Numérique et Sciences Informatiques
Chapitre IX - Structures de données relationnelles

Classe de terminale

Un graphe est une structure de données relationnelle. Elle est utilisée pour représentée des données structurées de façon relationnelle, c'est-à-dire ayant des liens entre elles.

êxemple

- Des bases de données
- le réseau routier
- le réseau électrique
- Internet
- les réseaux sociaux (vu en SNT)

On distingue deux type de graphes : les graphes non orienté et les graphes orientés.

I. Graphe non orienté

I.1. Généralités

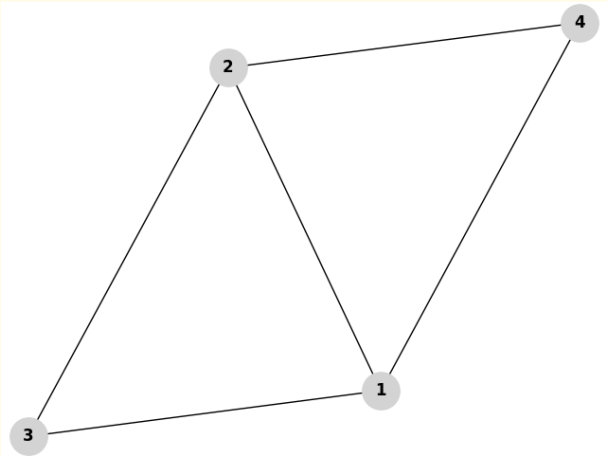
Définition

Un **graphe** est composé :

- de **sommets**
- d'**arêtes** reliant des couples de sommets.

Exemple de graphe

Voici la représetnation d'un graphe.



Il possède 4 sommets d'étiquettes 1, 2, 3 et 4. Il possède aussi 5 arêtes.
Deux sommets reliés par une arête sont **adjacents**.

Remarque

Un sommet peut n'être relié à aucune arête.

Vocabulaire

Une **arête incidente** d'un sommet est une arête qui touche le sommet.
Le **degré** d'un sommet est le nombre d'arêtes incidentes de ce sommet.
Les **voisins** d'un sommet S sont les sommets reliés à S par une arête.

Exemple

Avec le graphe non orienté précédent, on peut remplir le tableau ci-dessous :

Sommet	degré	voisins
1	3	3 - 2 - 4
2	3	1 - 3 - 4
3	2	1 - 2
4	2	1 - 2

I.2. Autres représentations de graphes non orientés

Deux autres façon de représenter les graphes non orienté est la liste des voisins ou une matrice d'adjacence.

Liste des voisins

Pour chaque sommet, on donne la liste des voisins.

Exemple

Dans notre exemple d'arbre non orienté, on a la représentation suivante :

- sommet 1 : 2 - 3 - 4
- sommet 2 : 1 - 3 - 4
- sommet 3 : 1 - 2
- sommet 4 : 1 - 2

Matrice d'adjacence

Une matrice d'adjacence est un tableau à double entrée de taille $n \times n$ avec n le nombre de sommet du graphe. Dans la colonne du sommet x et la ligne du sommet y, on indique le nombre d'arête menant de x à y.

Exemple

Dans notre exemple d'arbre non orienté, on peut établir le tableau suivant :

	sommet 1	sommet 2	sommet 3	sommet 4
sommet 1	0	1	1	1
sommet 2	1	0	1	1
sommet 3	1	1	0	0
sommet 4	1	1	0	0

La matrice d'adjacence est donc :

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

I.3. Parcours de graphes non orientés

Lorsqu'on passe d'un sommet à un autre en suivant des arêtes, on construit une **chaîne**.
Une chaîne peut emprunter plusieurs fois une même arête et un même sommet.
La **longueur** de la chaîne est le nombre de sommets empruntés.
Une chaîne **simple** est une chaîne qui n'emprunte pas deux fois la même arête.
Une chaîne **élémentaire** est une chaîne qui n'emprunte pas deux fois le même sommet.
Un **cycle** est une chaîne dont le dernier et le premier sommet sont identiques.

Exemple

Dans notre exemple d'arbre non orienté :

- 1 - 3 - 2 est une chaîne simple, élémentaire et de longueur 3
- 1 - 2 - 4 - 1 -3 est une chaîne simple de longueur 5
- 1 - 3 - 2 - 4 - 1 est un cycle. On pourra écrire le « cycle 1 - 3 - 2 - 4 », sous-entendu que le dernier sommet est relié au premier.

II. Graphe orienté

II.1. Généralités

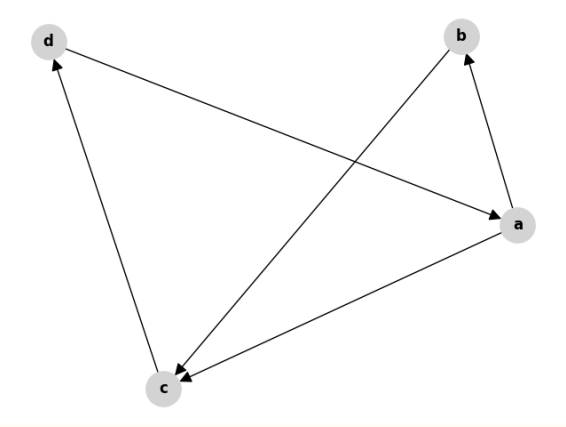
Définition

Un **graphe orienté** est composé :

- de **sommets**
- des **arcs** reliant des couples de sommets dans un sens donné, représentés par des flèches.

Exemple de graphe orienté

Voici la représentation d'un graphe.



Il possède 4 sommets d'étiquettes a, b, c, d. Il possède aussi 5 arcs :

- de a vers b
- de a vers c
- de d vers a
- de c vers d
- de b vers c

Vocabulaire

L'origine de l'arc est le sommet duquel part l'arc. On l'appelle aussi **source** de l'arc.
Le **but** de l'arc est le sommet vers lequel pointe l'arc.
Un **successeur** d'un sommet x est un sommet qui but d'un arc d'origine x.
Un **prédécesseur** d'un sommet y est un sommet origine d'un arc de but y.
Le **degré entrant** d'un sommet est égal au nombre d'arcs dont le but est le sommet. Il est aussi égal au nombre de prédécesseurs de ce sommet.
Le **degré sortant** d'un sommet est égal au nombre d'arcs dont l'origine est le sommet. Il est aussi égal au nombre de successeurs de ce sommet.
Le **degré** d'un sommet est égal à la somme dus degré entrant et du degré sortant.

Exemple

Avec le graphe orienté précédent, on peut remplir le tableau ci-dessous :

Sommet	prédécesseurs	successeurs	degré entrant	degré sortant	degré
a	d	b , c	1	2	3
b	a	c	1	1	2
c	a , b	d	2	1	3
d	c	a	1	1	2

II.2. Autres représentations de graphes orientés

Trois autres façons de représenter les graphes orienté sont la liste des successeurs, la liste des prédécesseurs ou une matrice d'adjacence.

Liste des successeurs

Pour chaque sommet, on donne la liste des successeurs.

Exemple

Dans notre exemple d'arbre orienté, on a la représentation suivante :

Liste des successeurs :

- sommet a : b, c
- sommet b : c
- sommet c : d
- sommet d : a

Liste des prédécesseurs

Pour chaque sommet, on donne la liste des prédécesseurs.

Exemple

Dans notre exemple d'arbre orienté, on a la représentation suivante :

Liste des prédécesseurs :

- sommet a : d
- sommet b : a
- sommet c : a, b
- sommet d : c

Matrice d'adjacence

Une matrice d'adjacence est un tableau à double entrée de taille $n \times n$ avec n le nombre de sommet du graphe. Dans la colonne du sommet y et la ligne du sommet x, on indique le nombre d'arcs menant de x à y.

Exemple

Dans notre exemple d'arbre non orienté, on peut établir le tableau suivant :

	sommet a	sommet b	sommet c	sommet d
sommet a	0	0	0	1
sommet b	1	0	0	0
sommet c	1	1	0	0
sommet d	0	0	1	0

La matrice d'adjacence est donc :

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

II.3. Parcours de graphes orientés

Lorsqu'on passe d'un sommet à un autre en suivant des arcs, on construit un **chemin**.
Une chemin peut emprunter plusieurs fois un même arc et un même sommet.
La **longueur** de la chaîne est le nombre de sommets empruntés.
Un chemin **simple** est un chemin qui n'emprunte pas deux fois le même arc.
Un chemin **élémentaire** est un chemin qui n'emprunte pas deux fois le même sommet.
Un **circuit** est un chemin dont le dernier et le premier sommet sont identiques.

III. Différents types de parcours de graphe

III.1. Parcours en profondeur d'abord

Parcours d'un graphe en profondeur d'abord

Pour un parcours en profondeur d'abord, on suit les étapes suivantes :

- On explore les sommets du graphe en passant de sommet en sommet en suivant l'un des voisins(ou successeurs) et en marquant les sommets visités.
- Lorsqu'il n'y a plus de sommets accessibles non encore visités, on revient au sommet précédent.

Ce parcours en profondeur d'abord est donc une procédure récursive qui peut se réaliser à l'aide d'une Pile.

Remarque

Suivant la façon dont on empile les voisins(ou successeurs), on peut trouver des ordres différents dans la liste des sommets lors du parcours en profondeur d'abord.

Le parcours en profondeur est le parcours typique que l'on fait lorsqu'on explore un labyrinthe.

Implémentation en Python

- en récursif :

```
def parcours_profondeur(graphe, sommet, visites = None, non_visites = None):
    if visites == None:
        visites = []
    if non_visites == None:
        non_visites = les_piles.Stack()
    if sommet not in visites:
        visites.append(sommet)
    for noeud in graphe.neighbors(sommet):
        if noeud not in visites:
            non_visites.push(noeud)
    if non_visites.is_empty():
        return visites
    else:
        return parcours_profondeur(graphe, non_visites.pop(), visites,non_visites)
```

- en itératif :

```
def parcours_profondeur_iter(graphe, sommet):
    visites = []
    non_visites = les_piles.Stack()
    non_visites.push(sommet)
       while not non_visites.is_empty():
        sommet = pile.pop()
        if sommet not in visites:
            visites.append(sommet)
            for noeud in graphe.neighbors(sommet):
                if noeud not in visites:
                    pile.push(noeud)
    return visites
```

III.2. Parcours en largeur d'abord

Parcours un graphe en largeur d'abord

Pour un parcours en largeur d'abord, on suit les étapes suivantes :

- On explore les sommets du graphe en explorant d'abord tous les voisins (ou successeurs) d'un sommet.
- Ensuite on explore les voisins des voisins(ou successeurs des successeurs).

Ce parcours en largeur d'abord est donc une procédure récursive qui peut se réaliser à l'aide d'une File.

Remarque

Suivant la façon dont on enfile les voisins(ou successeurs), on peut trouver des ordres différents dans la liste des sommets lors du parcours en largeur d'abord.

Le parcours en largeur est le parcours typique que l'on fait lorsqu'on cherche un plus court chemin.

Implémentation en Python

- en récursif :

```
def parcours_largeur(graphe,sommet, visites=None,non_visites=None):
    if visites==None:
        visites=[]
    if non_visites==None:
        non_visites=les_files.Queue()
    if sommet not in visites:
        visites.append(sommet)
    for noeud in graphe.neighbors(sommet):
        if noeud not in visites:
            non_visites.enqueue(noeud)
    if non_visites.is_empty():
        return visites
    else:
        return parcours_largeur(graphe, non_visites.dequeue(), visites, non_visites)
```

- en itératif :

```
def parcours_largeur_iter(graphe,sommet):
    visites=[]
    file=les_files.Queue()
    file.enqueue(sommet)
    while not file.is_empty():
        sommet=file.dequeue()
        if sommet not in visites:
            visites.append(sommet)
            for noeud in graphe.neighbors(sommet):
                if noeud not in visites:
                    file.enqueue(noeud)
    return visites
```

IV. Algorithmes

IV.1. sur les graphes non orienté

Chercher une chaîne élémentaire d'un sommet A à un sommet B

À partir du sommet A, on suit un parcours en profondeur pour chaque voisin.
On arrête le parcours en profondeur dès qu'il n'y a plus de voisin disponible, ou dès que le sommet B est atteint.

Recherche d'un cycle

Il suffit de :

- lister les chaînes élémentaires
- vérifier qu'un des voisins du dernier sommet de chaque chaîne est le sommet de départ.

IV.2. sur les graphes orientés

Chercher un chemin menant d'un sommet A à un sommet B

À partir du sommet A, on suit un parcours en profondeur pour chaque successeur.
On arrête le parcours en profondeur dès qu'un successeur est B ou qu'il n'y a plus de successeur disponibles.

Présence d'un circuit

Il suffit de :

- lister les chemins élémentaires
- vérifier qu'un des successeur du dernier sommet de chaque chemin est le sommet de départ.

IV.3. Implémentation en Python

Recherche d'une chaîne ou d'un chemin

```
1  def cherche_chemin(graphe, sommet_depart, sommet_arrivee):
2      pile=les_piles.Stack()
3      pile.push((sommet_depart,[sommet_depart]))
4
5      while not pile.is_empty():
6          sommet,chemin=pile.pop()
7          voisins=[noeud for noeud in graphe.neighbors(sommet) if noeud not in chemin]
8          for noeud_suivant in voisins:
9              if noeud_suivant == sommet_arrivee:
10                 return chemin+[noeud_suivant]
11             else:
12                 pile.push((noeud_suivant,chemin+[noeud_suivant]))
```

Présence d'un cycle ou d'un circuit

```
1  def presence_cycle(graphe, sommet_depart):
2      pile=les_piles.Stack()
3      pile.push((sommet_depart,[sommet_depart]))
4
5      while not pile.is_empty():
6          sommet,chemin=pile.pop()
7          distance=len(chemin)
8          voisins=[noeud for noeud in graphe.neighbors(sommet) if noeud not in chemin]
9          if distance>2:
10             if sommet_depart in graphe.neighbors(sommet):
11                 return True
12         for noeud_suivant in voisins:
13             if noeud_suivant == sommet_depart:
14                 return True
15             else:
16                 pile.push((noeud_suivant,chemin+[noeud_suivant]))
17     return False
```