Algorithmique
Correction Partiel n° 4 (P4)

Info-spé (S4) - API – Epita

16 mai 2017 - 10*h*

Solution 1 **(ARM et PCC ... –** ***3 points*****)**

1. Les graphes orientés sur lesquels on peut exécuter l'algorithme de Bellman sont de coûts quelconques et sans cycle.
2. L'algorithme déterminant l'arm d'un graphe non orienté dont le principe est proche de celui de Dijkstra est PRIM.
3. L'ARM du graphe est celui de la figure 1.

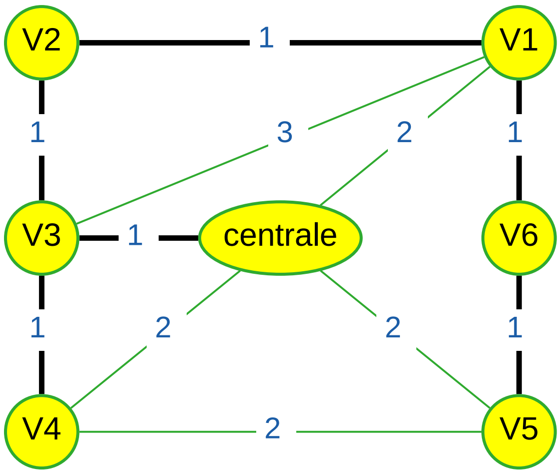


Figure 1 – ARM sur le graphe.

4. L'arbre des plus courts chemins de racine "centrale" du graphe est celui de la figure 2.

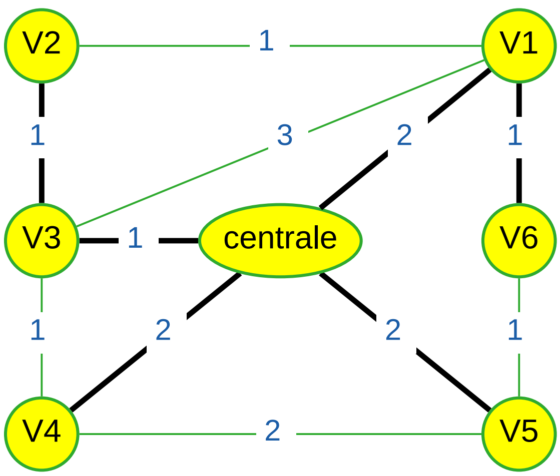


Figure 2 – Arbre de racine "centrale" sur le graphe.

Solution 2 (Graphe réduit)

Spécifications :

La fonction `condensation(`G`,` scc`)` avec G un graphe orienté et scc sa liste de composantes fortement connexes retourne le graphe réduit G_R ainsi que le vecteur des composantes : un vecteur qui pour chaque sommet de G indique à quelle composante il appartient (le numéro du sommet dans G_R).

```
def condense(G, scc):

    comp = [-1] * G.order
    k = len(scc)
    for i in range(k):
        L = scc[i]                          for s in scc[i]:
        for j in range(len(L)):                 comp[s] = i
            comp[L[j]] = i

    Gr = graph.Graph(k, directed = True)
    for s in range(G.order):
        for adj in G.adjLists[s]:
            (x, y) = (comp[s], comp[adj])
            if x != y: # (and y not in Gr.adjLists[x])
                Gr.addEdge(x, y)      # Gr.adjLists[x].append(y)

    return (Gr, comp)
```

Solution 3 (Graphes et mystère – *3 points*)

1.

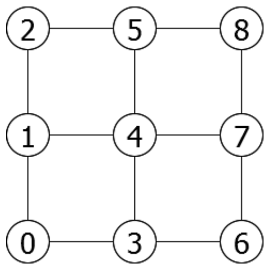
	Nombre d'appels	*Résultat retourné*
(a) `test(`G_2`)`	5	False
(b) `test(`G_3`)`	7	True

2. *Quelle information est retournée par* `test(G)` *?*

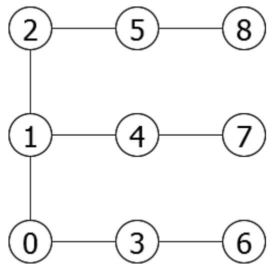
   `test(`G`)` vérifie si G est fortement connexe.

***Solution 4* (T-spanner – *10 points*)**

1. (a) t-spanners pour un stretch factor de 2



(b) t-spanners pour un stretch factor de 5



2. (a) **Spécifications :**
La fonction `Dijkstra(G, src, dst)` retourne la longueur du plus court chemin entre *src* et *dst* dans G, $+\infty$ si le chemin n'existe pas.

```
def Dijkstra(G, src, dst):

    dist = [inf] * G.order
    dist[src] = 0
    H = Heap(G.order)
    update(H, src, 0)

    while not H.isEmpty():
        (_, cur) = pop(H)
        if cur == dst:
            return dist[dst]
        for s in G.adjLists[cur]:
            if dist[s] > dist[cur] + G.costs[(cur, s)]:
                dist[s] = dist[cur] + G.costs[(cur, s)]
                update(H, s, dist[s])

    return dist[dst]    #inf
```

(b) **Spécifications :**
La fonction pathGreedy(n, L, t) retourne un t-spanner (de stretch factor $= t$) pour l'ensemble de n points (numérotés de 0 à $n-1$) avec L la liste des triplets (p, q, $|pq|$).

```
def pathGreedy(order, edges, stretch):

    edgeHeap = []
    for (x, y, w) in edges:
        heappush(edgeHeap, (w, x, y))

    G = graph.Graph(order, False, costs = True)
    while edgeHeap != []:
        (w, x, y) = heapq.heappop(edgeHeap)
        if Dijkstra(G, x, y) > stretch * w:
            G.addEdge(x, y, w)

    return G
```

bonus Lorsque le stretch factor est $n-1$ avec n le nombre de points, à quoi correspond le t-spanner ?

Le t-spanner est alors un ARPM.