SAE 2.2 - Exploration Algorithmique
Recherche de plus court chemin dans un graphe

2. Représentation d’un graphe

2.2 Implémentation

Création de la classe Nœud qui représente un nœud du graphe, et de la classe Arc qui représente un arc partant d’un nœud, qui nous servirons à créer un Graphe. Création de l’interface Graphe

2.3 Classe GrapheListe

Création de de GrapheListe qui implémente l’interface Graphe et permet de représenter les données associé à un graphe.
Ajout d’une méthode toString, qui permet de représenter un graphe avec une chaîne de caractère.
Ajout d’une méthode toGraphviz qui permet de réprésenter graphiquement un graphe grâce à Graphviz.
Test sur Noeud, Arc, et GrapheListe.
Ajout d’un constructeur qui permet de créer un graphe à partir d’un fichier.

3. Calcul du plus court chemin par point fixe

3.1 Algorithme du point fixe

```
fonction resoudre(graphe:Graphe,depart:String):L(X)
   debut
   pour chaque noeud dans graphe faire:
      L(noeud)=infini
      parent(noeud)=null
   fpour
   L(depart)=0
   changement=vrai
   tant que changement faire
      changement=faux
      pour chaque arc(u,v) dans graphe faire
         si L(v)>L(u)+arc(u,v).cout alors
            changement=vrai
            L(v)=L(u)+arc(u,v).cout
            parent(v)=u
         fsi
      fpour
   fpour
   retourner L(x)
   fin
```

3.3 Programmation de l’algorithme du point fixe

Création de la classe BellmanFord et de la méthode resoudre qui permet de faire la méthode du point fixe sur un graphe donné.

3.4 Test de l’algorithme de point fixe

Test sur la classe BellmanFord

3.5 Calcul du meilleur chemin

Ajout de la méthode CalculerChemin dans la classe Valeur, qui donne le chemin le plus court en partant d’un point de départ à une destination donné.

4. Calcul du meilleur chemin par Dijkstra

4.2 Algorithme de Dijkstra

Création de la classe Dijkstra, et de la méthode resoudre qui fait l’algorithme de Dijkstra

4.3 Validation Dijkstra

Test sur la classe Dijkstra

5. Validation et expérimentation

5.1 Comportement qualitatif des algorithmes

21. L’algorithme de Dijkstra ne change que les valeurs des voisins de la valeur traité à chaque itération alors que l’algorithme de Bellman Ford change toute les valeurs à chaque itérations.

22. L’algorithme de Dijkstra fait donc moins de choses à chaque itérations que l’algorithme de Bellman Ford. Ce qui veut dire que l’algorithme de Dijkstra est probablement plus efficace que l’algorithme de Bellman Ford

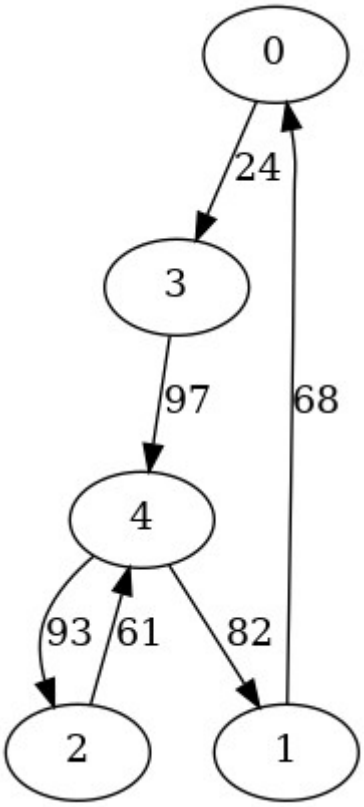
5.2 Graphe de tailles différentes

Création de la classe TestEfficaciteAlgo qui permet de comparer l’efficacité des 2 algos en calculant le temps d’exécution sur plusieurs graphes venant de fichiers et en faisant la moyenne.

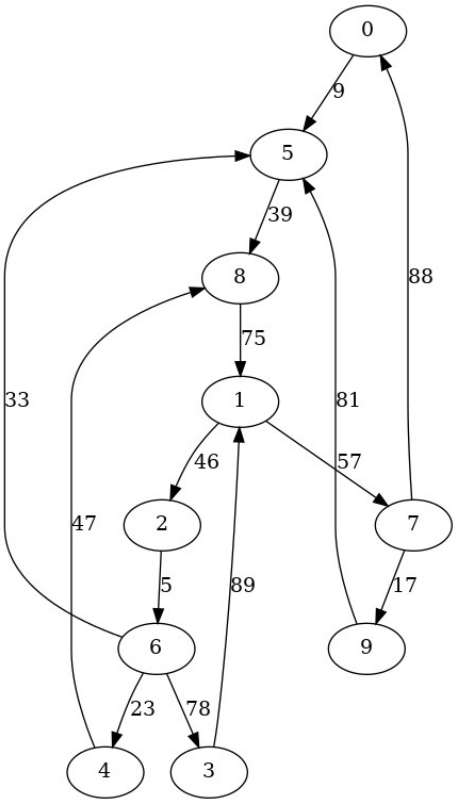
5.3 Génération de Graphes

Création de la classe generationGraphe qui permet de générer des graphes aléatoirement avec un nombre de nœuds donné

25.



5 nœuds



10 nœuds

5.4 Comparaison de résultats et conclusion

26. 100 graphes de taille 10

L’algorithme de Bellman Ford est plus lent de 0.2 ms

100 graphes de taille 25

L’algorithme de Bellman Ford est plus lent de 0.75 ms

100 graphes de taille 50

L’algorithme de Bellman Ford est plus lent de 1.6 ms

100 graphes de taille 100

L’algorithme de Bellman Ford est plus lent de 6 ms

Selon ces résultats le plus rapide est dijkstra.

27. 100 graphes de taille 10

Dijkstra 1.7 fois plus rapide

100 graphes de taille 25

Dijkstra 1.72 fois plus rapide

100 graphes de taille 50

Dijkstra 2.7 fois plus rapide

100 graphes de taille 100

Dijkstra 2.6 fois plus rapide

On peut conclure que ce rapport dépend du nombre de nœuds. Plus le nombre de nœuds est grand plus l'algoritme de Bellman Ford est lent.

28. On peut conclure que l'algorithme de Dijkstra est plus rapide dans tout les cas.