Simonin Enzo S2A
Mangin Florian

Rapport SAE Graphe

Pour faire fonctionner notre code vous devez installer les librairies ;

Lien pour télécharger les packages

Ces librairies nous ont été utiles dans la partie 4 pour stocker nos résultats dans des fichiers excel pour mieux analyser les résultats.

Partie 1 : Représentation d'un graphe (4h)

Dans cette partie nous avons effectuez :

- la classe Noeud
  - un attribut nom de type String
  - une List d'Arc adj
  - un constructeur qui initialise la liste et le nom
  - la méthode equals qui vérifie si deux Noeuds sont égaux
  - la méthode void ajouterArc(String destination, double cout) qui permet d'ajouter un Arc au Noeud
- la classe Arc
  - un attribut dest de type String qui représente la destination
  - un attribut cout de type double qui représente le coût de l'arc
  - un constructeur qui initialise ces 2 attributs
  - un getter getDest() qui retourne la destination
  - un getter getCout() qui retourne le coût
- l'interface Graphe
  - une méthode listeNoeuds qui retourne une liste avec tous les noeuds du graphe
  - une méthode suivants(String n) qui retourne un liste avec tous les arcs partant du Noeud n
- une classe GrapheListe qui implémente Graphe
  - un attribut ensNom qui est une List de String
  - un attribut ensNoeuds qui est une List Noeud
  - la méthode listeNoeuds() de l'interface Graphe
  - la méthode suivants() de l'interface Graphe
  - la méthode ajouterArc(String depart, String destination, double cout) qui sert a ajouter un arc dans le graphe
  - une méthode toString() pour afficher le graphe
  - une méthode toGraphviz() qui retourne une visualisation de graphe
  - un constructeur pour initialiser un graphe avec un fichier texte
- une classe main pour nous permettre de faire fonctionner et tester le code avec un graphe
- une classe TestGrapheListe
  - une méthode testFigure1InitialisationALaMain() pour tester le code effectuez dans cette première partie

<u>Partie 2 :</u> Calcul du plus court chemin par point fixe (2h)
Question 13 :

[Pastebin de notre algorithme de point fixe]

Dans cette partie nous avons effectuez :

- la classe BellmanFord
    - une méthode resoudre(Graphe g, String depart) qui permet de trouver le chemin les plus courts possible dans un graphe depuis le point depart
- Nouvelle classe TestPointFixe
    - la méthode testMethodeDuPointFixe() qui permet de tester la méthode résoudre
    - la méthode testMethodeDuPointFixe2() qui permet de tester la méthode résoudre mais en partant d'un point différent de la première
- dans la classe Main nous avons ajouté du code pour voir les résultats de la méthode résoudre sur un graphe
- dans le classe Valeur qui nous était donnée nous avons ajouté :
    - la méthode calculerChemin(String destination) qui permet de calculer le chemin le plus court vers le Noeud destination

<u>Partie 3 :</u> Calcul du meilleur chemin par Dijkstra (2h)

Dans cette partie nous avons :

- Classe Dijkstra:
    - traduit l'algorithme qui nous était donné dans une classe Dijkstra, il permet de résoudre les plus court chemin dans un graphe avec la méthode de Dijkstra
- Classe TestDjikstra
    - ajouté le méthode testMethodeDijkstra() dans la classe TestDijkstra qui permet de tester notre méthode résoudre dans la classe Dijkstra
    - ajouté la méthode testCalculerCheminDijkstra() dans la classe TestDijkstra qui permet de test la méthode calculerChemin avec la méthode Dijkstra
- Classe MainDjikstra
- Ce main permet d'effectuer une résolution des plus courts chemins avec la méthode de Dijkstra en fonction de paramètre qui nous sont demandés

Partie 4 : Validation et expérimentation (4h)

Question 21 :
L'algorithme de Dijkstra fait 7 itération avant d'avoir le résultat final alors que l'algorithme de BellmanFord en fait seulement 3. Dans l'algorithme de Dijkstra, celui fonctionne par sommet du graphe un par un ce qui demande beaucoup d'itération, néanmoins il donne de bon résultat au final.
Dans l'algorithme de BellmanFord il essaie de faire tous les sommets possible par itération ce qui, pour un graphe comme celui-ci, est très efficace.

Question 22 :
Nous pouvons en déduire que pour une graphe assez linéaire comme celui-ci avec peu de successeurs, l'algorithme de Dijkstra est beaucoup moins efficace que celui de BellmanFord qui modifie tous les points en même temps.

Question 23 :

```
Temps moyen de Dijkstra : 436776389
Temps moyen de Bellman : 19733646
```

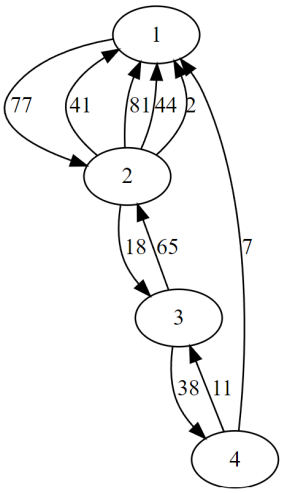
Le temps moyen pour l'algorithme de Dijkstra est d'environ de 0,44 secondes par graphes.
Le temps moyen pour l'algorithme de BellmanFord est d'environ 0,02 secondes par graphes.
Selon nous l'algorithme le plus efficace est celui de BellmanFord qui modifie tous les points en même temps.

Nous avons utilisé l'API Apache Poi pour générer un fichier Excel à partir des données.
Lien vers l'API :
http://www.codeurjava.com/2015/04/lecture-ecriture-dans-un-fichier-excel-apache-poi.html

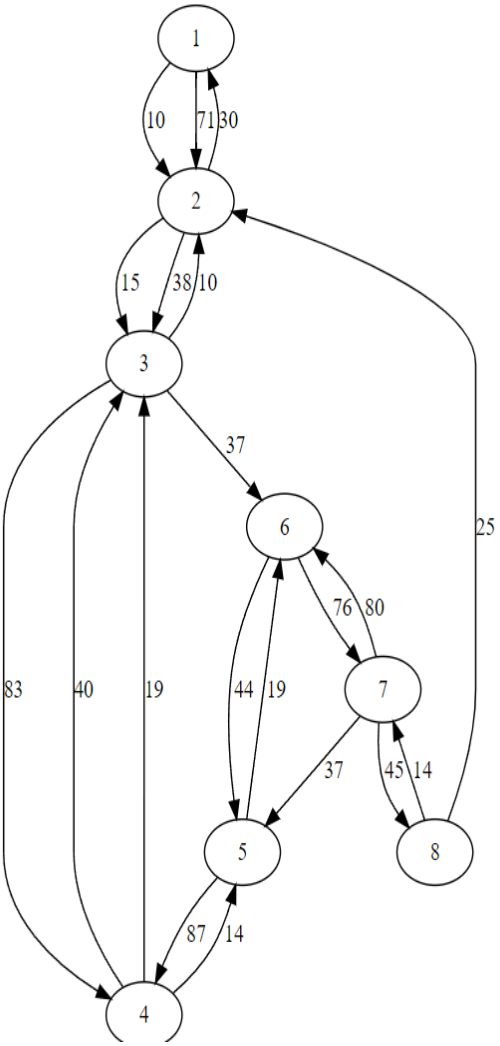
Question 24 :
Dans cette partie nous devions générer des graphes aléatoirement nous avons ajouté la classe GenerationGraphe, dans cette classe nous générons un graphe à partir de plusieurs paramètres qui nous sont demandés (le nom du graphe, sont nombres de noeuds, sont noeuds de départ et de fin).

Question 25 :
Pour générer des graphes aléatoirement nous utilisons la classe GenerationGraphe que nous avons créé.
Quelques exemples de graphes générés aléatoirement :

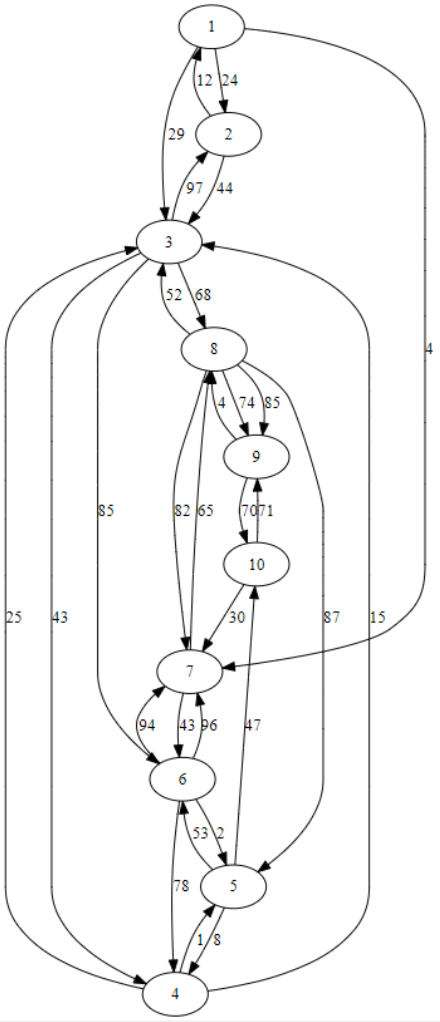
Graphe 1 :



Graphe 2 :



Graphe 3 :



Question 26 :

```
Temps moyen de Dijkstra : 9420925
Temps moyen de Bellman : 1901650
```

Le temps moyen par graphe généré avec l'algorithme de Dijkstra est d'environ 0,0094 secondes.
Le temps moyen par graphe généré avec l'algorithme de BellmanFord est d'environ 0,0019 secondes.
D'après ces résultats on peut en déduire que l'algorithme de Dijkstra est moins performant que celui de BellmanFord.

Question 27 :

Quelques ratio de performance entre l'algorithme de Dijkstra et de BellmanFord selon le nombre de noeuds :

- 10 noeuds : 1,48
- 40 noeuds : 1,18
- 90 noeuds : 2,44
- 500 noeuds : 8,75
- 800 noeuds : 9,93

On peut remarquer que plus le nombre de noeuds est élevé plus la différence de temps entre l'algorithme de Dijkstra et de BellmanFord est grande, l'algorithme de Dijkstra prend de plus en plus de temps lorsqu'il y a beaucoup de noeuds il est donc plus efficace dans des petits graphes.

Question 28 :

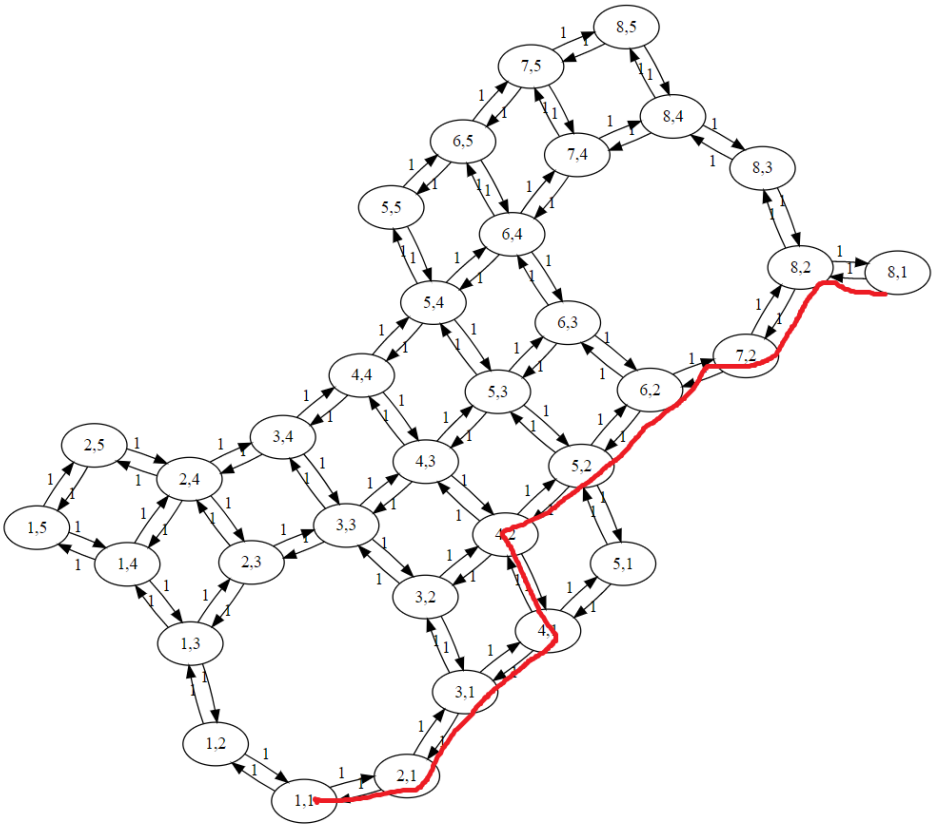
D'après nos tests on peut en conclure que l'algorithme de Dijkstra et l'algorithme de BellmanFord sont tous les deux utiles et peuvent être rapides mais pas dans les mêmes graphes. L'algorithme de Dijkstra est plus rapide dans les petits graphes alors que l'algorithme de BellmanFord est plus utile dans les grand graphes donc dans les graphes avec beaucoup de nœuds et d'arc.

Question 29 :

Nous avons ajouté une méthode genererGraphe dans la classe Labyrinthe qui permet de générer un graphe à partir d'un labyrinthe, les nœuds représentent des Noeuds et les Arcs représentent les liens entre les cases.
Nous avons également ajouté une classe TestLabyrinthe pour tester cette méthode genererGraphe, ainsi qu'une classe MainLabyrinthe pour visualiser les résultats.

Question 30-31 :

Pour trouver les résultats ci-dessous nous avons dû utiliser le pattern adapter.
Nous avons ajouté une classe AdapterGrapheLabyrinthe qui implémente la classe Graphe. Cette classe ressemble à la classe GrapheListe sauf qu'elle contient un attribut labyrinthe en plus et permet d'adapter un labyrinthe en générant un graphe pour nous permettre de faire la méthode du point fixe et de Dijkstra.
Nous avons également ajouté une classe TestAdapterGrapheLabyrinthe pour tester ceci.



["1,1", "2,1", "3,1", "4,1", "4,2", "5,2", "6,2", "7,2", "8,2", "8,1"]

On peut voir que le chemin pour relier "1,1" à "8,1" est bien un des chemins les plus courts.

Conclusion :

Dans cette SAE qui mélange ce que nous avons appris depuis le début de l'année en programmation et le cours de graphe, nous avons appris à utiliser des algorithmes mathématiques de manière automatique. Nous avons aussi réussi à implémenter la méthode de BellmanFord (point fixe) et la méthode de Dijkstra et à tester ces deux méthodes sur les graphes qui nous étaient fournis.

D'après nos tests, l'algorithme le plus efficace est celui du point fixe surtout dans des grands graphes parce qu'il permet de modifier tous les nœuds en même temps.

Nous n'avons pas eu de grosses difficultés pendant cette SAE mais nous avons quand même passé plus de temps que ce que nous pensions sur la méthode du point fixe et sur celle de Dijkstra, nous avons eu du mal à proposer les deux algorithmes pour résoudre les chemins les plus courts. Puisqu'il fallait mélanger les mathématiques avec des algorithmes lourds et l'informatique pour ainsi traduire ces algorithmes.