# Structures de données


**Pr. AIT LAHCEN**

Automne 2017

# Les graphes

# Plan de la séance

**Introduction**

- Définitions
- Graphes non orientés (ou symétriques)
- Graphes orientés
- Graphes orientés ou non orientés

**Exemples de graphes**

- Un réseau de communication
- Ordonnancement de tâches (graphe orienté sans cycle)
- Programme informatique

**Parcours d'un graphe**

- Description d'un graphe
- Parcours en profondeur d'un graphe
- Parcours en largeur d'un graphe

**Représentation d'un graphe**

- Représentation simple sous forme de matrices
- Représentation plus complète
- Implémentation

# Introduction

# Introduction

## Définitions

Un graphe est une structure de données composée d'un ensemble de sommets, et d'un ensemble de relations entre ces sommets.

Si la relation n'est pas orientée, la relation est supposée exister dans les deux sens. Le graphe est dit non orienté ou symétrique.

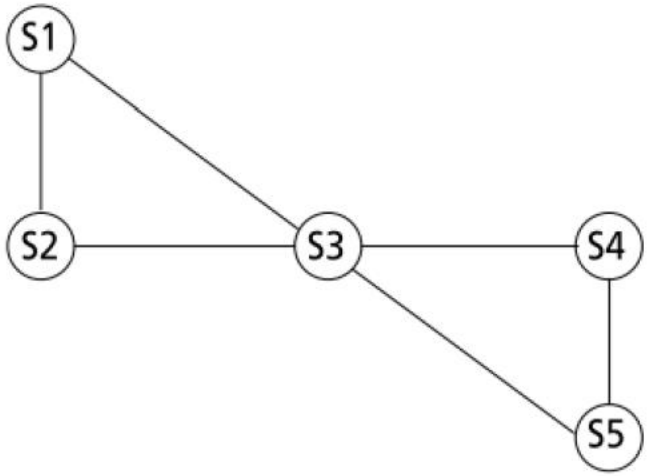
Dans le cas contraire, si les relations sont orientées, le graphe est dit orienté.

Une relation est appelée un arc (quelquefois une arête pour les graphes non orientés). Les sommets sont aussi appelés nœuds ou points.

## Graphes non orientés (ou symétriques)

Le graphe de la Figure suivante peut se noter comme suit :

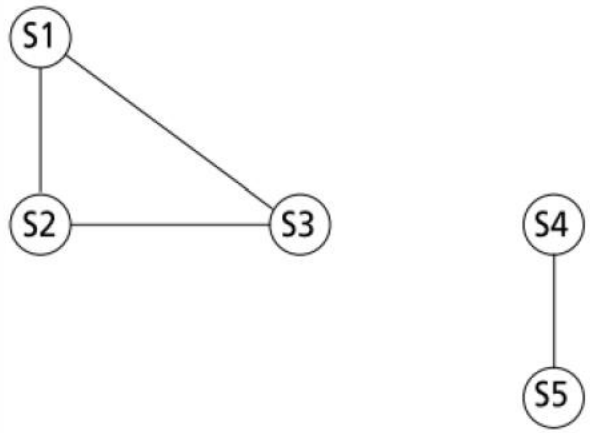
- S = {S1, S2, S3, S4, S5} ; ensemble des sommets
- A = {S1S2, S1S3, S2S3, S3S4, S3S5, S4S5} ; ensemble des relations symétriques. Par exemple, S1S2 est équivalente à S2S1.



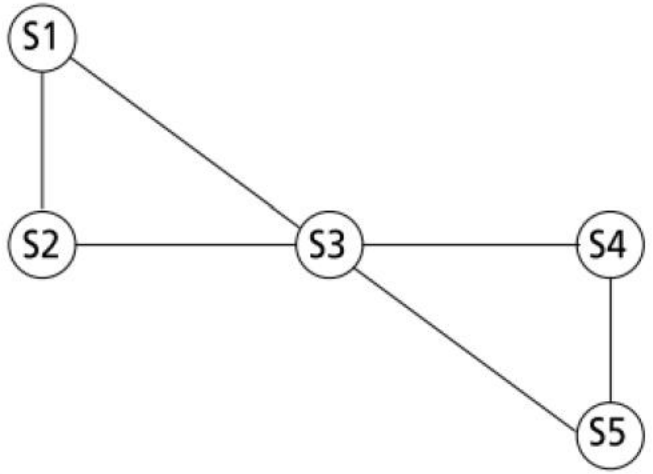
# Introduction

## Graphes non orientés (ou symétriques)

**Graphe connexe :** un graphe non orienté est dit connexe si on peut aller de tout sommet vers tous les autres sommets.



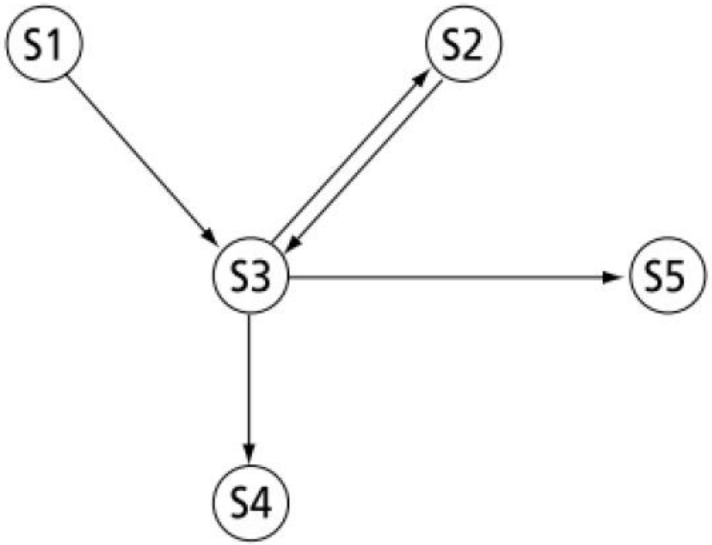

Le graphe n'est pas connexe




Le graphe est connexe

## Graphes orientés

Si les relations sont orientées, le graphe est dit orienté.



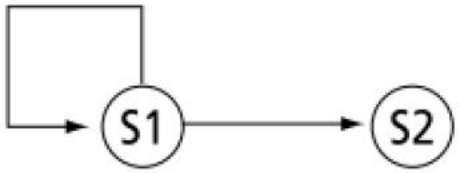

**Degrés d'un graphe orienté**

d°(S3) = 5 degré du sommet S3 : nombre d'arcs entrants ou sortants
d+(S3) = 3 nombre d'arcs sortants : demi-degré extérieur ou degré d'émission
d-(S3) = 2 nombre d'arcs entrants : demi-degré intérieur ou degré de réception

## Graphes orientés ou non orientés

Les définitions suivantes s'appliquent aux graphes orientés comme aux graphes non orientés.

**Une autoboucle** est une relation (Si, Si) :

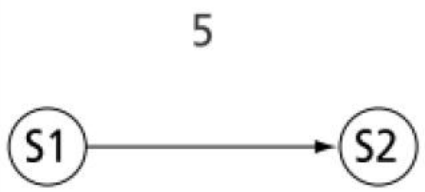



**Un multigraphe ou graphe multiple** est un graphe tel qu'il existe plusieurs arcs entre certains sommets :

## Graphes orientés ou non orientés

**Un graphe simple** est un graphe sans boucle et sans arc multiple.

**Un graphe est dit valué** (pondéré) si à chaque arc on associe une valeur représentant le coût de la transition de cet arc :




**Un chemin** dans un graphe est une suite d'arcs consécutifs.

**La longueur d'un chemin** est le nombre d'arcs constituant ce chemin.

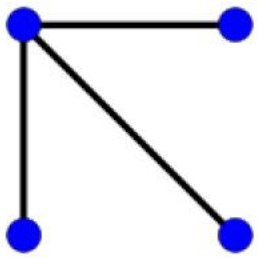
**Un chemin simple** est un chemin où aucun arc n'est utilisé 2 fois.

## Graphes orientés ou non orientés

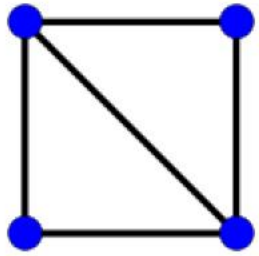
**Un circuit simple** est un chemin simple tel que le premier et le dernier sommet sont les mêmes.

**Un circuit hamiltonien** est un circuit qui passe une et une seule fois par tous les sommets.

**Un graphe hamiltonien** est un graphe possédant au moins un cycle passant par tous les sommets une fois et une seule.



Graphe non-hamiltonien

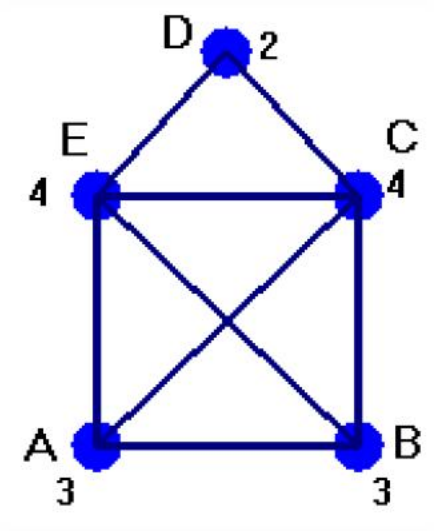


Graphe hamiltonien

## Graphes orientés ou non orientés

**Un circuit eulérien** est un circuit qui passe une et une seule fois par tous les arcs.

**Un graphe est eulérien** si et seulement si il admet un circuit eulérien.

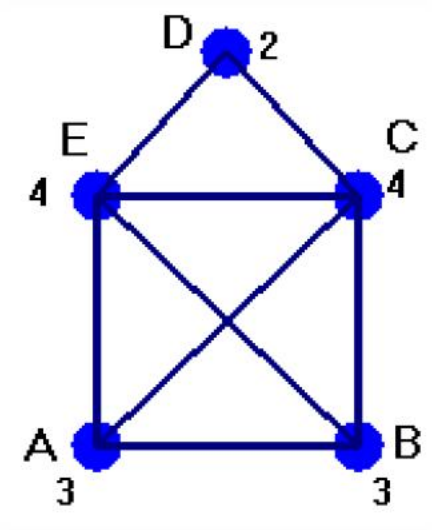


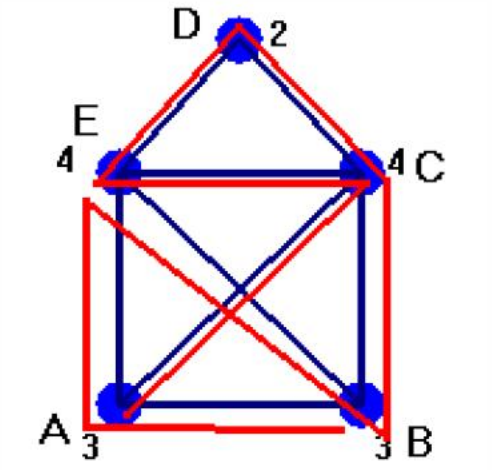

Est-il possible de dessiner le graphe sans lever le crayon et sans passer deux fois sur le même trait ?

## Graphes orientés ou non orientés

**Un circuit eulérien** est un circuit qui passe une et une seule fois par tous les arcs.

**Un graphe est eulérien** si et seulement si il admet un circuit eulérien.



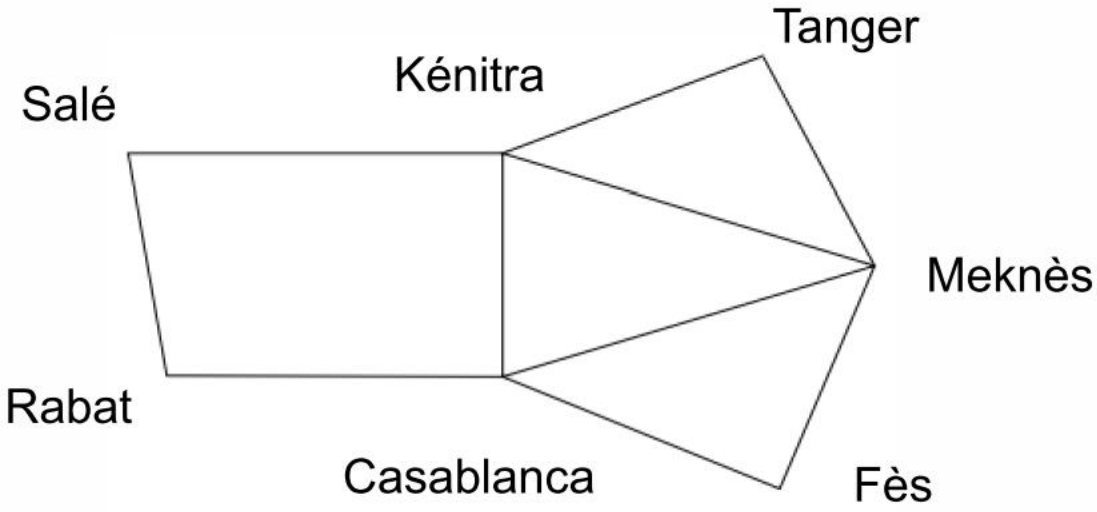



# Exemples de graphes

# Exemples de graphes

## Un réseau de communication

Un réseau de communication (routier, aérien, électrique, d'alimentation en eau, etc.) entre différents lieux peut être schématisé sous forme d'un graphe comme l'indique la figure suivante.



# Parcours d'un graphe

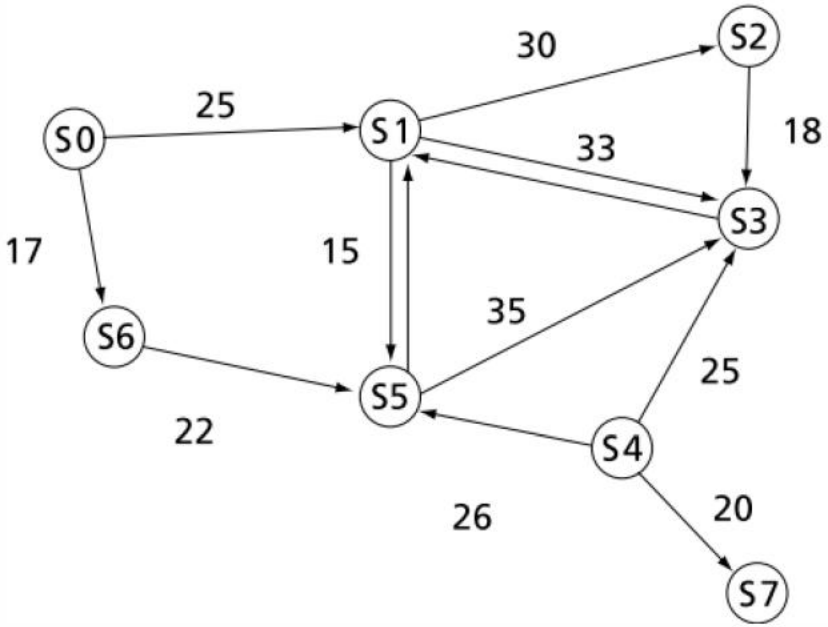
# Parcours d'un graphe

## Description d'un graphe

**Le graphe de la figure peut être décrit comme suit :**

S0 S1 S2 S3 S4 S5 S6 S7 ; liste des sommets

S0: S1 (25) S6 (17) ;
S1: S2 (30) S3 (33) S5 (15) ;
S2: S3 (18) ;
S3: S1 (33) ;
S4: S3 (25) S5 (26) S7 (20) ;
S5: S1 (15) S3 (35) ;
S6: S5 (22) ;



# Parcours d'un graphe

## Parcours en profondeur d'un graphe

On part d'un sommet donné. On énumère le premier fils de ce sommet (par ordre alphabétique par exemple), puis on repart de ce dernier sommet pour atteindre le premier petit-fils, etc.

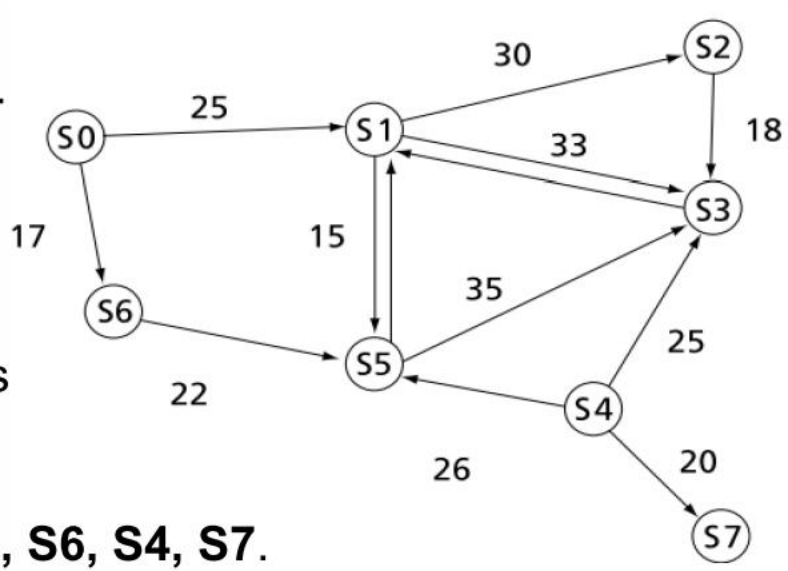
Il s'agit pour chaque sommet visité, de choisir un des sommets successeurs du sommet en cours, jusqu'à arriver sur une impasse ou un sommet déjà visité.

Dans ce cas, on revient en arrière pour repartir avec un des successeurs non visité du sommet courant.

# Parcours d'un graphe

## Parcours en profondeur d'un graphe

En partant de S0, on peut aller en S1 ou S6.
On choisit S1. De S1, on peut aller en S2, S3 ou S5.
On choisit S2. De S2, on peut aller en S3.
De S3, on pourrait aller en S1 mais S1 a déjà été marqué.
On revient en arrière sur S2 où il n'y a pas d'autre alternative.
On revient en arrière sur S1 ; reste à essayer S3 et S5.
S3 a déjà été visité. On prend donc le chemin S5.
De S5, on ne peut explorer de nouveaux sommets.
On revient en S1, puis S0.
Pour S0, on peut aller en S6.



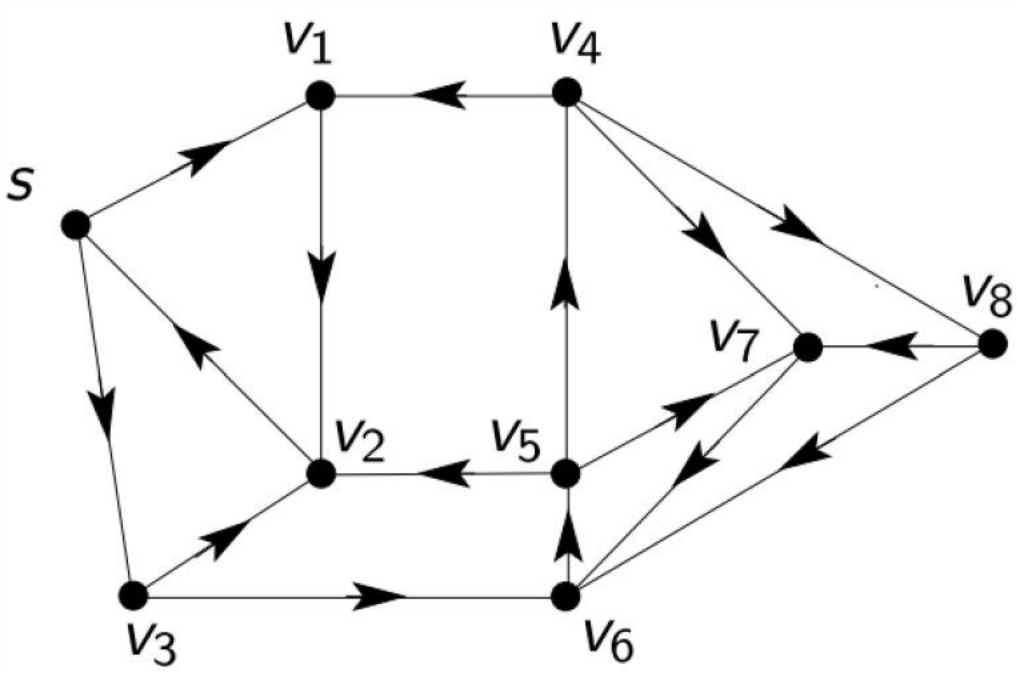

Tous les sommets n'ont pas été visités de S0.
Il faut repartir d'un des sommets non encore visités
On repart de S4 qui mène à S7.

Le parcours en profondeur est : **S0, S1, S2, S3, S5, S6, S4, S7**.

# Parcours d'un graphe

## Parcours en profondeur d'un graphe

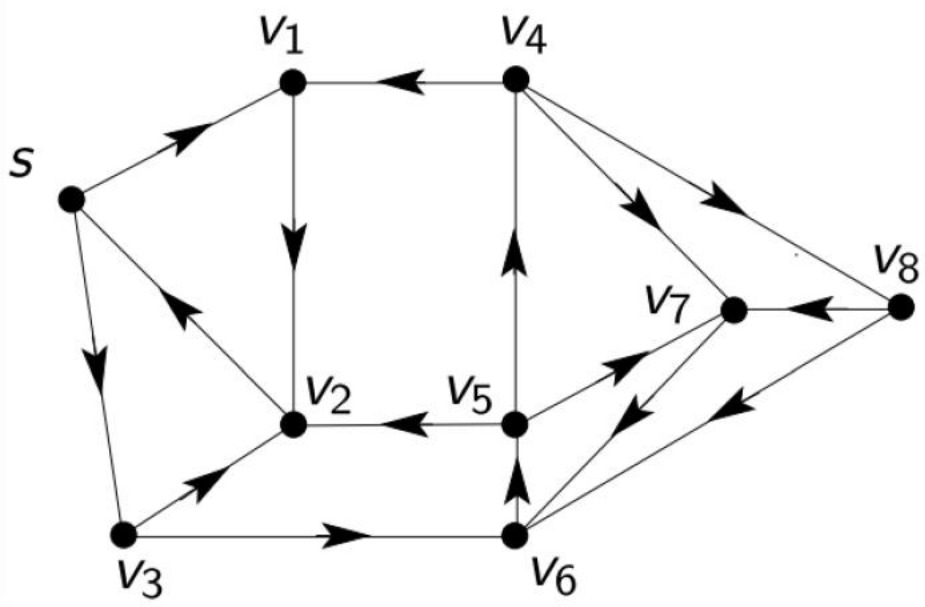
**Exercice rapide** : Donner le parcours en profondeur à partir de s du graphe suivant :

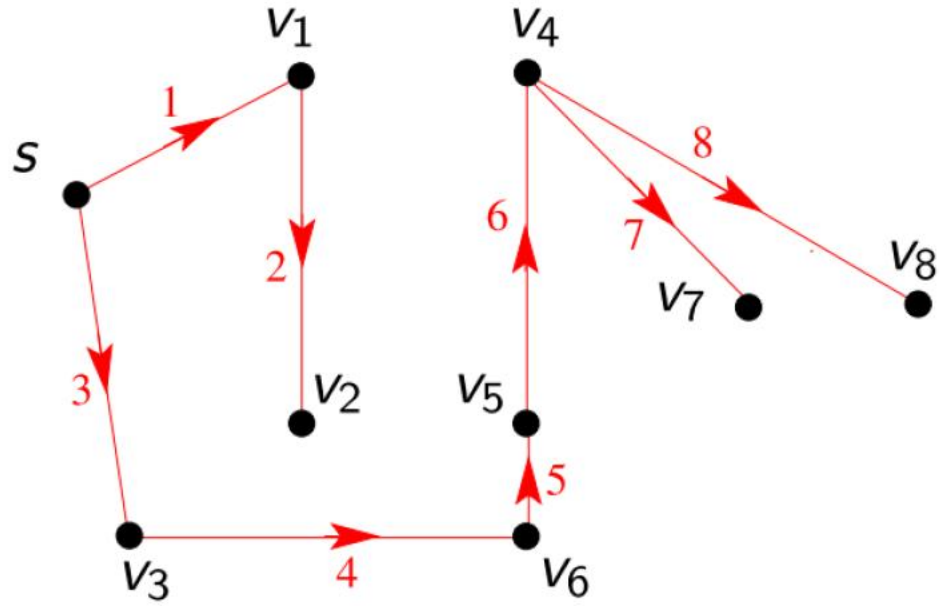


# Parcours d'un graphe

## Parcours en profondeur d'un graphe

**Exercice rapide** : Donner le parcours en profondeur à partir de s du graphe suivant :



# Parcours d'un graphe

## Parcours en largeur d'un graphe

On part d'un sommet donné. On énumère tous les fils (les suivants) de ce sommet, puis tous les petits-fils non encore énumérés, etc.

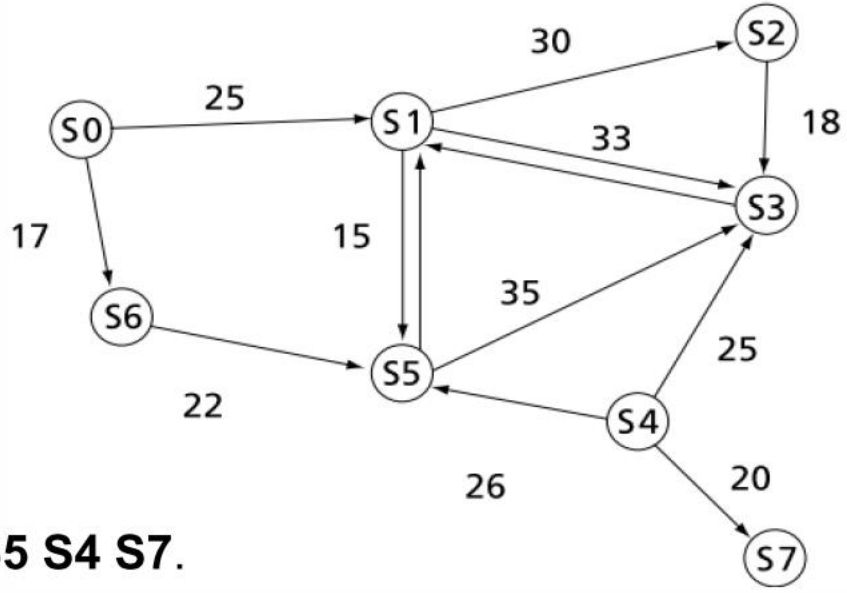
C'est une énumération par génération :

- les successeurs directs,
- puis les successeurs au 2ème degré,
- etc.

# Parcours d'un graphe

## Parcours en largeur d'un graphe

En partant de S0, on visite S1 et S6.
De S1, on visite S2, S3 et S5.
De S6, on ne peut pas explorer de nouveaux sommets.
De S2, S3 et S5, on ne peut pas explorer de nouveaux sommets.
Il faut également repartir d'un sommet non encore visité et non accessible de S0.
On repart avec S4 qui nous conduit à S7.

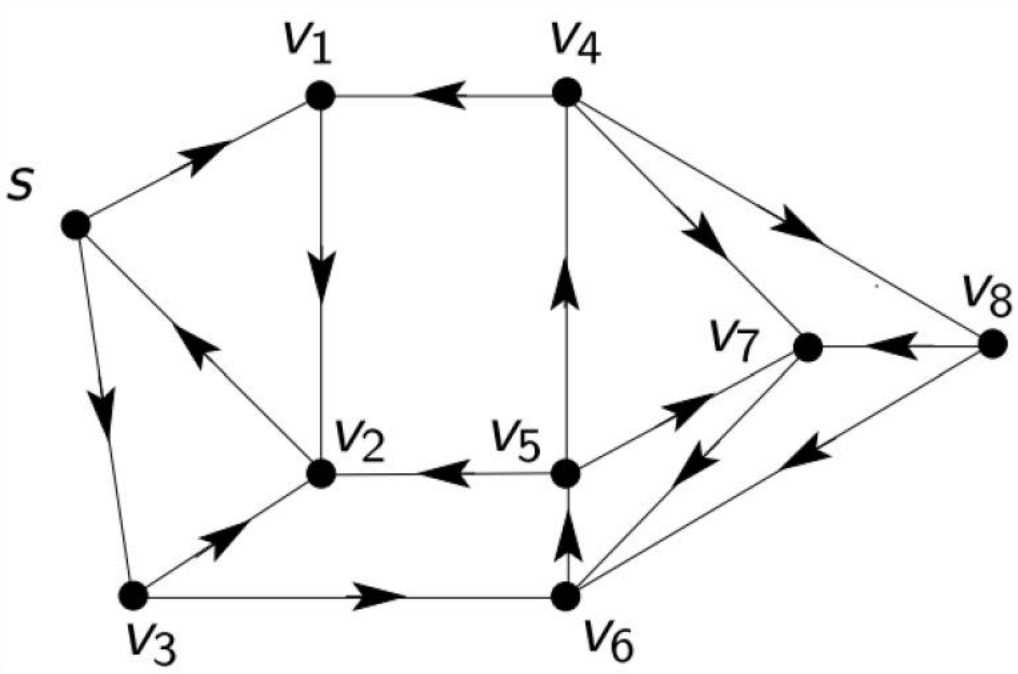



Le parcours en largeur est : **S0 S1 S6 S2 S3 S5 S4 S7**.

# Parcours d'un graphe

## Parcours en largeur d'un graphe

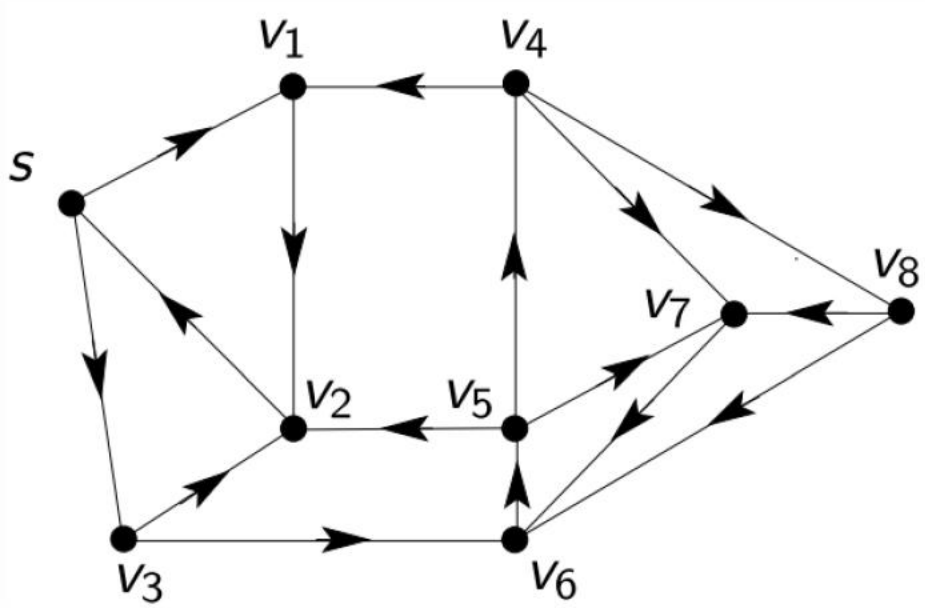
**Exercice rapide** : Donner le parcours en largeur à partir de s du graphe suivant :

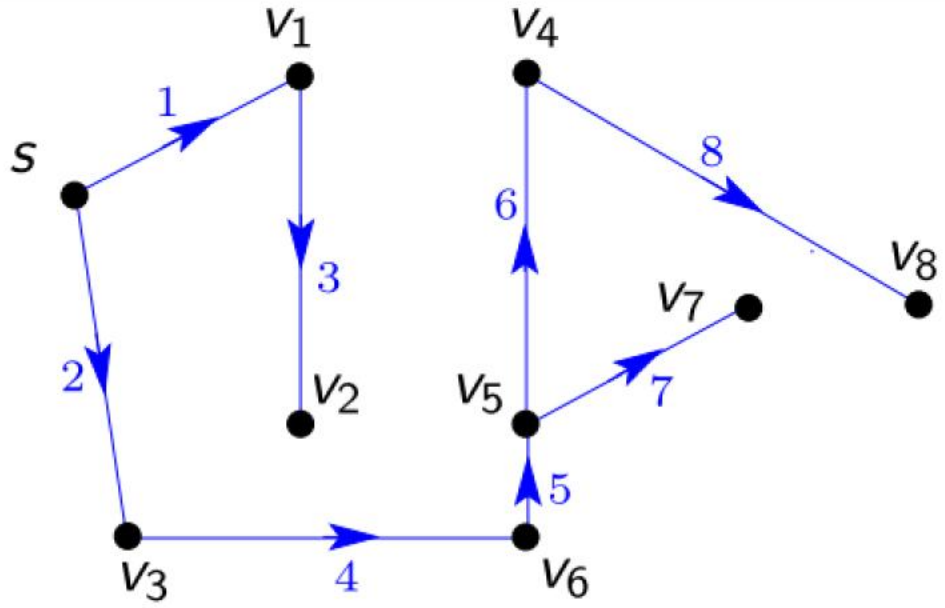


# Parcours d'un graphe

## Parcours en largeur d'un graphe

**Exercice rapide** : Donner le parcours en profondeur à partir de s du graphe suivant :



# Représentation d'un graphe

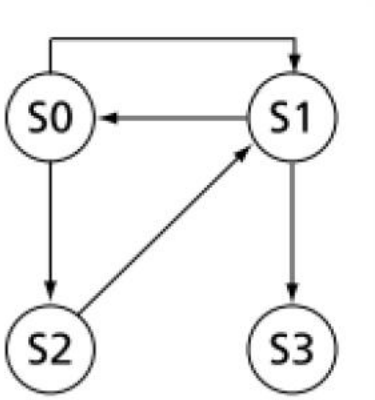
# Représentation d'un graphe

## Représentation simple sous forme de matrices

Chaque arc (i, j) est représenté par un booléen V (vrai) dans la matrice.

Une valeur F (faux) indique une absence de relation entre le sommet i et le sommet j.

Un tableau en plus de la matrice est utilisé pour contenir les noms des sommets et leurs caractéristiques.

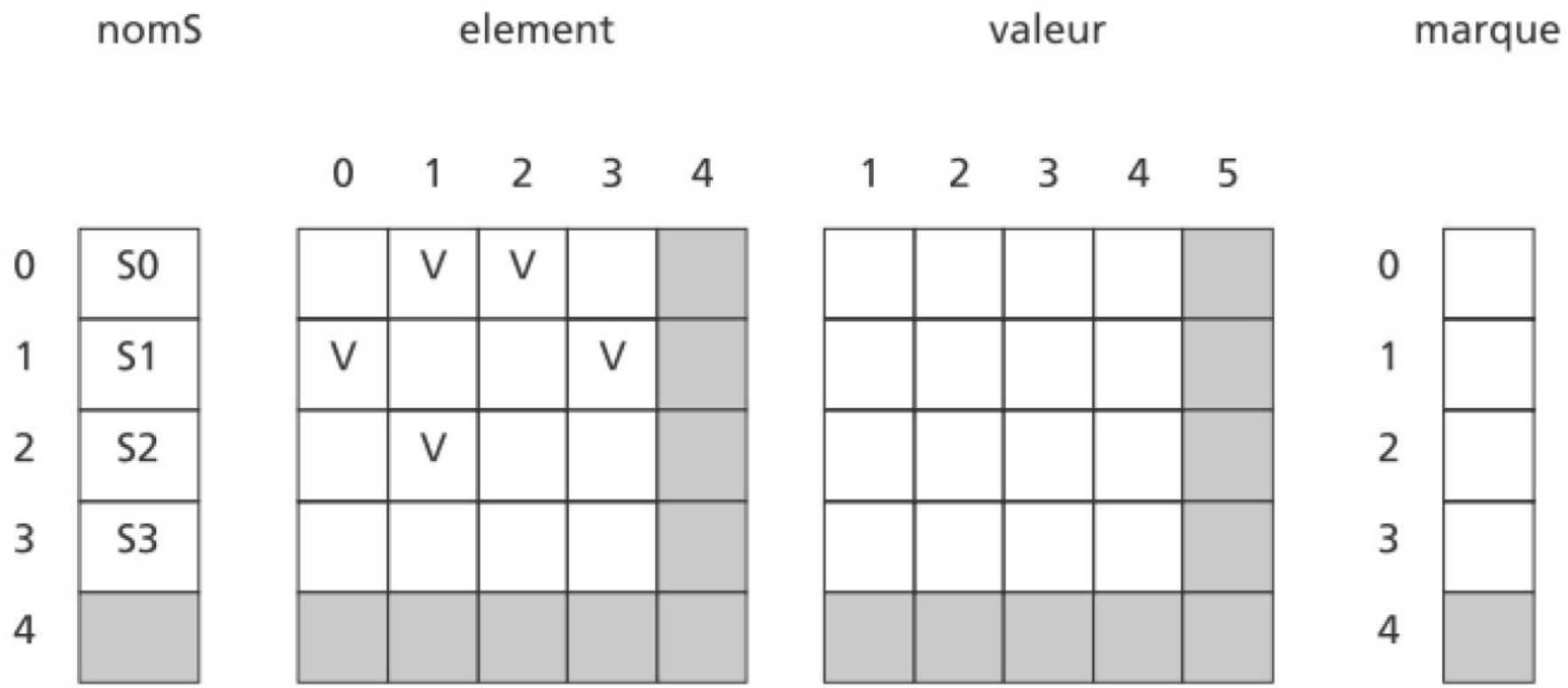



| | nomS |
|---|---|
| 0 | S0 |
| 1 | S1 |
| 2 | S2 |
| 3 | S3 |
| | |
| nMax-1 | |

| 0 | 1 | 2 | 3 | | nMax-1 |
|---|---|---|---|---|---|
| | V | V | | | |
| V | | | V | | |
| | V | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# Représentation d'un graphe

## Représentation plus complète



# Représentation d'un graphe

*graphemat.h*

```
#include <stdio.h>

typedef int  booleen;
#define faux 0
#define vrai 1
typedef char NomSom[20];  // nom d'un sommet
#define INFINI INT_MAX

typedef int* Matrice;

typedef struct {
  int      n;       // nombre de sommets
  int      nMax;    // nombre max de sommets
  booleen  value;   // graphe valué ou non
  NomSom*  nomS;    // noms des sommets
  Matrice  element; // existence d'un arc (i, j)
  Matrice  valeur;  // cout de l'arc (i, j)
  booleen* marque;  // sommet marqué (visité) ou non
} GrapheMat;
```

# Représentation d'un graphe

*graphemat.h (suite)*

```
GrapheMat* creerGrapheMat     (int nMax, int value);
void       detruireGraphe     (GrapheMat* graphe);
void       ajouterUnSommet    (GrapheMat* graphe, NomSom nom);
void       ajouterUnArc       (GrapheMat* graphe, NomSom somD,
                                  NomSom somA, int cout);
void       ecrireGraphe       (GrapheMat* graphe);

void       parcoursProfond    (GrapheMat* graphe);
void       parcoursLargeur    (GrapheMat* graphe);
```

# Représentation d'un graphe

## Création et destruction d'un graphe (matrices)

La fonction **creerGrapheMat** alloue dynamiquement et initialise une structure de type **GrapheMat**.

Si une relation n'existe pas entre 2 sommets i et j, la valeur est notée INFINI (plus grand entier sur ordinateur).

La fonction **detruireGraphe** désalloue une structure de type **GrapheMat** allouée avec **creerGrapheMat**.

```
GrapheMat* creerGrapheMat (int nMax, int value) {

  // allocation de graphe
  GrapheMat* graphe  = (GrapheMat*) malloc (sizeof (GrapheMat));
  graphe->n        = 0;
  graphe->nMax     = nMax;
  graphe->value    = value;
  graphe->nomS     = (NomSom*)  malloc (sizeof(NomSom)  *nMax);
  graphe->marque   = (booleen*) malloc (sizeof(booleen) *nMax);
  graphe->element = (int*)     malloc (sizeof(int)*nMax*nMax);
  graphe->valeur   = (int*)     malloc (sizeof(int)*nMax*nMax);

  // initialisation par défaut
  for (int i=0; i<nMax; i++) {
    for (int j=0; j<nMax; j++) {
      graphe->element [i*nMax+j] = faux;
      graphe->valeur  [i*nMax+j] = INFINI;
    }
  }
  razMarque (graphe);

  return graphe;
}
```

# Représentation d'un graphe

## Création et destruction d'un graphe (matrices)

La fonction **razMarque** (utilisée dans **creerGrapheMat** ) met simplement par défaut tous les champs marque à faux (sommet non visité).

```
// remise à faux du tableau marqué (pour les parcours de graphe)
static void razMarque (GrapheMat* graphe) {
  for (int i=0; i<graphe->n; i++) graphe->marque [i] = faux;
}
```

# Représentation d'un graphe

## Création et destruction d'un graphe (matrices)

```
// désallocation d'un graphe
void detruireGraphe (GrapheMat* graphe) {
   free (graphe->nomS);
   free (graphe->marque);
   free (graphe->element);
   free (graphe->valeur);
   free (graphe);
}
```

# Représentation d'un graphe

## Insertion d'un sommet ou d'un arc dans un graphe

La fonction **rang** (utilisée dans **ajouterUnSommet**) fournit le rang dans le tableau nomS du nom.

```
static int rang (GrapheMat* graphe, NomSom nom) {
  int      i       = 0;
  booleen  trouve = faux;

  while ( (i<graphe->n) && !trouve) {
    trouve = strcmp (graphe->nomS [i], nom) == 0;
    if (!trouve) i++;
  }
  return trouve ? i : -1;
}
```

# Représentation d'un graphe

## Insertion d'un sommet ou d'un arc dans un graphe

```
void ajouterUnSommet (GrapheMat* graphe, NomSom nom) {
  if (rang (graphe, nom) == -1) {
    if (graphe->n < graphe->nMax) {
     strcpy (graphe->nomS [graphe->n++], nom);
    } else {
      printf ("\nNombre de sommets > %d\n", graphe->nMax);
    }
  } else {
    printf ("\n%s déjà défini\n", nom);
  }
}
```

# Représentation d'un graphe

## Insertion d'un sommet ou d'un arc dans un graphe

La fonction **ajouterUnArc** ajoute au graphe un arc entre somD (sommet de départ) et somA (sommet d'arrivé) .

```
void ajouterUnArc (GrapheMat* graphe, NomSom somD, NomSom somA, int cout) {
  int nMax = graphe->nMax;
  int rd = rang (graphe, somD);
  int rg = rang (graphe, somA);
  graphe->element [rd*nMax+rg] = vrai;
  graphe->valeur  [rd*nMax+rg] = cout;
}
```

NB: **rd*nMax+rg** nous permet de se positionner dans la matrice.

# Représentation d'un graphe

## Insertion d'un sommet ou d'un arc dans un graphe

Rappel sur l'arrangement en mémoire d'une matrice :

Voyez ces deux exemples équivalents :

```
int tab [3] [4] =    { { 1, 2, 3, 4 } ,
                       { 5, 6, 7, 8 },
                       { 9,10,11,12 } }

int tab [3] [4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 } ;
```

La première forme revient à considérer notre tableau comme formé de trois tableaux de quatre éléments chacun.

La seconde exploite la manière dont les éléments sont rangés en mémoire et elle se contente d'énumérer les valeurs du tableau suivant cet ordre.

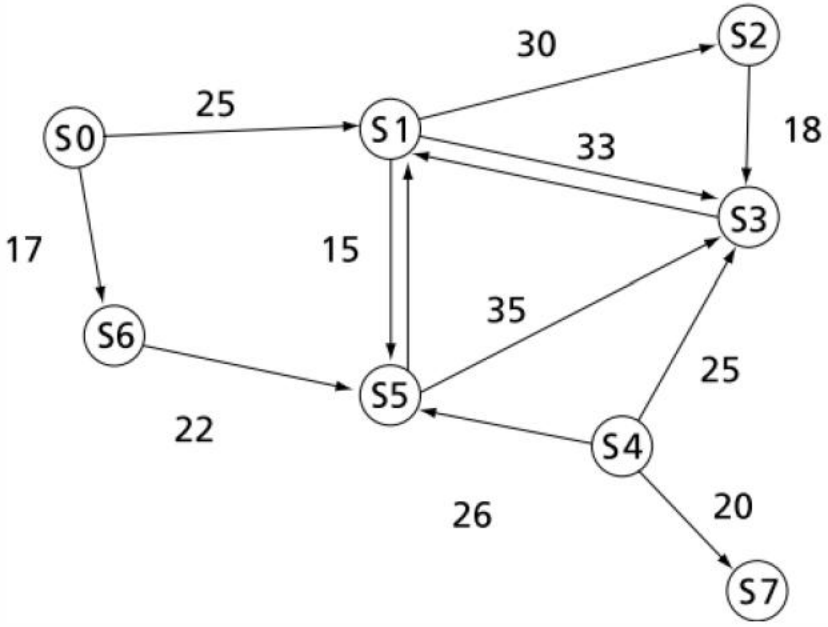
# Représentation d'un graphe

## Écriture d'un graphe

**Exercice :** Donnez le code de la fonction **void ecrireGraphe (GrapheMat* graphe)** qui affiche un graphe comme suit :

S0 S1 S2 S3 S4 S5 S6 S7

S0 : S1 ( 25) S6 ( 17) ;
S1 : S2 ( 30) S3 ( 33) S5 ( 15) ;
S2 : S3 ( 18) ;
S3 : S1 ( 33) ;
S4 : S3 ( 25) S5 ( 26) S7 ( 20) ;
S5 : S1 ( 15) S3 ( 35) ;
S6 : S5 ( 22) ;



# Représentation d'un graphe

## Écriture d'un graphe

```
void ecrireGraphe (GrapheMat* graphe) {
  int nMax = graphe->nMax;

  for (int i=0; i<graphe->n; i++) printf ("%s ", graphe->nomS[i]);
  printf (";\n");

  for (int i=0; i<graphe->n; i++) {
    printf ("\n%s : ", graphe->nomS[i]);
    for (int j=0; j<graphe->n; j++) {
      if (graphe->element [i*nMax+j] == vrai) {
        printf ("%s ", graphe->nomS[j]) ;
        if (graphe->value) {
          printf (" (%d) ", graphe->valeur [i*nMax+j] );
        }
      }
    }
    printf (";");
  }
}
```

## Parcours en profondeur d'un graphe

**Exercice** : Implémentez le parcours en profondeur d'un graphe

Rappel :

On part d'un sommet donné. On énumère le premier fils de ce sommet , puis on repart de ce dernier sommet pour atteindre le premier petit-fils, etc.

Il s'agit pour chaque sommet visité, de choisir un des sommets successeurs du sommet en cours, jusqu'à arriver sur une impasse ou un sommet déjà visité.

Dans ce cas, on revient en arrière pour repartir avec un des successeurs non visité du sommet courant.

# Représentation d'un graphe

```
void parcoursProfond (GrapheMat* graphe) {
  razMarque (graphe);
  for (int i=0; i<graphe->n; i++) {
    if (!graphe->marque [i]) profondeur (graphe, i);
  }
}

static void profondeur (GrapheMat* graphe, int numSommet) {
  int nMax = graphe->nMax;
  graphe->marque [numSommet] = vrai;
  printf ("%s\n", graphe->nomS [numSommet]);

  for (int i=0; i<graphe->n; i++) {
    if ( (graphe->element [numSommet*nMax+i] == vrai)
          && !graphe->marque [i] ) {
      profondeur (graphe, i);
    }
  }
}
```