence éventuelle d'un attelage de remorque, il faut avoir le permis correspondant. On trouve un résumé des différents permis de transport de marchandises dans la table 1 et la figure 3. Le permis C1 permet ainsi de conduire un véhicule isolé dont le PTAC est compris entre 3,5t et 7,5t, tandis que le permis C1E permet de conduire un ensemble de véhicules dont le tracteur appartient à la catégorie C1 et dont le poids total ne dépasse pas 12t.

Dans sa forme actuelle, la base de données ne permet pas d'éviter les erreurs d'affectation. Il est pour l'instant tout à fait possible d'affecter un conducteur de moins de 21 ans à un véhicule isolé de plus de 7,5t de PTAC. Pour empêcher ce type d'erreur, l'entreprise souhaite améliorer la base de données en modifiant le schéma de manière à pouvoir effectuer une vérification

| Permis | PTAC             | Véhicule isolé | Remorque | Âge minimum | Équivalence |
|--------|------------------|----------------|----------|-------------|-------------|
| C1     | 3,5 à 7,5 t      | oui            | < 750 kg | 18          | -           |
| C1E    | > 3,5 t, max 12t | non            | > 750 kg | 18          | -           |
| C      | > 3,5 t          | oui            | < 750 kg | 21          | C1          |
| CE     | > 3,5 t          | non            | > 750 kg | 21          | C1E         |

TABLE 1 – Les permis poids lourd pour le transport des marchandises.

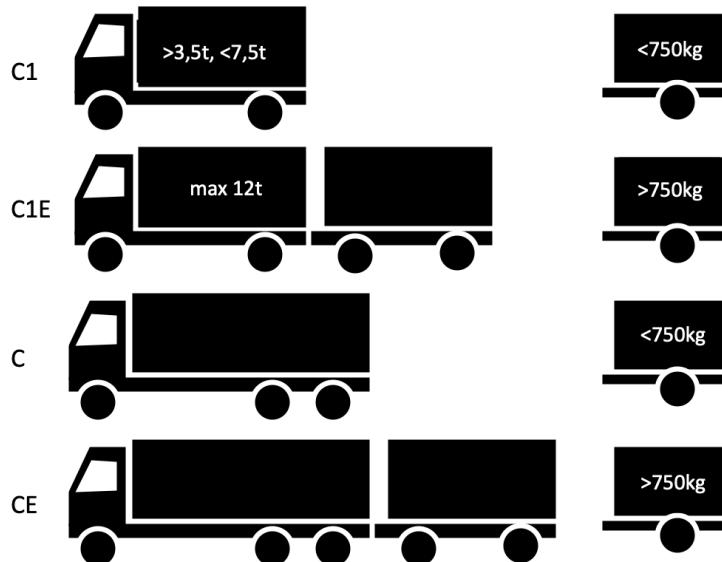


FIGURE 3 – Les permis poids lourd pour le transport de marchandises.

automatique de compatibilité d'affectation entre un véhicule et un conducteur. Pour ceci, il est nécessaire de pouvoir stocker les spécifications des différents permis dans la base et il a été décidé d'ajouter une table **Permis** au schéma relationnel.

**Question 27.** Modéliser la nouvelle table **Permis** sous une forme similaire au listing 4 en précisant le domaine pour chaque attribut. (Pas de requête en SQL.)

**Question 28.** Une analyse plus approfondie du schéma relationnel montre que le type d'un véhicule est stocké sous la forme d'une simple chaîne de caractères. Discuter les limites de ce choix de modélisation.

L'entreprise a fait un inventaire de sa flotte de véhicules et il s'est avéré que les véhicules peuvent être catégorisés selon les types suivants :

| Type              | longueur | largeur | hauteur | PTAC  | capacité palettes 80x120 |
|-------------------|----------|---------|---------|-------|--------------------------|
| Semi-remorque     | 16,5 m   | 2,55 m  | 4 m     | 26 t  | 33                       |
| Porteur 19 t      | 10,70 m  | 2,55 m  | 4 m     | 19 t  | 20                       |
| Porteur 12 t      | 9,5 m    | 2,55 m  | 4 m     | 12 t  | 14                       |
| 20 m <sup>3</sup> | 7 m      | 2,5 m   | 3,10 m  | 3,5 t | 8                        |

**Question 29.** Proposez une extension du schéma relationnel qui permet la vérification automatique du permis de conduire lors d'une affectation d'un conducteur à un véhicule. Présenter votre proposition sous une forme similaire à la figure 2 et au listing 4. Justifiez vos choix.

## Partie V. Utilisation des éléments mobiles

Dans cette partie, nous allons mettre en place une application web qui permettra à un chauffeur de visualiser ses retards dans sa tournée, ainsi que de valider ses livraisons.

Un aperçu de l'application est donné dans la figure 4. Le chauffeur verra la liste des clients à livrer avec l'heure de livraison. Le code couleur suivant permettra de visualiser directement l'état de la livraison :

**blanc** Le créneau de la livraison est dans le futur.

**gris clair** Le créneau de la livraison est dépassé d'une heure au plus.

**gris foncé** Le créneau de livraison est largement dépassé (plus d'une heure).

**noir** La livraison a été effectuée.

| Client    | Heure de livraison |
|-----------|--------------------|
| Silva     |                    |
| Fergusson | 12                 |
| Milan     | 14                 |
| Berger    | 17                 |

FIGURE 4 – Application web : Visualisation de tournée, heure actuelle : 14h13.

Le bouton **Valider la livraison** permet de changer le statut de la livraison et l'affichage du créneau horaire passe en noir.

Ainsi, on comprend dans l'exemple de la figure 4 que la livraison de Silva a été validée. La livraison pour Fergusson est très en retard tandis que la livraison pour Milan est un peu en retard. La livraison pour Berger peut être effectuée sans retard.

**Question 30.** En utilisant HTML et CSS uniquement, programmer la partie statique de la page, c'est-à-dire l'affichage du tableau sans la logique des changements de couleur en supposant qu'il n'y a que les quatre clients visibles sur la figure 4 à afficher.

Listing 5 – Début du code de la page statique.

```
<!DOCTYPE html>
<html>
 <head>
 <style>
 .ontime {
 background-color: white;
 }
 .hurry {
 background-color: lightgrey;
 }
 .late {
 background-color: dimgray;
 }
 .done {
 background-color: black;
 }
 </style>
 </head>
 <body>
 <!-- TODO -->
 </body>
</html>
```

**Question 31.** En utilisant JavaScript, programmer l'affichage en couleurs des créneaux de livraison par rapport à l'heure actuelle. La première ligne du code est déjà écrite (voir listing 6).

Listing 6 – Début du code de la coloration automatique des créneaux de livraison.

```
<script>
let hours = document.querySelectorAll('.heure')

const d = new Date()
let heureActuelle = d.getHours()

//TODO
</script>
```

**Question 32.** En utilisant JavaScript, programmer la validation d'une livraison : dès que le bouton **Valider la livraison** a été actionné, le créneau de livraison s'affiche en noir et le bouton disparaît. La première ligne du code est déjà écrite (voir listing 7).

Listing 7 – Début du code de la fonctionnalité du bouton.

```
<script>
let rows = document.querySelectorAll(".row")
//TODO
</script>
```

Il arrive aux conducteurs de vouloir changer le créneau de livraison pour un client dans l'application. Dans ce cas, le conducteur saisit le nom du client ainsi que le nouveau créneau dans un formulaire avec deux champs dans l'application. L'application envoie une requête HTTP vers un serveur web qui s'occupe de vérifier si le changement de créneau est possible. Plus précisément, l'application envoie une requête GET vers **/traitement** avec les paramètres **client** et **nHeure** avec les données fournies par le conducteur via le formulaire web. Si on souhaite changer l'heure du client **Fergusson** à 18h et en supposant que l'adresse du serveur est **http://www.appli-tournee.fr**, alors une telle requête serait : GET //www.appli-tournee.fr/traitement?client=Fergusson&nHeure=18 . Si la réponse du serveur est positive, alors l'affichage du créneau horaire du client doit changer sur le site. Par contre, en cas de réponse négative, une alerte sera donnée au conducteur signalant que le changement n'est pas possible. On rappelle que la fonction JavaScript "alert(message)" permet de lancer une alerte à l'utilisateur.

**Question 33.** En utilisant des fonctions asynchrones de JavaScript, programmer le script qui permet de changer le créneau horaire d'un client. Le listing 8 donne la première ligne du code.

Listing 8 – Début du code pour mettre à jour un créneau de livraison.

```
<script>
let buttonMAJ = document.getElementById("maj")
//TODO
</script>
```

## Rappels

- `element.innerHTML` : la propriété `Element.innerHTML` de `Element` récupère ou définit la syntaxe HTML décrivant les descendants de l'élément.
- `EventTarget.addEventListener()` : la méthode `addEventListener()` d'un `EventTarget` attache une fonction à appeler chaque fois que l'événement spécifié est envoyé à la cible.
- `document.querySelector` : la méthode `querySelector()` de l'interface `Document` renvoie le premier `Element` dans le document correspondant au sélecteur - ou groupe de sélecteurs - spécifié(s), ou `null` si aucune correspondance n'est trouvée.

---

## Fondements de l'informatique

---

**Préliminaires généraux** Les questions de programmation doivent être traitées en langage *OCaml*. On pourra utiliser toutes les fonctions des modules `Array` et `List`, ainsi que les fonctions de la bibliothèque standard (celles qui s'écrivent sans nom de module, comme `max`, `incr` ainsi que les opérateurs comme `@`). L'utilisation d'autres modules est interdite. Si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction de tester si les hypothèses sont bien satisfaites. On ne cherchera pas à gérer les exceptions. Lorsque les choix d'implémentation ne découlent pas directement des spécifications de l'énoncé, il est vivement conseillé de les expliquer.

Il est attendu des candidates et des candidats des réponses construites. Ils seront également évalués sur la précision, le soin et la clarté de la rédaction. Les questions avec une étoile (c'est-à-dire de la forme **Question\***) sont a priori plus difficiles ou nécessitent plus d'attention. Il est autorisé d'admettre le résultat d'une question (en particulier celles avec une étoile) pour traiter les questions suivantes.

Ce sujet s'intéresse à différents modèles de neurones et réseaux de neurones formels, à leur puissance en tant que modèles de calculs, ainsi qu'à la complexité de leur apprentissage exact.

Fixons une fonction  $\mathcal{A} : \mathbb{R} \rightarrow \mathbb{R}$  ( $\mathbb{R}$  désigne l'ensemble des réels). Jusqu'à la fin de la partie IV, la fonction  $\mathcal{A}$  (qui est appelée la *fonction d'activation*) sera la *fonction de Heaviside*, c'est-à-dire la fonction  $\mathcal{H}(x)$  qui vaut 1 pour  $x \geq 0$ , 0 sinon.

Un *neurone à n entrées* est décrit par des paramètres  $w_1, w_2, \dots, w_n$  et  $h$ , qui sont des réels. Chacun des  $w_i$  est appelé un *poids*, et  $h$  est appelé le *seuil* du neurone. Un tel neurone calcule une fonction de  $\mathbb{R}^n$  dans  $\mathbb{R}$  de la façon suivante : sur les entrées  $x_1, x_2, \dots, x_n$ , il prend la valeur  $\mathcal{A}(w_1x_1 + w_2x_2 + \dots + w_nx_n - h)$ . Cette valeur s'appelle *sa valeur d'activation*.

**Objectifs généraux du sujet** Dans la partie I, on étudie les neurones pris isolément. De la même manière qu'en informatique en partant des portes logiques élémentaires comme le *ET*, le *OU* et la négation *NOT*, on s'intéresse classiquement aux circuits booléens construits à partir de ces portes comme modèles de calculs, on s'intéressera à partir de la partie II aux circuits (“réseaux”) de neurones construits à partir de neurones.

Notez que les poids et les seuils sont a priori des réels quelconques. On parlera de neurone ou de réseau de neurones à *poids unitaires* lorsque tous les poids<sup>1</sup> prennent leur valeur dans  $\{-1, 0, 1\}$ .

---

1. Dans un réseau à poids unitaire, on n'impose rien sur les seuils : cela peut très bien être des réels quelconques.



## Partie I. Le cas d'un neurone à seuil

On note  $\mathbf{B} = \{0, 1\}$  : l'intuition dans tout le sujet est que le réel 0 représente la valeur logique *faux*, et que le réel 1 représente la valeur logique *vrai*. On répète que dans cette partie, comme jusqu'à la fin de la partie IV, la fonction d'activation  $\mathcal{A}$  est la fonction de Heaviside : on parle de *neurone à seuil* dans ce cas.

### 1 Fonction booléennes linéaires à seuil

On dit qu'une fonction  $F : \mathbf{B}^n \rightarrow \mathbf{B}$  est une *fonction booléenne linéaire à seuil* (et plus généralement qu'une fonction  $F : S \subseteq \mathbb{R}^n \rightarrow \mathbf{B}$  est une *fonction linéaire à seuil*) si elle correspond à la fonction calculée par un neurone à seuil : dit autrement, il existe des réels  $w_1, \dots, w_n$  et  $h$  tels que, pour tout  $\mathbf{x} = (x_1, \dots, x_n)$  dans  $\mathbf{B}^n$  (ou plus généralement dans  $S$ ),  $F(\mathbf{x}) = 1$  si et seulement si  $\sum_{i=1}^n w_i x_i \geq h$ .

On appelle  $(w_1, \dots, w_n, h)$  une *représentation de  $F$* . On dit que la représentation est à poids unitaires lorsque  $w_1, w_2, \dots, w_n$  restent dans  $\{-1, 0, 1\}$ . Bien entendu, une même fonction booléenne linéaire à seuil peut avoir plusieurs représentations.

Par exemple, la fonction *ET logique* à  $n$ -arguments  $\text{AND}_n(x_1, \dots, x_n)$ , qui vaut 1 si et seulement si chacun des  $x_i$  vaut 1, est une fonction booléenne linéaire à seuil (et même à poids unitaires). En effet, elle admet la représentation  $(\underbrace{1, \dots, 1}_n, n)$ , car  $\text{AND}_n(x_1, \dots, x_n) = 1$  si et seulement si  $1x_1 + 1x_2 + \dots + 1x_n \geq n$ .

**Question 1.** Montrer que le *OU logique* (à  $n$ -arguments)  $\text{OR}_n$ , la *NEGATION logique*  $\text{NOT}_1$  (à un argument) sont également des fonctions booléennes linéaires à seuil. Montrer que c'est également le cas de la fonction  $\text{MAJORITY}_n : \mathbf{B}^n \rightarrow \mathbf{B}$  définie comme la fonction qui vaut 1 si et seulement si au moins  $n/2$  de ses entrées valent 1.

**Question 2.** Montrer que le *OU EXCLUSIF* à deux arguments (noté  $\text{XOR}_2$  ou encore  $\oplus$ ) n'est pas une fonction booléenne linéaire à seuil.

**Question 3.** En déduire que pour tout  $n \geq 2$ , la fonction parité à  $n$  arguments  $\text{PARITY}_n(x_1, \dots, x_n)$  (qui vaut 1 si et seulement si un nombre pair de ses arguments valent 1) n'est pas une fonction booléenne linéaire à seuil.

### 2 Cas de domaines bornées

Lorsque  $\delta$  est un nombre réel strictement positif, une représentation  $(w_1, \dots, w_n, h)$  est dite  $\delta$ -séparable sur un ensemble  $S \subseteq \mathbb{R}^n$  si pour tout  $x_1, \dots, x_n \in S$

$$\sum_{i=1}^n w_i x_i < h \text{ si et seulement si } \sum_{i=1}^n w_i x_i \leq h - \delta.$$

Une fonction linéaire à seuil  $F$  est dite séparable sur un ensemble  $S$  s'il existe  $\delta > 0$  tel que  $F$  admette une représentation  $\delta$ -séparable sur  $S$ .

**Question 4.** Montrer que toute fonction linéaire à seuil sur un ensemble fini est séparable.

La *masse* d'une représentation  $(w_1, \dots, w_n, h)$  est définie comme  $\max \{|w_i| \mid 1 \leq i \leq n\}$ , c'est-à-dire le maximum des valeurs absolues de ses poids.

**Question 5.** Montrer que pour tout  $\delta > 0$ ,  $S \subseteq \mathbb{R}^n$ , et toute fonction linéaire à seuil  $F$ , si  $F$  possède une représentation qui est  $\lambda$ -séparable de masse  $w$ , alors  $F$  possède également une représentation  $\delta$ -séparable de masse  $w\delta/\lambda$ .

*On en déduit que pour tout  $\delta > 0$ , toute fonction linéaire à seuil  $F$  séparable possède une représentation  $\delta$ -séparable.*

Une fonction  $F$  est une fonction linéaire à *seuil nul* si c'est une fonction linéaire à seuil et qu'elle admet une représentation  $(w_1, \dots, w_n, 0)$ , c'est-à-dire dont le seuil vaut 0.

**Question 6.** Montrer que pour toute fonction linéaire à seuil  $F : \mathbb{R}^n \rightarrow \mathbf{B}$ , on peut construire une fonction linéaire à seuil nul  $G : \mathbb{R}^{n+1} \rightarrow \mathbf{B}$  (et donc avec une dimension (entrée) supplémentaire) telle que pour tout  $x_1, \dots, x_n \in \mathbf{B}$ ,  $F(x_1, \dots, x_n) = G(x_1, \dots, x_n, 1)$ .

*Ne serait-ce que pour des raisons de représentation sur ordinateur, il est parfois intéressant de se ramener au cas où tous les poids sont entiers : on dit qu'une représentation  $(w_1, \dots, w_n, h)$  est entière si chacun des  $w_i$  et  $h$  sont des éléments de  $\mathbb{Z}$ , l'ensemble des entiers relatifs.*

**Question 7.** Soit  $\delta > 0$ . On considère une fonction linéaire à seuil sur un ensemble borné  $S \subseteq [0, 1]^n$  avec une représentation  $n$ -séparable, de la forme  $(w_1, \dots, w_n, h)$  avec  $w_1, \dots, w_n \geq 0$  de masse  $nw/\delta$ . Montrer que la fonction possède également une représentation entière de même dimension de masse au plus  $nw/\delta$ .

**Question 8.** Montrer que toute fonction linéaire à seuil (sans restriction sur le signe des poids) sur un ensemble borné  $S \subseteq [0, 1]^n$  avec une représentation  $\delta$ -séparable de masse  $w$ , possède une représentation entière de masse au plus  $nw/\delta$ .

*On déduit donc (de la question 8 et de la remarque qui suit la question 5) que toute fonction linéaire à seuil séparable sur un ensemble borné possède une représentation entière.*

### 3 Apprentissage d'un neurone à seuil

Soit  $D = [0, 1]$  ou  $D = [-1, 1]$ . On considère dans cette partie *le problème de l'apprentissage exact des fonctions linéaires à seuil sur le domaine  $D$*  : c'est-à-dire, étant donné un ensemble fini de  $N$  couples, c'est-à-dire  $(\mathbf{x}_i, y_i)_{1 \leq i \leq N}$ , avec  $\mathbf{x}_i \in D^n$ ,  $y_i \in \mathbf{B}$ , on cherche à déterminer s'il existe une fonction linéaire à seuil  $F$  telle que pour tout  $1 \leq i \leq N$ ,  $F(\mathbf{x}_i) = y_i$ . Si une telle fonction  $F$  existe, on dit que le problème d'apprentissage *admet une solution* (qui est la fonction  $F$ ). On appelle  $n$  la dimension.

L'ensemble  $X = \{x_1, x_2, \dots, x_N\}$  est appelé *l'ensemble des exemples*. Lorsque  $y_i = 1$  on dit que  $\mathbf{x}_i$  est un exemple *positif*. Lorsque  $y_i = 0$  on dit que  $\mathbf{x}_i$  est un exemple *négatif*.

On cherche à résoudre ce problème pour  $D = [0, 1]$  dans le cas le plus général. Pour cela, on va chercher à le réduire à des variantes plus faciles à traiter. On parle de problème d'apprentissage :

- à *seuil nul* lorsqu'on cherche à déterminer s'il existe  $F$ , une fonction linéaire à seuil comme ci-dessus, avec de plus  $F$  à seuil nul.
- à *exemples négatifs* lorsque l'on a  $y_i = 0$  pour tous les  $1 \leq i \leq N$ .

**Question 9.** Montrer que le problème de l'apprentissage exact des fonctions linéaires à seuil sur le domaine  $D$  en dimension  $n$  se réduit au problème de l'apprentissage exact des fonctions linéaires à seuil nul sur le domaine  $D$  en dimension  $n + 1$ .

**Question 10.** Montrer que le problème de l'apprentissage exact des fonctions linéaires à seuil sur le domaine  $[0, 1]$  en dimension  $n$  se réduit au problème de l'apprentissage exact des fonctions linéaires à seuil nul à exemples négatifs sur le domaine  $[-1, 1]$  en dimension  $n + 1$ .

*Autrement dit, il suffit de savoir résoudre le problème de l'apprentissage exact dans le cas où  $F$  est à seuil nul, et où l'on n'a que des exemples négatifs, et  $D = [-1, 1]$ .*

On va s'intéresser en fait au problème de calcul associé : étant donné  $X$  avec des exemples négatifs,  $D = [-1, 1]$ , lorsqu'il existe une fonction linéaire à seuil nul  $F$  solution, produire une représentation d'une telle fonction  $F$ . Introduisons pour cela l'algorithme suivant.

```

1 Procedure apprentissage(n, X)
2 for $i \in \{1, 2, \dots, n\}$ do
3 $w_i := 0$
4 repeat
5 $fin := 1$;
6 for $\mathbf{x} = (x_1, \dots, x_n) \in X$ do
7 if $\sum_{i=1}^n w_i x_i \geq 0$ then
8 $fin := 0$;
9 for $i \in \{1, 2, \dots, n\}$ do
10 $w_i := w_i - x_i$
11 until $fin = 1$;
12 return $(w_1, w_2, \dots, w_n, 0)$
```

Chaque itération de la boucle *repeat* est appelé *une époque*. Lorsque la condition  $\sum_{i=1}^n w_i x_i \geq 0$  est satisfaite à la ligne 7, on dit qu'une *erreur à été commise*.

**Question 11.** On fixe la déclaration de type

---

```
type vecteur = float array
```

---

Proposer une implémentation `apprentissage: int -> vecteur array -> vecteur` de cet algorithme en OCaml. On rappelle qu'en OCaml, les opérations arithmétiques sur les float sont désignées par `+, -, *, /`.

*On veut montrer que l'algorithme termine avec une représentation valide si et seulement si le problème d'apprentissage exact admet une solution à seuil nul.*

**Question 12.** Montrer que si l'algorithme termine, alors le problème d'apprentissage exact à seuil nul défini par l'ensemble des exemples négatifs  $X$  admet une solution, dont une représentation est donnée par  $(w_1, \dots, w_n, 0)$ .

*Il reste à prouver l'implication réciproque.*

**Question 13.** Soit  $X \subseteq [-1, 1]^n$  un ensemble fini de  $N$  exemples négatifs, et  $F$  une fonction linéaire à seuil nul solution. Démontrer que si  $F$  possède une représentation  $(v_1, \dots, v_n, 0)$  qui est  $n$ -séparable alors l'algorithme termine et est correct.

On pourra considérer  $d(\mathbf{w}, \mathbf{v}) = \sum_{i=1}^n (w_i - v_i)^2$  où  $\mathbf{w} = (w_1, \dots, w_n)$  décrit les poids à chaque époque, et on rappelle que  $X$  est constitué d'exemples négatifs.

*Dans le cas général, on considère  $F$  une fonction linéaire à seuil et un ensemble fini d'exemples. Par la question 4 et la partie 2, elle possède une représentation entière.*

**Question\* 14.** Soit  $X \subseteq [-1, 1]^n$  un ensemble fini de  $N$  exemples négatifs. On suppose que le problème de l'apprentissage exact à exemples négatifs associé à  $X$  admet une solution 1-séparable à coefficients entiers et de masse inférieure ou égale à  $w$ . Montrer que l'algorithme commet alors au plus  $\mathcal{O}(n^2w^2)$  erreurs et termine en temps au plus  $\mathcal{O}(n^2(n + N)w^2)$ .

## Partie II. D'un neurone à seuil à un réseau de neurones

On introduit maintenant les circuits de neurones, également appelés “réseaux de neurones”.

Fixons un ensemble  $K$ . Fixons un ensemble  $\mathcal{G}$  de portes élémentaires sur  $K$  : chaque porte élémentaire  $g$  de  $\mathcal{G}$  est une fonction  $g : K^{n_g} \rightarrow K$  pour un certain entier  $n_g$ , que l'on appelle son arité. Par exemple, on peut prendre  $K = \mathbf{B}$ , et  $\mathcal{G} = \mathcal{G}_{\text{bool}} = \{\text{AND}_2, \text{OR}_2, \text{NOT}_1, 0, 1\}$ , où 0, et 1 sont d'arité 0, et l'arité des autres portes élémentaires est indiquée par l'indice dans son nom.

On définit la notion de circuits sur un ensemble de portes  $\mathcal{G}$ , en généralisant la notion usuelle de circuit booléen (qui correspond au cas  $\mathcal{G} = \mathcal{G}_{\text{bool}}$  sur  $K = \mathbf{B}$ ) à un ensemble de portes élémentaires  $\mathcal{G}$  plus général.

**Définition 1.** Un circuit sur  $\mathcal{G}$  est donné par un graphe fini  $(V \cup X, E)$  orienté sans cycle. Les sommets de  $X$  sont appelés des entrées, et n'ont pas d'arc entrant. Tout sommet de  $V$  est appelé une porte, et est étiqueté par un élément  $g$  de  $\mathcal{G}$  : si cet élément  $g$  est d'arité  $n_g$ , alors ce sommet possède exactement  $n_g$  arcs entrants. Les sommets de  $V$  qui n'ont pas d'arc sortant sont appelés des sorties.

Notons que les sommets de  $X$ , et de  $V$  qui ne sont pas des sorties, peuvent avoir plusieurs arcs sortants. S'il y a  $n$  entrées et  $m$  sorties, on fixe une numérotation de celles-ci de 1 à  $n$  et de 1 à  $m$  respectivement.

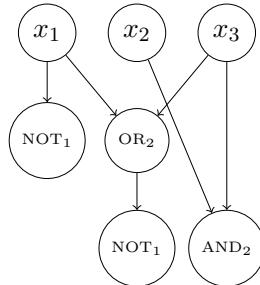
La *taille* d'un tel circuit est définie comme le nombre de sommets dans  $V$ , et donc son nombre de portes.

Étant donnée une valeur pour chacune des entrées, la valeur de chacune des sorties s'obtient en propageant la valeur des entrées vers les sorties en évaluant la valeur de chaque porte selon la fonction associée. On dit que le circuit calcule la fonction  $f : S \subseteq K^n \rightarrow K^m$ , si pour tout  $(x_1, \dots, x_n) \in S$ , si on note  $(y_1, \dots, y_m) = f(x_1, \dots, x_n) \in K^m$ , alors si l'on fixe l'entrée numéroté  $i$  à  $x_i$ , pour  $1 \leq i \leq n$ , la valeur de la sortie numéroté  $j$  est  $y_j$ , pour  $1 \leq j \leq m$ .

Par exemple, un circuit booléen avec  $n$  entrées et  $m$  sorties calcule une fonction de  $\mathbf{B}^n$  dans  $\mathbf{B}^m$ .

La fonction profondeur *prof* qui à un sommet de  $V \cup X$  associe *sa profondeur* se définit de manière récursive de la façon suivante :  $\text{prof}(x) = 0$  pour chaque  $x \in X$ , et  $\text{prof}(v) = 1 + \max_{(s,v) \in E} \text{prof}(s)$ . La *profondeur d'un circuit* est la plus grande profondeur d'un de ses sommets. On appelle *couche*  $k$  l'ensemble des sommets de profondeur  $k$ .

Voici un circuit booléen sur  $\mathcal{G} = \mathcal{G}_{\text{bool}}$  à 3 entrées et 3 sorties, de taille 4. Si l'on numérote les sorties de gauche à droite, il calcule une certaine fonction  $f : \mathbf{B}^3 \rightarrow \mathbf{B}^3$ ; on a par exemple  $f(0, 1, 0) = (1, 1, 0)$ .



**Question 15.** Quelle est la profondeur de ce circuit ? Détails chacune de ses couches. Décrire complètement la fonction calculée par ce circuit.

## 4 Des circuits booléens aux réseaux de neurones à seuil

En prenant pour  $\mathcal{G}$  l'ensemble des neurones à seuil (un neurone à  $n$  entrées ayant l'arité  $n$ ) sur  $K = \mathbb{R}$ , on obtient un *circuit de neurones*, aussi appelé *un réseau de neurones*.

Cette construction se généralise au cas d'une fonction d'activation  $\mathcal{A}$  différente de la fonction de Heaviside : nous reviendrons sur ce point à la fin de la partie IV.

**Question 16.** On souhaite coder les réseaux de neurones en *OCaml*. Pour la commodité d'implémentation en *OCaml*, on utilise une représentation différente (bien qu'équivalente) de celle donnée plus haut pour un réseau de neurones. *Ce choix de représentation en machine est fixé uniquement pour cette question et ne sera plus utilisé dans le reste du sujet.* On voit un réseau de neurones comme un graphe, où chaque arc est étiqueté par un poids, et chaque sommet est étiqueté par son seuil. On souhaite coder le graphe selon le principe d'une liste d'adjacence : supposons que les sommets sont numérotés de 0 à  $n - 1$ . Le graphe est alors représenté sous la forme d'un tableau  $T$  de taille  $n$  tel que  $T.(i)$  contienne la liste des extrémités et poids des arcs sortants du sommet  $i$  ainsi que le seuil associé au sommet  $i$ .

On définit les types suivants :

```
type sommet = int
type seuil = float
type poids = float
type reseau_adjacence = ((sommet * poids) list * seuil) array
```

Proposer une implémentation en *OCaml* de la fonction `creer : int -> reseau_adjacence` qui permet de créer un réseau avec un nombre donné de sommets.

Même question pour `teste_arc : reseau_adjacence -> sommet -> sommet -> bool` qui permet de tester l'existence d'un arc entre deux sommets.

Même question pour la fonction

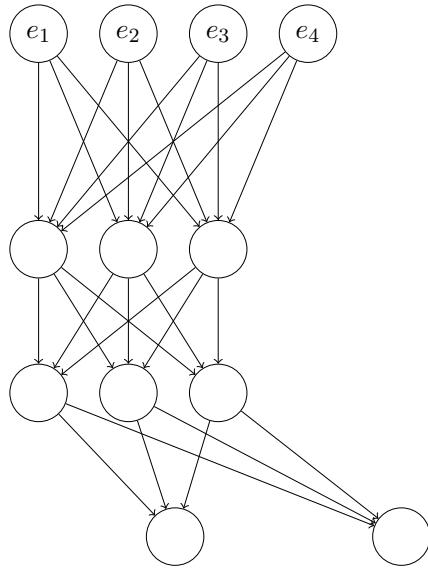
```
ajoute_arc : reseau_adjacence -> sommet -> sommet -> poids -> reseau_adjacence
```

qui ajoute un arc entre deux sommets, avec un poids donné.

**Question 17.** Soit  $C$  un circuit booléen de taille  $t$  et de profondeur  $d$  qui calcule la fonction  $f : \mathbf{B}^n \rightarrow \mathbf{B}^m$ . Montrer qu'il existe un réseau de neurones à seuil à poids unitaires de taille  $t$  et de profondeur  $d$  qui calcule la fonction  $f$ .

*On va voir dans la partie suivante qu'il est parfois possible de calculer certaines fonctions de façon beaucoup plus efficace en terme de taille avec des réseaux de neurones.*

On dit qu'un réseau de neurones est *bien formé* si chaque arc  $(u, v)$  de  $E$  est tel que si  $u$  appartient à la couche  $i$ , alors  $v$  appartient à la couche  $i + 1$ . Par exemple, un réseau qui aurait l'architecture (c'est-à-dire le graphe sous-jacent, les poids et seuils ne sont pas représentés) suivante est bien formé.

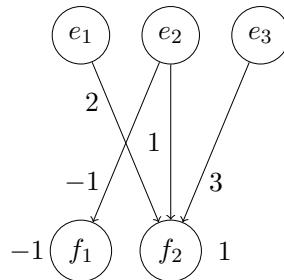


**Question 18.** Montrer qu'étant donné un réseau  $N$  de neurones à seuil, on peut construire un réseau de neurones  $N'$  à seuil de même profondeur bien formé équivalent : si  $N$  possède  $n$  entrées,  $N'$  également, et  $N$  et  $N'$  calculent la même fonction.

**Question 19.** On utilise dans cette question une représentation en machine des réseaux de neurones différente de celle de la question 16 ; ici, on représente un réseau de neurones bien formé par une liste de couples  $l$ . Si  $n$  est le nombre de neurones de la  $(i-1)$ -ème couche (ou, pour  $i=0$ , le nombre d'entrées du réseau) et  $m$  le nombre de neurones de la  $i$ -ème couche, le  $i$ -ème élément de  $l$  est composé d'une matrice  $W = (w_{jk})$  de nombres flottants à  $n$  lignes et  $m$  colonnes et d'un vecteur  $H = (h_k)$  de nombres flottants de taille  $m$ . Si la valeur du neurone  $j$  de la couche  $i-1$  (ou, pour  $i=0$ , l'entrée  $j$  du réseau) est une entrée du neurone  $k$  de la couche  $i$ ,  $w_{jk}$  est la valeur du poids correspondant ;  $w_{jk}$  vaut 0 sinon. Enfin, le nombre  $h_k$  est la valeur du seuil associé au neurone  $k$  de la couche  $i$ .

Ainsi, si l'exemple suivant représente deux couches consécutives (les nombres indiqués à côté des sommets étant les seuils associés, ceux à côté des arêtes les poids associés), la matrice associée

à la seconde couche sera alors  $\begin{pmatrix} 0 & 2 \\ -1 & 1 \\ 0 & 3 \end{pmatrix}$  et le vecteur  $\begin{pmatrix} -1 \\ 1 \end{pmatrix}$ .



On introduit les types suivants :

---

```
type vecteur = float array
type matrice = float array array
type reseau_matrice = (matrice * vecteur) list
```

---

Proposer le code *OCaml* d'une fonction `eval: reseau_matrice -> vecteur -> vecteur` qui renvoie le résultat de l'évaluation d'un tel réseau de neurones sur un vecteur donné (ledit vecteur décrivant la valeur des entrées du réseau de neurones).

## 5 À propos des circuits booléens

Dans les circuits booléens tels que définis plus haut, on a considéré que  $\mathcal{G}$ , l'ensemble des portes élémentaires était  $\mathcal{G} = \{\text{AND}_2, \text{OR}_2, \text{NOT}_1, 0, 1\}$ . On parlera de circuits booléen *généralisé* si l'on s'autorise  $\mathcal{G} = \{\text{AND}_n, \text{OR}_n, \text{NOT}_1, 0, 1 | n \in \mathbb{N}\}$  : autrement dit, les portes *ET logique* et *OU logique* peuvent avoir plus que deux arguments. On appellera *porte AND généralisée* (respectivement : *porte OR généralisée*) une telle porte  $\text{AND}_n$  (respectivement :  $\text{OR}_n$ ) pour un certain entier  $n$ .

Une formule en forme normale conjonctive (autrement dit sous la forme d'une conjonction de disjonctions de littéraux, un littéral étant une variable ou sa négation) peut s'écrire comme un circuit *NOT – OR – AND* : c'est-à-dire, comme un circuit généralisé avec :

- une unique sortie, étiquetée par une porte AND généralisée ;
- les entrées, ou les portes 0 et 1, qui ont leur(s) arc(s) sortant(s) qui vont soit vers une porte NOT<sub>1</sub>, soit vers l'une des portes OR généralisées ;
- les portes NOT<sub>1</sub> qui ont leur(s) arc(s) sortant(s) qui va vers une porte OR généralisée ;
- les portes OR généralisées ont leur arc sortant qui va vers la porte AND généralisée.

Symétriquement pour une formule en forme normale disjonctive qui peut s'écrire comme un circuit *NOT – AND – OR*, défini en inversant le rôle des OR et AND dans la description précédente.

On appellera circuit 2-alternant un circuit qui est soit *NOT – OR – AND*, soit *NOT – AND – OR*. On souhaite prouver que tout circuit 2-alternant qui calcule la fonction PARITY a une taille au moins  $2^{n-1} + 1$ .

**Question 20.** Montrer que dans tout circuit *NOT–AND–OR* qui calcule la fonction PARITY, pour chaque entrée  $b = (b_1, \dots, b_n) \in \mathbf{B}^n$  avec  $\text{PARITY}(b_1, \dots, b_n) = 1$ , il y au moins une porte AND généralisée, que l'on appellera  $A_b$ , dont la valeur de sortie sur l'entrée  $(b_1, \dots, b_n)$  est 1.

**Question\* 21.** Montrer que dans la question précédente, on peut supposer de plus qu'il existe dans le circuit un chemin de chacune des  $n$  entrées vers cette porte  $A_b$ .

**Question 22.** Montrer que tout circuit  $NOT - AND - OR$  qui calcule la fonction PARITY a une taille au moins  $2^{n-1} + 1$ . En déduire que tout circuit 2-alternant qui calcule la fonction PARITY a une taille au moins  $2^{n-1} + 1$ .

## 6 À propos des réseaux de neurones à seuil à poids unitaires

Une fonction  $f : \mathbf{B}^n \rightarrow \mathbf{B}^m$  est dite *symétrique* si sa valeur ne dépend pas de l'ordre de ses arguments.

**Question 23.** Soit  $m \leq n$  des entiers. Proposer un neurone à seuil à poids unitaires qui renvoie 1 sur les entrées  $x_1, \dots, x_n \in \mathbf{B}$  si et seulement si au moins  $m$  de ces entrées ont la valeur 1. Proposer par ailleurs un neurone à seuil à poids unitaires qui renvoie 1 sur les entrées  $x_1, \dots, x_n \in \mathbf{B}$  si et seulement si au plus  $m$  de ces entrées ont la valeur 1.

**Question 24.** Montrer que toute fonction symétrique  $f : \mathbf{B}^n \rightarrow \mathbf{B}$  peut être calculée par un réseau de neurones à seuil à poids unitaires de taille  $\mathcal{O}(n)$  et profondeur 2.

**Question 25.** En déduire que l'on peut calculer la fonction PARITY avec une profondeur 2 et une taille  $\mathcal{O}(n)$ .

*Rappelons que l'on a prouvé qu'on ne pouvait pas la calculer en profondeur 1. C'est donc un exemple de fonction qui ne se calcule pas avec une couche, mais peut se calculer avec deux, et se calcule avec beaucoup moins de portes en utilisant des réseaux de neurones qu'avec des circuits booléens.*

## Partie III. Complexité de l'apprentissage d'un circuit

*Nous étudions dans cette partie la complexité de déterminer les poids d'un réseau de neurones dont l'architecture est fixée.*

Appelons *architecture* un graphe comme dans la définition 1 (c'est-à-dire un circuit sans les étiquettes des sommets de  $V$ ), avec  $n$  entrées et  $m$  sorties. Une *tâche* est alors un élément de  $\mathbf{B}^n \times \mathbf{B}^m$ .

**Définition 2.** On dit que l'architecture est compatible avec la tâche  $(x_1, \dots, x_n, y_1, \dots, y_m)$  s'il existe des valeurs des poids dans  $\{-1, 0, 1\}$  et des valeurs entières des seuils pour les portes du graphe telles qu'on obtient la sortie  $(y_1, \dots, y_m)$  quand on évalue le réseau de neurones correspondant sur l'entrée  $(x_1, \dots, x_n)$ .

On s'intéresse au problème de décision **Apprentissage exact** : étant donnée une architecture et une liste finie de tâches, déterminer si l'architecture est compatible avec toutes les tâches de la liste au sens de la définition 2 (c'est-à-dire avec des poids unitaires et des seuils entiers).

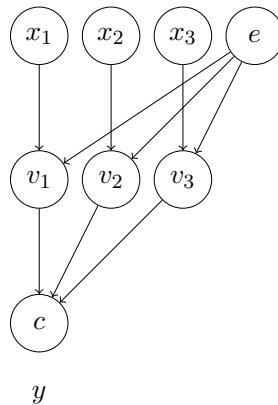
**Question 26.** Montrer que le problème est dans la classe NP.

On va chercher à prouver que le problème est NP-complet, en utilisant la NP-complétude de 3-SAT, que l'on admettra. On rappelle qu'il s'agit de déterminer la satisfiabilité d'une formule en forme normale conjonctive, avec 3 littéraux par clause (on rappelle qu'une *clause* est une disjonction de littéraux).

**Question 27.** On considère l'architecture à 4 entrées (à savoir  $x_1, x_2, x_3, e$ ) et 1 sortie (à savoir  $y$ ) représentée graphiquement ci-dessous.

On considère les 8 tâches que l'on obtient en faisant varier  $x_1, x_2, x_3 \in \mathbf{B}$ , et en ayant fixé  $e = 0$ , en prenant  $y$  qui vaut 1 sauf dans l'unique cas  $x_1 = 1, x_2 = 0, x_3 = 1$  où  $y$  vaut 0.

Prouver que l'architecture ci-dessous est compatible avec ces 8 tâches.



**Question 28.** On suppose qu'on a fixé les seuils et poids de l'architecture de la question précédente de manière à obtenir un réseau de neurones qui, sur les 8 tâches de cette dernière question, produit les sorties attendues. Notons  $f_i(x)$  la valeur d'activation du noeud  $v_i$  sur l'entrée  $(x, 0)$ . Montrer que  $f_i$  est soit la fonction identité, soit la fonction  $x \mapsto \text{NOT}(x)$ .

**Question 29.** On suppose toujours donné un réseau de neurones comme à la question précédente. Montrer que la valeur d'activation de  $c$ , notée  $y$ , s'exprime sous la forme  $y = \text{NOT}(x_1) \vee x_2 \vee \text{NOT}(x_3)$ , où l'entrée  $x_i = f_i^{-1}(a_i)$  est soit  $a_i$ , soit  $\text{NOT}(a_i)$ .

*Nous avons donc montré que tout réseau construit par apprentissage exact à partir de l'architecture et des tâches données à la question 27 calcule la fonction  $\text{NOT}(x_1) \vee x_2 \vee \text{NOT}(x_3)$ .*

*Plaçons-nous maintenant dans le contexte où l'architecture et les tâches de la fonction 27 sont un sous-graphe et des sous-tâches d'une architecture  $\mathcal{A}$  plus grande. Si l'on ajoute aux 8 tâches initiales une tâche de la forme  $(x_1, x_2, x_3, y, e) = (\alpha_1, \alpha_2, \alpha_3, 1, 1)$  où  $\alpha_1, \alpha_2, \alpha_3$  sont arbitraires, on déduit des questions précédentes que si l'architecture  $\mathcal{A}$  est compatible avec cet ensemble de tâches, il existe des valeurs de  $x_1, x_2, x_3$  telles que la clause  $\text{NOT}(x_1) \vee x_2 \vee \text{NOT}(x_3)$  est satisfaite.*

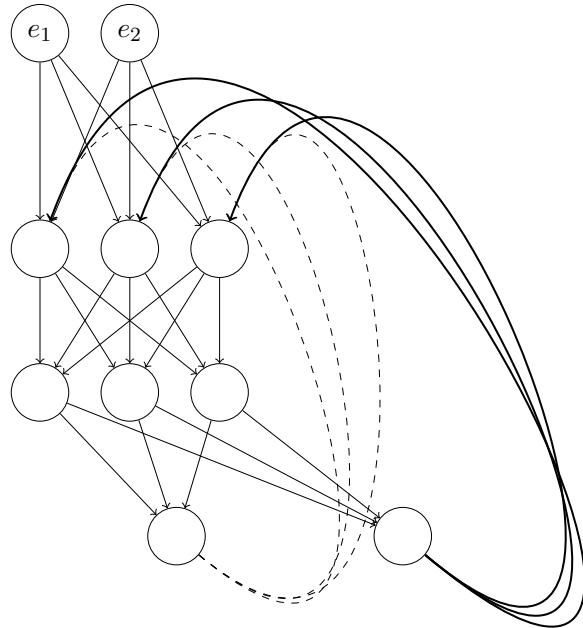
**Question\* 30.** Déduire de ce qui précède que le problème de l'apprentissage exact des réseaux de neurones de profondeur 2 est NP-complet.

## Partie IV. Réseaux de neurones récurrents et automates finis

On considère maintenant des réseaux de neurones *récurrents*, c'est-à-dire où l'organisation en couche est cyclique. On obtient un tel réseau de la façon suivante : on part d'un réseau de neurones  $R$  bien formé avec  $p + n$  entrées, et  $n$  sorties, pour deux entiers  $n$  et  $p$  : ses entrées sont numérotées de 1 à  $n + p$  et ses sorties de 1 à  $n$ . On identifie les  $n$  sorties avec les  $n$  dernières entrées : c'est-à-dire que chaque arc sortant de l'entrée numéro  $p + i$  vers un certain sommet  $s$  devient un arc sortant de la sortie numéro  $i$  vers le sommet  $s$ . Il reste  $p$  entrées que l'on considère comme les  $p$  entrées du réseau récurrent.

Dans un réseau récurrent, les valeurs d'activations des neurones évoluent au cours du temps : à chaque pas de temps, le résultat des  $n$  sorties au temps  $t$  sont réinjectées comme entrées à l'étape  $t + 1$ .

Par exemple, en partant du réseau  $R$  représenté en haut de la page 23, en considérant  $p = 2$  et  $n = 2$ , on obtient l'architecture suivante (les différents types des arêtes, ordinaire, épais, et pointillés, sont là pour améliorer la lisibilité du graphe et n'ont pas de sémantique particulière).



Si le graphe de  $R$  était le graphe acyclique  $(V \cup X, E)$  (voir la définition 1), alors le graphe obtenu s'écrit  $(V \cup X', E')$  avec  $X'$  constitué de  $p$  sommets, qui correspondent aux entrées restantes. On peut supposer que  $V$  s'écrit  $V = \{1, 2, \dots, m\}$  où  $m$  est le nombre de sommets de  $V$ . À un instant  $t$  on décrit les valeurs d'activation de chacun des neurones de  $V$  par le vecteur  $\mathbf{z}(t) = (z_1(t), z_2(t), \dots, z_m(t))$ . Initialement,  $z_i(0) = 0$  pour tout  $1 \leq i \leq m$ .

À chaque instant  $1 \leq t$ , on note  $x_1(t), x_2(t), \dots, x_p(t)$  la valeur des entrées au temps  $t$ . La valeur de  $z_j(t+1)$  est alors obtenue comme

$$z_j(t+1) = \mathcal{A} \left( \sum_{1 \leq i \leq p: (i,j) \in E'} w_{i,j} x_i(t) + \sum_{i' \in V: (i',j) \in E'} w_{i',j} z_{i'}(t) - h_j \right),$$

où  $w_{i,j}$  désigne le poids correspondant à l'arc de l'entrée  $x_i$  vers le sommet  $j$ , et  $w_{i',j}$  désigne le poids de l'arc du sommet  $i'$  de  $V$  vers le sommet  $j$ , et  $h_j$  le seuil du sommet  $j$ . Cela correspond à propager les valeurs selon la dynamique de chacun des neurones de façon synchrone (c'est-à-dire tous en même temps).

On va chercher à caractériser ce que l'on peut calculer avec de tels réseaux.

## 7 Réseaux de neurones à seuil et reconnaissance immédiate

On rappelle qu'un *automate fini* (déterministe) est la donnée de  $(Q, \Sigma, q_0, A, \delta)$  :  $Q$  est l'ensemble fini de ses états,  $\Sigma$  est son alphabet fini d'entrée,  $q_0 \in Q$  est son état initial,  $A \subseteq Q$  est son ensemble des états acceptants, et  $\delta : Q \times \Sigma \rightarrow Q$  est sa fonction de transition. On suppose que  $\Sigma = \mathbf{B} = \{0, 1\}$  est l'alphabet de tous les mots et langages mentionnés dans la suite.

À l'automate fini  $M$  et un mot  $\omega = a_1 \dots a_\ell$ , avec chaque  $a_i \in \Sigma$ , on associe la suite d'états définie par  $q(0) = q_0$ , et  $q(t+1) = \delta(q(t), a_t)$ . Le mot  $\omega$  est accepté si  $q(\ell) \in A$ . Le langage  $L(M) \subseteq \Sigma^*$  accepté par  $M$  est l'ensemble des mots acceptés. Un langage est *régulier* s'il est accepté par un automate fini.

On admettra que l'on pourrait considérer qu'il n'y a qu'un seul état acceptant (autrement dit formellement supposer que  $A$  est un singleton), et qu'il n'est pas possible de faire une transition vers l'état initial  $q_0$  (autrement dit,  $\delta(r, s) \neq q_0$  pour tout  $r \in Q, s \in \Sigma$ ) : cela ne change rien à la classe des langages acceptés. On note  $q$  le nombre d'éléments de  $Q$ .

On peut considérer que  $Q$ , l'ensemble des états, est donné comme  $\{\mathbf{e}_1, \dots, \mathbf{e}_q\} \subseteq \mathbf{B}^q$ , où le vecteur  $\mathbf{e}_i$  est le vecteur de  $\mathbf{B}^q$  dont toutes les coordonnées sont nulles, sauf la  $i$ -ème qui vaut 1.

**Question 31.** Montrer que l'on peut construire un réseau de neurones<sup>2</sup> à seuil à poids unitaires, avec  $q+1$  entrées et  $q$  sorties, qui calcule cette fonction.

Introduisons un encodage alternatif qui rend possible la construction d'un réseau de neurone de profondeur 1 : on utilise  $2q$  variables  $x_{i,e}$  pour  $1 \leq i \leq q$ , et  $e \in \mathbf{B}$ . Chaque variable  $x_{i,e}$  prend ses valeurs dans  $\mathbf{B}$ , et vaut 1 si et seulement si l'état de l'automate est  $q_i$  et le dernier symbole lu est le symbole  $e$ . À tout instant  $t \geq 1$ , exactement une de ces variables vaut 1, et toutes les autres valent 0. La fonction  $\delta : Q \times \mathbf{B} \rightarrow Q$  peut alors se voir comme une fonction<sup>3</sup>  $\mathbf{B}^{2q+1} \rightarrow \mathbf{B}^{2q+1}$  dans  $\mathbf{B}^{2q}$  qui met à jour ces variables.

2. Non-récurrent.

3. fonction partielle, au sens où elle n'est potentiellement définie que sur un sous-ensemble de  $\mathbf{B}^{2q+1}$ .

**Question 32.** Montrer que l'on peut construire un réseau de neurones<sup>4</sup> à seuil, de profondeur 1, à poids unitaires, avec  $2q + 1$  entrées et  $2q$  sorties, qui calcule cette fonction.

On veut considérer un réseau de neurones récurrent comme *reconnaissant un langage* : on considère qu'un des neurones est *le neurone de décision*. On suppose  $p = 1$  (une unique entrée). On note  $\omega = x_1(1) \cdot x_1(2) \cdot \dots \cdot x_1(\ell)$  le mot de taille  $\ell$  sur l'alphabet  $\Sigma$  obtenu en concaténant la valeur de l'entrée  $x_1(t)$  aux temps  $t = 1, \dots, \ell$ . On dit que le mot  $\omega$  est *accepté* si au temps  $\ell$ , le neurone de décision a sa valeur d'activation à 1, *rejeté* sinon.

**Question 33.** Prouver que si un langage est régulier, alors il correspond à l'ensemble des mots acceptés par un réseau de neurones récurrent à seuil.

**Question 34.** Prouver la réciproque : tout langage accepté par un réseau de neurones récurrent à seuil est régulier.

## 8 Réseaux de neurones à seuil et reconnaissance hors ligne

Dans un automate fini, la décision d'accepter (ou de rejeter) est prise immédiatement à la fin de la lecture du mot. On considère un modèle d'automate fini *hors-ligne*, c'est-à-dire que le calcul dans l'automate peut continuer après la fin de la lecture du mot reçu en entrée ; la décision d'acceptation ou de rejet intervient alors à la fin du calcul.

Formellement, un *automate fini hors-ligne* est la donnée de  $(Q, \Sigma, q_0, A, \delta)$  définis exactement comme pour un automate fini, si ce n'est que sa fonction de transition  $f$  n'envoie pas  $Q \times \Sigma$  sur  $Q$ , mais  $Q \times (\Sigma \cup \{\$\})$  sur  $Q$ , où  $\$$  est un symbole spécial ( $\$ \notin \Sigma$ ) qui encode le fait que le mot en entrée a été complètement lu par la machine. Ici encore, on suppose toujours que  $\Sigma = \mathbf{B}$ .

À l'automate fini hors-ligne  $M$  et un mot  $\omega = a_1 \dots a_\ell$ , avec chaque  $a_i \in \Sigma$ , on associe la suite d'états définie par  $q(0) = q_0$ , et  $q(t+1) = \delta(q(t), a_t)$  pour  $t = 1, 2, \dots, \ell$ , puis  $q(t+1) = \delta(q(t), \$)$  pour  $t \geq \ell$ . Le mot  $\omega$  est accepté s'il existe  $t \geq \ell$  tel que  $q(t) \in A$ .

**Question 35.** Montrer qu'un langage est régulier si et seulement s'il est accepté par un automate fini hors-ligne.

On veut considérer un réseau de neurones récurrent comme *reconnaissant un langage hors ligne* : on considère  $p = 2$  : le réseau récurrent possède deux entrées, l'une appelée  $D$ , disons  $x_1$ , qui sert à entrer les données, et l'autre  $V$ , disons  $x_2$ , qui sert à indiquer quand l'entrée est valide (ou si l'on préfère, de façon dual, à dire quand on a terminé de donner l'entrée). Jusqu'à un certain temps  $\ell$ ,  $V$  vaut 1, puis ensuite  $V$  est maintenu à 0. La valeur de l'entrée  $D$  au temps  $t = 1, 2, \dots, \ell$ , peut se voir comme un mot  $\omega$  de longueur  $\ell$  sur l'alphabet  $\Sigma = \mathbf{B}$ .

On considère qu'un des neurones est *le neurone de décision G*, et qu'un autre neurone est *le neurone de validation H*. On dit que le réseau *calcule* tant que  $H$  est à 0.

---

4. Non-récurrent.

Soit  $\omega \in \Sigma^*$ . On définit le temps d'arrêt  $T_\omega$  comme le plus petit entier (éventuellement infini)  $t$  tel que  $H(t) = 1$ , sur l'entrée  $\omega$ . Le mot  $\omega$  est accepté si  $T_\omega$  est fini et  $G(T_\omega)$  vaut 1.

**Question 36.** Prouver que si un langage est régulier alors il correspond à l'ensemble des mots acceptés par un réseau de neurones récurrent à seuil hors ligne.

*La réciproque est vraie : tout langage accepté par un réseau de neurones récurrent à seuil hors ligne est régulier. C'est essentiellement le même raisonnement que dans la question 34.*

## 9 Réseaux de neurones linéaires saturés à coefficients entiers

On s'intéresse dans cette question, et dans les parties suivantes à des circuits de neurones (réseaux de neurones) pour lesquels la fonction d'activation  $\mathcal{A}$  n'est pas nécessairement la fonction de Heaviside. Étant donnée une certaine fonction  $\mathcal{A}$  sur  $K$ , on considère les circuits que l'on obtient en prenant pour  $\mathcal{G}$  l'ensemble des neurones (un neurone à  $n$  entrées ayant l'arité  $n$ ) construits à partir de cette fonction d'activation  $\mathcal{A}$ .

En particulier, on appelle *sigmoïde idéale* la fonction continue  $\sigma(x)$  qui vaut  $x$  pour  $x \in [0, 1]$ , 0 pour  $x \leq 0$ , et 1 pour  $x \geq 1$ . On parle de neurone *linéaire saturé* pour un neurone construit en prenant cette fonction  $\sigma$  comme fonction  $\mathcal{A}$  sur  $K = \mathbb{R}$ . On parle de réseaux de neurones *linéaires saturés* lorsque tous les neurones du réseau le sont.

**Question 37.** Soit  $L$  un langage sur l'alphabet  $\mathbf{B} = \{0, 1\}$ . Prouver que les assertions suivantes sont équivalentes :

1. Le langage  $L$  est régulier.
2. Le langage  $L$  est accepté par un réseau de neurones linéaire saturé dont les poids et seuils sont entiers.
3. Le langage  $L$  est accepté hors ligne par un réseau de neurones linéaire saturé dont les poids et seuils sont entiers.

## Partie V. Réseaux de neurones récurrents ReLU

Dans cette section, on considère cette fois des réseaux de neurones où la fonction  $\mathcal{A}$  est la fonction *ReLU* (pour *Rectified Linear Unit* en anglais, soit *Unité Linéaire Rectifiée* en français), c'est-à-dire la fonction continue  $\mathcal{R}(x) = \max(x, 0)$  qui vaut  $x$  pour  $x \geq 0$ , et 0 pour  $x \leq 0$ . On parle de neurone ReLU pour un neurone construit en prenant cette fonction  $\mathcal{R}$  comme fonction  $\mathcal{A}$  sur  $K = \mathbb{R}$ , et on parle de réseau de neurones ReLU lorsque tous les neurones du réseau le sont.

Une *machine à compteurs* possède un compteur d'instruction  $R$  et un nombre fini  $k$  de compteurs  $r_1, r_2, \dots, r_k$ , qui prennent leurs valeurs dans l'ensemble  $\mathbb{N}$  des entiers naturels. L'état de la machine à un instant donné est donné par la valeur du  $k+1$ -uplet d'entiers  $(R, r_1, \dots, r_k) \in \mathbb{N}^k$ .

Un programme d'une telle machine est constitué d'une liste finie  $I_1, I_2, \dots, I_q$  d'instructions. Chaque instruction est de l'un des trois types suivants :

- $\text{Inc}(c, j)$ , pour un certain  $1 \leq c \leq k$  et  $0 \leq j \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(j, r_1, \dots, r_{c-1}, r_c + 1, r_{c+1}, \dots, r_k)$ . Autrement dit, on incrémente le compteur  $c$  et on va à l'instruction  $j$ .
- $\text{Decr}(c, j)$ , pour un certain  $1 \leq c \leq k$  et  $0 \leq j \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(j, r_1, \dots, r_{c-1}, \max(0, r_c - 1), r_{c+1}, \dots, r_k)$ . Autrement dit, on décrémente le compteur  $c$  s'il était strictement positif, on le laisse inchangé sinon, et on va à l'instruction  $j$ .
- $\text{IsZero}(c, i, j)$ , pour un certain  $1 \leq c \leq k$  et  $0 \leq i \leq q$ ,  $0 \leq j \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(i, r_1, \dots, r_k)$  si  $r_c = 0$  et  $(j, r_1, \dots, r_k)$  sinon. Autrement dit, on teste si le compteur  $c$  vaut 0, et on va à l'instruction  $i$  si c'est le cas, à l'instruction  $j$  sinon.

On fixe une valeur initiale pour les compteurs  $r_1, \dots, r_k$ . Le compteur d'instruction  $R$  vaut initialement 1. On convient que lorsque  $R = 0$ , la machine s'arrête. À chaque instant  $t$ , tant que  $R \neq 0$ , on regarde la valeur du compteur d'instruction  $R$ . On exécute alors l'instruction  $I_R$  correspondante, qui met à jour  $R$ , et les valeurs des compteurs  $r_1, \dots, r_k$  selon les règles plus haut pour cette instruction  $I_R$ .

**Question 38.** Décrire un programme d'une machine à deux compteurs qui multiplie par 2, c'est-à-dire qui, sur l'entrée  $(1, n, 0)$  arrive, après un certain nombre d'étapes, dans l'état  $(0, 2n, 0)$ .

**Question 39.** Montrer que l'on peut construire un neurone ReLU (c'est-à-dire dont la fonction d'activation est la fonction ReLU) avec une entrée et une sortie, et qui calcule la fonction qui à  $r \in \mathbb{R}$  associe  $r + 1$ . Même question pour la fonction qui à  $r \in \mathbb{R}$  associe  $\max(0, r - 1)$ . Même question pour la fonction qui à  $r \in \mathbb{R}$  associe 1 si  $r = 0$  et 0 sinon.

**Question 40.** On fixe trois entiers  $c, u, v$  et  $l$ . Décrire un programme d'une machine à compteurs (avec  $k$  suffisamment grand) tel que, lorsque la machine démarre dans l'état initial  $(1, r_1, \dots, r_k)$ , elle se retrouve après un certain temps dans l'état  $(l, r_1, \dots, r_{c-1}, r_u + r_v, r_{c+1}, \dots, r_k)$ .

*En utilisant un tel programme, on peut par conséquent considérer que le jeu d'instruction n'est pas limité aux trois types d'instructions plus haut, mais peut également être une instruction du type  $\text{Add}(c, u, v, l)$ , pour certains  $1 \leq c \leq k$ ,  $1 \leq u \leq k$ ,  $1 \leq v \leq k$ ,  $0 \leq l \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(l, r_1, \dots, r_{c-1}, r_u + r_v, r_{c+1}, \dots, r_k)$ .*

*De même on peut autoriser une instruction du type  $\text{Sub}(c, u, v, l)$ , pour certains  $1 \leq c \leq k$ ,  $1 \leq u \leq k$ ,  $1 \leq v \leq k$ ,  $0 \leq l \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(l, r_1, \dots, r_{c-1}, \max(0, r_u - r_v), r_{c+1}, \dots, r_k)$ .*

**Question 41.** On considère un neurone ReLU unitaire : pour certains poids  $w_1, w_2, \dots, w_n$  dans  $\{-1, 0, 1\}$ , et un seuil  $h$  entier, il calcule la fonction qui aux entiers naturels  $x_1, \dots, x_n$  associe  $\mathcal{R}(w_1x_1 + w_2x_2 + \dots + w_nx_n - h)$ .

Décrire un programme d'une machine à compteurs qui calcule cette fonction.

*On peut par conséquent se convaincre que pour tout réseau de neurones récurrent ReLU à poids unitaires, on peut construire une machine à compteur qui le simule.*

**Question 42.** Prouver la réciproque : montrer que pour tout programme de machine à compteurs, on peut construire un réseau ReLU à poids unitaires qui le simule. On précisera en particulier comment est codé le compteur d'instruction, et sa mise à jour. On s'autorisera à ce que la simulation ne soit pas *en temps réel* :  $t$  instructions sont simulées par  $t' > t$  étapes (c'est-à-dire  $t' > t$  propagations des valeurs selon la dynamique) du réseau : on précisera la relation entre  $t$  et  $t'$  dans la simulation proposée.

*Autrement dit, sur les entrées entières, les réseaux de neurones récurrents ReLU à poids unitaires et seuils entiers ont la même puissance que les machines à compteurs. On admettra que cela correspond également à celle des machines de Turing.*

## Partie VI. Réseaux de neurones récurrents linéaires saturés

Dans cette section, on considère à nouveau des réseaux de neurones où la fonction  $\mathcal{A}$  est la fonction *sigmoïde idéale*, c'est-à-dire la fonction continue  $\sigma(x)$  qui vaut  $x$  pour  $x \in [0, 1]$ , 0 pour  $x \leq 0$ , et 1 pour  $x \geq 1$ . On rappelle que l'on parle de neurones ou de réseaux de neurones *linéaires saturés* dans ce cas.

**Question 43.** Montrer que toute fonction calculée par un réseau de neurones linéaire saturé se calcule par un réseau de neurones ReLU (de taille et profondeur plus grande).

*La réciproque est fausse car la fonction  $\mathcal{R}$  peut prendre des valeurs en dehors de  $[0, 1]$  que l'on ne peut pas générer avec la fonction  $\sigma$ . Les résultats précédents donnent un moyen de simuler une machine de Turing (un calcul arbitraire) par un réseau ReLU, mais on utilise des neurones dont la valeur d'activation peut devenir arbitrairement grande. Par ailleurs, la simulation de  $t$  étapes d'une machine de Turing se fait en pratique avec un nombre non polynomial en  $t$  de mise à jour du réseau de neurones. On va prouver que l'on peut rester sur un domaine borné, en l'occurrence  $[0, 1]$ , en utilisant les réseaux de neurones linéaires saturés, et par ailleurs de façon beaucoup plus efficace en temps.*

On appelle *pile* une variable  $r$  qui prend ses valeurs dans l'ensemble  $\mathbf{B}^*$  des mots sur l'alphabet  $\mathbf{B}$ . Autrement dit, à un instant donné,  $r$  s'écrit  $r = w_1 w_2 \dots w_n$  avec chacun des  $w_i \in \mathbf{B}$ . On note  $push_0 : \mathbf{B}^* \rightarrow \mathbf{B}^*$  la fonction qui envoie  $r$  sur le mot  $0r$ , soit le mot qui s'écrit  $0w_1 w_2 \dots w_n$ . On note  $push_1 : \mathbf{B}^* \rightarrow \mathbf{B}^*$  la fonction qui envoie  $r$  sur le mot  $1r$ , soit le mot qui s'écrit  $1w_1 w_2 \dots w_n$ . On note  $pop : \mathbf{B}^* \rightarrow \mathbf{B}^*$  la fonction qui envoie  $r$  sur le mot obtenu enlevant sa première lettre soit  $w_2 \dots w_n$ , ou le mot vide si  $r$  était le mot vide. On note  $top : \mathbf{B}^* \rightarrow \mathbf{B}$  la fonction qui envoie  $r$  sur sa première lettre  $w_1$  (ou sur 0 si  $r$  était le mot vide).

**Question 44.** On code chaque mot  $r = w_1 \dots w_n$  sur l'alphabet  $\Sigma = \mathbf{B}$  par le rationnel de  $[0, 1]$  défini par  $\gamma(r) = \sum_{i=1}^n \frac{2w_i+1}{4^i}$ .

Montrer qu'on peut construire un neurone linéaire saturé qui simule l'effet de *top* sur une pile : si on lui donne en entrée  $\gamma(r)$ , alors sa sortie (valeur d'activation) correspond à  $top(r)$ . Montrer qu'on peut construire un neurone linéaire saturé qui simule l'effet de *push<sub>0</sub>* sur une

pile : si on lui donne en entrée  $\gamma(r)$ , alors sa sortie correspond à  $\gamma(push_0(r))$ . Même question pour  $push_1$  et  $pop$ .

Une *machine à piles* possède un compteur d'instruction  $R$  et un nombre fini  $k$  de piles  $r_1, r_2, \dots, r_k$ . L'état de la machine à un instant donné est donné par la valeur du  $k+1$ -uplet d'entiers  $(R, r_1, \dots, r_k) \in \mathbb{N} \times (\mathbf{B}^*)^k$ .

Un programme d'une telle machine est constitué d'une liste finie  $I_1, I_2, \dots, I_q$  d'instructions. Chaque instruction est de l'un des quatre types suivants :

- Push<sub>0</sub>( $c, j$ ), pour un certain  $1 \leq c \leq k$  et  $0 \leq j \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(j, r_1, \dots, r_{c-1}, push_0(r_c), r_{c+1}, \dots, r_k)$ .
- Push<sub>1</sub>( $c, j$ ), pour un certain  $1 \leq c \leq k$  et  $0 \leq j \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(j, r_1, \dots, r_{c-1}, push_1(r_c), r_{c+1}, \dots, r_k)$ .
- Pop( $c, j$ ), pour un certain  $1 \leq c \leq k$  et  $0 \leq j \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(j, r_1, \dots, r_{c-1}, pop(r_c), r_{c+1}, \dots, r_k)$ .
- Top( $c, i, j$ ), pour un certain  $1 \leq c \leq k$  et  $0 \leq i \leq q$ ,  $0 \leq j \leq q$  : si l'état de la machine est  $(R, r_1, \dots, r_k)$ , il devient  $(i, r_1, \dots, r_k)$  si  $top(r_c) = 0$  et  $(j, r_1, \dots, r_k)$  si  $top(r_c) = 1$ .

On fixe une valeur initiale pour les piles  $r_1, \dots, r_k$ . Le compteur d'instruction  $R$  vaut initialement 1. On convient que lorsque  $R = 0$ , la machine s'arrête. À chaque instant  $t$ , tant que  $R \neq 0$ , on regarde la valeur du compteur d'instruction  $R$ . On exécute alors l'instruction  $I_R$  correspondante, qui met à jour  $R$ , et les valeurs des piles  $r_1, \dots, r_k$  selon les règles plus haut pour cette instruction  $I_R$ .

**Question 45.** Montrer que toute machine de Turing peut être simulée par une machine à 2 piles.

**Question 46.** Prouver que pour toute machine de Turing, on peut construire un réseau de neurones linéaires saturés dont tous les coefficients sont rationnels qui la simule. On s'autorisera à ce que la simulation ne soit pas *en temps réel*<sup>5</sup>.

*La réciproque est vraie : tout réseau de neurones linéaires saturés dont tous les coefficients sont rationnels peut être simulé par une machine de Turing. Cela découle de la thèse de Church-Turing, ou peut se prouver en construisant explicitement un programme de machine de Turing adéquat.*

*Les réseaux de neurones linéaires saturés dont les coefficients sont des rationnels sont donc essentiellement des machines de Turing, en terme de puissance de calcul.*

**Question 47.** Formaliser le problème de l'apprentissage exact d'un réseau de neurones linéaires saturés. Prouver que le problème est indécidable.

*On remarquera que l'on cherche souvent dans le contexte des réseaux de neurones, et de l'apprentissage profond à apprendre un réseau de façon approchée, alors que ces résultats concernent*

---

5. Voir l'énoncé de la question 42.

*la question de l'apprentissage exact (dans le sens où on cherche un réseau qui correspond exactement aux exemples).*

**Question 48.** Le problème de l'apprentissage exact reste-t-il indécidable si on fixe une architecture ? Pour un réseau de neurones dont on fixe l'architecture ainsi que tous les poids et seuils, sauf un ? Discuter ce que l'on obtient.

\* \*  
\*