Rapport - Optimisation et Recherche Opérationnelle

Nathan Coustance & Yanis Benreguig

Présentation

Ce rapport présente le travail que nous avons effectué lors de notre projet, notre objectif étant de traduire le pseudo-code de plusieurs algorithmes de théorie des graphes en langage R.
Il s'agit des algorithmes de ***Prim***, de ***Ford-Bellman*** ainsi que de ***Ford-Fulkerson***.

- ***Prim***: Calcul d'un arbre couvrant minimal dans un graphe valué non orienté.

- ***Ford-Bellman***: Calcul des plus courts chemins depuis un sommet source dans un graphe valué orienté.

- ***Ford-Fulkerson***: Calcule le flot maximal d'un graphe à partir d'un sommet source et d'un sommet puit dans un graphe valué orienté.

Algorithme de Prim

Description

Soit un graphe G valué non orienté, on y cherche un arbre couvrant minimal *(Minimum Spanning Tree)*. Cet algorithme consiste, à partir d'un sommet aléatoire, à trouver un ensemble d'arête de G formant un arbre de telle sorte à ce que la somme des poids de ces arêtes soit minimale.
On part donc de ce sommet pris au hasard puis on construit petit à petit notre arbre en trouvant, à chaque étape, une arête de poids minimal ayant exactement un sommet en commun avec notre arbre en construction. Une fois tous les sommets présents dans l'arbre, l'algorithme a fini son travail.

Pseudo-Code

Input : $G = [X, U]$
$X' \leftarrow \{i\}$ où i est un sommet de X pris au hasard
$U' \leftarrow \emptyset$
Tant que $X' \neq X$ **faire**
    Choisir une arête (j, k) de poids minimal tel que $j \in X'$ et $k \notin X'$
    $X' \leftarrow X' \cup \{k\}$
    $U' \leftarrow U' \cup \{(j, k)\}$
Fin Tant que
Output : $G' = [X', U']$

Code R

```r
Prim = function(X, A) {
  visited = c(sample(X,1)) # Initialisation de la liste des sommets visités par un sommet pris au hasard
  mst = c() # Initialisation de notre Minimum Spanning Tree
  edges = which(A!=0, arr.ind=T) # Récupération des arêtes à partir de notre matrice d'adjacence

  while(length(visited) != length(X)) {
    possible = list() # Liste des arêtes possibles
    for (node in visited) {
      neighbours = edges[which(edges[,'row']==node), 'col'] # On récupère la liste des voisins d'un noeud
      neighbours = neighbours[which(!(neighbours %in% visited))] # On prend uniquement ceux qui ne sont pas visités
      for (neighbour in neighbours) {
        possible[[length(possible)+1]] = c(node, neighbour) # On les ajoute à la liste des arêtes possibles
      }
    }
    minval = Inf # On itialise un minimum à l'infini
    cursor = c() # Variable utilisée pour contenir notre arête minimale
    for (edge in possible) { # Pour chaque arête possible
      # Si sa valeur est inférieure au minimum stocké, on met le curseur dessus et on change le minimum
      if (A[edge[1], edge[2]] < minval) {
        minval = A[edge[1], edge[2]]
        cursor = edge
      }
    }
    visited = append(visited, cursor[2]) # On ajoute notre nouveau noeud visité
    mst = append(mst, paste(cursor[1],'-',cursor[2], sep="")) # On ajoute l'arête possible minimale à notre arbre
  }
  return(list(visited, mst))
}
```

Exemple

Soit un graphe G avec X la liste de ses sommets et A sa matrice d'adjacence représentée ci-dessous :

$$\begin{pmatrix} 0 & 5 & 8 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 4 & 2 & 0 & 0 \\ 8 & 0 & 0 & 0 & 5 & 2 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 7 \\ 0 & 2 & 5 & 0 & 0 & 0 & 3 \\ 0 & 0 & 2 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 7 & 3 & 3 & 0 \end{pmatrix}$$

```r
Prim(X, A)
```

```
## [[1]]
## [1] 3 6 7 5 2 4 1
##
## [[2]]
## [1] "3-6" "6-7" "7-5" "5-2" "2-4" "2-1"
```

Ici on a en premier la liste de nos sommets ajoutés dans l'ordre chronologique.
En deuxième valeur de retour, nous avons la liste des arêtes qui constitue notre arbre.

Algorithme de Ford-Bellman

Description

Pseudo-Code

Input : $G = [X, U], s$

$\pi(s) \leftarrow 0$

Pour tout $i \in \{1, 2, \ldots, N\} \setminus \{s\}$ **faire**

    $\pi(i) \leftarrow +\infty$

Fin Pour

Répéter

    Pour tout $i \in \{1, 2, \ldots, N\} \setminus \{s\}$ **faire**

        $\pi(i) \leftarrow \min(\pi(i), \min_{j \in \Gamma^{-1}(i)} \pi(j) + l_{ji})$;

    Fin Pour

Tant que une des valeurs $\pi(i)$ change dans la boucle Pour

Output : π

Code R

Exemple

Algorithme de Ford-Fulkerson

Description

Pseudo-Code

Input : $G = [X, U, C]$, φ un flot réalisable
1 $m_s \leftarrow (\infty, +)$ et $S = \{s\}$
2 **Tant que** $\exists (j \in \overline{S}, i \in S) : (c_{ij} - \varphi_{ij} > 0) \vee (\varphi_{ji} > 0)$ **faire**
3   **Si** $c_{ij} - \varphi_{ij} > 0$ **faire**
4         $m_j \leftarrow (i, \alpha_j, +)$ avec $\alpha_j = \min\{\alpha_i, c_{ij} - \varphi_{ij}\}$
5   **Sinon Si** $\varphi_{ji} > 0$ **faire**
6         $m_j \leftarrow (i, \alpha_j, -)$ avec $\alpha_j = \min\{\alpha_i, \varphi_{ji}\}$
7   **Fin Si**
8     $S \leftarrow S \cup \{j\}$
9   **Si** $j = p$ **faire**
10         $V(\varphi) \leftarrow V(\varphi) + \alpha_p$
11   Aller en 14
12   **Fin Si**
13 **Fin Tant que**
14 **Si** $p \in S$ **faire**
15   **Tant que** $j \neq s$ **faire**
16   **Si** $m_j(3) = +$ **faire**
17             $\varphi_{m_j(1)j} \leftarrow \varphi_{m_j(1)j} + \alpha_p$
18   **Sinon Si** $m_j(3) = -$ **faire**
19             $\varphi_{jm_j(1)} \leftarrow \varphi_{jm_j(1)} - \alpha_p$
20   **Fin Si**
21         $j \leftarrow m_j(1)$
22   **Fin Tant que**
23   Aller en 1
24 **Sinon faire**
25   **Output :** φ
26 **Fin Si**

Code R

Exemple