TD03 [Graphes – 3] : Graphes non orientés

1 Rappels

- Vocabulaire représentation des graphes : corrigé Info1A_Graphes-1_TP13_2021-22
- Parcours de graphes : corrigé Info1A_Graphes-1_TP14_2021-22
- Algorithme de Dijkstra : corrigé Info1A_Graphes-1_TP15_2021-22

2 Préliminaires

Question 1. Rappeler quel est le nombre maximal d'arêtes d'un graphe (simple, *i.e.* sans arêtes multiples et sans boucle) non orienté d'ordre n.
Si toutes ces arêtes sont présentes dans le graphe, on parle d'un **graphe complet**.

On rappelle que la fonction `ord`, avec pour argument un caractère, renvoie l'entier codant ce caractère selon le standard Unicode : par exemple, l'appel `ord('A')` renvoie 65), et que la fonction `chr` réalise l'opération inverse : l'appel `chr(65)` renvoie le caractère `'A'`.
On rappelle que les 26 lettres majuscules de l'alphabet latin, sont codés par des entiers consécutifs.

Question 2. Définir à l'aide d'une boucle et en faisant appel à la fonction `chr`, une liste `alphabet`, de longueur 26, dont les éléments sont les 26 lettres majuscules de l'alphabet latin.
Proposer également une construction de la liste `alphabet` à l'aide d'une compréhension de listes.

Un graphe non orienté d'ordre n peut être défini par la liste de ses sommets, $s_1, s_2, \dots, s_n$ et la liste de ses arêtes, une arête d'extrémités u et v pouvant être représentée par un couple (u, v) où l'ordre n'importe pas dans la mesure où il est posé que le graphe est non orienté.

Question 3. Construire à l'aide d'une boucle une liste des arêtes, `LaretesC5`, pour un graphe (simple) complet d'ordre 5, dont les sommets sont étiquetés A, B, C, D et E.

Question 4. À l'aide du module `matplotlib.pyplot`, produire une représentation graphique d'un graphe simple complet d'ordre 5. Les sommets en seront représentés par des points situés aux sommets d'un pentagone régulier et les arêtes par des segments.
On créera, un dictionnaire `DCoords5`, dont les clés seront les noms des sommets, et où la valeur associée à chacun des sommets sera le tuple, (x, y), des ses coordonnées.

Le module sera importé à l'aide de l'instruction

```
import matplotlib.pyplot as plt
```

On pourra utiliser :

- pour représenter un nuage de points, l'instruction

```
plt.scatter(Lx, Ly, color = 'black')
```

  où `Lx` et `Ly` désignent, respectivement, la liste des abscisses et la liste des ordonnées des points du nuage.
- pour représenter chaque arête, l'instruction

```
plt.plot([x1, x2], [y1, y2], color = 'black', linestyle = '-')
```

On rappelle que pour obtenir l'affichage de ces différents éléments graphiques dans une fenêtre, il faut ensuite exécuter l'instruction

```
plt.show()
```

On souhaite définir une fonction permettant de créer un graphe non orienté d'ordre n, avec $1 \leq n \leq 26$, comprenant un nombre d'arêtes aléatoire.

Question 5. Définir une fonction, `graphe_alea_no(n, p)`, prenant en argument un entier, n, et un flottant p compris entre 0 et 1, et renvoyant un dictionnaire des listes d'adjacences correspondant à un graphe non orienté d'ordre n que l'on aura généré aléatoirement.
Les n sommets du graphe seront étiquetés par les n premières lettres majuscules de l'alphabet latin.

Les arêtes du graphe seront générées selon l'algorithme suivant :
- pour chacune des arêtes qu'il est possible de définir dans un tel graphe (*cf.* **question 1.**), on effectuera un tirage aléatoire à l'aide de la fonction `random` du module `random`, de sorte que chaque arête possible aura une probabilité p d'être construite ou non.

On rappelle que la fonction `random`, appelée sans argument, renvoie un flottant compris dans l'intervalle $[0\,;1[$, généré aléatoirement.
On testera la fonction avec $n = 5$ et $p = 0{,}7$.

Question 6. Définir une fonction `affiche_graphe_no(DLAdj, DCoords)`, prenant en argument un dictionnaire des listes d'adjacence d'un graphe non orienté et un dictionnaire des coordonnées des sommets de ce graphe, tous deux ayant pour clés les noms des sommets, et générant un affichage de ce graphe, les sommets étant indiqués par des points, étiquetés par leur nom, et les arêtes par des segments.

On donne l'instruction `plt.text(x, y, text)` permettant d'afficher une chaîne de caractères, `text`, à partir du point de coordonnées (x, y).
On testera la fonction avec le graphe généré à la question **5**. et le dictionnaire `DCoords5`.

On souhaite pouvoir sauvegarder dans un fichier texte un graphe non orienté généré de façon décrite précédemment.

Question 7. Définir une fonction `sauve_graphe_no(DLAdj, DCoords, chemin_fichier)`, prenant en argument un dictionnaire des listes d'adjacence d'un graphe non orienté et un dictionnaire des coordonnées des sommets de ce graphe, tous deux ayant pour clés les noms des sommets, et un chemin vers le fichier à créer, créant dans le répertoire spécifié un fichier texte avec le nom et l'extension voulus, tel que :
- la première ligne comportera uniquement un entier donnant l'ordre du graphe ;
- les n premières lignes comporteront, chacune, séparés par une tabulation, le nom d'un sommet, l'abscisse et l'ordonnée à laquelle on souhaite le faire apparaître dans une représentation graphique ;
- les autres lignes comporteront, chacune, séparés par une espace, les extrémités d'une arête du graphe.

On testera la fonction avec le graphe généré à la question **5**. et le dictionnaire `DCoords5`.

On rappelle que dans un graphe non orienté, on appelle **chemin** une séquence de sommets, $(s_0, s_1, \dots, s_n)$ telle que, pour tout i compris entre 0 et $n-1$, (s_i, s_{i+1}) soit une arête du graphe (analogue d'un circuit pour un graphe orienté).
On appelle **cycle**, un chemin dont les premier et dernier sommets sont identiques (analogue d'un circuit dans un graphe orienté).
On souhaite écrire une fonction détectant si un graphe non orienté comporte ou non un cycle.

Question 8. Décrire un algorithme de complexité $O(n)$, s'appuyant sur un parcours du graphe et permettant de détecter la présence ou l'absence d'un cycle dans un graphe non orienté d'ordre n.
Note : ce problème est nettement plus facile à résoudre si le graphe est non orienté.

Question 9. Implémenter une fonction `est_acyclique_no(DLAdj)`, prenant en argument un dictionnaire des listes d'adjacence d'un graphe non orienté, et renvoyant `True` si le graphe est acyclique et `False` sinon.
On testera la fonction sur des graphes non orientés générés aléatoirement à l'aide des fonctions précédemment définies.

On souhaite maintenant mettre en œuvre l'algorithme de Dijkstra sur des graphes non orientés pondérés par des poids positifs, générés aléatoirement.

Question 10. Modifier la fonction définie à la **question 5.**, en une fonction `graphe_alea_pond_no(n, p, m)` qui génèrera un graphe non orienté de la même façon et renverra un dictionnaire des listes d'adjacence de ses sommets, mais en outre un dictionnaire dont les clés seront les arêtes du graphe, sous forme de couples (tuples) de noms de sommets, avec pour valeurs associées, un entier tiré aléatoirement entre 1 et m.

Question 11. Définir une fonction `plus_court_chemin(DLAdj, DPoids, s0, s_cible)`, prenant en argument un dictionnaire des listes d'adjacence d'un graphe, un dictionnaire donnant pour chaque arête, (décrite par un tuple de noms de sommets) son poids (positif), et deux noms de sommets, et renvoyant la longueur d'un plus court chemin entre les sommets `s0` et `s_cible`, ainsi que la liste des sommets d'un chemin réalisant ce minimum.
On appliquera l'algorithme de Dijkstra.