Algorithmique
Correction Partiel n° 2

API – Epita

14 mai 2019

Solution 1 **(Gisement épuisant... –** ***3 points*****)**

1. Une possibilité serait par exemple le graphe de la figure 1.

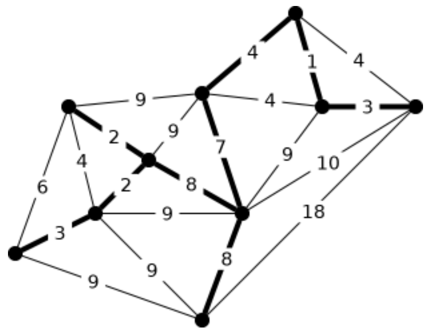

Figure 1 – Arbre couvrant de poids minimum du graphe de la figure 2.

2. Non, la solution n'est pas unique.
   Les coûts des arêtes ne sont pas distincts deux à deux, il n'y a donc pas unicité d'ARPM.

Solution 2 **(Floyd revisité –** ***3 points*****)**

1. La modification est simple. Un circuit absorbant va renvoyer un coût négatif sur la distance calculée d'un sommet x à ce même sommet x (circuit et absorbant). Il suffit donc de tester, lorsque le sommet $x = y$, si la valeur de distance calculée est négative. Si c'est le cas, on provoque le débranchement de la procédure.

2. Là encore l'utilisation n'est pas très compliquée. On utilise la matrice renvoyant les plus petites distances pour chaque couple de sommets (x, y) du graphe. Pour chaque sommet x de 1 à n, on calcule sa valeur d'excentricité en conservant sa plus grande valeur de distance avec les autres sommets (le max de ses plus petites distances). Il ne reste plus alors qu'à comparer les n excentricités calculées (une pour chaque sommet) et de déterminer la plus petite. Le sommet auquel elle appartient est le centre du graphe.

Solution 3 **(Graphes bipartis (Bipartite graph) –** ***5 points*****)**

Une autre manière de voir le graphe biparti : ses sommets peuvent être "colorés" en deux couleurs de sorte que deux sommets de même couleurs ne sont pas adjacents.

Principe

Pour tester si un graphe est biparti, il suffit de faire un parcours en largeur ou en profondeur en vérifiant que pour chaque arête empruntée, le sommet de départ n'est pas dans le même ensemble que le sommet d'arrivée. Il faut utiliser un système de marquage à deux valeurs (-1,1) afin de distinguer chaque ensemble.

Le parcours est ici fait en largeur. Dès qu'un sommet non marqué est trouvé, il prend la marque opposée de celle de son père. Si un sommet déjà marqué a la même marque que son prédécesseur le parcours s'arrête (le graphe n'est pas biparti).

Si aucune arête ne relie deux sommets de même marque (le parcours n'a pas été interrompu), le graphe est biparti.

Spécifications :

La fonction `bipartite(`G`)` indique si le graphe non orienté G est biparti.

```python
def __bipartiteBFS(G, s, Set):
    q = queue.Queue()
    q.enqueue(s)
    Set[s] = 1
    while not q.isempty():
        s = q.dequeue()
        for adj in G.adjlists[s]:
            if Set[adj] == 0:
                Set[adj] = -Set[s]
                q.enqueue(adj)
            else:
                if Set[adj] == Set[s]:
                    return False
    return True

#---------------------------------------------------------------

def bipartiteBFS(G):
    Set = [0] * G.order
    for s in range(G.order):
        if Set[s] == 0:
            if not __bipartite(G, s, Set):
                return False
    return True
```

```python
def __bipartite(G, s, Set):
    for adj in G.adjlists[s]:
        if Set[adj] == 0:
            Set[adj] = -Set[s]
            if not __bipartite(G, adj,
 Set):
                return False
        else:
            if Set[adj] == Set[s]:
                return False
    return True

#---------------------------------------------------------------

def bipartite(G):
    Set = [0] * G.order
    for s in range(G.order):
        if Set[s] == 0:
            Set[s] = 1
            if not __bipartite(G, s, Set):
                return False
    return True
```

Solution 4 **(Mangez des crêpes – *8 points*)**

1. Le graphe représentant la recette :

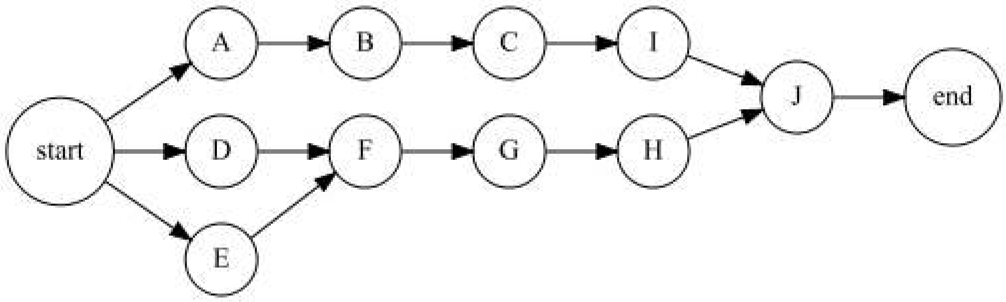

Figure 2 – Crêpe à la banane flambée !

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*.

   (b) **Spécifications :** La fonction `tri_topo` (G) retourne une solution de tri topologique pour le graphe G sans circuit, dont tous les sommets sont atteignables depuis le sommet 0.

```
def dfsSuff(G, s, M, L):
    M[s] = True
    for adj in G.adjlists[s]:
        if not M[adj]:
            dfsSuff(G, adj, M, L)
    L.insert(0, s)

def topologicalOrder(G):
    M = [False] * G.order
    L = []
    dfsSuff(G, 0, M, L)
    return L
```

   (c) **Spécifications :** La fonction `is_tri_topo` (G, L) vérifie si L peut être une solution de tri topologique pour le graphe G sans circuit. La liste L peut être "détruite"...

```
def testTopologicalOrder(G, L):
    M = [False]*G.order
    while L != []:
        s = L.pop()
        for adj in G.adjlists[s]:
            if not M[adj]:
                return False
        M[s] = True
    return True
```

Solution 5 **(What does it do ? – *4 points*)**

1. Résultat retourné par `build(`G_3`)` (vecteur des demi-degrés intérieurs des sommets de G_3) :

	0	1	2	3	4	5	6	7	8
`V`	1	2	1	1	0	3	2	1	2

2. La fonction `what` :
   (a) `what(`G_3`)` retourne :
   la liste $[4, 3, 7, 6, 0, 1, 2, 5, 8]$
   (b) `what(`G`)` représente une solution de tri topologique pour G.

   (c) Propriété de G pour que `what(`G`)` ne "plante" pas ?

   G doit être sans circuit.