Concours blanc

INFORMATIQUE COMMUNE - Devoir n° 5 - Olivier Reynet

Consignes :

1. utiliser une copie différente pour chaque partie du sujet,
2. écrire son nom sur chaque copie,
3. écrire de manière lisible et intelligible,
4. préparer une réponse au brouillon avant de la reporter sur la feuille.

Les trois parties de ce contrôle sont indépendantes.

A Multiplier des matrices

On dispose de plusieurs fichiers de type csv qui contiennent des matrices de nombres flottants. Voici un exemple d'un tel fichier :

```
97.0,35.0,74.0,61.0
61.0,46.0,24.0,42.0
87.0,63.0,94.0,92.0
```

qui représente la matrice

$$P = \begin{bmatrix} 97.0 & 35.0 & 74.0 & 61.0 \\ 61.0 & 46.0 & 24.0 & 42.0 \\ 87.0 & 63.0 & 94.0 & 92.0 \end{bmatrix}$$

Les objectifs de cette section sont :

1. d'importer les matrices à partir des fichiers dans une structure de type liste de liste,
2. de multiplier deux matrices,
3. d'enregistrer le résultat dans un fichier.

A1. Écrire une fonction de signature `import_mat(filename: str)—> list[list[float]]` où `filename` est un nom du fichier contenant une matrice. Cette fonction renvoie la matrice décrite par le fichier sous la forme d'une liste de liste de flottants. Par exemple, pour le fichier représentant la mlatrice P on obtient `[[97.0, 35.0, 74.0, 61.0], [61.0, 46.0, 24.0, 42.0], [87.0, 63.0, 94.0, 92.0]]`. Utiliser les fonctions de l'annexe.

A2. Écrire une fonction de signature `zeros_matrice(n,m)—> list[list[float]]` qui renvoie une matrice de taille $n \times m$ ne contenant que des zéros.

A3. Écrire une fonction de signature `dimensions(M: list[list[float]])—> (int,int)` dont le paramètre est une matrice et qui renvoie sa dimension sous la forme d'un tuple d'entiers (nombre de lignes, nombre de colonnes).

A4. On pose $P = AB$. Donner l'expression de l'élément p_{ij} de P en fonction de ceux de A de dimension (n,m) et B de dimension (m,q). Bien remarquer que les indices commencent à 0 en informatique!

A5. Écrire une fonction de signature `produit(A,B)` qui renvoie le produit des matrices `A` et `B`. On garantira par une assertion la faisabilité du calcul.

A6. Quelle est la complexité temporelle de la fonction `produit` en fonction de n, m et q?

A7. Écrire une fonction de signature `write_matrix(A : list[list[]float], filename: str)` qui écrit la matrice `A` dans le fichier désigné par `filename`. Comme dans l'exemple initial, les nombres sont séparés par des virgules. Une nouvelle ligne est utilisée pour chaque ligne de la matrice. Utiliser les fonctions en annexe.

A8. En examinant les résultats des multiplications des matrices de flottants, l'ingénieur responsable des calculs met en évidence que certains résultats n'ont pas de sens. En y regardant de plus près, il observe par exemple que $16777220.0 + 2.0 \times 0.2 + 3.0 \times 0.33$ résulte en 16777220.0. Les nombres flottants sont stockés en simple précision, c'est-à-dire que la pseudo-mantisse est codée sur 23 bits. Quel est le mécanisme des nombres flottants qui permet d'expliquer ce résultat?

B Coloration de graphe

■ **Définition 1 — Coloration valide.** Une coloration d'un graphe est valide lorsque deux sommets adjacents n'ont jamais la même couleur.

■ **Définition 2 — Nombre chromatique.** Le nombre chromatique d'un graphe G est le plus petit nombre de couleurs nécessaires pour obtenir une coloration valide de ce graphe. On le note généralement $\chi(G)$.

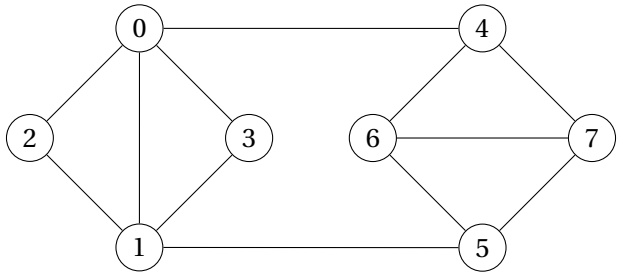

FIGURE 1 – Graphe `g`

On se donne un graphe `g` (cf. figure 1) ainsi qu'une liste de couleur utilisables pour colorer une graphe.

```
1     colors = ["pink", "turquoise", "yellow", "blue","magenta", "orange", "lime"]
```

B9. En utilisant les listes Python, représenter par une liste d'adjacence le graphe `g` de la figure 1.

B10. Quel est le nombre chromatique du graphe de la figure 1? Trouver ce nombre par le dessin.

L'algorithme DSATUR (cf. algorithme 1) est un algorithme de coloration de graphe relativement efficace. Les questions qui suivent visent à l'implémenter en Python.

Algorithme 1 DSATUR

```
1 : Fonction DSATUR(g, couleurs)
2 :     n ← le nombre de sommets de g
3 :     sdmax ← le sommet de degré maximum
4 :     cmap ← un dictionnaire vide dont les clefs sont les sommets du graphe et la valeur une couleur.
5 :     cmap[sdmax] ← couleurs[0]                          ▷ La première couleur
6 :     pour i de 1 à n-1 répéter                          ▷ Pour chaque sommet non coloré
7 :         s ← le sommet dont le nombre de couleurs différentes parmi les voisins est maximal
8 :         cmap[s]← la première couleur possible parmi les couleurs
9 :     renvoyer cmap
```

B11. Écrire une fonction de prototype `deg(g)` dont le paramètre est un graphe sous la forme d'une liste d'adjacence. Cette fonction renvoie la liste des tuples des degrés des sommets. Par exemple, `[[1], [0, 2], [1, 3], [2]]` renvoie `[1, 2, 2, 1]`.

B12. Écrire une fonction de prototype `sdmax(g)` dont le paramètre d'entrée est un graphe sous la forme d'une liste d'adjacence. Cette fonction renvoie le sommet dont le degré est maximal dans le graphe.

B13. (bonus points) Écrire une fonction de signature `saturation(g, cmap)` dont les paramètres sont un graphe `g` et un dictionnaire `cmap` qui associe un sommet à une couleur. Cette fonction renvoie le degré de saturation d'un sommet, c'est-à-dire le nombre de couleurs différentes parmi les voisins de chaque sommet. Par exemple, pour le graphe `[[1], [0, 2], [1, 3], [2]]` et le dictionnaire `{0 : "deeppink", 2 :"yellow"}`, la fonction renvoie `[0, 2, 0, 1]`.

B14. Écrire une fonction de signature `smaxsat(g, cmap)` dont les paramètres sont un graphe `g` et un dictionnaire `cmap` qui associe un sommet à une couleur. Cette fonction renvoie le sommet dont le degré de saturation est maximum. En cas d'égalité, on choisit le sommet dont le degré est le plus élevé.

B15. (bonus points) Écrire une fonction de signature `p_couleur_possible(used_colors, colors)` qui renvoie la première couleur possible de la liste `colors` pour un sommet en tenant compte des couleurs utilisées par ses voisins (`used_colors`).

B16. Écrire une fonction de prototype `dsatur(g, colors)` qui implémente l'algorithme 1. Cette fonction renvoie un dictionnaire `cmap` : chaque clef est le numéro d'un sommet et la valeur associée à cette clef est la couleur affectée au sommet. S'appuyer sur les fonctions précédentes.

B17. Appliquer à la main l'algoritme 1 sur le graphe `g` de la figure 1. Que pouvez-vous en conclure ?

B18. Comment pourrait-on qualifer cet algorithme ?

C Voyages d'un représentant de commerce

Un représentant de commerce doit vendre ses produits dans n villes. Il connaît la distance entre chaque ville et peut donc modéliser la carte de sa zone commerciale par un graphe `g` pondéré positivement en km.

```
g = [[(1, 20), (2, 50), (4, 70), (5, 90)],
     [(0, 20), (2, 20)],
     [(0, 50), (1, 20), (3, 10)],
     [(2, 10), (4, 140), (5, 10)],
     [(0, 70), (3, 140)],
     [(0, 90), (3, 10)]]
```

Cette section à pour but :

1. de calculer les plus courts chemins depuis une ville de départ vers toutes les autres
2. de trouver une solution (pas nécessairement optimale) au problème du voyageur de commerce.

On souhaite connaître les chemins les plus courts à partir du sommet 0 en direction de tous les autres sommets du graphe.

C19. Appliquer à la main l'algorithme de Dijkstra au graphe pondéré `g` en partant du sommet 0. On prendra soin de représenter le déroulement dans un tableau dont les colonnes sont les sommets du graphe et l'ensemble des sommets visités et les lignes les distances aux sommets à chaque itération.

Le problème du voyageur de commerce (ou TSP pour Traveling Salesman Problem) concerne un représentant de commerce qui doit écouler ses produits dans n villes. Il planifie donc sa tournée de manière à passer **une seule fois** dans une même ville et **à minimiser** le nombre de kilomètres parcourus.

En termes savants, il s'agit de trouver un cycle hamiltonien de coût minimal dans un graphe pondéré.

C20. La distance maximale entre deux villes est un entier naturel inférieur à 1976 km. Combien de bits au minimum sont-ils nécessaires pour coder un nombre représentant une distance en km en mémoire ?

C21. Le représentant doit visiter 90 villes. Chaque ville est représentée par une chaîne de caractères codée sur 12 octets. Quelle est la taille de l'espace mémoire nécessaire pour pouvoir stocker toutes les villes ? Donner la réponse en Ko (Kilooctets).

On suppose maintenant que le graphe pondéré des villes est complet, c'est-à-dire qu'il existe une liaison (aérienne par exemple) entre chaque ville. L'algorithme 2 permet de trouver une solution au problème, pas nécessairement optimale.

Algorithme 2 TSP glouton

```
1 : Fonction GTSP(g)
2 :     n ← l'ordre du graphe g
3 :     C ← une liste vide                                  ▷ Le circuit
4 :     A ← la liste des arêtes triées par poids croissant
5 :     tant que la longueur du circuit est strictement inférieure à n répéter
6 :         Ajouter à C l'arête de poids la plus faible de A si :
```

— celle-ci ne crée pas un cycle de longueur strictement inférieure à n,

— celle-ci ne fait pas croître le degré des sommets solutions au-delà de 2.

```
7 :     renvoyer C
```

C22. Pour le graphe pondéré `gt = [[(1,4),(2,2),(3,6)],[(0,4),(2,5),(3,2)],[(0,2),(1,5),(3,6)],[(0,6),(1,2),(2,6)]]`, appliquer à la main l'algorithme précédent pour trouver une solution au problème TSP. La solution sera exprimée par une liste de triplets représentant les arêtes sélectionnées : `(s,t,w)` pour une arête entre `s` et `t` de poids `w`.

On suppose qu'on dispose d'une fonction `triplets(g)` qui renvoie la liste des arêtes de `g` sans doublons, sous la forme de triplets `(s,t,w)`.

C23. Écrire une fonction de signature `asort(t)` qui trie la liste des triplets d'un graphe pondéré en utilisant le tri fusion.

On suppose qu'on dispose des fonctions :

1. `has_cycle_mn(t)` qui prend une liste de triplets et renvoie `True` si le graphe engendré par les arêtes sélectionnées forment un cycle de taille strictement inférieure à n, `False` sinon.
2. `has_deg_s2(t)` qui prend une liste de triplets et renvoie `True` si le graphe engendré par les arêtes sélectionnées possède un sommet dont le degré est supérieur strictement à 2, `False` sinon.

C24. Écrire une fonction de signature `gtsp(g)` qui implémente l'algorithme 2 et renvoie le circuit du voyageur de commerce trouvé sous la forme d'une liste d'entier.

C25. Quelle est la complexité temporelle de la fonction `gtsp`? On supposera que les fonctions `triplets`, `has_cycle_mn` et `has_deg_s2` sont de complexité linéaire en fonction du nombre de sommet du graphe engendré.

D Annexe

Python	Résultat / Effet
`with open("monfichier.txt","r")as f:`	Ouvre un bloc qui permet de lire dans le fichier `f`
`f.readlines()`	Renvoie la liste de toutes les lignes du fichier. Chaque ligne est une chaîne de caractères.
`with open("monfichier.txt","w")as f:`	Ouvre bloc qui permet d'écrire dans le fichier `f`
`f.writeline(line)`	Écrit la chaîne de caractères `line` dans le fichier représenté par `f`. Cette fonction en renvoie rien.
"33,45,67".split(',')	Renvoie la liste des champs séparés par une virgule. Chaque champ est une chaîne de caractères. Dans ce cas, le résultat est `["33","45","67"]`
`s += s + "\n"`	Ajouter un retour à la ligne à une chaîne de caractères.
`s[:-1]`	La chaîne de caractères `s` sans le dernier caractère.
`d = {}`	Création d'un dictionnaire vide.
`d[clef] = val`	Ajout d'une entrée pour la clé `clef` de valeur `val`
`if clef in d:`	Teste si une clé existe dans le dictionnaire.