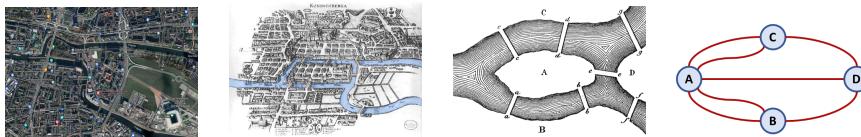




Base des graphes

10.1 Introduction

Historiquement, le problème des sept ponts de Königsberg (Kaliningrad) était de déterminer s'il existe une promenade qui partait de n'importe quel quartier et qui y revenait, tout en passant une et une seule fois par chacun des ponts. Euler mis en forme le problème et le résolut... en montrant que dans ce cas, il n'existe pas de solution.



Les applications des graphes sont assez nombreuses :

- recherche d'itinéraire pour aller d'un point à un autre;
- routage des paquets sur Internet;
- analyse des réseaux sociaux...

10.2 Vocabulaire des graphes

Définition – Graphe

Un graphe est un ensemble de **sommets** et **relations** entre ces sommets. Lorsque deux sommets sont en relation, on dit qu'il existe une **arête** entre ces sommets.

Définition – Graphe non orienté – Arêtes

Un graphe non orienté G est un couple $G = (S, A)$, où S est un ensemble fini de sommets (appelés aussi noeuds) et où A est un ensemble fini de paires ordonnées de sommets, appelées arêtes.

On note $x - y$ l'arête $\{x, y\}$. x et y sont les deux extrémités de l'arête.

- Vocabulaire des graphes : graphe orienté, graphe non orienté. Sommet (ou nœud); arc, arête. Boucle. Degré (entrant et sortant). Chemin d'un sommet à un autre. Cycle. Connexité dans les graphes non orientés.
- Notations : graphe $G = (S, A)$, degrés $d(s)$ (pour un graphe non orienté), $d_+(s)$ et $d_-(s)$ (pour un graphe orienté).
- Pondération d'un graphe. Étiquettes des arcs ou des arêtes d'un graphe. On ajoute d'information à un graphe par des exemples concrets.

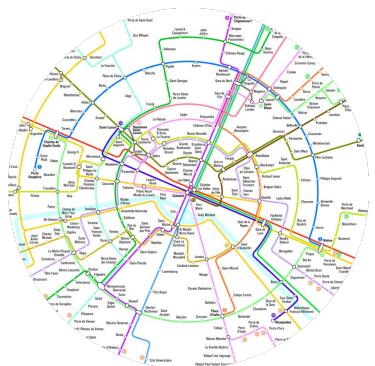


FIGURE 10.1 – Représentation circulaire du métro parisien

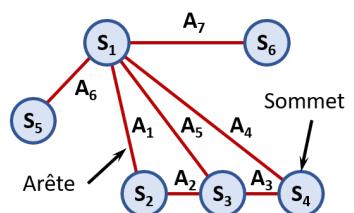


FIGURE 10.2 – Graphe non orienté

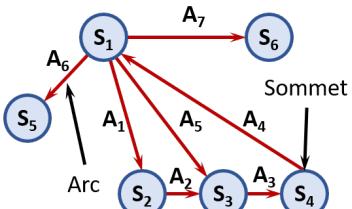


FIGURE 10.3 – Graphe orienté

Définition – Graphe orienté – Arcs

[ref_01] Un graphe orienté G est un couple $G = (S, A)$, où S est un ensemble fini de sommets et où A est un ensemble fini de paires ordonnées de sommets, appelées arcs.

On note $x \rightarrow y$ l'arc (x, y) . x est l'extrémité initiale de l'arc, y est son extrémité terminale. On dit que y est successeur de x et que x est prédécesseur de y .

Remarque

On peut noter le graphe non orienté $G = ([1, 6], E)$ où $E = (\{1, 2\}, \{2, 3\}, \{3, 4\}, \{1, 4\}, \{1, 3\}, \{1, 5\}, \{1, 6\})$ désigne les arêtes.

On peut noter le graphe orienté $G = ([1, 6], E)$ où $E = ((1, 2), (2, 3), (3, 4), (1, 4), (1, 3), (1, 5), (1, 6))$ désigne les arcs.

Définition – Adjacence

Deux arcs (resp. arêtes) d'un graphe orienté (resp. non orienté) sont dits adjacents s'ils ont au moins une extrémité commune.

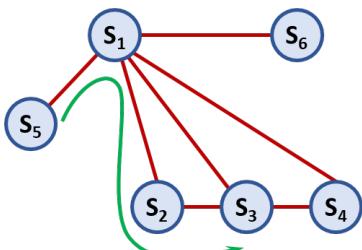
Deux sommets d'un graphe non orienté sont dits adjacents s'il existe une arête les joignant.

Dans un graphe orienté, le sommet y est dit adjacent au sommet x s'il existe un arc $x \rightarrow y$.

Définition – Graphes pondérés

Étiqueter les arêtes d'un graphe (S, A) (orienté ou non), c'est se donner une fonction $f : A \rightarrow V$ (où V est un ensemble de valeurs). On dit qu'un graphe est pondéré si ses arêtes sont étiquetées par des nombres. On parlera alors du poids d'une arête.

10.3 Chemins



Définition – Chemin dans un graphe

On appelle chemin dans un graphe une suite finie $\{S_0, \dots, S_{n-1}\}$ de n sommets tels que pour tout $i \in \llbracket 0, n-1 \rrbracket$, une arête relie S_i à S_{i+1} . On dit que ce chemin relie le sommet de départ S_0 au sommet de fin S_{n-1} .

Dans le cas d'un graphe non orienté, les arêtes sont notées $\{S_i, S_{i+1}\}$ pour $i \in \llbracket 0, n-1 \rrbracket$.

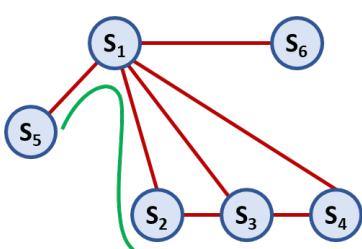
Dans le cas d'un graphe orienté, les arcs sont notées (S_i, S_{i+1}) pour $i \in \llbracket 0, n-1 \rrbracket$.

Définition – Chemin fermé

Chemin dont le sommet de départ et le sommet d'arrivée sont identiques.

Définition – Chemin élémentaire

Chemin n'empruntant que des arêtes distinctes.



Définition – Chemin simple

Chemin tel que les $n - 2$ sommets intermédiaires si, pour $i \in \llbracket 1, n - 1 \rrbracket$ soient deux à deux distincts et tous distincts du sommet de départ S_0 et du sommet d'arrivée S_{n-1} et tels que ce chemin n'est pas de la forme a, b, a dans le cas non-orienté.

Définition – Circuit

Chemin fermé de longueur non nulle.

Définition – Cycle

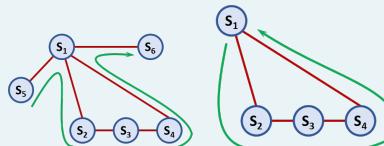
Circuit élémentaire (chemin fermé de longueur non nulle dont toutes les arêtes sont distinctes).

Définition – Cycle simple

Chemin fermé et simple de longueur non nulle.

Définition – Chemin et cycle eulérien

Chemin (resp. cycle) contenant une et une seule fois toutes les arêtes du graphe.



Remarque

Pour certains auteurs, un chemin élémentaire est ce que nous avons appelé un chemin simple et réciproquement. Pour d'autres, un cycle est ce que nous avons appelé un cycle simple.

Définition – Connexité dans les graphes non orientés

Un graphe $G = (S, A)$ est dit connexe si, pour deux sommets quelconques S_i et S_j de S , il existe un chemin de S_i à S_j .

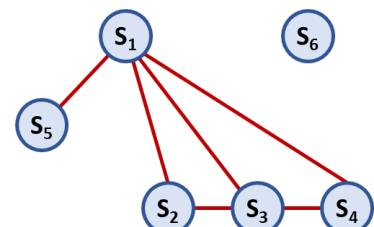


FIGURE 10.4 – Graphe ayant 2 composantes connexes

10.3.1 Notations

Définition – Degré d'un sommet

On appelle degré d'un sommet s et on note $d(s)$ le nombre d'arcs (ou d'arêtes) dont s est une extrémité.

Définition – Degré entrant et sortant

On note s le sommet d'un graphe orienté. On note :

- ▶ $d_+(s)$ le demi-degré extérieur de s , c'est-à-dire le nombre d'arcs ayant leur extrémité initiale en s (ces arcs sont dits incidents à s vers l'extérieur);
- ▶ $d_-(s)$ le demi-degré intérieur de s , c'est-à-dire le nombre d'arcs ayant leur

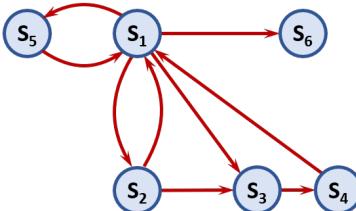


FIGURE 10.5 – Graphe orienté

extrémité finale en s (ces arcs sont dits incidents à s vers l'intérieur).
Dans ce cas, on a $d^\circ(s) = d_-(s) + d_+(s)$.

Exemple –

- $d_-(S_1) = 3$.
- $d_+(S_1) = 4$.
- $d^\circ(S_1) = 7$.

10.4 Implémentation des graphes

10.4.1 Liste d'adjacence

Définition – Liste d'adjacence

Soit un graphe de n sommets d'indices $i \in \llbracket 0, n - 1 \rrbracket$. Pour implémenter le graphe, on utilise une liste G de taille n pour laquelle, $G[i]$ est la liste des voisins de i .

Remarque

Cette implémentation est plutôt réservée au graphes « creux », c'est-à-dire ayant peu d'arêtes.

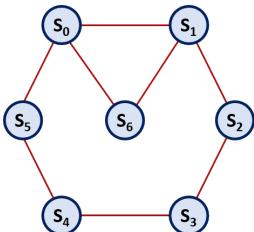


FIGURE 10.6 – Graphe non orienté

Exemple – Graphe non orienté

Dans ce cas S_0 est voisin de S_1, S_5 et S_6 ; donc $G[0]=[1,5,6]$. S_3 est voisin de S_2 et S_4 ; donc $G[3]=[2,4]$.

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

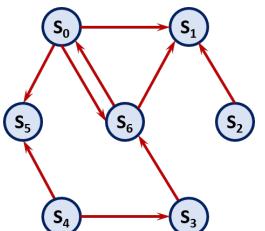


FIGURE 10.7 – Graphe orienté

Exemple – Graphe orienté

Dans ce cas, le graphe est orienté. La liste d'adjacence contient la liste des successeurs. Ainsi, les successeurs de S_0 sont S_1, S_5 et S_6 ; donc $G[0]=[1,5,6]$. S_1 n'a pas de successeur donc $G[1]=[]$.

```
1 | G = [[1,5,6],[],[1],[6],[3,5],[],[0,1]]
```

10.4.2 Dictionnaire d'adjacence

Dans la même idée, il est aussi possible d'utiliser des dictionnaires d'adjacence dans lequel les clés sont les sommets (qui peuvent être des entiers ou des chaînes de caractères), et les valeurs sont des listes de voisins ou de successeurs (qui peuvent donc être des listes d'entiers ou de chaînes de caractères).

```

1 # Graphe non orienté
2 G = {"S0":["S1","S5","S6"], "S1":["S0","S2","S6"], "S2":["S1","S3"], "S3":["S2",""
    "S4"], "S4":["S3","S5"], "S5":["S4","S0"], "S6":["S1","S0"]}
3 # Graphe orienté
4 G = {"S0":["S1","S5","S6"], "S1":[], "S2":["S1"], "S3":["S6"], "S4":["S3","S5"], ""
    "S5":[], "S6":["S1","S0"]}

```

10.4.3 Matrice d'adjacence

Définition – Matrice d'adjacence

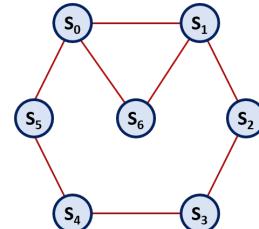
Soit un graphe de n sommets d'indices $i \in \llbracket 0, n - 1 \rrbracket$ et E l'ensemble des arêtes (on notera $G = (\llbracket 0, n - 1 \rrbracket, E)$). Pour implémenter le graphe, on utilise la matrice d'adjacence carrée de taille n , \mathcal{M}_n G de taille n pour laquelle, $m_{i,j} = \begin{cases} \text{True} & \text{si } \{i, j\} \in E \\ \text{False} & \text{sinon} \end{cases}$ avec $i, j \in \llbracket 0, n - 1 \rrbracket$.

Remarque

Cette implémentation est plutôt réservée au graphes « denses » ayant « beaucoup » d'arêtes.

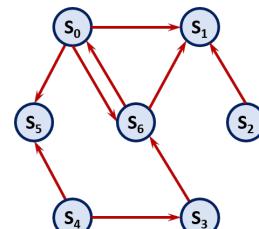
Exemple – Graphe non orienté

$$M = \begin{pmatrix} \text{False} & \text{True} & \text{False} & \text{False} & \text{False} & \text{True} & \text{True} \\ \text{True} & \text{False} & \text{True} & \text{False} & \text{False} & \text{False} & \text{True} \\ \text{False} & \text{True} & \text{False} & \text{True} & \text{False} & \text{False} & \text{False} \\ \text{False} & \text{False} & \text{True} & \text{False} & \text{True} & \text{False} & \text{False} \\ \text{False} & \text{False} & \text{False} & \text{True} & \text{False} & \text{True} & \text{False} \\ \text{True} & \text{False} & \text{False} & \text{False} & \text{True} & \text{False} & \text{False} \\ \text{True} & \text{True} & \text{False} & \text{False} & \text{False} & \text{False} & \text{False} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



Exemple – Graphe orienté

$$M = \begin{pmatrix} \text{False} & \text{True} & \text{False} & \text{False} & \text{False} & \text{True} & \text{True} \\ \text{False} & \text{False} & \text{False} & \text{False} & \text{False} & \text{False} & \text{False} \\ \text{False} & \text{True} & \text{False} & \text{False} & \text{False} & \text{False} & \text{False} \\ \text{False} & \text{False} & \text{False} & \text{False} & \text{False} & \text{False} & \text{True} \\ \text{False} & \text{False} & \text{False} & \text{True} & \text{False} & \text{True} & \text{False} \\ \text{False} & \text{False} & \text{False} & \text{False} & \text{False} & \text{False} & \text{False} \\ \text{True} & \text{True} & \text{False} & \text{False} & \text{False} & \text{False} & \text{False} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



Remarque

- ▶ Dans le cas d'un graphe non orienté, la matrice est symétrique.
- ▶ Si on avait un bouclage sur un sommet, il y aurait des valeurs non nulles sur la diagonale.

10.4.4 Graphes pondérés

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

- ▶ $M[i][j] = w_{ij}$ s'il y a un arc de poids w_{ij} d'extrémité initiale s_i et d'extrémité terminale s_j ;

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

10.4.5 Taille d'un graphe

Définition –

Taille d'un graphe Soit un graphe $G = (V, E)$ composé de sommets V et d'arêtes E . On appelle taille du graphe la quantité $|V| + |E|$.

Remarque

On retrouve dans des ouvrages la notation $|V|$ ou $|E|$ correspondant respectivement au nombre de sommets et d'arêtes. Ainsi, si un algorithme est linéaire en la taille du graphe, on pourra utiliser la notation $\Theta(|V| + |E|)$.

10.4.6 Formule des degrés

Définition – Formule des degrés

Soit un graphe $G = (V, E)$, alors : $\sum_{v \in V} \deg(v) = 2|E|$.

10.4.7 Graphe complet

Définition – Graphe complet

Un graphe complet est un graphe possédant toutes les arêtes possibles.

Remarque

- Le nombre maximal d'arêtes d'un graphe à n sommets est donné par : $\binom{n}{2}$.
- Pour chaque sommet v on a $d^o(v) = n - 1$.
- Le nombre d'arêtes est en $\Theta(|V|^2)$.

10.4.8 Matrices d'adjacences

Avantages

- Le test d'adjacence de deux sommets se fait en temps constant.
- Dans le cas d'un graphe orienté, obtenir les prédécesseurs n'est pas plus compliqué (ni plus coûteux) que d'obtenir les successeurs.
- Rajouter ou supprimer une arête (ou un arc) peut se faire en temps constant.

Inconvénients

- Ajouter ou supprimer un sommet nécessite un temps proportionnel à n^2 .
- Pour récupérer les voisins/successeurs d'un nœud, il faut parcourir toute la ligne de la matrice : l'opération se fait donc en $\Theta(n)^1$, ce qui est regrettable si le degré sortant du noeud est petit devant n .
- On consomme une mémoire proportionnelle à $|V|^2$, même quand la taille du graphe est de l'ordre de n (graphe creux).

1: $\Theta(n)$

10.4.9 Listes d'adjacences

Avantages

- ▶ La mémoire utilisée est de l'ordre de $|E| + |V|$, ce qui est nettement mieux que $|V|^2$ si le graphe est creux.
- ▶ On a directement accès à la liste des voisins d'un noeud : la parcourir prend un temps proportionnel au degré sortant du noeud.
- ▶ Ajouter un noeud peut normalement se faire en temps $|V|$ (si l'ajout n'impose pas de renuméroter les noeuds déjà présents).

Inconvénients

- ▶ Le test d'adjacence ne se fait plus en temps constant (mais en temps proportionnel au degré du noeud).
- ▶ Si le graphe est dense, on consommera plus de mémoire (d'un facteur constant) qu'avec une matrice d'adjacence.
- ▶ Ajouter ou supprimer une arête n'est pas aussi évident que dans une matrice d'adjacence : suivant l'opération précise que l'on souhaite faire, la complexité peut être unitaire ou proportionnelle aux degrés des noeuds impactés.
- ▶ Supprimer un noeud n'est pas pratique : le plus simple est de reconstruire entièrement le graphe (en un temps $\mathcal{O}(|E| + |V|)$).

10.5 Applications

10.5.1 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^{e} élément de la liste non-ordonnée avec $1 \leq i \leq \text{liste}[0]$.

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

```
1 | >>> creerListeVide(3)
2 | [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.

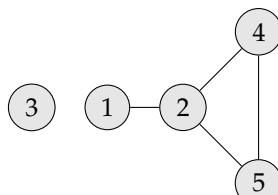
La complexité de cette fonction est linéaire en fonction du nombre d'élément maximum que peut contenir la liste. Nous disposons aussi d'une fonction `ajouteDansListe(liste : list, x)` qui modifie la liste pour ajouter l'élément x s'il n'est pas présent et ne fait rien sinon. La complexité est aussi linéaire 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 $[v_1, v_2, \dots, v_k]$. 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 :

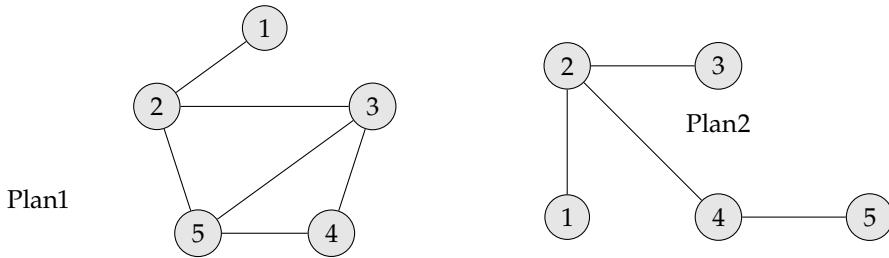
- ▶ plan[0] contient un tableau de deux éléments où :
 - plan[0][0] a pour valeur le nombres n de villes ;
 - plan[0][1] a pour valeur le nombres m de routes.
- ▶ Pour toute ville $v \in n$, plan[v] contient un tableau à n éléments représentant la liste à au plus $(n - 1)$ éléments des villes voisines 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], ..., plan[v][n[v][0]] sont les indices des villes voisines de v .



Graphe d'un plan à 5 villes et 4 routes

```
1 | plan1= [[5,4] ,
2 | [1,2,None,None,None] ,
3 | [3,4,1,5,None] ,
4 | [0,None,None,None,None] ,
5 | [2,2,5,None,None] ,
6 | [2,4,2,None,None]]
```

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



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 :

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

Question 6 Quelle est la complexité de votre procédure ?