Numérique et sciences informatiques
Classe de terminale

II Structures de données
D Les graphes
1. Notions générales et implémentation

I Définition, vocabulaire et représentation graphique

I.1 Les graphes non orientés

a) Définition et représentation graphique

• Un **graphe non orienté** est défini par :
  * un ensemble S de **sommets** (ou de **nœuds**) ;
  * un ensemble A **d'arêtes.**

• Une **arête** est un *ensemble* d'éléments de S comportant exactement deux éléments. Un ensemble ne comporte pas d'ordre : si a et b appartiennent à S, alors $\{a\ ;\ b\}$ et $\{a\ ;\ b\}$ est la même arête.
Remarque : Dans certains graphes, on peut accepter que certaines arrêtes ne comportent qu'un seul élément. Dans ce cas, on dit qu'il s'agit d'une **boucle**.

Représentation graphique d'un graphe non orienté

Il ne faut pas confondre un graphe avec sa représentation graphique : On peut former *différentes représentations* graphiques pour un *même graphe*.

b) Vocabulaire des graphes non orientés

• L'**ordre** d'un graphe est le cardinal de S, c'est-à-dire le nombre de ses sommets.

Remarque : un graphe peut être infini.

• On appelle **voisins** d'un sommet les sommets auxquels il est rattaché par une arête.

• Deux sommets sont **adjacents** s'ils sont voisins l'un de l'autre.

• Le **degré** d'un sommet est le nombre de ses voisins.

Exemples :

c) Chaînes et cycles dans un graphe non orienté

• Une **chaîne** est une succession de sommets tels que deux sommets consécutifs de la chaîne sont toujours reliés par une arête.

* une **chaîne simple** est une chaîne qui ne suit pas plus d'une seule fois une arrête donnée.

* une **chaîne élémentaire** est une chaîne élémentaire ne passe pas plus d'une fois par un sommet. Toutefois le premier sommet et le dernier sommet final peuvent être identiques**.**

* La **longueur d'une chaîne** est le nombre des sommets qui la forment.

• Un **cycle** est une *chaîne élémentaire* telle que le premier sommet et le dernier sommet final sont identiques**.**

• La **distance** entre deux sommets est la longueur de la plus petite chaîne qui les relie.

Exemples :

d) Graphes étiquetés et graphes pondérés

• On peut associer à chaque arête une valeur d'un type donné (une chaîne de caractères, un nombre, un couple, etc.). Dans ce cas on dit qu'il s'agit d'un **graphe étiqueté**. La valeur associée à un arc est son **étiquette**.

• Si les valeurs associées à chaque arête sont des nombres on dit qu'il s'agit d'un **graphe pondéré** (ou **valué**). Le nombre associé à une arrête est alors son **poids** ou son **coût**. Dans le cas d'un graphe pondéré, les poids sont souvent des entiers positifs.

Exemples :

I.2 Graphes orientés

• Un **graphe orienté** est défini par :

* un ensemble S de **sommets** ;

* un ensemble A **d'arcs**.

• Un **arc** est un *couple* d'éléments de S. Contrairement aux arêtes dans un graphe non orienté, *les arcs comportent un ordre* : si a et b sont deux sommets, l'arc $(a\ ;\ b)$ ne coïncide pas avec l'arc $(b\ ;\ a)$. Le sommet a est **l'origine** et le sommet b est **l'extrémité** de l'arc $(a\ ;\ b)$
Remarque : Dans certains graphes, on accepte les arcs de la forme $(a\ ;\ a)$ où a est un élément de S. Dans ce cas on dit aussi que l'arc $(a\ ;\ a)$ est une **boucle**.

Représentation graphique d'un graphe orienté

b) Ordre d'un graphe et degré d'un sommet :

• L'**ordre** d'un graphe orienté est le cardinal de S, c'est-à-dire le nombre d'éléments de ses sommets.

• Si s est le sommet d'un graphe orienté,
  * on appelle **prédécesseurs de *s*** les sommets s' tels qu'il existe un arc $(s' ; s)$;
  * on appelle **successeurs de *s*** les sommets s" tels qu'il existe un arc $(s ; s")$. *Exemples :*

c) Chemins et circuits dans un graphe orienté

• Dans un graphe orienté, un **chemin** est une succession de sommets telle que pour -deux sommets s et s' consécutifs (dans cet ordre) il existe un arc $(s ; s')$. Comme pour les graphes non orientés, on utilise les notions de **chemin simple**, de **chemin élémentaire** et de **longueur** d'un chemin

• Dans un graphe orienté, un **circuit** est un *chemin élémentaire* tel que le premier et le dernier sommets coïncident.

• Comme les graphes non orientés, les graphes orientés peuvent être étiquetés ou pondérés.

I.3 Propriétés d'un graphe

• Un graphe est **simple** s'il ne comporte pas de boucle.

• Un graphe non orienté (resp. orienté) est **complet** si, pour deux sommets quelconques s et s', il existe une arête $\{s ; s'\}$ (resp. un arc $(s ; s')$).

• Un graphe non orienté est **connexe** si pour deux sommets quelconques s et s', il existe une chaîne dont s est le premier sommet et s' est le dernier sommet.

• Un graphe orienté est **fortement connexe** si, pour deux sommets quelconques s et s', il existe un chemin dont s est le premier sommet et s' est le dernier sommet.

II Structure de données permettant de représenter un graphe

II.1 Matrice d'adjacence

a) Graphes orienté et non pondéré

• Considérons un graphe G **orienté et non pondéré** d'ordre n, où n est un entier naturel supérieur ou égal à 2 et associons à chaque sommet de G un entier compris entre 1 et n. On notera donc $s_1, s_2, \ldots, s_n$ les sommets de ce graphe.

• Pour représenter ce graphe, on définir une matrice carrée d'ordre n $A = [a_{ij}]$ de la façon suivante : pour tout entier i compris entre 1 entre n et tout entier j compris entre 1 entre n :

* s'il existe un arc $(s_i\ ;\ s_j)$ (donc de s_i vers s_j) alors $a_{ij} = 1$
* sinon $a_{ij} = 0$.

On dira que A est la **matrice d'adjacence** du graphe G

Exemple : Ainsi le graphe

sera représenté par la matrice d'adjacence suivante :

b) Graphes non orienté et non pondéré

• Si le graphe G est **non orienté et non pondéré**, il suffit de définir la matrice d'adjacence A de la façon suivante :

* s'il existe une arête $\{s_i\ ;\ s_j\}$ alors $a_{ij} = 1$ *et* $a_{ji} = 1$;
* sinon, $a_{ij} = 0$ *et* $a_{ij} = 0$.

Remarque : dans ce cas, la matrice d'adjacence sera toujours symétrique par rapport à la diagonale principale, c'est-à-dire que pour tout i et tout j entre 1 et n, on a $a_{ij} = a_{ji}$

Exemple : Ainsi le graphe

sera représenté par la matrice d'adjacence suivante :

c) Graphes pondérés

• Supposons que le graphe G est **orienté et pondéré** et tel que les poids associés à chaque arc *sont tous non nuls*. On peut alors définir une **matrice de valuation** $V = [v_{ij}]$ de la façon suivante :

* s'il existe un arc $(s_i\ ;\ s_j)$ de poids p alors $v_{ij} = p$
* s'il n'existe pas d'arc $(s_i\ ;\ s_j)$ alors $v_{ij} = 0$.

• On peut bien entendu définir une matrice de valuation $V = [v_{ij}]$ dans le cas d'un graphe non orienté et non pondéré de la façon suivante :

* s'il existe une arête $\{s_i\ ;\ s_j\}$ de poids p alors $v_{ij} = p$ et $v_{ji} = p$
* s'il n'existe pas d'arête $\{s_i\ ;\ s_j\}$, $v_{ij} = 0$ et $v_{ij} = 0$

Exemple : Ainsi le graphe

sera représenté par la matrice d'adjacence suivante :

II.2 Listes d'adjacence

• Une autre façon de représenter un graphe est de créer un dictionnaire ou un tableau permettant d'associer à chaque sommet :

* la **liste de ses voisins** s'il s'agit d'un graphe non orienté ;
* la **liste de ses successeurs** s'il s'agit d'un graphe orienté.

Remarque : Bien entendu, dans le cas d'un graphe orienté, on pourra aussi représenter un graphe en associant à chaque sommet la liste de ses *prédécesseurs*.

• En Python, le graphe pourra par exemple être représenté par un dictionnaire dont les clés sont les (noms des) sommets du graphe et les valeurs associés sont des tableaux (ou des ensembles) contenant la liste de ses voisins, de ses successeurs ou de ses prédécesseurs.

• *Exemple :* Ainsi le graphe orienté

sera représenté par le dictionnaire suivant :

III Une implémentation d'une classe `Graphe` en Python

III.1 La classe abstraite `Graphe`

a) Les attributs pour une classe graphe

• Les attributs de la classe `Graphe` :

* `adj` : une matrice d'adjacence (de type **`list`**) ou un dictionnaire d'adjacence (de type **`dict`**).
* `ordre` : le nombre de sommets dans le graphe (de type **`int`**).

• Il faut faire la distinction entre « graphe statique » et « graphe dynamique ».
* l'usage d'une matrice d'adjacence nous contraint quasiment à représenter des graphes statiques ;
* l'usage d'un dictionnaire d'adjacence autorise à représenter des graphes dynamiques.

b) Les méthodes pour une classe graphe « statique »

Nom de la méthode graphe non orienté ou orienté	**Action effectuée**
`__init__(self, n)`	crée un graphe **d'ordre n** sans arc ou sans arête
`afficher(self)`	affiche à l'écran la liste des voisins ou des successeurs de chaque sommet et ne renvoie rien
`ajouter_arete(self, s1, s2)` ou `ajouter_arc(self, s1, s2)`	modifie l'objet et ne renvoie rien
`supprimer_arete(self, s1, s2)` ou `supprimer_arc(self, s1, s2)`	modifie l'objet et ne renvoie rien.
`voisins(self, s)` ou `successeurs(self, s)` et `predecesseurs(self, s)`	renvoie un tableau des voisins, des successeurs ou des prédécesseurs
`arete_existe(self, s1, s2)` ou `arc_existe(self, s1, s2)`	renvoie un booléen

c) Les méthodes pour une classe graphe orienté « dynamique »

Nom de la méthode graphe orienté	**Action effectuée**
`__init__(self)`	crée un graphe vide sans sommet
`afficher(self)`	affiche à l'écran la liste des voisins ou successeurs de chaque sommet et ne renvoie rien
`ajouter_sommet(self, s)`	modifie l'objet et ne renvoie rien.
`sommet_existe(self, s)`	renvoie un booléen.
`ajouter_arc(self, s1, s2)`	modifie l'objet et ne renvoie rien

	crée les sommets s1 et/ou s2 s'ils ne sont pas des sommets du graphe
`supprimer_arc(self, s1, s2)`	modifie l'objet et ne renvoie rien.
`successeurs(self, s)` et `predecesseurs(self, s)`	renvoie la liste des voisins, des successeurs ou des prédécesseurs
`arc_existe(self, s1, s2)`	renvoie un booléen
`sommets(self)`	renvoie le tableau des sommets du graphe

III.2 Implémentation avec une matrice d'adjacence

L'implémentation d'une classe `Graphe` avec une matrice d'adjacence comporte l'inconvénient une certaine rigidité : la création d'un nouveau sommet y serait particulièrement lourde puisqu'il faudrait « plonger » la matrice d'adjacence dans une matrice plus grande. Cette implémentation convient donc plus particulièrement pour un graphe « statique » dans lequel *le nombre de sommets est fixé à la création du graphe et n'évolue plus ensuite*.

a) Graphe orienté

```
class Graphe_O:

    def __init__(self, n) :
        self.ordre = n
        self.adjacence = [[0]*n for k in range(n)]

    def afficher(self) :
        for s in range(self.ordre):
            print(s," -> ", end="")
            for v in range(self.ordre) :
                if self.adjacence [s][v] :
                    print(v, end=" ")
            print()

    def ajouter_arc(self, s1, s2):
        self.adjacence[s1][s2] = 1

    def supprimer_arc(self, s1, s2) :
        self.adjacence[s1][s2] = 0

    def successeurs(self, s) :
        return [k for k in range(self.ordre) \
                 if self.adjacence[s][k]==1]

    def predecesseurs(self, s) :
        return [k for k in range(self.ordre) \
                if self.adjacence[k][s] == 1]

    def arc_existe(self, s1, s2) :
        return self.adjacence[s1][s2] == 1
```

b) Graphe non orienté

```
class Graphe_nonO:

     def __init__(self, n) :
         ... # identique au cas orienté

    def afficher(self) :
         ... # identique au cas orienté

    def ajouter_arete(self, s1, s2):
         self.adjacence[s1][s2] = 1 # la matrice est
         self.adjacence[s2][s1] = 1 # symétrique

    def supprimer_arete(self, s1, s2) :
         self.adjacence[s1][s2] = 0
         self.adjacence[s2][s1] = 0

    def voisins(self, s) :
         return [k for k in range(self.ordre) \
         if self.adjacence[s][k] == 1]

    def arete_existe(self,s1, s2) :
         return self.adjacence [s1][s2] == 1
```