INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

Algorithme de Ford et Fulkerson

Conception Détaillée

PARIS Geoffrey
PINEAU Anna
ITI4

Recherche Opérationnelle & Théorie des graphes

23 novembre 2020

Table des matières

1 Introduction

L'algorithme de Ford Fulkerson est un algortihme de résolution de problème du flot maximum. IL a été crée par Lester Randolph Ford Junior et D. R. Fulkerson.

C'est un algorithme de complexité en $O(m^2n)$ pour un graphe avec n noeuds et m arêtes.

2 Analyse descendante

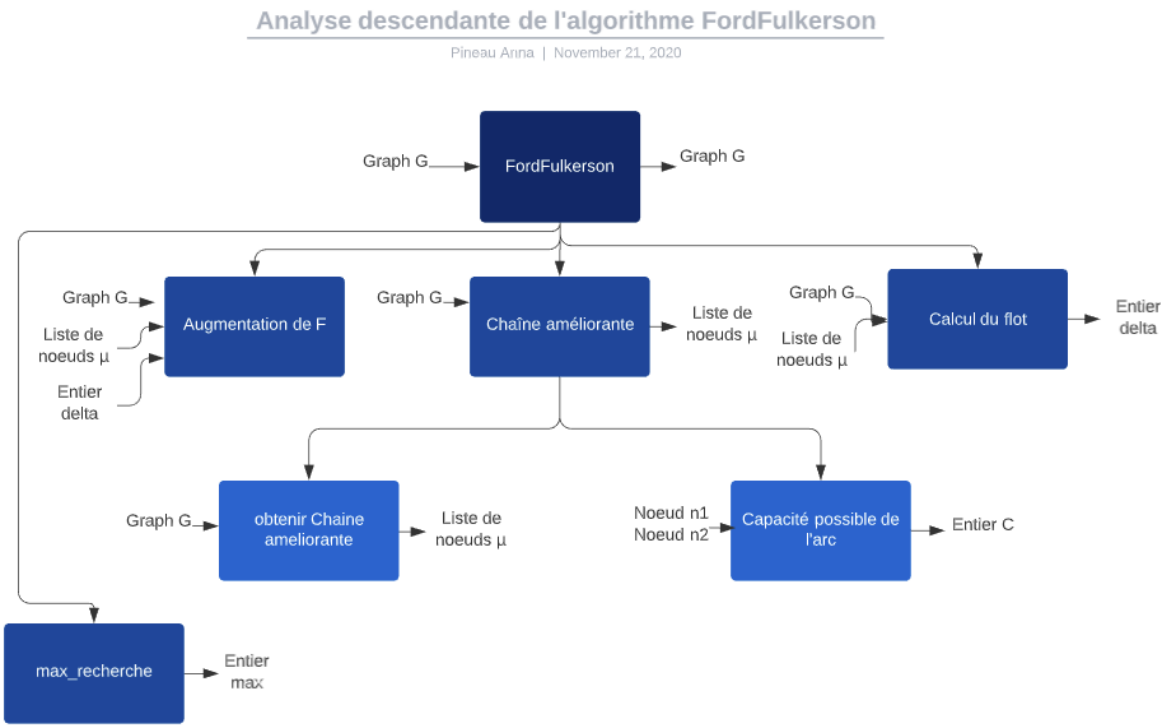

FIGURE 1 – Analyse Descendante

3 Algorithme général : FordFulkerson

procédure FordFulkerson (**E** G : **Graphe**, **S** F : **Graphe**)
debut
  F ← G
  repeter
    μ ← chaine_ameliorante(F)
    si non vide(μ) **alors**
      δ ← calcul_flot(μ)
      maj_flux(F,μ,δ)

```
    finsi
  jusqu'a ce que  vide(μ)
fin
```

3.1 fonction de la recherche du flot maximum pouvant être atteint

```
fonction max_recherche () : Entier
  Déclaration  max1 : Entier, max2 : Entier
debut
  pour Noeud j  ←predecesseurs(t) à  faire
    max1 += ObtenirCapacité(j,t)
  finpour
  pour Noeud j  ←successeurs(s) à  faire
    max2 += ObtenirCapacité(s,t)
  finpour
  retourner min(max1,max2)
fin
```

3.2 fonction chaine améliorante

```
fonction chaine_ameliorante (G : Graphe) : liste de noeuds
  Déclaration  z : Liste de Noeuds, arc_arriere : Booleen, i,s,t : Noeud, visiter :
               Dictionnaire
debut
  s ← obtenirNoeud(G,'s')
  t ← obtenirNoeud(G,'t')
  visiter ←  dictionnaire(obtenirChaqueNoeud(G), False)
  z ← liste()
  ajouterliste(z,s)
  visiter[s] = True
  repeter
    arc_arriere ← True
    x ← premierElementFile(z)
    retirerListe(z,x)
    noeuds_successeurs ← successeurs(x)
    pour i ←1 à longueur(noeuds_successeurs) faire
      si !visiter[noeuds_successeurs[i]] et capacite_possible( G, x, noeuds_successeurs[i])
      alors
        ajouterfile(z,noeuds_successeurs[i])
        visiter[noeuds_successeurs[i]] = True
        noeudAttributAjout(obtenirNoeud(G,noeuds_successeurs[i]), x)
        arc_arriere ← false
      finsi
```

```
            si arc_arriere alors
               si ObtenirFlot(G,j,i)¿0 et !visiter[noeuds_successeurs[i]]  alors
                  ajouterfile(z,noeuds_successeurs[i])
                  visiter[noeuds_successeurs[i]] = True
                  noeudAttributAjout(obtenirNoeud(G,noeuds_successeurs[i]), x)
               finsi
            finsi
         finpour
      jusqu'a ce que z = 0 ou visiter[t]
      si visiter[t] alors
         retourner obtenirChaine(G)
      finsi
      retourner [ ]
fin
```

3.2.1 fonction obtenir chaine ameliorante

```
fonction obtenirChaine (G : graphe) : Liste de Noeuds
debut
   x ← obtenirNoeud(G,t)
   chaine_am ← [ ]
   repeter
      y ← obtenirAttribut(G,x)
      ajouterAuDebut(chaine_am,obtenirNoeud(y))
      x ← y
   jusqu'a ce que x = obtenirNoeud(G,s)
   retirerAttributNoeuds(G)
   retourner chaine_am
fin
```

3.2.2 fonction capacite_possible

```
fonction capacite_possible (G : Graphe , n1,n2 : Noeud) : Entier
   Déclaration  arete : Arete, capacite, flot : Entier
debut
   arete ← obtenirArete(G,n1,n2)
   capacite ← obtenirCapacite(arete)
   flot ← obtenirFlot(arete)
   retourner capacite - flow
fin
```

3.3 procédure calcul de l'augmentation de flot

```
fonction calcul_delta (μ : Liste de Noeuds, G : Graphe) :  Entier
```

```
   Déclaration  aretes : Tableau[1 .. longueur(μ) - 1]  de Aretes, delta : Entier
debut
   delta ← MAX_FLOW
   aretes ← obtenirChaqueAretes(μ)
   pour chaque arete de aretes
      si estPresent(successeurs(source(arete)),destination(arete) alors
         delta ← min(delta, capaciteDisponible(source(arete)),destination(arete)
      finsi
      si estPresent(successeurs(destination(arete)),source(arete)) alors
         delta ← min(delta, obtenirFlot(destination(arete),source(arete)))
      finsi
      retourner delta
   finpour
fin
```

3.4 procédure augmentation de F et des flux des arcs

```
procédure maj_flux (mu : tableau de noeuds, G : Graph, δ : Entier)
   Déclaration  aretes : tableau d'arêtes
debut
   aretes ← obtenirChaqueArete(μ)
   pour i ←0 à longueur(aretes) faire
      si successeurs(i,i-1) alors
         donnerFlot(aretes[i],obtenirFlot(arete[i])+δ)
      finsi
      si successeurs(b,a alors
         donnerFlot(aretes[i],obtenirFlot(arete[i])-δ)
      finsi
   finpour
fin
```

3.5 fonctions prédecesseurs et successeurs

```
fonction predecesseurs (noeud : Noeud) : Tableau[1..degreEntrant(noeud)] de Noeud
   Déclaration  i : Entier, res : Tableau[1..degreEntrant(noeud)] de Noeud
debut
   pour i ←1 à degreEntrant(noeud) faire
      res[i] ← obtenirNoeudSource(obtenirAreteEntrante(noeud,i))
   finpour
   retourner res
fin

fonction successeurs (noeud : Noeud) : Tableau[1..degreSortant(noeud)] de Noeud
```

Déclaration i : **Entier**, res : **Tableau**[1..degreSortant(noeud)] **de** Noeud
debut
  pour i ←1 **à** degreSortant(noeud) **faire**
    res[i] ← obtenirNoeudDestination(obtenirAreteSortante(noeud,i))
  finpour
  retourner res
fin

4 Déroulement de l'algorithme

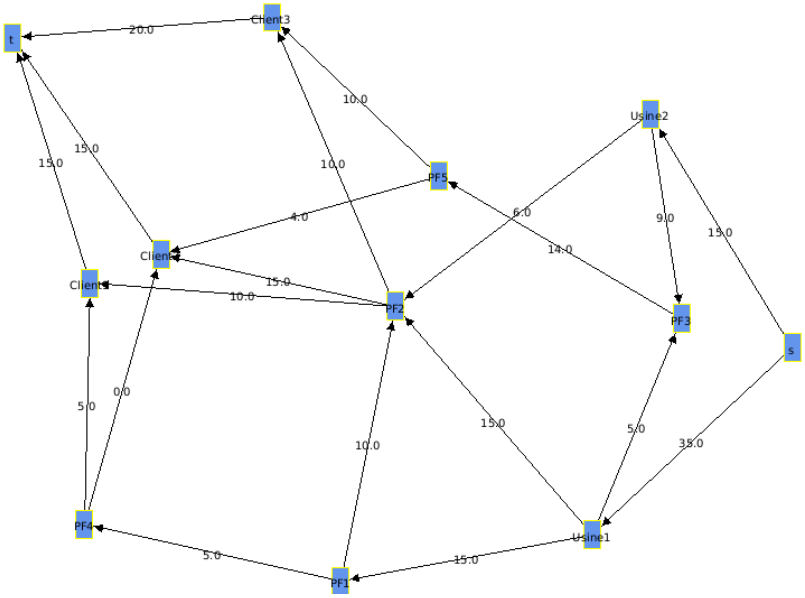

FIGURE 2 – Affichage Graphique

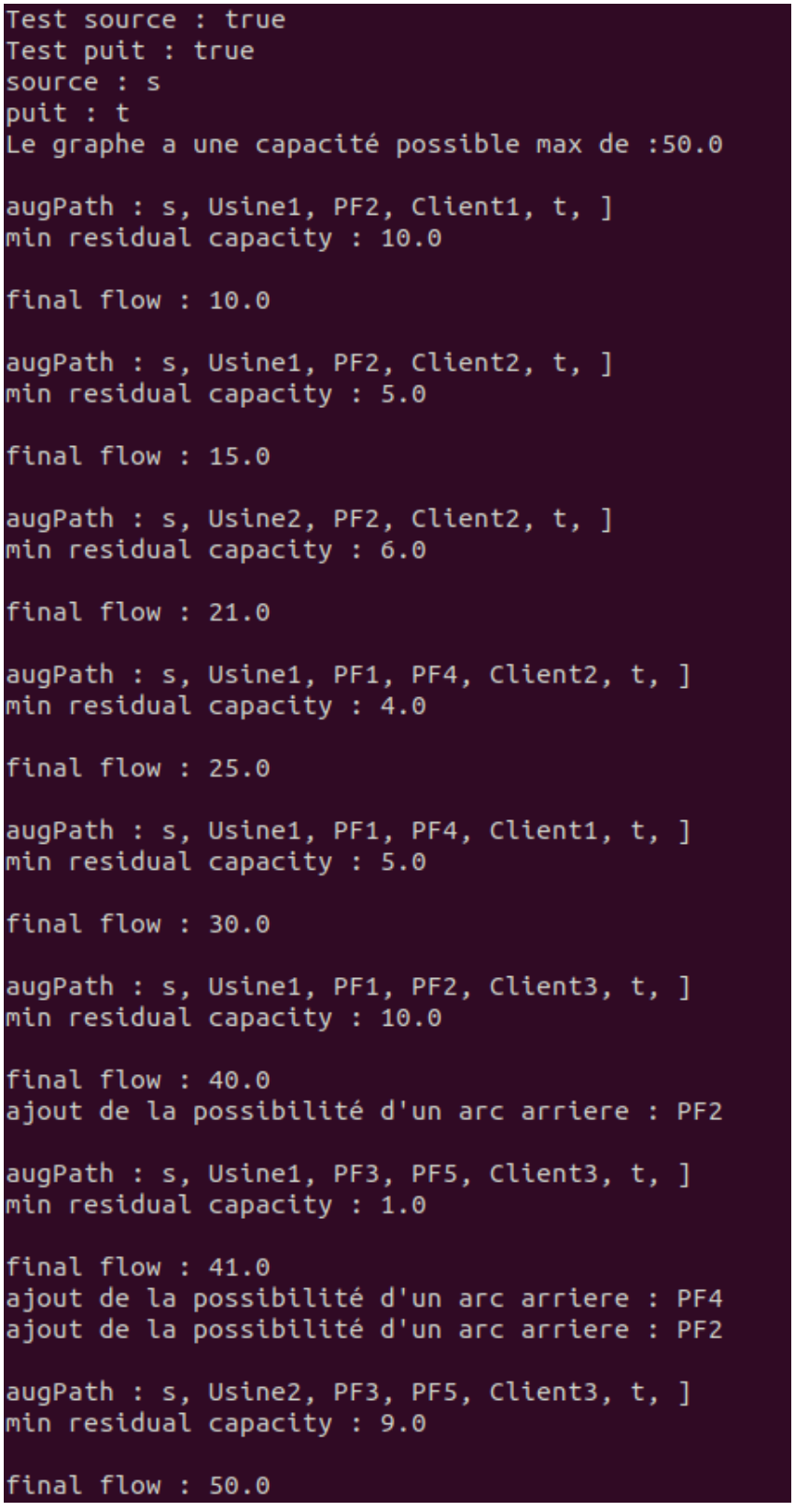

FIGURE 3 – Déroulement de l'algorithme avec les différentes chaines améliorantes

5 Gestion des arcs arrières

Nous avons géré les arcs arrières. Nous avons ensuite testé notre programme avec le graphe de la slide 21.

```
Algorithme de Ford-Fulkerson appliqué au graphe G_arcs_arrieres
source : s
puit : t
Le graphe a une capacité possible max de :2.0

augPath : s, 1, t, ]
min residual capacity : 1.0

final flow : 1.0

augPath : s, 2, t, ]
min residual capacity : 1.0

final flow : 2.0
```

FIGURE 4 – Déroulement de l'algorithme avec les différentes chaines améliorantes

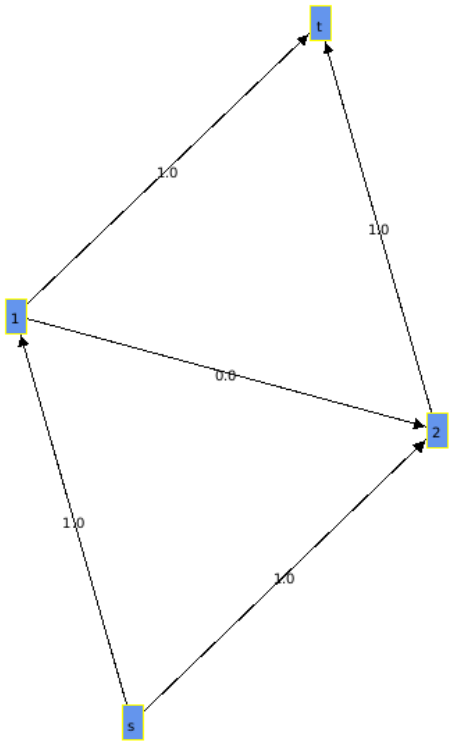

FIGURE 5 – Affichage Graphique

6 Fonctions et procédures présentes dans les bibliothèques Graph et FlowAlgorithmeBase

obtenirChaqueArete - getEachEdge

donnerFlot- setFlow

obtenirFlot - getFlow

obtenirNoeuds - getEachNode

obtenirCapacité - getCapacity

Références

Géraldine Del Mondo - https://moodle.insa-rouen.fr/pluginfile.php/93576/mod_resource/content/4/seance3_2020.pdf Novembre 2020

RANDRIANARIVELO Tiana MSSALAK Meryem ZERJAL Dimitri AMIRAULT Martin - https://moodle.insa-rouen.fr/pluginfile.php/31526/mod_label/intro/documentation.pdf Novembre 2020

https://moodle.insa-rouen.fr/pluginfile.php/130520/mod_resource/content/5/TP4-RO-sujet.pdf

https://www.pairform.fr/doc/1/32/180/web/co/FlotFulkerson.html#:~:text=Pour%20trouver%20une%20cha%C3%AEne%20am%C3%A9liorante,algorithme%20d'exploration%20en%20largeur.&text=Si%20on%20arrive%20%C3%A0%20marquer,existe%20pas%20de%20telle%20cha%C3%AEne.