Graphes

Introduction

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

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

Graphe orienté :

Ensemble de couples (u, v), u et v appartiennent à un ensemble de sommet(ou de nœuds). u est appelé la tête et v la queue.

A chaque couple (u, v) est associé un arc u-->v. Nous dirons que u est adjacent à v.

Un chemin est la séquence v1, v2, ...vn de sommets tels que v1->v2, v2-->v3, ...vn-1-->vn sont des arcs.

Longueur d’un chemin : nombre d'arcs qui le composent.

Un chemin simple : tous les nœuds, sauf éventuellement le premier et le dernier, sont distincts.

Graphe étiqueté.

Graphe orienté

Composante fortement connexe d'un graphe est composé de l'ensemble maximal de nœuds dans lequel il existe un chemin de tout nœud de l'ensemble vers chaque autre nœud de l'ensemble.

Formellement, soit G=(V, E) un graphe. On peut partitionner V en classes d'équivalence Vi : 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.

Graphe non orienté

Ensemble de paires ou d'arêtes (u, v). Chaque arête (u, v) représente en fait les deux arcs u-->v et v-->u.

On parlera de cycle au lieu de circuit, Chaîne au lieu de chemin. Graphe connexe, ect...

Arbre : Graphe non orienté connexe sans cycle.

Graphe orienté fortement connexe et sans circuit.

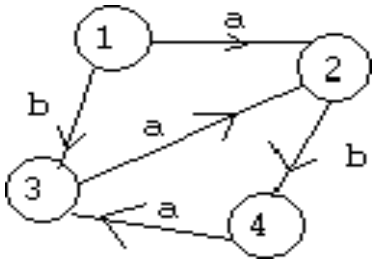
Représentation mémoire

1 . Matrice

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 i vers j. Si le graphe est étiqueté, A[i, j] représentera la valeur.

2 . Tableaux de listes linéaires chaînées

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



Graphe étiqueté

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

Matrice

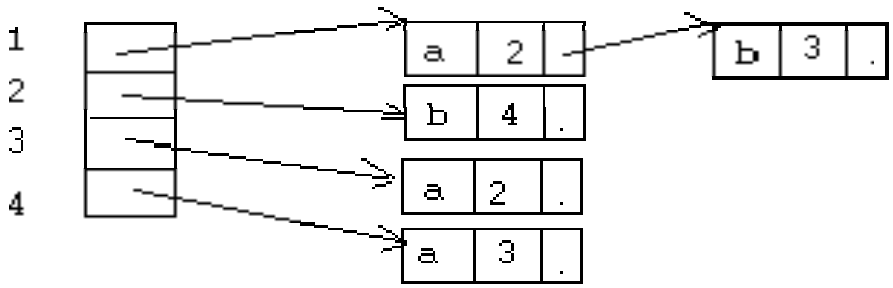


Tableau de listes

Modèle :

CréerNoeud(u, info)

LibérerNoeud(u)

CréerArc(u, v, info)

SupprimerArc(u, v)

Valeur(u) : valeur ratachéau noeud u

Arc(u, v) : valeur ratachée à l'arc(u, v)

Adjacent(u, i) : ièmenoeud adjacent à u.

Parcours des Graphes

1- **Depht first search**

parcours très utilisé sur les graphes : Recherche en profondeur d'abord".

C'est la généralisation du parcours Préordre sur les arbres.

Parcours d'un graphe : Depht first search

- **Principe** :

— Initialement tous les nœuds 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 nœuds soient visités.

```
Algorithme     Dfs(v):
 Début
  Mark(v) := " visité"
  Pour chaque noeud w adjac à v
    Faire
        Si Mark(w)= " non visité"
                Alors Dfs(w)  Fsi
    Fait
 Fin
```

Initialisation :

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

Et comme Algorithme principale appelant

```
Début
        Pour v =1, n
           faire
              Si Mark(v) non visité:
                    alors    dfs(v)
              Fsi
           Fait
Fin.
```

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

Parcours d'un graphe :Breadh first search

• **Algorithme BFS (v)** (non récursif et utilise une file d'attente)

```
Début
  Mark(v) := "visité" ;   CreerFile(F) ; Enfiler(F, v) ;
  Tantque Non Filevide(F)
   faire
       Défiler(F, v)
       << Traitement sur le nœud v>>
       Pour tout w adjacent à v
         faire
            Si Mark(w) = non-visité
                       alors
                            Mark(w):="visité" ;
                            Enfiler(F, w) ;
            Fsi
         Fait ;
    Fait ;
 Fin.
```

Initialisation :

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

Et comme Algorithme principale appelant

```
Début
       Pour v =1, n
          faire
             Si Mark(v) non visité:
                    alors    BFS(v)
             Fsi
          Fait
Fin.
```

Exemples d'application

Application 1 : Détermination du plus court chemin (Algorithme de Dijkstra)

Soit G = (V, E) un graphe où V est l'ensemble des nœuds et E l'ensemble des arcs. Chaque arc a une valeur positive. Soit S un nœud quelconque.

Le problème consiste à déterminer le coût du plus court chemin de S vers tous les autres nœuds de G, où la longueur d'un chemin désigne la somme des coûts des arcs sur ce chemin.

G peut être par exemple, une carte de vols aériens, dans laquelle les nœuds représentent les villes et chaque arc u-->v représente la ligne aérienne de u vers v. La valeur de l'arc est donc le temps de vol de u vers v.

Algorithme de Dijkstra

Soient V = { 1, 2, ..., n} et S = 1.

C[1..n, 1..n] tableau de coûts tel que C[i, j] est le coût de l'arc i-->j sinon c'est l'infini.

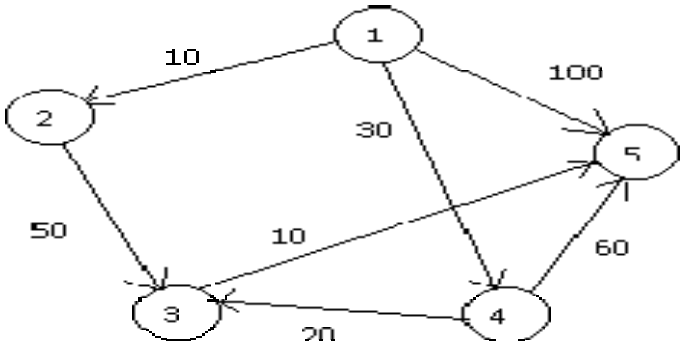
Le tableau D contiendra après exécution, la plus courte distance de S vers tous les autres nœuds.

S = {1}

```
Début
   { initialisation des distances de S vers tous les autres nœuds }
   pour i :=2 à n
         D[i] := C[1, i]
        Fin_pour

   Pour i :=1 à  n-1
     faire
       - Choisir un nœud w dans V-S tel que D[w] est minimale.
      - Ajouter w à S
      - Pour chaque nœud v dans V-S
            faire
               D[v] := Min (D[v], D[w]+C[w,v] ) ;
             Fait ;
     Fait ;
Fin.
```

Algorithme de Dijkstra**(Exemple)**



Graphes

Algorithme de Djikstra(Trace)

S	{1}	{1,2}	{1,2,4}	{1,2,4,3}	{1 ,2,4,3, 5}
w	-	2	4	3	5
D[2]	10	10	10	10	10
D[3]	&	60	50	50	50
D[4]	30	30	30	30	30
D[5]	100	100	90	60	60

Application 2 : Problème du Voyageur du Commerce (PVC)

Un cycle Hamiltolnien dans un graphe non orienté est un chemin incluant tous les nœuds du graphe.

Le PVC consiste à trouver dans un graphe étiqueté non orienté un cycle Hamiltolnien de poids minimal.