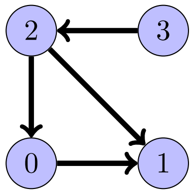
TP : Théorie des graphes : chemins optimaux

Ce TP introduit la notion de graphe représenté par matrice d'adjacence et présente quelques algorithmes de recherche de chemin optimaux.

1 Introduction

Un **graphe orienté** est un ensemble de n sommets reliés par un ensemble de m arcs. Chaque arc a un sommet d'origine et de destination. On le représente habituellement par une figure :



Ce graphe possède 4 sommets numérotés de 0 à 3 et 4 arcs. On le représentera dans ce TP par sa **matrice d'adjacence** c'est-à-dire une matrice carrée $G = (g_{i,j})_{i,j}$ d'ordre n telle que $g_{i,j}$ vaut `True` s'il existe un arc d'origine `i` et de destination `j` et `False` sinon. Dans l'exemple la matrice d'adjacence est

$$G = \begin{pmatrix} F & T & F & F \\ F & F & F & F \\ T & T & F & F \\ F & F & T & F \end{pmatrix}$$

Exercice 1
Pour tout sommet s d'un graphe on appelle **degré sortant** le nombre d'arcs ayant s comme origine et **degré entrant** le nombre d'arcs ayant s comme destination. Écrire des fonctions `degout` et `degin` prenant en argument un graphe et un de ses sommets et calculant respectivement le degré sortant et entrant du sommet s.

Dans un graphe orienté un **chemin** de longeur n est une liste de sommets `[x0, x1, ..., xn]` telle que pour tout $i \in [0, n-1]$ il existe un arc reliant x_i à x_{i+1}.

Exercice 2
Écrire une fonction `cheminValide` telle que `cheminValide(g, c)` retourne `True` si la liste c représente bien un chemin du graphe g et `False` sinon.

Dans un graphe orienté, on peut également ajouter un poids entier positif à chaque arc, on dit alors que le graphe est **pondéré**. On codera les poids sous forme d'une deuxième matrice, par exemple

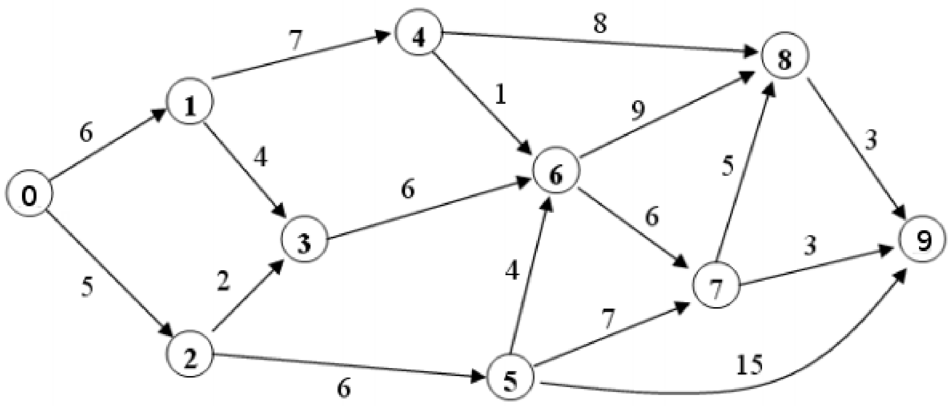
$$P = \begin{pmatrix} +\infty & 5 & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty \\ 1 & 4 & +\infty & +\infty \\ +\infty & +\infty & 3 & +\infty \end{pmatrix}$$

Une case de cette matrice pourra alors prendre la valeur $+\infty$ lorsqu'il n'y a pas d'arc. En `Numpy`, cette valeur peut-être obtenue avec `np.inf`.

Dans un graphe pondéré chaque chemin à un **coût** qui est la somme des poids des arcs le constituant.

Exercice 3
Écrire une fonction `coutChemin` telle que `coutChemin(g, p, c)` retourne le cout du chemin g dans le graphe pondéré (g, p). Que retourne la fonction si le chemin est invalide ?

Le but du TP est la recherche d'un **chemin de coût minimal**. Lorsqu'il n'y a pas de cycle de coût négatif, il existe un tel chemin optimal. Nous présentons 3 algorithmes. On testera nos algorithmes sur l'exemple suivant :



Exercice 4
Représenter le graphe ci-dessus en Python à l'aide de sa matrice d'adjacence et de poids.

2 Algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme glouton qui procède par parcours en largeur depuis un sommet d'origine s_0.

Il fonctionne avec trois tableaux de longueur le nombre de sommets n

- un tableau `optimal` tel que `optimal[i]` contient le coût d'un chemin optimal menant de s_0 à i,
- un tableau de booléens `avisiter` tel que `avisiter[i]` vaut `True` si le sommet n'a pas encore été traité

— un tableau `pred` tel que `pred[i]` est le sommet qui précède le sommet `i` dans le chemin optimal menant à i construit par l'algorithme depuis s_0.

Initialement, le tableau `avisiter` ne contient que des `True` et le tableau `pred` que des valeurs `-1`. Le tableau `optimal` ne contient que des valeurs $+\infty$ sauf pour la case du sommet d'origine s_0 qui peut-être atteinte sans coût donc `optimal[s0] = 0`.

L'algorithme procède itérativement ainsi :

— On cherche parmi les sommets restants à visiter celui qui possède pour l'instant un coût d'accès minimal et différent de $+\infty$. Notons s le sommet obtenu.

— On le considère alors comme visité et $optimal[s]$ contient le coût minimal c_s d'un chemin à s.

— Puis pour chaque arc (s, i) tel que i est à visiter, si $c_s + poids((s, i))$ est strictement inférieur (donc meilleur) que `optimal[i]` alors on a amélioré le coût d'accès au sommet i. On remplace `optimal[i]` par cette nouvelle valeur et on indique dans `pred[i]` qu'on a atteint le sommet `i` en arrivant par `s`.

— On recommence alors avec un nouveau sommet `s`.

Exercice 5
Écrire une fonction `indiceMin(t, b)` prenant un tableau de valeurs numériques `t` et un tableau de booléens `b` de même longueur et retourne un indice `i0` tel que `b[i0]` est `True`, `t[i0]` minimal dans `t` et `t[i0] < inf` Lorsqu'aucun `i0` ne vérifie cette condition alors on retourne `None`.

Exercice 6
Écrire une fonction `dijkstra(g, p, s0)` prenant en entrée un graphe pondéré (g, p) et un sommet départ s_0 et qui retourne le couple de tableaux `(optimal, pred)` défini ci-dessus.

Exercice 7
Écrire une fonction `construitChemin(debut, fin, pred)` reconstruisant le chemin optimal menant de `debut` à `fin` à l'aide du tableau des prédécesseurs `pred`.

Exercice 8
Déduire des questions précédentes un programme affichant pour le graphe exemple un chemin optimal menant de 0 à 9 ainsi que son coût.

3 Algorithme de Bellman-Ford

L'algorithme de Bellman-Ford est un algorithme utilisant la **programmation dynamique**. On considère un sommet de départ s_0 et on considère $t_{i,p}$ le coût d'un chemin optimal menant de s_0 à i et de longueur au plus p.

Exercice 9

1. Que vaut $t_{i,0}$?
2. Pour $p > 0$, justifier la relation

$$t_{i,p} = \min\left\{t_{i,(p-1)}, \min_{0 \le k < n}\left\{t_{k,(p-1)} + poids(k, i)\right\}\right\}$$

3. En déduire un programme calculant la matrice $(t_{i,j})_{1\leq i,j<n}$
4. Écrire une fonction `bellmanFord(g, p, s0)` retournant un tableau `optimal` tel que `optimal[i]` contient le coût d'un chemin optimal menant de s_0 à i.
5. Écrire une seconde version `bellmanFord2(g, p, s0)` retournant comme pour Dijkstra un couple `(optimal, pred)` permettant la reconstruction du chemin optimal. Il faudra alors au cours de l'algorithme mettre à jour `pred` dès qu'un nouveau chemin a été trouvé.
6. Vérifier sur l'exemple qu'on obtient le même résultat qu'avec Dijkstra.

4 Algorithme de Floyd-Warshall

L'algorithme de Floyd-Warshall est également un algorithme de programmation dynamique. Cette fois-ci on ne considère plus un sommet de départ particulier : les sommets sont traités dans leur ensemble. On notera : $w_{i,j,p}$ le coût d'un chemin optimal menant du sommet i au sommet j et n'utilisant comme sommets intermédiaires que des sommets de l'ensemble $\{0, 1, \ldots, p-1\}$.

Exercice 10 1. Soit i et j deux sommets, que vaut $w_{i,j,0}$?

2. Pour $p > 0$, justifier la relation

$$w_{i,j,p} = \min\left\{w_{i,j,(p-1)}, \quad w_{i,(p-1),(p-1)} + w_{(p-1),j,(p-1)}\right\}$$

3. En déduire une fonction `warshall(g, p)` retournant une matrice $A = (a_{i,j})_{1\leq i,j<n}$ telle que $a_{i,j}$ contienne le coût d'un chemin optimal menant de i à j. On ne cherchera pas à reconstruire les chemins optimaux.