Partie 2 : Représentation d'un graphe

[Question 1]

Écriture de la classe Arc et de son constructeur

[Question 2]

Écriture de la classe Arcs

[Question 3]

Écriture de l'interface Graphe

[Question 4]

Écriture de la classe GrapheListe

[Question 5]

Écriture d'une main qui créer le graphe présenté dans l'énoncé

[Question 6]

Écriture d'une méthode toString qui affiche le graphe de la manière demandé dans l'énoncé

[Question 7]

Écriture d'une classe TestGrapheListe qui teste :

- Si le graphe est construit correctement
- La recherche d'un arc inexistant
- Avec des sommets sans adjacents
- Avec un graphe construit, mais la liste de nœud Incomplète

Partie 3 : Calcul du plus court chemin par point fixe

[Question 8]

Écriture d'un Algorithme pour la fonction pointFixe.
Celui-ci a été mis en commentaire dans la classe BellmanFord.java

[Question 9]

Écriture de la méthode resoudre qui prend en paramètre un graphe et un sommet de départ (String)
Et qui nous renvoie un objet de classe Valeur qui contient un arbre des plus cours chemin

[Question 10]

Nous avons écrit un main dans la classe Main qui créer un graphe et utilise la méthode resoudre de la classe BellmanFord pour trouver les plus cours chemins.

[Question 11]

Nous avons fait plusieurs tests afin de tester la methode resoudre de l'algorithme de BellmanFord :

- Une methode qui vérifie les chemins en partant du point A avec le graphe fourni dans le sujet.
- Une methode qui vérifie les chemins en partant du point E avec le graphe fourni dans le sujet.
- Une méthode qui vérifié que la méthode renvoie null si le sommet de départ n'existe pas dans le graphe

[Question 12]

Nous ne l'avons pas fait par manque de temps

Partie 4 : Calcul du meilleur chemin par Dijkstra

[Question 13]

Écriture de la méthode resoudre qui prend en paramètre un graphe et un sommet de départ (String)
Et qui nous renvoie un objet de classe Valeur qui contient un arbre des plus cours chemin

[Question 14]

Nous avons fait plusieurs tests afin de tester la methode resoudre de l'algorithme de Dijkstra :

- Une methode qui vérifie les chemins en partant du point A avec le graphe fourni dans le sujet.
- Une methode qui vérifie les chemins en partant du point E avec le graphe fourni dans

le sujet.
- Une méthode qui vérifié que la méthode renvoie null si le sommet de départ n'existe pas dans le graphe

[Question 15]
Nous avons écrit une classe MainDijkstra qui créer un graphe et utilise la méthode resoudre de Dijkstra pour trouver les plus cours chemins.

Partie 5: Validation et expérimentation

[Question 16]
Avec le graphe fourni dans l'énoncé et en partant de A, le résultat obtenu est :
A -> V:0.0 p:indefini
B -> V:12.0 p:A
C -> V:76.0 p:D
D -> V:66.0 p:E
E -> V:23.0 p:B
V: représente le coût pour aller de A vers le somment
p: Montre l'antécédent du sommet.

[question 17]
Les résultats obtenus sont égaux malgré l'utilisation de deux algorithmes différents.
Cela s'explique par la taille du graphe et ses chemins. En effet, celui-ci est tès petit et possède peu de chemin.
Il n'y a donc pas beaucoup de possibilité.
L'algorithme qui fonctionne le mieux est celui de Dijkstra, car en bloquant à chaque fois un nœud, il permet une exécution plus rapide avec un résultat égal sur un petit graphe.

[Question 18]
Pour moi l'algorithme le plus efficace est celui de Dijkstra, car sa complexité est de (Sommets+arrêtes)*ln(Sommets) tandis que celui de Bellman Ford est de (Arrêtes* Sommets)
Ainsi, plus le graphe est grand, moins l'algorithme de Bellman Ford est efficace tandis que celui de Dijkstra est une addition multipliée par un logarithme ce qui rend un plus petit nombre.

[Question 19]
Nous n'avons pas eu le temps de la faire

Conclusion générale :
Nous avons appris a :
- retransmettre un Algorithme en code Java
- utiliser un dépôt Git pour faciliter le travail a plusieurs
- Analyser un problème avec méthode

Nous avons rencontré beaucoup de difficulté.
Tout d'abord notre classe (S2C) avait 2 heures de cours en présentiel de moins que les autres classes
Cela nous a donc contraint à travailler à la maison pour essayer de finir cette SAE
Cela à également poser des difficultées pour la communication à distance.
Pour finir, nous avons eu du mal à écrire les deux algorithmes, car les méthodes étaient plus compliqué que celle vues en cours

Pour conclure, nous avons appris de nombreuses choses avec cette SAE comme le travail a plusieurs à l'aide de Git,
l'implémentation en code d'un algorithme et la gestion du temps.