Graphes : Représentations

Quentin Fortier

September 15, 2022

On souhaite implémenter une structure de graphe possédant les opérations :

1. ajouter / supprimer une arête
2. savoir s'il existe une arête entre 2 sommets
3. connaître la liste des voisins d'un sommet
4. ...

Avec, si possible, une faible complexité en temps et espace.

Représentation de graphe

Les sommets seront des entiers consécutifs à partir de 0.

Il existe principalement deux représentations possibles de graphes :

1. Par **matrice d'adjacence**
2. Par **liste d'adjacence**

On verra aussi la représentation par dictionnaire d'adjacence, qui est la plus générale.

Matrice d'adjacence : Définition

On peut représenter un graphe non orienté (V, E), où $V = \{0, ..., n-1\}$ par une **matrice d'adjacence** A de taille $n \times n$ définie par :

- $A_{i,j} = 1 \iff \{i, j\} \in E$
- $A_{i,j} = 0 \iff \{i, j\} \notin E$

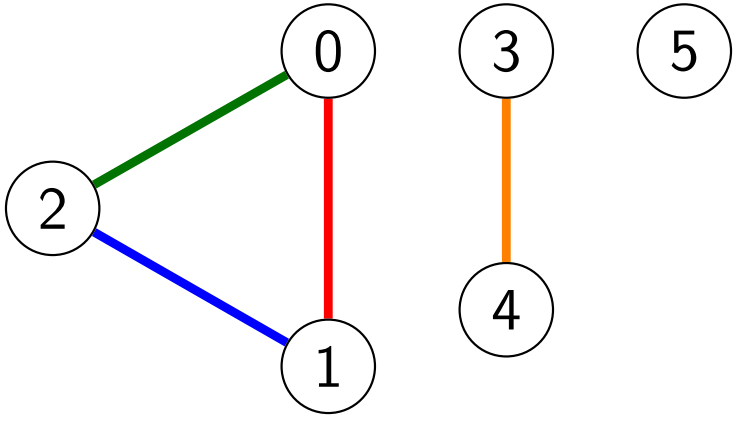
Remarque : A est symétrique.

Pour un graphe orienté $(V, \vec{E})$:

- $A_{i,j} = 1 \iff (i, j) \in \vec{E}$
- $A_{i,j} = 0 \iff (i, j) \notin \vec{E}$

A n'est pas symétrique (a priori).

Matrice d'adjacence : Exemple



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

Quitte à permuter lignes et colonnes (i.e renuméroter les sommets), les composantes connexes apparaissent par bloc.

Matrice d'adjacence : Exemple

En OCaml, **une matrice est un tableau de tableaux**.
Par exemple, voici une matrice avec sa représentation en OCaml :

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

```
let m = [|[|1; 0; 3|]; [|2; 2; 1|]|]
```

`m.(i)`	`m.(i).(j)`	`Array.length m`	`Array.length m.(0)`
$i$ème ligne	élément ligne i, colonne j	nombre de lignes	nombre de colonnes

Et `Array.make_matrix n p 0` renvoie une matrice $n \times p$ remplie de 0.

Matrice d'adjacence : Degré

Exercice

Écrire une fonction `deg g v` pour calculer le degré d'un sommet `v` dans un graphe représenté par matrice d'adjacence `g`.

On regarde le nombre de 1 sur la ligne `v` de la matrice :

```
let deg g v =
    let r = res 0 in
    for j = 0 to Array.length g.(0) - 1 do
        if g.(v).(j) = 1 then incr r
    done;
    !r
```

Matrice d'adjacence : Puissance

Question

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 $k = 2$:

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

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

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

Matrice d'adjacence : Puissance

Question

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.

Matrice d'adjacence : Puissance

Soit $M(n)$ la complexité pour multiplier 2 matrices $n \times n$ ($M(n) = \mathrm{O}(n^3)$ en naïf, $\mathrm{O}(n^{2,8})$ avec la méthode de Strassen).

On peut calculer $A^k = A \times ... \times A$ en $\mathrm{O}(kM(n))$.

Ou, mieux, par exponentation rapide en utilisant :

$$\begin{cases} A^k = (A^{\frac{k}{2}})^2 & \text{si } k \text{ est pair} \\ A^k = A(A^{\frac{k-1}{2}})^2 & \text{sinon} \end{cases}$$

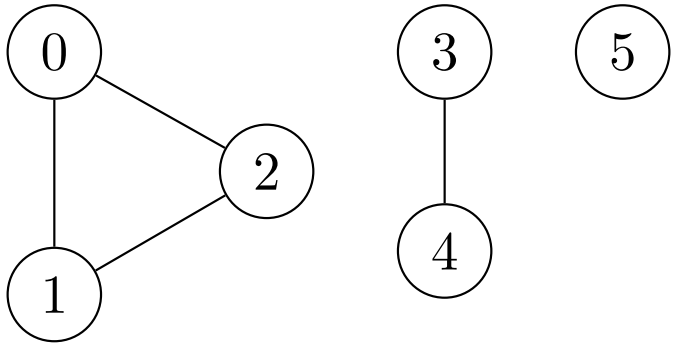
Complexité : $\mathrm{O}(\log(k)M(n))$.

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 : Exemple



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];
 []|]
```

Liste d'adjacence : Voisins

Renvoyer la liste des voisins d'un sommets u dans un graphe G :

- **Avec matrice d'adjacence** :

```
let voisins g v =
    let rec aux j =
        if j = Array.length g then []
        else let q = aux (j + 1) in
            if g.(v).(j) = 1 then j::q
            else q in
    aux 0
```

- **Avec liste d'adjacence** :

```
let voisins g v = g.(v)
```

Liste d'adjacence : Voisins

Exercice

Écrire une fonction pour calculer le nombre d'arêtes d'un graphe orienté représenté par liste d'adjacence.

Exercice

Écrire deux fonctions pour convertir une matrice d'adjacence en liste d'adjacence et vice-versa.

Exercice

Ecrire une fonction pour renvoyer la transposée d'un graphe orienté représenté sous forme de liste d'adjacence, c'est-à-dire en inversant le sens de toutes les flèches.

Exercice 2

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

```
let lst_to_mtx g =
    let n = Array.length g in
    let mtx = Array.make_matrix n n 0 in
    let rec add i lst =
        match lst with
            |[] -> ()
            |t::q -> mtx.(i).(t) <- 1; add i q in
        for i = 0 to n-1 do
            add i g.(i)
            done;
    mtx
```

Comparaison matrice d'adjacence / liste d'adjacence

Pour un graphe orienté à n sommets et m arêtes :

Opération	Matrice d'adjacence	Liste d'adjacence
espace	$O(n^2)$	$O(n+m)$
ajouter (u, v)	$O(1)$	$O(1)$
supprimer (u, v)	$O(1)$	$O(\deg^+(u))$
existence (u, v)	$O(1)$	$O(\deg^+(u))$
voisins de u	$O(n)$	$O(\deg^+(u))$
ajouter sommet	X	X
supprimer sommet	X	X

Si $m = \Theta(n^2)$ (graphe dense) : matrice d'adjacence conseillée.
Si $m = O(n)$ (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.

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).

Dictionnaire d'adjacence : Rappels

Définition

Un **dictionnaire** est une structure de données abstraite stockant des associations (clé, valeur), avec (au moins) les opérations suivantes :

- Ajout d'une nouvelle association (clé, valeur).
- Obtention d'une valeur associée à une clé.

Dictionnaire d'adjacence : Par table de hachage

Définition

Une **table de hachage** permet d'implémenter un dictionnaire avec :

1. Un **tableau** contenant les valeurs.
2. Une **fonction de hachage** h telle que, si k est une clé, $h(k)$ est l'indice du tableau où se trouve la valeur associée à k.

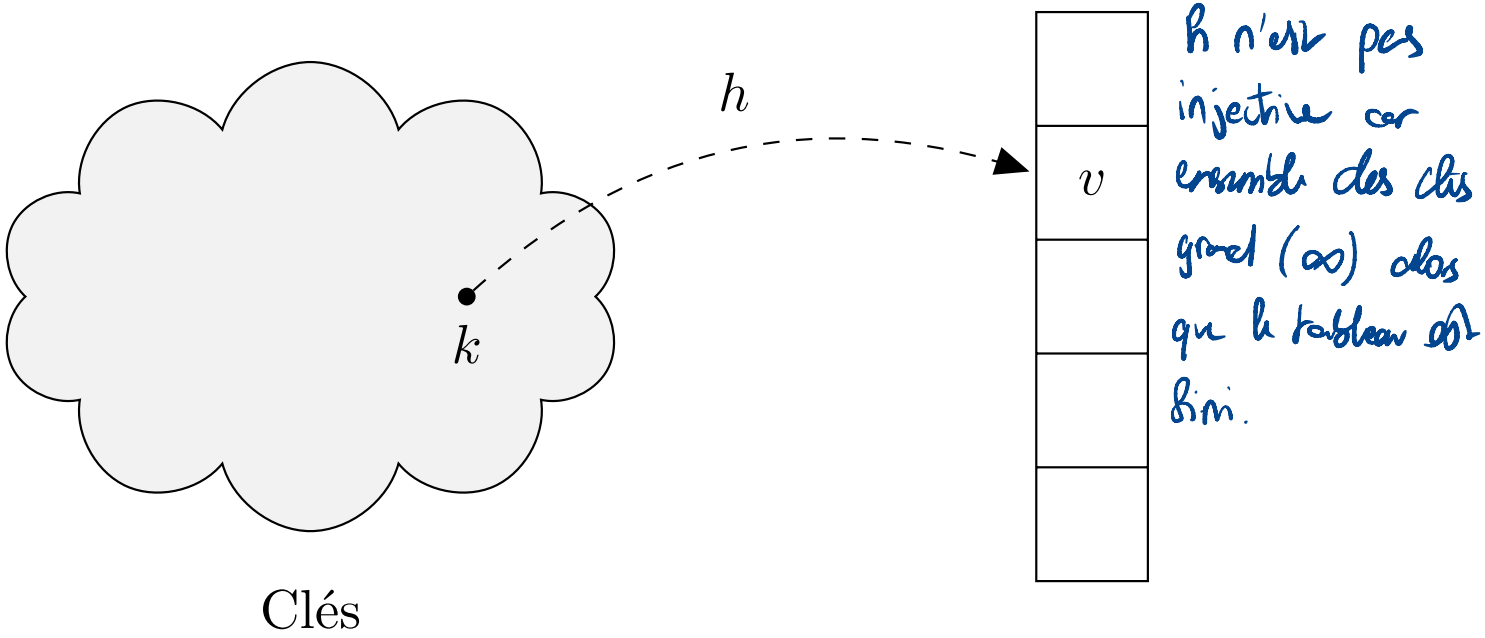


Table de hachage $\approx$ tableau dont les indices (clés) ne sont pas forcément entiers. Les clés doivent être persistantes

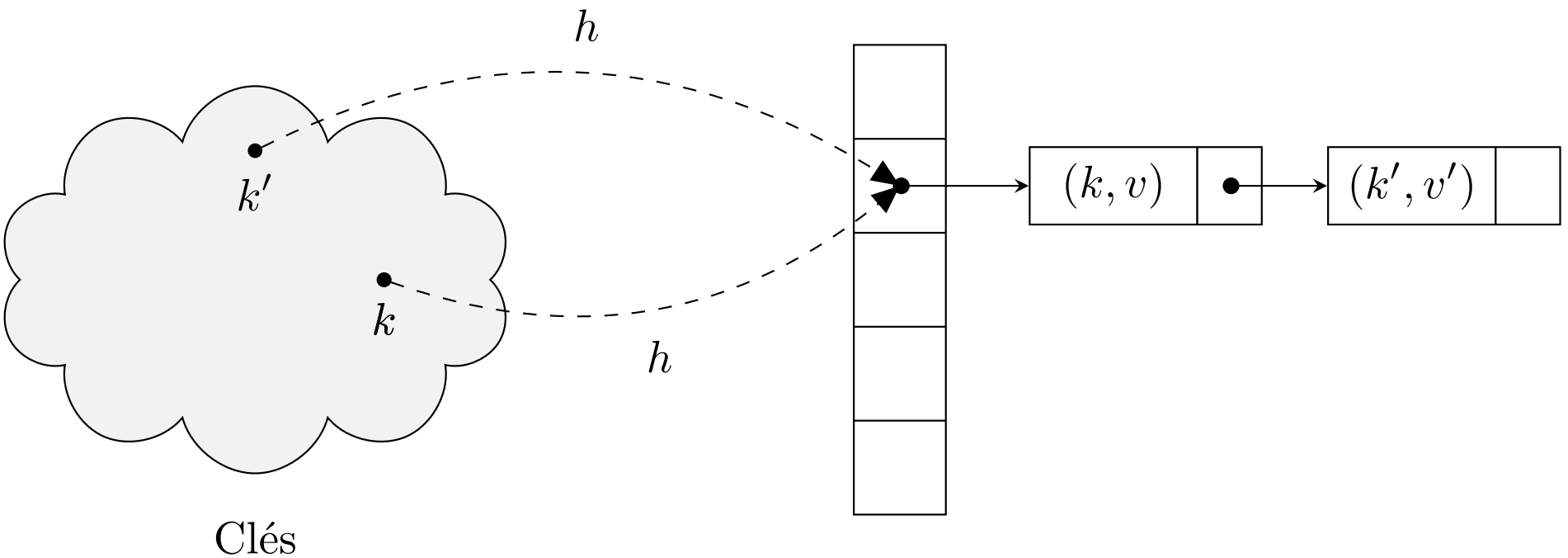
Dictionnaire d'adjacence : Par table de hachage

Remarques :

- Si les clés sont des entiers (non consécutifs), on choisit souvent $h(k) = k \mod n$, où n est la taille du tableau.
- Les clés doivent être hachable (être dans l'ensemble de définition de la fonction de hachage).
- Si h se calcule en $O(1)$ et sous hypothèse de bonne répartition des images de h, les opérations de dictionnaire sont en $O(1)$ en moyenne.
- Quand les clés sont plus compliquées (listes, arbres...), il est difficile de trouver une fonction de hachage à la fois rapide à calculer et n'engendrant pas trop de collisions.

Dictionnaire d'adjacence : Par table de hachage

Souvent, il y a beaucoup plus de clés possibles que de cases du tableau, ce qui conduit à des **collisions** : plusieurs clés ayant la même image par h. On peut résoudre ces collisions par **chaînage**, en stockant une liste à chaque position de la table de hachage :



Autre possibilité de résolution de collisions : **adressage ouvert**.

Dictionnaire d'adjacence : Par table de hachage

En OCaml, le module **Hashtbl** implémente une table de hachage :

```
let t = Hashtbl.create 42;; (* taille initiale du tableau *)
(* ajoute la clé "red" avec la valeur (255, 0, 0) *)
Hashtbl.add t "red" (255, 0, 0);;
Hashtbl.find t "red";; (* renvoie la valeur associée à "red" *)
Hashtbl.remove t "red";; (* supprime une association *)
```

Le tableau est redimensionné lorsqu'il y a trop de valeurs (pour éviter trop de collisions).

Dictionnaire d'adjacence : Par arbre binaire de recherche

On peut aussi implémenter un dictionnaire par arbre binaire de recherche où les noeuds sont des couples (clé, valeur) et où la condition d'arbre binaire de recherche s'effectue sur les clés :

```
type ('k, 'v) tree =
    | E
    | N of ('k * 'v) * ('k, 'v) tree * ('k, 'v) tree
```

Dictionnaire d'adjacence : Par arbre binaire de recherche

```
type ('k, 'v) tree =
    | E
    | N of ('k * 'v) * ('k, 'v) tree * ('k, 'v) tree

(* ajoute une association clé, valeur dans l'arbre a *)
let rec add cle valeur a = match a with
    | E -> N((cle, valeur), E, E)
    | N(r, g, d) ->
        if cle < fst r then N(r, add cle valeur g, d)
        else N(r, g, add cle valeur d)

(* donne la valeur associée à cle *)
let rec get cle a = match a with
    | E -> failwith "Clé introuvable"
    | N((k, v), g, d) ->
        if k = cle then v
        else if k < cle then get cle g
        else get cle d
```

Dictionnaire d'adjacence : Ensemble

Définition

Un ensemble est une structure de données possédant les opérations :

- Ajouter un élément.
- Tester si un élément appartient à l'ensemble.
- (Éventuellement) Supprimer un élément.

Implémentations :

- Table de hachage où les valeurs sont bidons (`()` par exemple). Les clés ayant une valeur dans le tableau sont les éléments de l'ensemble. Il faut que les éléments soient hachables.
- Arbre binaire de recherche dont les noeuds sont les éléments de l'ensemble. Il faut une relation d'ordre sur les éléments.
  Le module `Set` d'OCaml implémente un ensemble par un ABR.

Module (HP)

Un **module** OCaml permet de regrouper ensemble plusieurs fonctions/variables, pour avoir du code mieux structuré.

Type de module possible pour un graphe :

```
module type Graph = sig
    type vertex
    type t
    val empty : int -> t
    val n : t -> int
    val is_edge : vertex -> vertex -> t -> bool
    val add_edge : vertex -> vertex -> t -> unit
    val del_edge : vertex -> vertex -> t -> unit
end
```

Intérêts : code plus propre, ne pas avoir besoin de réécrire le même algorithme pour les différentes implémentations de graphe...

Module (HP)

Implémentation par matrice d'adjacence :

```
module MatrixGraph : (Graph with type vertex := int) = struct
    type vertex = int
    type t = vertex array array
    let empty n = Array.make_matrix n n 0
    let n = Array.length
    let is_edge i j m = m.(i).(j) <> 0
    let add_edge i j m = m.(i).(j) <- 1
    let del_edge i j m = m.(i).(j) <- 0
end
```

```
let g = MatrixGraph.empty 4;; (* graphe à 4 sommets *)
MatrixGraph.add_edge 0 1;; (* ajout d'une arête entre les sommets
MatrixGraph.is_edge 0 3;; (* false *)
MatrixGraph.add_edge 0 3;;
MatrixGraph.is_edge 0 3;; (* true *)
```