Parcours en largeur(BFS)

Sy, Ibrahima

Institut Supérieur Informatique (**ISI**)
Licence 2 GL

May 17, 2022

Overview

Introduction

Définition

L'algorithme de parcours en largeur (ou BFS, pour Breadth First Search en anglais) permet le parcours d'un graphe ou d'un arbre de la manière suivante : on commence par explorer un nœud source, puis ses successeurs, puis les successeurs non explorés des successeurs, etc. L'algorithme de parcours en largeur permet de calculer les distances de tous les nœuds depuis un nœud source dans un graphe non pondéré (orienté ou non orienté). Il peut aussi servir à déterminer si un graphe non orienté est connexe.

Principe

Un parcours en largeur débute à partir d'un nœud source. Puis il liste tous les voisins de la source, puis ensuite il les explore un par un. Ce mode de fonctionnement utilise donc une file dans laquelle il prend le premier sommet et place en dernier ses voisins non encore explorés. Les nœuds déjà visités sont marqués afin d'éviter qu'un même nœud soit exploré plusieurs fois. Dans le cas particulier d'un arbre, le marquage n'est pas nécessaire. Voici les étapes de l'algorithme :

1. Mettre le nœud source dans la file.
2. Retirer le nœud du début de la file pour le traiter.
3. Mettre tous ses voisins non explorés dans la file (à la fin).
4. Si la file n'est pas vide reprendre à l'étape 2.

Principe

L'algorithme de parcours en profondeur diffère du parcours en largeur car il continue l'exploration jusqu'à arriver un cul-de-sac. C'est seulement à ce moment là qu'il revient en arrière pour explorer depuis un autre nœud voisin. L'utilisation d'une pile au lieu d'une file transforme l'algorithme du parcours en largeur en l'algorithme de parcours en profondeur.

Pseudo code

```
fonction plusCourtChemin(G = (V, E), u, v) //V est de taille n
    visités[u] = ∅
    file = [u]
    fini = False
    tant que ¬file.vide() et ¬fini faire
        w = file.pop()
        pour chaque voisin z de w faire
            si z ∉ visités alors
                visités[z] = w
                file.append(z)
                si z == v alors
                    fini = True
        fin
    fin
    si v ∉ visités alors
        return "Aucun chemin n'existe"
    C = (v)                          // on va reconstruire le chemin
    w = v
    tant que w ≠ u faire
        w = visités[w]
        C.insérerDébut(w)
    fin
    return C
```

Exemple