# RO202 - Initiation à la Recherche Opérationnelle

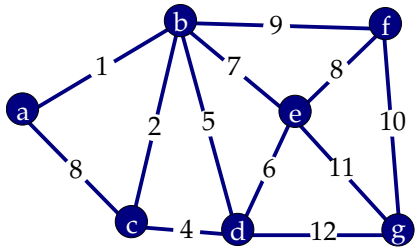
Zacharie ALES, Cristian DURAN

2021 - 2022

EXERCICES - Arbres couvrants et plus courts chemins

## Exercise 1 Arbres

Une banque veut connecter ses 7 succursales via un réseau. Les coûts de construction des liens entre les succursales sont donnés sur le graphe ci-dessous. La banque ne dispose que d'un budget limité et a décidé de construire un réseau en arbre.

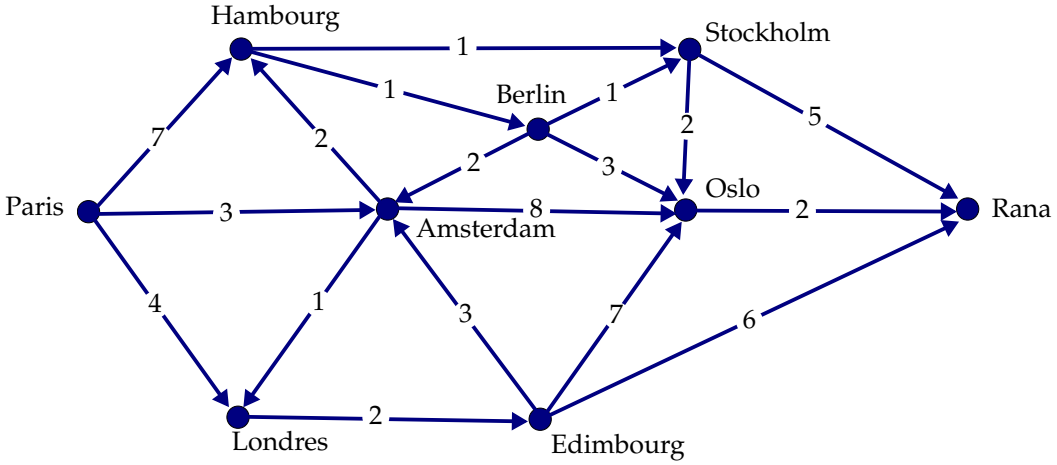



1. Appliquez manuellement l'algorithme de Kruskal pour trouver l'arbre à construire et donner son coût.
2. Quel est l'inconvénient majeur du choix d'une telle structure pour la banque ?

## Exercise 2 Chemins

Un étudiant de l'Ensta souhaite se rendre à Rana. Après avoir fait le tour de quelques compagnies, il a recensé plusieurs connexions aériennes possibles lui permettant d'aller de Paris à Rana. Il les a représentées à l'aide du graphe suivant. Les valuations portées sur les arcs correspondent au temps nécessaire (en heures) pour parcourir ces arcs, compte tenu des éventuels temps de transit pour les escales.

1. Aider cet étudiant à déterminer le chemin le plus rapide pour se rendre de Paris à Rana en appliquant l'algorithme de Dijkstra.




## Exercise 3 Implémentation de l'algorithme de Kruskal

1. **Comment exécuter et modifier un fichier java ?**
   a) Ouvrir un terminal

      `Touche windows > Écrire "terminal" > Entrée`

   b) Se déplacer dans le répertoire contenant le fichier `Kruskal.java` en utilisant dans le terminal la commande `cd` (exemple : `cd TP1-arbres-chemins/sources`).
   c) Pour compiler le programme, tapez la commande suivante dans le terminal :

```
javac Kruskal.java
'c' pour compilation ↵
```

*Remarque* : Les fichiers `Edge.java` et `Graph.java` seront également compilés car ils sont utilisés dans `Kruskal.java`

*Remarque 2* : Cette commande va créer les fichiers : `Edge.class`, `Graph.class` et `Kruskal.class`.

d) Pour exécuter la méthode `main` figurant dans le fichier `Kruskal.class`, taper la commande suivante dans le terminal :

```
java Kruskal
```

*Remarque* : A chaque modification d'un fichier, il vous faudra retaper les commandes `javac` et `java` afin de réexecuter votre programme.

*Astuce* : Pour éviter de réécrire les commandes à chaque fois, utilisez la flèche du haut du clavier dans le terminal.

e) Ouvrir l'éditeur de texte `gedit` :

```
Touche windows > Écrire "gedit" > Entrée
```

f) Ouvrir dans gedit le fichier `Kruskal.java`.

2. Compléter la méthode `public static Graph kruskal(Graph g)` du fichier `Kruskal.java` permettant d'appliquer l'algorithme de Kruskal à un graphe `g`.

   *Indication* : Vous pourrez vous servir de la méthode `Graph.createACycle(Edge)` qui renvoie vrai si l'ajout d'une arête à un graphe créé un cycle et faux sinon.
3. Lancer le programme et vérifiez que vous obtenez la même solution qu'à l'exercice précédent.
4. Utiliser cette méthode pour résoudre le problème d'arbre couvrant de poids minimal sur les deux graphes représentés en Figure 1.
5. On souhaite maintenant pouvoir obtenir des arbres couvrants de poids maximal. Ajouter un argument booléen `computeMin` à la méthode `kruskal` indiquant si l'algorithme cherche l'abre couvrant de poids minimal ou maximal. Adapter votre code afin de prendre en compte ce paramètre et calculer les arbres couvrants de poids maximal des graphes représentés en Figure 1.

   *Indication* : Vous pourrez utiliser la méthode `Collections.reverse(List l)` permettant d'inverser l'ordre des éléments d'une liste.
6. (plus difficile, à faire après l'exercice d'implémentation de Dijkstra) La complexité de la méthode `createACycle` est mauvaise. Une façon plus efficace de tester si l'ajout d'une arête créé un cycle est de stocker pour chaque sommet un numéro associé à sa composante connexe. Ainsi, il sera simple de vérifier si l'ajout d'une arête $[ab]$ créé un cycle en testant si l'identifiant des composantes connexes de $a$ et $b$ est le même.
   a) Créer la méthode `Kruskal.kruskalCC(Graph g)`.
   b) Dans cette méthode créer un tableau d'entiers nommé `component` de taille `g.n` indiquant la composante connexe de chaque sommet. Initialement, `component[i]` est égal à `i` car tous les sommets sont dans des composantes connexes différentes.
   c) Implémenter l'algorithme de Kruskal dans cette méthode `kruskalCC` en utilisant le tableau `component` pour tester la création de cycle.

## Exercise 4 Implémentation de l'algorithme de Dijkstra

1. Compléter la méthode `public static Graph dijkstra(Graph g, String origin)` du fichier `Dijkstra.java` permettant d'appliquer l'algorithme de Dijkstra pour trouver, dans le graphe `g`, les plus courts chemins entre le sommet `origin` et les autres.
2. Vérifiez que vous obtenez une solution de même coût que celle obtenu dans l'exercice précédent.
3. Utiliser cette méthode pour résoudre le problème de plus courts chemins sur les deux graphes représentés en Figure 2.

## Exercise 5 Mise en niveau et Ordonnancement

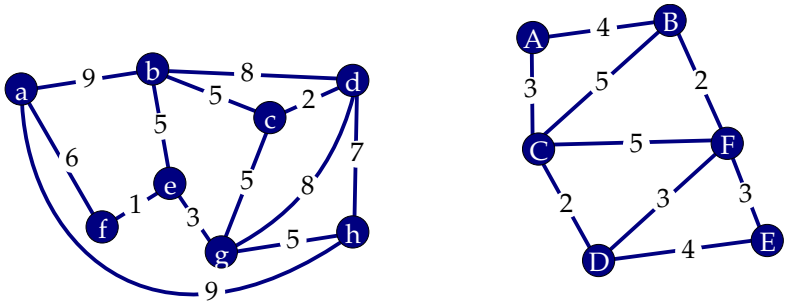



FIGURE 1 – Graphes pour lesquels on cherche un arbre couvrant de poids minimal.

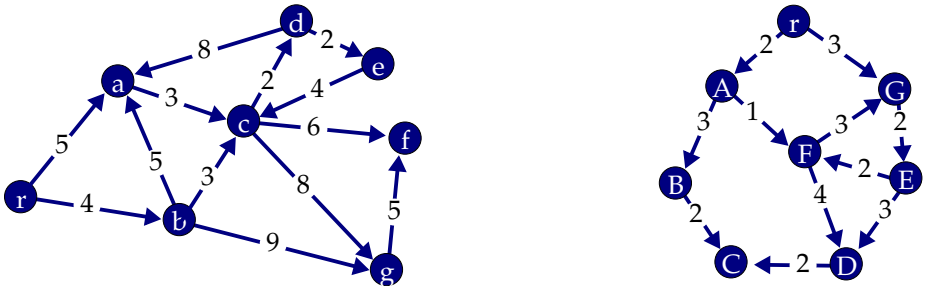



FIGURE 2 – Graphes pour lesquels on cherche le plus court chemin entre les sommets $r$ et les autres sommets du graphe.

On veut évaluer la durée minimale d'un gros projet de construction d'un barrage. On a évalué 9 tâches principales :

a. construction des routes d'accès
b. préparation des carrières et terrassements
c. construction d'une cité pour les personnels
d. commande du matériel hydraulique
e. construction de la centrale
f. construction du barrage et des digues
g. construction des galeries et conduites forcées
h. montage des machines
i. essais de fonctionnement

Pour chaque tâche on a déterminé sa durée et les tâches qui doivent impérativement être terminées avant qu'elle ne commence. Ces informations sont résumées dans le tableau suivant. On a ajouté une tâche D (sans durée) correspondant au début du projet et une tâche F (sans durée) correspondant à la fin du projet.

| Tâches | Durées (en mois) | Tâches préalables |
|---|---|---|
| D | | |
| a | 4 | D |
| b | 6 | c |
| c | 4 | D |
| d | 12 | D |
| e | 24 | a,b |
| f | 10 | b |
| g | 7 | c |
| h | 10 | d,f,g |
| i | 3 | e, h |
| F | | i |

1. On veut représenter le problème par un graphe : un sommet du graphe correspond à une tâche et il y a un arc de x vers y si la tâche x est une tâche préalable à y dans le tableau. Le graphe associé à un problème d'ordonnancement est sans circuit. Pourquoi ? On va donc pouvoir dessiner ce graphe en "niveaux" (ou en ordre) : $N_0$, $N_1$, ... $N_i$,...,$N_n$ . On place au niveau $N_i$ toutes les tâches dont les prédécesseurs ont déjà été placés (sur les niveaux $N_0$,...,$N_{i-1}$). Dessiner le graphe obtenu (pour que le graphe soit clair, on place les tâches d'un même niveau l'une sous l'autre et dans l'ordre alphabétique).
2. On éclate chaque sommet x (x=a,...,i) en un arc ($x_1$,$x_2$) et on value cet arc avec la durée de la tâche associée. Les arcs du graphe initial sont conservés et sont de valeur 0. Tracer le nouveau graphe obtenu de façon à conserver les informations précédentes.
3. On veut trouver grâce à ce graphe la durée minimale du projet. Que doit-on chercher sur le graphe ? Pensez-vous qu'on puisse utiliser l'algorithme de Dijkstra pour chercher cette durée ? Il n'est pas demandé d'appliquer un algorithme pour résoudre le problème, mais on peut trouver la réponse facilement sur ce petit graphe.
4. On appelle "tâche critique" une tâche dont la durée doit impérativement être respectée sous peine de retarder la durée totale du projet. Faire apparaître clairement sur le graphe les arcs qui correspondent à ces tâches critiques et les donner. Ce chemin est appelé "chemin critique".

## Exercise 6

On veut connaître les longueurs des chemins maximaux entre toutes les paires de sommets du graphe ci-joint. Quel algorithme allez-vous utiliser ? Appliquez cet algorithme à l'exemple. Et si on voulait les chemins min ?