Chapitre 2 : Les graphes

Option Informatique – MP

Lycée Chateaubriand

Table des matières

1 **Les Graphes**
- Les graphes orientés
- Les graphes non orientés

2 **Implémentation**
- Matrices d'adjacence
- Listes d'adjacence
- Passage d'une implémentation à l'autre
- Occupation mémoire

3 **Les parcours**
- Généralités
- Parcours en profondeur (DFS)
- Parcours en largeur (DFS)
- Quelques exemples

4 **Chemin de plus faible poids**
- Les graphes pondérés
- Algorithme de Floyd Warshall
- Algorithme de Dijkstra

Plan

1. Les Graphes
2. Implémentation
3. Les parcours
4. Chemin de plus faible poids

1 Les Graphes
- Les graphes orientés
- Les graphes non orientés

2 Implémentation

3 Les parcours

4 Chemin de plus faible poids

1 Les Graphes
- Les graphes orientés

2 Implémentation

3 Les parcours

4 Chemin de plus faible poids

Définition 1 (Graphes)

Un graphe (orienté) G est la donnée d'un ensemble fini non vide S et d'une partie de A de $S \times S$.

- *Les éléments de S s'appellent les sommets ou noeuds*
- *Les éléments de A s'appellent les arêtes ou arcs*

Définition 1 (Graphes)

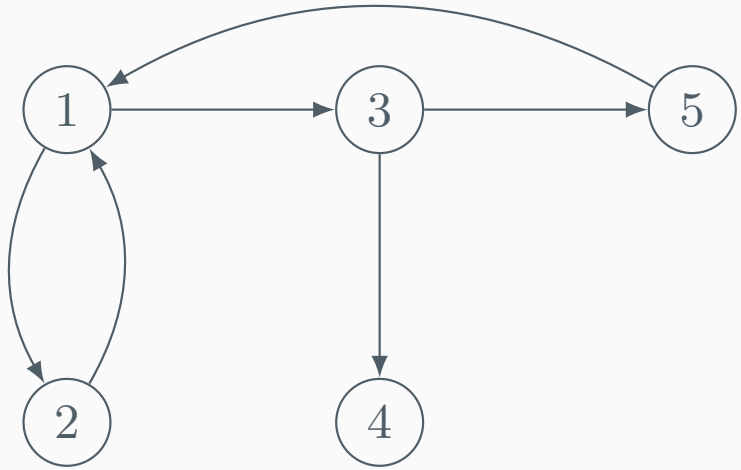
Un graphe (orienté) G est la donnée d'un ensemble fini non vide S et d'une partie de A de $S \times S$.

- *Les éléments de S s'appellent les sommets ou noeuds*
- *Les éléments de A s'appellent les arêtes ou arcs*

- Pour les problèmes de complexité on s'intéressera au nombre de sommets $|S|$ et au nombres d'arêtes $|A|$.
- De manière usuelle on note les graphes (S, A) ou (V, E) où V désigne les sommets (*vertex* en anglais) et E désigne les arêtes (*edges* en anglais).

Un exemple

Le graphe suivant correspond à $S = \{1, 2, 3, 4, 5\}$ et $A = \{(1, 2); (1, 3); (2, 1); (3, 4); (3, 5); (5, 1)\}$



Autres exemples classiques

Autres exemples classiques

- Les arbres sont des cas particuliers de graphes (voir plus loin)

Autres exemples classiques

- Les arbres sont des cas particuliers de graphes (voir plus loin)
- Un graphe peut être utilisé pour modéliser un plan de route / de métro.

Autres exemples classiques

- Les arbres sont des cas particuliers de graphes (voir plus loin)
- Un graphe peut être utilisé pour modéliser un plan de route / de métro.
- Le World Wide Web peut être modélisé par un graphe. Les sommets sont les pages web et les arêtes les liens hypertextes

Définition 2 (Sous-graphe)

Soit $G = (S, A)$ un graphe. On appelle sous-graphe de G un graphe $G' = (S', A')$ où $S' \subset S$ et $A' \subset A \cap (S' \times S')$.

Définition 3

Soit $G = (S, A)$ un graphe.

- *Soit x un sommet du graphe, tous les sommets y tels que (x, y) soit une arête du graphe s'appellent les voisins de x. On note souvent V_x l'ensemble des voisins de x.*
- *Soit x un sommet. Son degré sortant est le nombre de ses voisins. Son degré entrant est le nombre de sommets dont x est un voisin.*

On trouve aussi le terme **arité** pour désigner le degré.

Définition 4

Soit $G = (S, A)$ un graphe.

- *Soit x et y deux sommets du graphe. Un chemin de x à y est une suite $(x_0, \ldots, x_n)$ de sommets telle que $x_0 = x$, $x_n = y$ et pour tout $k \in [\![0, n-1]\!]$, (x_k, x_{k+1}) soit une arête du graphe.*

Définition 4

Soit $G = (S, A)$ un graphe.

- *Soit x et y deux sommets du graphe. Un chemin de x à y est une suite $(x_0, \dots, x_n)$ de sommets telle que $x_0 = x$, $x_n = y$ et pour tout $k \in [\![0, n-1]\!]$, (x_k, x_{k+1}) soit une arête du graphe.*
- *Soit $(x_0, \dots, x_n)$ un chemin. Si $n > 0$ et $x_0 = x_n$ on dit que c'est un cycle.*

Définition 4

Soit $G = (S, A)$ un graphe.

- *Soit x et y deux sommets du graphe. Un chemin de x à y est une suite $(x_0, \ldots, x_n)$ de sommets telle que $x_0 = x$, $x_n = y$ et pour tout $k \in [\![0, n-1]\!]$, (x_k, x_{k+1}) soit une arête du graphe.*
- *Soit $(x_0, \ldots, x_n)$ un chemin. Si $n > 0$ et $x_0 = x_n$ on dit que c'est un cycle.*
- *Soit $(x_0, \ldots, x_n)$ un chemin. Il est dit que longueur n. C'est le nombre d'arêtes du chemin.*

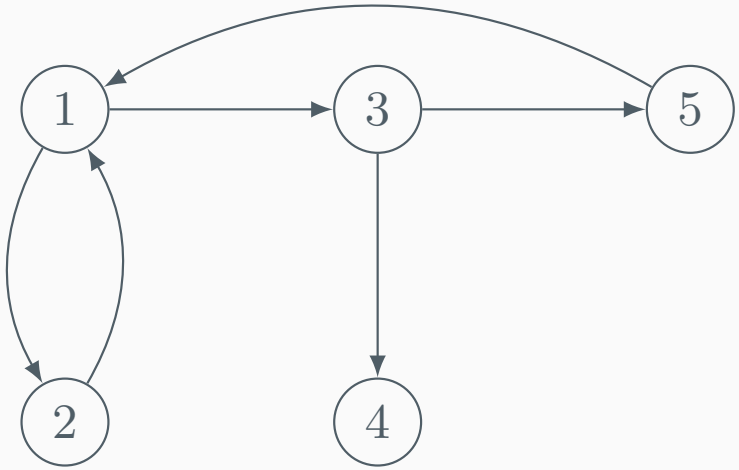
Définition 4

Soit $G = (S, A)$ un graphe.

- *Soit x et y deux sommets du graphe. Un chemin de x à y est une suite $(x_0, \dots, x_n)$ de sommets telle que $x_0 = x$, $x_n = y$ et pour tout $k \in [\![0, n-1]\!]$, (x_k, x_{k+1}) soit une arête du graphe.*
- *Soit $(x_0, \dots, x_n)$ un chemin. Si $n > 0$ et $x_0 = x_n$ on dit que c'est un cycle.*
- *Soit $(x_0, \dots, x_n)$ un chemin. Il est dit que longueur n. C'est le nombre d'arêtes du chemin.*
- *Soit x et y deux sommets, la distance de x à y est la plus petite longueur d'un chemin de x à y.*

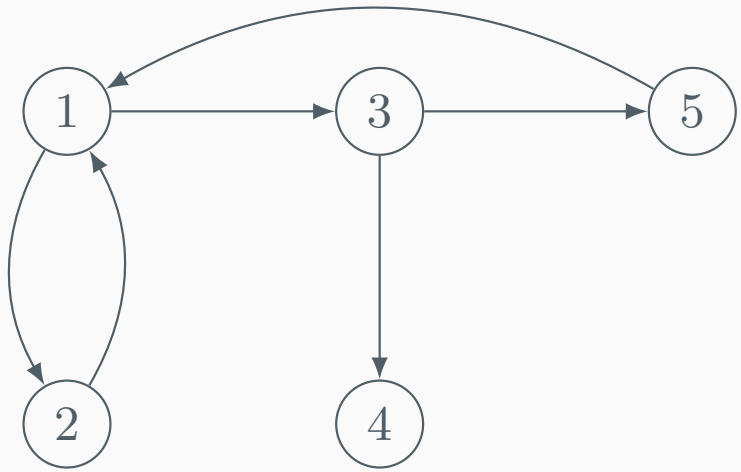
Exemple

On reprend notre graphe



Exemple

On reprend notre graphe



La distance de 1 à 5 est 2, par contre celle de 5 à 1 est 1.

Remarque

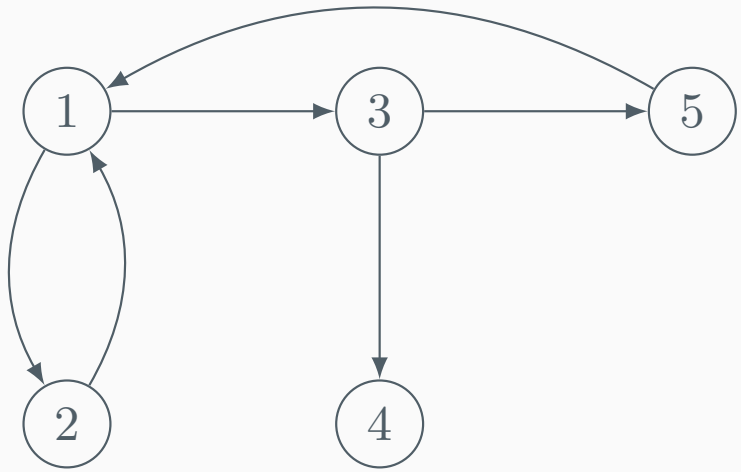
Un chemin a au moins une arête. Par contre, s'il existe une arête (x_0, x_0), qui « boucle » sur le sommet x_0, on peut considérer le cycle (x_0, x_0).

Définition 5

- *Un graphe est dit fortement connexe si pour tout couple (x, y) de sommets du graphe (avec $x \neq y$) il existe un chemin de x vers y **et** un chemin de y vers x.*
- *On appelle composante fortement connexe tout sous-graphe fortement connexe maximal.*

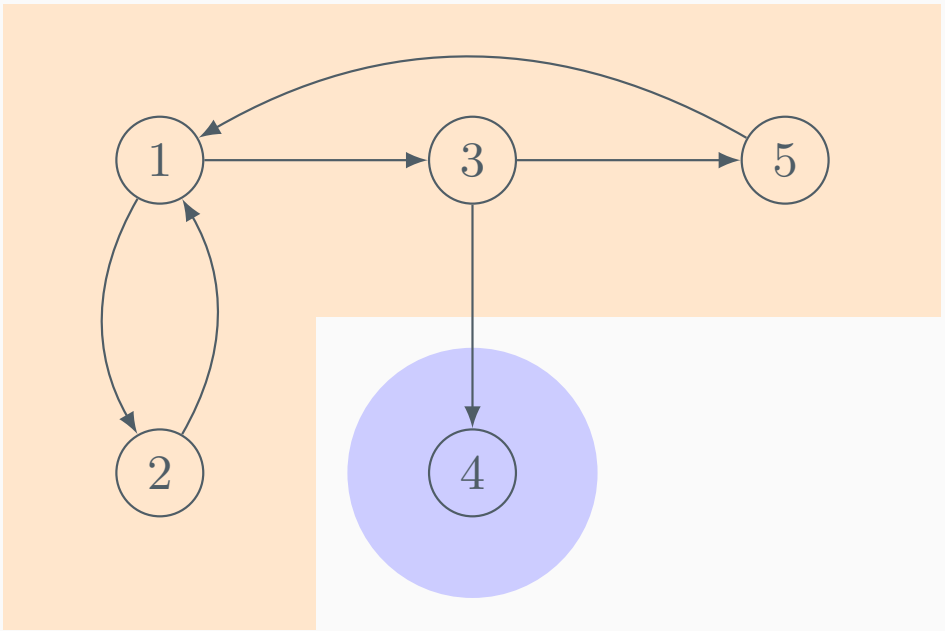
Exercice

Déterminer les composantes fortement connexes du graphe



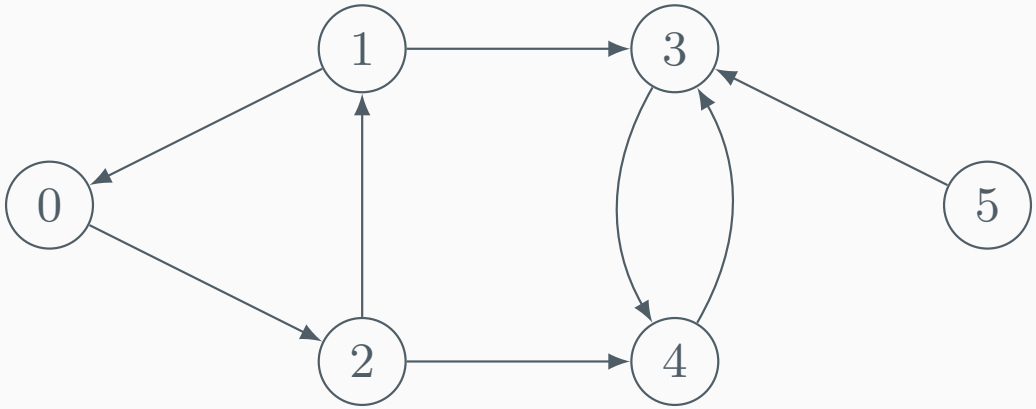
Exercice

Déterminer les composantes fortement connexes du graphe



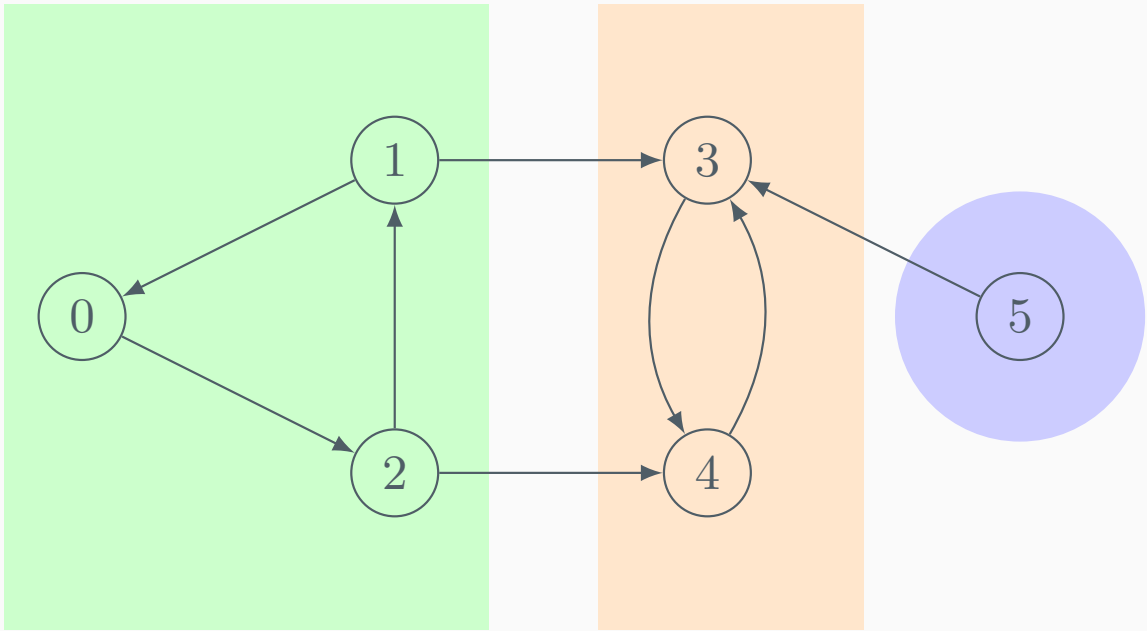
Exercice

Déterminer les composantes fortement connexes du graphe



Exercice

Déterminer les composantes fortement connexes du graphe



1 Les Graphes

Les graphes non orientés

2 Implémentation

3 Les parcours

4 Chemin de plus faible poids

Définition 6

On appelle graphe (non orienté) la donnée d'un ensemble fini non vide S et d'une partie A de l'ensemble des parties de cardinal 2 de S (noté $\mathscr{P}_2(S)$)

Définition 6

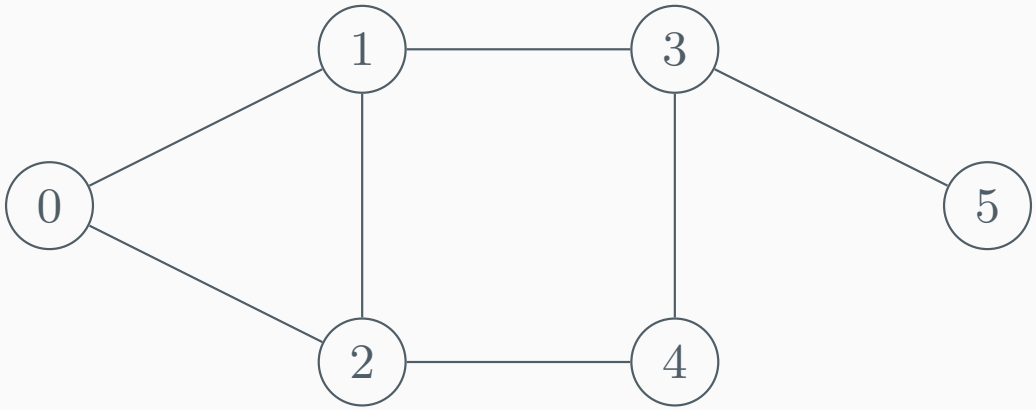
On appelle graphe (non orienté) la donnée d'un ensemble fini non vide S et d'une partie A de l'ensemble des parties de cardinal 2 de S (noté $\mathscr{P}_2(S)$)

Il n'est pas toujours très aisé de travailler avec des parties. On peut garder la représentation des arêtes par des couples. Il faut alors assurer que si $(x, y) \in A$ alors $(y, x) \in A$. C'est cette définition que nous prendrons.

Remarque

Notons que cela signifie en particulier, que l'on ne considère pas de boucles sur un sommet. Contrairement au cas des graphes orientés, dans le cas d'un graphe non-orienté on ne peut pas avoir une arête qui part d'un sommet pour arriver à ce même sommet.

Exemple



Définition 7

Soit $G = (S, A)$ un graphe non orienté.

- *Soit x un sommet du graphe, un sommet y est un voisin de x si $(x, y) \in A$.*
- *Le degré (ou arité) d'un sommet est le nombre de ses voisins. Il n'y a plus lieu de séparer degré entrant et sortant.*
- *Les notions de chemins, cycles, longueur d'un chemin et distance entre deux nœuds se généralisent de manière immédiate. (voir remarque)*

Remarque

Il faut juste ajouter qu'un chemin ne peut pas faire demi-tour, c'est-à-dire que si $(x_0, x_1, \ldots, x_n)$ est un chemin alors pour tout $i \leq n-2$, $x_i \neq x_{i+2}$.

Définition 8

Soit $G = (S, A)$ un graphe non orienté.

- *Le graphe est dit connexe si pour tout couple (x, y) de sommets du graphe il existe un chemin de x vers y. (On ne parle plus de fortement connexe).*

Définition 8

Soit $G = (S, A)$ un graphe non orienté.

- *Le graphe est dit connexe si pour tout couple (x, y) de sommets du graphe il existe un chemin de x vers y. (On ne parle plus de fortement connexe).*
- *On appelle composante connexe tout sous graphe connexe maximal.*

Définition 9 (Arbre)

Soit $G = (S, A)$ un graphe (non orienté) non vide. On dit que c'est un arbre s'il est sans cycle et connexe.

Définition 9 (Arbre)

Soit $G = (S, A)$ un graphe (non orienté) non vide. On dit que c'est un arbre s'il est sans cycle et connexe.

La définition ci-dessus est un peu différente de la définition du chapitre 1. Ici on parle d'arbres qui ne sont pas enracinés.

Par contre, on peut alors choisir n'importe quel sommet x_0 pour la racine de l'arbre, on se retrouve alors avec un arbre enraciné comme au chapitre 1.

Pour tout sommet $x \in S \setminus \{x_0\}$,

- Il existe un chemin qui relie x à x_0 car le graphe est connexe.

Pour tout sommet $x \in S \setminus \{x_0\}$,

- Il existe un chemin qui relie x à x_0 car le graphe est connexe.
- Montrons qu'il n'existe pas deux chemins distincts reliant x à x_0. Supposons par l'absurde qu'il en existe deux. On note

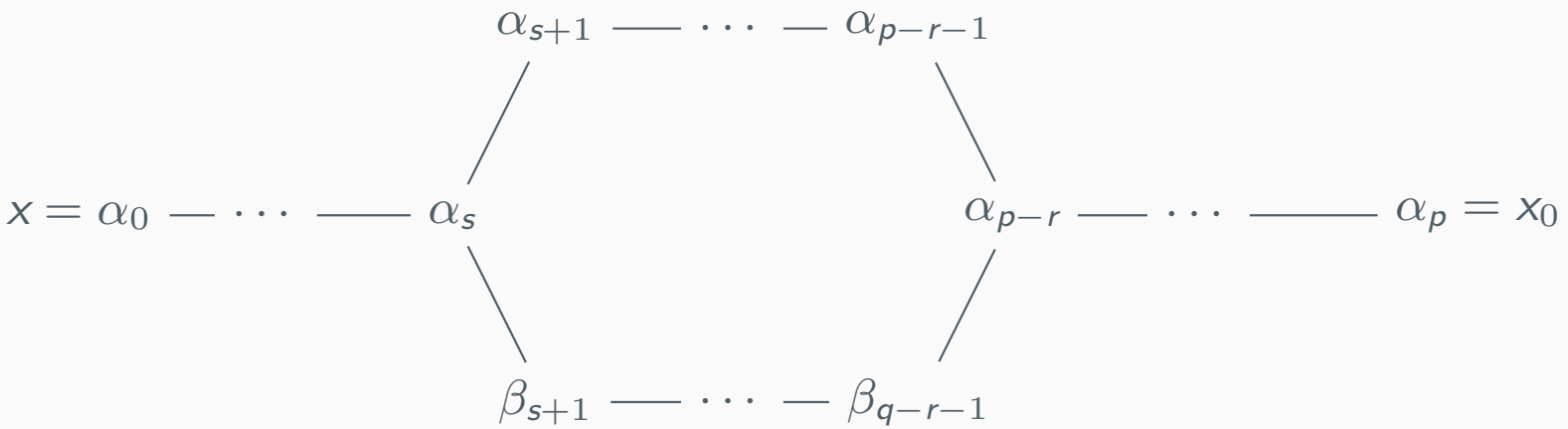
$$x = \alpha_0 - \alpha_1 - \alpha_2 - \cdots - \alpha_p = x_0$$

le premier chemin et

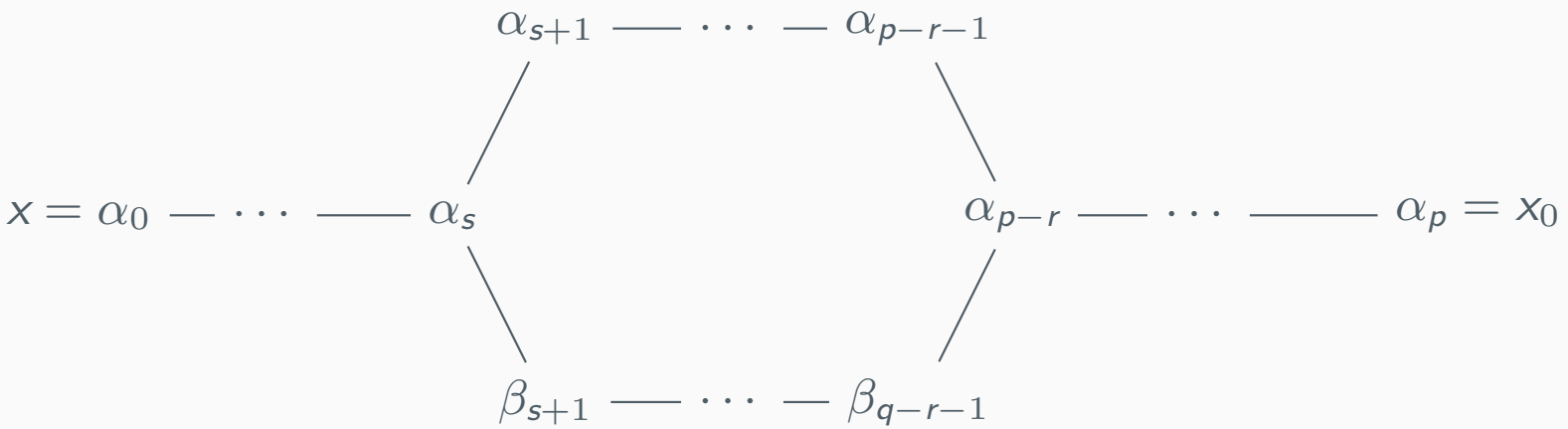
$$x = \beta_0 - \beta_1 - \beta_2 - \cdots - \beta_q = x_0$$

le deuxième.

On considère s le plus grand entier tel que pour $\forall i \leq s, \alpha_i = \beta_i$ et r le plus grand entier tel que $\forall i \leq r, \alpha_{p-i} = \beta_{q-i}$.



On considère s le plus grand entier tel que pour $\forall i \leq s, \alpha_i = \beta_i$ et r le plus grand entier tel que $\forall i \leq r, \alpha_{p-i} = \beta_{q-i}$.

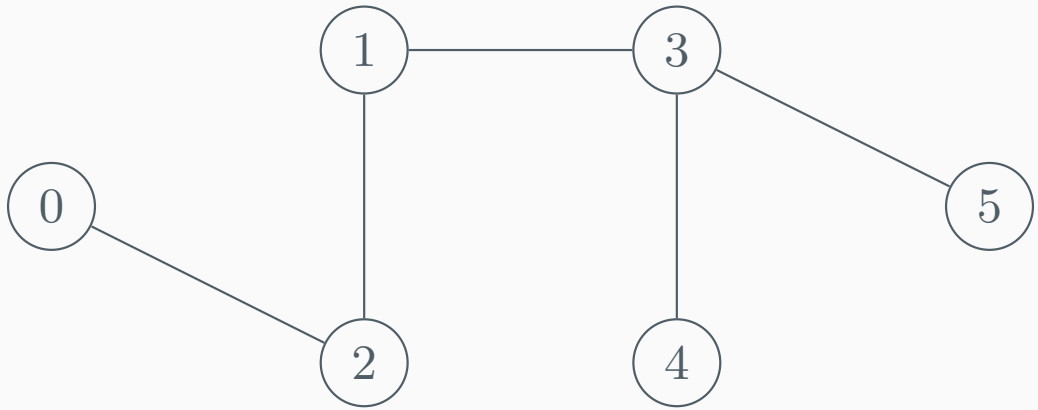


On voit alors qu'il y a un cycle

On peut donc définir le **père** du sommet x comme étant le premier sommet dans l'unique chemin qui relie x à x_0.

Exemple

Pour le graphe suivant (qui est un arbre)

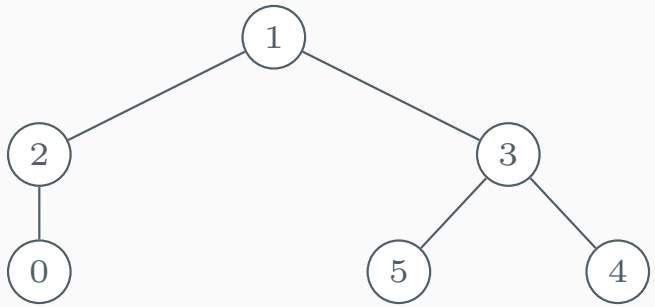


Exemple

En prenant 1 comme racine

Exemple

En prenant 1 comme racine

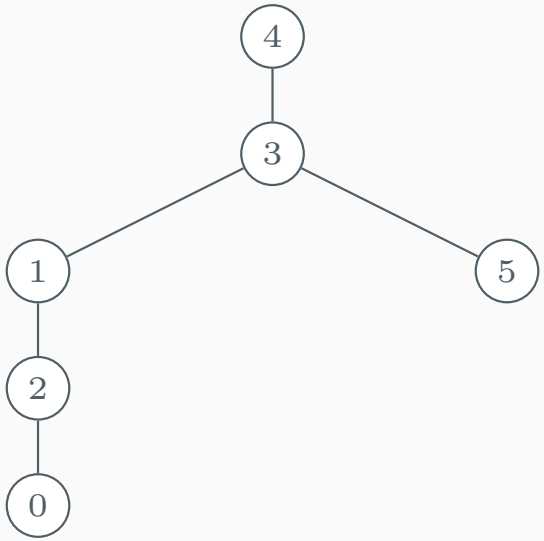


Exemple

En prenant 4 comme racine

Exemple

En prenant 4 comme racine



1 Les Graphes

2 Implémentation

- Matrices d'adjacence
- Listes d'adjacence
- Passage d'une implémentation à l'autre
- Occupation mémoire

3 Les parcours

4 Chemin de plus faible poids

1 Les Graphes

2 Implémentation

Matrices d'adjacence

3 Les parcours

4 Chemin de plus faible poids

Définition

Nous allons présenter les deux implémentations les plus classiques des graphes. Nous travaillerons avec des graphes orientés en précisant les modifications à apporter pour travailler avec des graphes non orientés.

On considère un graphe $G = (S, A)$. On peut le représenter par une matrice $A \in \mathscr{M}_n(\mathbb{Z})$ où $n = |S|$

Définition 10 (Matrice d'adjacence)

Si on note $S = \{x_1, \ldots, x_n\}$. La matrice d'adjacence du graphe est la matrice $A = (a_{ij})$ où

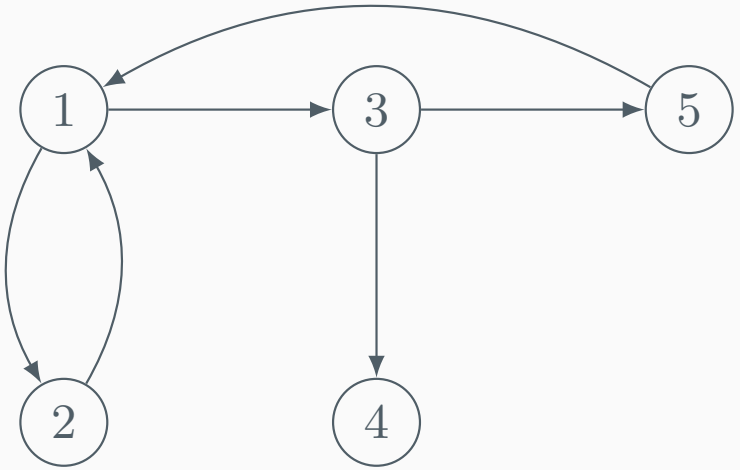
$$a_{ij} = \begin{cases} 1 & \text{si } (x_i, x_j) \in A \\ 0 & \text{sinon} \end{cases}$$

Remarque

Dans le cas d'un graphe orienté, les éléments diagonaux peuvent valoir 1 s'il y a une boucle sur le sommet en question. Dans le cas d'un graphe non-orienté, comme ces boucles sont interdites, les éléments diagonaux valent 0.

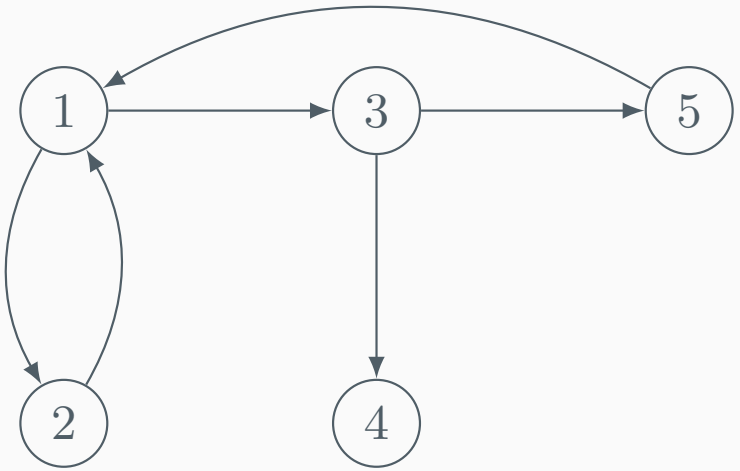
Exemple 1

On considère le graphe



Exemple 1

On considère le graphe



sa matrice est

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

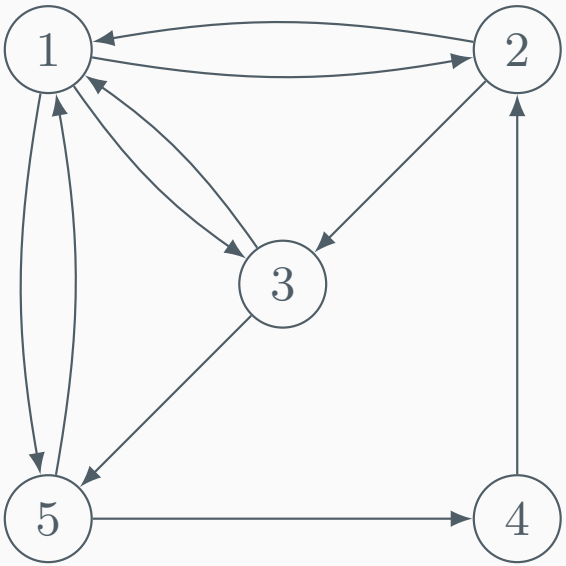
Exemple 2

La matrice est

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Exemple 2

Le graphe est



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Est-il fortement connexe ?

Implémentation

On peut donc implémenter un graphe par :

Implémentation

On peut donc implémenter un graphe par :

```
type graphe = int array array
```

ou

```
type graphe = bool array array
```

Attention

Attention

Dans la description mathématique on numérote les sommets de 1 à n, dans la description informatique on les numérote de 0 à $n-1$.

Exercice

Ecrire les fonctions `ajoute_arete g i j` et `supprime_arete g i j` qui ajoute ou supprime l'arête (x_i, x_j).

Exercice

Ecrire les fonctions `ajoute_arete g i j` et `supprime_arete g i j` qui ajoute ou supprime l'arête (x_i, x_j).

```
let ajoute_arete g i j = g.(i).(j) <- true;;

let supprime_arete g i j = g.(i).(j) <- false;;
```

Graphes non orientés

Un graphe non orienté est un graphe dont la matrice d'adjacence est symétrique

Ecrire une fonction `symetrique : int array array -> bool`qui renvoie un booléen selon que la matrice est symétrique ou non.

```
let symetrique g =
  let n = Array.length g in
  let sym = ref true in
  let i = ref 0 in
  let j = ref 1 in
  while !i <= (n-1) && !sym do
    j  := !i+1;
    while !j <= (n-1) && !sym do
      sym  := g.(!i).(!j) = g.(!j).(!i);
      j  := !j+1;
    done;
    i := !i+1;
  done;
  !sym;;
```

1 Les Graphes

2 Implémentation

Listes d'adjacence

3 Les parcours

4 Chemin de plus faible poids

Definiton

On considère un graphe (S, A). On peut le représenter par un tableau de listes.

```
type graphe = (int list) array
```

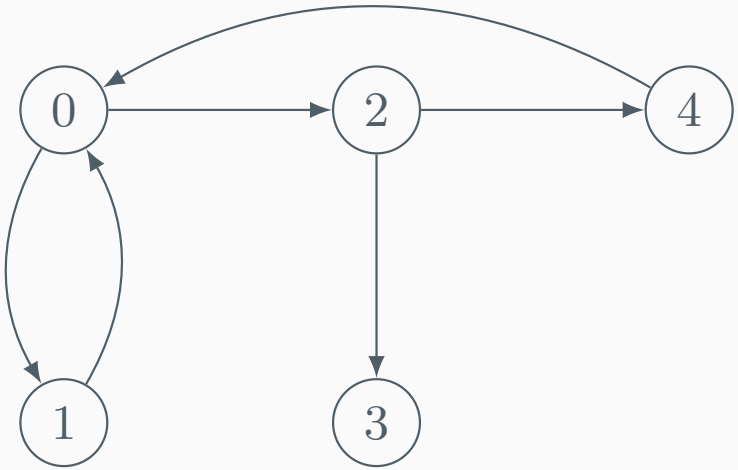
On considère un graphe (S, A). On peut le représenter par un tableau de listes.

```
type graphe = (int list) array
```

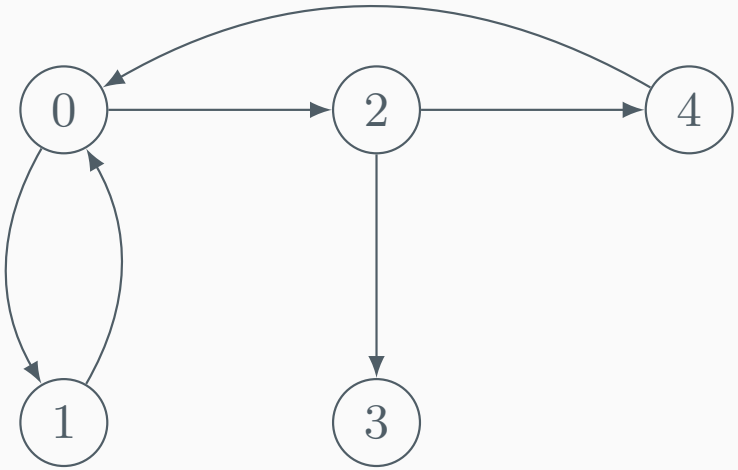
Définition 11 (Listes d'adjacences)

Si on note $S = \{x_0, \ldots, x_{n-1}\}$ on appelle listes d'adjacences du graphe le tableau t de listes où pour tout i compris entre 0 et $n-1$, $t.(i)$ est la liste des voisins de x_i.

Exemple



Exemple



Ses listes d'adjacences sont

i	0	1	2	3	4
`t.(i)`	$[1;2]$	$[0]$	$[3,4]$	$[]$	$[0]$

Exercice

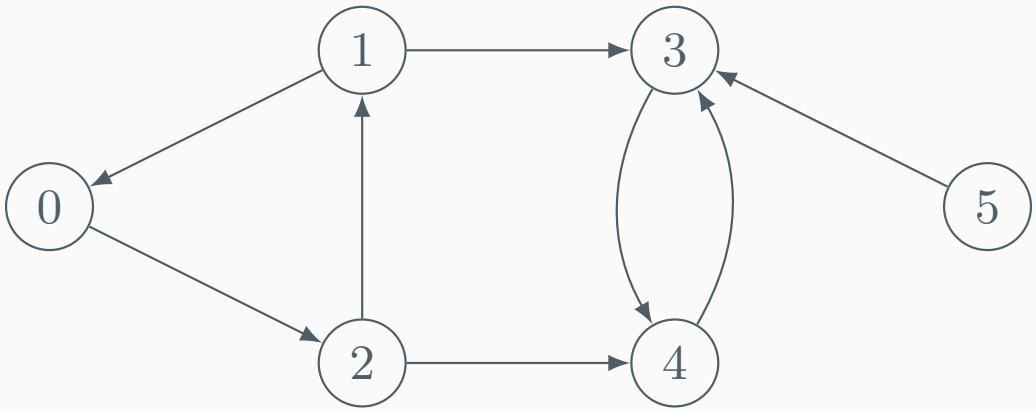
Déterminer le graphe donné par les listes d'adjacences :

i	0	1	2	3	4	5
`t.(i)`	$[2]$	$[0;3]$	$[1;4]$	$[4]$	$[3]$	$[3]$

Exercice

Déterminer le graphe donné par les listes d'adjacences :

i	0	1	2	3	4	5
`t.(i)`	$[2]$	$[0;3]$	$[1;4]$	$[4]$	$[3]$	$[3]$



Exercice

Ecrire les fonctions `ajoute_arete g i j` et `supprime_arete g i j`

Exercice

Ecrire les fonctions `ajoute_arete g i jet supprime_arete g i j`

```
let ajoute_arete_liste t i j =
  let rec ajoute_liste l x =
    match l with
      |[] -> [x]
      |t::q -> if t=x then t::q else t::(ajoute_liste q x)
  in
  t.(i) <- ajoute_liste t.(i) j;;
```

Exercice

Ecrire les fonctions `ajoute_arete g i jet supprime_arete g i j`

```
let ajoute_arete_liste t i j =
  let rec ajoute_liste l x =
    match l with
      |[] -> [x]
      |t::q -> if t=x then t::q else t::(ajoute_liste q x)
  in
  t.(i) <- ajoute_liste t.(i) j;;
```

```
let suppr_arete_liste t i j =
  let rec suppr_liste l x =
    match l with
      |[] -> []
      |t::q -> if t=x then q else t::(suppr_liste q x)
  in
  t.(i) <- suppr_liste t.(i) j;;
```

Graphes non orientés

On peut représenter un graphe non orienté en s'assurant que si i est dans `t.(j)` alors j est dans `t.(i)`.

Graphes non orientés

Ecrire une fonction `valid_non_oriente t`qui prend un graphe (donné par listes d'adjacences) et renvoi un booléen selon qu'il représente bien un graphe non orienté ou non

Ecrire une fonction `valid_non_oriente t`qui prend un graphe (donné par listes d'adjacences) et renvoi un booléen selon qu'il représente bien un graphe non orienté ou non

```
let val_non_oriente t =
  let n = Array.lenght t in
  let oriente = ref true in
  let rec aux l i = match l with
    |[] -> true
    |j::q -> if existe i t.(j) then aux q i else false
    (*La fonction aux l i vérifie que la liste t.(i) vérifie
    les conditions*)
  in let i = ref 0 in
     while !i < n && !oriente do
       oriente  := aux t.(!i) !i;
       i := !i+1
     done;
  !oriente;;
```

1 Les Graphes

2 Implémentation

Passage d'une implémentation à l'autre

3 Les parcours

4 Chemin de plus faible poids

Des matrices aux listes

Écrire la fonction `mat_to_liste`

Des matrices aux listes

Écrire la fonction `mat_to_liste`

```
let mat_to_liste a =
  let n = Array.length a in
  let t = Array.make n [] in
  for i = 0 to (n-1) do
        for j = 0 to (n-1) do
          if a.(i).(j) then t.(i) <- j::t.(i)
      done;
  done;
   t;;
```

Des listes aux matrices

Écrire la fonction `liste_to_mat`

Des listes aux matrices

Écrire la fonction `liste_to_mat`

```
let liste_to_mat t =
    let n = Array.length t in
    let a = Array.make_matrix n n 0 in
    let rec aux l i = match l with
      |[] -> ()
      |t ::q -> a.(i).(t) <-1 ; aux q i
    in
    for k = 0 to (n-1) do
      aux t.(k) k;
    done;
    a;;
```

Occupation mémoire

1 Les Graphes

2 Implémentation

Occupation mémoire

3 Les parcours

4 Chemin de plus faible poids

Occupation mémoire

On dispose d'un graphe avec $|S|$ sommets et $|A|$ arêtes.

- La représentation en mémoire du graphe par listes d'adjacence nécessite $|S|$ listes dont la somme des longueurs est $|A|$ (ou $2|A|$ pour un graphe non-orienté). La complexité en espace est donc

Occupation mémoire

On dispose d'un graphe avec $|S|$ sommets et $|A|$ arêtes.

- La représentation en mémoire du graphe par listes d'adjacence nécessite $|S|$ listes dont la somme des longueurs est $|A|$ (ou $2|A|$ pour un graphe non-orienté). La complexité en espace est donc

$$O(|S| + |A|).$$

- La représentation en mémoire du graphe par matrice d'adjacence nécessite une matrice de taille $|S|^2$. La complexité en espace est donc

Occupation mémoire

On dispose d'un graphe avec $|S|$ sommets et $|A|$ arêtes.

- La représentation en mémoire du graphe par listes d'adjacence nécessite $|S|$ listes dont la somme des longueurs est $|A|$ (ou $2|A|$ pour un graphe non-orienté). La complexité en espace est donc

$$O(|S| + |A|).$$

- La représentation en mémoire du graphe par matrice d'adjacence nécessite une matrice de taille $|S|^2$. La complexité en espace est donc

$$O(|S|^2).$$

Occupation mémoire

On dispose d'un graphe avec $|S|$ sommets et $|A|$ arêtes.

- La représentation en mémoire du graphe par listes d'adjacence nécessite $|S|$ listes dont la somme des longueurs est $|A|$ (ou $2|A|$ pour un graphe non-orienté). La complexité en espace est donc

$$O(|S| + |A|).$$

- La représentation en mémoire du graphe par matrice d'adjacence nécessite une matrice de taille $|S|^2$. La complexité en espace est donc

$$O(|S|^2).$$

Dans la majorité des cas, sauf pour les graphes très denses (avec beaucoup d'arêtes), on préférera les listes d'adjacences.

1 Les Graphes

2 Implémentation

3 Les parcours

- Généralités
- Parcours en profondeur (DFS)
- Parcours en largeur (DFS)
- Quelques exemples

4 Chemin de plus faible poids

1 Les Graphes

2 Implémentation

3 Les parcours

Généralités

4 Chemin de plus faible poids

Parcours en profondeur (DFS)

1 Les Graphes

2 Implémentation

3 **Les parcours**

- Parcours en profondeur (DFS)

4 Chemin de plus faible poids

1 Les Graphes

2 Implémentation

3 Les parcours

Parcours en largeur (DFS)

4 Chemin de plus faible poids

1 Les Graphes

2 Implémentation

3 Les parcours

Quelques exemples

4 Chemin de plus faible poids

4 Chemin de plus faible poids

- Les graphes pondérés
- Algorithme de Floyd Warshall
- Algorithme de Dijkstra

1 Les Graphes

2 Implémentation

3 Les parcours

4 Chemin de plus faible poids

- Les graphes pondérés

4 Chemin de plus faible poids

Algorithme de Floyd Warshall

4 Chemin de plus faible poids

Algorithme de Dijkstra