RO202 - Initiation à la Recherche Opérationnelle

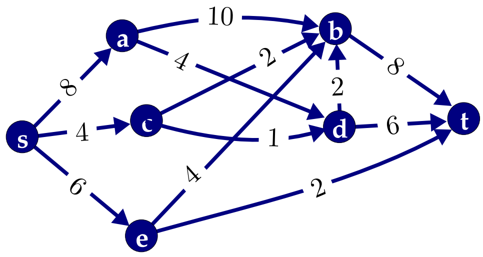
Zacharie Ales, Nidhal Gammoudi

2019 - 2020

EXERCICES 2 - Flots

Exercise 1 Ford-Fulkerson

On donne le graphe avec capacités suivant :



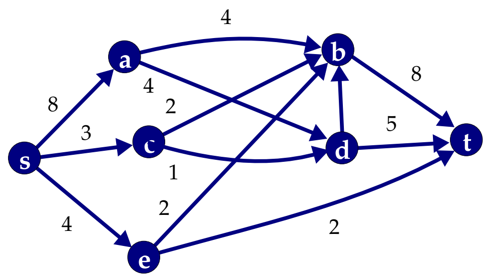
On utilisera les graphes dessinés au dos de la feuille pour traiter cet exercice.

1. Trouver un flot complet (en prenant obligatoirement les sommets selon l'ordre alphabétique).

2. Donner une borne supérieure de la valeur d'une coupe minimale (considérer les arcs incidents à s puis à t).
3. Appliquer l'algorithme de Ford-Fulkerson pour trouver un flot maximal (en marquant obligatoirement les sommets selon l'ordre alphabétique) et en déduire la coupe minimale.
4. On a la possibilité d'augmenter la capacité d'un arc. Lequel choisir pour pouvoir augmenter le flot au maximum ?

Answer of exercise 1

1. 8 en haut, 1 au milieu, 2 en bas



2. borne supérieure : $16 = \min(18, 16)$
3. (et) et (bt) permettent d'augmenter le flot de 2.

Exercise 2 Implémentation de Ford-Fulkerson

1. Inclure à votre projet java RO202 le fichier `FordFulkerson.java`.
2. Compléter la méthode `public static Graph fordFulkerson()` permettant d'appliquer l'algorithme de Ford-Fulkerson à un graphe g entre une source s et un puits t.
   Remarque : Pour simplifier, vous commencerez avec un flot initial nul plutôt que complet.
3. Vérifier que votre méthode retourne le même résultat que celui obtenu à l'exercice précédent.
4. Utiliser cette méthode pour résoudre le problème de flot maximal des graphes représentés en Figure 1.

Answer of exercise 2

```java
/* Seek improving chain while an improving chain has been found at last step
 * (i.e., while t has been reached by a mark) */
do {

        System.out.println("---␣New␣iteration");
        boolean newNodeMarked;

        /* Set all the nodes as unmarked */
        for(int i = 0; i < g.n; i++)
                mark[i] = Integer.MAX_VALUE;

        mark[s] = 0;

        /* While a new node has been marked at the previous step and t
        is not marked, try to mark another node */
        do{

                /* Browse all the arcs to find a new mark */
                newNodeMarked = false;
                int idArc = 0;

                /* While no new node is marked and all the arcs have not been browsed */
                while(!newNodeMarked && idArc < arcs.size()) {

                        Edge arc = arcs.get(idArc);
                        int i = arc.id1;
                        int j = arc.id2;

                        /* If j must be marked +i */
                        if(mark[i] != Integer.MAX_VALUE
                                        && mark[j]== Integer.MAX_VALUE
                                        && flow.adjacency[i][j] < g.adjacency[i][j]) {
                                mark[j] = i;
                                newNodeMarked = true;
                        }

                        /* If i must be marked +j */
                        else if(mark[i] == Integer.MAX_VALUE
                                        && mark[j] != Integer.MAX_VALUE
                                        && flow.adjacency[i][j] > 0) {
                                mark[i] = -j;
                                newNodeMarked = true;
                        }
                        else
                                idArc++;
                }

        }while(newNodeMarked && mark[t] == Integer.MAX_VALUE);

        /* If node t has been marked (i.e., if an improving chain has been found) */
        if(mark[t] != Integer.MAX_VALUE) {

                List<Integer> chain = new ArrayList<>();
                chain.add(t);

                int j;
                int i = t;

                double chainImprovement = Integer.MAX_VALUE;

                /* While the path is not over */
                do{

                        /* Get the next arc (ij) */
                        j = i;
                        i = (int)Math.abs(mark[j]);

                        /* Add i in the chain */
                        chain.add(0, i);

                        /* See how much the flow can be improved along this arc */
                        double potentialImprovement = g.adjacency[i][j] - flow.adjacency[i][j];
```

```
                        if(mark[j] < 0)
                                potentialImprovement = flow.adjacency[j][i];

                        /* Update the chain improvement if necessary */
                        if(potentialImprovement < chainImprovement)
                                chainImprovement = potentialImprovement;

                }while(i != s);

                System.out.println("Improving␣chain:␣"+ chain);
                System.out.println("Improvement␣along␣that␣chain:␣" + chainImprovement);

                /* Update the flow along the chain */
                for(int id = 0; id <= chain.size() - 2; id++) {

                        i = chain.get(id);
                        j = chain.get(id+1);

                        if(mark[j] >= 0)
                                flow.adjacency[i][j] += chainImprovement;
                        else
                                flow.adjacency[i][j] -= chainImprovement;

                }
        }


}while(mark[t] != Integer.MAX_VALUE);
```

Solution du premier graphe en Figure 1
Flot maximal : 23
s - 1 (12.0)
s - 2 (11.0)
1 - 3 (12.0)
2 - 4 (11.0)
3 - t (19.0)
4 - 3 (7.0)
4 - t (4.0)
Solution du second graphe en Figure 1
Flot maximal : 28
s - A (10.0)
s - C (8.0)
s - E (10.0)
A - B (9.0)
A - D (1.0)
B - t (9.0)
C - D (8.0)
D - t (9.0)
E - F (10.0)
F - t (10.0)

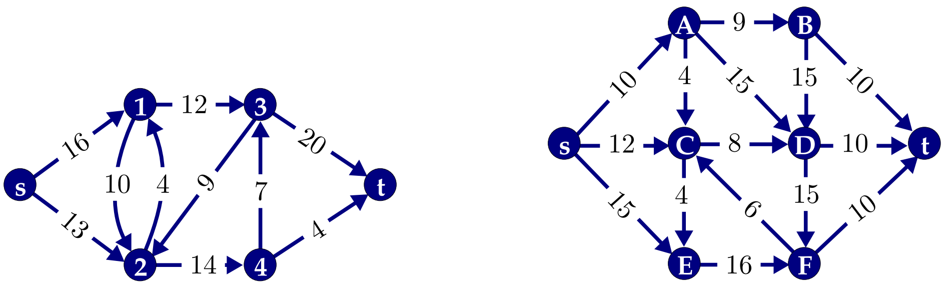


FIGURE 1 – Graphes pour lesquels on cherche un flot maximal.

Exercise 3 Couplage

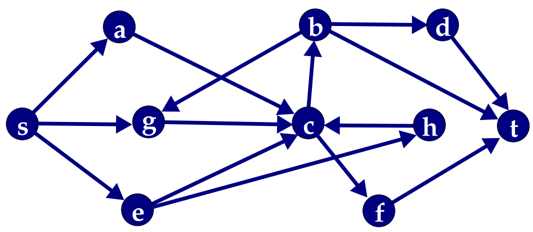
Un coffre-fort est composé de r serrures qu'il faut déverrouiller simultanément pour l'ouvrir. On dispose de l clefs de sorte que toute clef ouvre au moins une serrure, une serrure peut être ouverte par au moins une clef et $l \geq r$. Sachant quelles serrures chaque clef ouvre, on se demande si l'on peut ouvrir le coffre. On représente la situation à l'aide d'un graphe biparti $G = (V_1, V_2, A)$ où les sommets de V_1 sont les clefs, les sommets de V_2 sont les serrures et il y a un arc dans A d'un sommet clef de V_1 vers et un sommet serrure de V_2 si l'une permet d'ouvrir l'autre. Proposez une méthode générale qui permet d'obtenir une solution s'il en existe une, et montre comment déverrouiller un maximum de serrures dans le cas contraire.

Answer of exercise 3

On ajoute
— s relié à tous les sommets de V_1
— t relié à tous les sommets de V_2
— l'arc fictif ts de capacité $+\infty$
Les arcs ont des capacités de 1.

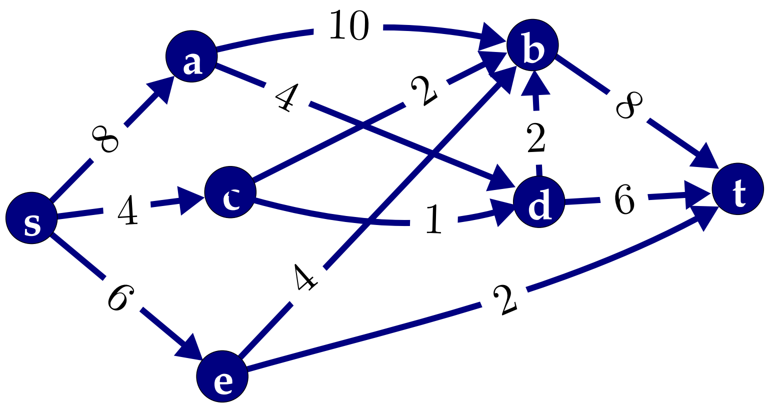
Exercise 4 Chemins disjoints

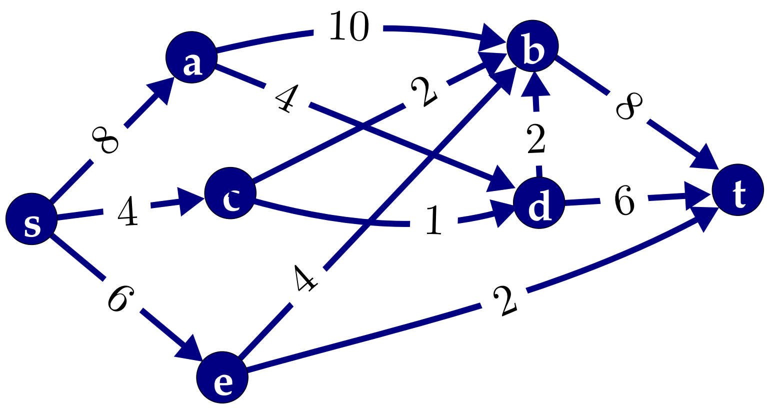
1. Donner une méthode générale permettant de trouver un nombre maximum de chemins arcs-disjoints entre un sommet source s et une destination t dans un réseau.
2. Appliquez la méthode pour le réseau ci-dessous (d'abord à la main, puis en utilisant votre implémentation de l'algorithme de Ford-Fulkerson).
3. Comment faire si on veut en plus que les chemins soient sommets-disjoints ?
4. Appliquez cette seconde méthode pour le réseau ci-dessous (d'abord à la main, puis en utilisant votre implémentation de l'algorithme de Ford-Fulkerson).

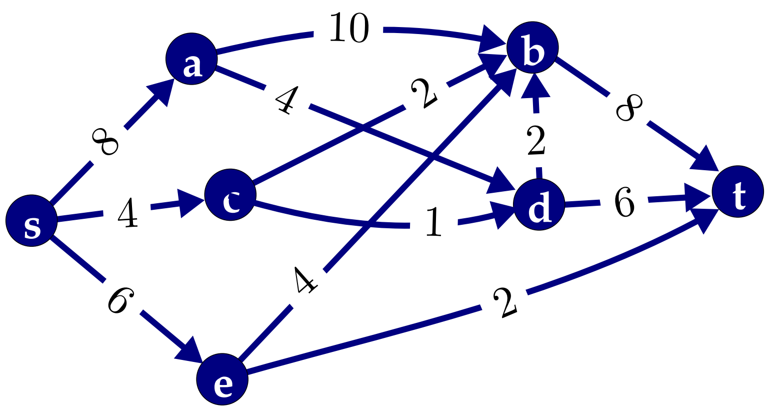


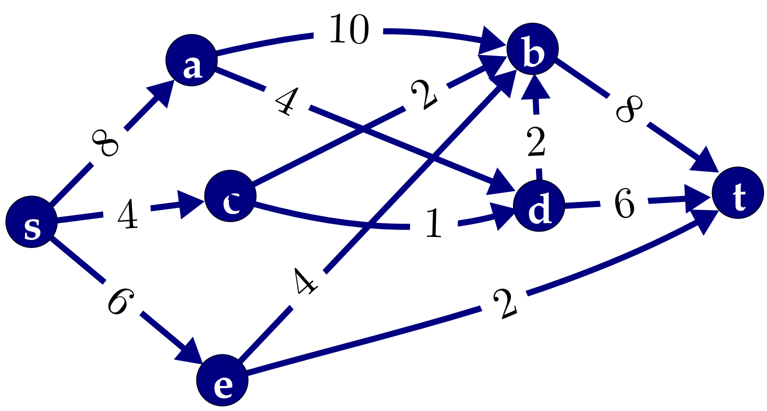
Answer of exercise 4

1. Trouver un flot maximal quand tous les arcs ont une capacité de [1].
2. 2 chemins maximum
3. Remplacer chaque sommet a par deux sommets a_1 et a_2 de telle sorte que :
   — Tous les arcs entrant en a entrent en a_1 ;
   — Tous les arcs sortant en a sortnt en a_2 ;
   — l'arc $(a_1 a_2)$ a une capacité de [1].


s
a
b
c
d
e
t
8
4
6
10
4
2
1
4
2
2
6
8
2


s
a
b
c
d
e
t
8
4
6
10
4
2
1
4
2
2
6
8
2


s
a
b
c
d
e
t
8
4
6
10
4
2
1
4
2
2
6
8
2


s
a
b
c
d
e
t
8
4
6
10
4
2
1
4
2
2
6
8
2