**Dans tout le sujet, on impose la contrainte suivante : lorsque vous écrirez du code CAML, quel qu'il soit, vous ne devez utiliser aucune boucle while, aucune référence ; en revanche vous pouvez utiliser des tableaux, des boucles for et des fonctions récursives.** . En revanche, quand vous décrirez en pseudo-code des algorithmes, vous adopterez le style qui vous semble le plus approprié.

# Problème 1 - Cycles eulériens

Soit $G = (S, A)$ un graphe **non orienté**. On note $n$ le nombre de sommets du graphe et $m$ le nombre d'arêtes. Les sommets sont dans ce problème numérotés de 0 à $n-1$. On supposera toujours, même si ce n'est pas rappelé dans l'énoncé, que $n \geqslant 2$.

Deux sommets sont voisins quand ils sont les extrémités d'une arête. Le degré d'un sommet $s$, noté $\deg(s)$, est le nombre de ses voisins. Dans ce problème, on suppose que les graphes ne possèdent pas de point isolé, *i.e.* tous les sommets sont de degrés non nuls.

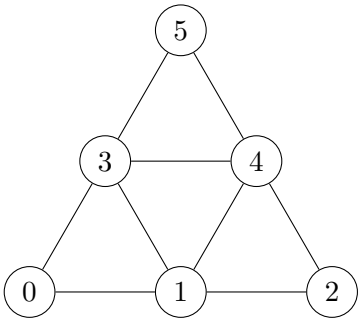
Un chemin dans le graphe $G$ est une suite $(s_0, \ldots, s_p)$ de sommets tels que pour tout $i \in [\![0, p-1]\!]$, $\{s_i, s_{i+1}\}$ est une arête (avec $p = 0$, un chemin peut être réduit à un sommet). Dans ce cas, $p$ est la longueur du chemin, c'est-à-dire le nombre d'arêtes du chemin ; $s_0$ est l'origine du chemin et $s_p$ le but. Toute arête $\{s_i, s_{i+1}\}$ est dite empruntée par le chemin.

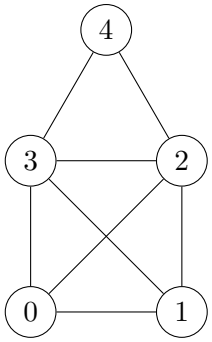
Un cycle est un chemin dont l'origine et le but sont égaux.

On appelle chemin eulérien dans le graphe $G$ un chemin qui emprunte toutes les arêtes du graphe une et une seule fois. On définit de même la notion de cycle eulérien.

## Partie 1

**Question 1)** Sur les graphes suivants, déterminez un chemin eulérien :




On note $S_p$ l'ensemble des sommets de degrés pairs et $S_i$ celui des sommets de degrés impairs.

**Question 2)**

a) Que vaut $\displaystyle\sum_{s \in S} \deg(s)$ ? Justifiez.

b) Montrez que $\operatorname{card} S_i$ est un entier pair.

**Question 3)**

a) Montrez que si $G$ possède un chemin eulérien, alors $G$ est connexe.

b) Montrez que si $G$ possède un cycle eulérien, alors $\operatorname{card} S_i = 0$.

c) Que dire de $\operatorname{card} S_i$ quand $G$ possède un chemin eulérien qui n'est pas un cycle.

## Partie 2 - Cycle eulérien

Dans cette partie du problème, on suppose que $G$ est un graphe connexe et que $\operatorname{card} S_i = 0$.

**Question 1)** Justifiez brièvement que $n \geqslant 3$ et que tous les sommets sont de degré au moins 2.

**Question 2)** On choisit un sommet $s_0$ et on construit par récurrence un chemin $(s_0, \ldots, s_k)$ de la façon suivante : si $s_k$ est le but du chemin, alors tant qu'on le peut, on choisit $s_{k+1}$ parmi ses voisins puis on supprime dans $G$ l'arête $\{s_k, s_{k+1}\}$.

a) Justifiez que ce processus de construction d'un tel chemin termine et que le chemin ainsi construit n'emprunte pas deux fois la même arête.

b) Comment évoluent les degrés des sommets du graphe lors de la construction du chemin ? En particulier, montrez qu'à chaque étape, il ne peut y avoir que deux points de degré impair au plus.

c) Montrez que le chemin ainsi construit est un cycle.

**Question 3)**

a) On applique l'algorithme précédent. Justifiez que si tous les points du cycle sont de degré 0 à la fin du processus, alors ce cycle contient tous les points du graphe, puis qu'il passe par toutes les arêtes du graphe initial : il est donc eulérien.

b) Dans le cas contraire, on choisit un point de degré non nul dans le cycle et on recommence le même processus : on obtient un deuxième cycle. À partir de ces deux cycles, construisez un cycle qui passe par toutes les arêtes des deux cycles : on dit qu'on a fusionné les deux cycles. Quelle complexité peut-on attendre d'une telle fusion ?

**Question 4)** Montrez que le graphe $G$ possède un cycle eulérien et donnez le principe en pseudo-code d'un algorithme qui construit un tel cycle eulérien.

On représente les graphes par des tableaux de listes d'adjacences : **type** graphe == int list vect

Et un cycle (ou un chemin) est représenté par la liste de ses sommets.

**Question 5)** Écrivez une fonction supprimer_arete a b g de type `int -> int -> graphe -> unit`, de paramètres $a$ et $b$ deux sommets d'une arête dans le graphe $g$, qui supprime l'arête $\{a, b\}$ dans le graphe $g$ (le graphe est donc modifié). Pour éviter de réinventer la roue à chaque fois, vous pourrez vous servir de la fonction prédéfinie en CAML except : 'a −> 'a list −> 'a list, qui supprime une occurence d'un élément dans une liste.

**Question 6)** Écrivez une fonction cycle s g de type `int -> graphe -> int list`, qui construit un cycle à partir du sommet $s$ dans le graphe $g$ selon le procédé décrit ci-dessus.

**Question 7)** Écrivez une fonction rechercher c g de type `graphe -> int list -> int list` de paramètres un graphe $g$ et une liste de sommets $c$ et qui calcule la liste vide si tous les sommets de la liste n'ont aucun voisin dans le graphe ou une liste à un sommet $[s]$ tel que $s$ appartienne à la liste $c$ et possède au moins un voisin dans le graphe.

**Question 8)** Écrivez une fonction fusionner c1 c2 s de type `int list -> int list -> int -> int list` qui prend en paramètres deux cycles $c1$ et $c2$ et un sommet commun aux deux cycles et qui fusionne ces deux cycles en un seul.

**Question 9)** Écrivez une fonction cycle_eulerien g de type `graphe -> int list`, qui donne la liste des sommets d'un cycle eulérien dans l'ordre du parcours du cycle.

### **Partie 3** - Chemin eulérien

Dans cette partie du problème, on suppose que $G$ est un graphe connexe et que $\operatorname{card} S_i = 2$. On pose alors $S_i = \{a, b\}$.

**Question 1)** Montrez que si $G$ possède un chemin eulérien, son origine et son but sont les sommets $a$ et $b$.

**Question 2)** Montrez que $G$ possède effectivement un chemin eulérien en donnant le principe d'un algorithme qui calcule un tel chemin (indication : on peut modifier légèrement le graphe $g$ au début de l'algorithme).

## Problème 2 - Arborescence couvrante de poids minimum : algorithme de Prim

Soit $G = (S, A)$ un graphe **non orienté** valué et connexe. On note $n$ le nombre de sommets du graphe et $m$ le nombre d'arêtes. Les sommets sont dans ce problème numérotés de 0 à $n - 1$.

Un sommet est donc un entier : **type** sommet == int

On représente le graphe par un tableau de listes d'adjacences : **type** graphe == (int $*$ sommet) list vect où le premier entier du couple est la valeur de l'arête, le second est l'extrémité de l'arête.

On suppose que les valuations des arêtes sont toutes strictement positives. Pour alléger les notations, une arête d'extrémités $a$ et $b$ est notée $a{-}b$ et la valuation d'une arête est notée $v(a{-}b)$.

Le poids du graphe est la somme des valuations de ces arêtes.

Un sous-graphe de $G$ est un couple $G' = (S', A')$ tel que $S' \subset S$ et $A' \subset A$. Un sous-graphe couvrant est un sous-graphe $(S', A')$ tel que $S' = S$. Si $G' = (S', A')$ est un sous-graphe, on appelle bord de $G'$, notée $B(G')$, l'ensemble des arêtes dont l'une des extrémités est dans $S'$ et l'autre n'y est pas

$$B(G') = \{a{-}b \ / \ a \in S' \text{ et } b \notin S'\}$$

Un chemin dans le graphe $G$ est une suite $(s_0, \ldots, s_p)$ de sommets tels que pour tout $i \in [\![0, p-1]\!]$, $s_i{-}s_{i+1}$ est une arête (avec $p = 0$, un chemin peut être réduit à un sommet). Dans ce problème, un cycle est un chemin $(s_0, \ldots, s_p)$ tel que $s_0 = s_p$ et dont les arêtes sont toutes distinctes (on impose ceci pour éviter d'appeler cycle un chemin empruntant la même arête en sens contraires).

Un graphe est dit acyclique quand il ne possède pas de cycles.

**Question 1)** Montrez que parmi tous les sous-graphes couvrants connexes de $G$, il en existe un de poids minimal et qu'un tel sous-graphe est nécessairement acyclique.

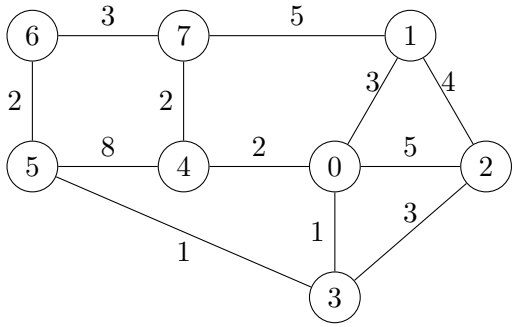
**Question 2)**

a) Soit $G' = (S', A')$ un sous-graphe de $G$. Montrez que si $G'$ est connexe et $B(G')$ n'est pas vide, alors en choisissant une arête $a{-}b$ dans $B(G')$ ($a \in S'$ et $b \notin S'$), le graphe $\mathrm{e}(G') = (S' \cup \{b\}, A' \cup \{a{-}b\})$ est un graphe connexe.

b) On choisit un sommet de $G$, noté $s_0$ et on pose $G_0 = (\{s_0\}, \varnothing)$, puis on définit une suite de sous-graphes de $G$ par $G_{i+1} = \mathrm{e}(G_i)$. Montrez que cette suite est finie et que son dernier terme est un sous-graphe couvrant de $G$.

**Question 3)** Dans la suite de sous-graphes construite précédemment, le choix de l'arête $a{-}b$ est « au hasard ». Montrez que si elle est choisie à chaque fois judicieusement, le sous-graphe couvrant final est de poids minimal.

**Question 4)** Sur le graphe ci-dessous, on choisit $s_0 = 0$. Donnez les sous-graphes successivement calculés ainsi que leurs bords.




**Question 5)** Soit $G' = (S', A')$ un sous-graphe de $G$ tel que $B(G')$ n'est pas vide : pour construire $\mathrm{e}(G')$, on a choisi une arête $a{-}b$ dans $B(G')$ ($a \in S'$ et $b \notin S'$) et on a posé $G'' = \mathrm{e}(G') = (S' \cup \{a\}, A' \cup \{a{-}b\})$.

a) Donnez une idée simple pour calculer le bord d'un sous-graphe (on ne demande pas de code, juste un mode opératoire).

b) Est-ce une bonne idée ici ? Expliquez comment calculez à moindre coût $B(G'')$ à partir de $B(G')$.

**Question 6)** Décrivez en pseudo-code l'algorithme précédent qui construit un sous-graphe couvrant de poids minimal : pour cette description, vous pouvez utiliser une boucle « tant que ».

**Question 7)** Dans l'algorithme précédent, quelle est l'opération qui est de complexité prépondérante ? Quel type abstrait peut-on utiliser ici pour représenter le bord d'un graphe afin de minimiser le coût ? Justifiez votre choix en estimant la complexité de l'algorithme et en la comparant à une version naïve.

Dans toute la suite, une fois ce choix de type abstrait effectué, vous introduirez toutes les fonctions classiques associées à ce type dont vous avez besoin, sans écrire leurs codes, et que vous appellerez par leur nom habituel (new, add, etc) en précisant à chaque fois leur utilité, leur signature CAML (comme new : unit −> abstrait, etc) et leur complexité.

**Question 8)** Les arêtes qui appartiennent au bord du sous-graphe sont représentées par des triplets $(a, v, b)$ où $a$ est l'extrémité dans le sous-graphe, $v$ est la valeur de l'arête et $b$ est l'extrémité qui n'est pas dans le sous-graphe.

**type** arete == (sommet ∗ int ∗ sommet)

a) On souhaite représenter les sous-graphes par des couples de type `bool vect * arete list` : pourquoi adopter cette représentation ?

b) Écrivez le code CAML nécessaire pour traduire l'algorithme précédent.

c) Estimez la complexité de votre fonction.

**Devoir surveillé 2 - Corrigé**

# Problème 1

## Partie 1

**Question 1)** Il y a bien sûr plusieurs réponses possibles.

Le premier graphe admet pour cycle eulérien le cycle $0, 1, 2, 4, 1, 3, 4, 5, 3, 0$,

le second le cycle eulérien $0, 1, 2, 0, 3, 2, 4, 3, 1$.

**Question 2)**

a) C'est du cours : $\sum_{s \in S} \deg(s) = 2m$, car le degré d'un sommet est le nombre d'arêtes qui en sont issues, donc en sommant les degrés, on compte deux fois le nombre d'arêtes, puisque chaque arête relie exactement deux sommets.

b) On coupe la somme en deux : $S$ est la réunion disjointe de $S_p$ et $S_i$ donc $\sum_{s \in S} \deg(s) = \sum_{s \in S_p} \deg(s) + \sum_{s \in S_i} \deg(s)$.
Dans la première somme, on additionne des entiers pairs donc on obtient un entier pair. Or la somme des deux vaut $2m$, un entier pair, donc $\sum_{s \in S_i} \deg(s)$ est un entier pair. Comme on additionne dans cette somme des entiers impairs, cette somme contient donc un nombre pair de termes, c'est-à-dire $\operatorname{card} S_i$ est pair.

**Question 3)**

a) Si $G$ possède un chemin eulérien, alors il existe un chemin qui passe par toutes les arêtes, donc *a fortiori* par tous les sommets, donc tous les sommets sont reliés par un sous-chemin de celui-ci : $G$ est connexe.

b) Si $G$ possède un cycle eulérien, alors en chaque sommet du graphe, chaque fois que ce cycle atteint un sommet, il en repart par une autre arête donc il emprunte un nombre pair d'arêtes issues d'un même sommet. Or comme il emprunte toutes les arêtes, chaque sommet possède donc un nombre pair d'arêtes incidentes. Il n'y a aucun sommet de degré impair.

c) Le même raisonnement s'applique à tout point du chemin eulérien qui n'est pas l'une de ses deux extrémités : ces points sont tous de degrés pairs. Les deux extrémités sont de degrés impairs, car le chemin n'est pas un cycle : le chemin débute en $a$ et finit en $b \neq a$, donc si le chemin repasse éventuellement par $a$, il passe par deux arêtes à chaque passage, donc le nombre d'arêtes issues de $a$ est un (la première arête du chemin) plus un nombre pair, et de même pour $b$.

## Partie 2

**Question 1)** Un sommet de degré 0 est un sommet isolé des autres : le graphe n'est alors pas connexe. Donc tous les sommets sont ici de degrés non nuls, donc comme ils sont tous de degrés pairs, leurs degrés sont au moins 2.

Si $n = 2$, alors les deux sommets sont reliés par une seule arête, donc les degrés sont égaux à 1 : contradiction. Donc comme $n \geqslant 2$, alors $n \geqslant 3$.

**Question 2)**

a) Le processus décrit termine, car le nombre d'arêtes diminue strictement dans $\mathbb{N}$, qui est un ensemble bien fondé (propriété classique d'arrêt d'un algorithme).
De plus, comme on supprime chaque arête dès qu'on l'a empruntée, le chemin ainsi construit ne peut évidemment pas passer deux fois par la même arête.

b) On remarque que chaque suppression d'une arête fait baisser de 1 les degrés de deux sommets, donc dans la construction du chemin, les degrés des points autres que les deux extrémités ont été diminués d'un nombre pair, donc leurs degrés restent pairs. Si les deux extrémités sont différentes, alors leurs degrés sont alors devenus impairs ; s'ils sont égaux (le chemin est un cycle), leurs degrés sont restés pairs. Quant aux autres sommets, leurs degrés n'ont pas variés donc ils restent pairs. En conclusion, à la fin de chaque étape, soit tous les sommets sont de degrés pairs, soit deux points sont de degrés impairs et ce sont les deux extrémité du chemin.

c) Il reste à prouver que le chemin ainsi construit finit au point de départ $s_0$ choisi.

Quand l'algorithme termine, alors cela signifie que le dernier point choisi était de degré 1 juste avant d'être choisi (donc de degré impair). Si ce point n'est pas l'origine du chemin, alors comme ce point est de degré impair, il a été obtenu lors de la construction du chemin puisque tous les degrés étaient pairs initialement, donc il est au « milieu » du chemin, ce qui est impossible d'après la remarque précédente. On a donc obtenu un cycle.

**Question 3)**

a) Si tous les points du cycle sont de degré 0 à la fin du processus, alors plus aucune arête ne permet d'accéder à ces points à partir d'un point extérieur au cycle, ils forment donc une composante connexe de $G$. Or $G$ est connexe donc $G$ n'a qu'une seule composante connexe, autrement dit le cycle contient tous les points.

Et comme on a supprimé toutes les arêtes entre ces points, on a donc supprimé toutes les arêtes du graphe, donc le cycle contient toutes les arêtes : il est eulérien.

b) On part d'un point commun aux deux cycles : quitte à effectuer une permutation circulaire, on peut supposer que les deux cycles débutent par le sommet commun $c$ :

le premier cycle est $[c, a_1, \ldots, a_n, c]$ et le second est $[c, b_1, \ldots, b_p, c]$

Alors la suite de sommets $[c, a_1, \ldots, a_n, c, b_1, \ldots, b_p, c]$ est un cycle.

Pour construire la fusion de deux cycles, on a donc besoin de découper chaque liste en deux listes (les sommets avant $c$ et ceux après $c$), à les échanger pour faire débuter les cycles en $c$ et à concaténer des listes. Tout ça peut se faire en complexité linéaire par rapport à la somme des longueurs des deux listes.

**Question 4)** On considère l'algorithme suivant :

cycle(G) :
on choisit s un sommet de G et on pose c = [s]
tant que c contient un sommet de degré $> 0$ :
  on pose s ce sommet
  on construit le cycle associé comme ci-dessus
  on remplace c par la fusion de ce cycle avec c
fin du tant que
on retourne c

D'après les questions précédentes, cet algorithme construit un cycle eulérien. Et on est sûr que l'algorithme termine car le nombre d'arêtes diminue strictement dans $\mathbb{N}$ à chaque itération.

**Question 5)**

```
let supprimer_arete a b g =
  g.(b) <- except a g.(b); g.(a) <- except b g.(a);;
```

**Question 6)**

```
let cycle s g =
  let rec cyclaux l =
   match l with
   | s :: _ -> if g.(s) = [] then l
     else (let s' = hd g.(s) in
             supprimer_arete s s' g;
             cyclaux (s' :: l))
   | _ -> []
  in
  cyclaux [s];;
```

**Question 7)**

```
let rec rechercher cy g =
  match cy with
  | [] -> []
  | c :: qy -> if g.(c) <> [] then [c] else rechercher qy g;;
```

**Question 8)**

```
let rec decouper l c =
 match l with
 | [] -> [],[]
 | [a] -> [a],[]
 | a :: q -> if a = c then [c],q
   else let av, ar = decouper q c in
    a :: av, ar;;

let permuter l c =
  let av, ar = decouper l c in
  c :: ar @ tl av;;

let fusionner l1 l2 c =
  let m1 = permuter l1 c in
  let m2 = permuter l2 c in
  m1 @ tl m2;;
```

**Question 9)**

```
let cycle_eulerien g =
  let rec cyclaux cy =
    let r = rechercher cy g in
    match r with
    | [] -> cy
    | s :: _ -> let c = cycle s g in
      cyclaux (fusionner cy c s)
  in
  cyclaux [0];;
```

### Partie 3

**Question 1)** On a déjà répondu à cette question en partie 1, question 3c.

**Question 2)** On fait deux cas :

- si les deux sommets $a$ et $b$ ne sont pas liés par une arête, alors on ajoute l'arête $a-b$ au graphe : tous ses sommets sont alors de degrés pairs, donc il possède un cycle eulérien, qui passe donc par cette arête ajoutée : on la retire du cycle, on obtient un chemin eulérien ;
- sinon on ajoute quand même l'arête $a-b$ au graphe : il possède une arête double, ce qui n'est pas habituel, mais rend le même service qu'avant (retirer l'arête est aussi une idée, mais alors on peut obtenir deux composantes connexes dans $G$ et ça complique la vie)

Dans les deux cas, on peut trouver un chemin eulérien.

## Problème 2

**Question 1)** L'ensemble des sous-graphes couvrants connexes de $G$ est non vide (il contient $G$ lui-même) et fini donc l'ensemble des poids de tels sous-graphes est une partie non vide et finie de $\mathbb{R}$, elle possède donc un minimum. Si un sous-graphe réalisant ce minimum contient un cycle, lors en supprimant une quelconque arête

appartenant au cycle, on conserve un sous-graphe connexe couvrant et qui est de poids strictement inférieur : contradiction.

**Question 2)**

a) $G'$ étant connexe, deux sommets quelconques de $G'$ sont reliés par un chemin dans $G'$, donc ils le sont encore dans e($G'$). De plus tout sommet de $G'$ est lié à $a$ dans $G'$ donc dans e($G'$) et on ajoute l'arête $a-b$, donc tout sommet de $G'$ est relié à $b$ dans e($G'$).
Au total, e($G'$) est connexe.

b) À chaque itération, le nombre de sommets en dehors du sous-graphe diminue strictement dans $\mathbb{N}$, ensemble bien fondé, donc l'algorithme termine. De plus, quand il termine, ça signifie que le bord est vide, *i.e.* le graphe contient tous les sommets de $G$ : c'est donc un sous-graphe couvrant. Il est de plus connexe d'après la remarque précédente, car initialement il l'est et chaque itération conserve la connexité.

**Question 3)** Si on choisit à chaque étape l'arête de valeur minimale parmi celles disponibles, on obtient sans nul doute un sous-graphe couvrant de poids minimal, qui est donc connexe par construction et donc acyclique d'après la première question.

**Question 4)**

**Question 5)**

a) Il suffit de réunir les arêtes reliant chaque sommet du sous-graphe à ses voisins et de retirer ensuite les arêtes du sous-graphe.

b) Évidemment, c'est trop coûteux, puisqu'en ajoutant un sommet du bord dans le sous-graphe, on ne change quasiment pas le sous-graphe, donc son bord : il suffit de retirer du bord toutes les arêtes d'extrémités le nouveau sommet, puis d'ajouter les arêtes d'extrémités ce sommet et tout autre point qui n'est pas dans le sous-graphe.

**Question 6)**

```
prim(G) :
on choisit s un sommet de G
on pose G' le sous-graphe ayant pour seul sommet s et aucune arête
on pose B l'ensemble des arêtes issues de s (c'est le bord de G')
tant que B est non vide :
   on choisit une arête dans B de valuation minimale (a l'extrémité dans G', b celle qui n'y est
    pas)
   on ajoute le sommet b et l'arête (a,b) au graphe G'
   on retire l'arête (a,b) de B et toutes les autres arêtes d'extrémité b
   on ajoute à B les arêtes issues de b qui ont l'autre extrémité en dehors de G'
fin du tant que
on retourne G'
```

**Question 7)** Suivant le type retenu pour représenter le bord, la complexité est différente.

Une idée naïve est de prendre pour B une liste : dans ce cas, ajouter à B des arêtes est de complexité constate, mais retirer des arêtes ou choisir une arête de valeur minimale est de complexité linéaire en la longueur de la liste, qui peut potentiellement contenir toutes les arêtes. Donc chaque itération est de complexité $O(m)$, répétée $n$ fois (puisque le graphe construit contient $n$ sommets), donc on a une complexité totale en $O(mn)$.

Une meilleure idée est de choisir une file de priorité ordonnée selon les valeurs des arêtes : dans ce cas, le choix et l'ajout d'arêtes est de complexité $O(\log m)$. Seul problème : on doit aussi retirer les arêtes d'extrémité $b$. Pour le faire au même coût, il suffit de connaître la position de ces arêtes dans la file. Pour cela, on numérote les arêtes, on calcule pour chaque sommet la liste des numéros des arêtes qui en sont issues (coût global $O(n+m)$), autrement dit on remplace les arêtes par des numéros. Plutôt que de travailler directement sur les arêtes, on va travailler sur leurs numéros : ce sont les numéros des arêtes qu'on met dans la file de priorité et on maintient parallèlement à la file proprement dite un tableau qui contient en case $i$ la position de l'arête numéro $i$ dans la file de priorité. Le retrait et l'ajout d'arêtes d'extrémité $b$ coûte alors $O(\deg(b)\log m)$. Le coût total est alors en $\sum_{b\in S} O(deg(b)\log m) = O(m\log m) = O(m\log n)$ car $2m = \sum_b \deg(b)$.

**Question 8)**

a) Le tableau de booléens sert à noter l'appartenance ou non des sommets au sous-graphe : $t[i] = vrai$ si et seulement si $i$ est un sommet du graphe. L'intérêt de cette représentation est qu'on peut facilement ajouter un sommet au sous-graphe et savoir si un sommet appartient ou pas au sous-graphe en complexité constante.

La liste contient en vrac les arêtes entre les sommets. On n'a pas besoin de plus, puisque la seule opération sur les arêtes du sous-graphe est l'ajout d'arêtes : encore en complexité constante.

b)

```
type sommet == int;;
type arete == sommet * int * sommet;;

new : unit -> arete file;;
add : arete -> arete file -> unit;;
take : arete file -> arete;;
delete : arete file -> pos -> unit;;
empty : arete file -> bool;;

(* numeroter les aretes *)
let supprimer_arete a b g =
  let rec supaux l x =
    match l with
    | [] -> []
    | (v,y) :: q -> if y = x then supaux q x else (v,y) :: supaux q x
  in
  g.(a) <- supaux g.(a) b; g.(b) <- supaux g.(b) a;;

let nombre_arete g =
  let n = vect_length g in
  let rec nombraux i m =
   if i = n then m
   else nombraux (i+1) (m + list_length g.(i))
  in
  (nombraux 0 0) / 2;;

let numeroter g =
  let n = vect_length g in
  let g' = make_vect n [] in
  let ta = make_vect (nombre_arete g) (0,0,0) in
  let rec numaux i l k =
    match l with
    | [] -> k
    | (v,x) :: q ->
      g'.(i) <- k :: g'.(i); g'.(x) <- k :: g'.(x);
      ta.(k) <- (i,v,x);
      supprimer_arete i x g;
      numaux i q (k+1)
  in
  let rec numaux2 i k =
    if i = n then ()
    else
      let k' = numaux i g.(i) k in
      numaux2 (i+1) k'
  in
  numaux2 0 0;
  (g', ta);;
(* ta : tableau des aretes,
   ta.(i) = (a,v,b) est l'arete numero i d'extremites a et b et de valeur v
   g' : tableau des listes d'adjacence dans lesquelles
```

```
   on met les numeros des aretes, plutot que les aretes elles-memes
   g.(i) = liste des numeros des aretes d'extremite i *)

(* selectionner dans une liste d'aretes du graphe g
   celles dont une extremite n'est pas dans g'
   t : ensemble des sommets de g' *)
let rec selectionner la t ta =
  match la with
  | [] -> []
  | num :: qa -> let lar = selectionner qa t ta in
    let (x,v,y) = ta.(num) in
    if (t.(x) && not t.(y)) ||  (t.(y) && not t.(x)) then
      num :: lar
    else
      lar;;

(* ajout d'aretes  au bord *)
let rec ajouter l bord =
  match l with
  | [] -> ()
  | num :: q -> add num bord; ajouter q bord;;

(* retrait d'aretes au bord *)
let rec retirer l bord =
 match l with
 | [] -> ()
 | num :: q -> delete num bord; retirer q bord;;

let prim g =
  let n = vect_length g in
  let (h, ta) = numeroter g in
  let g' = (make_vect n false, []) in
  (fst g').(0) <- true;
  let bord = new () in
  ajouter h.(0) bord;
  let rec primaux bord g' =
    let t = fst g' and la = snd g' in
    if empty bord then g'
    else begin
       printars bord ta;
      let ar = take bord ta in
      let (x,v,y) = ta.(ar) in
      (* x : extremite dans g', y : celle en dehors de g' *)
      let x,y = if t.(x) then x,y else y,x in
      t.(y) <- true;
      let la' = ar :: la in
      let lb = selectionner h.(y) t ta in
         retirer h.(y) bord;
         ajouter lb bord;
         primaux bord (t,la')
   end
  in
  primaux bord g';;
```

c) Déjà fait avant.