université de BORDEAUX

Année 2019-2020 — DS décembre 2019

Épreuve de : Recherche Opérationnelle

Date : 3 décembre 2019 — **Durée :** 1h30

Épreuve de : O. Baudon — **Nb. pages :** 5

Aucun document autorisé. Une attention toute particulière sera portée à la présentation et à la rédaction de la copie.

Dans tous les exercices, on désignera par $V(G)$ et $E(G)$ respectivement, l'ensemble des sommets et l'ensemble des arêtes d'un graphe G.

1 Plus court chemin

1.1) Un étudiant du Master Miage de l'Université de Bordeaux, désirant faire un séjour linguistique, décide de se rendre en Suède. Après avoir fait le tour de quelques compagnies, il a recensé plusieurs connexions aériennes lui permettant d'aller de Bordeaux à Stockholm. Il les a représentées à l'aide du graphe suivant (Figure 1) :

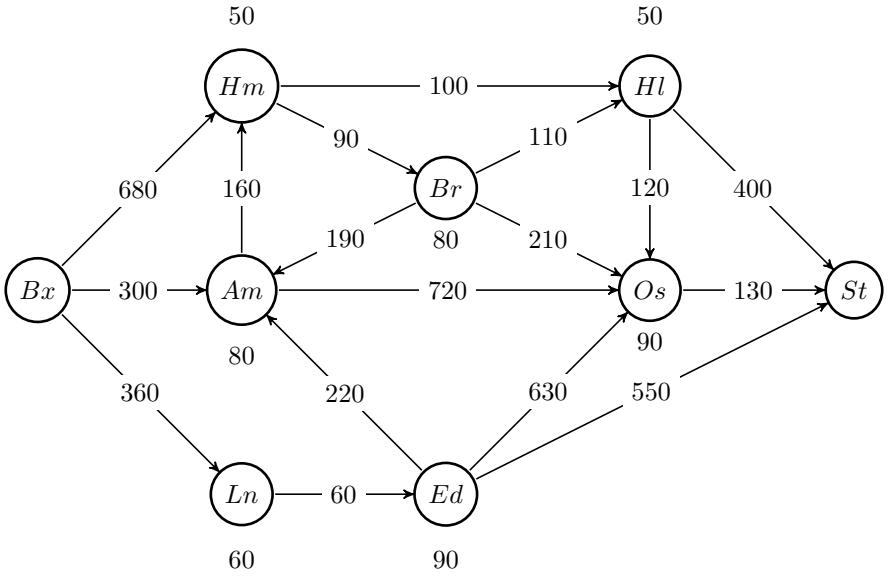

Figure 1 – Voyage

où Bx = Bordeaux, Hm = Hambourg, Am = Amsterdam, Ln = Londres, Ed = Edimbourg, Br = Berlin, Hl = Helsinki, Os = Oslo, St = Stockholm.

Cependant, les horaires des vols sont tels que l'étudiant est obligé de passer la nuit dans chacune des villes où il fait escale.

Les valeurs sur les arcs correspondent alors aux prix en Euros pour les vols, et les valeurs

à coté des sommets représentent le prix en Euros à payer pour passer la nuit dans un hôtel de la ville correspondante.

1. On désire trouver une solution optimale pour ce problème. Quelles modifications faut-il appliquer au graphe pour pouvoir utiliser l'algorithme de Dijkstra ?
2. Calculer la solution optimale. On donnera en particulier la suite des valeurs prises par la variable `pivot` et pour chacune de ces valeurs, les modifications des valeurs du tableau `d` associées.

Rappel :

```
0   Dijkstra(G,w,s)
1     pour chaque sommet v de V(G) faire
2         d[v] <- infini
3         pere[v] <- nil
4         couleur[v] <- BLANC
5     d[s] <- 0
6     F <- FILE_PRIORITE(V(G), d)
7     tant que F non vide faire
8         pivot <- EXTRAIRE_MIN(F)
9         couleur[pivot] <- GRIS
10        pour tout arc e = (pivot, v) arc sortant de pivot faire
11            si couleur(v) = BLANC et d[v] > d[pivot] + w(e)
12                alors d[v]  <- d[pivot] + w(e)
13                      pere[v]  <- pivot
14        couleur[pivot] <- NOIR
```

2 Flot Maximum

2.1) En utilisant l'algorithme de Ford-Fulkerson rappelé ci-dessous, trouver le flot maximum entre les sommets s et t du réseau de la figure 2. Le flot possède déjà une valeur initiale donnée par le premier nombre porté sur les arcs, le second étant la capacité de l'arc. Par exemple, l'arc `(s,A)` possède un flot initial de 8 et une capacité de 13. A chaque exécution, au cas ou il y a plusieurs possibilités, on utilisera l'ordre lexicographique pour effectuer les choix. On donnera pour chaque exécution de la procédure de marquage le chemin augmentant obtenu et l'augmentation correspondante.

2.2) Donner la coupe minimum correspondante et redessiner le graphe avec le flot maximum.

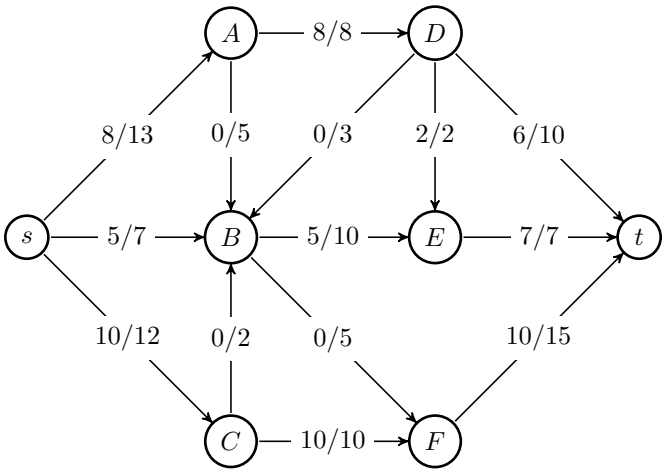

FIGURE 2 – Réseau

Rappel :

Le paramètre c désigne les capacités du graphe G, s la source et t la destination du flot f calculé. $I(e)$ et $T(e)$ désignent respectivement le sommet initial et le sommet terminal d'un arc e.

```
0   FlotMaximum(G,s,t,b,c,fi)
1     pour tout e de E(G) faire
2         f[e] <- fi[e]
3     repeter
4         Marquage(G,c,f,s,t)
5         si t appartient a Y
6             alors v <- t
7                   C+ <- {(t,s)}
8                   C- <- {}
9         tant que v different de s faire
10            e <- A[v]
11            si v = T(e)
12                alors C+ <- C+ U {e}
13                      v <- I(e)
14                sinon C- <- C- U {e}
15                      v <- T(e)
16        pour tout e de C+ faire
17            f(e) <- f(e) + delta[t]
18        pour tout e de C- faire
19            f(e) <- f(e) - delta[t]
20    tant que t appartient a Y
```

```
0   Marquage(G,c,f,s,t)
1     Y <- {s}
2     delta[s] <- infini
3     Max <- faux
4     tant que Y ne contient pas t et Max = faux faire
5         si il existe e = (u,v) avec Y contenant u mais pas v et f(e) < c(e)
6             alors Y <- Y U {v}
7                   A[v] <- e
8                   delta[v] <- min(delta[u], c(e)-f(e))
9         sinon si il existe e = (u,v) avec Y contenant v mais pas u et f(e) > 0
10            alors Y <- Y U {u}
11                  A[u] <- e
12                  delta[u] <- min(delta[v], f(e))
13        sinon Max <- vrai
```

3 Modification de l'algorithme de Kruskal

Soit G un graphe ayant des arêtes munies d'une fonction de poids w.

On rappelle qu'un graphe partiel de G est un graphe contenant tous les sommets de G et seulement une partie des arêtes de G.

3.1) Si G contient des arêtes de poids négatifs, le graphe partiel connexe de poids minimum de G est-il toujours un arbre ? Justifier votre réponse.

3.2) Modifier la version de l'algorithme de Kruskal donnée ci-dessous pour l'adapter à un graphe possédant des arêtes de poids négatifs.

3.3) Appliquer votre algorithme au graphe ci-dessous. On donnera l'ordre des arêtes rajoutés au graphe partiel connexe obtenu.

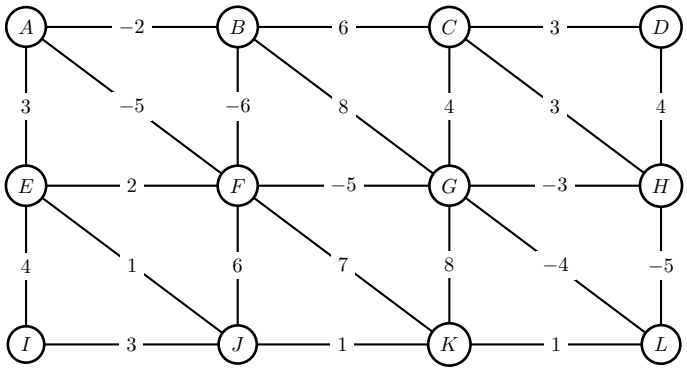

FIGURE 3 – Graphe G

```
0   Kruskal(G,w,s)
1     pour chaque sommet v de V(G) faire
2         composante(v) <- {v}
3         ncomposantes <- ncomposantes + 1
4     fin pour
5     trier E(G) dans un ordre croissant (e1, ..., em) en fonction de w(e)
6     i <- 1
7     E(H) <- {}
8     tant que ncomposantes > 1 faire
9         si ei = (u, v) et composante(u) != composante(v) alors
10            E(H) <- E(H) U {ei}
11            Unifier(composante(u), composante(v))
12            ncomposantes <- ncomposantes - 1
13        fin si
14    fin tant que
15    retourner E(H)
```