Corrigé : arbres couvrants de poids minimum

Partie I. Implémentation des graphes

Question 1. Le graphe n'étant pas orienté, on évite de faire apparaître deux fois chaque arête :

```
let liste_aretes g =
  let n = vect_length g in
  let aretes = ref [] in
  for i = 0 to n-1 do
    for j = i+1 to n-1 do
      if g.(i).(j) > 0 then aretes := (i, j, g.(i).(j))::!aretes ;
    done
  done ;
  !aretes ;;
```

Question 2. On commence par rédiger le tri fusion (c'est une question de cours) :

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

let poids (_, _, x) = x ;;

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

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

Il reste alors à définir :

```
let liste_aretes_triees g = tri_fusion (liste_aretes g) ;;
```

Partie II. Préliminaires sur les arbres

Question 3. Notons C_u et C_v les composantes connexes de G.
Si $C_u = C_v$, l'arête uv relie deux sommets d'une même composante connexe donc G et $G+uv$ ont les mêmes composantes connexes. De plus, il existe un chemin $u \rightsquigarrow v$ de G qui relie u et v, donc $u \rightsquigarrow v \to u$ est un cycle de $G+uv$.
Si $C_u \neq C_v$, considérons deux sommets quelconques $a \in C_u$ et $b \in C_v$. Il existe deux chemins $a \rightsquigarrow u$ et $b \rightsquigarrow v$ de G. Alors $a \rightsquigarrow u \to v \rightsquigarrow b$ est un chemin de $G+uv$ ce qui montre que les sommets de $C_u \cup C_v$ appartiennent à la même composante connexe de $G+uv$. Ce dernier possède donc une composante connexe de moins que G. Enfin, s'il existait un cycle de $G+uv$ passant par l'arête uv il existerait un chemin de G reliant u et v, ce qui ne se peut puisque ces deux sommets appartiennent à deux composantes connexes distinctes de G.

Question 4.

- Montrons que (i) $\Longrightarrow$ (ii).
  Soit m le nombre d'arêtes de G. Pour tout $i \in [\![1, m]\!]$, $G_i = G_{i-1} + e_i$. Puisque G et donc G_i ne contient aucun cycle, la question 3 montre que G_i possède une composante connexe de moins que G_{i-1}. On en déduit que $G = G_m$ possède $n - m$ composantes connexes. Or G est connexe donc $n - m = 1$, soit $m = n - 1$.

– Montrons que (ii) $\Longrightarrow$ (iii).
Pour tout $i \in [\![1, n-1]\!]$, G donc G_i ne contient aucun cycle donc G_i possède une composante connexe de moins que G_{i-1}. On en déduit que $G = G_m$ possède $n - m$ composantes connexes. Or $m = n - 1$ donc G est bien connexe.
– Montrons que (iii) $\Longrightarrow$ (iv).
Quitte à numéroter les arêtes, supposons que $e_m = e$. Alors $G - e = G_{m-1}$ et $G = G_m$.
Pour tout $i \in [\![1, m]\!]$, G_i a au moins $n - i$ composantes connexes donc G_{m-1} a au moins $n - m + 1$ composantes connexes. Or $m = n - 1$ donc $G - e$ possède au moins deux composantes connexes et n'est donc pas connexe.
– Montrons que (iv) $\Longrightarrow$ (v).
Puisque G est connexe, il existe un chemin de G d'extrémités x et y. S'il en existait un second nous pourrions considérer une arête e appartenant à l'un des deux chemins mais pas à l'autre. Dans ces conditions $G - e$ resterait connexe et G ne serait pas minimalement connexe.
– Montrons que (v) $\Longrightarrow$ (vi).
Si G contenait un cycle passant par deux sommets distincts x et y, il existerait deux chemins distincts d'extrémités x et y, ce qui ne se peut.
Considérons deux sommets non adjacents u et v de G. il existe par hypothèse un chemin $u \rightsquigarrow v$ de G reliant u à v. Dans ce cas $u \rightsquigarrow v \to u$ est un cycle de $G + uv$. G est donc maximalement sans cycle.
– Montrons que (vi) $\Longrightarrow$ (1).
Considérons deux sommets u et v de G. Puisque G est acyclique et que $G + uv$ contient un cycle, ce cycle passe nécessairement par l'arête uv. D'après la question 3 u et v sont dans la même composante connexe de G ce qui prouve que G est connexe et est donc un arbre.

Question 5. Considérons l'ensemble des sous-graphes connexes $G' = (V, E')$ de G ayant les mêmes sommets que G. Cet ensemble est non vide car il contient G donc il contient un élément minimal au sens de l'inclusion. Par définition ce graphe est minimalement connexe donc d'après la question précédente, il s'agit d'un arbre.

Question 6. Le graphe $G_k = (V, E_k)$ est par construction acyclique. Supposons qu'il ne soit pas connexe. Il possède alors au moins deux composantes connexes. Passons en revue les arêtes de G qui ne sont pas présentes dans G_k. Puisque G est connexe, l'une au moins doit relier deux des composantes connexes de G_k ; notons la e_i. Mais d'après la question 3 $G_k + e_i$ ne contient aucun cycle donc e_i aurait du être ajouté à l'étape i de l'algorithme, ce qui est contradictoire.
On en déduit que G_k est acyclique et connexe, bref qu'il s'agit d'un arbre.

Question 7. Si le graphe G n'est pas connexe cet algorithme va retourner une forêt couvrante de G, autrement dit un graphe dont chacune des composantes connexes est un arbre couvrant l'une des composantes connexes de G.

Question 8. S'il existe une arête $e \in E_2$ dont les extrémités relient deux composantes distinctes de (V, E_1) alors $e \in E_2 \setminus E_1$ et d'après la question 3, $(V, E_1) + e$ est sans cycle.
Supposons maintenant qu'une telle arête n'existe pas : chacune des arêtes de E_2 relie deux sommets d'une même composante connexe de (V, E_1). Puisque (V, E_1) est un graphe acyclique, chacune de ces composantes connexes C_i est un arbre, et le nombre d'arêtes de E_1 présente dans C_i est égal à $|C_i| - 1$. Puisque $|E_2| > |E_1|$ il existe au moins une de ces composantes connexes C_j dont les sommets sont reliés par au moins $|C_j|$ arêtes de E_2. Mais d'après la question 4 ceci implique que (V, E_j) contient un cycle, ce qui est absurde.

Partie III. Algorithme de Kruskal

Question 9. Plusieurs solutions sont possibles suivant la manière d'ordonner les arêtes de même poids. On obtient l'un des quatre arbres représentés figure 1.

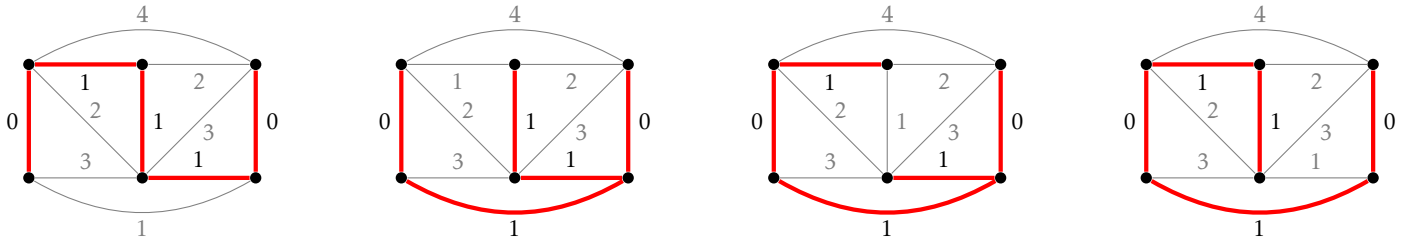

Figure 1 – Les quatres arbres couvrants de poids minimum qu'on peut obtenir.

Question 10. D'après la question 6 on sait déjà que l'algorithme de Kruskal retourne un arbre croissant, qui possède donc $n-1$ arêtes. Notons $e_{i_1}, e_{i_2}, \dots, e_{i_{n-1}}$ ces arêtes, avec $i_1 < i_2 < \cdots < i_{n-1}$.
Considérons un arbre couvrant T de poids minimal, et notons $e_{j_1}, e_{j_2}, \dots, e_{j_{n-1}}$ ses arêtes. Comme le suggère l'énoncé nous allons prouver par récurrence sur $k \in [\![0, n-1]\!]$ que : $w(e_{i_1}) + \cdots + w(e_{i_k}) \leqslant w(e_{j_1}) + \cdots + w(e_{j_k})$ ce qui permettra de conclure.

- Si $k = 0$ le résultat est évident.
- Si $l \in [\![1, n-1]\!]$, supposons le résultat acquis au rang $k-1$.
  Posons $\mathrm{E}_1 = \{e_{i_1}, \dots, e_{i_{k-1}}\}$ et $\mathrm{E}_2 = \{e_{j_1}, \dots, e_{j_k}\}$. D'après la question 8, il existe $j \in \{j_1, \dots, j_k\}$ tel que $j \notin \{i_1, \dots, i_{k-1}\}$ et $(\mathrm{V}, \mathrm{E}_1 + e_j)$ est acyclique. Ceci impose à j d'être supérieur ou égal à i_k (car sinon il aurait été sélectionné par l'algorithme de Kruskal). On a donc $w(e_j) \geqslant w(e_{i_k})$.
  Par ailleurs, $w(e_j) \leqslant w(e_{j_k})$ (les arêtes sont ordonnées par poids croissant) et par hypothèse de récurrence on a $w(e_{i_1}) + \cdots + w(e_{i_{k-1}}) \leqslant w(e_{j_1}) + \cdots + w(e_{j_{k-1}})$ donc : $w(e_{i_1}) + \cdots + w(e_{i_k}) \leqslant w(e_{j_1}) + \cdots + w(e_{j_k}) + w(e_j)$.

Question 11. En appliquant l'algorithme de Kruskal au graphe du préambule on obtient l'arbre couvrant présenté figure 2, de poids total égal à 1664.

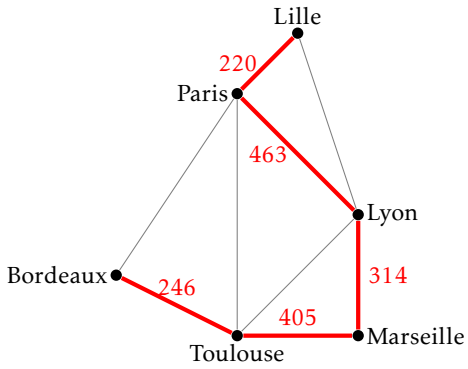

Figure 2 – L'arbre couvrant de poids minimal.

Question 12. Considérons un graphe dont les sommets sont chacune des cinq langues du tableau, chaque couple de sommet étant relié par une arête pondérée par le coût de la traduction. On cherche un ensemble d'arêtes qui assure la connexité du graphe et dont le poids total soit minimal, autrement dit un arbre couvrant minimal.
L'algorithme de Kruskal sélectionne successivement les arêtes Français-Italien, Français-Espagnol, Anglais-Allemand, Espagnol-Anglais pour un coût total de 10 000 euros (ce n'est pas la seule solution). Le document est traduit en espagnol et en allemand, le document espagnol est ensuite traduit en français, et le document français traduit en italien.

Question 13.

```
let creer n = init_vect n (function i -> i) ;;
```

Question 14.

```
let composante c i = c.(i) ;;
```

Question 15.

```
let fusionner c i j =
  let n = vect_length c in
  for k = 0 to n-1 do
    if c.(k) = j then c.(k) <- i
  done ;;
```

Question 16. La fonction **`creer`** est de complexité linéaire (tant spatiale que temporelle), la fonction **`composante`** est de coût constant et la fonction **`fusionner`** de coût linéaire.

Question 17.

```
let kruskal g =
  let n = vect_length g in
  let c = creer n in
  let rec aux acc = function
    | []           -> acc
    | (u, v, p)::q when composante c u = composante c v  -> aux acc q
    | (u, v, p)::q -> fusionner c (composante c u) (composante c v) ;
                      aux ((u, v, p)::acc) q
  in aux [] (liste_aretes_triees g) ;;
```

La fonction effectue une fois la fonction **creer**, au plus $4m$ fois la fonction **composante**, et $n-1$ fois la fonction **fusionner**. Sachant que le tri initial est de coût $O(m \log n)$, le coût total est un $O(m \log n + n + m + n^2) = O(m \log n + n^2)$.

Partie IV. Implémentation efficace

Question 18.

```
let rec composante parent i = match parent.(i) with
  | j when i = j -> i
  | j            -> composante parent j ;;
```

Question 19. On ne doit mettre à jour le tableau des hauteurs que dans le cas où i et j sont les racines de deux arbres de même hauteur. Si on choisit de faire de j le fils de i, la hauteur de ce dernier doit être incrémenté d'une unité.

```
let rec fusionner parent hauteur = fun
  | i j when hauteur.(i) < hauteur.(j) -> parent.(i) <- j
  | i j when hauteur.(i) > hauteur.(j) -> parent.(j) <- i
  | i j -> parent.(j) <- i ; hauteur.(i) <- hauteur.(i) + 1 ;;
```

Question 20. Raisonnons par récurrence sur $h(i)$, la hauteur de l'arbre enraciné en i.

- Si $h(i) = 0$, cet arbre contient uniquement le sommet i, soit 2^0 sommets.
- Si $h(i) > 0$, supposons le résultat acquis aux rangs inférieurs, et intéressons-nous à la dernière incrémentation de la valeur de $h(i)$: celle-ci a eu lieu lors de la fusion de deux arbres de hauteurs $h(i)-1$. Par hypothèse de récurrence chacun de ces deux arbres contient au moins $2^{h(i)-1}$ sommets donc l'arbre qui en a résulté contient au moins $2^{h(i)}$ sommets.

Puisque chaque arbre enraciné contient au maximum n sommets, on en déduit que $h(i) \leqslant \log n$. La complexité de l'algorithme de Kruskal avec cette implémentation est donc un $O(m \log n + n + m \log n) = O(m \log n)$.

Question 21. Il suffit de procéder par énumération : un premier parcours du tableau compte le nombre d'occurrence de chacun des entiers de $[\![0, k-1]\!]$ et range ces résultats dans un tableau de longueur k (pour un coût temporel en $O(n)$ et un coût spatial en $\Theta(k)$) ; un parcours de ce second tableau permet de modifier le tableau initial (pour un coût temporel en $O(n+k)$).

```
let tri_lineaire k t =
  let n = vect_length t in
  let occ = make_vect k 0 in
  for i = 0 to n-1 do
    occ.(t.(i)) <- occ.(t.(i)) + 1
  done ;
  let i = ref 0 in
  for j = 0 to k-1 do
    for p = 1 to occ.(j) do
      t.(!i) <- j ;
      incr i
    done
  done ;;
```

Question 22. Lors du parcours du chemin qui relie le sommet i à la racine de l'arbre, la compression de chemin consiste à faire de chacun des sommets rencontrés le fils de la racine de l'arbre, ce qui a en général pour résultat d'en diminuer la hauteur.

Partie V. Coupe minimum

Question 23. Notons $C_1, \dots, C_i$ les composantes connexes de G. Une arête reliant deux composantes distinctes C_j et $C_{j'}$ appartient à la fois à $\delta(C_j)$ et à $\delta(C_{j'})$, donc le nombre d'arêtes ayant leurs extrémités dans deux composantes connexes distinctes est égal à : $\frac{1}{2}\sum_{j=1}^{i} |\delta(C_j)|$.

Or par définition de k on a : $\forall j \in [\![1, i]\!]$, $|\delta(C_j)| \geqslant k$, donc le nombre total de telles arêtes est supérieur ou égal à $\frac{ik}{2}$.

Question 24. L'arête que l'on rajoute à l'étape i doit nécessairement relier deux composantes distinctes de T ; d'après ce qui précède il y en a au moins $\frac{ik}{2}$. Parmi celles-ci, au plus k appartiennent à Y, donc la probabilité de sélectionner une arête de Y est au plus égale à $\frac{k}{ik/2} = \frac{2}{i}$.

Question 25. Lorsque T a exactement deux composantes connexes X et V – X on a $\delta(X) = \delta(Y)$ si et seulement si aucune des arêtes de T n'est dans $\delta(Y)$. Lors des $n-2$ étapes de l'algorithme de Kruskal il faut qu'à aucun moment une arête de Y n'ait été sélectionnée. D'après la question précédente la probabilité de cet évènement est au moins égale à :

$$\prod_{i=3}^{n}\left(1-\frac{2}{i}\right) = \prod_{i=3}^{n}\left(\frac{i-2}{i}\right) = \frac{2(n-2)!}{n!} = \frac{2}{n(n-1)}.$$

Question 26. D'après la question précédente, la probabilité qu'en réalisant l'algorithme t fois aucune des coupes obtenues ne soit minimale est au plus égale à $\left(1-\frac{2}{n(n-1)}\right)^t \leqslant 1-\frac{2t}{n(n-1)} \leqslant \exp\left(\frac{-2t}{n(n-1)}\right)$.
Lorsque $t = 5n(n-1)$ cette probabilité est donc inférieure à e^{-10}. D'après la partie précédente le coût de l'algorithme de Kruskal peut être réduit à un $O(m \log n)$ donc le coût total de l'exécution de cet algorithme probabiliste est un $O(n^2(m + m\log n)) = O(n^2 m \log n)$.