Solutions : Chapitre 9

Question # 1

Élaborez un algorithme vorace pour résoudre le problème de rendre la monnaie avec un montant de n et des dénominations de $d_1 > d_2 > \cdots > d_m$. Quelle est la classe d'efficacité en temps de votre algorithme en fonction de m ?

Solution :

Algorithme 1 : RendreMonaieVorace($n, D[1 \ldots m]$)

```
1 pour i = 1..m faire
2 |   N[i] ← ⌊n / D[i]⌋
3 |   n ← n mod D[i]
4 si n = 0 alors
5 |   retourner N
6 sinon
7 |   retourner «Aucune solution»
```

L'efficacité de cet algorithme est $\Theta(m)$. Si on arrête l'algorithme dès que n atteint la valeur 0, l'efficacité devient alors $O(m)$.

***Question # 2**

Soit le problème de construire un horaire pour l'exécution de n tâches par un processeur. Les tâches requièrent des temps $t_1, t_2, \ldots, t_n$ pour s'exécuter. Les tâches peuvent être exécutées dans n'importe quel ordre, une seule à la fois. Il faut construire un horaire d'exécution des tâches afin de réduire le temps total passé dans le système pour toutes les tâches. Le temps passé dans le système pour une tâche est le temps d'attente de cette tâche plus son temps d'exécution t_i.

A) Élaborez un algorithme vorace pour résoudre ce problème.

Solution :

Trier les tâches en ordre croissant de leur temps d'exécution et les exécuter dans cet ordre.

B) Est-ce que l'algorithme vorace produit toujours une réponse optimale ?

Solution :

Oui. Soit $p_1, p_2, \ldots, p_n$ une permutation des tâches. On exécute donc la tâche p_1 suivie de la tâche p_2 et ainsi de suite. Soit s_i le temps de traitement de la i-ème tâche à être exécutée, c'est-à-dire que $s_i = t_{p_i}$. Le temps total passé dans le système pour la permutation p est donc donné par l'équation qui suit.

$$T(p) = \sum_{i=1}^{n} \sum_{j=1}^{i} s_j = s_1 + (s_1 + s_2) + (s_1 + s_2 + s_3) + \ldots + (s_1 + \ldots + s_n)$$

$$= ns_1 + (n-1)s_2 + (n-2)s_3 + \ldots + 2s_{n-1} + s_n = \sum_{i=1}^{n} (n - i + 1)s_i$$

Supposons que pour la permutation p, il existe un a et un b tels que $a < b$ et $s_a > s_b$, c'est-à-dire que la a-ième tâche a un temps de traitement plus grand que la b-ième tâche et que le a-ième tâche est exécutée avant la b-ième tâche. Considérons la permutation p' où la tâche à la position a

a été échangée avec celle à la position b. Nous avons donc

$$T(p) = \sum_{i=1}^{n}(n-i+1)s_i = (n-a+1)s_a + (n-b+1)s_b + \sum_{i\notin\{a,b\}}(n-i+1)s_i$$
$$T(p') = \sum_{i=1}^{n}(n-i+1)s'_i = (n-a+1)s'_a + (n-b+1)s'_b + \sum_{i\notin\{a,b\}}(n-i+1)s'_i$$

Puisque $s_i = s'_i$ pour tout $i \notin \{a, b\}$, nous avons que

$$T(p) - T(p') = (n-a+1)(s_a - s'_a) + (n-b+1)(s_b - s'_b).$$

Puisque $s'_a = s_b$ et $s'_b = s_a$, nous avons que

$$T(p) - T(p') = (n-a+1)(s_a - s_b) + (n-b+1)(s_b - s_a) = (b-a)(s_a - s_b)$$

Finalement, puisque $b > a$ et $s_a > s_b$, nous pouvons conclure que

$$T(p) - T(p') > 0;$$
$$T(p) > T(p').$$

Nous avons donc prouvé que pour toute permutation où les temps de traitement ne sont pas triés en ordre non-décroissant, il existe une autre permutation dont le temps total passé dans le système est plus court. Conséquemment, seule la permutation des tâches triée en ordre de temps de traitement peut être optimale.

Question # 3

Lequel des algorithmes, Kruskal ou Prim, fonctionnera correctement sur un graphe contenant des arêtes de poids négatif ?

Solution :

Les deux algorithmes fonctionnent correctement sur un graphe contenant des arêtes de poids négatif.

À titre indicatif, il est possible de transformer un graphe g contenant des poids négatifs en un graphe g' n'en contenant pas en ajoutant v au poids de toutes les arêtes, où v est la valeur absolue de l'arête ayant le poids le plus petit (donc le plus négatif). Tout arbre de recouvrement minimal sur g' est un arbre de recouvrement minimal sur g et vice versa.

Question # 4

Vrai ou faux :

A) *Si e est une arête de coût minimal parmi les arêtes d'un graphe connexe valué g, alors e doit faire partie d'au moins un arbre de recouvrement minimal de g.*
Solution :
Vrai.

B) *Si e est une arête de coût minimal parmi les arêtes d'un graphe connexe valué g, alors e doit faire partie de tous les arbres de recouvrement minimal de g.*
Solution :
Faux.

C) *Si les poids de toutes les arêtes du graphe connexe valué g sont tous distincts, alors g a exactement un arbre de recouvrement minimal.*
Solution :
Vrai.

D) *Si les poids de toutes les arêtes du graphe connexe valué g ne sont pas tous distincts, alors g a plus qu'un arbre de recouvrement minimal.*
Solution :
Faux.

Question # 5
Élaborez un algorithme pour trouver un arbre de recouvrement ***maximal*** *pour un graphe connexe valué.*
Solution :
Soit un graphe g. On construit le graphe g' de la façon suivante : g' a exactement les même noeuds que g ; pour chaque arête $a(n_1, n_2)$ de valeur p_a dans g, ajouter une arête $a(n_1, n_2)$ de valeur $-p_a$ à g'. Ensuite, appliquer l'algorithme de Prim ou celui de Kruskal qui peuvent construire un arbre de recouvrement minimal sur un graphe valué négativement (voir la solution à la question 3). Un arbre de recouvrement minimal sur g' est un arbre de recouvrement maximal sur g.

Question # 6
A) *Réécrivez l'algorithme de Kruskal en utilisant les opérations sur les structures d'ensembles disjoints.*
Solution :

Algorithme 2 : KruskalDisjoints($G = \langle V, E \rangle$)

```
1  Trier les arêtes en ordre non décroissant de leur valeur
2  R ← creerEnsemblesDisjoints(V)
3  E_t ← ∅ ; k ← 0
4  c ← 0
5  tant que c < |V| − 1 faire
6      k ← k + 1
7      {u, v} ← e_k            //e_k est la k^ieme plus petite arête.
8      r_u ← Trouver(u, R)
9      r_v ← Trouver(v, R)
10     si r_u ≠ r_v alors
11         E_t ← E_t ∪ {e_k}
12         c ← c + 1
13         Fusionner(r_u,r_v,R)
14 retourner E_t
```

La première étape de l'algorithme de Kruskal est le tri des arêtes en ordre non décroissant des poids. La fonction ***creerEnsemblesDisjoints*** *crée ensuite la structure d'ensembles disjoints R sur les noeuds de V. À chaque itération, nous représenterons R par une forêt où chaque arbre représente un ensemble. Pour chaque arbre (graphe dirigé acyclique), la racine est le représentant de l'ensemble.*

On initalise ensuite E_t à l'ensemble vide et les compteurs c et k à 0. Le compteur k nous indique la position actuelle dans le vecteur trié des arêtes. Le compteur c nous indique le nombre

d'arêtes ajoutées dans E_t.

Les arêtes sont ensuites parcourues en ordre non décroissant de leur coût tant que $c < |V| - 1$. Notons qu'un arbre couvrant d'un graphe G connexe a exactement $|V| - 1$ arêtes. Si, à une itération donnée, l'ajout de l'arête (u, v) dans E_t ne crée pas de cycle, alors (u, v) fait partie de l'arbre de recouvrement minimal E_t.

La vérification du cycle s'effectue à l'aide de la condition Trouver$(v, R) \neq$ Trouver(u, R), où Trouver(x, R) retourne le représentant auquel appartient le sommet $x \in V$. Si le représentant de l'ensemble auquel appartient v est le même que celui de l'ensemble auquel appartient u alors l'ajout de (u, v) dans E_t créera un cycle et on passe à l'arête suivante. Si les représentants ne font pas partie du même ensemble, l'ajout de (u, v) dans E_t ne créera pas de cycle. On utilise donc la fonction Fusionner(u, v, R) qui permet la fusion des ensembles disjoints auxquels appartiennent les sommets u et v. L'arbre couvrant minimal en sortie est représenté par un ensemble d'arêtes nommé E_t.

B) *Soit le graphe connexe valué $G = (V, E)$ tel que représenté à la Figure 1. Nous avons que :*

1. $V = \{A, B, C, D, E, F, G\}$;
2. $E = \{(A, B), (A, D), (A, E), (B, C), (B, D), (C, D), (C, E), (C, G), (D, E), (E, F), (F, G)\}$,

La fonction de coût, représentées sur les arêtes, est : $c(A, B) = 1$, $c(A, D) = 2$, $c(A, E) = 8$, $c(B, C) = 3$, $c(B, D) = 12$, $c(C, D) = 1$, $c(C, E) = 5$, $c(C, G) = 15$, $c(D, E) = 10$, $c(E, F) = 9$ *et* $c(F, G) = 6$.

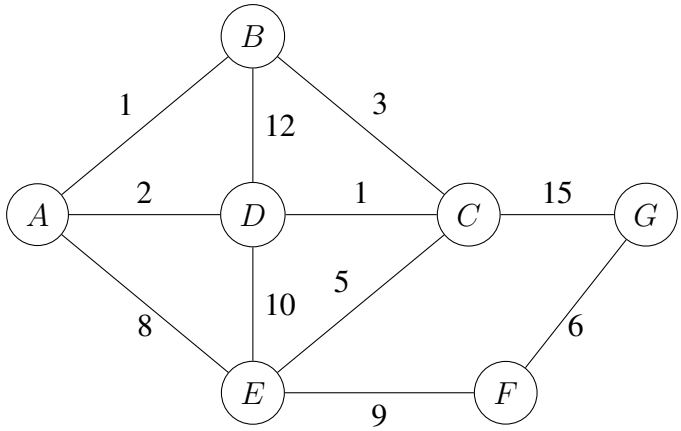


FIGURE 1 – Un graphe valué G

Effectuez une trace de votre algorithme en utilisant le graphe de la Figure 1. Notez, pour chacune des itérations de la boucle principale, l'arête traitée, l'état de la structure des ensembles disjoints et l'arbre couvrant actuel.

Solution :

L'ordre des arêtes obtenu après le tri initial est : (A, B), (C, D), (A, D), (B, C), (C, E), (F, G), (A, E), (E, F), (D, E), (B, D), (C, G)

Chacune des itérations de la boucle est illustrée dans le Tableau 1. Notez que les appels à Trouver font également des compressions dans la structure d'ensembles disjoints. L'arbre couvrant minimal résultant est $E_t = \{(A, B), (C, D), (A, D), (C, E), (F, G), (E, F)\}$. *L'ensemble est ordonné dans l'ordre des ajouts.*

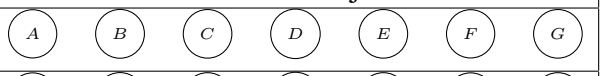
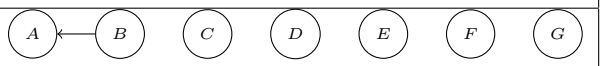
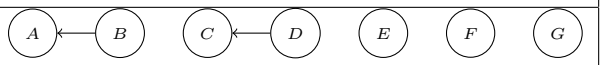
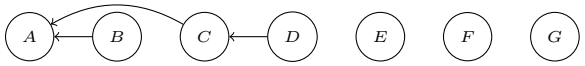
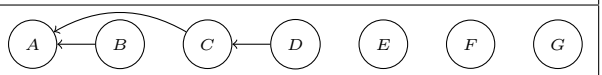
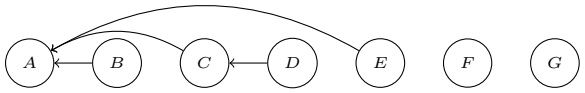
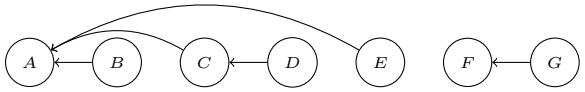
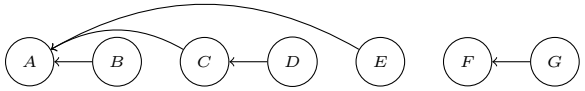
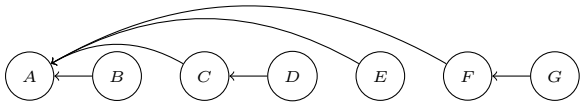
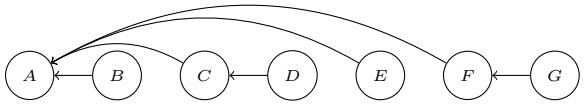
c	e_k	r_u	r_v	E_t	Ensembles disjoints
0	-	-	-	-	
0	(A, B)	A	B	(A, B)	
1	(C, D)	C	D	(A, B), (C, D)	
2	(A, D)	A	C	(A, B), (C, D), (A, D)	
3	(B, C)	A	A	(A, B), (C, D), (A, D)	
3	(C, E)	A	E	(A, B), (C, D), (A, D), (C, E)	
4	(F, G)	F	G	(A, B), (C, D), (A, D), (C, E), (F, G)	
5	(A, E)	A	A	(A, B), (C, D), (A, D), (C, E), (F, G)	
5	(E, F)	A	G	(A, B), (C, D), (A, D), (C, E), (F, G), (E, F)	
6	-	-	-	(A, B), (C, D), (A, D), (C, E), (F, G), (E, F)	

TABLEAU 1 – Trace de l'algorithme de Kruskal avec la structure d'ensemble disjoints 2 avec la Figure 1 comme graphe de départ.