Architecture des systèmes d'information

Recherche opérationnelle

Algorithme de Ford-Fulkerson

Abécassis Zoé - Prieur Maxime

Géraldine Del Mondo

25 Octobre 2020

I. Introduction

Les objectifs de ce travail sont multiples. Nous cherchons à illustrer la dernière démarche de résolution de l'un des problèmes traités en théorie des graphe, l'implémentation d'un algorithme. Se familiariser avec l'utilisation d'une API existante (GraphStream) et enfin, améliorer notre compréhension de l'algorithme de recherche du flot maximal de Ford et Fulkerson.

A. Démarche suivie

Dans un premier temps nous avons découvert Graphstream, un outil permettant la création et la visualisation de graphe en Java.

Ensuite, nous avons réalisé l'analyse descendante puis la conception détaillée de l'algorithme de Ford-Fulkerson, ce qui inclut donc l'algorithme de recherche de chaîne améliorante. Enfin, nous avons implémenté les différents algorithmes en java à l'aide de Graphstream et avons interprété les résultats.

II. Analyses descendantes

A. Algorithme de recherche de chaîne améliorante

On montre ci-dessous l'analyse descendante de notre implémentation d'un algorithme de recherche de chaîne améliorante.

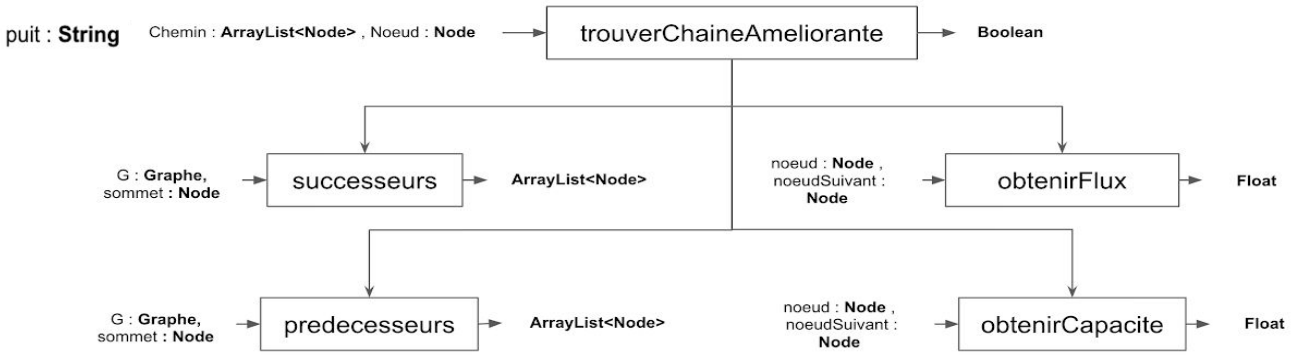

Image 1 : Analyse descendante de l'algorithme de chaîne améliorante

B. Algorithme de Ford-Fulkerson

Voici ici, l'analyse descendante concernant notre implémentation de l'algorithme de Ford-Fulkerson.

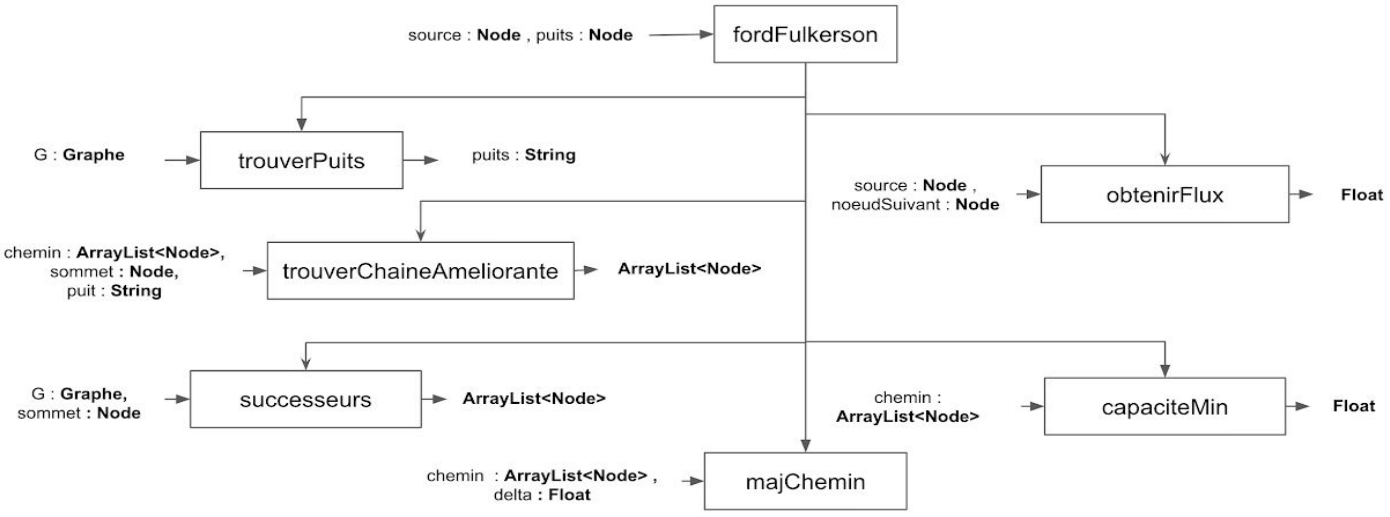

Image 2 : Analyse descendante de l'algorithme de Ford-Fulkerson

III. Analyse détaillée

A. Algorithme de recherche de chaîne améliorante

Affiché ci-dessous, le pseudo code de notre implémentation de l'algorithme de recherche de chaîne améliorante.

```
procédure trouverChaîneAmeliorante
Entrées : chemin : Liste, noeud: Noeud, puit : chaine de caractère
Sorties : Booléen
début
    ajouter(chemin,noeud) si noeud = obtenirNoeud(puit alors
        retourner Vrai
    fin
    listeNoeudsSuc ← successeurs(G,noeud)
    listeNoeudsPred ← predecesseurs(G,noeud)
    pour chaque unNoeud de listeNoeudsSuc faire
        si non(contient(chemin,unNoeud)) et (obtenirCapacite(noeud,
          unNoeud)-obtenirFlux(noeud,unNoeud))> 0 alors
            si trouverChaineAmeliorante(chemin,unNoeud,puit)=Vrai
              alors
                retourner Vrai
            fin
        fin
    fin
    pour chaque unNoeud de listeNoeudsPred faire
        si non(contient(chemin,unNoeud)) et
          obtenirFlux(noeud,unNoeud)> 0 alors
            si trouverChaineAmeliorante(chemin,unNoeud,puit)=Vrai
              alors
                retourner Vrai
            fin
        fin
    fin
    retirerEnIemePosition(chemin,longueur(chemin)-1)
    retourner Faux
fin
Algorithme 1 : Algorithme de recherche de chaîne améliorante
```

Image 3 : analyse détaillée algorithme chaîne améliorante

trouverChaieAmeliorante est une fonction récursive permettant d'ajouter au fur et à mesure un noeud à la chaine si il existe un noeud voisin avec le dernier noeud ajouté et si il est possible d'augmenter le flux (successeurs) ou bien le diminuer (prédécesseur).

B. Algorithme de Ford-Fulkerson

Affiché ci-dessous, le pseudo code de notre implémentation de l'algorithme de Ford-Fulkerson.

```
fonction FordFulkerson
Entrées : G : Réseau
Sorties : Flot
début
    S ← obtenirSGraphe(G)
    puit ← trouverIdPuit(G)
    maximumFlux leftarrow 0
    successeurOrigine ← successeurs(G,S)
    chemin ← listeVide()
    tant que trouverChaineAmeliorante(chemin,S,puit) faire
        δ ← capaciteMin(chemin)
        majChemin(chemin,δ)
        vider(chemin)
    fin
    pour chaque successeur de successeurOrigine faire
        maximumFlux ← maximumFlux + obtenirFlux(S,successeur)
    fin
    retourner maximumFlux
fin
```

Algorithme 2 : Algorithme de Ford Fulkerson

Image 4 : Analyse détaillée Algorithme de Ford-Fulkerson

Tant qu'il existe une chaîne améliorante, on augmente le flux de chaque arc de cette chaîne par delta (la capacité minimale d'amélioration le long du chemin). Une fois terminé on obtient le flux max en sommant le flux de tous les arcs voisins au noeud source.

C. Prise en compte des arcs arrières

Nous avons eu le temps de modifier notre algorithme pour prendre en compte les arcs arrière comme nous l'avons indiqué dans le pseudocode de l'algorithme. Voici ci dessous le résultat textuel et graphique de l'algorithme sur le graphe de la slide 21 du cours.

```
Algorithme FF : Source S , target t
Chaine améliorante : [S Noeud 1 Noeud 2 Puit ]
Min res capacité :1.0

Chaine améliorante : [S Noeud 2 Noeud 1 Puit ]
Min res capacité :1.0

Flot Maximal determiné : 2.0
S->Noeud 1 Capacité :1.0 / Flot :1.0
S->Noeud 2 Capacité :1.0 / Flot :1.0
Noeud 1->Noeud 2 Capacité :1.0 / Flot :0.0
Noeud 1->Puit Capacité :1.0 / Flot :1.0
Noeud 2->Puit Capacité :1.0 / Flot :1.0
```

Image 5 : résultat textuel exécution algorithme sur le graphe du cours

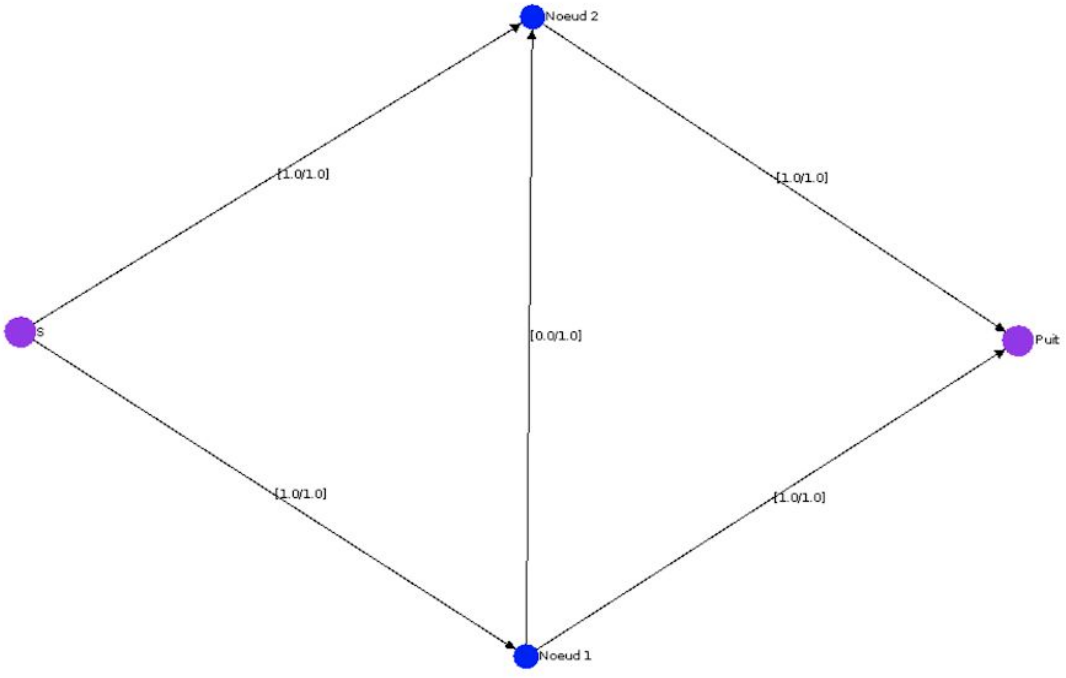

Image 6 : résultat graphique exécution algorithme sur le graphe du cours

IV. Résultat

A. Résultat graphique

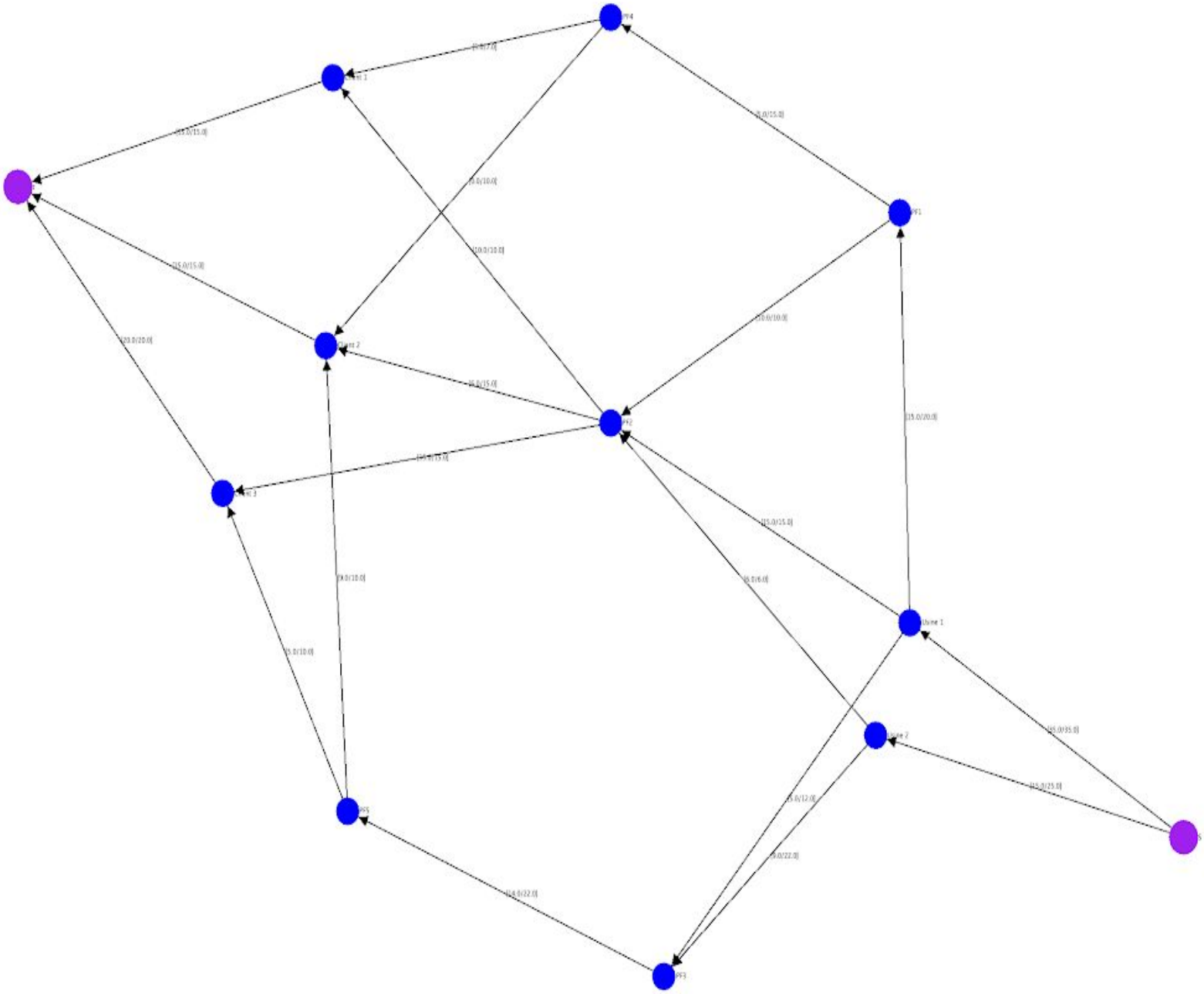

Image 7 : Affichage graphique du graphe du TP

On représente le noeud source, S, et le noeud puits, T, de couleur violette. Le reste des noeuds sont de couleur bleue. Pour tous les arcs on affiche le flux et la capacité.

B. Résultat textuel

```
Algorithme FF : Source S , target t
Chaine améliorante : [S Usine 1 PF1 PF2 Client 1 t ]
Min res capacité :10.0

Chaine améliorante : [S Usine 1 PF1 PF4 Client 1 t ]
Min res capacité :5.0

Chaine améliorante : [S Usine 1 PF1 PF4 Client 1 PF2 Client 2 t ]
Min res capacité :2.0

Chaine améliorante : [S Usine 1 PF1 PF4 Client 2 t ]
Min res capacité :3.0

Chaine améliorante : [S Usine 1 PF2 Client 1 PF4 Client 2 t ]
Min res capacité :2.0

Chaine améliorante : [S Usine 1 PF2 Client 2 t ]
Min res capacité :8.0

Chaine améliorante : [S Usine 1 PF2 Client 3 t ]
Min res capacité :5.0

Chaine améliorante : [S Usine 2 PF2 Client 2 PF4 PF1 Usine 1 PF3 PF5 Client 3 t ]
Min res capacité :5.0

Chaine améliorante : [S Usine 2 PF2 Client 3 t ]
Min res capacité :1.0

Chaine améliorante : [S Usine 2 PF3 PF5 Client 2 PF2 Client 3 t ]
Min res capacité :9.0

Flot Maximal determiné : 50.0
S->Usine 1 Capacité :35.0 / Flot :35.0
S->Usine 2 Capacité :25.0 / Flot :15.0
Usine 1->PF1 Capacité :20.0 / Flot :15.0
Usine 1->PF2 Capacité :15.0 / Flot :15.0
Usine 1->PF3 Capacité :12.0 / Flot :5.0
Usine 2->PF2 Capacité :6.0 / Flot :6.0
Usine 2->PF3 Capacité :22.0 / Flot :9.0
PF1->PF2 Capacité :10.0 / Flot :10.0
PF1->PF4 Capacité :15.0 / Flot :5.0
PF2->Client 1 Capacité :10.0 / Flot :10.0
PF2->Client 2 Capacité :15.0 / Flot :6.0
PF2->Client 3 Capacité :15.0 / Flot :15.0
PF3->PF5 Capacité :22.0 / Flot :14.0
PF4->Client 1 Capacité :7.0 / Flot :5.0
PF4->Client 2 Capacité :10.0 / Flot :0.0
PF5->Client 2 Capacité :10.0 / Flot :9.0
PF5->Client 3 Capacité :10.0 / Flot :5.0
Client 1->t Capacité :15.0 / Flot :15.0
Client 2->t Capacité :15.0 / Flot :15.0
Client 3->t Capacité :20.0 / Flot :20.0
```

Image 8 : résultat textuel exécution de l'algorithme sur le graphe du TP

On observe dans un premier temps les itérations de l'algorithme lors de sa recherche de chaîne améliorante. C'est à dire, la chaîne améliorante déterminée et le de cette chaîne. La deuxième partie est l'affichage du flot maximal final déterminé par l'algorithme. Finalement on peut lire les différents chemin du graphe avec leur capacité et le flot associé une fois l'algorithme de Ford-Fulkerson terminé.