Chapitre n°6

Graphes pondérés

Exercice n°6.1 ★

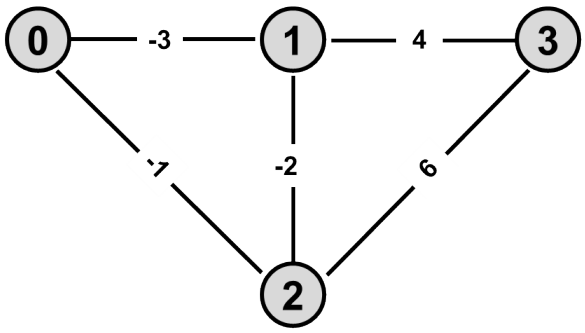
Implémentation de graphes pondérés sous la forme de matrices

Q1 Représenter le graphe G défini par la matrice `G` suivante :

Code Python

```
>>> I=float("inf") # I contient le nombre "infini"
>>> G = [[I, 4, I, 5], [I, I, I, -3], [-1, I, I, 8], [2, I, I, I]]
```

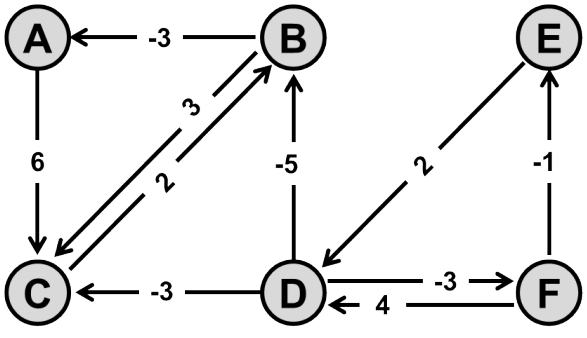
Q2 Écrire une instruction en Python pour représenter la matrice d'adjacence `G` du graphe ci-dessous :



Exercice n°6.2 ★

Poids et distance usuels dans un graphe pondéré

On considère le graphe orienté pondéré suivant :



Q1 Proposer une représentation matricielle de ce graphe.

On considère la définition usuelle pour le poids associé à un chemin, c'est-à-dire que le poids d'un chemin $\mathcal{C}$ est égal à la somme des poids des arcs qui composent ce chemin.

Q2 Quel est le poids du chemin $\mathcal{C}_{F\longrightarrow A} = (F \longrightarrow E \longrightarrow D \longrightarrow C \longrightarrow B \longrightarrow A)$?

Q3 Proposer un chemin depuis le sommet F vers le sommet A de poids $+1$.

On considère la définition usuelle pour la distance entre deux sommets dans un graphe pondéré, c'est-à-dire que la distance du sommet B à l'origine A est égale au poids du chemin $\mathcal{C}_{A\longrightarrow B}$ allant du sommet d'origine A vers le sommet d'arrivée B de poids minimal.

Q4 Quelle est la distance $\delta(C, A)$ du sommet A depuis l'origine C ?

Q5 Quelle est la distance $\delta(A, D)$ du sommet D depuis l'origine A ?

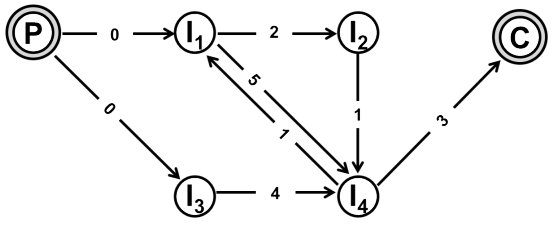
Q6 Quelle est la distance $\delta(F, A)$ du sommet A depuis l'origine F ?

Exercice n°6.3 ★★★

Directement du producteur au consommateur !

On considère une chaîne d'approvisionnement depuis un (unique) producteur vers un (unique) consommateur. L'ensemble de la chaîne d'approvisionnement est représentée par un graphe orienté pondéré $G(S, A, p)$; le poids $p(s_i, s_j) > 0$ associé à chaque arc $(s_i, s_j) \in A$ correspond au surcoût ajouté par l'intermédiaire $s_i \in S$ au moment de la vente – par exemple, un arc (s_i, s_j) de poids $p(s_i, s_j) = 10$ indique que l'intermédiaire s_i a revendu le bien 10 unités plus cher qu'il ne l'a acheté. On suppose qu'aucun intermédiaire n'est philanthrope : personne ne revend le bien pour une somme inférieure à celle qu'il a dépensée pour réaliser l'achat du bien. Le producteur est identifié par le sommet s_p tandis que le consommateur est identifié par le sommet s_c. Par convention, tout arc de la forme (s_p, s_j) possède un poids nul.

On cherche à trouver un chemin $\mathcal{C}_{s_p\longrightarrow s_c} = (s_p \longrightarrow ... \longrightarrow s_c)$ permettant de minimiser les coûts liés aux différents intermédiaires. À titre d'exemple, on peut considérer le graphe $G_{exemple}$ suivant :



Q1 Justifier que le graphe G soit forcément un graphe simple.

Q2 Quelles définitions peut-on choisir pour :

- le poids $\mathcal{P}(\mathcal{C}_{s_p\longrightarrow s_c})$ d'un chemin représentant une vente conclue depuis le producteur jusqu'au consommateur ?
- la distance $\delta(s_p, s_c)$ entre producteur et le consommateur.

afin de répondre au problème posé ? Déterminer, à la main, la distance $\delta(s_p, s_c)$ dans le graphe $G_{exemple}$ en précisant un plus court chemin associé à cette distance.

Q3 Justifier qu'on puisse utiliser l'algorithme de DIJKSTRA pour résoudre ce problème.

On suppose que le graphe G est implémenté sous la forme d'une matrice d'adjacence `G`. Par convention, on choisit le sommet s_p comme étant le sommet n° 0 (première ligne / colonne de la matrice d'adjacence) et le sommet s_c comme étant le sommet n° $Card(S)-1$ (dernière ligne / colonne de la matrice d'adjacence).

Q4 Proposer une suite d'instructions permettant d'implémenter le graphe $G_{exemple}$ en respectant les conventions énoncées ci-dessus.

Dans le script fourni en ligne :

`http://lejeune.enseignement.free.fr/PCSI-MPSI_Informatique/files/chap6/PCSI-MPSI_Informatique_chap6_exo8_script.py`

on propose une fonction `dijkstra` à compléter, prenant comme arguments une liste de listes d'entiers positifs `G` (représentant la matrice d'adjacence associée au graphe G et respectant les conventions énoncées ci-dessus), et retournant un tuple constitué d'un entier `delta` représentant la distance $\delta(s_p, s_c)$ entre producteur et le consommateur ainsi que d'une liste d'entiers `C` représentant (dans l'ordre) les numéros des sommets constituant le plus court chemin depuis s_p vers s_c (cette liste commence donc nécessairement par 0 et se termine donc nécessairement par $Card(S)-1$). La liste `L` permet de stocker le coût $\lambda_{s_p}(s_i)$ associé à chaque sommet s_i, constituant ainsi une borne supérieure de la distance $\delta(s_p, s_i)$ obtenue par relâchements successifs des arcs lors du parcours du graphe. La liste `Pi` constitue la liste des prédécesseurs associée aux sommets du graphe.

Q5 Compléter la définition de la fonction `dijkstra`.

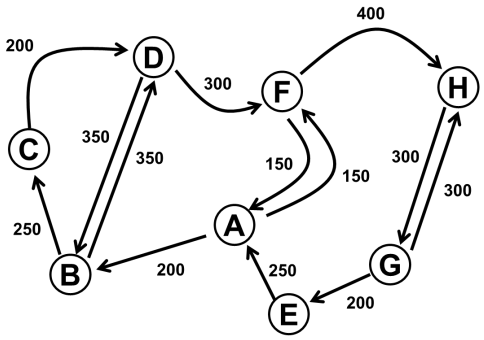
Q6 Proposer une suite d'instructions permettant de retrouver la distance $\delta_(s_p, s_c)$ associée au graphe $G_{exemple}$.

Exercice n°6.4 ★★

CIRCULATION EN CENTRE-VILLE

La ville de Trifouillis-lès-Oies est une charmante bourgade du Nulparistan. Elle est réputée pour son centre-ville médiéval caractérisé par ses ruelles étroites et pavées. On représente les ruelles du centre-ville de Trifouillis-lès-Oies par un graphe orienté $G(S, A, p)$ dont les sommets représentent les intersections et les arcs représentent les ruelles (en raison de l'étroitesse des ruelles, la plupart sont à sens unique, d'où le choix d'un graphe orienté). À chaque arc du graphe G, on associe un poids p correspondant à la largeur maximale (exprimée en cm) d'un véhicule pouvant circuler dans la ruelle.

À titre d'exemple, voici une carte du centre-ville de la ville voisine de Trucmuche-sur-Bidule, représentée sous la forme du graphe $G_{trucmuche}$ respectant les conventions énoncées précédemment :



On définit le poids $\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j})$ associé au chemin $\mathcal{C}_{s_i \longrightarrow s_j} = (u_0 = s_i \longrightarrow u_1 \longrightarrow ... \longrightarrow u_\ell = s_j)$ comme étant la largeur maximale d'un véhicule pouvant emprunter ce chemin. La distance $\delta(s_i, s_j)$ est ainsi définie comme étant la largeur maximale d'un véhicule pouvant aller du sommet s_i au sommet s_j.

Q1 Dans la matrice d'adjacence $\mathcal{G}$ associée au graphe G, quel coefficient doit-on attribuer au coefficient $\mathcal{G}_{i,j}$ s'il n'existe pas de ruelle allant du sommet s_i au sommet s_j ?

Q2 Donner l'expression analytique du poids $\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j})$ associé au chemin $\mathcal{C}_{s_i \longrightarrow s_j} = (u_0 = s_i \longrightarrow u_1 \longrightarrow ... \longrightarrow u_\ell = s_j)$ de longueur ℓ en fonction des poids des arcs constituant ce chemin.

Q3 Déterminer le poids du chemin $(B \longrightarrow C \longrightarrow D \longrightarrow F)$ du graphe $G_{trucmuche}$.

Q4 Donner l'expression analytique de la distance $\delta(s_i, s_j)$ entre les sommets s_i et s_j du graphe en fonction des poids $\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j})$ des chemins de s_i vers s_j.

Q5 Déterminer les distances $\delta(B, F)$ et $\delta(F, B)$ du graphe $G_{trucmuche}$.

Afin de déterminer la distance entre deux sommets s_i et s_j du graphe G, on procède par force brute : dans un premier temps, on génère tous les chemins élémentaires issus du sommet s_i ; on détermine ensuite le poids de chacun de ces chemins arrivant au sommet s_j ; puis enfin, on déduit la distance entre les sommets s_i et s_j.

Q6 Pourquoi se restreint-on à l'étude des chemins élémentaires ?

Q7 Écrire une fonction `poids_chemin` prenant comme arguments une liste de listes d'entiers `G` (représentant la matrice d'adjacence du graphe pondéré G) ainsi qu'une liste d'entiers `C` (représentant les index successifs des sommets constituant un chemin $\mathcal{C}$), et retournant le poids $\mathcal{P}(\mathcal{C})$ associé à ce chemin.

Q8 Écrire une fonction `generer_chemins` prenant comme arguments une liste de listes d'entiers `G` (représentant la matrice d'adjacence du graphe pondéré G) ainsi qu'un entier `i` (représentant l'index d'un sommet s_i du graphe G), et retournant la liste de tous les chemins élémentaires d'origine s_i. Chaque chemin de la liste générée sera donné sous la forme d'une liste d'entiers représentant les index successifs des sommets constituant ce chemin.

Q9 Écrire une fonction `distance` prenant comme arguments une liste de listes d'entiers `G` (représentant la matrice d'adjacence du graphe pondéré G) ainsi que deux entiers `i` et `j` (représentant les index de deux sommets s_i et s_j du graphe G), et retournant la distance $\delta(s_i, s_j)$ entre le sommet s_i et le sommet s_j. Un appel à la fonction `distance` retourne la valeur 0 s'il n'existe aucun chemin allant de s_i à s_j. On utilisera les fonctions `poids_chemin` et `generer_chemins`.

Q10 Quelle est la complexité associée à cette approche par force brute ? Quelle autre approche, moins gourmande, aurait-on pu envisager (plutôt que la force brute) ?