Devoir surveillé n°3

Durée : 2 heures, calculatrices et documents interdits

I. Sélection du $(k+1)^{\text{e}}$ 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 $L = [9; 1; 2; 4; 7; 8]$ alors le 1^{er} plus petit élément de L est 1 et 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.
Étant donné un réel a, on note $\lfloor a \rfloor$ le plus grand entier inférieur ou égal à a.

Important. Dans cette partie, les fonctions demandées ne doivent faire intervenir aucun trait impératif du langage OCaml (références, tableaux, boucles, etc.).

A. Quelques fonctions utiles

1) Écrire une fonction récursive de signature

```
longueur : 'a list -> int
```

qui calcule la longueur de la liste passée en paramètre.

2) 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`.

3) 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.

4) 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 [3;2;6;4;1;15] 3` est égal à 4.

5) 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]]`.

6) 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 $a_0 \leqslant a_1 \leqslant \cdots \leqslant a_{n-1}$, l'élément médian désigne $a_{\lfloor n/2 \rfloor}$.
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;2];[5;1;2;4]]`↙ est égal à `[3;3;2;4]`.

7) 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 petits que `p`, `l2` est la liste des éléments de `l` strictement plus grands que `p`, `n1` et `n2` sont respectivement les longueurs de `l1` et `l2`.

B. La fonction de sélection et sa complexité

On propose de mettre en œuvre l'algorithme suivant pour sélectionner le $(k+1)^e$ plus petit élément d'une liste ℓ.

```
1  Fonction SELECTION L k :
2  /* L est une liste, k est un entier positif
3  Début
4      n ← LONGUEUR L
5      Si n ⩽ 5 Alors
6          M ← TRI_INSERTION L
7          Renvoyer l'élément de rang k de M
8      Sinon
9          L_Cinq ← PAQUETS_DE_CINQ L
10         M ← MEDIANS L_Cinq
11         pivot ← SELECTION M ((n + 4)//5)//2
12         /* L'opérateur // désigne le quotient d'entiers.
13         /* Le rang ((n + 4)//5)//2 correspond au rang du médian de la liste M
14         L_1, L_2, n_1, n_2 ← PARTAGE pivot L
15         Si k < n_1 Alors
16             Renvoyer SELECTION L_1 k
17         Sinon
18             Renvoyer SELECTION L_2 (k-n_1)
19         FinSi
20     FinSi
21 Fin
```

8) 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 précédemment décrit.

9) 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. Une analyse de l'algorithme fournit la relation suivante, que nous admettons :

$$\forall n \geqslant 55, T(n) \leqslant T\left(\left\lfloor \frac{n+4}{5} \right\rfloor\right) + T\left(\left\lfloor \frac{8n}{11} \right\rfloor\right) + 4n.$$

Montrer que $\forall n \geqslant 1, T(n) \leqslant (200 + T(55))n$.

On pourra remarquer, sans démonstration particulière, que T est une fonction croissante.

II. Études des classes sylvestres

Étant donné un arbre binaire de recherche T, sa classe sylvestre est l'ensemble des mots qui donnent l'arbre T après insertion dans l'arbre vide.

L'objectif de cet exercice est de donner une caractérisation et une description de cet ensemble. On commence par rappeler la structure des arbres binaires de recherche ainsi que leurs propriétés usuelles. Puis, on introduit la notion de S-équivalence sur les mots qui permet de caractériser la classe sylvestre d'un arbre T. Enfin, à l'aide du produit de mélange, on présente un algorithme calculant la classe sylvestre d'un arbre donné.

Déf. 1 (Alphabet) Un *alphabet fini* est un ensemble fini non vide Σ de lettres. Un *mot sur* Σ une suite finie d'éléments de Σ. Par exemple, *abbcab* est un mot sur l'alphabet $\{a, b, c\}$. On note $\Sigma^\star$ l'ensemble des mots sur Σ.

Enfin, pour tout mot w sur Σ, on pourra noter $|w|$ sa longueur (i.e son nombre de lettre).
Dans toute la suite, Σ désigne un alphabet fini totalement ordonné ne contenant pas ε. Le symbole ε désigne le mot vide (i.e le mot de longueur 0).

Algorithme d'insertion dans un arbre binaire

Déf. 2 (Arbre binaire) Un *arbre binaire* T (étiqueté par les éléments de Σ) est récursivement soit :
- l'arbre vide que l'on note $\circ$;
- un triplet (T_g, r, T_d) où r est un élément de Σ, T_g et T_d des arbres binaires. Les éléments r, T_g et T_d sont respectivement appelés racine, sous-arbre gauche et sous-arbre droit de T.

Déf. 3 (Arbre binaire de recherche) Un *Arbre Binaire de Recherche* (abrégé en **ABR**) T est récursivement soit :
- l'arbre vide ;
- un triplet (T_g, r, T_d) où r est un élément de Σ, T_g et T_d des **ABR**. De plus, toute valeur apparaissant dans T_g est strictement inférieure à r et toute valeur apparaissant dans T_d est supérieure ou égale à r.

Déf. 4 (Insertion dans un arbre binaire) L'insertion d'un élément a de Σ dans un arbre binaire T est une opération que l'on note $T \leftarrow a$ définie récursivement comme suit :
- si $T = \circ$, alors $T \leftarrow a = (\circ, a, \circ)$;
- si $T = (T_g, r, T_d)$ et $r \leqslant a$, alors $T \leftarrow a = (T_g, r, T_d \leftarrow a)$;
- si $T = (T_g, r, T_d)$ et $r > a$, alors $T \leftarrow a = (T_g \leftarrow a, r, T_d)$.

On définit récursivement alors l'insertion d'un mot w dans un arbre binaire T noté également $T \leftarrow w$ comme suit :
- si $w = \varepsilon$, alors $T \leftarrow w = T$;
- si $w = av$ avec $a \in \Sigma$, alors $T \leftarrow w = (T \leftarrow a) \leftarrow v$.

Dans le cas où T est un **ABR**, on admet que $T \leftarrow a$ et $T \leftarrow w$ sont également des **ABR**.
Étant donné un mot w sur Σ, *l'arbre binaire de recherche associé* à w est l'arbre $\circ \leftarrow w$.

Déf. 5 (Classe sylvestre) Soit T un arbre binaire de recherche. La *classe sylvestre* de T que l'on note $\mathrm{Syl}(T)$ est l'ensemble des mots w vérifiant : $(\circ \leftarrow w) = T$.
Par exemple, si $T = ((\circ, a, \circ), b, (\circ, c, \circ))$, on a $\mathrm{Syl}(T) = \{bac, bca\}$.
Et si $T = \circ$, alors $\mathrm{Syl}(T) = \{\varepsilon\}$.

Dans la suite, les lettres de Σ sont représentées en Ocaml par des `char` et les mots sur l'alphabet Σ par des `char list`.
Ainsi, le mot w="arbre" est représenté par la liste `w = ['a'; 'r'; 'b'; 'r'; 'e']`.

On représente les arbres binaires en Ocaml à l'aide de la structure arbre suivante :

```
type 'a arbre = Vide | Noeud of 'a arbre * 'a * 'a arbre
```

Ainsi, l'arbre $T = (\circ, a, (\circ, b, o))$ est représenté en Ocaml par :

```
Noeud (Vide, 'a', Noeud (Vide, 'b', Vide))
```

1) Représenter le graphe de l'**ABR** associé au mot "fantastique", l'ordre sur les lettres étant l'ordre alphabétique.

2) Écrire une fonction récursive en Ocaml de signature :

```
insertion_lettre : char -> char arbre -> char arbre
```

et telle que `insertion_lettre a t` est l'arbre binaire obtenu en insérant la lettre a dans l'arbre binaire t.

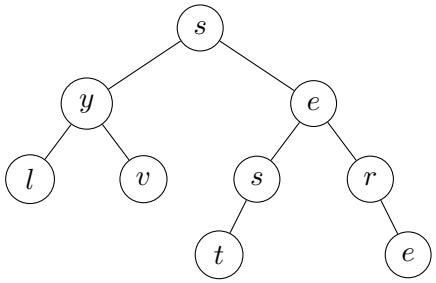
3) Écrire une fonction récursive en Ocaml de signature :

```
insertion_mot : char list -> char arbre -> char arbre
```

et telle que `insertion_mot w t` est l'arbre binaire obtenu en insérant le mot w dans l'arbre binaire t.

Déf. 6 (Lecture préfixe) Soit T un arbre binaire. La lecture préfixe de T que l'on note w_T est le mot défini récursivement par :
- si $T = 0$, alors $w_T = \varepsilon$;
- si $T = (T_g, r, T_d)$, alors $w_T = r w_g w_d$ où w_g et w_d sont les lectures préfixes respectives de T_g et de T_d

4) Expliciter w_T pour l'arbre binaire T représenté ci-dessous :



5) L'arbre précédent est-il un **ABR** ?

6) Écrire une fonction récursive en Ocaml de signature :

```
prefixe : char arbre -> char list
```

et telle que `prefixe t` est le mot qui correspond à la lecture préfixe de l'arbre t.

7) Soit T un **ABR**, soit w_T la lecture préfixe de T. Montrer que T est l'**ABR** associé à w_T.

Une relation d'équivalence sur les mots

Déf. 7 (S-adjacence) Soient u et v deux mots sur Σ. On dit que u est *S-adjacent* à v (ou que u et v sont S-adjacents) s'il existe trois lettres $\mathbf{a} < \mathbf{b} \leqslant \mathbf{c}$ de Σ et trois mots $\mathrm{f}_1, \mathrm{f}_2, \mathrm{f}_3$ sur Σ vérifiant :

$$(u = \mathrm{f}_1\mathbf{b}\mathrm{f}_2\mathbf{ac}\mathrm{f}_3 \text{ et } v = \mathrm{f}_1\mathbf{b}\mathrm{f}_2\mathbf{ca}\mathrm{f}_3) \text{ ou } (u = \mathrm{f}_1\mathbf{b}\mathrm{f}_2\mathbf{ca}\mathrm{f}_3 \text{ et } v = \mathrm{f}_1\mathbf{b}\mathrm{f}_2\mathbf{ac}\mathrm{f}_3)$$

Déf. 8 (S-équivalence) Soient u et v deux mots sur Σ. On dit que u est S-équivalent à v (ou que u et v sont S-équivalents) s'il existe $n \in \mathbb{N}, (w_0, w_1, \ldots, w_n) \in (\Sigma^\star)^{n+1}$ vérifiant :

$$u = w_0, v = w_n, \forall i \in \{0, 1, \ldots, n-1\}, w_i \text{ est S-adjacent à } w_{i+1}$$

On admet que la relation "être S-équivalent à" est bien une relation d'équivalence sur $\Sigma^\star$.

8) Soient a, b, c trois lettres de Σ vérifiant $a < b \leqslant c$. Montrer que pour tout arbre binaire de recherche T contenant au moins la lettre b, on a : $T \leftarrow ac = T \leftarrow ca$. On pourra raisonner par récurrence sur le nombre de nœuds de T.

9) Montrer que si deux mots u et v sont S-équivalents, alors les $\mathbf{ABR}(\circ \leftarrow u)$ et $(\circ \leftarrow v)$ sont égaux.

On peut montrer que la réciproque est vraie, et que deux mots sont donc S-équivalents si et seulement si ils sont des éléments d'une même classe sylvestre.

On admet ce résultat pour la suite de l'exercice.

Construction de classes sylvestres

Pour construire des classes sylvestres, il est utile d'utiliser le produit de mélange.

Déf. 9 (Mélange) Soient u et v deux mots sur Σ. Le mélange de u et v, noté $u \sqcup v$, est l'ensemble récursivement défini comme suit :

— si $v = \varepsilon$, alors $u \sqcup v = \{u\}$;

— si $u = \varepsilon$, alors $u \sqcup v = \{v\}$; si $u = au'$ et $v = bv'$ où a et b sont des lettres, u' et v' des mots, alors :

$$u \sqcup v = \{aw \mid w \in u' \sqcup v\} \cup \{bw \mid w \in u \sqcup v'\}$$

Déf. 10 (Langage) Un *langage* sur l'alphabet Σ est un ensemble de mots sur Σ. Autrement dit, un langage sur Σ est une partie de $\Sigma^\star$.

Déf. 11 (Mélange de langages) Soient L et M deux langages. Le mélange des langages L et M, noté $L \sqcup M$, est défini par :

$$L \sqcup M = \bigcup_{u \in L, v \in M} u \sqcup v$$

Si au moins un des deux langages est égal à l'ensemble $\varnothing (\neq \{\varepsilon\})$, alors on a $L \sqcup M = \varnothing$. Étant donné un $\mathbf{ABR}$ $T = (T_g, r, T_d)$, on peut montrer que :

$$\mathrm{Syl}(T) = \{rw \mid w \in \mathrm{Syl}\,(T_g) \sqcup \mathrm{Syl}\,(T_d)\}$$

On admet ce résultat pour la suite de cette sous-partie. On note que $\mathrm{Syl}(\circ) = \{\varepsilon\}$.

10) Expliciter sans justification tous les éléments de l'ensemble $abba \sqcup ba$.

11) Écrire une fonction récursive en Ocaml de signature :

```
ajout_lettre : char -> char list list -> char list list
```

et telle que `ajout_lettre a liste` est la liste de mots de la forme `a::w` où w est un élément de `liste`.

12) Écrire une fonction récursive en Ocaml de signature :

```
shuffle : char list-> char list -> char list list
```

et telle que `shuffle u v` est une liste contenant exactement tous les mots de $u \sqcup v$ avec répétitions possibles d'un même mot. On devra utiliser la fonction auxiliaire `ajout_lettre` et l'opérateur de concaténation `@`.

13) Écrire une fonction récursive en Ocaml de signature :

```
shuffle_l : char list list -> char list list -> char list list
```

et telle que `shuffle_l l m` est une liste contenant exactement tous les mots de $L \sqcup M$ où la liste l (respectivement m) représente le langage fini L (respectivement M), avec répétitions possibles d'un même mot. Seules les fonctions définies précédemment peuvent être utilisées en fonctions auxiliaires.

14) Écrire une fonction récursive en Ocaml de signature :

```
sylvestre_T : char arbre -> char list list
```

et telle que `sylvestre_T t` est une liste contenant exactement tous les éléments de la classe sylvestre de t. Seules les fonctions définies précédemment peuvent être utilisées en fonctions auxiliaires.

— FIN —