Algorithmique
Correction Partiel n° 4 (P4)

Info-spé S4 – Epita

14 mai 2019

Solution 1 **(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.

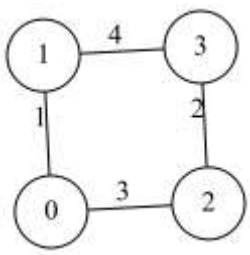
Solution 2 **(ARM ou non ? –** ***2 points*****)**

NON, le graphe partiel T n'est pas un ARM de G!

Le principe énoncé ressemble à celui de Kruskal, mais la différence est que la construction se fait à l'envers. Le graphe est découpé en deux à chaque fois sans se préoccuper des arêtes de plus faibles coûts.

Prenons l'exemple de la figure suivante, si le sous-ensemble $S_1 = \{0, 2\}$ et le sous-ensemble $S_2 = \{1, 3\}$, nous avons alors les ARMs $T_1 =< S_1, A_1 = \{\{0, 2\}\} >$ et $T2 =< S2, A_2 = \{\{1, 3\}\} >$. Si l'on choisit l'arête de plus faible coût $\{0, 1\}$ permettant de relier ces deux ARMs, nous n'obtenons pas un ARM de G puisqu'il aurait fallu choisir l'arête $\{2, 3\}$ au lieu de l'arête $\{1, 3\}$.

En fait le problème est essentiellement du à la découpe arbitraire de S en deux-sous-ensemble de sommets. Toujours sur cet exemple, si la découpe de S avait donnée $S_1 = \{0, 1\}$ et $S_2 = \{2, 3\}$, il n'y avait pas de problème, le choix de l'arête de liaison au plus faible coût (à savoir $\{0, 2\}$ nous garantissait d'obtenir un ARM T de G.



Solution 3 **(Mangez des crêpes –** ***11 points*****)**

1. Graphe représentant la recette :

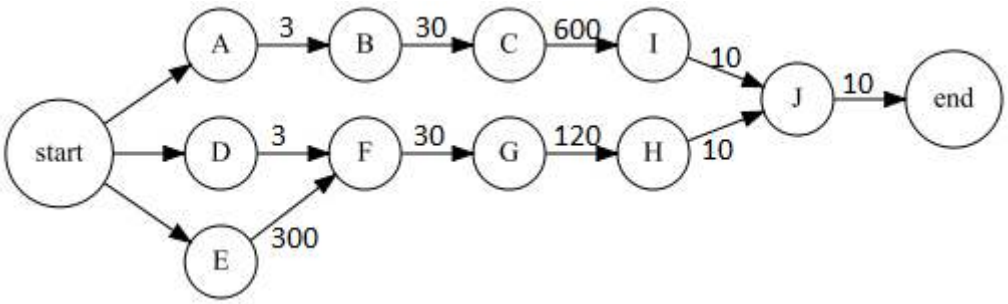


FIGURE 1 – Crêpe à la banane flambée

2. **Le cuisinier est tout seul en cuisine :**

   (a) **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.

```python
def topologicalSort(G):
    # compute half-degrees
    dIn = [0] * G.order
    for s in range(G.order):
        for adj in G.adjlists[s]:
            dIn[adj] += 1

    # 0 is the only vertex without predecessors
    sort = [0]
    first = 0
    while len(sort) < G.order:
        s = sort[first]
        dIn[s] = -1
        for adj in G.adjlists[s]:
            dIn[adj] -= 1
            if dIn[adj] == 0:
                sort.append(adj)
        first += 1
    return sort
```

3. **Le cuisinier a trouvé des aides, les tâches peuvent donc être réalisées en parallèle.**

   (a) Les plus longs chemins depuis la tâche de début (*start*) donne les **dates au plus tôt** pour chaque tâche.

   Dates au plus tôt pour la recette :

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

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

   (c) Pour obtenir les **dates au plus tard**, il faut considérer le graphe inverse 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.

(d) **Spécifications :**
La fonction `duration(G)` calcule la durée minimale du projet représenté par le graphe G.

```
def duration(G):
    """
    return dist: the longest paths in G
    """
    L = topologicalSort(G)
    dist = [0] * G.order
    for i in range(G.order-1):
        x = L[i]
        for y in G.adjlists[x]:
            if dist[x] + G.costs[(x, y)] > dist[y]:
                dist[y] = dist[x] + G.costs[(x, y)]
    return dist[L[-1]]  # the last vertex is the end of the project
```

Solution 4 **(Prim, tout simplement – *5 points*)**

```
def Prim(G):
    """
    G is connected
    return a MST of G
    """

    T = graph.Graph(G.order, directed = False, costs = True)
    costs = [inf] * G.order
    p = [-1] * G.order
    M = [False] * G.order
    H = heap.Heap(G.order)
    costs[0] = 0
    x = 0
    for _ in range(G.order-1):  # graph is connected
        for y in G.adjlists[x]:
            if not M[y] and G.costs[(x, y)] < costs[y]:
                costs[y] = G.costs[(x, y)]
                p[y] = x
                H.update(y, costs[y])
        (_, x) = H.pop()
        M[x] = True
        T.addedge(x, p[x], costs[x])
    return T
```