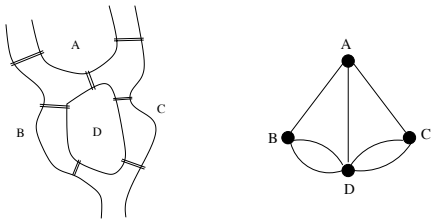
NOTIONS SUR LES GRAPHES

La théorie des graphes est née en 1736 quand Euler démontra qu'il était impossible de traverser chacun des sept ponts de la ville russe de Königsberg (aujourd'hui Kaliningrad) une fois exactement et de revenir au point de départ. Les ponts enjambent les bras de la Pregel qui coulent de part et d'autre de l'île de Kneiphof. Dans la figure suivante, les noeuds représentent les rives.



Les 7 ponts de Königsberg

De manière générale, un graphe permet de représenter simplement la structure, les connexions, les cheminements possibles d'un ensemble complexe comprenant un grand nombre de situations, en exprimant les relations, les dépendances entre ses éléments. Les graphes sont très utiles pour modéliser les réseaux de communication, les réseaux ferroviaire ou routier, les arbre généalogiques, les diagrammes de succession de tâches en gestion de projet, etc ...

En plus de son existence purement mathématique, le graphe est aussi une structure de données puissante pour l'informatique.

1 Définitions

Il existe deux types de graphes, les **graphes orientés** et les **graphes non-orientés**.

1.1 Graphes orientés

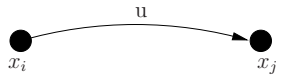
Soit S = $\{x_1, x_2, ..., x_n\}$ un ensemble fini de cardinal n, dont les éléments seront par la suite appelés **sommets**.

Soit U une partie de S × S, formée donc de couples (x_i, x_j). U est appelé ensemble des **arcs**.

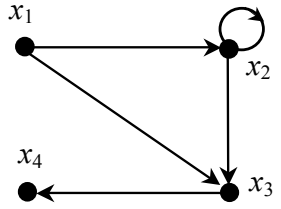
Définition : graphe orienté

Un **graphe orienté** est un couple G = (S, U) où S est un ensemble fini appelé **ensemble des sommets** et où U est une partie de S × S, appelé **ensemble des arcs** associé à S.

Pour un arc $u = (x_i, x_j)$, x_i est l'origine et x_j est la destination. L'arc u part de x_i et arrive à x_j. Graphiquement, l'arc u se représente de la manière suivante (diagramme sagittal) :



Exemple : prenons S = $\{x_1, x_2, x_3, x_4\}$ et U = { (x_1, x_2), (x_1, x_3), (x_1, x_4), (x_2, x_2), (x_2, x_3) }. Le graphe G = (S, U) est représenté sur la figure suivante :

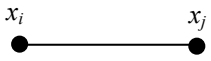


Un arc (x_i, x_i) s'appelle une **boucle**.

1.2 Graphes non-orientés

Dans un graphe non-orienté, U est un ensemble formé uniquement de singletons de type $\{x_i\}$ ou de paires $\{x_i, x_j\}$ d'éléments de S. U est appelé ensemble d'**arêtes**. Un graphe non orienté est alors simplement défini comme un couple G = (S, U).

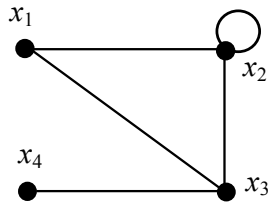
Une arête $\{x_i, x_j\}$ est représentée par un simple trait liant les deux sommets x_i et x_j.



Par exemple, la version non-orientée du graphe précédent est donnée par S = $\{x_1, x_2, x_3, x_4\}$ et :

$$\text{U} = \{\ \{x_2\},\ \{x_1, x_2\},\ \{x_1, x_3\},\ \{x_1, x_4\},\ \{x_2, x_3\}\ \}$$

Ce graphe est représenté ci-dessous :



1.3 Sommets adjacents (ou voisins)

Définition

Deux sommets sont dits **adjacents** (ou **voisins**) s'ils sont joints par un arc (dans le cas orienté) où une arête (dans le cas non-orienté).

1.4 Représentation par une matrice d'adjacence

La représentation d'un graphe dans un ordinateur peut se faire à l'aide d'une **matrice d'adjacence** A [1].

Dans ce but, étant donné la bijection qui existe entre tout ensemble fini S = $\{x_1, x_2, ...x_n\}$ et l'intervalle entier $[\![0, n-1]\!]$, l'ensemble des sommets sera dorénavant cet intervalle entier. Un sommet sera donc un entier naturel compris entre 0 et $n-1$.

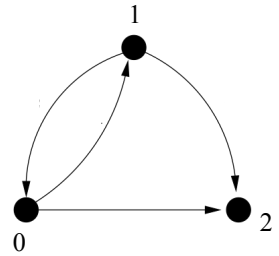
Définition (matrice d'adjacence)

Étant donné un graphe G = (S, U) (orienté ou non), avec S = $[\![0, n-1]\!]$, la **matrice d'adjacence** A est la matrice $n \times n$ définie par :

$$\text{A}_{ij} = \begin{cases} 1 & \text{s'il existe un arc ou une arête entre } i \text{ et } j \\ 0 & \text{sinon} \end{cases}$$

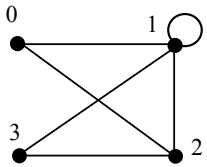
Exemples :

1) Quelle est la matrice d'adjacence associée au graphe représenté ci-après ?



2) Quelle est la matrice d'adjacence associée au graphe non-orienté ci dessous ?

1. Une deuxième possibilité est d'utiliser des listes d'adjacence, dont nous ne parlerons pas ici, mais que les étudiants en option informatique peuvent avoir déjà étudié.



3) Quelle est la propriété d'une matrice d'adjacence associée à un graphe non-orienté ?

Cas d'un graphe pondéré

Un graphe peut être **pondéré**. Dans un tel graphe, les différents arcs ou arête sont affectés de **poids** qui sont des nombres représentant une propriété.

Par exemple, si un graphe a pour sommets des villes et pour arêtes des routes entre chaque ville, alors on peut décider que chaque arête aura comme poids le nombre de kilomètres entre les deux villes qu'elle relie.

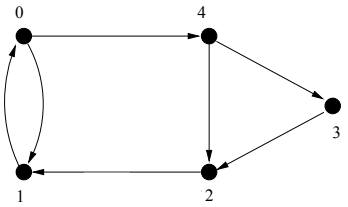
Dans le cas où le graphe est pondéré, on peut convenir de remplacer les 1 par les poids w_{ij} associé à l'arc (ou l'arête) reliant i et j et les 0 par un nombre qui est différent de tous les poids w_{ij} rencontrés.

$$\mathrm{A}_{ij} = \begin{cases} w_{ij} & \text{s'il existe un arc ou une arête entre } i \text{ et } j \\ a & \text{sinon} \end{cases}$$

avec $\forall\,(i,j),\ a \neq w_{ij}$.

1.5 Chemin dans un graphe

Un chemin entre deux sommets i et j d'un graphe G est une suite d'arcs (ou d'arêtes) qui relient ces deux sommets. i est l'origine du chemin et j en est l'extrémité. Par exemple :

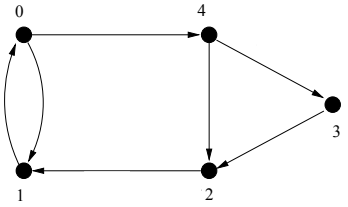


$<(0,4),(4,3),(3,2)>$ est un chemin entre 0 et 2.

1.6 Arbre des chemins associé à un sommet

Étant donné un sommet `s`, on peut construire un **arbre des chemins** dont ce sommet est le point de départ. La raison d'être de cet arbre est de nous indiquer quels sont les sommets qui sont accessibles en prenant `s` comme point de départ. On dit que `s` est la **racine** de cet arbre. Ce concept marche pour des graphes orientés et non-orientés.

Version naïve de la construction de l'arbre
Version efficace de la construction de l'arbre



Exemple : construire l'arbre des chemins associé au sommet 0, puis celui associé au sommet 3.

1.7 Application

Téléchargez le programme graphe.py sur le site de la mp1. Celui-ci contient :

- Un graphe simple (c'est à dire non pondéré) G1 = (S1, U1) où S1 est déclaré sous la forme d'une liste de sommets et où U1 est une liste de listes à 2 éléments qui sont les arcs entre deux sommets.
- Un graphe pondéré G2 = (S2, U2) où U2 est donné sous la forme d'une liste de listes à 3 éléments $[i, j, \text{wij}]$: les deux premiers éléments représentent l'arc (i, j) et le troisième élément wij est le poids de cet arc. Tous les poids étant strictement positifs, on pourra choisir 0 dans la matrice d'adjacence pour indiquer qu'il n'y a pas d'arc entre i et j.

1. Dessiner les deux graphes G1 et G2.
2. Écrire une fonction `MatAdj(S, U)` qui prend `S` et `U` en paramètres et qui retourne la matrice d'adjacence du graphe (S, U).
3. Écrire de même une fonction `MatAdjPond(S, U)` qui fait la même chose dans le cas d'un graphe pondéré.
4. Écrire une liste `couleur` de n = Card(S) éléments et telle que `couleur[i]` soit la couleur du sommet `i`. Pour sa création, cette liste sera initialisée en donnant `"blanc"` comme valeur à tous ses éléments.

2 Algorithme de Dijkstra : plus court chemin entre deux sommets dans un graphe

L'algorithme de Dijkstra permet de trouver le plus court chemin entre un sommet origine `s0` et tout sommet accessible `s` dans un graphe pondéré où tous les poids sont **positifs**. Typiquement, le poids w_{ij} de l'arc (i, j) représente la distance entre i et j.

Cet algorithme fonctionne aussi bien pour les graphes non orientés que pour les graphes orientés.

Principe : sur feuille.

Résumé : l'algorithme utilise :

1. Une liste `distance` qui contient les distances entre les sommets `i` et l'origine `s0`. Cette liste est initialisée :

   $$\texttt{distance[i]} = \infty \text{ si } \texttt{i} \neq \texttt{s0} \text{ et } \texttt{distance[s0]} = 0$$

   La liste est mise à jour à chaque étape de l'algorithme.

2. Une liste de sommets `F` dont la distance peut encore varier, appelée file de priorité.
3. Une fonction `extraireMin(F)` qui élimine de `F` le sommet `i0` dont la distance à `s0` est la plus petite. Cette fonction renvoie `i0`.
4. Une fonction `recalculer(u, v, A)` qui recalcule la distance entre les sommets `u` et `v`. `A` est la matrice d'adjacence et `A[u,v]` est le poids de l'arc (arête) (u, v).

3 Parcours d'un graphe en profondeur

Partant d'un sommet origine $s_0 \in$ S, nous voulons visiter (explorer) tous les sommets accessibles à partir de s_0. Il y a deux façons de le faire : par un **parcours en largeur** ou par un **parcours en profondeur**. Nous n'examinerons ici que le parcours en profondeur.

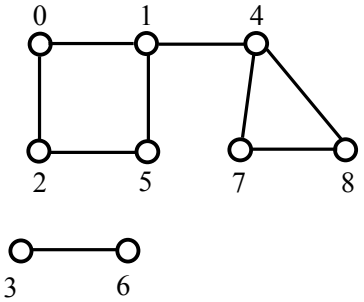
3.1 Parcours en profondeur

Dans le parcours en profondeur, on utilise une **fonction récursive** `visiter(u, S, A)` où `u` est un sommet du graphe, `S` l'ensemble des sommets et `A` la matrice d'adjacence associée.

Principe : sur feuille
Implémentation en Python : dans fichier .py

3.2 Questions

1. Dessiner l'arbre des appels récursifs obtenu en lançant la fonction `visiter(0, S, A)` pour le graphe représenté ci-dessous. Vous indiquerez pour chaque sommet l'ordre dans lequel il a été visité.



2. Même question si on part du sommet 4 en lançant `visiter(4, S, A)`.
3. On peut programmer l'ordre des visites des sommets grâce à une liste `ordre` qui est d'abord initialisée à [] (liste vide). À chaque fois qu'un sommet est visité, on l'insère "à droite" dans cette liste. Écrire la fonction `preTraitement(u)` pour qu'elle mette à jour la liste `ordre` (déclarée comme une variable globale) lors de la visite du sommet `u`.
4. Tester `visiter` sur le graphe précédent qui est G3 = (S3, U3) dans le fichier graphe.py
5. Comment écrire un programme qui permet de visiter tous les sommets du graphe ?

3.3 Composantes connexes d'un graphe non orienté

Dans le graphe G3 précédent, il y a deux parties disjointes formées des sommets {0, 1, 2, 4, 5, 7} et {3, 6} qu'on appelle composantes connexes du graphe. Par définition :

Déf : composante connexe d'un graphe non orienté

On appelle **composante connexe** d'un graphe G = (S, U) un sous-ensemble C $\subset$ S de sommets, tel que, $\forall (i, j) \in$ C, il existe un chemin entre i et j.

Le parcours en profondeur permet de trouver toutes les composantes connexes d'un graphe.

1. Écrire une liste `connexe` de n = Card(S) éléments qui sera initialisée avec des 0. Par définition : `connexe[i]` contiendra le numéro de la composante connexe associée au sommet `i`.
2. Écrire un programme utilisant la fonction `visiter` qui actualise la liste connexe (on pourra utiliser une variable `numConnexe`) au fur et à mesure de la découverte des sommets du graphe.
3. Appliquer ce programme au graphe G3.