Graphes : révisions de MP2I

Quentin Fortier

October 10, 2024

Définitions

Définition

Un graphe (non orienté) est un couple $G = (S, A)$ où :

- S est un ensemble fini (de sommets)
- A est un ensemble dont chaque élément, appelé arête, est un ensemble de 2 sommets

Un graphe orienté est un couple (S, A) où A est un ensemble d'arcs (couples de sommets).

Définitions

Définition

Un graphe (non orienté) est un couple $G = (S, A)$ où :

- S est un ensemble fini (de sommets)
- A est un ensemble dont chaque élément, appelé arête, est un ensemble de 2 sommets

Un graphe orienté est un couple (S, A) où A est un ensemble d'arcs (couples de sommets).

Exercice

Soit S un ensemble de n sommets.

1. Combien y a t-il de graphes non-orientés ayant S comme ensemble de sommets ?
2. Combien y a t-il de graphes orienté ayant S comme ensemble de sommets ?

Définitions

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

- Si $e = \{u, v\} \in A$ on dit que u et v sont les extrémités de e et que u et v sont voisins (ou adjacents).
- Le degré d'un sommet $v \in S$, noté $\deg(v)$, est son nombre de voisins. Si $\deg(v) = 1$, v est une feuille.
  Pour un graphe orienté, on note $\deg^-(v)$ et $\deg^+(v)$ les degrés entrants et sortants de v.
- Si $e \in A$, on note $G - e$ le graphe obtenu en supprimant e : $G - e = (S, A - \{e\})$.
- Si $v \in S$, on note $G - v$ le graphe obtenu en supprimant v : $G - v = (S - \{v\}, A')$, où A' est l'ensemble des arêtes de A n'ayant pas v comme extrémité.

Définitions

Exercice

Un graphe est simple s'il ne contient pas de boucle (arête reliant un sommet à lui-même) ni d'arête multiple.
Montrer que dans tout graphe (non orienté, simple) avec au moins deux sommets, il existe deux sommets de même degré.

Définitions

Formule des degrés (HP)

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

$$\sum_{v \in S} \deg(v) = 2|A|$$

Définitions

Formule des degrés (HP)

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

$$\sum_{v \in S} \deg(v) = 2|A|$$

Preuve (par double comptage) :
On divise chaque arête en deux demi-arêtes. Le nombre de demi-arêtes est égal à :

1. $2\ |A|$ car chaque arête a 2 extrémités.
2. $\sum_{v \in S} \deg(v)$ car chaque sommet v est extrémité de $\deg(v)$ arêtes.

Autre preuve : récurrence sur le nombre d'arêtes.

Pour un graphe orienté :

Définitions

Formule des degrés (HP)

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

$$\sum_{v \in S} \deg(v) = 2|A|$$

Preuve (par double comptage) :
On divise chaque arête en deux demi-arêtes. Le nombre de demi-arêtes est égal à :

1. $2\ |A|$ car chaque arête a 2 extrémités.
2. $\sum_{v \in S} \deg(v)$ car chaque sommet v est extrémité de $\deg(v)$ arêtes.

Autre preuve : récurrence sur le nombre d'arêtes.

Pour un graphe orienté : $\sum \deg^+(v) = \sum \deg^-(v) = 2|A|$.

Définitions

Lemme des poignées de main (Handshake lemma)

Tout graphe possède un nombre pair de sommets de degrés impairs.

Preuve :

Définitions

Lemme des poignées de main (Handshake lemma)

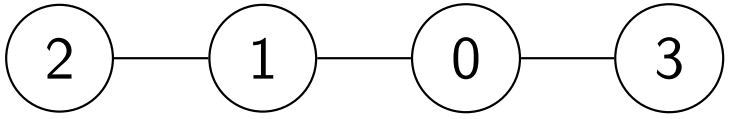
Tout graphe possède un nombre pair de sommets de degrés impairs.

Preuve :

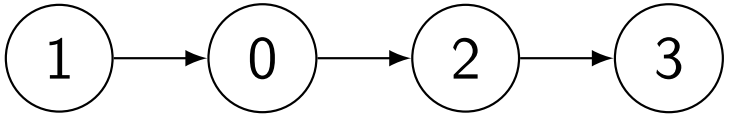
$$\underbrace{\sum_{\deg(v) \text{ pair}} \deg(v)}_{\text{pair}} + \sum_{\deg(v) \text{ impair}} \deg(v) = \underbrace{2|A|}_{\text{pair}}$$

Connexité

Un chemin de G est une suite de sommets $v_0, v_1, ..., v_k$ telle que :
$\forall i \in \{0, ..., k-1\}$, $\{v_i, v_{i+1}\} \in A$.
C'est un cycle si $v_0 = v_k$.



Chemin (non orienté) entre 2 et 3



Chemin (orienté) de 1 à 3

Un chemin est élémentaire s'il ne passe pas deux fois par le même sommet.
La longueur d'un chemin est son nombre d'arêtes.
La distance de u à v est la plus petite longueur d'un chemin de u à v (∞ s'il n'y a pas de chemin).

Connexité

Définition

Un graphe $G = (S, A)$ est connexe si pour tout couple de sommets $u, v \in S$, il existe un chemin de u à v.

Remarque : cette définition est aussi valable pour les graphes orientés (pour tout sommets u, v, il faut un chemin de u à v et un chemin de v à u). On parle alors de graphe fortement connexe.

Connexité

Théorème

Un graphe connexe à n sommets possède au moins $n - 1$ arêtes.

Connexité

Théorème

Un graphe connexe à n sommets possède au moins $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe connexe à n sommets a au moins $n-1$ arêtes ».

1. Un graphe à 1 sommet possède 0 arête.

Connexité

Théorème

Un graphe connexe à n sommets possède au moins $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe connexe à n sommets a au moins $n-1$ arêtes ».

1. Un graphe à 1 sommet possède 0 arête.
2. Supposons $\mathcal{H}(n)$. Soit $G=(S,A)$ un graphe connexe à $n+1$ sommets.

Connexité

Théorème

Un graphe connexe à n sommets possède au moins $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe connexe à n sommets a au moins $n-1$ arêtes ».

1. Un graphe à 1 sommet possède 0 arête.
2. Supposons $\mathcal{H}(n)$. Soit $G=(S,A)$ un graphe connexe à $n+1$ sommets.
   - Si G a un sommet v de degré 1 alors

Connexité

Théorème

Un graphe connexe à n sommets possède au moins $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe connexe à n sommets a au moins $n-1$ arêtes ».

1. Un graphe à 1 sommet possède 0 arête.
2. Supposons $\mathcal{H}(n)$. Soit $G=(S,A)$ un graphe connexe à $n+1$ sommets.
   - Si G a un sommet v de degré 1 alors $G-v$ est un graphe connexe à n sommets donc, par $\mathcal{H}(n)$, $G-v$ a au moins $n-1$ arêtes.

Connexité

Théorème

Un graphe connexe à n sommets possède au moins $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe connexe à n sommets a au moins $n-1$ arêtes ».

1. Un graphe à 1 sommet possède 0 arête.
2. Supposons $\mathcal{H}(n)$. Soit $G=(S,A)$ un graphe connexe à $n+1$ sommets.
   - Si G a un sommet v de degré 1 alors $G-v$ est un graphe connexe à n sommets donc, par $\mathcal{H}(n)$, $G-v$ a au moins $n-1$ arêtes. Donc G a au moins n arêtes.
   - Sinon,

Connexité

Théorème

Un graphe connexe à n sommets possède au moins $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe connexe à n sommets a au moins $n-1$ arêtes ».

1. Un graphe à 1 sommet possède 0 arête.
2. Supposons $\mathcal{H}(n)$. Soit $G=(S,A)$ un graphe connexe à $n+1$ sommets.
   - Si G a un sommet v de degré 1 alors $G-v$ est un graphe connexe à n sommets donc, par $\mathcal{H}(n)$, $G-v$ a au moins $n-1$ arêtes. Donc G a au moins n arêtes.
   - Sinon, tous les sommets de G sont de degré ≥ 2.
     Alors $2|A| = \sum_{v\in S} \deg(v) \geq 2(n+1) \geq 2n$.
     Donc $|A| \geq n$, ce qui montre $\mathcal{H}(n+1)$.

Connexité

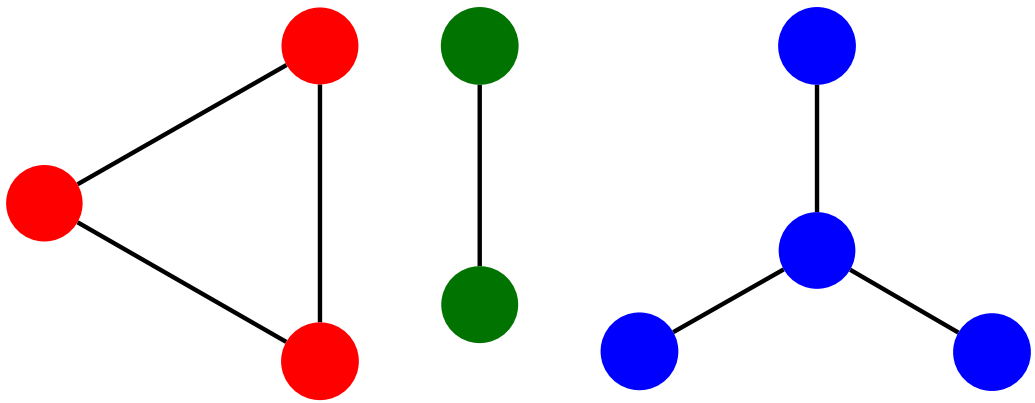
Définition

Soit $G = (S, A)$ un graphe non-orienté et $S' \subset S$.
On dit que S' est une composante connexe de G si S' est connexe et maximal pour cette propriété (c'est-à-dire que si $S' \subset S'' \subset S$, alors S'' n'est pas connexe).

On peut considérer la relation d'équivalence suivante :

$$u \sim v \iff \text{il existe un chemin entre } u \text{ et } v$$

Les composantes connexes de G sont alors les classes d'équivalences pour $\sim$.



Connexité

Exercice

Soit $G = (S, A)$ un graphe non-orienté. On note $\overline{G} = (S, \overline{A})$ le graphe complémentaire de G où $\overline{A} = \{\{u, v\} \mid \{u, v\} \notin A\}$.
Montrer que G ou $\overline{G}$ est connexe. Est-il possible que les deux soient connexes ?

Connexité

Si $G = (S, A)$ est orienté, la relation suivante est une relation d'équivalence :

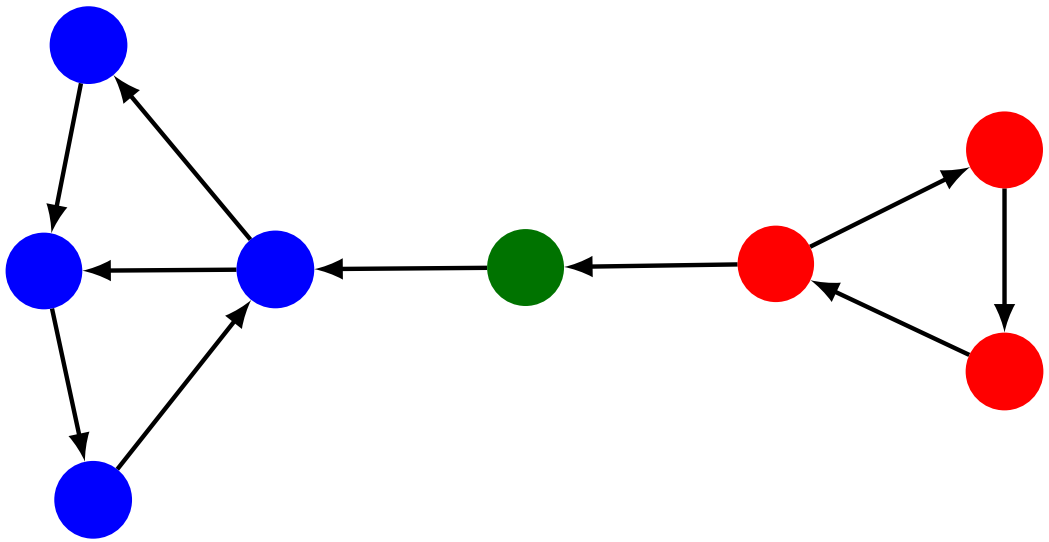
$$u \leftrightsquigarrow v \iff u \rightsquigarrow v \text{ et } v \rightsquigarrow u$$

Connexité

Si $G = (S, A)$ est orienté, la relation suivante est une relation d'équivalence :

$$u \leftrightsquigarrow v \iff u \rightsquigarrow v \text{ et } v \rightsquigarrow u$$

Les classes d'équivalences $S/\leftrightsquigarrow$ sont appelées composantes fortement connexes.



Un graphe orienté avec 3 composantes fortement connexes.

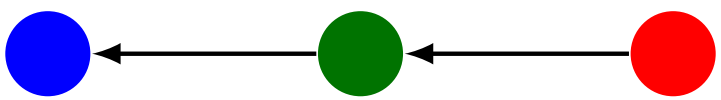
Connexité

Si $G = (S, A)$ est orienté, « $u \rightsquigarrow v \iff$ il existe un chemin de u à v » n'est pas une relation d'équivalence.

Par contre la relation suivante est une relation d'équivalence :

$$u \nleftrightsquigarrow v \iff u \rightsquigarrow v \text{ et } v \rightsquigarrow u$$

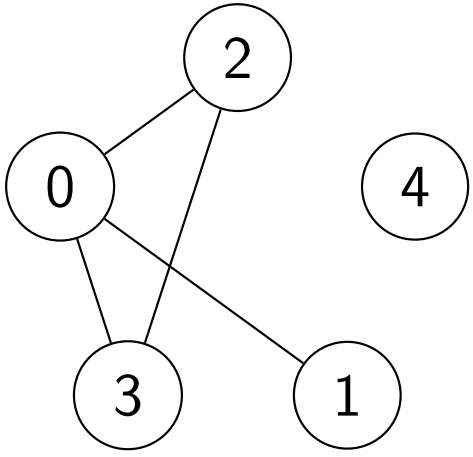
Les classes d'équivalences $S/\nleftrightsquigarrow$ sont appelées composantes fortement connexes.



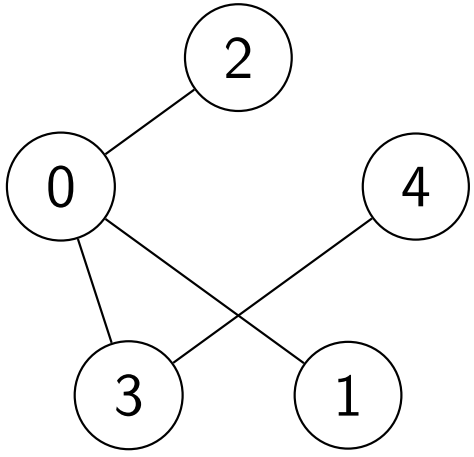
Le graphe des composantes fortement connexes est acyclique.

Connexité

Un graphe est acyclique (ou : sans cycle) s'il ne contient pas de cycle.



Graphe contenant un cycle



Graphe acyclique

Question

Quel est le nombre maximum d'arêtes d'un graphe acyclique à n sommets ?

Connexité

Montrons d'abord :

Lemme

Tout graphe acyclique contient un sommet de degré au plus 1.

Connexité

Montrons d'abord :

Lemme

Tout graphe acyclique contient un sommet de degré au plus 1.

Preuve : Supposons que tous les sommets soient de degrés ≥ 2.
Faisons partir un chemin depuis un sommet quelconque en visitant à chaque étape le sommet adjacent différent du prédécesseur (possible car les degrés sont ≥ 2).
Comme le nombre de sommets est fini, ce chemin revient nécessairement sur un sommet déjà visité, ce qui donne un cycle.

Connexité

Montrons d'abord :

Lemme

Tout graphe acyclique contient un sommet de degré au plus 1.

Preuve : Supposons que tous les sommets soient de degrés ≥ 2.
Faisons partir un chemin depuis un sommet quelconque en visitant à chaque étape le sommet adjacent différent du prédécesseur (possible car les degrés sont ≥ 2).
Comme le nombre de sommets est fini, ce chemin revient nécessairement sur un sommet déjà visité, ce qui donne un cycle.

Remarque : tout graphe acyclique avec au moins 2 sommets contient 2 sommets de degré ≤ 1.

Connexité

Théorème

Un graphe acyclique à n sommets possède au plus $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe acyclique à n sommets a au plus $n-1$ arêtes ».

Connexité

Théorème

Un graphe acyclique à n sommets possède au plus $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe acyclique à n sommets a au plus $n-1$ arêtes ».

1. Un graphe à 1 sommet a 0 arête.
2. Supposons $\mathcal{H}(n)$. D'après le lemme, un graphe G acyclique à $n+1$ sommets possède un sommet v de degré ≤ 1.

Connexité

Théorème

Un graphe acyclique à n sommets possède au plus $n-1$ arêtes.

Soit $\mathcal{H}(n)$: « un graphe acyclique à n sommets a au plus $n-1$ arêtes ».

1. Un graphe à 1 sommet a 0 arête.
2. Supposons $\mathcal{H}(n)$. D'après le lemme, un graphe G acyclique à $n+1$ sommets possède un sommet v de degré ≤ 1.
   Comme G est acyclique, $G-v$ l'est aussi et a au plus $n-1$ arêtes, par $\mathcal{H}(n)$.
   Donc G a au plus $n-1+\deg(v) \leq n$ arêtes, ce qui montre $\mathcal{H}(n+1)$.

Arbre

Théorème / définition

Soit $G = (S, A)$ un graphe non-orienté à n sommets.
G est un arbre s'il vérifie l'une des conditions équivalentes :

- G est connexe acyclique.
- G est connexe et a $n - 1$ arêtes.
- G est acyclique et a $n - 1$ arêtes.
- Il existe un unique chemin entre 2 sommets quelconques de G.

Remarque : il n'y a pas de racine, contrairement à un arbre

```
V | N of 'a * 'a arbre * 'a arbre.
```

Représentation de graphe

Représentations classiques d'un graphe :

1. Par matrice d'adjacence
2. Par liste d'adjacence
3. Par dictionnaire d'adjacence (moins courant)

Matrice d'adjacence

Un graphe non orienté $G = (S, A)$ où $S = \{0, ..., n-1\}$ peut être représenté par une matrice d'adjacence M de taille $n \times n$ telle que :

- $M_{i,j} = 1 \iff \{i, j\} \in A$
- $M_{i,j} = 0 \iff \{i, j\} \notin A$

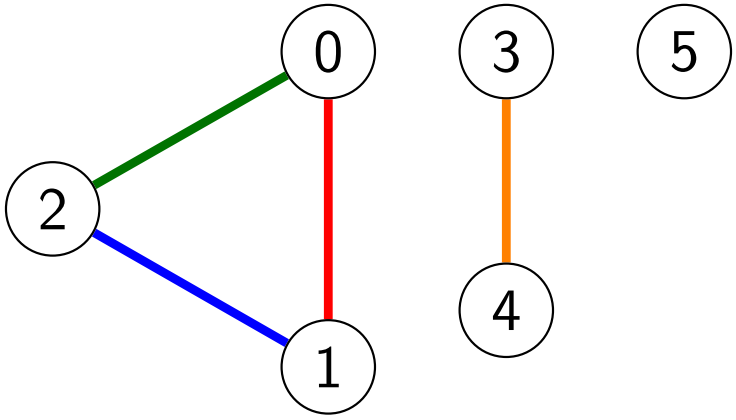
Remarque : M est symétrique.

Pour un graphe orienté (S, A) :

- $M_{i,j} = 1 \iff (i, j) \in A$
- $M_{i,j} = 0 \iff (i, j) \notin A$

M n'est pas symétrique (a priori).

Matrice d'adjacence



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

En OCaml :

```
let g = Array.make_matrix n p 0 in
g.(0).(1) <- 1; (* met une arête entre 0 et 1 *)
...
```

Matrice d'adjacence

Exercice

Soit $G = (S, A)$ un graphe orienté représenté par une matrice d'adjacence `m`.
Écrire une fonction `trou_noir m` renvoyant en $O(|S|)$ un sommet t vérifiant :

- $\forall u \neq t : (u, t) \in A$
- $\forall v \neq t : (t, v) \notin A$

Si G n'a pas de trou noir, on pourra renvoyer `-1`.

Matrice d'adjacence

Exercice

Si $A = (a_{u,v})$ est une matrice d'adjacence d'un graphe à n sommets, que représente les coefficients de $A^k = (a_{u,v}^{(k)})$?

Matrice d'adjacence

Exercice

Si $A = (a_{u,v})$ est une matrice d'adjacence d'un graphe à n sommets, que représente les coefficients de $A^k = (a_{u,v}^{(k)})$?

Pour $k = 2$:

$$a_{u,v}^2 = \sum_{w=0}^{n-1} a_{u,w} a_{w,v}$$

Matrice d'adjacence

Exercice

Si $A = (a_{u,v})$ est une matrice d'adjacence d'un graphe à n sommets, que représente les coefficients de $A^k = (a^{(k)}_{u,v})$?

Pour $k = 2$:

$$a^2_{u,v} = \sum_{w=0}^{n-1} a_{u,w} a_{w,v}$$

$$a_{u,w} a_{w,v} = 1 \iff u \to w \to v \text{ est un chemin}$$

$a^2_{u,v}$ est le nombre de chemins de longueur 2 de u à v.

Matrice d'adjacence

Exercice

Si $A = (a_{u,v})$ est une matrice d'adjacence d'un graphe à n sommets, que représente les coefficients de $A^k = (a_{u,v}^{(k)})$?

$$a_{u,v}^{(k)} = \sum_{w=0}^{n-1} a_{u,w}^{(k-1)} a_{w,v}$$

Matrice d'adjacence

Exercice

Si $A = (a_{u,v})$ est une matrice d'adjacence d'un graphe à n sommets, que représente les coefficients de $A^k = (a_{u,v}^{(k)})$?

$$a_{u,v}^{(k)} = \sum_{w=0}^{n-1} a_{u,w}^{(k-1)} a_{w,v}$$

Par récurrence sur k :

$$\boxed{a_{u,v}^{(k)} = \text{nombre de chemins de longueur } k \text{ de } u \text{ à } v}$$

Remarque : c'est vrai aussi bien pour les graphes orientés que non-orientés.

Liste d'adjacence

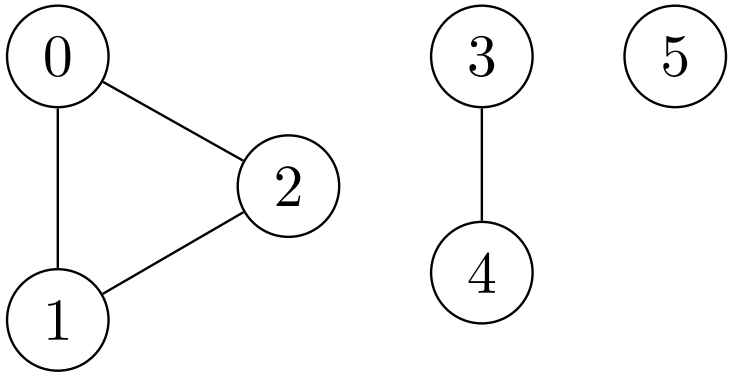
La représentation par liste d'adjacence consiste à stocker, pour chaque sommet, la liste de ses voisins.

Liste d'adjacence

La représentation par liste d'adjacence consiste à stocker, pour chaque sommet, la liste de ses voisins.

On utilise pour cela un tableau `g` de listes (`int list array`), telle que `g.(u)` soit la liste des voisins de `u`.

Liste d'adjacence



Matrice d'adjacence

```
[|[|0; 1; 1; 0; 0; 0|];
 [|1; 0; 1; 0; 0; 0|];
 [|1; 1; 0; 0; 0; 0|];
 [|0; 0; 0; 0; 1; 0|];
 [|0; 0; 0; 1; 0; 0|];
 [|0; 0; 0; 0; 0; 0|]|]
```

Liste d'adjacence

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

Comparaison matrice d'adjacence / liste d'adjacence

Pour un graphe orienté à n sommets et m arêtes :

Opération	Matrice d'adjacence	Liste d'adjacence
ajouter arête	$\mathsf{O}(1)$	$\mathsf{O}(1)$
existence arête	$\mathsf{O}(1)$	$\mathsf{O}(\deg^+(u))$
voisins de u	$\mathsf{O}(n)$	$\mathsf{O}(\deg^+(u))$
espace	$\mathsf{O}(\lvert S\rvert^2)$	$\mathsf{O}(\lvert S\rvert + \lvert A\rvert)$

Comparaison matrice d'adjacence / liste d'adjacence

Pour un graphe orienté à n sommets et m arêtes :

Opération	Matrice d'adjacence	Liste d'adjacence
ajouter arête	$\mathsf{O}(1)$	$\mathsf{O}(1)$
existence arête	$\mathsf{O}(1)$	$\mathsf{O}(\deg^+(u))$
voisins de u	$\mathsf{O}(n)$	$\mathsf{O}(\deg^+(u))$
espace	$\mathsf{O}(\lvert S\rvert^2)$	$\mathsf{O}(\lvert S\rvert + \lvert A\rvert)$

Si $m \approx n^2$ (graphe dense) : matrice d'adjacence conseillée.
Si $m = \mathsf{O}(n)$ (graphe creux, ex : arbre) : liste d'adjacence conseillée.

Dictionnaire d'adjacence

On peut aussi utiliser un dictionnaire qui à chaque sommet associe l'ensemble de ses voisins.

Dictionnaire d'adjacence

On peut aussi utiliser un dictionnaire qui à chaque sommet associe l'ensemble de ses voisins.

Implémentations possibles pour le dictionnaire et les ensembles :

- Table de hachage : ajout, suppression, appartenance en $O(1)$ en moyenne.
- Arbre binaire de recherche : ajout, suppression, appartenance en $O(h)$ ($O(\log(n))$) si équilibré).

Dictionnaire d'adjacence

On peut aussi utiliser un dictionnaire qui à chaque sommet associe l'ensemble de ses voisins.

Implémentations possibles pour le dictionnaire et les ensembles :

- Table de hachage : ajout, suppression, appartenance en $O(1)$ en moyenne.
- Arbre binaire de recherche : ajout, suppression, appartenance en $O(h)$ ($O(\log(n))$) si équilibré).

Intérêts :

- Les sommets ne sont pas forcément des entiers.
- Test d'adjacence efficace (comme pour une matrice d'adjacence).
- Complexité mémoire faible (comme pour une liste d'adjacence).

Parcours de graphe

Pour parcourir les sommets d'un graphe :

1. Parcours en profondeur (Depth-First Search) : on visite les sommets le plus profondément possible avant de revenir en arrière.
2. Parcours en largeur (Breadth-First Search) : on visite les sommets par distance croissante depuis une racine.

Si le graphe est connexe, tous les sommets sont visités.
Sinon, on peut appliquer un parcours sur chacune des composantes connexes.

Parcours de graphe

Pour simplifier, on va utiliser la fonction OCaml

```
List.iter : ('a -> unit) -> 'a list -> unit
```

qui applique une fonction à tous les éléments d'une liste.

Parcours de graphe

Un DFS sur $G = (S, A)$ depuis une racine r consiste, si r n'a pas déjà été visité, à le visiter puis s'appeler récursivement sur ses voisins :

Parcours de graphe

Un DFS sur $G = (S, A)$ depuis une racine r consiste, si r n'a pas déjà été visité, à le visiter puis s'appeler récursivement sur ses voisins :

```
let dfs g r = (* g : int list array est une liste d'adjacence *)
    let n = Array.length g in
    let vus = Array.make n false in
    let rec aux v =
        if not vus.(v) then (
            vus.(v) <- true;
            List.iter aux g.(v)
        ) in
    aux r
```

Exercice

Adapter `dfs` si `g` est représenté par matrice d'adjacence.

Parcours de graphe

```
let dfs g r = (* g : int list array  est une liste d'adjacence *)
    let n = Array.length g in
    let vus = Array.make n false in
    let rec aux v =
        if not vus.(v) then (
            vus.(v) <- true;
            List.iter aux g.(v)
        ) in
    aux r
```

Complexité :

Parcours de graphe

```
let dfs g r = (* g : int list array  est une liste d'adjacence *)
    let n = Array.length g in
    let vus = Array.make n false in
    let rec aux v =
        if not vus.(v) then (
            vus.(v) <- true;
            List.iter aux g.(v)
        ) in
    aux r
```

Complexité : $\boxed{O(|S| + |A|)}$ si représenté par liste d'adjacence car

1. `Array.make` est en $O(|S|)$
2. chaque arête donne lieu à au plus 2 appels récursifs de aux (1 si orienté), d'où $O(|A|)$ appels récursifs
3. chaque appel récursif est en $O(1)$ (`g.(v)` est en $O(1)$)

Parcours de graphe

Question

Comment déterminer si un graphe non orienté contient un cycle ?

Parcours de graphe

Question

Comment déterminer si un graphe non orienté contient un cycle ?

Ne pas considérer un cycle si on revient sur le prédécesseur :

```
let has_cycle (g : int list array) =
    let n = Array.length g in
    let pere = Array.make n (-1) in
    let ans = ref false in
    let rec aux p u = (* p a permis de découvrir u *)
        if pere.(u) = -1 then (
            pere.(u) <- p;
            List.iter (aux p) g.(u)
        )
        else if pere.(p) <> u then ans := true (* cycle trouvé *)
        in
    aux 0 0; (* cherche un cycle depuis le sommet 0 *)
    !ans
```

Parcours de graphe

Question

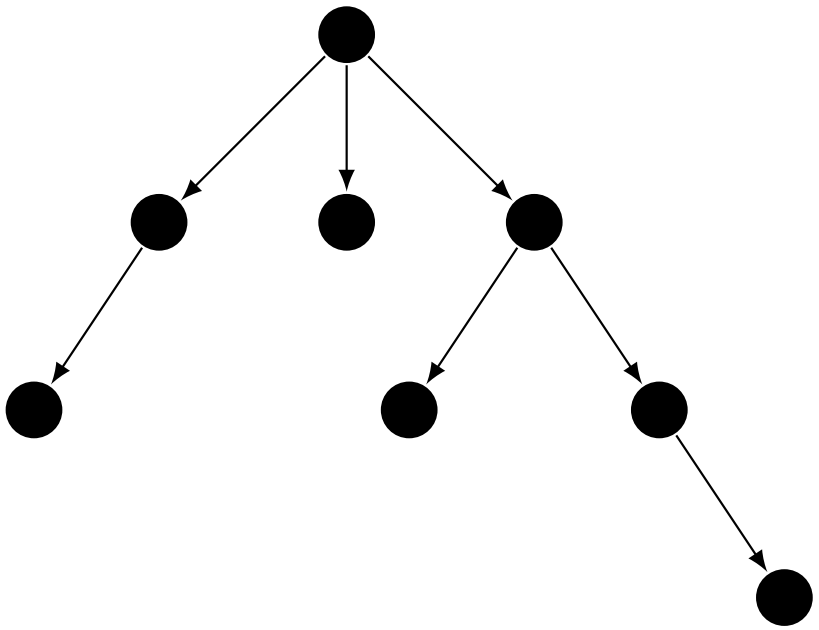
Comment déterminer si un graphe orienté $G = (S, A)$ contient un cycle ?

Parcours de graphe

Question

Comment déterminer si un graphe orienté $G = (S, A)$ contient un cycle ?

Soit A un arbre de parcours en profondeur de G.



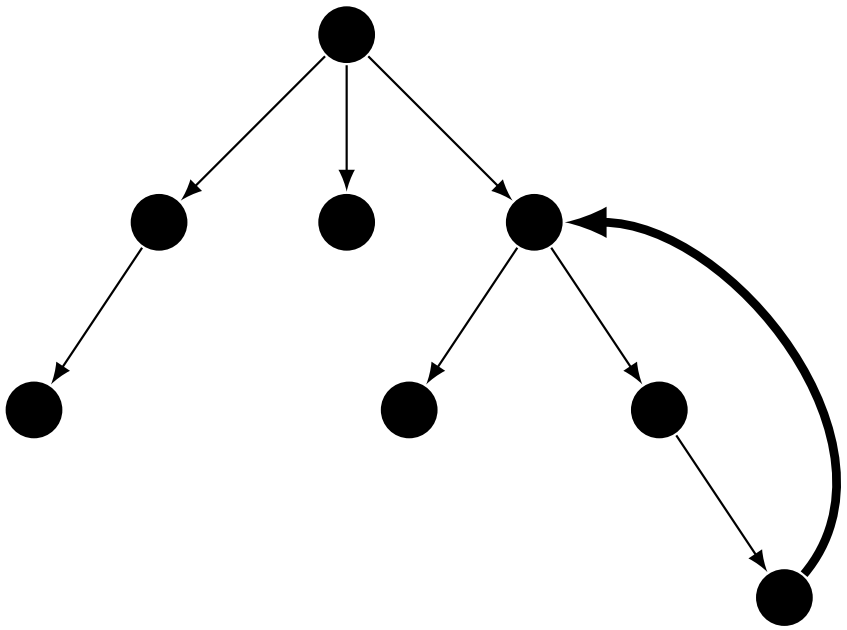
Un arc arrière de A est un arc $e \in A$ d'un sommet de A vers un de ses ancêtres.

Parcours de graphe

Question

Comment déterminer si un graphe orienté $G = (S, A)$ contient un cycle ?

Soit A un arbre de parcours en profondeur de G.



Un arc arrière de A est un arc $e \in A$ d'un sommet de A vers un de ses ancêtres.

Parcours de graphe

Soit A un arbre de parcours en profondeur de G depuis r :

Théorème

G a un cycle C atteignable depuis r

$\Longleftrightarrow$

A possède un arc arrière

Parcours de graphe

Soit A un arbre de parcours en profondeur de G depuis r :

Théorème

G a un cycle C atteignable depuis r

$\Longleftrightarrow$

A possède un arc arrière

Preuve :

$\Longleftarrow$: évident.

$\Longrightarrow$: Soit v_0 le premier sommet de C atteint par A. Notons $C = v_0 \to v_1 \to ... \to v_k \to v_0$.

Parcours de graphe

Soit A un arbre de parcours en profondeur de G depuis r :

Théorème

G a un cycle C atteignable depuis r

$\Longleftrightarrow$

A possède un arc arrière

Preuve :

$\Longleftarrow$: évident.

$\Longrightarrow$: Soit v_0 le premier sommet de C atteint par A. Notons $C = v_0 \to v_1 \to ... \to v_k \to v_0$.

Alors l'appel de `dfs` sur v_0 va visiter v_k :

Parcours de graphe

Soit A un arbre de parcours en profondeur de G depuis r :

Théorème

G a un cycle C atteignable depuis r

$\Longleftrightarrow$

A possède un arc arrière

Preuve :

$\Longleftarrow$: évident.

$\Longrightarrow$: Soit v_0 le premier sommet de C atteint par A. Notons $C = v_0 \to v_1 \to ... \to v_k \to v_0$.

Alors l'appel de `dfs` sur v_0 va visiter v_k : (v_k, v_0) est un arc arrière.

Parcours de graphe

On teste l'existence d'un arc arrière (qui revient sur un sommet en cours d'appel récursif) :

```
let has_cycle g =
    (* g : graphe orienté représenté par liste d'adjacence *)
    let n = Array.length g in
    let vus = Array.make n 0 in
    let ans = ref false in
    let rec aux v = match vus.(v) with
        | 0 -> vus.(v) <- 1;
               List.iter aux g.(v);
               vus.(v) <- 2
        | 1 -> ans := true
        | _ -> () in
    for i = 0 to n - 1 do
        aux i (* cherche un cycle depuis le sommet i *)
    done;
    !ans
```

Parcours de graphe

Exercice

Écrire une fonction `cc : int array array -> int` qui renvoie le nombre de composantes connexes d'un graphe non orienté défini par matrice d'adjacence.

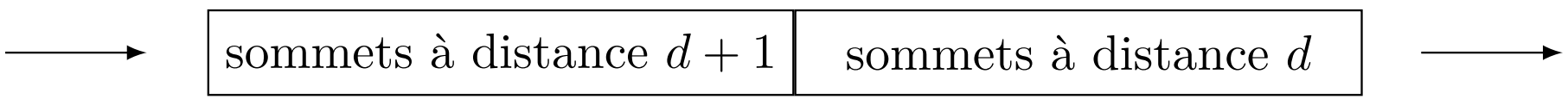
Parcours en largeur

Parcours en largeur avec une file q :

```
let bfs (g : int list array) (r : int) =
    let vus = Array.make (Array.length g) false in
    let q = Queue.create () in
    let add v =
        if not vus.(v) then (
            vus.(v) <- true; Queue.add v q
        ) in
    add r;
    while not (Queue.is_empty q) do
        let u = Queue.pop q in
        (* traiter u *)
        List.iter add g.(u)
    done
```

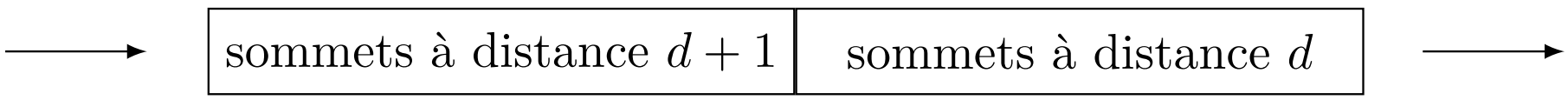
Parcours en largeur

La file q est toujours de la forme :



Parcours en largeur

La file `q` est toujours de la forme :



Les sommets sont donc traités par distance croissante à `s` : d'abord `s`, puis les voisins de `s`, puis ceux à distance 2...

Parcours en largeur

Soit $G = (S, A)$ un graphe non pondéré et $s \in S$.
Un parcours en largeur permet de calculer les distances de s à tous les sommets de G. Pour cela, on peut stocker des couples (sommet, distance) dans la file :

Parcours en largeur

Soit $G = (S, A)$ un graphe non pondéré et $s \in S$.
Un parcours en largeur permet de calculer les distances de s à tous les sommets de G. Pour cela, on peut stocker des couples (sommet, distance) dans la file :

```
let bfs g r =
    let dist = Array.make (Array.length g) (-1) in
    let q = Queue.create () in
    let add d v =
        if dist.(v) = -1 then (
            dist.(v) <- d;
            Queue.add (v, d) q
        ) in
    add 0 r;
    while not (Queue.is_empty q) do
        let u, d = Queue.pop q in
        List.iter (add (d + 1)) g.(u)
    done;
    dist  (* dist.(u) est la distance de r à u *)
```

Parcours en largeur

Question

Comment connaître un plus court chemin d'un sommet `r` à un autre ?

Parcours en largeur

Question

Comment connaître un plus court chemin d'un sommet `r` à un autre ?

On stocke dans `pred.(v)` le sommet qui a permis de découvrir `v` :

```
let bfs g r =
    let pred = Array.make (Array.length g) (-1) in
    let q = Queue.create () in
    let add p v = (* p est le père de v *)
        if pred.(v) = -1 then (pred.(v) <- p; Queue.add v q) in
    add r r;
    while not (Queue.is_empty q) do
        let u = Queue.pop q in
        List.iter (add u) g.(u)
    done;
    pred  (* pred.(v) = prédécesseur de v *)
```

Parcours en largeur

Question

Comment en déduire un plus court chemin de r à v ?

Parcours en largeur

Question

Comment en déduire un plus court chemin de r à v ?

```
let rec chemin pred v =
    if pred.(v) = v then [v]
    else v::chemin pred pred.(v)
```