Épreuve d'informatique (Mines 2016)

Corrigé par Jean-Pierre Becirspahic (jp.becir@info-llg.fr)

Partie I. Graphe du Web

Fonctions utilitaires

Question 1.

```
let aplatir lst = list_it (fun (x, l) b -> (x::l) @ b) lst [] ;;
```

Question 2. On rédige d'abord une fonction qui partage une liste de taille n en deux sous-listes de tailles respectives $\lceil n/2 \rceil$ et $\lfloor n/2 \rfloor$:

```
let rec scission = function
  | []     -> [], []
  | [a]    -> [a], []
  | a::b::q -> let l1, l2 = scission q in a::l1, b::l2 ;;
```

Cette fonction est de type *'a list -> 'a list * 'a list*.
On rédige ensuite une fonction qui fusionne deux listes de couples triées suivant leur seconde composante :

```
let rec fusion l1 l2 = match (l1, l2) with
  | [], _                          -> l2
  | _, []                          -> l1
  | a::q, b::_ when snd a > snd b -> a::(fusion q l2)
  | _, b::q                        -> b::(fusion l1 q) ;;
```

Cette fonction est de type *('a * 'b) list -> ('a * 'b) list -> ('a * 'b) list*.
Vient enfin la fonction de tri à proprement parler :

```
let rec tri_fusion = function
  | [] -> []
  | [a] -> [a]
  | l   -> let l1, l2 = scission l in fusion (tri_fusion l1) (tri_fusion l2) ;;
```

Les fonctions **`scission`** et **`fusion`** sont de complexité linéaire donc la complexité $C(n)$ de la fonction **`tri_fusion`** vérifie une relation de type : $C(n) = C(\lceil n/2 \rceil) + C(\lfloor n/2 \rfloor) + \Theta(n)$, ce qui conduit à $C(n) = \Theta(n \log n)$ d'après le théorème maître.

Question 3.

```
let unique lst =
  let rec aux n (l, d) = function
    | []                      -> (l, d)
    | s::q when contient s d -> aux n (l, d) q
    | s::q                    -> aux (n+1) (s::l, ajoute s n d) q
  in
  let l, d = aux 0 ([], dictionnaire_vide ()) lst
  in rev l, d ;;
```

La fonction auxiliaire **`aux`** utilise deux accumulateurs : **`n`** désigne la longueur de la liste **`l`** et **`(l, d)`** la liste *list'* et le dictionnaire *dict* en cours de construction. À la fin de l'exécution de cette fonction la liste *list'* est obtenue en sens inverse, ce qui nécessite l'usage de la fonction **`rev`** pour l'obtenir dans le sens souhaité.

Question 4. On peut majorer le coût des fonctions **`contient`** et **`ajoute`** par un $O(\log m)$ donc la complexité de la fonction **`aux`** est majorée par un $O(n \log m)$. Le coût de la fonction **`rev`** est un $\Theta(n)$ donc la complexité de la fonction **`unique`** est un $O(n \log m)$.

Crawler simple

Question 5.

```
let crawler_bfs n p =
  let rec aux acc dejavu n lst = match (n, lst) with
    | 0, _                          -> rev acc
    | _, []                         -> rev acc
    | _, p::q when contient p dejavu -> aux acc dejavu n q
    | _, p::q                       -> let l = recupere_liens p in
                                       aux ((p, l)::acc) (ajoute p n dejavu) (n-1) (q @ l)
  in aux [] (dictionnaire_vide ()) n [p] ;;
```

La fonction **aux** prend quatre paramètres : **acc**, de type *(string * string list) list*, stocke les pages déjà trouvées lors de l'exploration ; **n** est un entier qui dénote le nombre de pages restant à découvrir ; **dejavu** est un dictionnaire qui stocke les pages déjà rencontrées ; enfin **lst** est une liste de pages à explorer.
Les deux premiers motifs correspondent à la terminaison de cette fonction : soit parce que le nombre de pages requis est atteint, soit parce qu'il n'y a plus de pages à explorer. Les liens trouvés s'accumulent en tête de liste, c'est pourquoi on renvoie l'image miroir de **acc**.
Le troisième motif correspond au cas d'une page déjà rencontrée ; on poursuit la recherche dans la queue de la liste **lst**.
Enfin, dans le quatrième motif on ajoute la page rencontrée ainsi que ses liens en tête de l'accumulateur, on ajoute p au dictionnaire **dejavu** et la liste de ses liens l en queue de la liste des pages qui restent à explorer, avant de poursuivre l'exploration.

On notera que le coût de ce dernier ajout pourrait être réduit en utilisant une file au lieu d'une liste pour faciliter l'insertion en queue de q.

Question 6. Le parcours en profondeur se déroule de la même manière ; il suffit d'insérer en tête de la liste q et non plus en queue la liste des liens trouvés sur la page p :

```
let crawler_dfs n p =
  let rec aux acc dejavu n lst = match (n, lst) with
    | 0, _                          -> rev acc
    | _, []                         -> rev acc
    | _, p::q when contient p dejavu -> aux acc dejavu n q
    | _, p::q                       -> let l = recupere_liens p in
                                       aux ((p, l)::acc) (ajoute p n dejavu) (n-1) (l @ q)
  in aux [] (dictionnaire_vide ()) n [p] ;;
```

Question 7. La fonction demandée utilise une fonction auxiliaire **remplir_matrice** pour remplir la matrice d'adjacence en fonction d'un dictionnaire associant un indice à chaque sommet et de la liste des voisinages de chaque sommet :

```
let rec remplir_matrice dict g = function
  | []        -> ()
  | (p, l)::s -> do_list (function q -> let i = valeur p dict and j = valeur q dict in
                                        g.(i).(j) <- g.(i).(j) + 1) l ;
                 remplir_matrice dict g s ;;
```

La fonction principale s'écrit alors :

```
let construit_graphe crawl =
  let l, dict = unique (aplatir crawl) in
  let n = list_length l in
  let g = make_matrix n n 0 in
  remplir_matrice dict g crawl ;
  (l, g) ;;
```

On commence par calculer la liste l des sommets distincts présents dans le *crawl* ainsi qu'un dictionnaire *dict* associant un numéro à chaque sommet. Une fois la matrice d'adjacence créée, il reste à la remplir avec la fonction précédente.

Calcul de PageRank

Question 8. On rédige d'abord une fonction qui calcule le nombre de liens depuis la page d'indice i :

```
let nb_liens g i =
  let k = ref 0 in
  do_vect (function x -> k := !k + x) g.(i) ;
  !k ;;
```

La fonction principale s'écrit alors :

```
let surf_aleatoire d g =
  let n = vect_length g in
  let m = make_matrix n n (1. /. (float_of_int n)) in
  for i = 0 to n-1 do
    let ki = nb_liens g i in
    if ki > 0 then
    for j = 0 to n-1 do m.(i).(j) <- d *. m.(i).(j) +.
                                     (1. -. d) *. (float_of_int g.(i).(j)) /. (float_of_int ki)
    done
  done ;
  m ;;
```

Question 9.

```
let multiplie v m =
  let n = vect_length v in
  let w = make_vect n 0. in
  for j = 0 to n-1 do
    for i = 0 to n-1 do
      w.(j) <- w.(j) +. v.(i) *. m.(i).(j)
    done
  done ;
  w ;;
```

Question 10. On définit tout d'abord une fonction qui calcule la distance entre deux vecteurs de même taille pour la norme $\|\cdot\|_1$:

```
let dist v w =
  let n = vect_length v in
  let s = ref 0. in
  for i = 0 to n-1 do s := !s +. abs_float (v.(i) -. w.(i)) done ;
  !s ;;
```

Puis la fonction demandée :

```
let pagerank theta m =
  let n = vect_length m in
  let rec aux v =
    let w = multiplie v m in
    if dist v w <=. theta then w else aux w
  in aux (make_vect n (1. /. (float_of_int n))) ;;
```

Question 11. La fonction **`zip`**, de type *'a list -> 'b vect -> ('a * 'b) list*, associe à une liste l et un vecteur v de même taille la liste (l_k, v_k) de leurs éléments de même rang :

```
let zip l v =
  let rec aux k = function
    | []   -> []
    | t::q -> (t, v.(k))::(aux (k+1) q)
  in aux 0 l ;;
```

Il reste à combiner les différentes fonctions écrites :

```
let calcule_pagerank d theta crawl =
  let (l, g) = construit_graphe crawl in
  let p = pagerank theta (surf_aleatoire d g) in
  tri_fusion (zip l p) ;;
```

Partie II. Automates probabilistes

Question 12. L'état initial n'est pas acceptant donc $\Pr(\varepsilon) = 0$.

Seul le chemin $q_0 \xrightarrow{\mathtt{0}} q_1$ est acceptant pour le mot $\mathtt{0}$, donc $\Pr(\mathtt{0}) = \dfrac{1}{4}$.

Les chemins acceptants pour le mot $\mathtt{010}$ sont $q_0 \xrightarrow{\mathtt{0}} q_0 \xrightarrow{\mathtt{1}} q_0 \xrightarrow{\mathtt{0}} q_1$ et $q_0 \xrightarrow{\mathtt{0}} q_1 \xrightarrow{\mathtt{1}} q_0 \xrightarrow{\mathtt{0}} q_1$ donc

$$\Pr(\mathtt{010}) = \frac{3}{4} \times 1 \times \frac{1}{4} + \frac{1}{4} \times 1 \times \frac{1}{4} = \frac{1}{4}.$$

Question 13. Nous allons montrer que $\displaystyle\sum_{\rho \text{ chemin pour } u} \Pr(\rho) = 1$, ce qui prouvera l'égalité demandée.

Comme le suggère l'énoncé, raisonnons par récurrence sur $|u|$.

- Si $|u| = 0$ alors $u = \varepsilon$ et le seul chemin pour u est le chemin ρ de longueur nulle, pour lequel $\Pr(\rho) = 1$.
- Si $|u| > 0$, on note $u = av$ où a est la première lettre de u. On note $Q_0 = \left\{q \in Q \;\middle|\; \Pr(q_0 \xrightarrow{a} q) > 0\right\}$, et, pour tout $q \in Q_0$, on considère l'automate $\mathcal{A}_q = (Q, q, F, \Pr)$.

  Un chemin pour u est un chemin $\rho_u = (q_0 \xrightarrow{a} q) \circ \rho_{v,q}$ où $q \in Q_0$ et $\rho_{v,q}$ est un chemin pour v dans l'automate $\mathcal{A}_q$. Par hypothèse de récurrence, on a pour tout $q \in Q_0$, $\displaystyle\sum_{\rho_{v,q}} \Pr(\rho) = 1$ donc :

$$\sum_{\rho_u} \Pr(\rho_u) = \sum_{q \in Q_0} \Pr(q_0 \xrightarrow{a} q) \times \sum_{\rho_{v,q}} \Pr(\rho_{v,q}) = \sum_{q \in Q_0} \Pr(q_0 \xrightarrow{a} q) = 1$$

  donc le résultat est bien acquis pour le mot u.

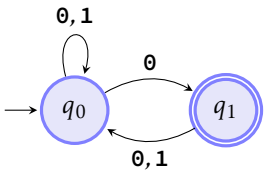
Question 14. Les mots u vérifiant $\Pr(u) = 0$ sont les mots pour lesquels il n'existe pas de chemin acceptant pour u ; ce sont les mots qui ne se terminent pas par le caractère $\mathtt{0}$ (donc le mot vide et les mots se terminant par $\mathtt{1}$).

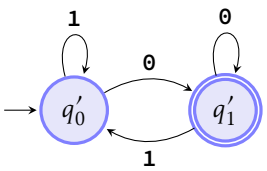
Les mots pour lesquels $\Pr(u) = 1$ sont les mots pour lesquels tout chemin pour u est acceptant dans l'automate ci-dessus ; il n'en n'existe pas car le chemin qui stationne en q_0 est un chemin non acceptant pour tout mot u.

Question 15. Les langage des mots u vérifiant $\Pr(u) > 0$ est le langage des mots qui se terminent par la lettre $\mathtt{0}$; il est décrit par l'expression rationnelle $(\mathtt{0} + \mathtt{1})^*\mathtt{0}$.

Question 16. À tout automate probabiliste $\mathcal{A}$ on associe l'automate non déterministe $\mathcal{A}' = (Q, \{q_0\}, F, \delta)$ où la fonction de transition δ est définie par $\delta(q, a) = \left\{q' \in Q \;\middle|\; \Pr(q \xrightarrow{a} q') > 0\right\}$. Alors $\Pr(u) > 0$ si et seulement si u est accepté par $\mathcal{A}'$, car pour tout chemin $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$, $\quad \Big(\Pr(\rho)) > 0 \iff \forall i \in [\![1, n]\!],\ \Pr(q_{i-1} \xrightarrow{a_i} q_i) > 0\Big)$.

Question 17. La construction suivante conduit à l'automate non déterministe dessiné à gauche, puis à sa déterminisation dessinée à droite :





avec $q'_0 = \{q_0\}$ et $q'_1 = \{q_0, q_1\}$.

Question 18. Si L est un langage rationnel, il existe un automate déterministe $\mathcal{A} = (Q, q_0, F, \delta)$ qui le reconnait. On définit alors l'automate probabiliste $\mathcal{A}' = (Q, q_0, F, \Pr)$ en posant $\Pr(q, a, q') = \begin{cases} 1 & \text{si } \delta(q, a) = q' \\ 0 & \text{sinon} \end{cases}$.

On a $u \in L \iff \Pr(u) = 1$, donc $L = \mathcal{L}_\eta(\mathcal{A}')$ pour tout $\eta \in [0, 1[$. L est bien stochastique.

Question 19. On calcule $\Pr(q_0 \xrightarrow{\texttt{1}} q_0 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{0}} q_0 \xrightarrow{\texttt{1}} q_1) = \frac{1}{2} \times \frac{1}{2} \times 1 \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{16}$ et on a $\frac{1}{16} = \underline{0,\texttt{0001}}_2$.

Question 20. Le seul chemin acceptant pour $\texttt{10}$ est $q_0 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{0}} q_1$ donc $\Pr(\texttt{01}) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} = \underline{0,\texttt{01}}_2$.

Question 21. Le mot $\texttt{1101}$ possède 5 chemins acceptants :

- $q_0 \xrightarrow{\texttt{1}} q_0 \xrightarrow{\texttt{1}} q_0 \xrightarrow{\texttt{0}} q_0 \xrightarrow{\texttt{1}} q_1$ de probabilité 1/8 ;
- $q_0 \xrightarrow{\texttt{1}} q_0 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{0}} q_1 \xrightarrow{\texttt{1}} q_1$ de probabilité 1/8 ;
- $q_0 \xrightarrow{\texttt{1}} q_0 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{0}} q_0 \xrightarrow{\texttt{1}} q_1$ de probabilité 1/16 ;
- $q_0 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{0}} q_1 \xrightarrow{\texttt{1}} q_1$ de probabilité 1/4 ;
- $q_0 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{1}} q_1 \xrightarrow{\texttt{0}} q_0 \xrightarrow{\texttt{1}} q_1$ de probabilité 1/8

donc $\Pr(\texttt{1101}) = \frac{1}{8} + \frac{1}{8} + \frac{1}{16} + \frac{1}{4} + \frac{1}{8} = \frac{11}{16} = \underline{0,\texttt{1011}}_2$.

Question 22. Les deux questions précédentes laissent penser que $\Pr(u) = \underline{0,\tilde{u}}_2$ où $\tilde{u}$ est l'image miroir du mot u. Montrons-le en raisonnant par récurrence sur $|u|$.

– Si $|u| = 0$ alors $u = \varepsilon$ et $\Pr(u) = 0 = \underline{0,}_2$ puisque q_0 n'est pas un état final.

– Si $|u| > 0$, posons $u = va$ où a est la dernière lettre de u et supposons le résultat acquis pour le mot v. Considérons alors un chemin ρ_u acceptant pour u.

- Si $a = \texttt{0}$, la dernière transition est nécessairement $q_1 \xrightarrow{\texttt{0}} q_1$ et $\rho_u = \rho_v \circ (q_1 \xrightarrow{\texttt{0}} q_1)$ où ρ_v est une transition acceptante pour v.

  Dans ce cas on a donc $\Pr(u) = \frac{1}{2} \times \Pr(v) = \underline{0,\texttt{0}\tilde{v}}_2$.

- Si $a = \texttt{1}$, la dernière transition est ou bien $q_1 \xrightarrow{\texttt{1}} q_1$, auquel cas $\rho_u = \rho_v \circ (q_1 \xrightarrow{\texttt{1}} q_1)$ où ρ_v est une transition acceptante pour v, ou bien $q_0 \xrightarrow{\texttt{1}} q_1$, auquel cas $\rho_u = \rho_v \circ (q_0 \xrightarrow{\texttt{1}} q_1)$ où ρ_v est une transition non acceptante pour v.

  D'après la question 13, on a : $\Pr(u) = 1 \times \Pr(v) + \frac{1}{2} \times (1 - \Pr(v)) = \frac{1}{2} \times (1 + \Pr(v)) = \underline{0,\texttt{1}\tilde{v}}_2$.

Dans les deux cas on obtient bien $\Pr(u) = \underline{0,\tilde{u}}_2$.

Question 23. L'égalité demandée résulte immédiatement des définitions et du résultat de la question précédente :

$$\mathcal{L}_\eta(\mathcal{A}_1) = \left\{u \in \Sigma^* \;\middle|\; \Pr(u) > \eta\right\} = \left\{u \in \Sigma^* \;\middle|\; \underline{0,\tilde{u}}_2 > \eta\right\} = \left\{a_1 \cdots a_n \in \Sigma^* \;\middle|\; \underline{0,a_n \cdots a_1}_2 > \eta\right\}.$$

Question 24. Si $0 \leqslant \eta < \eta' < 1$ on a bien évidemment $\mathcal{L}_{\eta'}(\mathcal{A}_1) \subset \mathcal{L}_\eta(\mathcal{A}_1)$. De plus, cette inclusion est stricte. En effet, puisque les nombres dyadiques sont denses dans $\mathbb{R}$, il existe un tel nombre d dans l'intervalle $[\eta, \eta'[$. Posons $d = \underline{0,d_1 \cdots d_n}_2$ et considérons le mot $u = d_n \cdots d_1$. Alors $u \in \mathcal{L}_{\eta'}(\mathcal{A}_1) \setminus \mathcal{L}_\eta(\mathcal{A}_1)$.
De ceci il résulte qu'il existe un nombre non dénombrable de langages stochastiques sur l'alphabet $\Sigma = \{\texttt{0},\texttt{1}\}$. Or pour un alphabet donné l'ensemble des langages rationnels est dénombrable : pour s'en convaincre, il suffit de constater que l'ensemble $\mathscr{A}_p$ des automates finis déterministes à p états est fini donc l'ensemble $\mathscr{A} = \bigcup_{p \in \mathbb{N}^*} \mathscr{A}_p$ des automates finis déterministes est dénombrable.
Il existe donc parmi les langages stochastiques $\mathcal{L}_\eta(\mathcal{A}_1)$ des langages qui ne sont pas rationnels.