Ch 18. Vocabulaire et bases des graphes.

1 Introduction

Le problème des sept ponts de Königsberg

Le problème des sept ponts de Königsberg est connu pour être à l'origine de la théorie des graphes. Résolu par Leonhard Euler en 1735, ce problème se présente de la façon suivante :

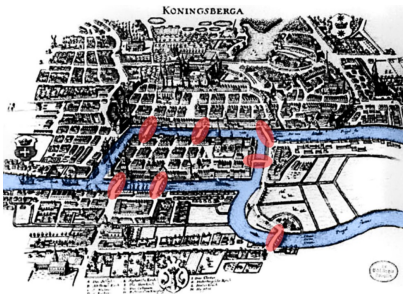


FIGURE 1 – Plan de la ville avec les ponts

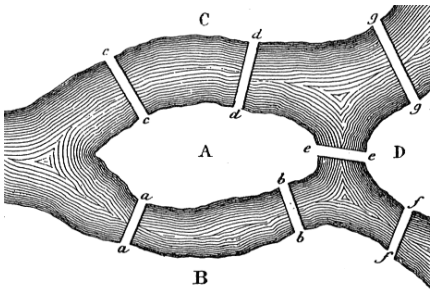


FIGURE 2 – Représentation schématisée vue du dessus

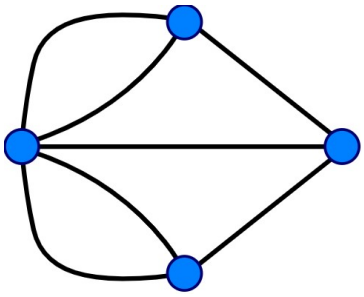


FIGURE 3 – Représentation sous forme de graphe

La ville de Königsberg (aujourd'hui Kaliningrad) est construite autour de deux îles situées sur le fleuve Pregel et reliées entre elles par un pont. Six autres ponts relient les rives de la rivière à l'une ou l'autre des deux îles, comme représentés sur le plan de la figure 1. Le problème consiste à déterminer s'il existe ou non une promenade dans les rues de Königsberg permettant, à partir d'un point de départ au choix, de passer une et une seule fois par chaque pont, et de revenir à son point de départ, étant entendu qu'on ne peut traverser le fleuve qu'en passant sur les ponts.

Les graphes, qui sont des diagrammes faits de cercles et de lignes, sont utilisés dans de nombreux domaines pour modéliser les relations entre différentes entités :

- liaisons mécaniques entre solides : le graphe de liaisons ;
- liens sur les réseaux sociaux virtuels ou réels : relations entre personnes ;
- les réseaux routiers : villes reliées par des routes ;
- les réseaux de distributions de biens, d'énergie ;
- en informatique, le routage et les relations entre des données : chaque sommet est une machine et les arêtes peuvent être une liaison physique ;
- etc.

On remarquera que certaines relations peuvent être symétriques (en mécanique, si une pièce **1** est en liaison avec une pièce **2** alors **2** est en liaison avec **1**) ou asymétriques (sur une chaine youtube, A est abonné au compte de B ne veut pas dire que B est abonné à A).

2 graphes et représentation

2.a Définitions

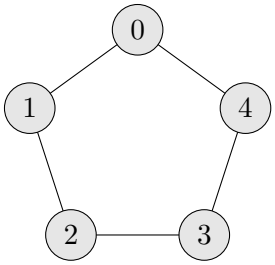
Définition :

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

- d'un ensemble S de points appelés **sommets** ;
- d'un ensemble A d'**arêtes**, tels qu'à chaque arête a_i sont associés deux éléments de S, appelés ses extrémités, et que nous noterons $[s_j, s_k]$.

Les extrémités de l'arête a_i sont dits **adjacents**.

Ces sommets sont aussi dits **voisins**. Pour les graphes **non orientés**, l'arête est une relation d'adjacence symétrique.
Les deux extrémités peuvent être distinctes ou confondues ; dans ce dernier cas, l'arête s'appelle une **boucle**.

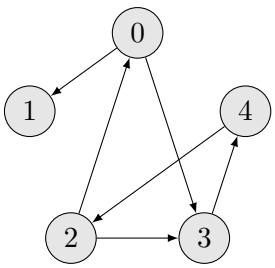


Ce graphe $G = (S, A)$ est défini par :

- $S =$
- $A =$

L'ordre des sommets

Il existe de nombreux domaines où les graphes sont orientés (circuit électrique en courant continu, plan de ville avec sens interdit, ordonnancement d'une fabrication...)
Les arêtes des graphes **orientés** sont des **arcs** dont le sens va de l'extrémité initiale ou origine à l'extrémité terminale. Dans ce cas, on les représente sous forme de flèches. Les arcs sont des relations d'adjacence asymétriques.



Ce graphe $G = (S, A)$ est défini par :

- $S =$
- $A =$

L'ordre des sommets

Définition :

Pour un graphe orienté

Un **chemin** est une suite finie de sommets telle que deux sommets consécutifs soient adjacents.

La longueur d'un chemin est le nombre d'arcs constituant le chemin.

Un **circuit** est un chemin dont l'extrémité terminale est l'extrémité initiale et toutes ses arêtes sont distinctes.

Définition :

Pour un graphe non orienté

Une **chaîne** est une suite finie de sommets telle que deux sommets consécutifs soient adjacents.

La longueur d'un chaîne est le nombre d'arêtes constituant la chaîne.

Un **cycle** est une chaîne dont l'extrémité terminale est l'extrémité initiale et toutes ses arêtes sont distinctes.

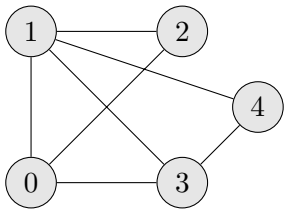
2.b Représentation graphique

Certains graphes jouent un rôle prépondérant dans la théorie des graphes. Par exemple, un graphe à zéro arête est dit **vide** s'il ne comporte que des sommets, l'ensemble A de ses arêtes est vide.

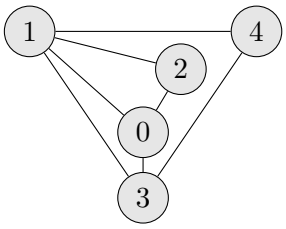
Un graphe simple (sans boucle ni arêtes parallèles) est dit **complet** si tous les sommets sont reliés entre eux.

Plusieurs représentations graphiques peuvent représenter le même graphe :

représentation 1



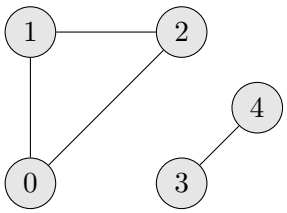
représentation 2



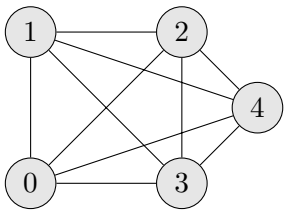
Définition :

Un graphe est dit **connexe** si pour tout couple de sommets (u, v) on trouve un chemin ou une chaîne entre u et v.

Un graphe peut être en plusieurs parties non reliées, ce graphe est non connexe.



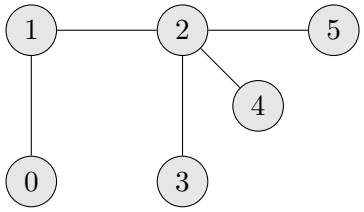
graphe non connexe



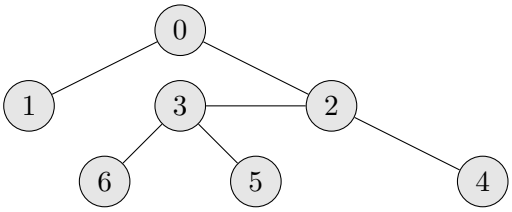
graphe complet

Définition :

Un **arbre** est un graphe connexe qui ne présente pas de cycle ou de circuit.



arbre



arbre binaire

2.c Caractéristiques d'un graphe orienté ou non orienté

Si (u, v) est un arc du graphe orienté $G = (S, A)$, on dit que l'arc (u, v) est incident aux sommets u et v ou encore que l'arc (u, v) quitte le sommet u et arrive au sommet v.
On dit que v est un **successeur** de u et que u est un **prédécesseur** de v.
Dans le cas non-orienté, on dit simplement que l'arête $\{u, v\}$ est incidente aux sommets u et v.

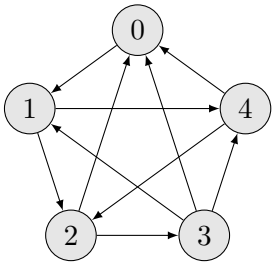
Définition :

On appelle **degré** du sommet v, noté $d(v)$, le nombre d'arêtes incidentes à ce sommet.
Le **degré d'un graphe** est le degré maximum de tous ses sommets.
L'**ordre** d'un graphe est le nombre n de sommets du graphe.

Si m est le nombre d'arêtes, pour un graphe non orienté on aura la relation :

Remarque : On peut en déduire que le nombre de sommets de degré impair est forcément pair.

On désigne par **demi-degré extérieur** d'un sommet v d'un graphe orienté, noté $d_+(v)$, le nombre d'arcs qui quittent v.
On désigne par **demi-degré intérieur** d'un sommet v d'un graphe orienté, noté $d_-(v)$, le nombre d'arcs qui arrivent en v.



Par exemple pour ce graphe on a :

$d_+(3) =$

$d_-(3) =$

$d(3) =$

Pour le graphe orienté représenté plus haut, nous avons comme chemin possible entre les sommets 3 et 2 :

$\chi(3, 2) =$

Une chaîne est **élémentaire** si chaque sommet du graphe apparait au plus une fois.

Une chaîne est **simple** si chaque arête du graphe apparait au plus une fois.

Définition :

> Une chaîne (resp. un chemin) est eulérienne si elle contient une fois et une seule chaque arête du graphe ; si la chaîne est un cycle, on l'appelle cycle eulérien.

3 Implémentation

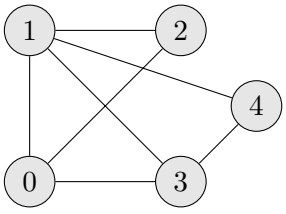
3.a Liste d'adjacence

Une liste d'adjacence d'un graphe non orienté est une liste de listes G qui possède les propriétés suivantes :

- elle est constituée de n listes correspondant aux n sommets du graphe ;
- la liste `G[i]` contient tous les voisins de i.

On a donc la représentation du graphe :

graphe 1



Dans le cas d'un graphe orienté, la liste d'adjacence contient la liste des successeurs.

3.b Dictionnaire d'adjacence

Un dictionnaire d'adjacence d'un graphe non-orienté est un dictionnaire D de listes qui possède les propriétés suivantes :

— il est constitué de n listes correspondant aux n sommets du graphe ;

— $D[s_i]$ est une liste contenant tous les voisins de s_i.

On a donc la représentation du graphe 1 :

L'avantage d'un dictionnaire d'adjacence est la possibilité de nommer les sommets. On peut l'adapter de la même manière que la liste par adjacence pour représenter un graphe orienté.

3.c Matrice d'adjacence

Une matrice d'adjacence d'un graphe non orienté est une matrice `M` qui possède les propriétés suivantes :

— elle est de taille $n \times n$, n étant le nombre de sommets du graphe ;

— $M[i][j] = 1$ s'il y a une arête entre s_i et s_j ;

— $M[i][j] = 0$ s'il n'existe pas d'arête entre s_i et s_j.

On a donc la représentation du graphe 1 :

```
M = [[0, 1, 1, 1, 0],
     [1, 0, 1, 1, 1],
     [1, 1, 0, 0, 0],
     [1, 1, 0, 0, 1],
     [0, 1, 0, 1, 0]]
```

On remarquera qu'un graphe non-orienté possède forcément une matrice d'adjacence symétrique.
Si le graphe présente une boucle sur le sommet i, alors la valeur $M[i][i] = 2$.

On peut remplacer les valeurs de la matrice par les booléens `True` et `False`.

```
M = [[False, True, True, True, False],
     [True, False, True, True, True],
     [True, True, False, False, False],
     [True, True, False, False, True],
     [False, True, False, True, False]]
```

On peut adapter cette matrice pour représenter un graphe orienté : on remplace les arêtes $\{s_i, s_j\}$ par les arcs (s_i, s_j).
Cette matrice peut-être codée avec une liste de listes ou un tableau `numpy`.

4 Graphe pondéré

On peut enrichir la notion de graphe en attribuant des poids aux arcs et aux arêtes. Concrètement, ces poids peuvent représenter des distances, des temps de parcours, des coûts de transport, des coûts de construction, etc.

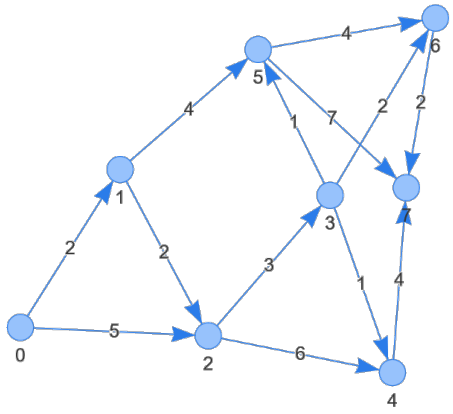


FIGURE 4 – Exemple de représentation d'un graphe orienté pondéré

4.a Vocabulaire

Définition :

Un arc pondéré est un couple (arc, poids).
Une arête pondérée est un couple (arête, poids).

Définition :

Un graphe orienté $G = (S, A)$ pondéré est un couple de sommets et d'arcs pondérés.
Un graphe non-orienté $G = (S, A)$ pondéré est un couple de sommets et d'arêtes pondérées.

4.b Matrice de pondération

Une matrice de pondération est une matrice d'adjacence adaptée :

— $M[i][j] = w$ s'il y a un arc de poids w d'extrémité initiale s_i et d'extrémité terminale s_j ;

— $M[i][j] = 0$ (ou $M[i][j] = \infty$) s'il n'existe pas un arc d'extrémité initiale s_i et d'extrémité terminale s_j.

On a donc la représentation du graphe de la figure 4 :

```
M = [[0, 2, 5, 0, 0, 0, 0, 0],
     [0, 0, 2, 0, 0, 4, 0, 0],
     [0, 0, 0, 3, 6, 0, 0, 0],
     [0, 0, 0, 0, 1, 0, 2, 0],
```

```
        [0, 0, 0, 0, 0, 0, 0, 4],
        [0, 0, 0, 1, 0, 0, 4, 7],
        [0, 0, 0, 0, 0, 0, 0, 2],
        [0, 0, 0, 0, 0, 0, 0, 0]]
```

Comme précédemment, on peut coder cette matrice avec un dictionnaire de dictionnaires afin de pouvoir utiliser des noms plus explicites au niveau des sommets.
On a donc la représentation du graphe de la figure 4 :

```
D = {0: {0: 0, 1: 2, 2: 5, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0},
     1: {0: 0, 1: 0, 2: 2, 3: 0, 4: 0, 5: 4, 6: 0, 7: 0},
     2: {0: 0, 1: 0, 2: 0, 3: 3, 4: 6, 5: 0, 6: 0, 7: 0},
     3: {0: 0, 1: 0, 2: 0, 3: 0, 4: 1, 5: 0, 6: 2, 7: 0},
     4: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 4},
     5: {0: 0, 1: 0, 2: 0, 3: 1, 4: 0, 5: 0, 6: 4, 7: 7},
     6: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 2},
     7: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0}}
```

On peut l'adapter pour représenter un graphe non-orienté : on remplace les arcs (s_i, s_j) par les arêtes $\{s_i, s_j\}$. Comme précédemment, une matrice de pondération d'un graphe non-orienté est forcément symétrique.

5 Applications

5.a Exemple d'implémentation d'après un sujet de l'XENS 2015

On souhaite stocker en mémoire une liste non-ordonnée d'au plus n entiers sans redondance (i.e. ou aucun entier n'apparaît plusieurs fois). Nous utilisons un tableau liste de longueur $n+1$ tel que :

- liste`[0]` contient le nombre d'éléments dans le tableau ;
- liste`[i]` contient le $i^{ème}$ élément de la liste non-ordonnée avec $1 \leqslant i \leqslant liste$`[0]`.

Nous disposons d'une fonction `creerListeVide(n:int)->list` qui permet de créer une liste pouvant contenir n éléments.

Python shell

```
>>> creerListeVide(3)
[0,None,None,None]
```

Nous disposons d'une fonction `estDansListe(liste:list,x)->bool` qui renvoie $True$ si l'élément x est dans la liste et $False$ sinon.
La complexité de cette fonction est linéaire en fonction du nombre d'élément maximum que peut contenir la liste. Nous disposons aussi d'une fonction `ajouteDansListe(liste:list,x)` qui modifie la liste pour ajouter l'élément x s'il n'est pas présent et ne fait rien sinon. La complexité est aussi linéaire du nombre d'élément maximum que peut contenir la liste.

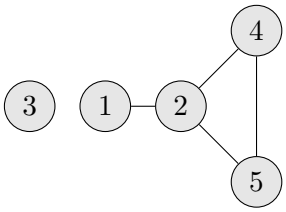
Un plan P est défini par un ensemble de n villes numérotées de 1 à n et un ensemble de m routes (toutes à double sens) reliant chacune deux villes. On dira que deux villes $x, y \in n$ sont voisines lorsqu'elles sont reliées par une route, ce que l'on notera (x, y). On appellera chemin de longueur $k-1$ toute suite de villes `[`$v_1, v_2 ..., v_k$`]`. Un plan P est alors un graphe dont les sommets sont les villes et les routes les arêtes.

Structure de données. Nous représentons tout plan P à n villes par un tableau plan de $(n+1)$ tableaux où :

— plan`[0]` contient un tableau de deux éléments où :

. plan`[0][0]` a pour valeur le nombres n de villes ;

. plan`[0][1]` a pour valeur le nombres m de routes.



Graphe d'un plan à 5 villes et 4 routes

```
plan1= [[5,4] ,
        [1,2,None,None,None],
        [3,4,1,5,None],
        [0,None,None,None,None],
        [2,2,5,None,None],
        [2,4,2,None,None]]
```

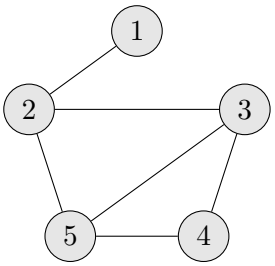
— Pour toute ville $v \in n$, plan`[v]` contient un tableau à n éléments représentant la liste à au plus $(n-1)$ éléments des villes voisines de v dans P dans un ordre arbitraire en utilisant la structure de liste sans redondance décrite plus haut. Ainsi :

. plan`[v][0]` a pour valeur le nombre de villes voisines de v ;

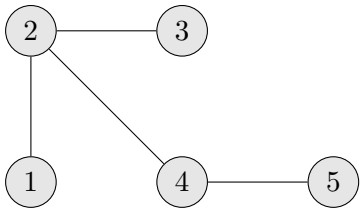
. plan`[v][1]` , ... ,n`[v][n[v][0]]` sont les indices des villes voisines de v

Question 1 : Représenter sous forme de tableaux les deux plans suivant :

Plan1



Plan2



Question 2 Écrire une fonction `creerPlanSansRoute(n:int)` qui crée, remplit et renvoie le tableau de tableaux correspondant au plan à n villes n'ayant aucune route.

Question 3 Écrire une fonction `estVoisine(plan,x,y)` qui renvoie $True$ si les villes x et y sont voisines dans le plan codé par le tableau de tableaux plan et renvoie $False$ sinon.

Question 4 Écrire une procédure `ajouteRoute(plan,x,y)` qui modifie le tableau de tableaux plan pour ajouter une route entre les villes x et y si elle n'était pas déjà présente et ne fait rien sinon. On prendra garde à bien mettre à jour toutes les cases concernées dans le tableau de tableaux plan. Y a-t-il un risque de dépassement de la capacité des listes ?

Question 5 Écrire une procédure `afficheToutesLesRoutes(plan)` qui affiche à l'écran la liste des routes du plan codé par le tableau de tableaux plan où chaque route n'apparaît qu'une seule fois. Par exemple, pour le graphe codé par le tableau de tableaux de la présentation votre procédure pourra afficher :

```
Ce plan contient 4 route(s) : (1-2) (2-4) (2-3) (4-5)
```

Quelle est la complexité de votre procédure ?

5.b Graphe d'intervalles : Énigme de Claude BERGE

A l'époque, le Duc de Densmore avait été tué par l'explosion d'une bombe artisanale, qui avait également détruit le château de Densmore où il s'était retiré. Les journaux d'alors relataient que le testament, détruit lui aussi par l'explosion, avait tout pour déplaire à l'une de ses huit ex-femmes. Or, avant sa mort, le Duc les avait toutes invitées à passer quelques jours dans sa retraite écossaise.
Les suspects sont donc ces huit femmes présentes sur l'île au moment des faits. Chacune n'y ayant effectué qu'un seul séjour. L'une d'entre elles a dû se cacher à un moment propice pour aller préparer et poser la bombe.
Les faits étant assez anciens, les suspects se souviennent des personnes qu'elles ont croisées, mais pas dans quel ordre. Voici les témoignages :

— Ann a déclaré y avoir rencontré Betty, Cynthia, Emily, Felicia et Georgia ;

— Betty a déclaré y avoir rencontré Ann, Cynthia et Helen ;

— Cynthia a déclaré y avoir rencontré Ann, Betty, Diana, Émily et Helen ;

— Diana a déclaré y avoir rencontré Cynthia et Emily ;

— Emily a déclaré y avoir rencontré Ann, Cynthia, Diana et Felicia ;

— Felicia a déclaré y avoir rencontré Ann et Emily ;

— Georgia a déclaré y avoir rencontré Ann et Helen ;

— Helen a déclaré y avoir rencontré Betty, Cynthia et Georgia.

Précision supplémentaire, personne d'autre n'est venu sur l'île. Par ailleurs si deux femmes ne se sont pas vues, c'est soit qu'elles n'étaient pas l'île en même temps, soit que l'une d'entre elles était cachée dans le labyrinthe et préparait la bombe artisanale ...

L'inspecteur qui vous a précédé sur l'affaire a constaté que les témoignages concordaient et avaient conclue à la culpabilité du majordome mort dans l'explosion avec le duc. Mais ce n'est pas le cas. Pouvez-vous trouver le coupable ?

Références :

T. Kovaltchouk, *Informatique Commune PCSI*, Reims
UPSTI, *Informatique Commune*

Plan du cours

1 Introduction **1**

2 graphes et représentation **2**

2.a Définitions 2

2.b Représentation graphique 3

2.c Caractéristiques d'un graphe orienté ou non orienté 4

3 Implémentation **5**

3.a Liste d'adjacence 5

3.b Dictionnaire d'adjacence 6

3.c Matrice d'adjacence 6

4 Graphe pondéré **7**

4.a Vocabulaire 7

4.b Matrice de pondération 7

5 Applications **8**

5.a Exemple d'implémentation d'après un sujet de l'XENS 2015 8

5.b Graphe d'intervalles : Énigme de Claude BERGE 10