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.

AVERTISSEMENT

- L'épreuve est composée de 3 exercices indépendants.
- Un candidat pourra toujours admettre le résultat des questions qu'il n'a pas faites pour faire les questions suivantes.
- Les programmes devront être écrits dans le langage de programmation Python pour l'exercice 1 et OCaml pour les exercices 2 et 3.

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 – Autour de la recherche par dichotomie

Partie 1 – Questions de cours

1 Rappeler le principe de la recherche dichotomique dans une liste d'entiers. Quel intérêt présente cette méthode ?

2 La mise en œuvre d'une recherche dichotomique est-elle possible sur une liste de couples d'entiers ? de chaînes de caractères ?

Partie 2 – Étude d'une fonction `dicho`

Voici le code d'une fonction Python élaboré pour tester par dichotomie si un entier `x` se trouve dans une liste d'entiers `liste` :

```
(1) def dicho(liste,x):
(2)     # Pré-conditions: x est un entier, liste est une
(3)     # liste d'entiers triée dans l'ordre croissant
(4)     n = len(liste)
(5)     if n == 0:
(6)         return False
(7)     g, d = 0, n-1
(8)     while d-g > 0:
(9)         m = (g+d)//2
(10)        if liste[m] >= x:
(11)            d = m
(12)        else:
(13)            g = m
(14)    return liste[g] == x
```

3 Pour quelles raisons ne remplace-t-on pas la précondition de la ligne (3) par un appel à une fonction qui trierait la liste `liste` dans l'ordre croissant ?

4 Justifier que le prédicat

$\mathscr{P}$: « l'entier `x` apparaît dans la sous-liste `liste[g :d+1]` des éléments de `liste` d'indices `g` à `d` »

est préservé à chaque tour de la boucle `while`.

5 Il s'avère que la fonction `dicho` ne termine pas. Donner un exemple où la fonction boucle.

6 Indiquer sans justification la ou les corrections à apporter pour que la fonction `dicho` termine, tout en restant correcte.

7 Justifier que la fonction corrigée termine et est correcte.

Partie 3 – Extensions du principe

Une première extension consiste à réduire la taille du problème non plus en 2 mais en 3 : c'est le principe de trichotomie.

8 Écrire en Python une fonction `tricho(liste,x)` qui renvoie `True` si l'élément `x` se trouve dans la liste `liste` et `False` sinon. Cette fonction sera récursive ou fera appel à une ou des fonctions auxiliaires récursives.

9 Estimer la complexité de la fonction `tricho(liste,x)` pour une liste `liste` à n éléments. Comparer avec la méthode par dichotomie.

Une seconde extension consiste à adapter le principe de dichotomie au cas d'une matrice d'entiers dont les éléments sont triés en colonne de haut en bas et de gauche à droite. L'illustration ci-dessous indique l'ordre des éléments d'une matrice à 4 lignes et 6 colonnes :

$$\begin{pmatrix} 0 & 4 & 8 & 12 & 16 & 20 \\ 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \end{pmatrix}$$

Une matrice sera implémentée en Python par une liste de ses lignes, elles-mêmes implémentées par des listes.

10 Écrire en Python une fonction `dicho_matrice(mat,x)` qui prend en argument un entier `x` et une matrice `mat` à n lignes et p colonnes ($n \geqslant 1$ et $p \geqslant 1$), d'entiers triés en colonne de haut en bas et de gauche à droite, et qui renvoie :

- le couple d'indices `(i,j)` minimal pour l'ordre défini plus haut tel que `x` se trouve en ligne `i` et colonne `j`, si l'entier `x` est bien présent dans la matrice `mat` ;
- le couple `(-1,-1)` si `x` n'est pas présent dans la matrice `mat`.

Cette fonction devra être de complexité logarithmique en $\max(n, p)$.

Tournez la page S.V.P.

Exercice 2 – Automates et langages de mots binaires

Dans cet exercice, on étudie différents langages sur l'alphabet $\mathrm{A} = \{0, 1\}$ à deux lettres. On note A^* l'ensemble des mots construits sur l'alphabet A. Le mot vide est noté ε. En OCaml, un mot sur A est implémenté par le type

```
type mot = bool list;;
```

à savoir par une liste de booléens où la lettre 0 est représentée par le booléen `false` et la lettre 1 par le booléen `true`.

11 On note L_1 le langage des mots de A qui représentent l'écriture binaire d'un entier naturel, où le bit de poids faible se situe en fin de mot. Pour assurer l'unicité de la représentation, l'écriture binaire d'un entier ne commence jamais par 0. C'est pourquoi l'entier nul est représenté par le mot vide.

11a) Donner l'écriture binaire de l'entier 41.

11b) Donner l'entier représenté par le mot 10101010.

11c) Pour un automate, que signifie la propriété d'être local standard ?

11d) Dessiner un automate local standard $\mathcal{A}_1$ reconnaissant le langage L_1.

11e) Écrire en OCaml une fonction `langage_1` de type `mot -> bool` qui prend en argument un mot m et qui renvoie `true` si et seulement si m appartient au langage L_1.

12 On note L_2 le langage dénoté par l'expression rationnelle $(0 + 1)^* \cdot 0$.

12a) Justifier que L_2 est un langage local.

12b) Dessiner un automate déterministe $\mathcal{A}_2$ reconnaissant le langage L_2.

12c) Écrire en OCaml une fonction `langage_2` de type `mot -> bool` qui prend en argument un mot m et qui renvoie `true` si et seulement si m appartient au langage L_2.

13 On note L_3 le langage reconnu par l'automate déterministe $\mathcal{A}_3$ défini par

- l'ensemble d'états $\mathrm{Q} = \{0, 1, 2\}$;
- l'état initial $i = 0$;
- l'ensemble d'états finals $\mathrm{F} = \{0\}$;
- la fonction de transition $\delta : \mathrm{Q} \times \mathrm{A} \longrightarrow \mathrm{Q}$ définie par

$$\forall\, q \in \mathrm{Q} \quad \forall\, a \in \mathrm{A} \qquad \delta(q, a) = (2q + a) \bmod 3$$

On rappelle que $(n \bmod 3)$ désigne le reste de la division euclidienne de l'entier n par 3.

13a) Dessiner l'automate $\mathcal{A}_3$.

13b) Écrire en OCaml une fonction `langage_3` de type `mot -> bool` qui prend en argument un mot m et qui renvoie `true` si et seulement si m appartient au langage L_3.

13c) Pour tout $n \in \mathbb{N}^*$, démontrer la propriété $\mathscr{P}(n)$ suivante :

$$\forall\, \omega_1, \ldots, \omega_n \in \mathrm{A} \qquad \delta^\star(0, \omega_1 \cdots \omega_n) = \sum_{k=1}^{n} \omega_k \, 2^{n-k} \bmod 3$$

où la fonction $\delta^\star : \mathrm{Q} \times \mathrm{A}^* \longrightarrow \mathrm{Q}$ est définie par

$$\forall\, q \in \mathrm{Q} \quad \forall\, \omega \in \mathrm{A}^* \quad \forall\, a \in \mathrm{A} \qquad \delta^\star(q, \varepsilon) = q \qquad \text{et} \qquad \delta^\star(q, \omega \cdot a) = \delta\big(\delta^\star(q, \omega), a\big)$$

14 On note $\mathrm{L}_4 = \mathrm{L}_1 \cap \mathrm{L}_2 \cap \mathrm{L}_3$.

14a) Décrire simplement l'ensemble des mots du langage L_4.

14b) Existe-t-il un automate reconnaissant le langage L_4 ?

Exercice 3 – Diamètre d'un graphe

Dans cet exercice, on considère des graphes non orientés connexes. Les sommets d'un graphe à n sommets ($n \in \mathbb{N}^*$) sont numérotés de 0 à $n-1$. On suppose qu'aucune arête ne boucle sur un même sommet.

Un *chemin de longueur* $p \in \mathbb{N}$ d'un sommet a vers un sommet b dans un graphe est la donnée de $p+1$ sommets $s_0, s_1, \ldots, s_p$ tels que $s_0 = a$, $s_p = b$ et, pour tout $1 \leqslant k \leqslant p$, les sommets s_{k-1} et s_k sont reliés par une arête.

Un *plus court chemin* d'un sommet a vers un sommet b dans un graphe G est un chemin de longueur minimale parmi tous les chemins de a vers b. Sa longueur est notée $d_{\mathrm{G}}(a, b)$.

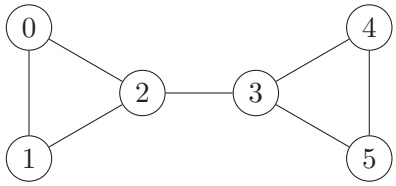
Le *diamètre* d'un graphe G, noté diam(G), vaut le maximum des longueurs des plus courts chemins entre deux sommets du graphe G. Autrement dit,

$$\mathrm{diam(G)} = \max_{a,b \text{ sommets de G}} d_{\mathrm{G}}(a, b)$$

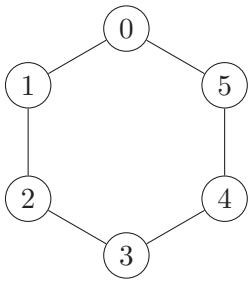
Un *chemin maximal* d'un graphe G est un plus court chemin de G de longueur diam(G).

Partie 1 – Exemples de graphes

15 Donner sans justification le diamètre et les chemins maximaux pour chacun des deux graphes G_1 et G_2 ci-dessous.



graphe G_1



graphe G_2

En OCaml, les graphes sont représentés par liste d'adjacence et implémentés par le type

```
type graphe = int list array;;
```

16 Graphes de diamètre maximal.

16a) Dessiner sans justification un graphe à 5 sommets ayant un diamètre le plus grand possible.

16b) Écrire en OCaml une fonction `diam_max` de type `int -> graphe` qui prend en argument un entier naturel n non nul et qui renvoie un graphe à n sommets de diamètre maximal.

17 Graphes de diamètre minimal.

17a) Dessiner sans justification un graphe à 5 sommets ayant un diamètre le plus petit possible.

17b) Écrire en OCaml une fonction `diam_min` de type `int -> graphe` qui prend en argument un entier naturel n non nul et qui renvoie un graphe à n sommets de diamètre minimal.

Tournez la page S.V.P.

Partie 2 – Algorithmes de calcul du diamètre

Dans cette partie, on suppose que les graphes sont représentés par listes d'adjacence.

18 Donner l'entrée et la sortie de l'algorithme de Dijkstra. Comment cet algorithme permet-il de calculer le diamètre d'un graphe ?

19 Quel parcours de graphe peut être utilisé pour le calcul du diamètre ?

20 Laquelle des deux méthodes précédentes est la mieux adaptée pour calculer le diamètre d'un graphe ?

Partie 3 – Diamètre d'un arbre binaire

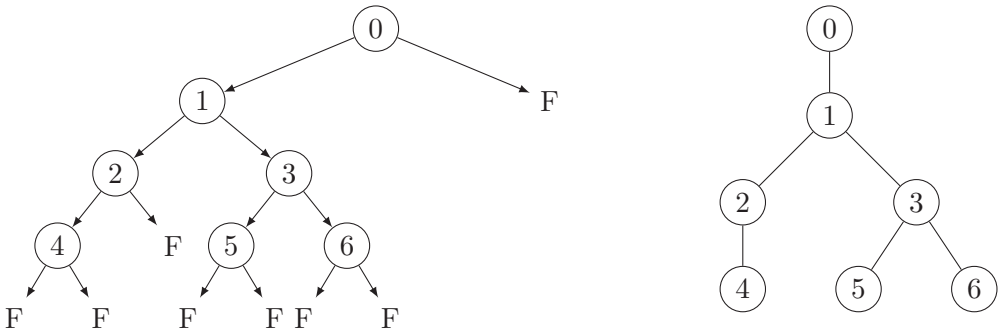
Dans cette partie, on s'intéresse aux arbres binaires, qui sont des cas particuliers de graphes. On travaille avec une représentation spécifique de ces graphes particuliers, implémentée en OCaml par le type suivant :

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

Le *graphe sous-jacent* $G_{\mathcal{A}}$ à un arbre binaire $\mathcal{A}$ est défini comme le graphe orienté dont

- les sommets correspondent aux nœuds de l'arbre (et pas aux feuilles) ;
- les arêtes correspondent aux branches de l'arbre reliant deux nœuds (et non pas celles reliant un nœud à une feuille).

Le diamètre d'un arbre binaire est alors défini comme le diamètre de son graphe sous-jacent.
Voici un exemple d'arbre binaire $\mathcal{A}$ (à gauche) et de son graphe sous-jacent $G_{\mathcal{A}}$ (à droite) :



21 Donner l'expression OCaml représentant l'arbre $\mathcal{A}$ de l'exemple. Donner le diamètre de $\mathcal{A}$ et les chemins maximaux du graphe sous-jacent $G_{\mathcal{A}}$.

22 Quel est le nombre r d'arêtes du graphe sous-jacent à un arbre binaire possédant n nœuds ?

Une première approche pour calculer le diamètre d'un arbre consiste à le transformer en un graphe et à employer un algorithme général sur les graphes de la partie 2.

23 Écrire en OCaml une fonction `nb_noeuds` de type `arbre -> int` qui renvoie le nombre de nœuds d'un arbre binaire donné en argument.

24 Écrire en OCaml une fonction `numerotation` de type `arbre -> arbre` qui prend en argument un arbre binaire $\mathcal{A}$ à n nœuds et qui renvoie un arbre binaire $\mathcal{A}'$ de même graphe sous-jacent que $\mathcal{A}$ et dont les nœuds sont étiquetés de 0 à $n-1$.

25 Écrire en OCaml une fonction `arbre_vers_graphe` de type `arbre -> graphe` qui prend en argument un arbre binaire $\mathcal{A}$ à n nœuds étiquetés de 0 à $n-1$ et qui renvoie le graphe $G_{\mathcal{A}}$ sous-jacent à $\mathcal{A}$ (le type `graphe` est défini dans la partie 1).

26 Décrire un algorithme qui calcule le diamètre d'un arbre de type `arbre` en se ramenant à un graphe. Quelle est sa complexité ?

Une seconde approche pour calculer le diamètre d'un arbre consiste à employer une technique diviser-pour-régner. Pour tout arbre $\mathcal{A}$ non réduit à une feuille, de la forme `Noeud (x, arbre_g, arbre_d)`, on note

- $\mathcal{A}_g$ le fils gauche de $\mathcal{A}$, représenté par `arbre_g` ;
- $\mathcal{A}_d$ le fils droit de $\mathcal{A}$, représenté par `arbre_d`.

La hauteur de l'arbre $\mathcal{A}$, notée $h(\mathcal{A})$, est la longueur du plus long chemin descendant de la racine vers une feuille. Dans l'arbre exemple de la partie 3, l'arbre $\mathcal{A}$ est de hauteur 4.

27 Quelle est la longueur d'un chemin maximal passant par la racine ?

28 Écrire en OCaml une fonction `diam_arbre` de type `arbre -> int` qui calcule le diamètre d'un arbre donné en argument. Cette fonction devra être de complexité linéaire en le nombre de nœuds de l'arbre.

FIN D'ÉPREUVE