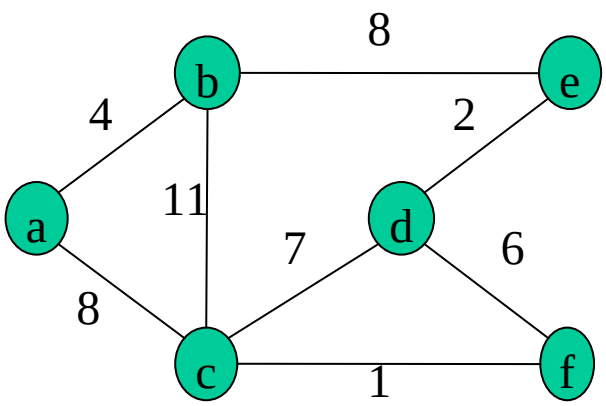
Corrigé : Examen Algorithmique 2

Question 1 : 8 points

On considère le graphe suivant :



On vous demande d'appliquer l'algorithme de Kruskal (voir Question 2) pour trouver l'arbre couvrant de poids minimum pour ce graphe. Vous devez formuler votre réponse en donnant les différentes étapes du déroulement de l'algorithme.

Réponse :

A = {{c,f},{d,e},{a,b},{d,f},{c,d},{a,c},{b,e},{b,c}}

CompteurA	Arrête {x,y}	CompteurT	CC
1		0	{a,b,c,d,e,f}
2	{c,f}	1	{a,b,c,d,e,c}
3	{d,e}	2	{a,b,c,d,d,c}
4	{a,b}	3	{a,a,c,d,d,c}
5	{d,f}	4	{a,a,d,d,d,d}
6	{c,d}		
7	{a,c}	5	{a,a,a,a,a,a}

Arbre couvrant de poids minimum : T = {{c,f},{d,e},{a,b},{d,f},{a,c}}

Question 2 : 6 points

Calculer la complexité en temps pour l'algorithme de Kruskal (ci-dessous) en détaillant votre calcul. La complexité dépend de :

- *n* : le nombre de sommets
- *m* : le nombre d'arrêtes

- l'opération élémentaire qui est l'affectation des entiers.

Rappel : l'algorithme utilisé pour trier les arêtes est « quick sort » et la complexité pour trier un tableau de taille t est *O(tlogt).*

Réponse :

```
Trier les arêtes par poids croissants et les ranger dans le tableau A selon cet ordre.   / /O(mlogm)
Pour i qui varie de 1 à n  faire                                                        // n affectations
        cc(i) = i                                                                       // 1 affectation
Fin Pour
compteurT = 0                                                                           // 1 affectation
compteurA = 1                                                                           // 1 affectation
Tant que compteurT < n -1, faire                                                        // (n-1)*(1+1+1+n*1)
        soit {x, y} l'arête A[compteurA]
        compteurA = compteurA + 1                                                       // 1 affectation
        Si cc(x) ¹ cc(y), alors
                compteurT = compteurT + 1                                               // 1 affectation
                T [compteurT] = {x,y}
                auxiliaire = cc(y)                                                      // 1 affectation
                Pour i qui varie de 1 à n, faire                                        // n*1 affectations
                        si cc(i) = auxiliaire, alors cc(i) = cc(x)                      // 1 affectation
                Fin Pour
        Fin Si
Fin Tant que
```

La complexité totale au pire de cas (tous les tests sont vrais) :

O(mlogm) + n + 2 + (n-1)*(1 + 1 + 1 + n*1)

qui est majorée par O(mlogm + n*n)

Question 3 : 6 points

Soit le problème suivant (problème NP-complet sac-à-dos) :

Nous avons un ensemble de *n* objets, chaque objet *i* vaut *vi* euros et pèse *wi* kilogrammes (*vi* et *wi* sont des entiers). On veut emporter ces objets dans un sac-à-dos donc le poids maximum qu'il peut supporter est W kilogrammes.

Proposer un algorithme le plus simple possible qui permet de déterminer quels objets qu'il faut mettre dans le sac pour avoir la plus grande valeur possible sans dépasser le poids maximum. Est-ce que votre algorithme permet toujours de trouver la solution optimale ? Sinon donner un contre-exemple.

Réponse
L'algorithme le plus simple est un algorithme glouton. L'idée est d'ajouter en priorité les objets les plus chers, jusqu'à saturation du sac. Voici l'algorithme écrit en pseudo-code :

```
trier les objets par ordre décroissant de valeur
w_conso := 0

pour i de 1 à n
  si w[i] + w_conso <= W alors
    x[i] := 1
    w_conso := w_conso + w[i]
  sinon
    x[i] := 0
  fin si
fin pour
```

Cet algorithme ne donne pas toujours la solution optimale. Un contre-exemple :
W = 8 kg
n = 5 (5 objets)

	1er objet	2ème objet	3ème objet	4ème objet	5ème objet
vi	100E	80E	70E	60E	10E
wi	3kg	4kg	2kg	2kg	2kg

Selon l'algorithme, on va commencer par les objets donc la valeur est la plus grande. On va mettre dans le sac le 1er et le 2ème objet :

- somme totale = 180E,
- poids = 7kg.

Par contre, la solution optimale est : 1er, 3ème et 4ème objet avec

- somme totale = 230E
- poids = 7kg