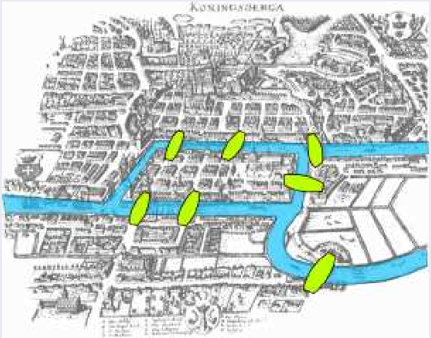
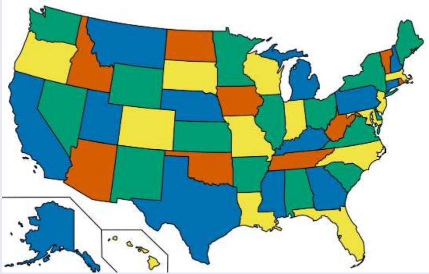
Les sept ponts de Konigsberg



credit : Wikipedia

Est-il possible de trouver un chemin qui passe une seule fois par chaque pont ?

Le problème des quatre couleurs



credit : Wikipedia

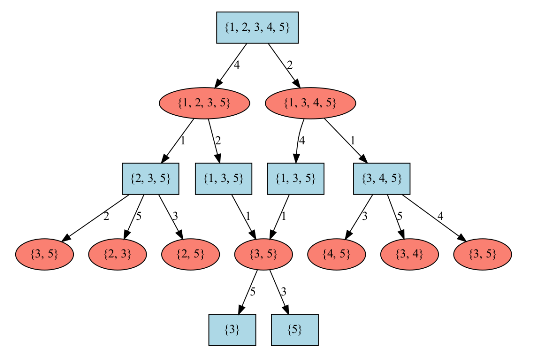
Peut-on colorier n'importe quelle carte en utilisant seulement 4 couleurs ? (sans que deux pays limitrophes aient la même couleur)

Le problème du voyageur de commerce

credit : Wikipedia

Trouver le chemin le plus court possible qui passe par toutes les villes une seule fois et revient à son point de départ.

Le jeu de Juniper Green



Existe-il une stratégie gagnante au jeu de Juniper Green ?

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,
- planification de tâches,

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,
- planification de tâches,
- réseaux : informatique, sociaux,

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,
- planification de tâches,
- réseaux : informatique, sociaux,
- énergie, transport, . . .

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,
- planification de tâches,
- réseaux : informatique, sociaux,
- énergie, transport, ...

La taille des graphes en jeu impose la recherche d'algorithmes efficaces :

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,
- planification de tâches,
- réseaux : informatique, sociaux,
- énergie, transport, ...

La taille des graphes en jeu impose la recherche d'algorithmes efficaces :

Graphe	Sommets	Arcs

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,
- planification de tâches,
- réseaux : informatique, sociaux,
- énergie, transport, ...

La taille des graphes en jeu impose la recherche d'algorithmes efficaces :

Graphe	Sommets	Arcs
Facebook	$\simeq 7,2 \times 10^8$	$\simeq 7 \times 10^9$

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,
- planification de tâches,
- réseaux : informatique, sociaux,
- énergie, transport, ...

La taille des graphes en jeu impose la recherche d'algorithmes efficaces :

Graphe	Sommets	Arcs
Facebook	$\simeq 7,2 \times 10^8$	$\simeq 7 \times 10^9$
OpenStreetMap	$\simeq 9 \times 10^9$	$\simeq 10^9$

Théorie des graphes

Ces problèmes se modélisent et se résolvent à l'aide de la théorie des graphes, un domaine central de l'informatique ayant des applications variées :

- GPS : recherches de chemins,
- planification de tâches,
- réseaux : informatique, sociaux,
- énergie, transport, ...

La taille des graphes en jeu impose la recherche d'algorithmes efficaces :

Graphe	Sommets	Arcs
Facebook	$\simeq 7,2 \times 10^8$	$\simeq 7 \times 10^9$
OpenStreetMap	$\simeq 9 \times 10^9$	$\simeq 10^9$
Jeu d'échecs	$\simeq 10^{47}$	?

Définition

Un graphe non-orienté est la donnée :

Définition

Un graphe non-orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini (V pour *vertice* en anglais.).

Définition

Un graphe non-orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini (V pour *vertice* en anglais.).
- D'un ensemble de paires de sommets A appelés arêtes (E pour *edges* en anglais).

Définition

Un graphe non-orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini (V pour *vertice* en anglais.).
- D'un ensemble de paires de sommets A appelés arêtes (E pour *edges* en anglais).

Remarques

Définition

Un graphe non-orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini (V pour *vertice* en anglais.).
- D'un ensemble de paires de sommets A appelés arêtes (E pour *edges* en anglais).

Remarques

- Une arête est donc une paire $\{x, y\}$, avec $x \in S$, $y \in S$ et $x \neq y$.

Définition

Un graphe non-orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini (V pour *vertice* en anglais.).
- D'un ensemble de paires de sommets A appelés arêtes (E pour *edges* en anglais).

Remarques

- Une arête est donc une paire $\{x, y\}$, avec $x \in S$, $y \in S$ et $x \neq y$.
- Puisque $\{x, y\} = \{y, x\}$, l'ordre n'a pas d'importance.

Définition

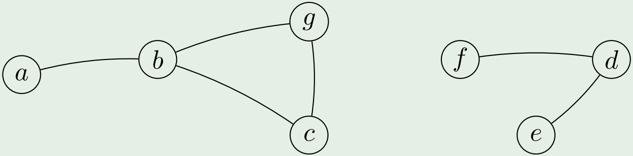
Un graphe non-orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini (V pour *vertice* en anglais.).
- D'un ensemble de paires de sommets A appelés arêtes (E pour *edges* en anglais).

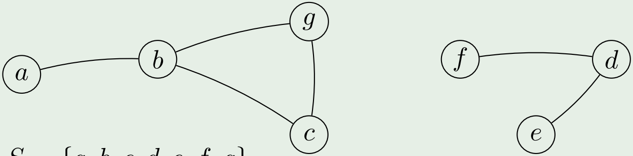
Remarques

- Une arête est donc une paire $\{x, y\}$, avec $x \in S$, $y \in S$ et $x \neq y$.
- Puisque $\{x, y\} = \{y, x\}$, l'ordre n'a pas d'importance.
- On notera $x - y$ ou plus simplement xy l'arête $\{x, y\}$.

Exemple

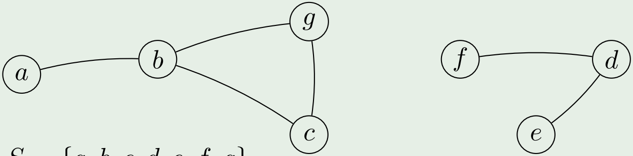


Exemple



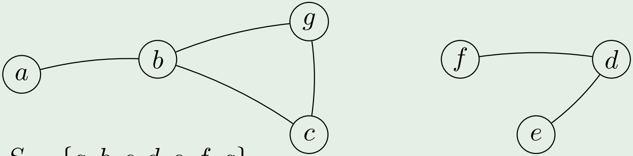
$S = \{a, b, c, d, e, f, g\}$

Exemple



$S = \{a, b, c, d, e, f, g\}$
$A = \{\{a, b\}, \{b, c\}, \{b, g\}, \{c, g\}, \{f, d\}, \{e, d\}\}$

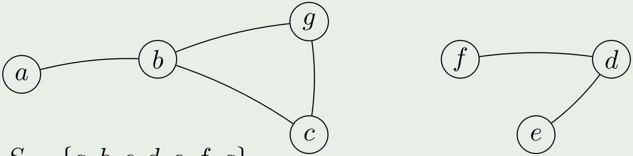
Exemple



$S = \{a, b, c, d, e, f, g\}$
$A = \{\{a, b\}, \{b, c\}, \{b, g\}, \{c, g\}, \{f, d\}, \{e, d\}\}$

Dessiner le graphe G' défini par $S' = \{1, 2, 3, 4\}$ et $A' = \{\{1, 2\}, \{3, 4\}, \{1, 4\}\}$

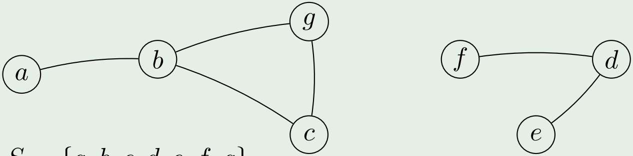
Exemple



$S = \{a, b, c, d, e, f, g\}$
$A = \{\{a, b\}, \{b, c\}, \{b, g\}, \{c, g\}, \{f, d\}, \{e, d\}\}$

Dessiner le graphe G' défini par $S' = \{1, 2, 3, 4\}$ et $A' = \{\{1, 2\}, \{3, 4\}, \{1, 4\}\}$

Exemple



$S = \{a, b, c, d, e, f, g\}$
$A = \{\{a, b\}, \{b, c\}, \{b, g\}, \{c, g\}, \{f, d\}, \{e, d\}\}$

Dessiner le graphe G' défini par $S' = \{1, 2, 3, 4\}$ et $A' = \{\{1, 2\}, \{3, 4\}, \{1, 4\}\}$

⚠ L'emplacement des nœuds sur la représentation graphique n'a pas d'importance.

Vocabulaire

- L'ordre d'un graphe est son nombre de sommets.

Vocabulaire

- L'ordre d'un graphe est son nombre de sommets.
- Les voisins d'un sommet s est l'ensemble $\mathcal{V}(s) = \{t \in S \text{ tel que } s - t \in A\}$.

Vocabulaire

- L'ordre d'un graphe est son nombre de sommets.
- Les voisins d'un sommet s est l'ensemble $\mathcal{V}(s) = \{t \in S \text{ tel que } s - t \in A\}$.
- Le degré d'un sommet s noté $d(s)$ est son nombre de voisins $d(s) = |\mathcal{V}(s)|$.

Vocabulaire

- L'ordre d'un graphe est son nombre de sommets.
- Les voisins d'un sommet s est l'ensemble $\mathcal{V}(s) = \{t \in S \text{ tel que } s - t \in A\}$.
- Le degré d'un sommet s noté $d(s)$ est son nombre de voisins $d(s) = |\mathcal{V}(s)|$.

Exercices

1. Dessiner un graphe d'ordre 5 dont tous les sommets sont de degré 2.

Vocabulaire

- L'ordre d'un graphe est son nombre de sommets.
- Les voisins d'un sommet s est l'ensemble $\mathcal{V}(s) = \{t \in S \text{ tel que } s - t \in A\}$.
- Le degré d'un sommet s noté $d(s)$ est son nombre de voisins $d(s) = |\mathcal{V}(s)|$.

Exercices

1. Dessiner un graphe d'ordre 5 dont tous les sommets sont de degré 2.
2. Donner un majorant du nombre d'arêtes d'un graphe d'ordre n.

Représentation par matrice d'adjacence

Soit un graphe $G = (S, A)$ avec $|S| = n$, on note $S = \{x_0, \dots, x_{n-1}\}$. La matrice d'adjacence du graphe G est la matrice carrée d'ordre n, M définie par : $M_{ij} = 1$ si $\{x_i, x_j\} \in A$ et $M_{ij} = 0$ sinon.

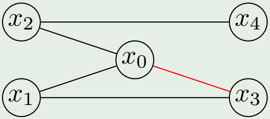
Représentation par matrice d'adjacence

Soit un graphe $G = (S, A)$ avec $|S| = n$, on note $S = \{x_0, \dots, x_{n-1}\}$. La matrice d'adjacence du graphe G est la matrice carrée d'ordre n, M définie par : $M_{ij} = 1$ si $\{x_i, x_j\} \in A$ et $M_{ij} = 0$ sinon.
M est une matrice symétrique.

Représentation par matrice d'adjacence

Soit un graphe $G = (S, A)$ avec $|S| = n$, on note $S = \{x_0, \dots, x_{n-1}\}$. La matrice d'adjacence du graphe G est la matrice carrée d'ordre n, M définie par : $M_{ij} = 1$ si $\{x_i, x_j\} \in A$ et $M_{ij} = 0$ sinon.
M est une matrice symétrique.

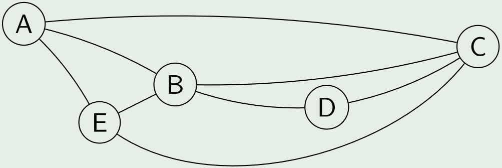
Exemple : Un graphe et sa matrice d'adjacence



$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

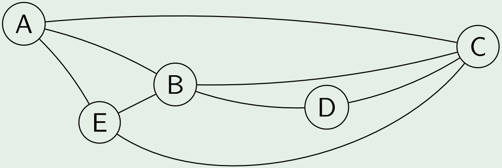
Exemple

1. En supposant les sommets numérotés dans l'ordre alphabétique, écrire la matrice d'adjacence du graphe suivant :



Exemple

1. En supposant les sommets numérotés dans l'ordre alphabétique, écrire la matrice d'adjacence du graphe suivant :



2. Dessiner le graphe ayant la matrice d'adjacence suivante (on appellera les sommets $x_0, x_1, \dots$) :

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Implémentation en C : tableau statique

Implémentation en C : tableau statique

```c
#define NMAX 100 // nombre maximal de sommets
typedef bool graphe[NMAX][NMAX]; // La matrice d'adjacence
```

Implémentation en C : tableau statique

```c
#define NMAX 100 // nombre maximal de sommets
typedef bool graphe[NMAX][NMAX]; // La matrice d'adjacence
```

On évite ainsi l'utilisation de pointeurs et on manipule directement la matrice d'adjacence. On doit passer la taille effective du graphe aux fonctions lorsque nécessaire.

Implémentation en C : tableau statique

```c
#define NMAX 100 // nombre maximal de sommets
typedef bool graphe[NMAX][NMAX]; // La matrice d'adjacence
```

On évite ainsi l'utilisation de pointeurs et on manipule directement la matrice d'adjacence. On doit passer la taille effective du graphe aux fonctions lorsque nécessaire.

Exemple

Ecrire la fonction de signature `void cree_arete(graphe g, int i, int j)` qui crée l'arête reliant les sommets `i` et `j`.

Implémentation en C : tableau statique

```c
#define NMAX 100 // nombre maximal de sommets
typedef bool graphe[NMAX][NMAX]; // La matrice d'adjacence
```

On évite ainsi l'utilisation de pointeurs et on manipule directement la matrice d'adjacence. On doit passer la taille effective du graphe aux fonctions lorsque nécessaire.

Exemple

Ecrire la fonction de signature `void cree_arete(graphe g, int i, int j)` qui crée l'arête reliant les sommets `i` et `j`.

```c
void cree_arete(graphe g, int i, int j){
    g[i][j] = true;
    g[j][i] = true;}
```

Implémentation en C : type structuré

On utilise un `struct` contenant un champ `int` (le nombre de sommets), et un pointeur vers la matrice d'adjacence linéarisée.

Implémentation en C : type structuré

On utilise un `struct` contenant un champ `int` (le nombre de sommets), et un pointeur vers la matrice d'adjacence linéarisée. L'indice de M_{ij} dans la matrice linéarisée est alors $i \times |S| + j$.

Implémentation en C : type structuré

On utilise un `struct` contenant un champ `int` (le nombre de sommets), et un pointeur vers la matrice d'adjacence linéarisée. L'indice de M_{ij} dans la matrice linéarisée est alors $i \times |S| + j$.

```
struct graphe
{
    int taille; // |S|
    bool * madj; // matrice linéarisée (à allouer)
};
typedef struct graphe graphe;
```

Implémentation en C : type structuré

On utilise un `struct` contenant un champ `int` (le nombre de sommets), et un pointeur vers la matrice d'adjacence linéarisée. L'indice de M_{ij} dans la matrice linéarisée est alors $i \times |S| + j$.

```c
struct graphe
{
    int taille; // |S|
    bool * madj; // matrice linéarisée (à allouer)
};
typedef struct graphe graphe;
```

Cette solution est plus efficace en terme d'occupation mémoire mais impose l'utilisation de pointeurs.

Exemple

Pour initialiser une variable de type graphe lorsque le nombre de sommets est connu, on propose la solution suivante :

Exemple

Pour initialiser une variable de type graphe lorsque le nombre de sommets est connu, on propose la solution suivante :

```c
graphe cree_graphe(int n){
    graphe g;
    g.taille = n;
    g.madj = malloc(sizeof(bool)*n*n);
    for (int i=0;i<n*n;i++)
    {g.madj[i]=false;}
    return g;}
```

Exemple

Pour initialiser une variable de type graphe lorsque le nombre de sommets est connu, on propose la solution suivante :

```
graphe cree_graphe(int n){
    graphe g;
    g.taille = n;
    g.madj = malloc(sizeof(bool)*n*n);
    for (int i=0;i<n*n;i++)
    {g.madj[i]=false;}
    return g;}
```

Ecrire la fonction `detruit_graphe` qui permet de libérer la mémoire allouée par la fonction d'initialisation ci-dessus

Exemple

Pour initialiser une variable de type graphe lorsque le nombre de sommets est connu, on propose la solution suivante :

```
graphe cree_graphe(int n){
    graphe g;
    g.taille = n;
    g.madj = malloc(sizeof(bool)*n*n);
    for (int i=0;i<n*n;i++)
    {g.madj[i]=false;}
    return g;}
```

Ecrire la fonction `detruit_graphe` qui permet de libérer la mémoire allouée par la fonction d'initialisation ci-dessus

```
void detruit_graphe(graphe g){
    free(g.madj);}
```

Implémentation en OCaml

- On crée le type graphe sous forme d'un type structuré

Implémentation en OCaml

- On crée le type graphe sous forme d'un type structuré

```
type graphe = {
  taille : int;
  madj : bool array array};;
```

Implémentation en OCaml

- On crée le type graphe sous forme d'un type structuré

```
type graphe = {
  taille : int;
  madj : bool array array};;
```

- La création d'un graphe peut alors se faire à l'aide d'une fonction d'initialisation

Implémentation en OCaml

- On crée le type graphe sous forme d'un type structuré

```
type graphe = {
  taille : int;
  madj : bool array array};;
```

- La création d'un graphe peut alors se faire à l'aide d'une fonction d'initialisation

```
let cree_graphe n =
  {taille = n; madj = Array.make_matrix n n false};;
```

Implémentation en OCaml

- On crée le type graphe sous forme d'un type structuré

```
type graphe = {
  taille : int;
  madj : bool array array};;
```

- La création d'un graphe peut alors se faire à l'aide d'une fonction d'initialisation

```
let cree_graphe n =
  {taille = n; madj = Array.make_matrix n n false};;
```

⚠ On rappelle qu'on initialise avec `Array.make_matrix` afin d'éviter le problème des références multiples à une même ligne.

Exemples

1. Ecrire la fonction `cree_arete : graphe -> int -> int -> unit` qui crée une arête dans un graphe.

2. Ecrire une fonction `degre : graphe -> int -> int` qui renvoie le degré d'un sommet.

Exemples

1. Ecrire la fonction `cree_arete : graphe -> int -> int -> unit` qui crée une arête dans un graphe.

```
let cree_arete graphe i j =
  graphe.madj.(i).(j) <- true;
  graphe.madj.(j).(i) <- true;;
```

2. Ecrire une fonction `degre : graphe -> int -> int` qui renvoie le degré d'un sommet.

```
let degre graphe i =
  let d = ref 0 in
  for j = 0 to (Array.length graphe.madj - 1) do
    if (graphe.madj.(i).(j)) then d:=!d+1;
  done;
  !d;;
```

Représentation par listes d'adjacence

On peut représenter un graphe à l'aide de listes d'adjacences, c'est à dire en mémorisant pour chaque sommet du graphe la liste de ses voisins.

Représentation par listes d'adjacence

On peut représenter un graphe à l'aide de listes d'adjacences, c'est à dire en mémorisant pour chaque sommet du graphe la liste de ses voisins.
Un graphe $G = (S, A)$ est alors un tableau de taille $|S|$ où la case d'indice i contient la liste des voisins du sommet x_i.

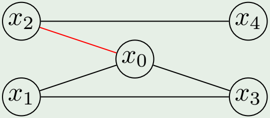
Représentation par listes d'adjacence

On peut représenter un graphe à l'aide de listes d'adjacences, c'est à dire en mémorisant pour chaque sommet du graphe la liste de ses voisins.
Un graphe $G = (S, A)$ est alors un tableau de taille $|S|$ où la case d'indice i contient la liste des voisins du sommet x_i.

Représentation par listes d'adjacence

On peut représenter un graphe à l'aide de listes d'adjacences, c'est à dire en mémorisant pour chaque sommet du graphe la liste de ses voisins.
Un graphe $G = (S, A)$ est alors un tableau de taille $|S|$ où la case d'indice i contient la liste des voisins du sommet x_i.

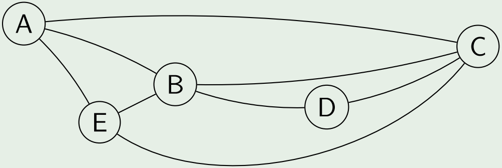
Exemple : Un graphe et ses listes d'adjacence




```
0  →  [1; 2; 3]
1  →  [0; 3]
2  →  [0; 4]
3  →  [0; 1]
4  →  [2]
```

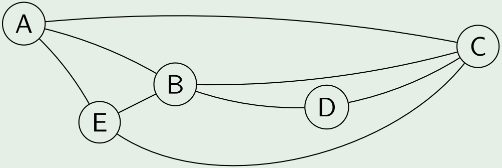
Exemple

1. Ecrire les listes d'adjacences du graphe suivante :



Exemple

1. Ecrire les listes d'adjacences du graphe suivante :



2. Dessiner le graphe représenté par le tableau T tel que :

```
T[0] = [2]
T[1] = [3; 4]
T[2] = [0; 1]
T[3] = [1; 2]
T[4] = [1; 2]
```

Implémentation en C : tableau statique d'entiers

Implémentation en C : tableau statique d'entiers

```c
#define NMAX  100 // nombre maximal de sommets
// graphe[i][0] contient le degré du sommet i
// graphe[i][1..] est la liste d'adjacence du sommet i
typedef int graphe[NMAX][NMAX];
```

Implémentation en C : tableau statique d'entiers

```c
#define NMAX  100 // nombre maximal de sommets
// graphe[i][0] contient le degré du sommet i
// graphe[i][1..] est la liste d'adjacence du sommet i
typedef int graphe[NMAX][NMAX];
```

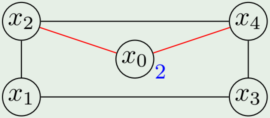
On évite de cette façon l'utilisation de pointeurs mais au prix d'un manque d'efficacité en terme d'occupation mémoire.

Implémentation en C : tableau statique d'entiers

```c
#define NMAX  100 // nombre maximal de sommets
// graphe[i][0] contient le degré du sommet i
// graphe[i][1..] est la liste d'adjacence du sommet i
typedef int graphe[NMAX][NMAX];
```

On évite de cette façon l'utilisation de pointeurs mais au prix d'un manque d'efficacité en terme d'occupation mémoire.

Exemple : Un graphe et sa matrice d'adjacence



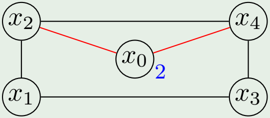
$$\begin{pmatrix} 2 & 2 & 4 & 0 & 0 \\ 2 & 2 & 3 & 0 & 0 \\ 3 & 0 & 1 & 4 & 0 \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \end{pmatrix}$$

Implémentation en C : tableau statique d'entiers

```c
#define NMAX  100 // nombre maximal de sommets
// graphe[i][0] contient le degré du sommet i
// graphe[i][1..] est la liste d'adjacence du sommet i
typedef int graphe[NMAX][NMAX];
```

On évite de cette façon l'utilisation de pointeurs mais au prix d'un manque d'efficacité en terme d'occupation mémoire.

Exemple : Un graphe et sa matrice d'adjacence



$$\begin{pmatrix} {\color{blue}2} & {\color{red}2} & {\color{red}4} & 0 & 0 \\ 2 & 2 & 3 & 0 & 0 \\ 3 & 0 & 1 & 4 & 0 \\ 2 & 1 & 4 & 0 & 0 \\ 3 & 2 & 0 & 3 & 0 \end{pmatrix}$$

Exemples

1. Ecrire la fonction de signature `int degre(graphe g, int i)` qui renvoie le degré du sommet de numéro `i`.

2. Ecrire la fonction de signature
`void cree_arete(graphe g, int i, int j)` qui crée l'arête entre les sommets `i` et `j` dans le graphe `g`

Exemples

1. Ecrire la fonction de signature `int degre(graphe g, int i)` qui renvoie le degré du sommet de numéro `i`.

```
int degre(graphe g, int i)
{
    return g[i][0];
}
```

2. Ecrire la fonction de signature
`void cree_arete(graphe g, int i, int j)` qui crée l'arête entre les sommets `i` et `j` dans le graphe `g`

Exemples

1. Ecrire la fonction de signature `int degre(graphe g, int i)` qui renvoie le degré du sommet de numéro `i`.

```c
int degre(graphe g, int i)
{
    return g[i][0];
}
```

2. Ecrire la fonction de signature `void cree_arete(graphe g, int i, int j)` qui crée l'arête entre les sommets `i` et `j` dans le graphe `g`

```c
void cree_arete(graphe g, int i, int j){
    g[i][0]++;
    g[j][0]++;
    g[i][g[i][0]] = j;
    g[j][g[j][0]] = i;}
```

Implémentation en C : tableau de listes chainées

On utilise un tableau de listes chainées afin d'optimiser l'espace mémoire occupé :

Implémentation en C : tableau de listes chainées

On utilise un tableau de listes chainées afin d'optimiser l'espace mémoire occupé :

```c
struct graphe{
    int taille;
    liste* ladj;};
typedef struct graphe graphe;
```

Implémentation en C : tableau de listes chainées

On utilise un tableau de listes chainées afin d'optimiser l'espace mémoire occupé :

```c
struct graphe{
    int taille;
    liste* ladj;};
typedef struct graphe graphe;
```

On rappelle la structure de liste chainée :

```c
struct maillon{
    int valeur;
    struct maillon* suivant;};
typedef struct maillon maillon;
typedef maillon* liste;
```

Implémentation en OCaml

On utilise le type `list` de OCaml

Implémentation en OCaml

On utilise le type `list` de OCaml

```
type graphe = {
taille : int;
ladj : int list array};;
```

Implémentation en OCaml

On utilise le type `list` de OCaml

```
type graphe = {
taille : int;
ladj : int list array};;
```

On peut alors écrire une fonction de création d'un graphe de taille `n` :

Implémentation en OCaml

On utilise le type list de OCaml

```
type graphe = {
taille : int;
ladj : int list array};;
```

On peut alors écrire une fonction de création d'un graphe de taille n :

```
let cree_graphe n =
  {taille=n; ladj = Array.make n []}
```

Implémentation en OCaml

On utilise le type `list` de OCaml

```
type graphe = {
taille : int;
ladj : int list array};;
```

On peut alors écrire une fonction de création d'un graphe de taille `n` :

```
let cree_graphe n =
  {taille=n; ladj = Array.make n []}
```

Exemple

Ecrire la fonction `cree_arete : graphe -> int -> int -> unit`

Implémentation en OCaml

On utilise le type list de OCaml

```
type graphe = {
taille : int;
ladj : int list array};;
```

On peut alors écrire une fonction de création d'un graphe de taille n :

```
let cree_graphe n =
  {taille=n; ladj = Array.make n []}
```

Exemple

Ecrire la fonction cree_arete : graphe -> int -> int -> unit

```
let cree_arete g i j =
  g.ladj.(i) <- j::g.ladj.(i);
  g.ladj.(j) <- i::g.ladj.(j);;
```

Comparatif

	Matrice d'adjacence	Liste d'adjacence
Occupation mémoire	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Modification d'arête	$O(1)$	$O(\|S\|)$
Test d'existence d'une arête	$O(1)$	$O(\|S\|)$
Enumération des voisins	$O(\|S\|)$	$O(1)$

Comparatif

	Matrice d'adjacence	Liste d'adjacence
Occupation mémoire	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Modification d'arête	$O(1)$	$O(\|S\|)$
Test d'existence d'une arête	$O(1)$	$O(\|S\|)$
Enumération des voisins	$O(\|S\|)$	$O(1)$

Un choix d'implémentation doit donc tenir compte :

Comparatif

	Matrice d'adjacence	Liste d'adjacence
Occupation mémoire	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Modification d'arête	$O(1)$	$O(\|S\|)$
Test d'existence d'une arête	$O(1)$	$O(\|S\|)$
Enumération des voisins	$O(\|S\|)$	$O(1)$

Un choix d'implémentation doit donc tenir compte :

- de l'ordre du graphe

Comparatif

	Matrice d'adjacence	Liste d'adjacence
Occupation mémoire	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Modification d'arête	$O(1)$	$O(\|S\|)$
Test d'existence d'une arête	$O(1)$	$O(\|S\|)$
Enumération des voisins	$O(\|S\|)$	$O(1)$

Un choix d'implémentation doit donc tenir compte :

- de l'ordre du graphe
- de la « *densité* » du graphe

Comparatif

	Matrice d'adjacence	Liste d'adjacence
Occupation mémoire	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Modification d'arête	$O(1)$	$O(\|S\|)$
Test d'existence d'une arête	$O(1)$	$O(\|S\|)$
Enumération des voisins	$O(\|S\|)$	$O(1)$

Un choix d'implémentation doit donc tenir compte :

- de l'ordre du graphe
- de la « *densité* » du graphe
- des algorithmes utilisés

Comparatif

	Matrice d'adjacence	Liste d'adjacence
Occupation mémoire	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Modification d'arête	$O(1)$	$O(\|S\|)$
Test d'existence d'une arête	$O(1)$	$O(\|S\|)$
Enumération des voisins	$O(\|S\|)$	$O(1)$

Un choix d'implémentation doit donc tenir compte :

- de l'ordre du graphe
- de la « *densité* » du graphe
- des algorithmes utilisés

⚠ Les complexités spatiales et temporelles des algorithmes dépendent de la représentation choisie !

Comparatif

	Matrice d'adjacence	Liste d'adjacence
Occupation mémoire	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Modification d'arête	$O(1)$	$O(\|S\|)$
Test d'existence d'une arête	$O(1)$	$O(\|S\|)$
Enumération des voisins	$O(\|S\|)$	$O(1)$

Un choix d'implémentation doit donc tenir compte :

- de l'ordre du graphe
- de la « *densité* » du graphe
- des algorithmes utilisés

⚠ Les complexités spatiales et temporelles des algorithmes dépendent de la représentation choisie !

Par exemple, le test d'existence d'une arête est une opération élémentaire dans le cas représentation par matrice d'adjacence et est de complexité linéaire (en $|S|$), dans le cas des listes d'adjacence.

Relation non symétrique

Dans de nombreuses situations (voie à sens unique, lien d'un site web à un autre, relation « *follower* » dans un réseau social, ...), on doit modéliser sur un ensemble S, des relations non symétriques, ce qui conduit à la notion de graphe orienté.

Relation non symétrique

Dans de nombreuses situations (voie à sens unique, lien d'un site web à un autre, relation « *follower* » dans un réseau social, ...), on doit modéliser sur un ensemble S, des relations non symétriques, ce qui conduit à la notion de graphe orienté.

Définition

Un graphe orienté est la donnée :

Relation non symétrique

Dans de nombreuses situations (voie à sens unique, lien d'un site web à un autre, relation « *follower* » dans un réseau social, ...), on doit modéliser sur un ensemble S, des relations non symétriques, ce qui conduit à la notion de graphe orienté.

Définition

Un graphe orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini.

Relation non symétrique

Dans de nombreuses situations (voie à sens unique, lien d'un site web à un autre, relation « *follower* » dans un réseau social, ...), on doit modéliser sur un ensemble S, des relations non symétriques, ce qui conduit à la notion de graphe orienté.

Définition

Un graphe orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini.
- D'un ensemble de **couples** de sommets $A \subset S \times S$ appelés arcs

Relation non symétrique

Dans de nombreuses situations (voie à sens unique, lien d'un site web à un autre, relation « *follower* » dans un réseau social, ...), on doit modéliser sur un ensemble S, des relations non symétriques, ce qui conduit à la notion de graphe orienté.

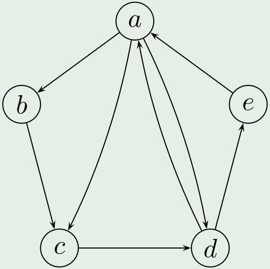
Définition

Un graphe orienté est la donnée :

- D'un ensemble de sommets (ou nœuds) S fini.
- D'un ensemble de **couples** de sommets $A \subset S \times S$ appelés arcs

L'ordre est important, $(x, y) \neq (y, x)$, on notera un arc $x \to y$ ou xy.

Exemple



- $S = \{a, b, c, d, e\}$
- $A = \{(a, b), (b, c), (c, d), (d, e), (e, a), (a, c), (a, d), (d, a)\}$

Vocabulaire

Vocabulaire

- Les successeurs (ou voisins sortants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_+(s) = \{t \in S \text{ tel que } s \to t\}$.

Vocabulaire

- Les successeurs (ou voisins sortants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_+(s) = \{t \in S \text{ tel que } s \to t\}$.
- Le degré sortant d'un sommet s noté $d_+(s)$ est son nombre de successeurs $d_+(s) = |\mathcal{V}_+(s)|$.

Vocabulaire

- Les successeurs (ou voisins sortants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_+(s) = \{t \in S \text{ tel que } s \to t\}$.
- Le degré sortant d'un sommet s noté $d_+(s)$ est son nombre de successeurs $d_+(s) = |\mathcal{V}_+(s)|$.
- Les prédecesseurs (ou voisins entrants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_-(s) = \{t \in S \text{ tel que } t \to s\}$.

Vocabulaire

- Les successeurs (ou voisins sortants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_+(s) = \{t \in S \text{ tel que } s \to t\}$.
- Le degré sortant d'un sommet s noté $d_+(s)$ est son nombre de successeurs $d_+(s) = |\mathcal{V}_+(s)|$.
- Les prédecesseurs (ou voisins entrants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_-(s) = \{t \in S \text{ tel que } t \to s\}$.
- Le degré entrant d'un sommet s noté $d_-(s)$ est son nombre de prédécesseurs $d_-(s) = |\mathcal{V}_-(s)|$.

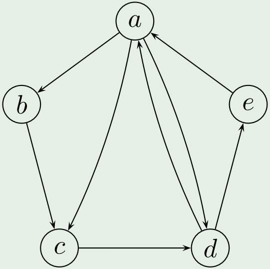
Vocabulaire

- Les successeurs (ou voisins sortants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_+(s) = \{t \in S \text{ tel que } s \to t\}$.
- Le degré sortant d'un sommet s noté $d_+(s)$ est son nombre de successeurs $d_+(s) = |\mathcal{V}_+(s)|$.
- Les prédecesseurs (ou voisins entrants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_-(s) = \{t \in S \text{ tel que } t \to s\}$.
- Le degré entrant d'un sommet s noté $d_-(s)$ est son nombre de prédécesseurs $d_-(s) = |\mathcal{V}_-(s)|$.
- Les voisins d'un sommet s sont les éléments de $\mathcal{V}(s) = \mathcal{V}_+(s) \cup \mathcal{V}_-(s)$

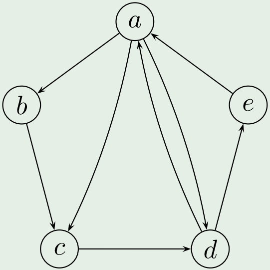
Vocabulaire

- Les successeurs (ou voisins sortants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_+(s) = \{t \in S \text{ tel que } s \to t\}$.
- Le degré sortant d'un sommet s noté $d_+(s)$ est son nombre de successeurs $d_+(s) = |\mathcal{V}_+(s)|$.
- Les prédecesseurs (ou voisins entrants) d'un sommet $s \in S$ sont les éléments de l'ensemble $\mathcal{V}_-(s) = \{t \in S \text{ tel que } t \to s\}$.
- Le degré entrant d'un sommet s noté $d_-(s)$ est son nombre de prédécesseurs $d_-(s) = |\mathcal{V}_-(s)|$.
- Les voisins d'un sommet s sont les éléments de $\mathcal{V}(s) = \mathcal{V}_+(s) \cup \mathcal{V}_-(s)$
- Le degré d'un sommet s noté $d(s)$ est la somme de ses degrés entrants et sortants $d(s) = d_-(s) + d_+(s)$

Exemple

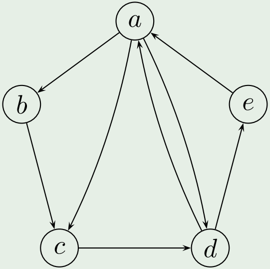


Exemple



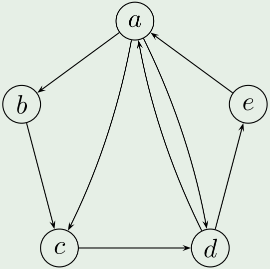
- $V_+(a) = ?$

Exemple



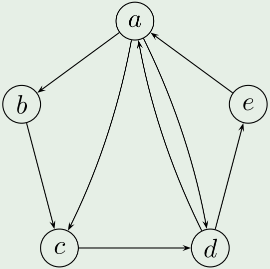
- $V_+(a) = ?$
- $V_+(d) = ?$

Exemple



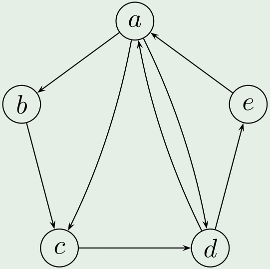
- $V_+(a) = ?$
- $V_+(d) = ?$
- $d_+(e) = ?$

Exemple



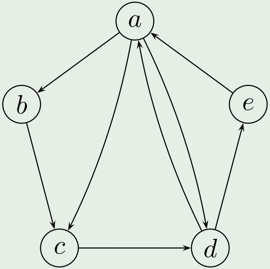
- $V_+(a) = ?$
- $V_+(d) = ?$
- $d_+(e) = ?$
- $d_-(a) = ?$

Exemple



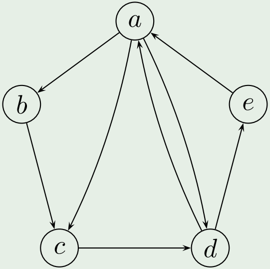
- $V_+(a) = \{b, c, d\}$
- $V_+(d) = ?$
- $d_+(e) = ?$
- $d_-(a) = ?$

Exemple



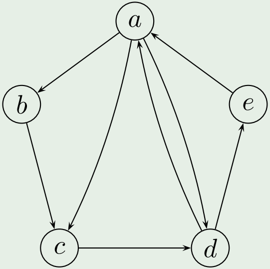
- $V_+(a) = \{b, c, d\}$
- $V_+(d) = \{a, c\}$
- $d_+(e) = ?$
- $d_-(a) = ?$

Exemple



- $V_+(a) = \{b, c, d\}$
- $V_+(d) = \{a, c\}$
- $d_+(e) = 1$
- $d_-(a) = ?$

Exemple



- $V_+(a) = \{b, c, d\}$
- $V_+(d) = \{a, c\}$
- $d_+(e) = 1$
- $d_-(a) = 2$

Représentation informatique

- La représentation par matrice d'adjacence reste valide mais la matrice n'est plus nécessairement symétrique.

Représentation informatique

- La représentation par matrice d'adjacence reste valide mais la matrice n'est plus nécessairement symétrique.
- Dans la représentation par liste d'adjacence, les listes contiennent les voisins sortants.

Représentation informatique

- La représentation par matrice d'adjacence reste valide mais la matrice n'est plus nécessairement symétrique.
- Dans la représentation par liste d'adjacence, les listes contiennent les voisins sortants.
- Les remarques sur le choix d'une représentation restent valides.

Représentation informatique

- La représentation par matrice d'adjacence reste valide mais la matrice n'est plus nécessairement symétrique.
- Dans la représentation par liste d'adjacence, les listes contiennent les voisins sortants.
- Les remarques sur le choix d'une représentation restent valides.
⚠ Attention cependant, dans le cas des listes d'adjacence on a un accès en $O(1)$ à la liste des voisins **sortants**, lister les voisins entrants s'avère plus compliqué (voir TP).

Graphes pondérés

Dans de nombreuses situations (distance entre deux villes, capacité d'une liaison dans un réseau, ...), on souhaite pouvoir ajouter des informations aux arêtes d'un graphe, ce qui conduit à la notion de graphe pondéré (orienté ou non).

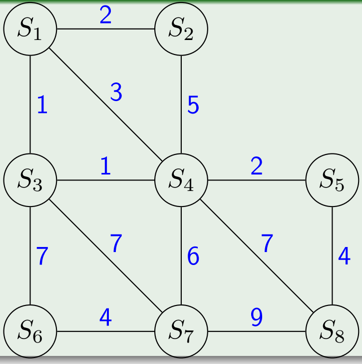
Graphes pondérés

Dans de nombreuses situations (distance entre deux villes, capacité d'une liaison dans un réseau, ...), on souhaite pouvoir ajouter des informations aux arêtes d'un graphe, ce qui conduit à la notion de graphe pondéré (orienté ou non).

Définition

Etant donné un graphe $G = (S, A)$ une fonction de pondération de G est un fonction $\omega : A \mapsto \mathbb{R}$. Le poids de l'arête (ou arc) a est le réel $w(a)$ et on dit que (G, S, ω) est un graphe pondéré

Exemple



Représentation informatique

En pratique, on se limitera le plus souvent à des poids entiers et positifs.

- Dans le cas d'une représentation par matrice d'adjacence, on posera :

$$M_{ij} = \begin{cases} 0 & \text{si } i = j \\ \omega(ij) & \text{si } i \neq j \text{ et } ij \in A \\ +\infty & \text{sinon} \end{cases}$$

Représentation informatique

En pratique, on se limitera le plus souvent à des poids entiers et positifs.

- Dans le cas d'une représentation par matrice d'adjacence, on posera :
$$M_{ij} = \begin{cases} 0 & \text{si } i = j \\ \omega(ij) & \text{si } i \neq j \text{ et } ij \in A \\ +\infty & \text{sinon} \end{cases}$$
- Dans le cas d'une représentation par liste d'adjacence, on stocke dans la liste d'adjacence d'un sommet s, les couples $(t, \omega(st))$

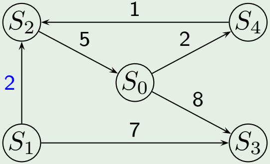
Représentation informatique

En pratique, on se limitera le plus souvent à des poids entiers et positifs.

- Dans le cas d'une représentation par matrice d'adjacence, on posera :
$$M_{ij} = \begin{cases} 0 & \text{si } i = j \\ \omega(ij) & \text{si } i \neq j \text{ et } ij \in A \\ +\infty & \text{sinon} \end{cases}$$
- Dans le cas d'une représentation par liste d'adjacence, on stocke dans la liste d'adjacence d'un sommet s, les couples $(t, \omega(st))$

⚠ L'absence d'arc est indiqué par un 0 dans la matrice d'adjacence d'un graphe non pondéré et par un $+\infty$ dans celle d'un graphe pondéré.

Exemple : Un graphe orienté pondéré et sa matrice d'adjacence



$$\begin{pmatrix} 0 & +\infty & +\infty & 8 & 2 \\ +\infty & 0 & \color{blue}{2} & 7 & +\infty \\ 5 & +\infty & 0 & +\infty & +\infty \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & 1 & +\infty & 0 \end{pmatrix}$$

Définition

- Un chemin de longueur n dans un graphe $G = (S, A)$ est une suite de sommets $(x_0, \ldots, x_n) \in S^n$ telle que pour tout $i \in [\![0; n-1]\!]$, $x_i x_{i+1} \in A$.

Définition

- Un chemin de longueur n dans un graphe $G = (S, A)$ est une suite de sommets $(x_0, \ldots, x_n) \in S^n$ telle que pour tout $i \in [\![0; n-1]\!]$, $x_i x_{i+1} \in A$.
- Un chemin est élémentaire lorsqu'il est sans répétition de *sommets*.

Définition

- Un chemin de longueur n dans un graphe $G = (S, A)$ est une suite de sommets $(x_0, \ldots, x_n) \in S^n$ telle que pour tout $i \in [\![0; n-1]\!]$, $x_i x_{i+1} \in A$.
- Un chemin est élémentaire lorsqu'il est sans répétition de *sommets*.
- Un chemin est simple lorsqu'il est sans répétition d'*arcs*.

Définition

- Un chemin de longueur n dans un graphe $G = (S, A)$ est une suite de sommets $(x_0, \dots, x_n) \in S^n$ telle que pour tout $i \in [\![0; n-1]\!]$, $x_i x_{i+1} \in A$.
- Un chemin est élémentaire lorsqu'il est sans répétition de *sommets*.
- Un chemin est simple lorsqu'il est sans répétition d'*arcs*.

Remarques

- Ces définitions sont valables dans le cas orienté $(x_i x_{i+1} = x_i \to x_{i+1})$ ou non orienté $(x_i x_{i+1} = x_i - x_{i+1})$

Définition

- Un chemin de longueur n dans un graphe $G = (S, A)$ est une suite de sommets $(x_0, \dots, x_n) \in S^n$ telle que pour tout $i \in [\![0; n-1]\!]$, $x_i x_{i+1} \in A$.
- Un chemin est élémentaire lorsqu'il est sans répétition de *sommets*.
- Un chemin est simple lorsqu'il est sans répétition d'*arcs*.

Remarques

- Ces définitions sont valables dans le cas orienté $(x_i x_{i+1} = x_i \to x_{i+1})$ ou non orienté $(x_i x_{i+1} = x_i - x_{i+1})$
- La longueur est le nombre d'*arcs* (un chemin de longueur n contient $n+1$ sommets)

Définition

- Un chemin de longueur n dans un graphe $G = (S, A)$ est une suite de sommets $(x_0, \dots, x_n) \in S^n$ telle que pour tout $i \in [\![0; n-1]\!]$, $x_i x_{i+1} \in A$.
- Un chemin est élémentaire lorsqu'il est sans répétition de *sommets*.
- Un chemin est simple lorsqu'il est sans répétition d'*arcs*.

Remarques

- Ces définitions sont valables dans le cas orienté $(x_i x_{i+1} = x_i \to x_{i+1})$ ou non orienté $(x_i x_{i+1} = x_i - x_{i+1})$
- La longueur est le nombre d'*arcs* (un chemin de longueur n contient $n+1$ sommets)
- Un cycle est un chemin simple avec $x_0 = x_n$ et $n > 0$.

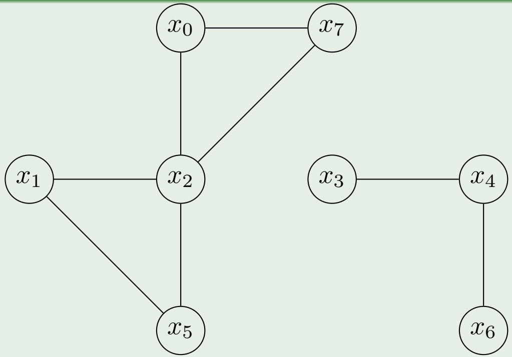
Définition

- Un chemin de longueur n dans un graphe $G = (S, A)$ est une suite de sommets $(x_0, \dots, x_n) \in S^n$ telle que pour tout $i \in [\![0; n-1]\!]$, $x_i x_{i+1} \in A$.
- Un chemin est élémentaire lorsqu'il est sans répétition de *sommets*.
- Un chemin est simple lorsqu'il est sans répétition d'*arcs*.

Remarques

- Ces définitions sont valables dans le cas orienté $(x_i x_{i+1} = x_i \to x_{i+1})$ ou non orienté $(x_i x_{i+1} = x_i - x_{i+1})$
- La longueur est le nombre d'*arcs* (un chemin de longueur n contient $n+1$ sommets)
- Un cycle est un chemin simple avec $x_0 = x_n$ et $n > 0$.
- Un cycle est dit élémentaire lorsque la seule répétition de sommets est celle des extrémités.

Exemple



- il n'existe pas de chemin de x_1 à x_6
- $(x_1, x_2, x_7, x_0, x_2, x_5)$ est un chemin simple (mais pas élémentaire).
- $(x_5, x_2, x_0, x_7, x_2, x_1, x_5)$ est un cycle (qui n'est pas élémentaire)

Définition

Soeint $G = (S, A)$ un graphe et $x, y \in S^2$ deux sommets, on dit que y est accessible depuis x lorsqu'il existe un chemin reliant x à y. On note alors $x \rightsquigarrow y$.

Définition

Soeint $G = (S, A)$ un graphe et $x, y \in S^2$ deux sommets, on dit que y est accessible depuis x lorsqu'il existe un chemin reliant x à y. On note alors $x \rightsquigarrow y$.

Propriété

La relation $\rightsquigarrow$ est :

- réflexive : $x \rightsquigarrow x$
- transitive : $x \rightsquigarrow y$ et $y \rightsquigarrow z \implies x \rightsquigarrow z$.

Définition

Soeint $G=(S,A)$ un graphe et $x,y \in S^2$ deux sommets, on dit que y est accessible depuis x lorsqu'il existe un chemin reliant x à y. On note alors $x \rightsquigarrow y$.

Propriété

La relation $\rightsquigarrow$ est :

- réflexive : $x \rightsquigarrow x$
- transitive : $x \rightsquigarrow y$ et $y \rightsquigarrow z \implies x \rightsquigarrow z$.
- symétrique *lorsque G est non orienté*.

Définition

Soeint $G=(S,A)$ un graphe et $x,y\in S^2$ deux sommets, on dit que y est accessible depuis x lorsqu'il existe un chemin reliant x à y. On note alors $x\rightsquigarrow y$.

Propriété

La relation $\rightsquigarrow$ est :

- réflexive : $x\rightsquigarrow x$
- transitive : $x\rightsquigarrow y$ et $y\rightsquigarrow z \implies x\rightsquigarrow z$.
- symétrique *lorsque G est non orienté*.

Dans le cas non orienté, on appelle composantes connexes les classes d'équivalence de la relation $\rightsquigarrow$.

Connexité dans le cas orienté

Dans le cas où $G = (S, A)$ est orienté on distingue :

Connexité dans le cas orienté

Dans le cas où $G = (S, A)$ est orienté on distingue :

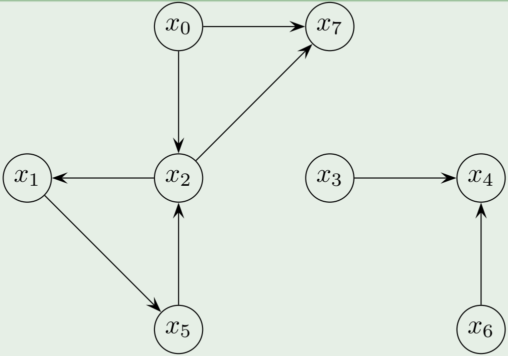
- les composantes connexes qui sont les composantes connexes de G dans lequel on a supprimé l'orientation des arcs

Connexité dans le cas orienté

Dans le cas où $G = (S, A)$ est orienté on distingue :

- les composantes connexes qui sont les composantes connexes de G dans lequel on a supprimé l'orientation des arcs
- les composantes fortement connexes qui sont les sous ensembles C de S maximaux pour l'inclusion tel que pour tout $(x, y) \in C^2$ $x \rightsquigarrow y$ et $y \rightsquigarrow x$.

Exemple



Déterminer les composantes connexes et fortement connexes de ce graphe.