U.P.S. I.U.T. A, Département d'Informatique Année 2023-2024

S.A.E. 2.02 : Exploration algorithmique d'un problème.

Responsable : Lahcen GHANNAM
lahcen.ghannam@iut-tlse3.fr, Bat. Informatique, bureau 122 1er étage

• But du projet :
I- Implémenter et tester deux algorithmes de plus courts chemins sur un graphe pondéré et comparer leurs résultats (selon les cas)et leurs complexités.
II- Déterminer un seuil de forte connexité pour un graphe.

• Retour attendu de chaque groupe (date limite : 04.06.24, 24h)
Un dossier, à déposer sur Moodle, contenant :
- un rapport (format pdf) répondant clairement aux questions posées, en respectant le plan donné dans ce document ;
- les fichiers éxécutables au format .py des codes Python.

A l'exception du §2, ces codes ne devront être récupérés sur des codes existants sur internet, ou générés par une I.A. de type ChatGPT ou autre.

Ces codes devront être abondamment commentés. Ce point comptera fortement dans l'évaluation et permettra de vérifier le point précédent. On utilisera $\sharp\cdots$ pour les commentaires courts, et $''''\cdots''''$ pour les commentaires de plusieurs lignes dans le fichier Python.

• Evaluation :

- une note sur le dossier rendu (coeff : 70%). Celle-ci reposera :
- pour moitié sur une évaluation du dossier (commune aux étudiants du groupe) sur laquelle on évaluera la qualité des résultats onbtenus, la qualité de la rédaction, la qualité des codes fournis ;
- pour moitié sur une évaluation individuelle : on vérifiera que chaque étudiant maitrise et comprend le contenu de son dossier. On pourra demander d'exécuter les codes fournis dans le dossier sur un exemple proposé et de commenter les résultats obtenus.

- une note sur la connaissance théorique du sujet de la SAE (coeff : 30%).

Les deux tests individuels (vérification de la compréhension du dossier fourni et vérification des connaissances théoriques) seront regroupés sur une séance et effectués sur Moodle.

En résumé :
- dossier (note commune) : 7 points.
- dossier (note individuelle) : 7 points (test Moodle)
- connaissance du sujet (note individuelle) : 6 points (test Moodle)

- **Calendrier :**

- lundi 29.04.24 (en TD de 1h30 l'après-midi) : présentation du sujet, constitution des groupes (3 ou 4 étudiants par groupe), programme de travail.

- lundi 06.05.24 (8h-12h30) : travail en autonomie. Une permanence sera assurée.
- lundi 06.05.24 (en TD de 1h30 l'après-midi) : Poursuite du travail + vérification par l'enseignant de l'avancement de chaque groupe.

- lundi 13.05.24 (8h-12h30) : travail en autonomie. Une permanence sera assurée.
- lundi 13.05.24 (en TD de 1h30 l'après-midi) : Poursuite du travail + vérification par l'enseignant de l'avancement de chaque groupe.

- Jeudi 23.05.24 (en TD de 1h30 l'après-midi) : Poursuite du travail + vérification par l'enseignant de l'avancement de chaque groupe.

- lundi 27.05.24 (8h-12h30) : travail en autonomie. Une permanence sera assurée.
- lundi 27.05.24 (en TD de 1h30 l'après-midi) : Poursuite du travail + vérification par l'enseignant de l'avancement de chaque groupe.

- Jeudi 30.05.24 (14h-18h30) : travail en autonomie. Une permanence sera assurée.

- lundi 03.06.24 (en TD de 1h30 l'après-midi) : Poursuite du travail + vérification par l'enseignant de l'avancement de chaque groupe.

- Mercredi 4 juin (24h) : date limite de remise du dossier sur Moodle.
- Lundi 10 juin (14h) : test Moodle sur la connaissance du dossier et sur les connaisssances théoriques acquises.

Le dossier remis devra respecter le plan ci-dessous :

Table des matières

Première partie

Comparaison d'algorithmes de plus courts chemins.

1 Connaissance des algorithmes de plus courts chemins.

1.1 Présentation de l'algorithme de Djikstra

Afin de se familiariser avec l'algorithme de Djikstra, on demande ici de l'expliquer en détail sur un exemple de graphe à poids positifs, choisi par vous. L'exemple devra être suffisamment petit pour pouvoir être traité à la main, et soigneusement rédigé pour suggérer le code à effectuer.

On choisira la matrice M et le sommet de départ d de sorte que, selon le sommet final s utilisé on obtienne les deux types de réponses : "existence et affichage d'un plus court chemin", ou "pas de chemin de d vers s".

1.2 Présentation de l'algorithme de Bellman-Ford

Idem. On choisira ici un exemple à poids de signes variables, de sorte que, selon le chois du sommet final s l'algorithme retourne les trois possibilités : affichage du plus court chemin de d vers s, ou la mentions "pas de chemin de d vers s" ou la mention "existence de chemins de d vers s mais pas de plus court chemin (présence d'un cycle de poids négatif)".

2 Dessin d'un graphe et d'un chemin à partir de sa matrice.

2.1 Dessin d'un graphe

A partir d'une recherche sur internet, trouver un outil, de préférence en Python qui, à partir de la matrice d'un graphe pondéré donne un dessin de ce graphe.
Présenter des exemples.

2.2 Dessin d'un chemin

Améliorer cet outil afin que étant donné un chemin donné par sa suite de sommets visités, celui-ci s'affiche en rouge sur le graphe.
Présenter des exemples.

3 Génération aléatoire de matrices de graphes pondérés

3.1 Graphes avec 50% de flèches

Construire une fonction Python `graphe(n,a,b)` qui prend en entrée un entier strictement positif n, deux entiers a et b, et qui génère aléatoirement une matrice de taille $n \times n$ présentant environ 50% de coefficients ∞ et 50% de coefficients avec des poids entiers dans l'intervalle $[a, b[$.
Présenter des exemples et leurs représentations graphiques.

Suggestion. On pourra s'inspirer de l'exercice 12 du cours, pour obtenir d'abord une

matrice avec approximativement 50% de 0 et 50% de 1, puis en remplaçant les 1 par des valeurs choisies aléatoirement et uniformément entre a et b, et les coefficients 0 par des $+\infty$ (en Python, `float('inf')` crée une valeur plus grande que tout réel).
Attention, pour cette dernière opération, il est nécessaire de convertir d'abord la matrice en coefficients de type float par : `M=M.astype('float64')`

3.2 Graphes avec une proportion variables p de flèches

On souhaite pouvoir faire varier la proportion p de flèches (c'est-à-dire de 1 dans la matrice). Modifier la fonction précédente en une fonction `graphe2(n,p,a,b)` qui génère aléatoirement la matrice d'un graphe à n sommets avec une proportion p (pouvant varier de 0 à 1) de flèches (coefficients différennts de l'infini). Présenter des exemples.

Suggestion. Utiliser la fonction `binomial` à la place de `randint`.

4 Codage des algorithmes de plus court chemin

4.1 Codage de l'algorithme de Dijkstra

Créer une fonction Python `Dijkstra(M,d)` qui prend en entrée la matrice d'un graphe pondéré à poids positifs, un sommet d de ce graphe donné par son indice dans la liste des sommets, et qui, en exécutant l'algorithme de Dijkstra, retourne pour chacun des autres sommets s :
- soit la longueur et l'itinéraire du plus court chemin de d à s ;
- soit la mention "sommet non joignable à d par un chemin dans le graphe G".
On codera à partir du pseudocode présenté dans le cours.

On pourra retourner le résultat sous forme de la liste des sommets du chemin obtenu, mais aussi en l'affichant graphiquement à l'aide de l'outil de §2.

Tester sur vos exemples du §1, et sur des exemples générés aléatoirement.

Suggestion. On pourra créer deux dictionnaires `dist` et `pred` dont les clés sont les sommets `s` et les valeurs `dist(s)`, ainsi qu'une version restreinte `distR` des sommets restant à traiter, que l'on mettra à jour en supprimant les sommets déjà traités. On se documentera pour ajouter, supprimer une entrée d'un dictionnaire, trouver une valeur à partir d'une clé, trouver les clés correspondant à une valeur.

4.2 Codage de l'algorithme de Belman-Ford

Créer une fonction Python `Bellman-Ford(M,d)` qui prend en entrée la matrice d'un graphe pondéré à poids de signe quelconque, un sommet d de ce graphe donné par son indice dans la liste des sommets, et qui, en exécutant l'algorithme de Bellman-Ford, retourne pour chacun des autres sommets s :
- soit la longueur et l'itinéraire du plus court chemin de d à s ;
- soit la mention "sommet non joignable depuis d par un chemin dans le graphe G".
- soit la mention "sommet joignable depuis d par un chemin dans le graphe G, mais pas de plus court chemin (présence d'un cycle négatif)".
On codera à partir du pseudocode présenté dans le cours.

On pourra retourner le résultat sous forme de la liste des sommets du chemin obtenu, mais aussi en l'affichant graphiquement à l'aide de l'outil de §2.

Tester sur vos exemples du §1, et sur des exemples générés aléatoirement avec des poids de signes variables.

5 Influence du choix de la liste ordonnée des flèches pour l'algorithme de Bellman-Ford

On peut choisir cette liste ordonnée des flèches de plusieurs manières :
- de manière arbitraire, par choix aléatoire ;
- par un ordre choisi "en largeur" : un parcours en largeur donne une liste ordonnée de sommets depuis le départ d. On prend *toutes les flèches* issues de ces sommets.
- par un ordre choisi "en profondeur" : un parcours en profondeur donne une liste ordonnée de sommets depuis d. On prend *toutes les flèches* issues de ces sommets.

On voudrait vérifier ici si cela influence fortement ou non le temps de calcul de l'algorithme de Bellman-Ford.
- Modifier le code de Bellman-Ford en 3 variantes selon le mode de construction de la liste des flèches.
- Rajouter dans le code un compteur afin d'afficher le nombre de tours effectués.
- Comparer les résultats sur un même graphe "grand" (par exemple 50 sommets) généré aléatoirement, et conclure.

6 Comparaison expérimentale des complexités

On veut connaitre expérimentalement la croissance du temps de calcul de chacun des deux algorithmes en fonction du nombre n de sommets. On reprend la même démarche que pour la comparaison des deux algorithmes de calcul de fermeture transitive, étudiée au R207 (exercices 12 et 18) :

6.1 Deux fonctions "temps de calcul"

Créer une fonction Python $TempsDij(n)$ qui prend en entrée un entier positif n et qui :
- génère aléatoirement une matrice d'un graphe pondéré à poids positifs de taille n ;
- calcule tous les plus courts chemins depuis le premier sommet vers tous les autres sommets par l'algorithme de Dijkstra (sans affichage du résultat) ;
- retourne le temps de calcul utilisé.

Créer une fonction Python similaire $TempsBF(n)$ qui utilise l'algorithme de Bellman-Ford, avec le choix de la liste de flèches le plus efficace, déterminé à la question précédente.

6.2 Comparaison et identification des deux fonctions temps

- Afficher sur un même graphique les représentations de ces deux fonctions $TempsDij(n)$ et $TempsBF(n)$ pour $n = 2, ...200$.
  Quel algorithme est le plus rapide ?
- Expliquer pourquoi si une fonction $t(n)$ est approximativement de type cn^a pour n grand (croissance polynomiale d'ordre a) alors sa représentation graphique en coordonnées log-log est approximativement une droite de pente a.

— Vérifier que les deux fonctions temps sont polynomiales et déterminer pour chacune d'elle l'exposant a.
— Faire deux tests de ces calculs, l'un avec une proportion p de flèches fixé, et l'autre avec une proportion p de flèches diminuant avec n, par exemple : $p = 1/n$.

6.3 Conclusion

— Résumer les résultats de comparaison obtenus.
— Expliquer, selon le type de graphe considéré, le choix de l'algorithme que l'on doit effectuer.

Deuxième partie

Seuil de forte connexité d'un graphe orienté.

7 Test de forte connexité

— En rappelant les définitions, expliquer pourquoi un graphe est fortement connexe si et seulement si la matrice de sa fermeture transitive n'a que des 1.

— Ecrire une fonction Python `fc(M)` qui prend en entrée la matrice d'un graphe orienté (non pondéré) et qui retourne `True` ou `False` à la question : "*le graphe G donné par la matrice M (après numérotation de ses sommets) est-il fortement connexe ?*" On pourra utiliser une des fonctions de fermeture transitive déjà créees en TD.

8 Forte connexité pour un graphe avec p=50% de flèches

On veut vérifier l'affirmation : "*Lorqu'on teste cette fonction `fc(M)` sur des matrices de taille n avec n grand, avec une proportion $p = 50\%$ de 1 (et 50% de 0), on obtient presque toujours un graphe fortement connexe.*"
On donnera un sens statistique à "presque toujours" : par exemple, "presque toujours" = "dans plus de 99% des cas testés". On demande donc ici une étude statistique en testant l'affirmation sur un grand nombre de cas (plusieurs centaines).

— Pour cela, on créera une fonction `test stat fc(n)` retournant le pourcentage de graphes de taille n fortement connexes pour un test portant sur quelques centaines de graphes.

— A partir de quel n l'affirmation ci-dessus est-elle vraie ?

9 Détermination du seuil de forte connexité

Pour n fixé, on va donc faire varier cette proportion p de flèches (ou de 1 dans la matrice) en la diminuant jusqu'à obtenir un seuil `seuil(n)` (il dépend de la taille n de la matrice) tel que pour $p \geq$`seuil(n)` le graphe est presque toujours fortement connexe et en dessous duquel il ne l'est plus.

— Améliorer la fonction `test stat fc(n)` en une fonction `test stat fc2(n,p)` où le test porte maintenant sur des matrices de taille n avec une proportion p de 1.

— Ecrire une fonction `seuil(n)` qui détermine ce seuil de forte connexité. (Pour une taille n donnée, on descendra p jusqu'à déterminer ce seuil).

10 Etude et identification de la fonction seuil

10.1 Représentation graphique de `seuil(n)`

Représenter cette suite `seuil(n)` sur l'intervalle $[10, 40]$ (ou plus si la puissance de calcul le permet).
Est-elle décroissante (comme on s'y attend) ?

10.2 Identification de la fonction `seuil(n)`

— La fonction `seuil(n)` est-elle asymptotiquement une fonction puissance ?
— Si oui, identifier cette fonction puissance cx^a (on pourra déterminer a et c à partir de l'équation de la droite de régression de la partie du nuage de points qui semble approximativement alignés sur le graphe log-log).
