Théorie des graphes - Chapitre 3 : Parcours de graphes

Table des matières

Un parcours va partir d'un sommet.
-Il va calculer un ordre de découverte/parcours
-Arbre de parcours/Arbre d'exploration

1 Parcours générique

Algorithm 1 PG(G,u) : GO

Require: G un graphe ; u un sommet
Ensure: GO un graphe orienté avec tous les points marqué.

```
// var
L : Une liste
x, y : Des sommets
// init
NouvelleListe(L) // L est vide
Marquer(u) // Marquer serait une fonction permettant de dire qu'on est passé par ce sommet.
L ← u // On ajoute u à la liste L
// process
while ¬ EstVide(L) do
    x ←TeteList(L) // On prend la tête de la liste
    GO ←ExtrairePremier(L) // On retire la première valeur de la liste
    foreach y ∈ Voisinage(x,g) do
        if nonMarquer(y)
            Marquer(y)
            L ← y
        end if
    end foreach
end while
return GO
```

`http://www-verimag.imag.fr/~wack/ALGO5/Cours10.pdf`
L'algorithme générique permet de déterminer si un graphe est connexe.

L'algorithme maintient 3 catégories de sommets
1) Les sommets traités
2) Les voisins des sommets traités
3) Les sommets non traités

2 Parcours en largeur (Breadth First Search)

Algorithm 2 BFS(G,u) :

Require: G un graphe ; u un sommet.

```
// var
F : Une file
// init
NouvelleFile(F)
F.enfiler(u)
Marquer(u)
// process
while ¬estVide(F) do
  u ← f.defiler()
  print u
  for all t ←Voisin(u) ∈ G do
    if nonMarquer(S) then
      F.enfiler(t)
      Marquer(t)
    end if
  end for
end while
```

`https://en.wikipedia.org/wiki/Breadth-first_search`

Application : Déterminer un plus cours chemin dans un graphe univalué. Déterminer facilement si un graphe est Biparti.