PC 7
Graphes

1. représentations
2. matrice d'adjacence
3. fermeture transitive
4. exploration
5. plus court chemin

Introduction

Un graphe (S, F) est donné par

- Un ensemble S de sommets
- Un ensemble $F \subset S \times S$ de *flèches* (ou arcs).

Un *chemin* dans G est une suite

$$(s_0, s_1), (s_1, s_2), \ldots, (s_{n-1}, s_n)$$

Variantes:

- On a aussi en général des *étiquettes* sur les flèches ou sur les sommets.
- Graphes non orientés.
- Graphes représentés (dans le plan par ex.)
- Multigraphes.

Exemples de graphes

Les graphes se rencontrent comme:

1. Modèle de liaisons spatiales: réseaux de transport, électriques, ...
2. Représentations de liens logiques: graphes de dépendance.
3. Modèles de l'évolution temporelle: les sommets sont des *états* d'un système.
4. Schémas de programmes: les sommets sont les instructions.

Exemples

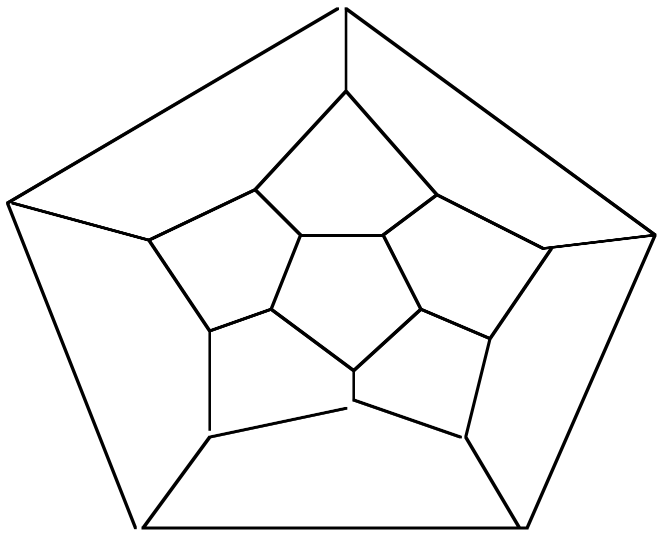

Figure 1: dodecaèdre

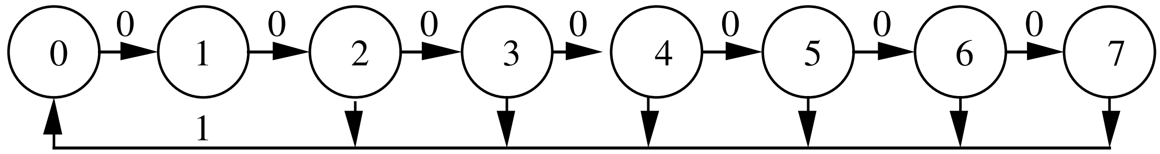

Figure 2: canal contraint

Représentation des graphes

1. Matrice d'adjacence:

$$A_{ij} = \begin{cases} 1 & \text{si } (i,j) \in F \\ 0 & \text{sinon} \end{cases}$$

   Les éléments peuvent être considérés comme des booléens ou comme des entiers.

2. Liste de successeurs: On donne pour chaque sommet la liste de ses successeurs:

$$L(s) = \{t \in S \mid (s,t) \in S\}$$

3. Matrice d'incidence: On donne pour chaque flèche son origine et son extremité:

$$I_{if} = \begin{cases} 1 & \text{si } f = (i,.) \\ -1 & \text{si } f = (.,i) \\ 0 & \text{sinon} \end{cases}$$

Exemple

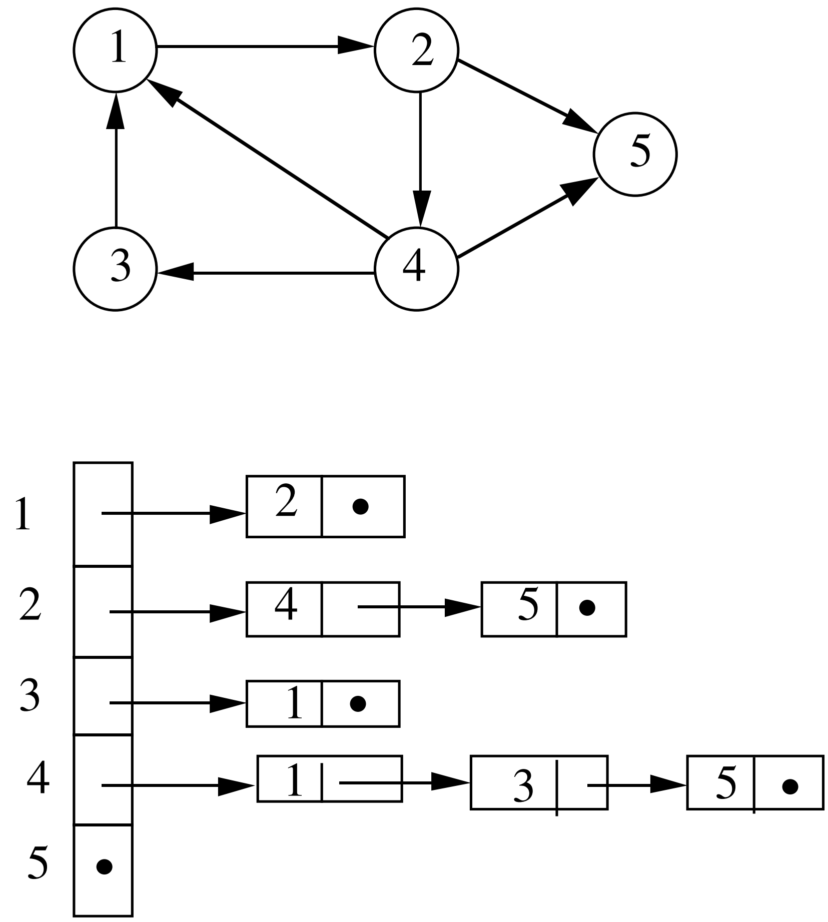

Figure 3:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fermeture transitive

On se donne un graphe $G = (S, F)$ par sa matrice d'adjacence A.

On veut calculer la *fermeture transitive* de G, i.e. le graphe $G^* = (S, F^*)$ défini par $(s,t) \in F^*$ ssi

il existe un chemin de s à t dans G

Propriété fondamentale de la matrice d'adjacence:

$$\boxed{A^k_{ij}\text{=nb de chemins de longueur } k \text{ de } i \text{ à } j}$$

ou en booléens

$$\boxed{\begin{array}{l} A^k_{ij} = 1 \Leftrightarrow \exists \ \text{ chemin de longueur } k \\ \qquad\qquad\qquad\qquad \text{de } i \text{ à } j \end{array}}$$

Formule donnant la matrice d'adjacence A^* de G^*(en booléens):

$$A^* = I + A + A^2 + \ldots + A^{n-1}$$

puisqu'un chemin de i à j peut être choisi de longueur au plus $n-1$.

$\Rightarrow$ Algorithme naïf en $O(n^4)$.

Exemple

Si

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Alors

$$A^2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad A^3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

et donc

$$A^* = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

L'algorithme de Warshall

Idée: au lieu de faire porter la récurrence sur la longueur du chemin, on peut la faire porter sur le nombre de sommets utilisés. Posons:

$$A_{ij}^{(k)} = \begin{cases} 1 & \text{s'il existe un chemin de } i \text{ à } j \\ & \text{ne passant par aucun} \\ & \text{sommet} > k \text{ entre temps} \\ 0 & \text{sinon} \end{cases}$$

L'algorithme est basé sur la formule de récurrence (en booléens):

$$A_{ij}^{(k)} = A_{ij}^{(k-1)} + A_{ik}^{(k-1)} A_{kj}^{(k-1)}$$

avec $A_{ij}^{(0)} = I_{ij} + A_{ij}$.

Calcul sur place dans une matrice T:

```
void Warshall(Matrice T) {
  int k,i,j;
  for (k= 0; k< n; ++k)
    for (i= 0; i< n; ++i)
      if (T[i][k]== 1)
        for (j= 0; j< n; ++j)
          if(T[k][j]== 1)
            T[i][j]= 1;
}
```

La complexité devient $O(n^3)$. C'est un exemple où on peut améliorer la complexité sans rien perdre.

Plus court chemin

On se donne sur chaque flèche(i,j) une *longueur* $l(i,j)$ positive. On cherche à déterminer le plus court chemin d'un sommet à un autre.

On représente le graphe par une matrice C définie par

$$C_{ij} = \begin{cases} l(i,j) & \text{si } (i,j) \in F \\ 0 & \text{si } i = j \\ \infty & \text{sinon} \end{cases}$$

On pose ensuite $C_{ij}^{(k)}$ = la longueur du plus court chemin de i à j utilisant au plus k flèches.Du fait que les longueurs sont positives, on a la relation de récurrence:

$$\boxed{C_{ij}^{(k+1)} = \min \begin{cases} C_{i,j}^{(k)} \\ \min \begin{array}{l} \{C_{i,u}^{(k)} + C_{u,j} \\ \mid 1 \leq u \leq n\} \end{array} \end{cases}}$$

Ceci conduit à un algorithme en $O(n^4)$.

Exemple

Si

$$C = \begin{bmatrix} 0 & 1 & \infty & \infty & \infty \\ \infty & 0 & \infty & 1 & 3 \\ 4 & \infty & 0 & \infty & \infty \\ 5 & \infty & 8 & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

La matrice des plus courts chemins est

$$D = \begin{bmatrix} 0 & 1 & 10 & 2 & 3 \\ 6 & 0 & 9 & 1 & 2 \\ 4 & 5 & 0 & 6 & 7 \\ 5 & 6 & 8 & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

et

$$P = \begin{bmatrix} - & - & 4 & 2 & 4 \\ 4 & - & 4 & - & 4 \\ - & 1 & - & 2 & 2 \\ - & 1 & - & - & - \\ - & - & - & - & - \end{bmatrix}$$

L'algorithme de Floyd

On considère cette fois $C_k[i,j]$ = la longueur du plus court chemin de i à j ne passant par aucun sommet $> k$. On a la relation de récurrence:

$$\boxed{C_{k+1}[i,j] = \min \begin{cases} C_k[i,j] \\ C_k[i,k+1] + C_k[k+1,j] \end{cases}}$$

d'où le programme:

```
void Floyd(Matrice A) {
   int k,i,j;
  for (k= 0; k< n; ++k)
    for (i= 0; i<n; ++i)
      for (j= 0; j<n; ++j)
        if (A[i][k]+A[k][j] < A[i][j])
        {
          A[i][j]= A[i][k]+A[k][j];
          P[i][j]= k;
        }
}
```

Le tableau P sert à retrouver le plus court chemin. Il est initialisé à -1.

```
void Chemin (int i, int j) {
  int k;
  k= P[i][j];
  if (k != -1)
  {
     Chemin(i,k);
     printf("%d",k);
     Chemin(k,j);
   }
}
```