

## TC Informatique

---

PC N° 8  
11 Janvier 2001

François Sillion

## Graphes

<http://w3.edu.polytechnique.fr/informatique>

## Contrôle HC

---

- Expliquer le principe de l'algorithme
- Commenter
- Justifier les points « dangereux » (écriture de `if ( x.suivant.suivant.val == 0 ) ...`)
- Utilisation ou non de tableaux annexes
- Notions de complexité

## Graphes

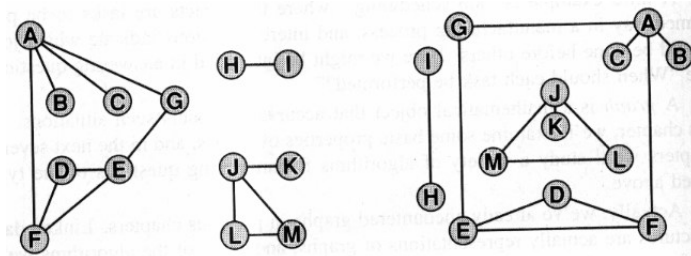
---

- $G = (S, A)$  est donné par
  - un ensemble  $S$  de *sommets*
  - Un sous-ensemble  $A$  de  $S \times S$ , les *arcs* de  $G$
- Le graphe peut être *orienté*, auquel cas les arcs ont une *origine* et une *extrémité*
- Un *chemin* est une suite d'arêtes telle que l'origine d'un arc est l'extrémité du précédent, et l'extrémité d'un arc est l'origine du suivant.
- Un *circuit* est un chemin tel que l'extrémité du dernier arc est l'origine du premier.
- Un graphe sans circuit est analogue à un arbre (avec des racines multiples, et des branches partagées...)

## Graphes

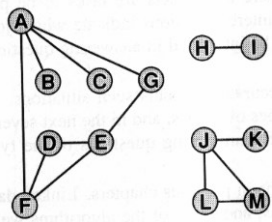
---

- Relations entre objets
- Géométrie
- Processus de fabrication (dépendence)
- Représentation graphique :



## Représentation d'un graphe

- Matrice d'adjacence



	A	B	C	D	E	F	G	H	I	J	K	L	M
A	1	1	1	0	0	1	1	0	0	0	0	0	0
B	1	1	0	0	0	0	0	0	0	0	0	0	0
C	1	0	1	0	0	0	0	0	0	0	0	0	0
D	0	0	0	1	1	1	0	0	0	0	0	0	0
E	0	0	0	1	1	1	1	0	0	0	0	0	0
F	1	0	0	1	1	1	0	0	0	0	0	0	0
G	1	0	0	0	1	0	1	0	0	0	0	0	0
H	0	0	0	0	0	0	0	1	1	0	0	0	0
I	0	0	0	0	0	0	0	1	1	0	0	0	0
J	0	0	0	0	0	0	0	0	0	1	1	1	1
K	0	0	0	0	0	0	0	0	0	1	1	0	0
L	0	0	0	0	0	0	0	0	0	1	0	1	1
M	0	0	0	0	0	0	0	0	0	1	0	1	1

Figure 29.3 Adjacency matrix representation.

```
class GrapheMA {
    int taille;
    boolean m[][];

    GrapheMA( int n ) {
        taille = n;
        m = new boolean[n][n];
    }
}
```

## Accessibilité

- Existe-t-il un chemin entre deux sommets ?
- Soit un graphe G avec une matrice d'adjacence M
- $M^2$  est la matrice d'adjacence des chemins de longueur 2
- $M^n$  correspond aux chemins de longueur n
- $\sum_{k=0}^n M^k$  correspond aux chemins de toute longueur
- Le graphe correspondant est la *fermeture transitive* de G

## Calcul de la fermeture transitive

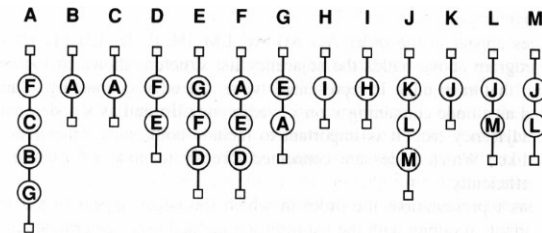
- Somme des itérées de la matrice
- Mieux : itération de l'opération de base  $x$  qui ajoute des arcs  $(y,z)$  tels que  $(y,x)$  et  $(x,z)$  existent dans  $G$ .

```
Static void FermetureTransitive ( GrapheMA g ) {
    for ( int k=0; k < g.taille; ++k ) {
        for ( int i=0; i < g.taille; ++i ) {
            if ( g.m[i][k] ) {
                for ( int j=0; j < g.taille; ++j ) {
                    if ( g.m[k][j] ) {
                        g.m[i][j] = true;
                    }
                }
            }
        }
    }
}
```

- Pourquoi est-ce que ça marche ?
- Complexité ?

## Représentation alternative

- Tableaux ou listes d'adjacence



```
Class ListeSuccesseur {
    int sommet;
    ListeSuccesseur suite;
    ListeSuccesseur( int n, ListeSuccesseur s ){
        sommet = n;
        suite = s;
    }
}
```

## Changement de représentation

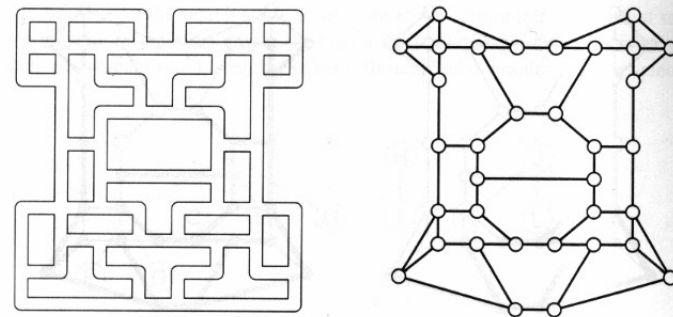
```
Class GrapheLS {
    int taille;
    ListeSuccesseurs[] successeurs;
    GrapheLS( int n ) {
        taille = n;
        successeurs = new ListeSuccesseurs[n];
    }
}
```

- **Matrice vers liste de successeurs**

```
static GrapheLS MAversLS( GrapheMA ma ) {
    GrapheLS ls = new GrapheLS( ma.taille );
    for ( int i=0; i < ma.taille ; ++i ) {
        for ( int j=0; j < ma.taille ; ++j ) {
            if ( ma.m[i][j] )
                ls.successeurs[i] =
                    new ListeSuccesseurs( j,
                                            ls.successeurs[j] );
        }
    }
    return ls;
}
```

## Parcours de graphe

- Visite systématique de tous les nœuds
- Profondeur d'abord
- Largeur d'abord
- Plus court chemin...



**Figure 29.15** A maze and an associated graph.

## Parcours en profondeur d'abord

- Analogue au parcours d'arbre
- Il faut savoir si un nœud a déjà été visité

```
...
boolean[] Dejavu;
...
static void parcoursComplet( GrapheLS g )
{
    Dejavu = new boolean[g.taille];
    for ( int i=0; i < g.taille; ++i )
        Dejavu[i] = false;
    for ( int k=0; k < g.taille ; k++ )
        if ( ! Dejavu[k] ) visiteRecursive( g, k );
}
static void visiteRecursive( GrapheLS g, int n )
{
    Dejavu[n] = true;
    ListeSuccesseurs l = g.successeurs[n];
    while ( l != null ) {
        if ( ! Dejavu[l.sommet] )
            visiteRecursive( g , l.sommet );
        l = l.suite;
    }
}
```

PC 8

François Sillion

11

## Parcours en profondeur d'abord (2)

- Version non récursive
- Modification du sens du tableau Dejavu...

```
static void traverseProfondeur( GrapheLS g, int n )
{
    int[] Dejavu = new int[g.taille];
    int numero = 0;
    for ( int i=0; i < g.taille; ++i )
        Dejavu[i] = 0;
    Pile f = CreePile();
    Empile( f , n );

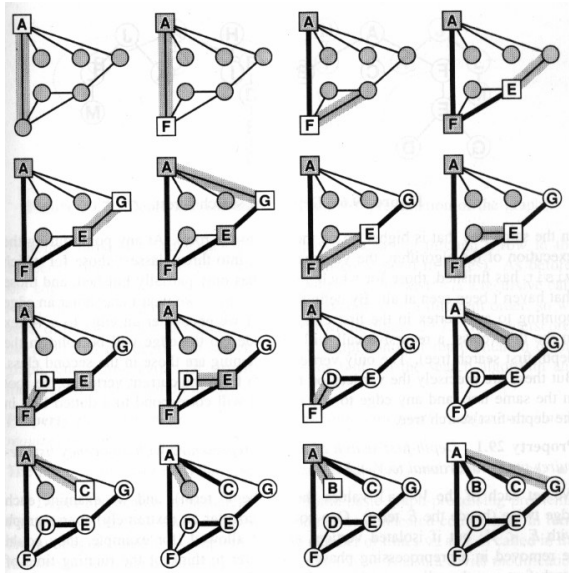
    while ( ! PileVide( f ) ) {
        n = Depile(f);
        visite( n );
        Dejavu[n] = ++numero;
        ListeSuccesseurs l = g.successeurs[n];
        while ( l != null ) {
            if ( Dejavu[l.sommet] == 0 ) {
                Empile( f , l.sommet );
                Dejavu[l.sommet] = -1;
            }
            l = l.suite;
        }
    }
}
```

PC 8

François Sillion

12

## Visualisation parcours Prof. d'abord



PC 8

François Sillion

13

## Parcours en largeur d'abord

- Analogue au parcours d'arbre
- Avec une file...

```
static void traverseLargeur( GrapheLS g, int n )
{
    int[] DejaVu = new int[g.taille];
    for ( int i=0; i < g.taille; ++i ) DejaVu[i] = -1;
    int numero = 0;
    File f = CreeFile();
    Ajoute( f , n );
    while ( ! FileVide( f ) ) {
        n = Retire(f);
        visite( n );
        DejaVu[n] = ++numero;
        ListeSuccesseurs l = g.successeurs[n];
        while ( l != null ) {
            if ( DejaVu[l.sommet] == 0 ) {
                Ajoute( f , l.sommet );
                DejaVu[l.sommet] = -1;
            }
            l = l.suite;
        }
    }
}
```

PC 8

François Sillion

14

## Profondeur et largeur...

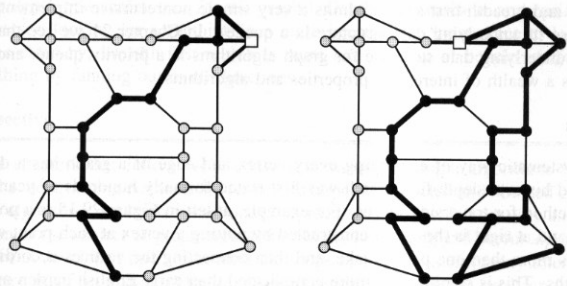


Figure 29.13 Depth-first search in a larger graph.

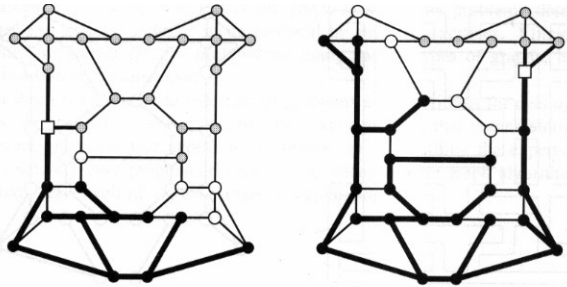


Figure 29.14 Breadth-first search in a larger graph.

## Plus court chemin

- Repart du parcours en largeur d'abord
- Utilise le tableau annexe pour marquer les distances à l'«origine»
- Invariant : pour chaque élément  $k$  dans la file,  $DejaVu[k]$  contient la distance de l'élément à partir duquel on a placé  $k$  sur la pile...

## Plus court chemin (suite)

---

```
static void MinDistance( GrapheLS g, int n )
{
    int[] DejaVu = new int[g.taille];
    for ( int i=0; i < g.taille; ++i ) DejaVu[i] = -1;
    File f = CreeFile();
    Ajoute( f , n );
    DejaVu[n] = 0;
    while ( ! FileVide( f ) ) {
        n = Retire(f);
        visite( n );
        DejaVu[n] = -DejaVu[n] + 1;
        ListeSuccesseurs l = g.successeurs[n];
        while ( l != null ) {
            if ( DejaVu[l.sommet] == 0 ) {
                Ajoute( f , l.sommet );
                DejaVu[l.sommet] = -DejaVu[n];
            }
            l = l.suite;
        }
    }
}
```