Algorithmique
Correction Partiel n° 2

Info-Spé – Epita

10 mai 2011 - 09 :00

Solution 1 **(Gisement épuisant...–** ***5 points*****)**

1. La solution est un arbre couvrant du graphe d'origine.
2. 9
3. Une possibilité serait par exemple le graphe de la figure 1.

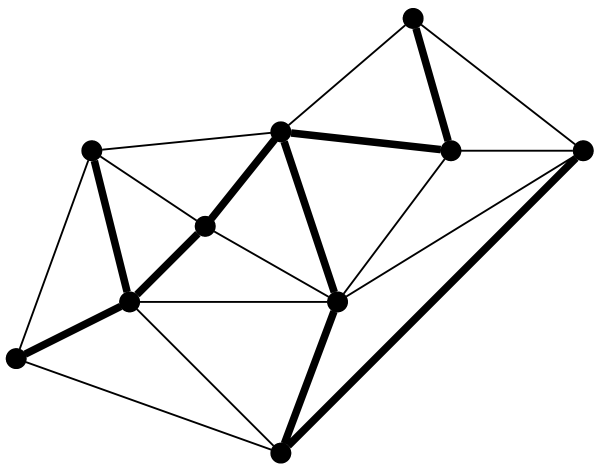

Fig. 1 – Sous-Graphe couvrant du graphe de la figure 1.

4. $N - 1$
5. La solution recherchée est un arbre couvrant du graphe d'origine et une des propriétés des arbres est d'être sans cycle avec $N - 1$ arêtes.
6. En recherchant un arbre de recouvrement de poids minimum.
7. Une possibilité serait par exemple le graphe de la figure 2.
8. Non
9. Les coûts des arêtes ne sont pas distincts deux à deux, il n'y a donc pas unicité d'ARPM.

Solution 2 **(Mangez des crêpes –** ***16 points*****)**

1. La figure 3 est le graphe représentant la recette.
2. **Le cuisinier est tout seul en cuisine :**

   (a) Une solution de tri topologique : *debut* - D - A - E - B - F - C - G - I - H - J - *fin*.

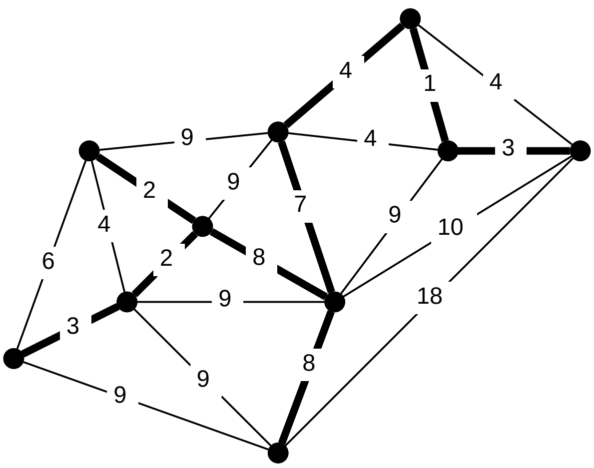

Fig. 2 – Sous-Graphe couvrant de poids minimum du graphe de la figure 2.

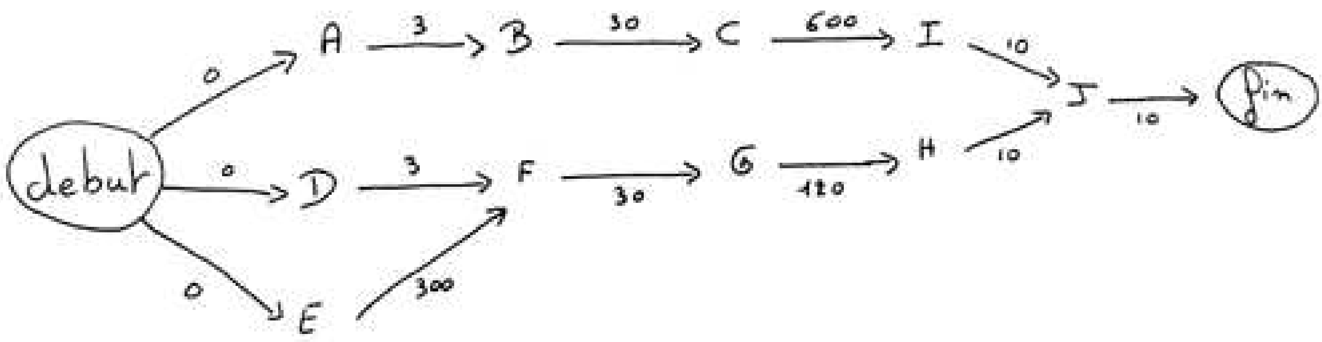

Fig. 3 – Crêpe à la banane flambée !

(b) Lors du parcours en profondeur, il suffit de prendre les sommets en ordre suffixe inverse.

(c) **Spécifications :** La procédure `tri_topo` (G, tri) donne le tri topologique (dans la pile tri) obtenu à partir du premier sommet du graphe G.

```
algorithme procedure tri_topo_rec
    parametres locaux
        t_listsom    ps
    parametres globaux
        t_pile            tri
        t_vect_booleens   marque

    variables
        t_listadj    pa
debut
    marque[ps↑.som] ← vrai
    pa ← ps↑.succ
    tant que pa <> NUL faire
        si non marque[pa↑.vsom↑.som] alors
            tri_topo_rec (pa↑.vsom, tri, marque)
        fin si
        pa ← pa↑.suiv
    fin tant que
    tri ← empiler (ps, tri)
fin algorithme procedure tri_topo_rec
```

```
algorithme procedure tri_topo
    parametres locaux
        t_graphe_d   G
    parametres globaux
        t_pile   tri    /* contient des t_listsom */

    variables
        t_vect_booleens   marque
        entier            i
debut
    pour i ← 1 jusqu'a G.ordre faire
        marque[i] ← faux
    fin pour
    tri ← pile_vide ()
    tri_topo_rec (G.lsom, tri, marque)      /* la source est le 1er sommet */
fin algorithme procedure tri_topo
```

3. **Le cuisinier a trouvé de l'aide :**

(a) La recherche du plus long chemin depuis la tâche de *debut* donne les **dates au plus tôt** pour chaque tâche.

Dates au plus tôt pour la recette :

debut	A	B	C	D	E	F	G	H	I	J	*fin*
0	0	3	33	0	0	300	330	450	633	643	653

(b) **Durée minimale avant de pouvoir déguster la crêpe :** 653 secondes (soit 10mn53s).

Le plus long chemin du début à la fin du projet donne sa durée minimale.

(c) Pour obtenir les **dates au plus tard**, il faut considérer le graphe inverse (où l'on inverse le sens des arcs !) et rechercher les plus longs chemins depuis la tâche de fin : la date au plus tard d'une tâche est la différence entre la durée minimale du projet et la longueur du chemin obtenu.

Dates au plus tard pour la recette :

debut	A	B	C	D	E	F	G	H	I	J	*fin*
0	0	3	33	480	183	483	513	633	633	643	653

(d) Les tâches critiques sont celles qui ont les mêmes dates au plus tôt et au plus tard.

Les tâches critiques sont : *debut*, A, B, C, I, J et fin.

(e) **Spécifications :** La fonction `duree_minimale` (G, *source*, *finale*) calcule la longueur du plus long chemin (la durée minimale du projet) dans le graphe G, entre les tâches *source* et *finale*.

```
algorithme fonction duree_minimale : reel
    parametres locaux
        t_graphe_d    G
        entier        s, f          /* ne sont pas obligatoires ici ! */

    variables
        entier           i, a
        t_vect_reels     dist
        t_pile_listsom   tri
        t_listadj        pa
        t_listsom        ps
debut
    pour i ← 1 jusqu'a G.ordre faire                /* init */
        dist[i] ← +∞
    fin pour

    tri_topo (G , tri)
    ps ← sommet (tri)
    tri ← depiler (tri)          /* le premier sommet est la source ! */
    faire
        pa ← ps↑.succ
        tant que pa <> NUL faire
            a ← pa↑.vsom↑.som
            si dist[s] -¹ pa↑.cout < dist [a] alors      /* relâchement arc sortant (s,a)*/
                dist[a] ← dist[s] - pa↑.cout
                pere[a] ← s
            fin si
            pa ← pa↑.suiv
        fin tant que
        ps ← sommet(tri)          /* ici ou au début, peu importe */
        s ← ps↑.som              /* le dernier sommet n'a pas de successeurs */
        tri ← depiler (tri)
    tant que non est-vide (tri)

    retourne (-dist[f])     /* ou -dist[s] */
fin algorithme fonction duree_minimale
```

Solution 3 Construire un ARPM par suppression – *9 points*

1. Il suffit d'effectuer un parcours profondeur depuis l'un des deux sommets dont on doit tester la connexité, si un des successeurs du sommet courant est le sommet de destination ou si l'appel récursif sur ce successeur renvoie vrai l'algorithme renvoie vrai, sinon, il renvoie faux.

2.
```
algorithme fonction lie_rec : booleen
    parametres locaux
        entier            dst
        t_listsom         ps
        t_mat_entiers     T
    parametres globaux
        t_vect_booleens   M
```

[1] Les coûts sont considérés positifs.

```
        variables
            t_listadj        pa
            entier           s,  sa
    debut
        s ← ps↑.som
        M[s] ← vrai
        pa ← ps↑.succ
        tant que (pa <> NUL) faire
            sa ← pa↑.vsom↑.som
            si (T[s, sa] = 0) et non M[sa] alors
                si (sa = dst) ou lie_rec(dst, pa↑.vsom, T, M) alors
                    retourne vrai
                fin si
            fin si
            pa ← pa↑.suiv
        fin tant que
        retourne faux
    fin algorithme fonction lie_rec
```

3. L'algorithme de suppression s'arrête lorsqu'il ne reste plus que $ordre - 1$ arêtes dans l'arbre.

4.

```
        algorithme procedure revdel
            parametres locaux
                t_graphe_d       g
            parametres globaux
                ensemble         E
                t_mat_entiers    T

            variables
                t_listsom   ps
                arete       a
                entier      s, sa, nba, i
        debut
            pour s ← 1 jusqu'a g.ordre faire
                pour sa ← 1 jusqu'a g.ordre faire
                    T[s, sa] ← 0
                fin pour
            fin pour
            nba ← card(E)
            tant que non est_vide(E) et (nba > g.ordre - 1) faire
                a ← supprime_max(E)
                ps ← a↑.src
                s ← ps↑.som
                sa ← a↑.dst↑.som
                T[s, sa] ← 1
                T[sa, s] ← 1
                si lie(g, sa, ps, T) alors
                    nba ← (nba - 1)
                sinon
                    T[s, sa] ← 0
                    T[sa, s] ← 0
                fin si
            fin tant que
        fin algorithme procedure revdel
```

5. $(3, 13)(2, 6)(11, 4)(8, 1)(6, 14)(7, 10)$ voir figure 4

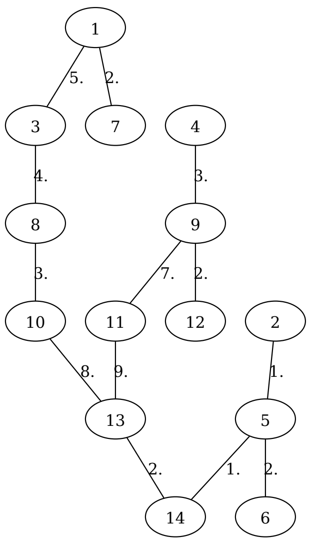

FIG. 4 – Solution ARPM