

# TP: Focus sur l'algorithme de Ford et Fulkerson

Léopold Maillard  
Lucie Clair

October 2020

## 1 Analyse descendante

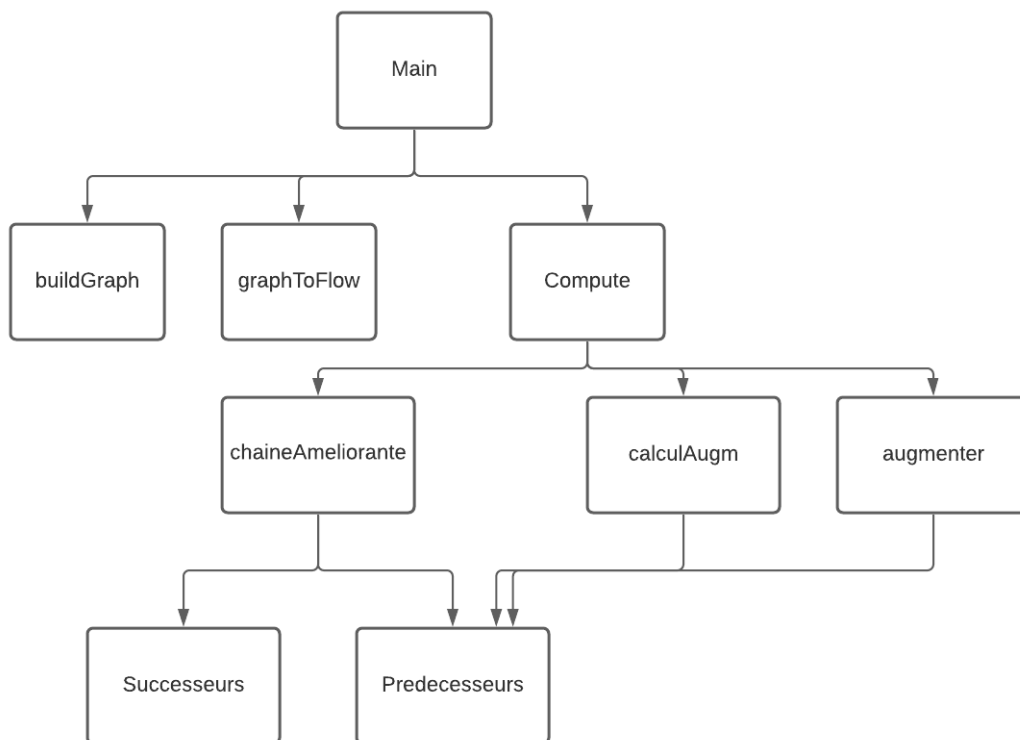


Figure 1: Analyse descendante

Nous avons fait le choix de ne faire apparaître dans l'analyse descendante que les fonctions que nous avons nous même implémentées. Cependant, il faut garder à l'esprit que ces fonctions utilisent les méthodes de la librairie GraphStream.

## **2 Conception détaillée**

### **2.1 Algorithme principal**

Dans l'algorithme principal, quatre tâches se distinguent: la création du graphe, son initialisation en tant que graphe de flots, le calcul de son flot maximal et l'affichage.

Nous créons donc tout d'abord une instance du type SingleGraph. Puis, dans la fonction buildGraph nous créons chaque noeud et arête en spécifiant les capacités. Nous n'avons pas implémenter la fonction d'initialisation, on utilise la fonction de GraphStream qui nous permet d'obtenir un graphe de flots à partir du graphe créé dans l'étape précédente. C'est dans la fonction compute que le coeur de l'algorithme Ford-Fulkerson se trouve. Elle va permettre de calculer le flow max.

Le programme principal utilise également une fonction d'affichage qui nous permet de visualiser le graphe.

### **2.2 La fonction Compute()**

Comme indiqué précédemment c'est la fonction qui nous permet d'obtenir le flot max du graphe. Dans ce but, elle va chercher des chaines améliorantes jusqu'à ne plus en trouver tout en mettant à jour les flots. Une chaine améliorante correspond à une chaîne qui améliore le flot max donc une chaîne non saturée.

### **2.3 Recherche de chaine améliorante**

La fonction renvoie un boolean, le but étant d'indiquer si oui ou non une chaîne a pu être trouvée.

Elle repose principalement sur l'algorithme de parcours de graphe en largeur (BFS). On part de la source, on visite tous ses voisins puis pour chaque voisins on visite tous les voisins, ainsi de suite jusqu'à atteindre le puit. Cependant, il ne sera pas toujours possible d'atteindre le puit, nous le visiterons que si une chaîne permet de l'atteindre. C'est d'ailleurs comme ça que nous saurons si une chaîne a été trouvée. Pour pouvoir parcourir le graphe de cette manière nous utilisons tout simplement une file dans laquelle on stock tous les noeuds à visiter.

Les noeuds à visiter sont les noeuds que l'on peut atteindre par un arc dont le flot est inférieur à sa capacité. En d'autre termes, ce sont les voisins que l'on peut atteindre par un arc non saturé.

Jusque là rien ne semble nous donner la chaîne en tant que telle. Cependant, pour l'obtenir, il suffit de stocker les parents de chaque noeud visité. Nous pourrons ensuite

reconstruire la chaine en partant du puit (en accédant à son parent, puis le parent de son parent.. jusqu'à la source). Nous avons choisi d'utiliser une hashmap pour stocker les parents sous la forme suivante :`parentj`.

A tout cela s'ajoute la gestion des arcs arrières. Nous autorisons la chaine à emprunter un arc dans le sens inverse dans le cas où son flot n'est pas nul. Concrètement, la gestion des arcs arrières n'est pas bien différente de celle des arcs avant: nous observons tous les predecesseurs qui vérifient nos conditions tout en faisant bien attention à faire référence au flot de l'arc avant. Puis comme pour un arc avant, on stock le parent du noeud (qui est, dans la réalité du graphe, son successeur). Le code suivant présente la répétition qui permet d'évoluer dans le graphe tout en stockant ce qui nous interesse pour reconstruire la chaine.

```
Node i=queue.remove();
```

```

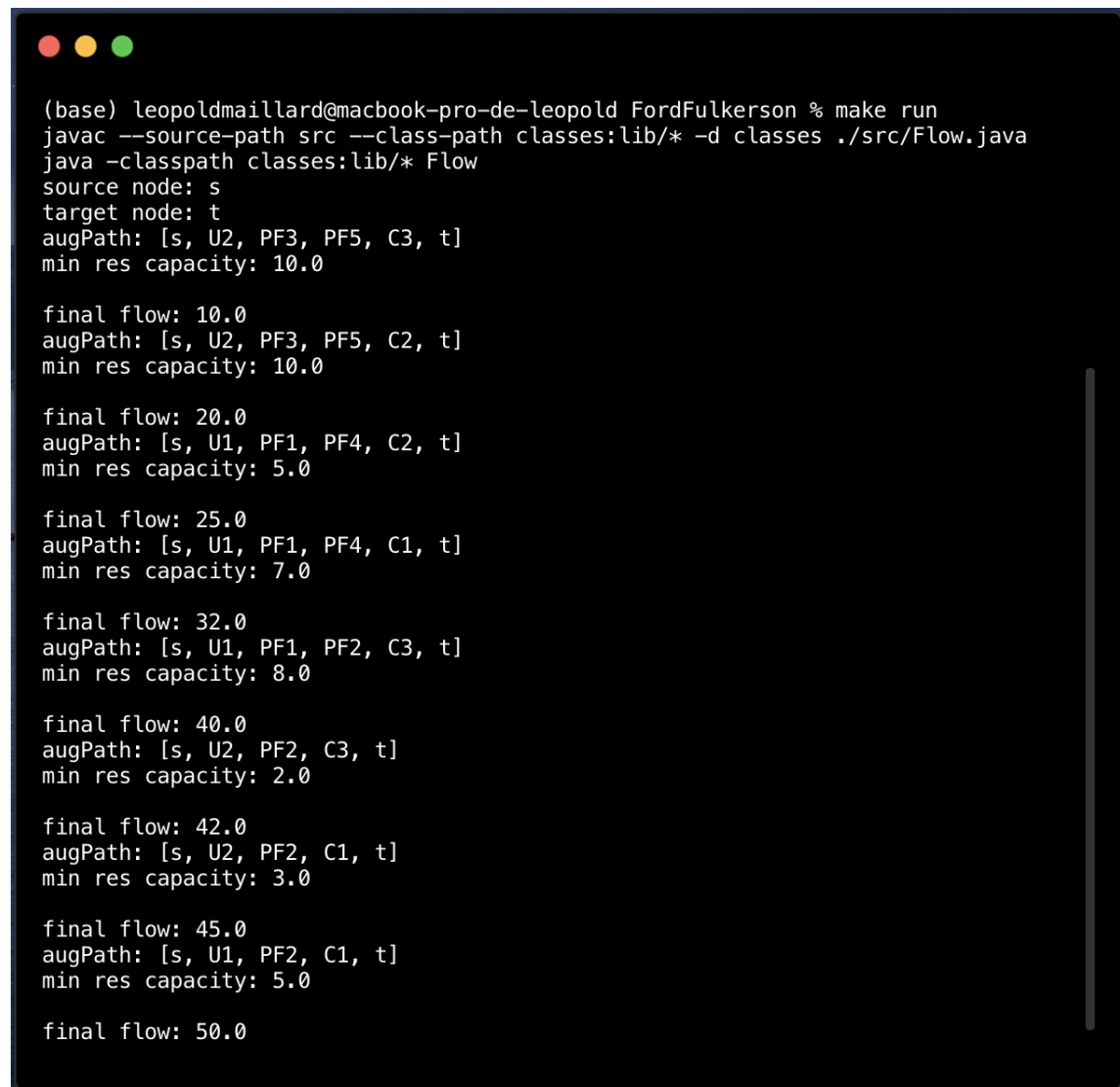
successeurs(i).forEach(j->{
    if (this.getFlow(i, j)<this.getCapacity(i, j)){
        queue.add(j);
        parent.put(j, i);
        j.setAttribute("marked", "true");
    }
});

predecesseurs(i).forEach(j->{
    if (this.getFlow(j, i)>0 && !j.getAttribute("marked").equals("true")){
        queue.add(j);
        parent.put(j, i);
        j.setAttribute("marked", "true");
    }
});

```

## 3 Affichage du résultat pour G

### 3.1 Affichage textuel



```
(base) leopoldmaillard@macbook-pro-de-leopold FordFulkerson % make run
javac --source-path src --class-path classes:lib/* -d classes ./src/Flow.java
java -classpath classes:lib/* Flow
source node: s
target node: t
augPath: [s, U2, PF3, PF5, C3, t]
min res capacity: 10.0

final flow: 10.0
augPath: [s, U2, PF3, PF5, C2, t]
min res capacity: 10.0

final flow: 20.0
augPath: [s, U1, PF1, PF4, C2, t]
min res capacity: 5.0

final flow: 25.0
augPath: [s, U1, PF1, PF4, C1, t]
min res capacity: 7.0

final flow: 32.0
augPath: [s, U1, PF1, PF2, C3, t]
min res capacity: 8.0

final flow: 40.0
augPath: [s, U2, PF2, C3, t]
min res capacity: 2.0

final flow: 42.0
augPath: [s, U2, PF2, C1, t]
min res capacity: 3.0

final flow: 45.0
augPath: [s, U1, PF2, C1, t]
min res capacity: 5.0

final flow: 50.0
```

Figure 2: Résultat du programme appliqué au graphe G

### 3.2 Affichage graphique

Les outils de visualisation de la librairie GraphStream permettent d'obtenir un affichage graphique de notre graphe, et en indiquant la valeur des flow traversant chaque arc après exécution de l'algorithme de Ford Fulkerson. On obtient ainsi l'affichage suivant :

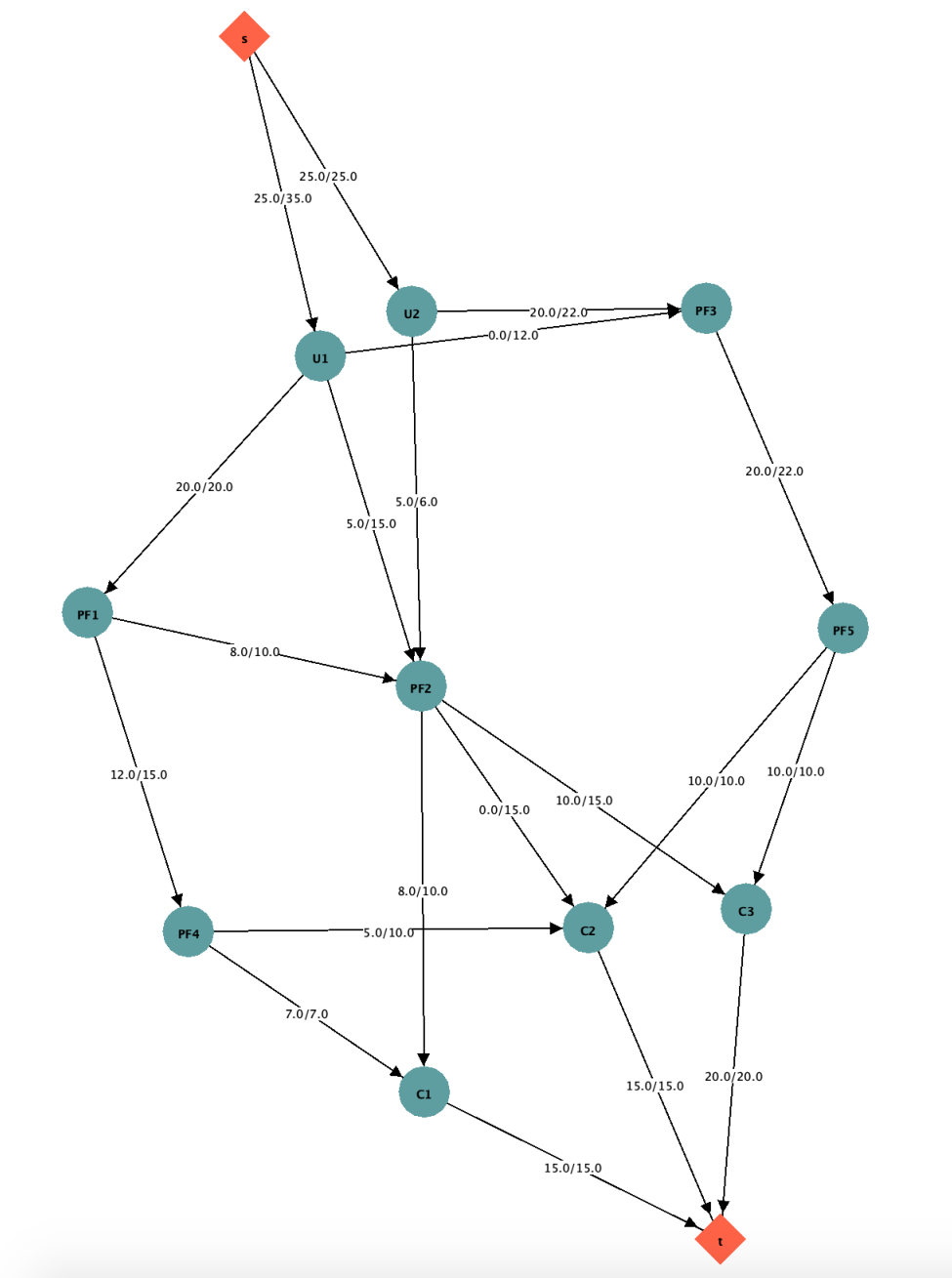


Figure 3: Visualisation du graphe G après maximisation des flux

## 4 Résultat du graphe de la slide 21

Appliquer le programme au graphe du cours à la slide 21 permet de visualiser clairement la gestion des arcs arrières.

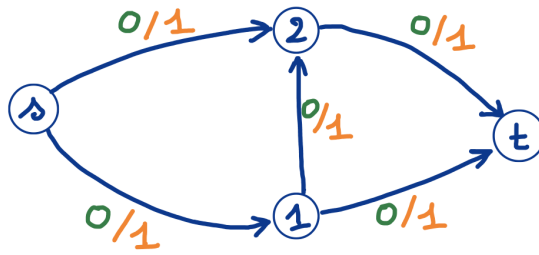


Figure 4: Graphe de la slide 21

L'exécution du programme appliqué à ce graphe renvoie le résultat suivant, on remarque l'utilisation d'un arc arrière lors de la recherche d'une seconde chaîne améliorante :

```

(base) leopoldmaillard@macbook-pro-de-leopold FordFulkerson % make run
javac --source-path src --class-path classes:lib/* -d classes ./src/Flow.java
java -classpath classes:lib/* Flow
source node: s
target node: t
augPath: [s, 1, 2, t]
min res capacity: 1.0

final flow: 1.0
augPath: [s, 2, 1, t]
min res capacity: 1.0

final flow: 2.0
(base) leopoldmaillard@macbook-pro-de-leopold FordFulkerson %
  
```

Figure 5: Résultat du programme pour ce graphe

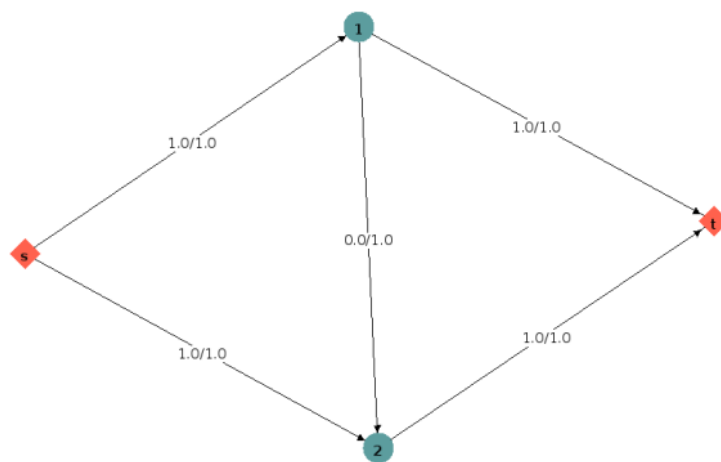


Figure 6: Visualisation du résultat