Algorithmique
Correction Partiel n° 4 (P4)

Info-spé S4 – Epita

15 mai 2018 - 10 : 00

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

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

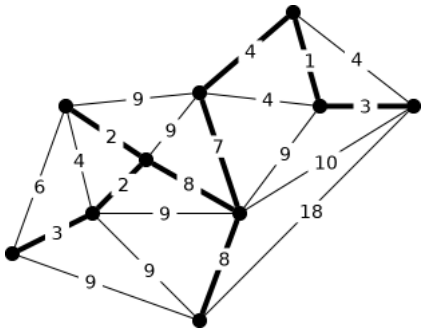


Figure 1 – Arbre couvrant de poids minimum du graphe de la figure 2.

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* (Asterix et le devin – *13 points*)**

1. **L'algorithme :**

   (a) *Quel est le nom de cet algorithme ?* C'est l'algorithme A* ("Astar" ou "A étoile" !)

   (b) *Ouverts :* Un tas sera utilisé, pour optimiser la recherche (et suppression) du minimum.
   fermés : Un simple vecteur de booléens (list). Il permettra de ne pas revenir sur les sommets "fermés" (une fois enlevé du tas).

   (c) ***Complexité** de l'algorithme :* Avec le tas, l'algorithme est (comme Dijkstra) en $O(p\log(n))$

   (d)

```python
def __buildPath(src, dst, p):
    path = [dst]
    while dst != src:
        dst = p[dst]
        path.insert(0, dst)
    return path

def asterix(G, src, dst):
    dist = [inf] * G.order
    p = [-1] * G.order
    closed = [False] * G.order
    dist[src] = 0
    openHeap = heap.Heap(G.order)
    x = src
    while x != dst:
        closed[x] = True
        for y in G.adjlists[x]:
            if not closed[y] and dist[x] + G.costs[(x, y)] < dist[y]:
                dist[y] = dist[x] + G.costs[(x, y)]
                p[y] = x
                openHeap.update(y, dist[y] + heuristix(G, y, dst))
        if openHeap.isEmpty():
            raise Exception("dst not reachable")
        (_, x) = openHeap.pop()
    return __buildPath(src, dst, p)

def astar2(G, src, dst, heur):
    dist = [inf] * G.order
    p = [-1] * G.order
    closed = [False] * G.order
    dist[src] = 0
    openHeap = heap.Heap(G.order)
    openHeap.push(src, 0)
    while not openHeap.isEmpty():
        (_, x) = openHeap.pop()
        if x == dst:
            return (dist, p)
        closed[x] = True
        for y in G.adjlists[x]:
            if not closed[y] and dist[x] + G.costs[(x, y)] < dist[y]:
                dist[y] = dist[x] + G.costs[(x, y)]
                p[y] = x
                openHeap.update(y, dist[y] + heuristix(G, y, dst))
    if dist[dst] == inf:
        raise Exception("dst not reachable")
    return __buildPath(src, dst, p)
```

2. **Les devins :**

   (a) ⋆ **Heuristix le Hollandais** (HeuristixD)
   Les sommets traités (dans l'ordre) : 1 3 6 7 2 8 5 9 4 10

   Le résultat :

	1	2	3	4	5	6	7	8	9	10
dist	0	4	1	6	5	2	3	4	5	6
pere	-1	1	1	2	2	3	6	7	8	5

   ⋆ **Heuristix du Nouveau Monde** (HeuristixM)
   Les sommets traités (dans l'ordre) : 1 3 2 5 10

   Le résultat :

	1	2	3	4	5	6	7	8	9	10
dist	0	4	1	6	5	2	∞	∞	∞	6
pere	-1	1	1	2	2	3	-1	-1	-1	5

   (b) La solution avec HeuristixD correspond à l'algorithme de Dijkstra.

   (c) *L'estimation d'HeuristixM est meilleurs que celle d'HeuristixB car :*
   — Elle donne le plus court chemin (plus efficace) ;
   — en utilisant un minimum de sommets (et algo plus optimal).

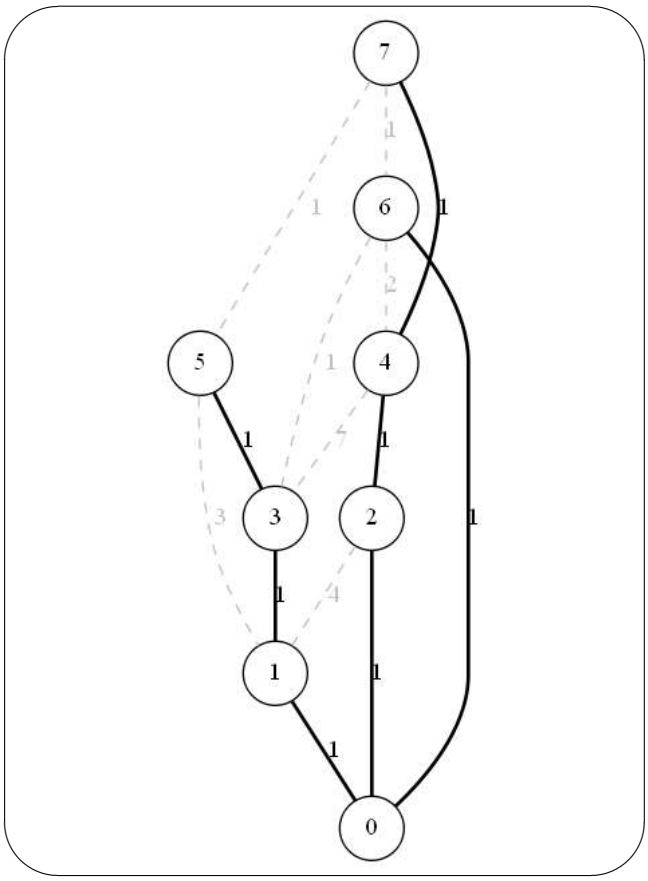
   L'heuristique d'HeuristixM est une approximation optimiste du chemin restant à parcourir, en ce sens que la valeur retournée est toujours inférieure à la véritable valeur. Par contre, l'heuristique d'HeuristixB ne varie pas en fonction du chemin restant et en l'occurrence elle renvoie une valeur supérieure au véritable chemin pour le sommet 2.
   Si l'heuristique reste raisonnable dans les valeurs renvoyées (en ce sens que ces valeurs restent inférieures à la réalité) alors les sommets du chemin le plus court seront sélectionnés avant que l'arrivée ne soit atteinte.

***Solution 3* (What is this ? – *4 points*)**

1. *Que teste la fonction* `dfs(G, x, y)` *?*

   Elle vérifie si i existe une chaîne entre x et y dans G, sans utiliser l'arête $x - -y$ (si elle existe).

2. La fonction `what` :
   (a) Le graphe :



   (b) *Quelle propriété a le graphe après application de la fonction ?*

   C'est un arbre de recouvrement de poids minimum du graphe initial.

   (c) Comment optimiser cette fonction ?

   Inutile de "vider" la liste L, on peut arrêter avant :
   — Compter le nombre d'arêtes pendant la construction de L ;
   — on arrête la deuxième boucle dès lors qu'il ne reste plus que $n - 1$ arêtes (n = G.order).