Rappels sur les Graphes

Introduction

Les graphes sont des modèles pour représenter des relations entre objets (graphe orienté) ou des relations symétriques entre eux (graphe non orienté)

Définitions - Terminologie

Graphe orienté :

Un graphe orienté G est un ensemble de sommets pouvant être connectés par des arcs.
G = (V,E) où V est l'ensemble des sommets et E l'ensemble d'arcs.

Les sommets d'un graphe peuvent par exemple représenter des objets et les arcs des relations entre objets.

S'il existe un arc entre le sommet u et v (u --> v) alors v est appelé un adjacent ou un voisin de u.

Dans la plupart des applications, on associe de l'information aux noeuds et/ou aux arcs. On parlera alors d'un graphe étiqueté.

Un chemin dans le graphe G=(V,E) est la séquence $\{v_1, v_2, ...v_n\}$ de sommets appartenant à V tels que les arcs (v_1-->v_2, v_2-->v_3, ..., v_{n-1}-->v_n) existent dans E. Sa longueur est le nombre d'arcs qui le composent. Dans le cas d'un graphe orienté étiqueté, le poids du chemin et la somme des poids des arcs qui le composent.

Un chemin est dit simple si tous ses sommets sont distincts.

Un circuit est un chemin de longueur >= 0 tel qu'il commence et finit au même sommet.

Une composante fortement connexe d'un graphe orienté G est un sous ensemble de ce graphe composé de l'ensemble maximal de noeuds dans lequel il existe un chemin de tout noeud de l'ensemble vers chaque autre noeud de l'ensemble.

Formellement, soit G=(V, E) un graphe. On peut partitionner V en classes d'équivalence : V1, V2, ...Vr
v equiv w <==> Il existe un chemin de v à w et un chemin de w à v.
Soit Ti l'ensemble des arcs avec les têtes et les queues dans Vi. Les graphes Gi=(Vi, Ei) sont appelés les composantes fortement connexes de G.

Un graphe est dit fortement connexe s'il a une seule composante fortement connexe.

Une arborescence est un graphe orienté ou il existe un sommet particulier 'r' relié à tout autre sommet du graphe par un chemin unique commençant par le sommet 'r'.

Graphe non orienté :

Un graphe non orienté G est un ensemble de sommets pouvant être connectés par des arêtes.
G = (V,E) où V est l'ensemble des sommets et E l'ensemble d'arêtes.
Chaque arête (u, v) représente en fait les deux arcs u-->v et v-->u.

On parlera de cycle au lieu de circuit, et de chaîne au lieu de chemin.

Un graphe non orienté est connexe si tous ses sommets sont connectés. Sinon chaque groupe de sommets connectés forme une composante connexe du graphe.

Un Arbre est un graphe non orienté connexe sans cycle.

Représentation mémoire

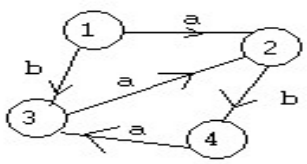
a) Par matrice associée

Si V = {v1, v2, ..., vn} est l'ensemble des sommets, le graphe est représenté par une matrice A[n, n] de booléens tels que A[i, j] est vrai s'il existe un arc de vi vers vj. Si le graphe est étiqueté, A[i, j] représentera la valeur.

b) Tableaux de listes linéaires chaînées (Liste d'adjacence)

Le graphe est représenté par un tableau Tete[1..n] où Tete[i] est un pointeur vers la liste des sommets adjacents à vi.

Exemple :



Graphe étiqueté

	1	2	3	4
1		a	b	
2				b
3		a		
4			a	

Matrice

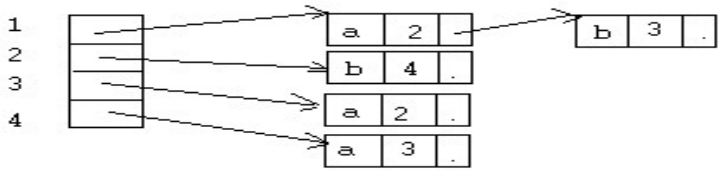


Tableau de listes

Parcours d'un graphe

a) Depht first search : Parcours en profondeur

C'est un type de parcours très utilisé sur les graphes : " Depht First Search " que l'on peut traduire par " Recherche en profondeur d'abord". C'est la généralisation du parcours Préordre sur les arbres. La plupart des algorithmes sur les graphes utilisent ce type de parcours que nous notons DFS par la suite.

Le principe est le suivant :

Initialement tous les noeuds sont marqués " non visités".

Choisir un noeud v de départ et le marquer " visité". Chaque noeud adjacent à v, non visité, est à son tour visité en utilisant DFS récursivement. Une fois tous les noeuds accessibles à partir de v ont été visités, la recherche de v (DFS(v)) est complète.

Si certains noeuds du graphe restent "non visités", sélectionner un comme nouveau noeud de départ et répéter le processus jusqu'à ce que tous les noeuds soient visités.

Nous obtenons ainsi l'algorithme suivant :

```
DFS(v):
  Mark(v) := "visité"
  Pour chaque noeud w adjacent à v:
        Si Mark(w)= "non visité"
              DFS(w)
        Fsi
  FP
```

Avec l'initialisation :

```
  Pour v=1, n
        Mark(v):="non visité"
  FP
```

Et comme appelant

```
  Pour v =1, n
        Si Mark(v)= "non visité": DFS(v)  Fsi
  FP
```

L'algorithme construit une forêt (implicite) de recouvrement des recherches.

On peut donner une version itérative du parcours en profondeur utilisant une pile:

```
DFS_iter( v )
  creer_pile( P );
  empiler( P , v );
  Tantque Non Pilevide( P )
        Depiler( P, v );
        Si Mark(v)= "non visité" :
              Mark(v) := "visité";
              Pour chaque w adjacent à v  // les prendre en ordre inverse
                    Si Mark(w) = "non visité" :
                          Empiler( P, w )
                    Fsi
              FP
        Fsi
  FTQ
```

b) Breadth first search : parcours en largeur

C'est la généralisation pour les graphes du parcours niveau par niveau d'un arbre.
Le principe est de visiter en premier les sommets les plus proche avant de s'éloigner du sommet initial.

Ce type de parcours est nécessaire quand on recherche un ou plusieurs noeuds dans des graphes infinis.

L'algorithme 'BFS(v)' est non récursif et utilise une file d'attente.

```
BFS(v)
 Mark(v) := "visité"
 CreerFile(F)
 Enfiler(F, v)
 Tantque Non Filevide(F) :
        Défiler(F, v)
        Pour chaque w adjacent à v
                Si Mark(w) = "non visité"
                        Mark(w):="visité"
                        Enfiler(F, w)
                Fsi
        FP
 FTQ
```

Avec l'initialisation :

```
  Pour v=1, n :
        Mark(v) := "non visité"
  FP
```

Et comme appelant

```
  Pour v =1, n
        Si Mark(v)= "non visité" :
                BFS(v)
        Fsi
  FP
```