Quatrième devoir non surveillé

Durée : quatre heures.
Le sujet comporte deux problèmes, le premier étant plus simple que le second.

1 Appel

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.

1.1 Dans une liste

1. Ecrire 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 ?

1.2 Dans un tableau

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. Ecrire une fonction `asseoir : int list → int → int array`, qui prend en argument une liste non vide d'entiers distincts et un entier n, et renvoie un tableau représentant une salle de classe pour n élèves où chaque élève de la liste à 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 n et k) de la fonction précédente ?

1.3 Plan de classe

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

8. On considère la fonction `place : int array → unit` suivante :

```
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 array → 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` ?

2 Parcours de la toile

Le World Wide Web, ou Web, est un ensemble de pages Web (identifiées de manière unique par leurs adresses Web, ou URL pour Uniform Resource Locators) reliées les unes aux autres par des hyperliens. Le Web est souvent modélisé comme un graphe orienté dont les sommets sont les pages Web et les arcs les hyperliens entre pages. Compte tenu de la taille du Web, on s'intéresse à des sous-graphes du Web obtenus en naviguant sur le Web, c'est-à-dire en le parcourant page par page, en suivant les hyperliens d'une manière bien déterminée. Ce parcours du Web pour en collecter des sous-graphes est réalisé de manière automatique par des logiciels autonomes appelés Web crawlers ou crawlers en anglais, ou collecteurs en français.

2.1 Fonctions utilitaires

Nous allons tout d'abord coder certaines fonctions de manipulation de structures de données de base, qui seront utiles dans le reste de l'exercice.

1. Coder une fonction `aplatir : (a * a list) list → a list`, telle que, si `liste` est une liste de couples $[(x_1, l_{x_1}); \ldots; (x_n, l_{x_n})]$, où chaque x_i est un élément de type `'a`, et l_{x_i} une liste déléments de type `'a` de la forme $[y_{i1}; \ldots; y_{ik_i}]$, `aplatir liste` est une liste déléments de type `'a`

$$[x_1; y_{11}; \ldots; y_{1k_1}; x_2; y_{21}; \ldots; y_{2k_2}; \ldots; x_n; y_{n1}; \ldots; y_{nk_n}].$$

   On va utiliser dans la suite de l'exercice un type de données `dictionnaire` persistant dont les clefs serons des chaînes de caractères et les valeurs des entiers. À chaque clef présente dans le dictionnaire est associée une seule valeur.

   Les fonctions suivantes sont supposées être prédéfinies :

   - `dictionnaire_vide : dictionnaire`.
     Il s'agit d'une constante représentant un dictionnaire vide.
   - `ajout : string → int → dictionnaire → unit`.
     L'appel `ajout clef valeur dict` renvoie le dictionnaire `dict` dans lequel le couple $(clef, valeur)$ a été ajouté.
   - `contient : string → dictionnaire → bool`.
     L'appel `contient clef dict` renvoie un booléen indiquant s'il y a un couple dont la clef est $clef$ dans le dictionnaire `dict`.
   - `valeur : string → dictionnaire → int`.
     L'appel `valeur clef dict` renvoie la valeur associée à la clef $clef$ dans le dictionnaire `dict`. Cette fonction ne peut être appelée que si la clef $clef$ est présente dans le dictionnaire.

   On suppose pour la suite de l'exercice que le type de données `dictionnaire` est prédéfini ; on ne demande pas de l'implémenter. Les fonctions `ajout`, `valeur`, et `contient` sont supposées s'exécuter en temps moyen $O(\log n)$ où n est le nombre de valeurs dans le dictionnaire.

2. Coder `numerotation : string list → string list * dictionnaire`, qui est telle que `numerotation liste` renvoie un couple $(liste', numero)$ où :

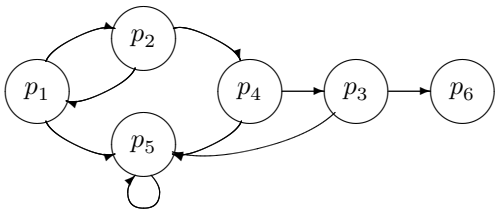
   - $liste'$ est la liste des chaînes de caractères de $liste$ distinctes (c'est-à-dire les éléments de $liste$ sans doublon), dans l'ordre de leur première occurrence dans $liste$.
   - $numero$ associe à chaque chaîne de caractères dans $liste'$ sa position dans $liste'$ (en numérotant à partir de 0).

   Par exemple l'appel `unique ["x";"zz";"x";"x";"zz";"yt"]` renvoie le couple formé de la liste `["x";"zz";"yt"]` et d'un dictionnaire associant à `"x"` la valeur `0`, à `"zz"` la valeur `1` et à `"yt"` la valeur `2`.
3. Quelle est la complexité de la fonction `numerotation` en terme de la longueur n de la liste $liste$ en argument et du nombre m déléments distincts dans la liste $liste$? Justifier la réponse.

2.2 Crawler

Nous allons maintenant implémenter un crawler simple en Caml. On suppose fournie une fonction `recupere_liens : string → string list` prenant en argument l'URL d'une page Web p et renvoyant la liste des URL des pages q pour lesquelles il existe un hyperlien de p à q.

Pour illustrer le comportement de cette fonction, nous considérons un exemple de mini-graphe du Web à six pages et neuf hyperliens comme suit :



Dans cette représentation, p_1, p_2, etc., sont les URL de pages Web , et les arcs orientés représentent les hyperliens entre pages Web.

On supposera pour les exemple que la fonction `recupere_liens` renvoie les URL dans l'ordre croissant de leurs indices dans l'énumération $(p_1, \ldots, p_6)$. Par exemple un appel à `recupere_liens p1` retourne la liste `[p2; p5]` (et non `[p5; p2]`).

Un crawler est un programme qui, à partir d'une URL, parcourt le graphe du Web en visitant progressivement les pages dont les liens sont présents dans chaque page rencontrée. Le processus sarrête quand une condition est atteinte (par exemple, un nombre fixé de pages ont été visitées). Le résultat renvoyé par le crawler, que l'on définira plus précisément plus loin, est appelé un crawl.

2.2.1 Files d'attente

On propose ici de programmer un type de files d'attentes de chaînes de carcatères persistantes. Une file d'attente sera représentée par deux listes. C'est dans la première, liste des «entrée», que les nouvelles valeurs seront insérées. Et c'est de la seconde, liste des « sorties », que les anciennes valeurs seront extraites. Lorsque la liste des sorties est vide et qu'on souhaite extraire un élément de la file, on commence par remplacer la liste de sortie par la liste des entrées retournée.

On utilise le type Caml suivant :

```
type file = {entree : string list; sortie : string list};;
```

1. Créer la constante `file_vide`.
2. Programmer `enfiled` de type `string -> file -> file`.
3. Programmer `defiled` de type `file -> string*file`.
4. Programmer une fonction `liste_enfilee` de type `string list -> file -> file` telle que pour toute liste de chaînes de caractères `l` et toute file `f`, `liste_enfilee l f` renvoie la file obtenue en enfilant dans `f` les éléments de `l`. La tête de `l` devra être enfilée en premier.

2.2.2 Parcours en largeur

À présent on veut coder un crawler qui parcourt en priorité les pages proches de la page de départ. Ainsi on parcourra en premier la page u elle-même, puis les pages à distance 1 de u, puis celles à distance 2, etc.

La fonction prendra en entrée un nombre n de pages et une URL u et renverra en sortie une liste de longueur au plus n. Cette liste sera formés de couples : (page visitée, listes des pages auxquelles on peut accéder depuis celle-ci).

1. Indiquer le résultat sur le mini-graphe exemple d'un parcours en largeur de longueur 4 en partant de p_1.
2. Programmer une fonction `crawlerEnLargeur` de type `int -> string * string list` implémentant cette méthode de parcours du web.

   Indication : La différence par rapport au cas des arbres vu en cours est qu'on peut accéder à une même page par différents chemins. Il faut donc éviter de traiter plusieurs fois une même page. Pour ce, utiliser un dictionnaire `deja_vu` pour enregistrer les pages déjà traitées.
3. (Bonus:) Programmer une version impérative, en utilisant le module `Queue` de Ocaml pour la file des pages à visiter, et `Hashtbl` pour le dictionnaire des pages déjà visitées.

2.3 Calcul de PageRank

PageRank est une manière d'affecter un score à chaque page du Web, imaginée par Sergey Brin et Larry Page, les fondateurs du moteur de recherche Google. L'introduction de PageRank a révolutionné la technologie des moteurs de recherche sur le Web. Nous allons maintenant implémenter le calcul de PageRank.

Étant donnée une partie du Web (où l'ensemble des pages est indexé entre 0 et $n-1$), la matrice de surf aléatoire dans cette partie du Web est la matrice g de taille $n \times n$ définie comme suit :

- s'il n'y a aucun lien depuis une page Web d'indice i, alors pour tout j, $m_{ij} := 1/n$.
- Sinon, s'il y a k_i liens depuis la page Web d'indice i, alors pour tout j, on a $m_{ij} := (1-d) \times g_{ij}/k_i + d/n$, où g_{ij} est le nombre de liens depuis la page d'indice i vers la page d'indice j et d est un nombre réel fixé appartenant à $[0,1]$ (on prend souvent $d = 0{,}15$).

Cette matrice peut être vue comme décrivant la marche aléatoire d'un surfeur sur le Web. À chaque fois que celui-ci visite une page Web :

- Si cette page ne comporte aucun lien, il visite une page Web arbitraire, choisie aléatoirement de façon uniforme.
- Si cette page comporte au moins un lien, il visite avec une probabilité égale à $1-d$ un des liens sortants de cette page, et avec une probabilité égale à d une page Web arbitraire, choisie aléatoirement de façon uniforme.

1. Écrire une fonction prenant en entrée un crawl et renvoyant la matrice `g` de format $n \times n$ (n étant le nombre de pages visitées) tel que pour tout $(i,j) \in [\![0,n[\![^2$, `g.(i).(j)` est le nombre de liens de la page i vers la page j. Cette matrice est appelée « matrice d'adjacence » de la portion de web concernée.
   Les pages seront numérotées par la fonction `numerotation` ; autrement dit si `numeros` est le dictionnaire renvoyé par celle-ci, alors le numéro d'une page `p` est `valeur p numeros`.
   Indication : On peut utiliser une fonction auxiliaire pour parcourir le crawl (puisque c'est une liste) et une autre pour parcourir les liens issus d'une page (puisque c'est encore une liste).
2. Coder une fonction `surf_aleatoire : float → int array array → float array array` telle que si d est un nombre entre 0 et 1, et si g est la matrice d'adjacence d'un sous-graphe partiel du Web, alors `surf_aleatoire d g` renvoie la matrice m de surf aléatoire dans ce sous-graphe.
3. Coder `multiplication : float array → float array array → float array`, une fonction prenant en argument un vecteur ligne v de taille n et une matrice m de taille $n \times n$ et renvoyant le vecteur ligne w de taille n résultant du produit de v par la matrice m : $w = vm$. En d'autres termes, pour tout j, $w_j = \sum_i v_i m_{ij}$.
4. Étant donnés deux vecteurs $v1$ et $v2$ de même longueur n, on définit leur distance, notée $d(v1, v2)$ ainsi : $d(v1, v2) = \sum_{i=0}^{n-1} |v1.(i) - v2.(i)|$.
   Coder une fonction pour calculer la distance entre deux vecteurs.

   On prouve par la formule des probabilités totales qui si v est le vecteur contenant les probabilités pour le surfeur d'être sur chacune des pages du Web à un instant donné, alors vm contient ces mêmes probabilités un clic plus tard.
   Le PageRank des pages d'un sous-graphe du Web à n pages se calcule par des multiplications successives d'un vecteur ligne par la matrice de surf aléatoire m de ce sous-graphe. Plus précisément, soit θ un nombre réel strictement positif (par exemple, $\theta = 10^{-4}$) et soit $v^{(0)}$ le vecteur ligne de taille n dont toutes les composantes valent $1/n$. On pose, pour tout entier naturel p, $v^{(p)} := v^{(0)}m^p$.
   Ainsi, $v^{(p)}$ contient les probabilité pour un surfeur aléatoire d'être sur chacune des page au bout de p clics s'il est parti initialement d'une page au hasard uniforme.
   L'algorithme de PageRank calcule la suite des $v^{(p)}$ pour $p = 0, 1, \ldots$ jusqu'à ce que $d(v^{(p+1)}, v^{(p)}) \leq \theta$ et renvoie alors le vecteur $v^{(p+1)}$, considéré comme le vecteur des scores de PageRank. On peut montrer (à l'aide du théorème de Perron-Frobenius) que l'algorithme termine dès que θ est strictement positif.
   PageRank est utilisé pour affecter un score d'importance aux pages du Web. Le vecteur de scores v retourné par l'algorithme de PageRank donne dans v_i le score d'importance de la page d'indice i. Les pages de plus haut score de PageRank sont considérées comme les plus importantes.

5. Coder `pagerank : float → float array array → float array`, une fonction prenant en argument un nombre $\theta > 0$ et une matrice M de surf aléatoire d'un sous-graphe du Web et renvoyant le vecteur des scores de PageRank pour θ et m. La fonction `pagerank` devra faire appel à la fonction `multiplie` précédemment codée.

6. Coder une fonction `calcule_pagerank : float → float → (string * string list) list → (string * float) list` telle que `calcule_pagerank d theta crawl` renvoie une liste de couples (u, s), un couple pour chaque URL découverte dans le crawl crawl, triée par valeur décroissante de s, où u est l'URL de cette page et s son score de PageRank. Ici, d et θ sont les deux paramètres nécessaires au calcul de la matrice de surf aléatoire et du PageRank respectivement. On pourra coder une fonction `tri_fusion` et utiliser l'ensemble des fonctions développées dans les questions précédentes.