Projet Recherche Opérationnelle
Implémentation de l'algorithme de Ford Fulkerson

BAFFERT Ambre, CASINO Lisa

10 Novembre 2020

1 Introduction

Au cours de l'enseignement Recherche Opérationnelle (RO), nous avons implémenté l'algorithme de Ford et Fulkerson. Nous avons utilisé l'API GraphStream. Cet algorithme d'optimisation consiste principalement à trouver le flot maximum dans un graphe G. Nous expliquerons nos choix d'implémentations à l'aide d'une analyse descendante et de pseudo code, puis nous montrerons le résultat de notre algorithme. Nous n'avons pas géré les arc arrières dans cette version du projet.

2 Analyse descendante

A l'aide des cours magistraux de Mme Del Monde et du pseudo code donnée dans les PDF concernant l'algorithme général et la recherche d'une chaîne améliorante, nous avons pu concevoir l'analyse descendante ci-dessous.

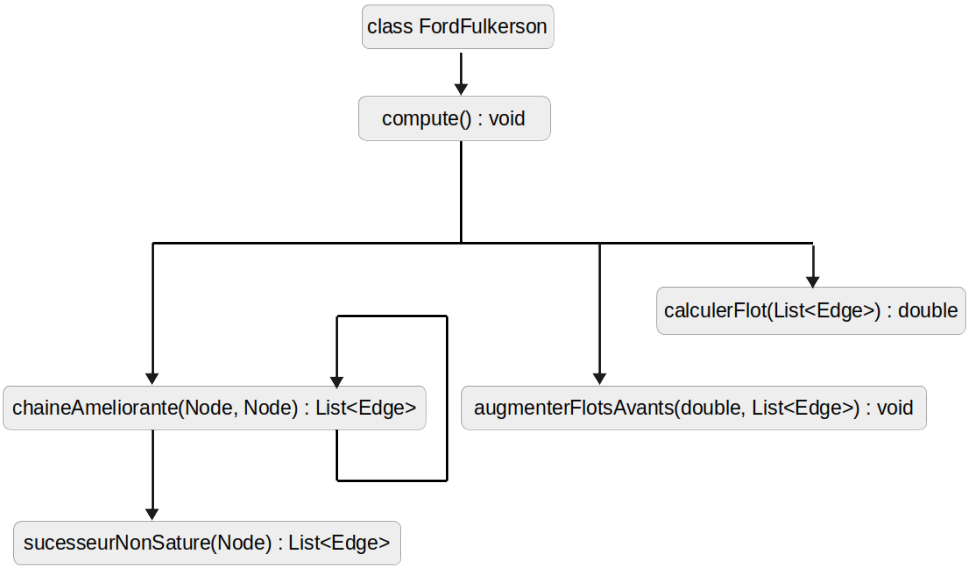

Figure 1: Analyse descendante

Celle-ci est assez simple du fait de l'utilisation de la librairie GraphStream qui contient un grand nombre de fonctions liées à l'utilisation d'un graphe.

Le fichier Flow.java récupère le fichier "graph2.dgs" qui contient les informations sur le graphe et qui se trouve dans un dossier nommé "ressource". A l'aide de la librairie graphstream et plus précisément de FlowAlgorithm, nous attribuons sur chaque arête la capacité maximale du flot et nous définissons la source et le puit. C'est ensuite que la fonction compute() est appelée. Celle-ci va chercher des chaînes améliorantes, les garder en mémoire et calculer un nouveau flot, jusqu'à ce qu'elle ne trouve plus aucune chaîne améliorante.

3 Conception détaillée

3.1 compute

procédure compute ()
Déclaration source,puit : **Noeud**;
maximumFlot : **Entier**;
nbAretes : **Entier**;
chaine : **Tableau**[nbAretes] **de arete** ;
G : Graph ;
Noeud1 : **Noeud**;
Noeud2 : **Noeud**;
delta : **Reel**;

```
debut
    nbAretes ← getEdgeCount(G)
    source ← getSource(G)
    puit ← getSink(G)
    maximumFlot ← 0
    pour i ←0 à nbAretes faire
        Noeud1 ← obtenirSourceNoeud(i)
        Noeud2 ← obtenirCibleNoeud(i)
        fixerFlow(Noeud1,Noeud2,0.0)
    finpour
    chargementCapacitéAttribut()
    affichageCapacitéArc();
    repeter
        chaine ← chaineAmeliorante(source,puit)
        si estNonVide(chaine) alors
            delta ← CalculerFlot(chaine)
            augmenterFlotAvant(delta,chaine)
            maximumFlot ← maximumFlot + delta
        finsi
    jusqu'a ce que estNonVide(chaine)
fin
```

3.2 chaineAmeliorante

```
fonction chaineAmeliorante (t,noeud : Noeud) : Tableau[nbAretes] de arete
    Déclaration  ret :Tableau[nbAretes] de arete
debut
    si noeud=t alors
        retourner Tableau[0] de arete
    sinon
        pour chaque arete de successeurNonSature(noeud)
            ret ← chaineAmeliorante(opposerDe(arete,noeud),t)
            si estNonNul(ret) alors
                ajouter(ret,arete)
                retourner ret
            finsi
        finpour
    finsi
    retourner null
fin
```

3.3 calculerFlot

```
fonction calculerFlot (chaine : Tableau[nbAretes] de arete) : Reel
    Déclaration  delta :
                 Constante MAXVALUE =
                 temp : Reel

debut
    i ← 0
    pour chaque arete de chaine
        temp ← capacite(obtenirSourceNoeud(i),obtenirCibleNoeud(i))−flow(obtenirCibleNoeud(i),obtenirSourceNoeud(i))

        si temp<delta alors
            delta ← temp
        finsi
    finpour
    retourner delta
fin
```

3.4 augmenterFlotsAvants

procédure augmenterFlotAvants (delta : **Reel**,chaine : **Tableau**[nbAretes] **de arete**)
Déclaration Noeud1 : **Noeud**;
Noeud2 : **Noeud**;
newFlow : **Reel**;
debut
pour chaque arete **de** chaine
Noeud1 ← obtenirSourceNoeud(arete)
Noeud2 ← obtenirCibleNoeud(arete)
newFlow ← delta + obtenirFlow(Noeud1,Noeud2)
fixerFlow(Noeud1,Noeud2,newFlow)
finpour
fin

4 Affichage du résultat

4.1 Graphe

Nous utilisons la commande "make" puis "make run" sur le terminal et avec l'aide du Makefile, le code s'exécute et voici le graphe qui est affiché dans une nouvelle fenêtre.

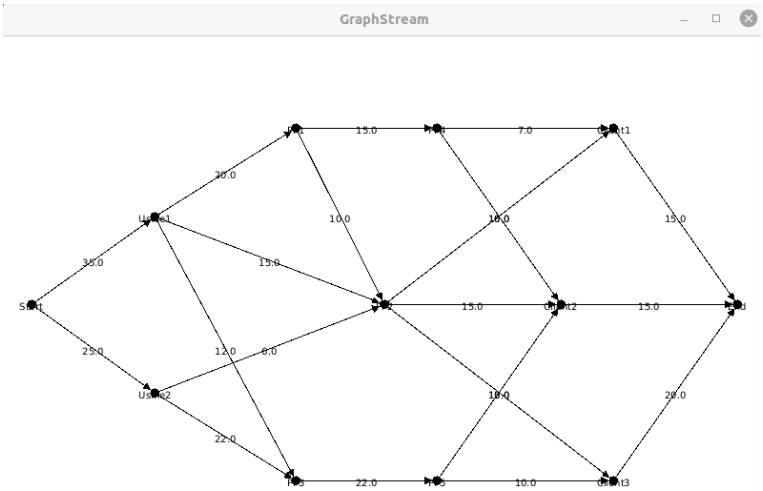

Figure 2: Affichage du Graphe

4.2 Chaîne améliorante et flot maximal

Après exécution du code et en même temps que le graphe s'ouvre dans une nouvelle fenêtre, les différentes chaînes améliorantes ainsi que le flot maximal que l'algorithme calcule s'affichent sur le terminal.

```
java -classpath classes:lib/* FordFulkerson.Flow
Chaine améliorante : [15d[Client1->T], 7[PF4->Client1], 15a[PF1->PF4], 20[Usine1->PF1], 35[S->Usine1]]
Chaine améliorante : [15e[Client2->T], 10a[PF4->Client2], 15a[PF1->PF4], 20[Usine1->PF1], 35[S->Usine1]]
Chaine améliorante : [15d[Client1->T], 10b[PF2->Client1], 10[PF1->PF2], 20[Usine1->PF1], 35[S->Usine1]]
Chaine améliorante : [15d[Client1->T], 10b[PF2->Client1], 15[Usine1->PF2], 35[S->Usine1]]
Chaine améliorante : [15e[Client2->T], 15b[PF2->Client2], 15[Usine1->PF2], 35[S->Usine1]]
Chaine améliorante : [20a[Client3->T], 15c[PF2->Client3], 15[Usine1->PF2], 35[S->Usine1]]
Chaine améliorante : [20a[Client3->T], 15c[PF2->Client3], 6[Usine2->PF2], 25[S->Usine2]]
Chaine améliorante : [20a[Client3->T], 10d[PF5->Client3], 22[PF3->PF5], 2[Usine2->PF3], 25[S->Usine2]]
Le flot maximal de ce graphe est : 50.0
```

Figure 3: Résultat chaîne améliorante et flot maximal