

CONCOURS ARTS ET MÉTIERS ParisTech - ESTP - POLYTECH

Épreuve d'Informatique MP

Durée 3 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.

ENSEIGNEMENT DE SPÉCIALITÉ

AVERTISSEMENT

- Le candidat s'assurera que le sujet comporte bien 6 page numérotées de 1 à 6.
- Le sujet est composé de 4 exercices indépendants.
- Dans chaque exercice, le candidat peut admettre un résultat précédemment donné dans le texte pour aborder les questions suivantes, à condition de l'indiquer clairement sur sa copie. La qualité et la précision de la rédaction seront prises en compte dans l'appréciation des copies.
- Sauf contre-indication explicite, les programmes devront être écrits dans le langage de programmation Caml. La qualité et la présentation du code (et des commentaires) seront prises en compte dans l'appréciation des copies.

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.

Tournez la page S.V.P.

Exercice 1.

On suppose défini le type arbre de la manière suivante :

```
type arbre = Feuille of int | Noeud of arbre * arbre;;
```

On dit qu'un arbre est un *peigne* si tous les nœuds à l'exception éventuelle de la racine ont au moins une feuille pour fils. On dit qu'un peigne est *strict* si sa racine a au moins une feuille pour fils, ou si il est réduit à une feuille. On dit qu'un peigne est *rangé* si le fils droit d'un nœud est toujours une feuille. Un arbre réduit à une feuille est un peigne rangé.

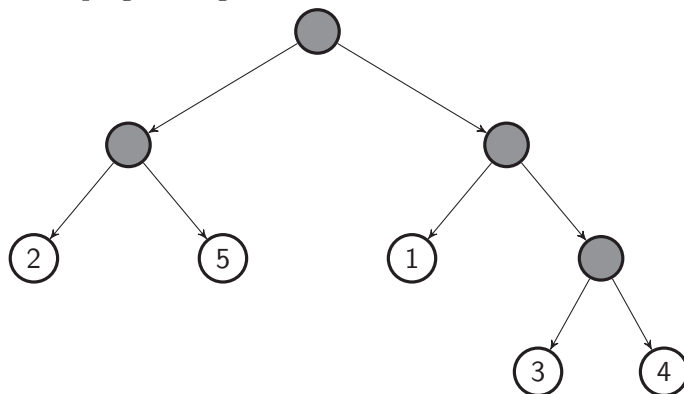


Figure 1 : Un peigne à cinq feuilles.

1. Représenter un peigne rangé à 5 feuilles.
2. La *hauteur* d'un arbre est le nombre de nœuds maximal que l'on rencontre pour aller de la racine à une feuille (la hauteur d'une feuille seule est 0). Quelle est la hauteur d'un peigne rangé à n feuilles ? On justifiera la réponse.
3. Écrire une fonction `est_range : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne rangé.
4. Écrire une fonction `est_peigne_strict : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne strict. En déduire une fonction `est_peigne : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne.
5. On souhaite ranger un peigne donné. Supposons que le fils droit N de sa racine ne soit pas une feuille. Notons A_1 le sous-arbre gauche de la racine, f l'une des feuilles du nœud N et A_2 l'autre sous-arbre du nœud N . On va utiliser l'opération de *rotation* qui construit un nouveau peigne où
 - le fils droit de la racine est le sous-arbre A_2 ;
 - le fils gauche de la racine est un nœud de sous-arbre gauche A_1 et de sous-arbre droit la feuille f .

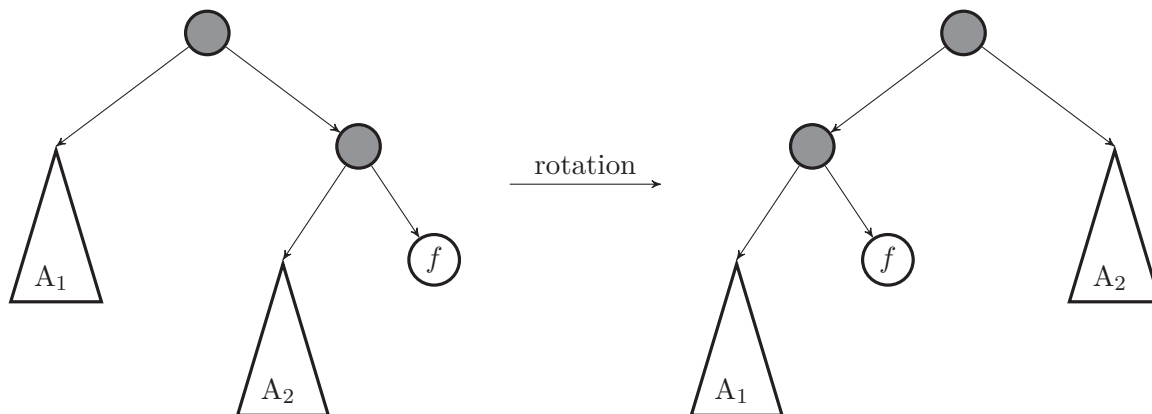


Figure 2 : Une rotation.

- (a) Donner le résultat d'une rotation sur l'arbre de la Figure 1.
- (b) Écrire une fonction `rotation : arbre -> arbre` qui effectue l'opération décrite ci-dessus. La fonction renverra l'arbre initial si une rotation n'est pas possible.
- (c) Écrire une fonction `rangement : arbre -> arbre` qui range un peigne donné en argument, c'est à dire qu'il renvoie un peigne rangé ayant les mêmes feuilles que celui donné en argument. La fonction renverra l'arbre initial si celui-ci n'est pas un peigne.

Exercice 2.

Dans une classe de n élèves, les élèves sont numérotés de 0 à $n - 1$. Un professeur souhaite faire l'appel, c'est-à-dire déterminer quels élèves sont absents.

Partie A.

1. Écrire une fonction `mini : int list -> (int * int list)`, qui prend en argument une liste non vide d'entiers distincts, et renvoie le plus petit élément de cette liste, ainsi que la liste de départ privée de cet élément (pas forcément dans l'ordre initial).
2. En notant k la longueur de la liste donnée en argument, quelle est la complexité en nombre de comparaisons de la fonction précédente ?
3. En utilisant la fonction `mini`, écrire une fonction `absents : int list -> int -> int list` qui, étant donné une liste non vide d'entiers distincts et n , renvoie, dans un ordre quelconque, la liste des entiers de $[0; n - 1]$ qui n'y sont pas.
4. En notant k la longueur de la liste donnée en argument, quelle est la complexité en nombre de comparaisons (en fonction de n et k) de la fonction précédente ?

Partie B.

Dans cette partie, une salle de classe pour n élèves est décrite par la donnée d'un tableau à n entrées. Si `tab` est un tel tableau et i un entier de $[0; n - 1]$, alors `tab.(i)` donne le numéro de l'élève assis à la place i (ou -1 si cette place est vide).

5. Écrire une fonction `asseoir : int list -> int -> int vect`, qui prend en argument une liste non vide d'entiers distincts, un entier n , et renvoie un tableau représentant une salle de classe pour n élèves où chaque élève de la liste a été assis à la place numérotée par son propre numéro. Les entiers supérieurs ou égaux à n seront ignorés.
6. En déduire une fonction `absent2 : int list -> int -> int list` qui étant donné une liste non vide d'entiers distincts et un entier n , renvoie la liste des entiers de $[0; n - 1]$ qui n'y sont pas. Les entiers supérieurs ou égaux à n seront ignorés.
7. En notant k la longueur de la liste donnée en argument, quelle est la complexité en nombre de lectures et d'écritures dans un tableau (en fonction de k et n) de la fonction précédente ?

Partie C.

Dans cette partie, indépendante des précédentes, les élèves sont déjà assis en classe.

8. On considère la fonction `place : int vect -> unit` ci-dessous :

```

let rec place tab i =
  if i <> -1 then
  begin
    let temp = tab.(i) in
      tab.(i) <- i;
      place tab temp
  end;;

```

On note `classe` le tableau `[|-1;4;5;-1;3;0|]` (on suppose $n = 6$ dans cette question). Donner le tableau `classe` après l'exécution de `place classe 1`. On donnera l'état de la variable `classe` à chaque appel récursif de la fonction.

9. On considère la fonction `placement : int vect -> int -> unit` ci-dessous :

```

let placement tab n =
  for i = 0 to n - 1 do
    if tab.(i) <> -1 && tab.(i) <> i then
    begin
      let temp = tab.(i) in
        tab.(i) <- -1;
        place tab temp
    end
  done;;

```

Si `classe` est un tableau d'entiers (les entiers positifs sont distincts) représentant une classe, que fait `placement classe` ?

Exercice 3.

On rappelle la définition de la suite de Fibonacci :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \text{ pour } n \geq 2 \end{cases}$$

Partie A. Calcul des termes de la suite.

1. On considère la fonction `fibonacci : int -> int` suivante :

```

let rec fibo = function
|0 -> 0
|1 -> 1
|n -> fibo (n-1) + fibo (n-2);;

```

Pourquoi est-il une mauvaise idée d'utiliser cette fonction pour calculer F_n ?

2. Écrire une fonction `fibonacci2 : int -> int` qui, étant donné n , calcule F_n en effectuant un nombre linéaire en n d'additions.

Partie B. Décomposition de Zeckendorf.

Pour $n \in \mathbb{N}$, on appelle décomposition de Zeckendorf de n une décomposition de n comme somme de termes distincts (d'indices supérieurs ou égal à 2) de la suite de Fibonacci, de telle manière qu'il n'y ait pas deux termes d'indices consécutifs dans la somme. Autrement dit, il existe des indices c_0, c_1, \dots, c_k tels que :

- $c_0 \geq 2$,
- pour tout $i < k : c_{i+1} > c_i + 1$ (pas d'indices consécutifs),
- $n = \sum_{i=0}^k F_{c_i}$

Par exemple : $2 + 5 = F_3 + F_5$ est une décomposition de Zeckendorf de 7 (avec $c_0 = 3$ et $c_1 = 5$), alors que $3 + 5 = F_4 + F_5$ n'est pas une décomposition de Zeckendorf de 8, Car F_4 et F_5 sont deux termes consécutifs de la suite de Fibonacci.

3. Déterminer une décomposition de Zeckendorf de 20, 21 et 22.
4. Montrer que tout entier strictement positif admet une décomposition de Zeckendorf (on admet qu'elle est unique).

Partie C. Codage de Fibonacci.

On appelle codage de Fibonacci d'un entier positif k un tableau `tab` de 0 et de 1 indiquant par des 1 les indices des termes de la suites de Fibonacci utilisés dans la décomposition de Zeckendorf de k (`tab.(0)` indique alors si F_2 est utilisé dans la représentation). On remarque que par définition de la décomposition de Zeckendorf, `tab` ne peut pas contenir deux 1 consécutifs.

`[|0,1,0,1|]` est ainsi une représentation de Fibonacci de 7.

5. Écrire une fonction `decode : int vect -> int` qui traduit en entier une représentation de Fibonacci d'un entier.
6. (a) Décrire, sans l'implémenter, une fonction `plusun : int vect -> int vect`, qui à partir d'une représentation de Fibonacci d'un entier k , renvoie une représentation de Fibonacci de l'entier $k + 1$.
(b) Décrire, sans l'implémenter, une fonction `moinsun : int vect -> int vect`, qui à partir d'une représentation de Fibonacci d'un entier k non-nul, renvoie une représentation de Fibonacci de l'entier $k - 1$.

Exercice 4.

Dans cet exercice, les programmes seront écrits avec le langage python. On pourra utiliser si besoin la bibliothèque numpy :

- `zeros([n,p])` renvoie un tableau bidimensionnel (n,p) rempli de 0.
- Si `T` est un tableau bidimensionnel, `T[i,j]` accède à l'élément ligne i colonne j de ce tableau.

Le réseau du métro d'une grande ville comporte N stations réparties un certains nombre de lignes interconnectées (par exemple, pour la ville de Lyon on a $N = 40$ avec 4 lignes).

On représente ce réseau par un graphe non orienté $G = ([0, N - 1], A)$ défini par :

- L'ensemble des sommets est $[0, N - 1]$. On a numéroté ces stations de 0 à $N - 1$. Le sommet i du graphe G représente la station i .

- A désigne l'ensemble des arêtes de G . Une arête entre les sommets i et j n'existe que si les stations i et j sont des stations adjacentes sur une même ligne de métro, pour tous i, j dans $\llbracket 0, N - 1 \rrbracket$. Ce graphe est représenté par la liste `Liste_A` (donnée en variable globale) de ses arêtes sous la forme $[i, j]$ pour i, j dans $\llbracket 0, N - 1 \rrbracket$ tels que $i < j$ et $(i, j) \in A$.
1. Écrire une fonction `voisin_G` qui prend en entrée un entier i dans $\llbracket 0, N - 1 \rrbracket$ et retourne la liste des sommets adjacents au sommet i dans le graphe G .

On suppose le graphe G connexe, et on cherche à déterminer le trajet le plus rapide entre deux stations du réseau :

Soit `duree` la fonction définie sur A qui attribue à une arête (i, j) la durée du trajet entre la station i et la station j en minutes, arrondie sur un nombre entier.

Un trajet entre deux stations i et j correspond à un chemin dans G qui part de i et arrive en j . La durée d'un tel trajet $i = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_n = j$ utilisant n arêtes est la somme des durées des n arêtes :

$$duree(i = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_n = j) = \sum_{k=0}^{n-1} duree((i_k, i_{k+1})).$$

Pour simplifier, on ne tient pas compte des durées correspondant aux changements de métro.

On dispose de la liste `Duree` des listes $[i, j, duree(i, j)]$, pour i, j dans $\llbracket 0, N - 1 \rrbracket$ tels que $i < j$ et $(i, j) \in A$. On note `D` la valeur de `sum([U[2] for U in Duree])`.

2. Soient i et j deux sommets de G . Démontrer qu'il existe au moins un trajet de durée minimale entre i et j .

Pour i et j deux sommets de G , on note $\delta_{min}(i, j)$ la durée minimale d'un trajet entre i et j .

3. Soient i et j deux sommets de G . Soit $i = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_n = j$ un chemin dans G qui part de i et arrive en j en utilisant n arêtes ($n \geq 1$). On suppose que ce chemin réalise un trajet de durée minimale entre i et j . Justifier que pour tout k entre 1 et n , $i = i_0 \rightarrow \dots \rightarrow i_k$ est un trajet de durée minimale entre i_0 et i_k et $i_k \rightarrow \dots \rightarrow i_n = j$ est un trajet de durée minimale entre i_k et j .
4. En déduire, pour i et j distincts tels que $(i, j) \notin A$, une expression de $\delta_{min}(i, j)$ en fonction des valeurs de $\delta_{min}(i, k)$ et $\delta_{min}(k, j)$, k parcourant la liste des sommets du graphe G , à l'exception de i et j . Justifier votre réponse.
5. Définir `Tinit` le tableau bidimensionnel (N, N) tel que :

$$\forall (i, j) \in \llbracket 0, N - 1 \rrbracket^2, \text{Tinit}[i, j] = \begin{cases} 0, & \text{si } i = j, \\ duree(i, j), & \text{si } (i, j) \in A, \\ D, & \text{sinon.} \end{cases}$$

6. Définir la fonction `FW` qui prend en entrée un tableau bidimensionnel (N, N) `T` et retourne le tableau bidimensionnel (N, N) `T'` défini par :

$$\forall (i, j) \in \llbracket 0, N - 1 \rrbracket^2, T'[i, j] = \min(T[i, j], T[i, k] + T[k, j], k \in \llbracket 0, N - 1 \rrbracket).$$

7. Écrire un programme qui, en utilisant le tableau `Tinit` et la fonction `FW`, permet de calculer un tableau bidimensionnel (N, N) dont le (i, j) -ème coefficient vaut $\delta_{min}(i, j)$, pour tous i et j , sommets de G . Combien d'itérations sont-elles nécessaires pour conclure ?
8. Expliquer comment modifier le programme pour obtenir un trajet de durée minimale entre i et j , pour tous i et j , sommets de G . On ne demande pas de le programmer précisément, mais d'expliquer ce qu'il faudrait ajouter au programme précédent pour obtenir de plus ces informations.

