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 $(k+1)^e$ plus petit élément

La sélection du $(k+1)^e$ plus petit élément d'une liste d'entiers L, non nécessairement triée, consiste à trouver le $(k+1)^e$ 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 $(k+1)^e$ 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 l_k apparaissant dans la liste de listes `l` l'élément médian de l_k. On convient que pour une liste A dont les éléments sont exactement $a_0 \leq a_1 \leq \ldots \leq a_{n-1}$, l'élément médian désigne $a_{\left\lfloor \frac{n}{2} \right\rfloor}$.

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 `l1,l2,n1,n2` où `l1` est la liste des éléments de `l` plus petit que `p`, `l2` est la liste des éléments de `l` strictement plus grand que `p`, `n1` et `n2` sont respectivement les longueurs de `l1` et `l2`.

I.2 - La fonction de sélection et sa complexité

On détaille 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 :

$$\forall n \geq 55, T(n) \leq T\left(\left\lfloor \frac{n+4}{5} \right\rfloor\right) + T\left(\left\lfloor \frac{8n}{11} \right\rfloor\right) + 4n. \tag{I}$$

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 ← MEDIANS 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 $S = \{0, 1, 2, \ldots, 6\}$. 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 $\{0, 1, \ldots, n-1\}$ 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 :

$$\texttt{[[j | } j \in S \texttt{ et } (i, j) \in A \texttt{] } i \in S\texttt{]}.$$

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 $\{0, 1, \ldots, n-1\}$, 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'`Algorithme 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'`Algorithme 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]
       /* n est le nombre de sommet de G                                  */
5      pour chaque s élément de S faire
6          si C est vide alors
7              Ajouter s dans C
8          fin
9          sinon
10             c ← premier élément de C
11             t ← FAUX
               /* t permet de vérifier si on a effectué certaines instructions  */
12             si (s,c) n'est pas une arête de G alors
13                 C ← [s]
14                 t ← VRAI
15             fin
16             si (c,s) n'est pas une arête de G alors
17                 C ← C
18                 t ← VRAI
19             fin
20             si t = FAUX alors
21                 Ajouter s à la fin de liste C
22             fin
23         fin
24     fin
25 fin
26 retourner C
```

Partie III - Autour de l'énumération des fractions positives

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

Notations

Dans la suite, on notera :

- $n \wedge d$ le PGCD (Plus Grand Commun Diviseur) de n et d ;
- $\lceil x \rceil$ la partie entière supérieure de x. Ainsi $\lceil 3.45 \rceil = 4$;
- $\lfloor x \rfloor$ la partie entière inférieure de x. Ainsi $\lfloor 3.45 \rfloor = 3$;
- log_2 la fonction logarithme de base 2. C'est la fonction réciproque de la fonction $i \mapsto 2^i$.

Une *fraction*, ou *nombre rationnel*, est le quotient de deux nombres entiers, le dénominateur étant par définition non nul. On notera $\mathbb{Q}^+$ l'ensemble des fractions positives.

L'objectif de cette partie est d'étudier une structure de données permettant d'énumérer l'ensemble des fractions positives.

Plusieurs travaux se sont intéressés à l'énumération des nombres rationnels, le plus connu étant très certainement la méthode de Cantor. Cependant, il n'est très souvent pas possible, à moins d'un travail assez complexe, de connaître une formule générale donnant le i-ième terme de cette énumération, ni même de donner le successeur dans l'énumération d'une fraction donnée.
Nous proposons dans la suite de répondre à ces questions en construisant un arbre, dit de Calkin-Wilf, permettant de manipuler cette énumération.

Définition 9 (Arbre binaire infini)
Un *arbre binaire homogène* est un arbre binaire dont tous les nœuds ont 0 successeur ou 2 successeurs, appelés fils gauche et droit. La *hauteur* h de l'arbre est la profondeur maximale des nœuds de l'arbre, c'est-à-dire la plus grande longueur d'un chemin de la racine vers une feuille de l'arbre. Lorsque h est infinie, on parle d'*arbre infini*.

Définition 10 (Arbre de Calkin-Wilf)
L'*arbre de Calkin-Wilf* est un arbre binaire homogène infini dont les nœuds sont des fractions positives. La racine de l'arbre est la fraction 1/1. Chaque sommet n/d a deux descendants : un fils gauche $n/(n+d)$ et un fils droit $(n+d)/d$.
Ainsi, si $N = n/d$, les deux fils de N sont $\dfrac{N}{1+N}$ (gauche) et $1+N$ (droit).

Q20. Dessiner l'arbre de Calkin-Wilf jusqu'à une profondeur de 3. Par convention, la racine de l'arbre est au niveau 0.

On propose le type record :

```
type fraction = {n:int; d:int}
```

pour définir une fraction de type n/d. La déclaration d'une telle fraction sera donc du type :

```
let f = {n=2;d=3};;
```

l'accès au numérateur (respectivement dénominateur) s'opérant par `f.n;;` (resp. `f.d;;`).

Q21. Proposer un type `OCaml` récursif permettant de décrire l'arbre de Calkin-Wilf.

Q22. Montrer par récurrence sur le niveau d'exploration de l'arbre que si n/d est un nœud de l'arbre, alors n et d sont premiers entre eux.

Q23. Écrire une fonction récursive `OCaml` de signature `pgcd : int * int -> int` qui calcule le PGCD de deux entiers naturels.

Q24. Écrire une fonction récursive `OCaml` de signature `fraction:int->int->fraction` qui construit une fraction positive irréductible à partir d'un numérateur n et un dénominateur d. La fonction vérifie que $n > 0$ et $d > 0$. Au besoin, elle simplifie la fraction par $n \wedge d$.

Par la suite, on ne construira des valeurs de type fraction que par l'intermédiaire de la fonction `fraction`, ce qui permettra de garantir l'invariant de type suivant : "pour toute valeur `f : fraction`, `f.n` et `f.d` sont premiers entre eux".

Q25. Soient deux nœuds k/l et n/d de l'arbre ayant des fils gauches (ou droits) identiques. Montrer qu'alors $k = n$ et $l = d$.

Q26. Soient N un nœud et $k \in \mathbb{N}$. Montrer que le nœud provenant d'une suite de k fils gauches de N est le nœud $\dfrac{N}{1+kN}$. Donner sans démonstration le nœud provenant d'une suite de k fils droits de N.

On définit alors une suite $(v_i)_{i\in\mathbb{N}}$, avec $v_0 = 0$, puis en effectuant un parcours en largeur, de gauche à droite, de l'arbre de Calkin-Wilf pour définir les termes de la suite.

Q27. Donner les huit premiers termes de la suite $(v_i)_{i\in\mathbb{N}}$.

Q28. Soient $n, d \in \mathbb{N}^*$ premiers entre eux. Montrer par récurrence sur $n + d$ que toute fraction n/d apparaît dans l'arbre.

Q29. Montrer qu'aucune fraction n'apparaît deux fois dans l'arbre.

Q30. Déduire des questions précédentes que la suite $(v_i)_{i\in\mathbb{N}}$ est une bijection de $\mathbb{N}$ dans $\mathbb{Q}^+$.

La suite $(v_i)_{i\in\mathbb{N}}$ permet donc d'affirmer que $\mathbb{Q}^+$ est dénombrable. Nous allons utiliser $(v_i)_{i\in\mathbb{N}}$ pour déterminer, dans l'énumération des fractions produite par cette suite, la position de n'importe quelle fraction positive n/d. En d'autres termes, étant donnée une fraction positive n/d, on se propose de rechercher i tel que $n/d = v_i$.

Q31. Soit $i \in \mathbb{N}^*$. Montrer que le fils gauche du nœud de valeur v_i a pour valeur v_{2i}. Montrer de même que le fils droit a pour valeur v_{2i+1}.

Pour pouvoir énoncer le i-ième terme dans cette énumération, on introduit alors une suite auxiliaire, aux nombreuses propriétés arithmétiques et liens avec d'autres objets mathématiques.

La *suite diatomique de Stern* (ou *suite de Stern-Brocot*) doit son nom à Moritz Stern (1807-1894), élève de Gauss et Achille Brocot (1817-1878), horloger qui s'intéressait aux fractions pour la fabrication d'horloges avec des engrenages comportant peu de dents, donc simples à fabriquer.

Définition 11 (Suite diatomique de Stern ou suite de Stern-Brocot)
La *suite diatomique de Stern* $(s_i)_{i\in\mathbb{N}}$ est définie par $s_0 = 0$, $s_1 = 1$ et pour tout $i \geq 1$:

$$\begin{cases} s_{2i} &= s_i \\ s_{2i+1} &= s_i + s_{i+1} \end{cases}.$$

Q32. Donner les dix premiers termes de la suite $(s_i)_{i\in\mathbb{N}}$.

Q33. Écrire une fonction récursive `OCaml` de signature `stern : int -> int` permettant de calculer les termes de cette suite.

Q34. Déduire par récurrence de la **Q31** que : $\forall i \in \mathbb{N},\ v_i = \dfrac{s_i}{s_{i+1}}$.

La suite diatomique de Stern permet d'exprimer le i-ième terme dans l'énumération des fractions positives qui est induite par le parcours en largeur de l'arbre de Calkin-Wilf décrit ci-dessus.

Voyons maintenant comment obtenir de manière rapide le successeur d'une fraction v_i donnée, sans connaître i, dans cette énumération. Trois cas peuvent se présenter :

- premier cas : les nœuds de valeur v_i et v_{i+1} sont à même profondeur k, fils d'un même nœud N,
- deuxième cas : le nœud de valeur v_i est le dernier nœud à droite à la profondeur k,
- troisième cas : les nœuds de valeur v_i et v_{i+1} sont à même profondeur k, mais ne sont pas fils d'un même nœud.

On pose pour tout $x > 0$, $f(x) = \dfrac{1}{1 + 2\lfloor x \rfloor - x}$.

Q35. Montrer que dans le premier cas, $v_{i+1} = f(v_i)$.

Q36. Montrer, en utilisant la **Q26**, que dans le deuxième cas on a encore $v_{i+1} = f(v_i)$.

On étudie enfin le dernier cas : les nœuds de valeur v_i et v_{i+1} sont sur une même profondeur k, mais ne sont pas les fils d'un même nœud. On va donc passer par la recherche d'un ancêtre commun de ces deux nœuds. Dans la suite, on s'intéressera toujours au premier ancêtre commun, c'est-à-dire celui de profondeur maximale.
En partant de la racine `r`, il est possible d'atteindre n'importe quel nœud $N = n/d$ de l'arbre par une suite de déplacements vers la gauche (`G`) ou vers la droite (`D`). Le *chemin* de `r` vers N peut donc être codé par un mot sur l'alphabet {`G`,`D`}.

Si un déplacement est représenté en OCaml par un type

```
type direction = G | D
```

alors un chemin est une liste `direction list`.

Q37. Écrire une fonction OCaml de signature `chemin : fraction -> direction list` qui calcule le chemin de la racine à un nœud quelconque de l'arbre. Cette fonction fera appel à une fonction auxiliaire récursive. Ainsi `chemin n d` calcule la liste des directions à prendre pour passer de `r` au nœud n/d. On pourra supposer l'existence d'une fonction de signature `rev : direction list -> direction list` qui renverse une liste.

Q38. Écrire une fonction OCaml de signature `noeud : direction list -> fraction` qui détermine le nœud obtenu en effectuant une série de déplacements depuis la racine. Ainsi `noeud [D;D;G;D]` renverra un couple d'entiers (n, d) correspondant au nœud de valeur n/d. Cette fonction fera appel à une fonction auxiliaire récursive.

Q39. Utiliser les deux fonctions précédentes pour écrire une fonction OCaml `ancetre` de signature `ancetre : fraction -> fraction -> fraction` qui détermine le premier ancêtre commun entre deux nœuds. Ainsi `ancetre (n,d) (p,q)` détermine le premier ancêtre commun à n/d et p/q. Cette fonction pourra utiliser une fonction auxiliaire récursive.

Q40. On suppose que le nœud de valeur v_p, le premier ancêtre commun des nœuds de valeur v_i et v_{i+1}, est à k' niveaux au-dessus d'eux. Donner le chemin entre v_p et v_i sous la forme d'une liste de `direction`. Donner de même le chemin entre v_p et v_{i+1}.

Q41. À partir de l'expression des fils gauche et droit de v_p, montrer que l'on a encore $v_{i+1} = f(v_i)$.

FIN