DOSSIER "OPTIMISATION ET RECHERCHE OPÉRATIONNELLE"

Julien Ah-Pine et JAP
Université Lyon 2 / ICOM M1 Informatique 2020/2021

Réalisé par :
ZINEEDDINE LAKHDARI
MAME LIBASSE MBOUP

Table des matières :

1-Introduction :

-Ce dossier entre dans le cadre de du contrôle continu des connaissances dans le contexte du cours de théorie des graphes .
-Dans ce dossier, nous allons présenter les différents algorithmes : leurs objectifs, leurs implémentations, leurs explications , comment nous avons procédé pour l'implémentation de ces derniers.
-Les algorithmes auxquels nous nous sommes intéressé sont les suivants :

1. Détection de l'arbre recouvrant de poids minimal par Prim.
2. Calcul des plus courts chemins par Ford-Bellman.
3. Détermination d'un flot maximal dans un réseau avec capacités par Ford-Fulkerson.

Nous détaillerons par la suite l'étude de chacun de ces différents algorithms .

2-Algorithme 1 : détection de l'arbre recouvrant de poids minimal par Prim

2.1 - Objet de l'algorithme :

Soit un graphe non orienté ,connexe, valué , que l'on notera par G = [X, U].
Ce graphe est représenté par sa matrice d'adjacence pondérée notée A.
Nous nous intéressons de déterminer un arbre partiel de poids minimal par PRIM.

En d'autres termes , cet algorithme trouve un sous-ensemble d'arêtes qui forment un arbre sur l'ensemble de tous les sommets du graphe initial G=[X,U] et tel que la somme des arêtes soit minimale .

2.2 - Pseudo-code de l'algorithme:

Nous rappelons ci-dessous le pseudo-code de l'algorithme qui permet de déterminer un arbre partiel de poids minimal par PRIM sur un graphe.

Input : G = [X, U]
1 X0 ← {i} où i est un sommet de X pris au hasard
2 U 0 ← ∅
3 **Tant que X' ≠ X** faire
5 **Choisir une arête (j, k)** de poids minimal tel que j ∈ X' et k /∈ X'
6 X' ← X'∪ {k}
7 U' ← U' ∪ {(j, k)}
8 **Fin Tant que**
9 **Output** : G' = [X' , U']

2.3 Code R de l'algorithme :

Voici ci-dessous le code R de notre implémentation de l'algorithme présenté dans le paragraphe précédent.

```
X=list(1,2,3,4,5,6,7)

Arbre_Couvrant = function(X,U){
 #i est le sommet que je l'ai sélectionné ici le 6
 i=6
 #Boolean ou je vais l'utiliser pour stoper m'as boucle .
 fin=TRUE
 #X' reçoit le sommet sélectionnée
 X_prime <- c(i)
 #Matrice égale a la matrice d'adjacence.
 A_prime= matrix(100,nrow =7,ncol= 7)
 #U' reçoit un tab vide
 U_prime<-c()
 #tant que la longueur de x est diff de x' cela veut dire que j'ai pas encore visité tous les sommets
   while (length(X_prime)!=length(X))
  {
    while (fin) {
```

```
      for(j in 1:nrow(A)){
        #si le sommet exisite dans x' je cherche son voisin dans la matrice d'adjacence
        if((is.element(i,X_prime)) & !(is.element(j,X_prime)) ){
          #si le sommet j est un voisin
          if(A[i,j]>0){
           #je sauvgarde cette valeur dans la matrice que je l'ai créer
           A_prime[i,j]=A[i,j]
          }
        }
      }
       #je cherche le minimum dans la matrice
      x<-which(A_prime==min(A_prime),arr.ind=T)
      #si cette element n'est pas dans x'
      if(!(is.element(x[1,2],X_prime)))
      {
        #je sauvgarde les coordonnées. dans U' et X'
        U_prime<-c(U_prime,x[1,])
        X_prime<-c(X_prime,x[1,2])
        #après le i devient le sommet sélectionné
        i=x[1,2]

        A_prime[x[1],x[2]]=100
      }
      #Condition d'arret de ma boucle si j'ai visité tous les sommets de ce graphe.
      if(length(X)==length(X_prime)){
        fin=FALSE
      }
     }
   }
  #Seulement pour voir mes résultats j'ai pas retourner j'ai afficher .
  print("U PRIME")
  print(U_prime)
  print("X PRIME")
  print(X_prime)
}
```

2.4 Illustration sur un exemple:

Nous illustrons le bon fonctionnement de l'algorithme implémenté en l'exécutant sur le graphe dont la matrice d'adjacence est donnée par :

```
A = rbind(c(0,5,8,0,0,0,0),
        c(5,0,0,4,2,0,0),
        c(8,0,0,0,5,2,0),
        c(0,4,0,0,0,0,7),
        c(0,2,5,0,0,0,3),
        c(0,0,2,0,0,0,3),
        c(0,0,0,7,3,3,0))
```

L'exécution du code R présent dans la sous-section précédente donne le résultat suivant :

```
> Arbre_Couvrant(X,A)
[1] "U PRIME"
 ( 6  3 ) ,( 6  7) , (7  5) ,( 5  2) , ( 2  4) , ( 2  1)
[1] "X PRIME"
 6  3  7  5  2  4  1
>
```

Si on fait l'addition des poids des arcs sélectionnée par l'algorithme on trouve bien le poids minimal qui est : 19.

3-Algorithme 2 : calcul des plus courts chemins par Ford-Bellman.

3.1 Objet de l'algorithme

Soit un graphe orienté, valué et sans circuit absorbant (de longueur négative) que l'on notera par G = [X,U]. Ce graphe est représenté par sa matrice d'adjacence notée A. Nous cherchons à déterminer le plus court chemin d'un sommet noté s aux autres sommets dans ce graphes dont les longueurs sont quelconques .
Explicitement nous allons travailler sur l'ensemble des prédécesseurs d'un sommet. L'algorithme affine successivement une borne supérieure de la longueur du plus
court chemin entre s et tous les autres sommets jusqu'à atteindre la longueur minimale.

3.2 Pseudo-code de l'algorithme

Nous rappelons ci-dessous le pseudo-code de l'algorithme qui permet de déterminer le plus court chemin du sommet s aux autres sommets .

Input : G = [X,U], s
1 π (s) $\leftarrow$ 0
2 **Pour tout** i $\in$ {1, 2, ...,N} \ {s} faire
3 π (i) $\leftarrow +\infty$
4 **Fin Pour**

5 **Répéter**
6 **Pour tout** i ∈ {1, 2, ...,N} \ {s} faire
7 π (i) ← min(π (i) , min [j ∈ Γ (-1)(i)] π (j) + lji) ;
8 **Fin Pour**
9 **Tant que** une des valeurs (i) change dans la boucle Pour
10 **Output** : π

3.3 Code R de l'algorithme

Voici ci-dessous le code R de notre implémentation de l'algorithme présenté dans le paragraphe précédent.

```
 #Algorithme de calcul des plus court chemin par Ford-Bellman
 #INPUT
 #X est l'ensemble des sommets
 #A est la matrice d'adjacence pondérée
 #s le sommet initial
 #OUTPUT
 #pi vecteur donnant la longueur du ppc entre s et les autres sommets

#Longueur du vecteur X
n=length(X)
pi=rep(0,n)
pi[s]=0
Sb=setdiff(X,s)#initialisation de Sb

for (i in Sb)
{
 pi[i] = Inf
}

#initialisation de la variable permettant de stocker
#la dernière valeur de pi
stock = NULL

#boucle repeat until
repeat{
 for (i in Sb)
 {
        #calcul du plus court chemin
        pi[i] = min(pi[i],pi[which(A[,i]!=0)]+A[which(A[,i]!=0),i])
 }

 #on sort de la boucle si la dernière valeur de pi ne change pas
 if (setequal(pi,stock)==TRUE) break
 stock = pi
```

```
}
return(pi)
}
```

3.4 Illustration sur un exemple

Nous illustrons le bon fonctionnement de l'algorithme implémenté en l'exécutant sur le graphe dont la matrice d'adjacence est donnée par :
D'abord sur la matrice A suivante :

```
A=rbind(c(0,5,8,0,0,0,0)
        ,c(5,0,0,4,-2,0,0)
        ,c(8,0,0,0,5,2,0)
        ,c(0,4,0,0,0,0,7)
        ,c(0,-2,5,0,0,0,3)
        ,c(0,0,2,0,0,0,-3)
        ,c(0,0,0,7,3,-3,0))
```

-On est sur un cas particulier car nous avons un circuit absorbant .
Ensuite sur la matrice A1 suivante:

```
A1=rbind(c(0,5,8,0,0,0,0)
        ,c(5,0,0,4,0,0,0)
        ,c(8,0,0,0,5,2,0)
        ,c(0,4,0,0,0,0,0)
        ,c(0,0,5,0,0,0,3)
        ,c(0,0,2,0,0,0,-3)
        ,c(0,0,0,0,3,-3,0))


X = c(1,2,3,4,5,6,7)
F = Ford_Belleman(X,A1,7)
F
[1]  7 12 -1 16  3 -3  0
```

4-Algorithme 3 : détermination d'un flot maximal dans un réseau avec capacités par Ford-Fulkerson.

4.1 Objet de l'algorithme

L'algorithme de Ford-Fulkerson est un algorithme qui permet de résoudre le problème du flot maximum, un problème d'optimisation classique dans le domaine de la recherche opérationnelle.

Soit un graphe, valué (de longueur non négative) que l'on notera par G = [X, U, C] .Ce graphe est représenté par sa matrice d'adjacence notée A,

ensemble de sommets X , un sommet source s et un sommet puits p de X . Nous cherchons à déterminer le flot P de valeur maximale obtenus par l'algorithme de Ford-Fulkerson.

4.2 Pseudo-code de l'algorithme

Nous rappelons ci-dessous le pseudo-code de l'algorithme qui permet de déterminer un flot maximal dans un réseau avec capacités.

Input : G = [X, U, C], un flot réalisable
1 ms ← (∞, +) et S = {s}
2 **Tant que** ∃ (j ∈ S, i ∈ S) : (cij − ij > 0) ∨ (ji > 0) **faire**
3 **Si** cij − ij > 0 **faire**
4 mj ← (i, αj , +) avec αj = min{αi , cij − ij}
5 **Sinon** Si ji > 0 **faire**
6 mj ← (i, αj , −) avec αj = min{αi , ji}
7 **Fin Si**
8 S ← S ∪ {j}
9 **Si** j = p **faire**
10 V () ← V () + αp
11 Aller en 14
12 **Fin Si**
13 **Fin Tant que**
14 **Si** p ∈ S **faire**
15 **Tant que** j ≠ s **faire**
16 **Si** mj (3) = + **faire**
17 mj (1)j ← mj (1)j + αp
18 **Sinon Si** mj (3) = − **faire**
19 jmj (1) ← jmj (1) − αp
20 **Fin Si**
21 j ← mj (1)
22 **Fin Tant que**
23 Aller en 1
24 **Sinon faire**
25 **Output :**

26 **Fin Si**

4.3 Code R de l'algorithme

Voici ci-dessous le code R de notre implémentation de l'algorithme présenté dans le paragraphe précédent .

```
FordFulkerson = function(X,A,s,p)
{

  phi = 0 #flot realisable
  #Preparation des triplets de chaque noeuds
  capacites = List(X,s)
  chaine = paste("m",s,sep="")
  capacites[chaine,2] = 9999
  capacites[chaine,3] = "+"
  #Initialisation d'une matrice P qui va contenir le flot qui passe entre deux
noeuds
  P = matrix(0, nrow = length(X), ncol = length(X))
  P[s,s]=9999 #flot qui part de la source, vaut :  +l'infini
  while(TRUE)
  {
        S = c(s)#Liste des noeuds marqués, s marqué dès le départ de chaque
itération
        Sb = setdiff(X,s)#Liste des noeuds non marqués
        tmp = A-P > 0 #Test Cij - PHIij > 0 de l'algorithme
        tmp2 = t(P) >0#Recuperation des PHI non nuls
        C= tmp | tmp2
        index = which(matrix(C[S,Sb] ==TRUE,nrow=length(S),
ncol=length(Sb)),arr.ind = TRUE)
        index2 = which(matrix(P[Sb,S] > 0, nrow=length(Sb), ncol = length(S)),
arr.ind = TRUE)

        while(length(index)>0 || length(index2)>0)#Ligne 2 algo
        {
        if(length(index)>0)
```

```
        {#ligne 3
        i = index[1,1]
        i = S[i]
        j = index[1,2]
        j = Sb[j]
        #Construction de la chaine de caracteres pour correspondre a la ligne du
triplet mi et mj
        chaine = paste("m",i,sep="")
        chaine2 = paste("m",j,sep="")
        capacites[chaine2,1] = i
        capacites[chaine2,2] = min(capacites[chaine,2],A[i,j]-P[i,j])
        capacites[chaine2,3] = "+"
        #mj=(i,min(capacites[chaine,2],A[i,j]-P[i,j]),"+")
        }
        else
        {#ligne 5
        if(length(index2)>0)
        {
        i = index2[1,1]
        i = S[i]
        j = index2[1,2]
        j = Sb[j]
        #On met a jour le triplet mj dans le dataframe
        chaine=paste("m",i,sep="")
        chaine2 = paste("m",j,sep="")
        capacites[chaine2,1] = i
        capacites[chaine2,2] = min(capacites[chaine,2],P[j,i])
        capacites[chaine2,3] = "-"
        #mj=(i,min(capacites[chaine,2],P[j,i]),"-")
        }
        }
        #On met dans S le sommet qui vient d'etre mis a jour(on le marque)
        S = cbind(S,j)#ligne 8
        Sb = setdiff(X,S)
        #Est ce que j est le puit?
        if(j==p)
```

```
        {#oui, on met a jour la valeur du flot maximal et on arrete la boucle
courrante
        phi = phi + capacites[chaine2,2]
        phiP = capacites[chaine2,2]
        break
        }
        #On met a jour l'index
        tmp = A-P > 0
        tmp2 = t(P)
        C= tmp | tmp2
        index = which(matrix(C[S,Sb] ==TRUE,nrow=length(S),
ncol=length(Sb)),arr.ind = TRUE)
        index2 = which(matrix(P[Sb,S] > 0, nrow=length(Sb), ncol = length(S)),
arr.ind = TRUE)
        }
        #Le puit est il marque?
        if(is.element(p,S))
        {#ligne 14
        repeat
        {#ligne 15
        if(j==s)
        {
        break
        }
        chaine = paste("m",j,sep="")
        if(capacites[chaine,3]=="+")
        {#ligne 16
        i = capacites[chaine,1]
        P[i,j] = P[i,j]+phiP#Mise a jour de P aux indices des sommets qui ont
permis d'atteindre le puit en additionnant la valeur du flot
        }
        else
        {
        if(capacites[chaine,3]=="-")
        {
        i = capacites[chaine,1]
```

```
        P[j,i] = P[j,i]-phiP#Mise a jour de P aux indices des sommets qui ont
permis d'atteindre le puit en soustrayant la valeur du flot
        }
        }
        j = capacites[chaine,1]#ligne 21
        }
        }
        else
        {
        break
        }

  }
  return(phi)
}
```

4.4 Illustration sur un exemple

Malheureusement l'implémentation de cet algo ne marche pas sur l'exemple représenté dans la Figure 1 .

5-CONCLUSION

Dans ce qu'il précède il résulte de notre travail sur l'implémentation d' algorithmes de graphes suivants : détection de l'arbre recouvrant de poids minimal par Prim ; calcul des plus courts chemins par Ford-Bellman et détermination d'un flot maximal dans un réseau avec capacités par Ford-Fulkerson .

Ce travail nous a permis de consolider nos compétences acquises en TD/TP sur la compréhension au déroulement de ces algorithmes , de leur cheminement en pseudo-code, enfin leur implémentation sur R .

Les principales difficultés que nous avons rencontrées ont été principalement sur l'algo 3 de Ford Fulkerson qui n'a pas marché sur l'exemple indiqué . Ainsi si nous avions eu encore un peu plus de temps nous aurions pu peut-être finaliser le code qui fonctionne pas sur cet exemple .