Devoir Surveillé 6

20 avril 2024

INFORMATIQUE MP2I

DURÉE DE L'ÉPREUVE : **4 heures**

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

Ce sujet comporte huit pages numérotées de 1/8 à 8/8

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.

Tournez la page S.V.P.

Vue d'ensemble du sujet

Ce sujet est composé de 3 parties indépendantes, utilisant les langage de programmation C et OCaml.

Les différentes parties sont indépendantes et peuvent être traités dans n'importe quel ordre. Il n'est pas nécessaire d'avoir répondu aux questions d'une partie pour répondre aux questions d'une autre partie.

- La partie I s'intéresse à la représentation en machine d'ensemble de nombres, en analysant dans un premier temps 3 approches classiques : listes, vecteurs et arbres binaires de recherches, puis dans un second temps en proposant une approche basée sur les arbres binaires complets.
- La partie II s'intéresse au problème de la chaîne de multiplications de matrices, et en propose une résolution par programmation dynamique.
- La partie III s'intéresse à une variante de l'algorithme de Rabin-Karp pour détecter des anagrammes dans un dictionnaire.

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat. En langage C, il est inutile de rappeler que les entêtes `<assert.h>`, `<stdbool.h>`, etc. doivent être inclus. En OCaml, les noms des arguments des fonctions sont donnés avec des indications de type. Il n'est pas nécessaire de recopier ces indications de type sur votre copie.

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.

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 Différentes implémentation d'ensembles de nombres – 35 + 40 pts

Dans cette section, on s'intéresse à des structures de données représentant des ensembles d'entiers naturels E contenus dans $[\![0, N-1]\!]$, « 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.

☐ 1 – Donner les successeur de 5 dans les ensembles suivants :

- $E_0 = \{1; 3; 5; 7\}$
- $E_1 = \{2; 3; 6; 8\}$
- $E_2 = \{10; 15; 20\}$

1.1 Avec une liste triée – OCaml

☐ 2 – 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.

1.2 Avec un vecteur trié – C

Soit N un entier naturel strictement positif, fixé pour toute cette partie. On choisit de représenter un ensemble d'entiers E de cardinal $n \leqslant N$ 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 $\{2, 5, 7\}$.

☐ 3 – 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).

☐ 4 – Par une méthode dichotomique, écrire une fonction `int succ_vect(int t[], int x)` 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).

☐ 5 – Calculer la complexité dans le pire cas de la fonction `succ_vect` en fonction de n.

☐ 6 – Écrire une fonction `int *union_vect(int N, int t_1[], int t_2[])` prenant en arguments deux tableaux `t_1` et `t_1`, 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 $|E_1 \cup E_2| \leqslant N$.

1.3 Avec un arbre binaire de recherche – OCaml

On choisit maintenant de représenter un ensemble d'entiers à l'aide d'un arbre binaire de recherche, les nœuds 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 nœud x, les étiquettes de tous les nœuds 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 nœuds 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.

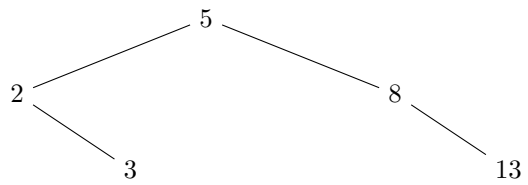

FIGURE 1 – Un arbre binaire de recherche pour l'ensemble $\{2, 3, 5, 8, 13\}$

☐ 7 – 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).

☐ 8 – 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.

☐ 9 – 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.

☐ 10 – 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.

1.4 Avec un arbre binaire complet – C (40 pts)

On considère dans cette partie des arbres binaires complets dont les nœuds sont étiquetés par des booléens. On rappelle que la profondeur d'un nœud est égale à la longueur du chemin reliant la racine à ce nœud et la hauteur d'un arbre est la plus grande profondeur de ses feuilles (la racine est donc à la profondeur 0). Les nœuds 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).

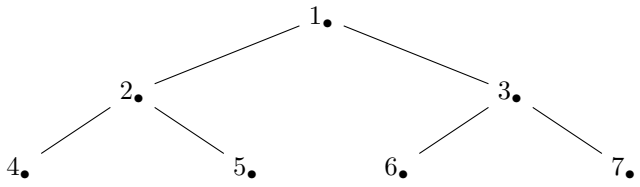

FIGURE 2 – Numérotation des nœuds

☐ 11 – Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, quel numéro peut avoir un nœud à la profondeur $k \in [\![0, p]\!]$? Combien le sous-arbre dont la racine a le numéro i a-t-il de feuilles ?

☐ 12 – Dans un arbre binaire complet de hauteur $p \in \mathbb{N}$, pour le nœud 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 $i \in [\![1, 2^{p+1} - 1]\!]$ contenant l'étiquette du nœud numéroté i. La case d'indice 0 est utilisée pour stocker la taille du tableau 2^{p+1}. On notera que dans tous les programmes de cette partie, on peut avoir accès à la valeur 2^p via la taille du tableau en case d'indice 0.
Soit $p \in \mathbb{N}$. On choisit de coder un ensemble $E \subset [\![0, 2^p - 1]\!]$ 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 nœud interne est étiqueté par le « ou logique» des étiquettes de ses deux fils.

```
typedef bool * ensemble;
```

Par exemple, l'ensemble $\{0, 1, 7\}$ est représenté par l'arbre représenté figure 3.

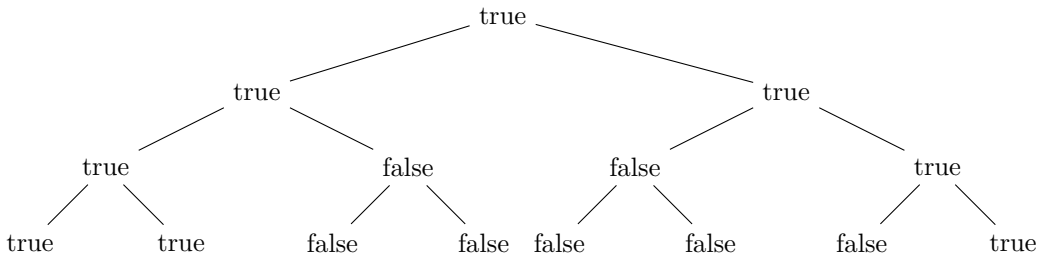

FIGURE 3 – Arbre associé à l'ensemble $\{0, 1, 7\}$ pour $p = 3$

☐ 13 – Écrire une fonction `bool appartient(ensemble E, int x)` qui détermine si un entier quelconque appartient ou non à un ensemble donné. Calculer sa complexité.

☐ 14 – Écrire une fonction `ensemble fabrique(int len, int tab, int 2_to_p)` qui prend en arguments un tableau `tab` d'entiers positifs distincts de taille `len` et une valeur pour 2^p et renvoie l'arbre associé à l'ensemble E dont les éléments sont ceux de `tab`. On supposera que tous les éléments de `tab` appartiennent à $[\![0, 2^p - 1]\!]$. Cette fonction devra s'exécuter en $\mathcal{O}(2^p)$.

☐ 15 – Écrire une fonction `void insere(ensemble E, int k)` 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.

☐ 16 – Écrire une fonction `void supprime(ensemble E, int k)` 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.

☐ 17 – Écrire une fonction `int minlocal(ensemble E, int i)` 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 $x \in [\![0, 2^p - 1]\!]$:
- on part de la case numéro i codant l'entier x dans E ;
- tant que i n'est pas le nœud 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.

☐ 18 – Prouver l'algorithme décrit ci-dessus.

☐ 19 – Écrire une fonction `int successeur(ensemble E, int x)` prenant en arguments l'ensemble E et un entier x positif et renvoyant son successeur dans E (-1 si cela n'existe pas).

☐ 20 – 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 $K(\log_2(\texttt{successeur}(x) - x) + 2)$. 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 $K(\log_2(2^p - x) + 2)$.

☐ 21 – **En utilisant la fonction `successeur`**, écrire une fonction `int cardinal(ensemble E)` prenant en argument un ensemble et renvoyant son cardinal.

☐ 22 – Déterminer la complexité de la fonction `cardinal` en fonction de p et $n = |E|$. On rappelle que la fonction $\log_2$ est concave.

☐ 23 – Quels sont les intérêts et inconvénients d'une telle structure ? Dans quels cas peut-elle s'avérer plus intéressante que des structures connues ?

2 Multiplication de matrices – C (35 pts)

Étant donné deux matrices $A = (a_{i,j}) \in \mathbb{R}^{n\times k}$ et $B = (b_{i,j}) \in \mathbb{R}^{k\times m}$, le produit de ces matrices est la matrice $AB = (c_{i,j}) \in \mathbb{R}^{n\times m}$ telle que

$$c_{i,j} = \sum_{i=1}^{k} a_{i,k} b_{k,j}$$

Le but de cette section est de s'intéresser au produit de ℓ matrices $M_1 M_2 \dots M_\ell$.
Attention, la notation mathématique commence sa numérotation à 1 alors que le langage C impose que la numérotation des tableaux commence à 0.
Nous représenterons en C les matrices par des tableaux bidimensionnels de flottants :

```
typedef double ** fmatrix;
```

☐ 24 – Écrire une fonction `fmatrix new_matrix(int n1, int n2)` qui prend en entrée deux entiers n_1 et n_2 et qui renvoie une nouvelle matrice de $\mathbb{R}^{n_1 \times n_2}$ ne contenant que des zéros.

☐ 25 – Écrire une fonction `void free_matrix(fmatrix m, int n1)` qui prend en entrée deux entiers n_1 et n_2 et qui renvoie une nouvelle matrice de $\mathbb{R}^{n_1 \times n_2}$ ne contenant que des zéros.

☐ 26 – Écrire une fonction `fmatrix mult_matrix(fmatrix m1, fmatrix m2, int n1, int n2, int n3)` qui prend en argument deux matrices $M_1 \in \mathbb{R}^{n_1 \times n_2}$ et $M_2 \in \mathbb{R}^{n_2 \times n_3}$. La complexité de la fonction ne doit pas dépasser $O(n_1 n_2 n_3)$, on ne demande pas de le justifier. Dans la suite de l'énoncé, on suppose que notre multiplication de matrice possède comme complexité $n_1 n_2 n_3$.

☐ 27 – Si la matrice M_i a pour dimensions $n_i \times n_{i+1}$, quel est la dimension de la matrice $M_1 M_2 \dots M_\ell$?

☐ 28 – Étant donné trois matrices $A \in \mathbb{R}^{2\times 3}$, $A \in \mathbb{R}^{3\times 4}$, et $A \in \mathbb{R}^{4\times 2}$, on peut calculer ABC de deux manières différentes : $(AB)C$ ou $A(BC)$. Quelle est la méthode la plus efficace ?

☐ 29 – Étant donné 4 matrices A, B, C, D, combien y a-t-il de manières différentes d'obtenir le produit $ABCD$?

☐ 30 – On note C_ℓ le nombre de manières différentes de calculer le produit de ℓ matrices.
($C_0 = C_1 = C_2 = 1, C_3 = 2$)
En s'intéressant au dernier produit calculé, donner une relation de récurrence vérifiée par C_ℓ. Vérifier que $C_5 = 14$.

Passons à l'implémentation d'une solution. Soit une instance du problème que l'on souhaite résoudre : un tableau de ℓ matrices numérotés de 0 à $\ell - 1$: $M_0, \dots M_{\ell-1}$, avec M_i de dimension $n_i \times n_{i+1}$ (On a juste décalé les indices de 1 par rapport à la question 27).
On considère comme sous-problèmes la recherche des méthodes les plus efficaces pour calculer le produit des $j - i + 1$ matrices $M_i \dots M_j$ pour chaque couple (i, j).
Notons $C(i, j)$ le coût du calcul le plus efficace de $M_i \dots M_j$. Le problème initial est de calculer $C(0, \ell - 1)$. Pour i allant de 0 à $\ell - 1$, on a $C(i, i) = 0$.

☐ 31 – Pour un k donné, quels sont les i pour lesquels $C(i, i + k)$ est défini ?

☐ 32 – Combien y a-t-il de sous-problèmes, quelle structure de données peut-on utiliser pour stocker leur résultat ?

☐ 33 – Pour i allant de 0 à $\ell - 1$, quelle est la complexité $C(i, i + 1)$?
On rapelle que pour multiplier deux matrices de dimension $n_1 \times n_2$ et $n_2 \times n_3$, la complexité est $n_1 n_2 n_3$.

☐ 34 – Donner une relation de récurrence permettant de calculer $C(i, j)$, pour $j > i + 1$.

☐ 35 – Dans quel ordre peut-on calculer les $C(i, j)$?

☐ 36 – Écrire une fonction `int **methode_optimale(fmatrix *m, int *n, int ell)` qui prend en entrée une liste de matrices de taille ℓ et une liste de dimensions de taille $\ell + 1$ et qui renvoie le tableau des $C(i, j)$.

☐ 37 – Expliquer comment retrouver l'ordre optimal des multiplications à effectuer à partir du tableau contenant les $C(i, j)$.

☐ 38 – Écrire une fonction `fmatrix mult_matrices(fmatrix *m, int *n, int ell, int **C, int i, int j)` qui prend en entrée ℓ matrices et le tableau calculé par la fonction précédente calcule le produit $M_i \dots M_j$ en utilisant la méthode optimale.

3 Scrabble – OCaml (20 pts)

Dans cette section, nous allons nous intéresser à un programme qui joue au Scrabble.
Chaque lettre de l'alphabet latin ('A','B',...,'Z') est associé à un nombre de points allant de 1 à 10.
Étant donné un ensemble de 7 lettres, on cherche à faire le mot valide valant le plus de points. Si on arrive à placer ses 7 lettres, on gagne un bonus de 50 points.
Nous allons ici ignorer le placement du mot sur le plateau.
Pour savoir si un mot est valide, nous disposons d'un fichier `dictionnaire.txt` qui contient la liste de tous les mots valides, séparés par des espaces (on suppose la présence d'un espace supplémentaire en première et dernière position, de telle manière que chaque mot est entouré par deux espaces). Nous affectons son contenu à une variable `dict : string`, dont on peut accéder au i-ième caractère par `dict.[i]`.

```
let f = open_in "dictionnaire.txt"
let dict = input_line f
let () = close_in f
```

Dans un premier temps, intéressons nous à la possibilité de faire un Scrabble, c'est-à-dire de placer les 7 lettres. Il faut donc chercher dans le dictionnaire tous les mots contenant nos 7 lettres.

☐ 39 – Dans le pire des cas, combien peut-on former de mots (pas nécessairement valides) différents que l'on doit tester avec 7 lettres ?

Nous allons adapter l'algorithme de Rabin-Karp pour trouver aussi les anagrammes d'un motif, c'est-à-dire un facteur du mot qui contient les mêmes lettres que le motif, mais pas nécessairement dans le même ordre. Pour ceci, on propose de donner un code à chaque caractère ainsi : '␣' -> 0, 'A' -> 1, 'B' -> 2... 'Z' -> 26.

```
let code c =
  if c = ' ' then 0 else
    begin
      assert ('A' <= c && c <= 'Z');
      int_of_char c - int_of_char 'A' + 1
    end
```

On propose d'utiliser la fonction de hachage suivante sur les mots de taille arbitraire :

$$h : c_0, \dots, c_n \mapsto \prod_{i=0}^{n} p_{\bar{c}_i},$$

où p_k désigne le k-ième nombre premier, et $\bar{c}$ désigne le code du caractère c.

☐ 40 – Montrer que deux mots $\mathbf{b}$ et $\mathbf{c}$ sont des anagrammes si et seulement si $h(\mathbf{b}) = h(\mathbf{c})$.

On suppose à présent qu'on dispose d'un tableau $\mathbf{p}$ contenant les 27 premiers nombres premiers :

```
let p = [|
     2; 3; 5; 7;11;
     13;17;19;23;29;
     31;37;41;43;47;
     53;59;61;67;71;
     73;79;83;89;97;
     101;103|]
```

☐ 41 – Écrire une fonction `hash : string -> int -> int` qui prend en entrée une chaîne de caractère $\mathbf{c}$ et un entier n et qui calcule $h(c_0, \dots, c_{n-1})$.
Cette fonction vérifiera que la taille de $\mathbf{c}$ est au moins n.

☐ 42 – Si on trouve un facteur $\mathbf{b} = b_0 \dots b_6$ du dictionnaire tel que $h(\mathbf{b}) = h(\mathbf{c})$, peut-on être sur de l'existence d'un anagramme de $\mathbf{c}$ dans le dictionnaire ? Justifier.

☐ 43 – Paul pense que vérifier l'égalité des images par h n'est pas suffisant et propose de tester que le motif contienne bien deux espaces en considérant les anagrammes de $\mathbf{c}' = c_0 \dots c_6$. Si on trouve un facteur $\mathbf{b} = b_0 \dots b_8$ du dictionnaire tel que $h(\mathbf{b}) = h(c_0 \dots c_6)$, peut-on être sur de l'existence d'un anagramme de $\mathbf{c}$ dans le dictionnaire ? Justifier.

☐ 44 – Écrire une fonction `verifie : 'a -> 'a -> string -> int -> int -> bool` qui prend en argument la valeur $h(\mathbf{c})$ pour un certain motif $\mathbf{c}$ de taille n, la valeur $h(b_i, b_{i+n+1})$, le dictionnaire $\mathbf{b}$, la taille du motif n et la position i et qui vérifie si on a bien un anagramme de $\mathbf{c}$ présent en position $i+1$ dans le dictionnaire $\mathbf{b}$.
remarque : la fonction ne prend pas le motif $\mathbf{c}$ *en argument.*

☐ 45 – Peut-on avoir un dépassement de capacité lorsque l'on calcule une image par h d'un facteur de taille 9 ? Pour la suite, on supposera que l'on a jamais de dépassement de capacité.

On propose l'implémentation suivante à compléter d'une variante de l'algorithme de Rabin-Karp :

variante de Rabin-Karp

```
let rabin_karp motif texte =
  let n = String.length motif in
  let h = hash motif n in
  let emp = hash texte n in
  let rec parcours emp i =
    if i > String.length texte - n then
      false
    else if verifie h emp texte n i then
      true
    else
      let nouv_emp = (* A COMPLETER *) in
      parcours nouv_emp (i+1)
  in
  parcours emp 0
```

☐ 46 – Compléter l'algorithme en remplaçant le commentaire.

☐ 47 – Quelle est la complexité de cet algorithme ?

☐ 48 – Peut-on adapter cette technique pour chercher aussi les anagrammes de sous-ensembles de 6 lettres de notre tirage ? Est-ce avantageux ?

Fin de l'épreuve