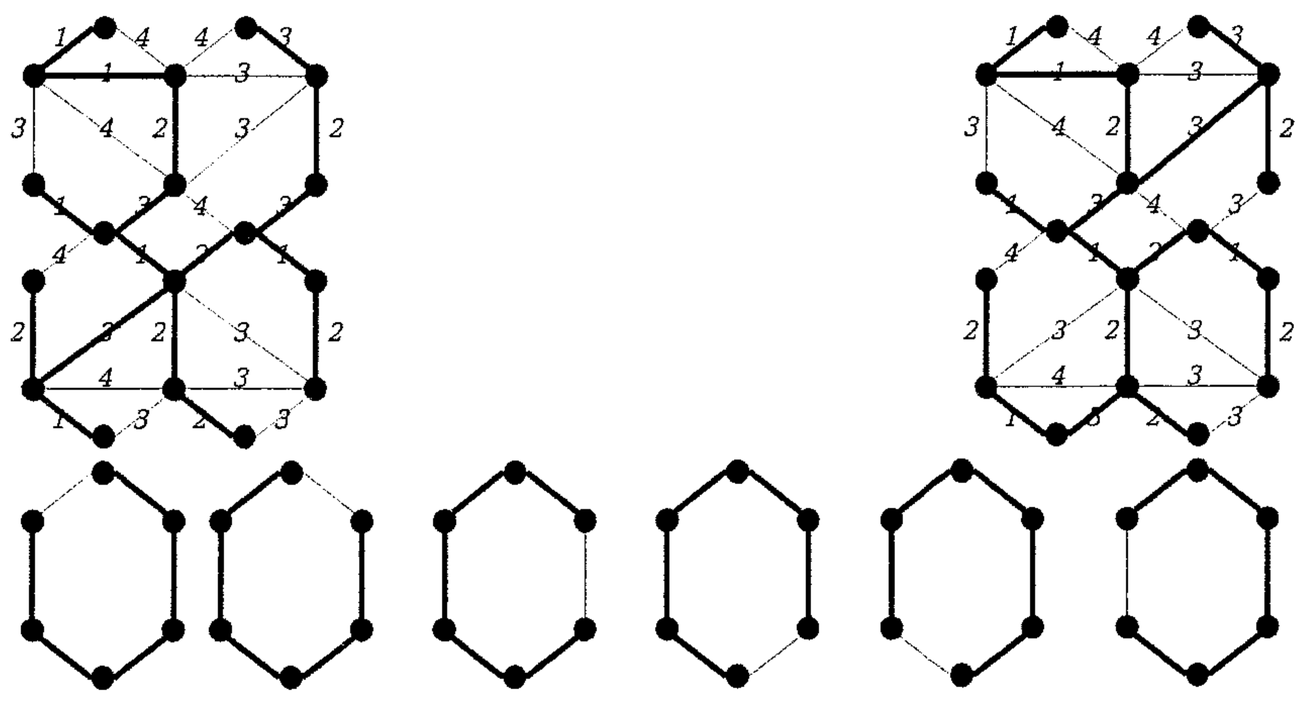
TD n°9 - Correction

Arbres couvrant minimum.

Exercice 1 [**Exemple**] Pour le cycle à 6 sommets, il suffit de supprimer n'importe quelle arête. Plus



généralement dans un graphe non valué (ou à valuation unitaire) tout arbre couvrant est un arbre de poids minimum. Pour le second graphe, il existe au moins deux arbres couvrants de poids minimum différents.

Exercice 2 [**Arête minimum**] Montrons le par l'absurde, supposons que T soit un ACPM et $e \notin A$. soit $e = (x, y)$ l'arête de poids minimum de G. Si l'on regarde dans T le chemin qui relie x à y alors toutes les arêtes sont de poids strictement supérieur à e. Sans perte de généralité prenons e' une arête de ce chemin.
Nous avons donc $\omega(e) < \omega(e')$ donc si nous construisons T'' l'arbre obtenu à partir de T en échangeant e et e', nous avons toujours un arbre et $\omega(T'') < \omega(T)$. Or c'est impossible puisque par hypothèse T est un ACPM. Nous en concluons que l'arête de poids minimum appartient à tout ACPM de G.

La réciproque est trivialement fausse en effet on peut construire un arbre qui contient e l'arête de poids minimum du graphe. Et prendre ensuite n'importe quel ensemble d'arêtes de façon à obtenir un arbre couvrant. De cette manière nous pouvons obtenir un arbre qui n'est pas nécessairement de poids minimum.

Exercice 3 [**Arête maximum**] C'est la preuve de correction. On a tué une arête e du cycle pour avoir un arbre couvrant. Si e est strictement plus légère et si l'on remplace l'arête la plus lourde du cycle (qui reste dans l'ACM par hypothèse) par e le poids diminue, donc on n'a pas un ACM. Donc e est la plus lourde, ou de même poids.

Ca donne l'algorithme de Kruskall à l'envers : en enlève, à chaque fois, l'arête la plus lourde qui n'est pas un isthme (qui ne déconnecte pas le graphe).

Exercice 4 [**variantes**]

1. Arbre couvrant de poids maximum.

Il existe au moins deux solutions. En réalité le problème de l'arbre couvrant de poids maximum et l'ACPM sont "identiques". Nous pouvons transformer le problème maximum en problème d'ACPM. Il suffit pour cela de changer toutes les poids des arêtes par leur inverse (i.e. $\omega(e') = \frac{1}{\omega(e)}$). Nous pouvons montrer que T est un arbre couvrant de poids maximum ssi T' obtenu en changeant les valuations est un arbre de poids minimum.

L'algorithme est identique à celui de Prim, il faut simplement changer le choix min par max.

2. Arbre couvrant de poids min,×.

Il suffit de voir que

$$a \times b \times c = e^{ln(a \times b \times c)} = e^{ln(a)+ln(b)+ln(c)}.$$

Par conséquent l'algorithme est le même, on peut utiliser soit Kruskal soit Prim.

Algorithme et complexité similaire à celle de l'exercice précédent.

Exercice 5 [Implémentation de Prim] on utilise un tas (une file de priorité). Trouver la clé minimale se fait en temps constant ; modifier les clés (y compris extractions) se fait en $O(\log n)$. L'algo est alors :

```
PRIM(G,s)
  Clé : tableau d'entiers indexé par V initialisé à [+∞, +∞,... +∞]
  Pere : tableau de sommets indexé par V initialisé à [null, null....null]
  F : file de priorité initialisée avec les valeurs de Clé

  Tant que la file F est non-vide faire
     u = extraire-min(F)
     Pour tout sommet v voisin de u faire
        Si v est dans la file et p(u,v) < Clé[v]
           Père[v] = u
           Clé[v] =  p(u,v)
           modifier-clé(F,v,p(u,v))
        Fin du si
     Fin du Pour
  Fin du tant que
  Retourner Pere
```

On note n le nombre de sommets et m le nombre d'arêtes. Créer la file prend un temps $O(n)$ (la longueur de la file). A chaque tour dans la boucle `Tant que` on extrait un sommet de la file : il y a exactement n tours. Chaque tour dans cette boucle examine l'adjacence de v. Donc chaque arête (u, v) est vue exactement deux fois : une fois lors du tour de u et une fois lors du tour de v. Le `Si` dans la boucle `Pour tout` sont donc faite exactement $2m$ fois au cours de l'aglorithme. Or ce `Si` contient un test et deux affectations en temps constant, plus un appel à `modifier-clé` en temps $O(\log n)$. On a donc en tout une complexité $O(n + m \log n)$. Comme on suppose le graphe connexe, cela fait $O(m \log n)$

Exercice 6 [implémentation de Kruskal] La complexité de l'algorithme de Kruskal est déterminé par deux étapes importantes dans l'algorithme. Tout d'abord le tri des arêtes qui se fait en $(m\ log\ m)$. Puis l'opération à la ligne 6, à savoir si cela ne crée pas de cycle. En utilisant une structure assez simple de tableau l'union de deux composantes se fait en $O(n)$, comme au pire nous devons le faire n fois, cela nous donne du $O(n^2)$. Cependant il existe une structure approprié pour accomplir cette tâche. Cette structure s'appelle l'union-find. Comme elle est un peu complexe, elle ne sera pas détaillée ici.

```
KRUSKAL(G)
  A : ensemble vide.
  Trier les arêtes de E par ordre croissant de poids.
  Initialiser la structure de données
  Pour toute arête (u,v) de E prise dans l'ordre croissant faire
     Si Find(u,v)
         ajouter (u,v) à A
         Union(u,v)
     Fin du si
  Fin du pour
  Retourner A
```

m Il y a exactement m appels à `Find` et $n-1$ appels à `Union`. On peut donc déjà dire que la complexité est $O(mf+nu)$ où f et u sont els temps d'execution de `Find` et `Union`.

Une implémentation "naïve" est de donner un numéro de composante à chaque sommet. `Find` se fait alors en temps constant mais `Union` demande de renuméroter tous les sommets d'une composante, en temps $O(n)$.

Une autre implémentation "naïve" est de faire des listes chaînées de sommets de chaque composante. Là c'est Union qui se fait alors en temps constant mais Find en temps $O(n)$

La bonne implémentation est d'utiliser des arbres. Chaque sommet d'une composante pointe vers un père (qui peut être lui-même) et deux sommets sont dans la même composante si et seulement si ils sont dans des arbres de même racine. `Find` se fait en $O(h)$: il suffit de remonter à la racine pour les deux. `Union` se fait aussi en $O(h)$: on remonte aux deux racine, puis le père de la racine de v devient la racine de u. Noter que cela augmente de 1 la hauteur de v mais pas celle de u. Si l'on choisit l'arbre de plus petite hauteur pour devenir fils de l'arbre de plus grande hauteur, alors on peut montrer que $h = O(\log n)$. Donc `Union` et `Find` tournent en temps $O(\log n)$.

Exercice 7 [Voyageur de commerce]
- Un chemin couvrant
- un chemin est un arbre donc pese plus que l'arbre min