**Rapport: SAE Graphes**

Binôme :
LATH Victor
LIM HOUN TCHEN Aimé

**Structure du dossier du projet:**

- Dossier /graphes : contient tous les graphes de la SAE
  - graphes_exemple1.txt
  - graphes_boucle.txt
  - les autres graphes pour la question 23
- Dossier Algorithme : contient l'algorithme du point fixe écrite lors de la question 13
- Question23.java : Cette classe permet de répondre à la question 23
  - resultat.txt : contient les données de la question sous forme de csv
- Question21,Question25,Question30,Question31 : réponds aux questions respectives
- Main : répond à la question 15
- MainDijkstra : répond à la question 20
- Interface Algorithme
- BellmanFord
- Dijkstra
- Classe de représentation des données :
  - Noeud
  - Arc
  - GrapheListe
  - Graphe
  - Valeur
- Laby :
  - GrapheLabyrinthe
  - Labyrinthe

**Tests:**

- GraphesListeTest.java
  - ajouterArc() : teste la méthode ajouterArc dans GraphesListe
  - TestToGraphViz() : teste la méthode toGraphViz
  - TestToString() : teste la méthode toString
  - testCalculerChemin() : teste la méthode CalculerChemin
- DijkstraTest.java
  - testRechercherMinTest() : teste la méthode rechercherMin utilisé dans la méthode résoudre
  - resoudreTest() : test la méthode resoudre de Dijkstra
- BellmanFordTest.java
  - testBellmanFord() : test la méthhode resoude de BellmanFord
- LabyTest.java
  - testCreerGrapheAvecLabyrinthe() : test la méthode genererGraphe() permettant de générer un graphe en fonction du labyrinthe

## 1. Présentation de la SAE

## 2. Représentation d'un graphe

Question 1:

Création classe Noeud et constructeur prenant en paramètre le nom du nœud et initialisant la liste adj a une liste vide dans src/

Question 2:

Ajout méthode boolean equals(Object o) dans la classe Noeud

Question 3:

Ajout méthode void ajouterArc(String destination, double cout) dans la classe Noeud

Question 4:

Création classe Arc et constructeur prenant en paramètre le noeud de destination et le coût (valeur strictement positive) de l'arc créé dans src/

Question 5:

Ajout interface Graphe dans src/

Question 6:

Ajout classe GrapheListe dans src/

Question 7:

Ajout classe Question7 dans src/ possédant un main créant le graphe représenté.

Question 8:

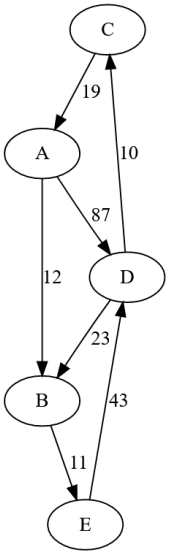
Ajout méthode toString dans GrapheListe

Question 9:

Ajout méthode toGraphviz dans GrapheListe

Question 10:

```
digraph {
C -> A [label = 19]
A -> D [label = 87]
A -> B [label = 12]
E -> D [label = 43]
D -> C [label = 10]
D -> B [label = 23]
B -> E [label = 11]
}
```




Le graphe est correctement généré et est visuellement correct

Question 11:

On a ajouté comme tests :

- Calcul du plus court chemin par point fixe

Question 12:

Ajout constructeur permettant de créer un objet GrapheListe à partir d'un nom de fichier contenant le descriptif du graphe

### 3. Calcul du plus court chemin par Point Fixe

Question 13 (algorithme du point fixe) :

```
fonction pointFixe(G InOut:Graphe, depart:Noeud)
  debut
    pour chaque sommet v de G faire
      v.distance <- infini
      v.précédent <- indéfini
    fpour
    depart.distance <- 0

    cpt <- 0
    vtmp <- null
    tant que non egal(v, vtmp) et cpt<nbSommets faire
      vtmp <- v.toString()
      pour chaque sommet v de G faire
        pour chaque arc (u, v) de G faire
          tmp <- u.distance + poids(u, v)
          si tmp < v.distance alors
            v.distance <- tmp
            v.précédent <- u
```

```
            fsi
          fpour
          cpt <- cpt+1
        fpour
      ftant
    fin
```

LEXIQUE:
G : Graphe, graphe orienté avec poids positive des arcs
depart : Noeud, un sommet de G
cpt : entier, nombre d'itérations
nbSommets : entier, nombre de sommets du graphe G

Question 14:

Ajout classe Valeur et BellmanFord dans src/ et méthode Valeur resoudre(Graphe g, String depart) dans la classe BellmanFord

Question 15:

A -> V:0.0 p:null
B -> V:12.0 p:A
C -> V:76.0 p:D
D -> V:66.0 p:E
E -> V:23.0 p:B

[A, B, E, D, C]

Question 16:

Ajout classe BellmanFordTest dans Test/ et méthode void testBellmanFord() qui teste le bon fonctionnement de l'algorithme de BellmanFord avec le graphe donné dans le sujet (figure 1)

Question 17:

Ajout méthode List<String> calculerChemin(String destination) dans la classe Valeur.

## 4. Calcul du meilleur chemin par Dijkstra

Question 18:

Ajout classe Dijkstra dans src/ et méthode Valeur resoudre(Graphe g, String depart) dans Dijkstra

Question 19:

Ajout classe DijkstraTest dans Test/ et méthode void resoudre() qui teste si les valeurs et le chemin sont bons et void testRechercherMin() qui teste qu'on retourne bien le minimum d'une liste d'arcs.

Question 20:

Ajout classe MainDijkstra dans src/ qui permet la lecture des graphes à partir de fichiers texte, le calcul des chemins les plus courts pour des nœuds données, l'affichage des chemins pour des nœuds donnés.

## 5. Validation et expérimentation

Question 21:

08 p:null
Dijkstra:
Iteration 0 :
A -> V:0.0 p:null
B -> V:1.7976931348623157E308 p:null
C -> V:1.7976931348623157E308 p:null
D -> V:1.7976931348623157E308 p:null
E -> V:1.7976931348623157E308 p:null
F -> V:1.7976931348623157E308 p:null
G -> V:1.7976931348623157E308 p:null

Iteration 1 :
A -> V:0.0 p:null
B -> V:20.0 p:A
C -> V:1.7976931348623157E308 p:null
D -> V:3.0 p:A
E -> V:1.7976931348623157E308 p:null
F -> V:1.7976931348623157E308 p:null
G -> V:1.7976931348623157E3
Iteration 2 :
A -> V:0.0 p:null
B -> V:20.0 p:A
C -> V:7.0 p:D
D -> V:3.0 p:A
E -> V:1.7976931348623157E308 p:null
F -> V:1.7976931348623157E308 p:null
G -> V:1.7976931348623157E308 p:null

Iteration 3 :
A -> V:0.0 p:null
B -> V:9.0 p:C
C -> V:7.0 p:D
D -> V:3.0 p:A
E -> V:1.7976931348623157E308 p:null
F -> V:1.7976931348623157E308 p:null
G -> V:1.7976931348623157E308 p:null

Iteration 4 :
A -> V:0.0 p:null
B -> V:9.0 p:C
C -> V:7.0 p:D
D -> V:3.0 p:A
E -> V:1.7976931348623157E308 p:null
F -> V:1.7976931348623157E308 p:null

G ->  V:19.0 p:B

Iteration 5 :
A ->  V:0.0 p:null
B ->  V:9.0 p:C
C ->  V:7.0 p:D
D ->  V:3.0 p:A
E ->  V:1.7976931348623157E308 p:null
F ->  V:24.0 p:G
G ->  V:19.0 p:B

Iteration 6 :
A ->  V:0.0 p:null
B ->  V:9.0 p:C
C ->  V:7.0 p:D
D ->  V:3.0 p:A
E ->  V:27.0 p:F
F ->  V:24.0 p:G
G ->  V:19.0 p:B
---
Bellman Ford:
Iteration 0 :
A ->  V:0.0 p:null
B ->  V:1.7976931348623157E308 p:null
C ->  V:1.7976931348623157E308 p:null
D ->  V:1.7976931348623157E308 p:null
E ->  V:1.7976931348623157E308 p:null
F ->  V:1.7976931348623157E308 p:null
G ->  V:1.7976931348623157E308 p:null

Iteration 1 :
A ->  V:0.0 p:null
B ->  V:9.0 p:C
C ->  V:7.0 p:D
D ->  V:3.0 p:A
E ->  V:38.0 p:F
F ->  V:35.0 p:G
G ->  V:30.0 p:B

Iteration 2 :
A ->  V:0.0 p:null
B ->  V:9.0 p:C
C ->  V:7.0 p:D
D ->  V:3.0 p:A
E ->  V:27.0 p:F
F ->  V:24.0 p:G
G ->  V:19.0 p:B

Iteration 3 :
A ->  V:0.0 p:null
B ->  V:9.0 p:C
C ->  V:7.0 p:D
D ->  V:3.0 p:A
E ->  V:27.0 p:F
F ->  V:24.0 p:G
G ->  V:19.0 p:B

Point fixe:
Dijkstra : 6 itérations
Bellman Ford : 3 itérations

La méthode Dijkstra cherche le chemin minimum pour chaque sommet alors que la méthode Bellman-Ford regarde le coût minimum en additionnant les antécédents des sommets.

Question 22:

Dans le cas de ce graphe,l'algorithme de Bellman-Ford est plus rapide car il ne prend que 3 itérations.

Question 23:

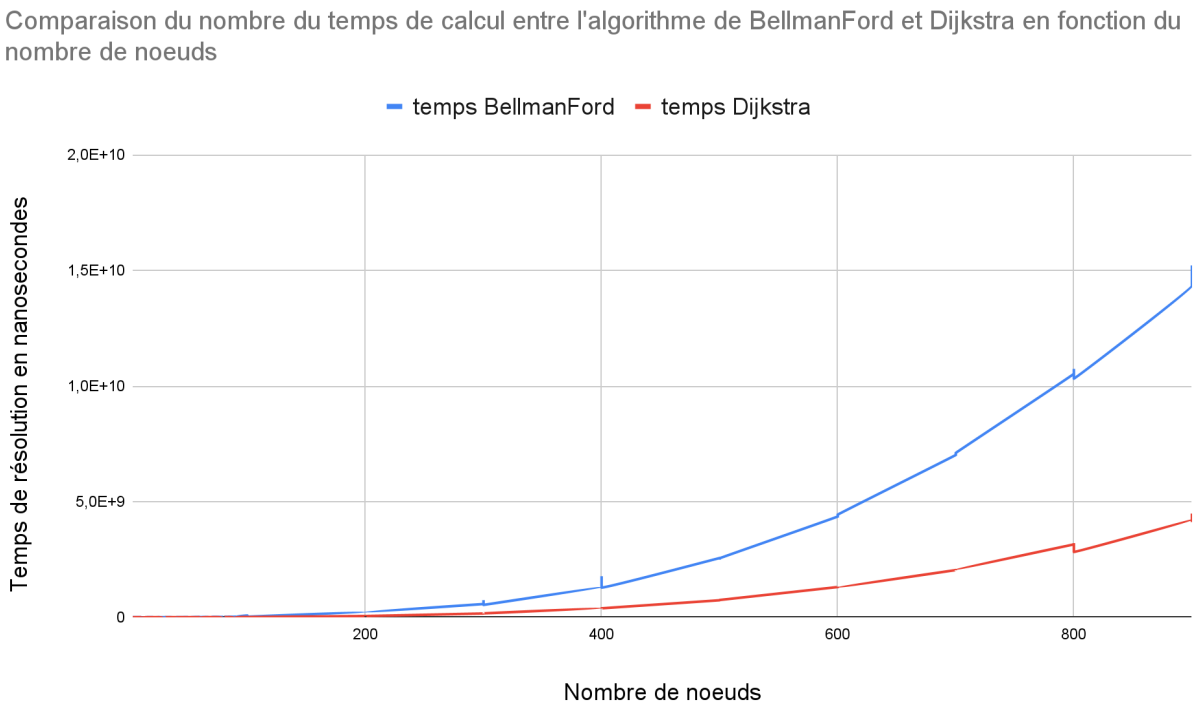
Données faites à l'aides des graphes données sur Arches et de la classe Question23

| nbNoeuds | nbArc | tempsBellmanFord | tempsDijkstra |
|---|---|---|---|
| 3 | 4 | 15400 | 20800 |
| 3 | 5 | 18000 | 22100 |
| 5 | 7 | 519800 | 194500 |
| 10 | 39 | 191300 | 100800 |
| 10 | 45 | 97600 | 68000 |
| 10 | 35 | 74800 | 56200 |
| 10 | 35 | 78800 | 60800 |
| 10 | 41 | 78900 | 59800 |
| 10 | 30 | 71400 | 53400 |
| 10 | 42 | 81900 | 60200 |
| 20 | 130 | 340500 | 127400 |
| 20 | 121 | 297100 | 112700 |
| 20 | 115 | 215200 | 110900 |
| 20 | 136 | 335100 | 121300 |
| 20 | 117 | 373700 | 120200 |
| 30 | 244 | 1058200 | 245200 |
| 30 | 274 | 839300 | 291700 |

| | | | |
|---|---|---|---|
| 30 | 259 | 791200 | 246500 |
| 30 | 263 | 787200 | 263100 |
| 30 | 248 | 778000 | 242300 |
| 40 | 441 | 1630300 | 523800 |
| 40 | 454 | 1683400 | 514300 |
| 40 | 426 | 1585900 | 534100 |
| 40 | 466 | 1722200 | 562700 |
| 40 | 474 | 1760300 | 565700 |
| 50 | 693 | 4095500 | 944300 |
| 50 | 704 | 4096800 | 1080500 |
| 50 | 667 | 3060600 | 1138100 |
| 50 | 701 | 3181000 | 967100 |
| 50 | 697 | 4385700 | 1518100 |
| 60 | 1002 | 7045700 | 1589600 |
| 60 | 935 | 5116600 | 1556500 |
| 60 | 977 | 8337700 | 1601600 |
| 60 | 977 | 6508900 | 1554200 |
| 60 | 965 | 5114800 | 1758700 |
| 70 | 1285 | 9712900 | 2463400 |
| 70 | 1302 | 9979100 | 2375800 |
| 70 | 1316 | 12836500 | 2355100 |
| 70 | 1329 | 7829100 | 2423700 |
| 70 | 1278 | 10234500 | 2346100 |
| 80 | 1675 | 11798200 | 3425100 |
| 80 | 1714 | 19361400 | 3911400 |
| 80 | 1693 | 15298800 | 3466900 |
| 80 | 1724 | 16113100 | 3586600 |
| 80 | 1668 | 14766500 | 3445900 |
| 90 | 2107 | 20932200 | 5038100 |
| 90 | 2094 | 17333300 | 5732200 |
| 90 | 2103 | 16578800 | 4869700 |
| 90 | 2167 | 22972600 | 4950800 |
| 90 | 2165 | 24667800 | 5729300 |
| 100 | 2685 | 73166400 | 11109700 |
| 100 | 2604 | 30348000 | 8120300 |
| 100 | 2696 | 34498400 | 8163700 |
| 100 | 2567 | 30031600 | 8343400 |

| | | | |
|---|---|---|---|
| 100 | 2547 | 22638500 | 8060200 |
| 200 | 10155 | 225134600 | 51402800 |
| 200 | 10342 | 233381500 | 50803700 |
| 200 | 10135 | 219807100 | 49811100 |
| 200 | 10053 | 204526200 | 46185800 |
| 200 | 10209 | 218460300 | 49683200 |
| 300 | 22810 | 569665200 | 160908200 |
| 300 | 22796 | 557269200 | 161039100 |
| 300 | 22962 | 738789200 | 159440900 |
| 300 | 22858 | 538178600 | 168163900 |
| 300 | 22927 | 532960100 | 166113000 |
| 400 | 40626 | 1306288000 | 396978900 |
| 400 | 40450 | 1323164300 | 426580400 |
| 400 | 40562 | 1774130100 | 380521700 |
| 400 | 40476 | 1736805100 | 372029700 |
| 400 | 40687 | 1286265900 | 385129800 |
| 500 | 63138 | 2561190400 | 734638400 |
| 500 | 63276 | 2537234800 | 750614000 |
| 500 | 63000 | 2557899400 | 755965200 |
| 500 | 62743 | 2488573000 | 733642700 |
| 500 | 63101 | 2550703400 | 744881500 |
| 600 | 90362 | 4358933100 | 1307155700 |
| 600 | 90545 | 4475181500 | 1327146900 |
| 600 | 90577 | 4390697400 | 1304324400 |
| 600 | 90432 | 4314336300 | 1257107500 |
| 600 | 90743 | 4437173200 | 1288383300 |
| 700 | 122868 | 7021748900 | 2039289200 |
| 700 | 123154 | 7126828700 | 2031217500 |
| 700 | 123216 | 6987022500 | 2090180800 |
| 700 | 122934 | 7035761600 | 2047966200 |
| 700 | 123490 | 7109094500 | 2060043000 |
| 800 | 161216 | 10531127300 | 3149843800 |
| 800 | 160776 | 10668341200 | 3126139300 |
| 800 | 161180 | 10743048000 | 3205077500 |
| 800 | 161005 | 10634383900 | 3079065600 |
| 800 | 160289 | 10324368100 | 2825675200 |
| 900 | 202660 | 14340384900 | 4222986900 |

| | | | |
|---|---|---|---|
| 900 | 203379 | 15174329100 | 4346832500 |
| 900 | 202692 | 14888345300 | 4155050100 |
| 900 | 204447 | 14670959700 | 4117173100 |
| 900 | 202857 | 15222602300 | 4486551500 |




D'après nos données on peut conclure que peu importe le nombre de nœuds, l'algorithme de Dijkstra est plus rapide que celui de Bellman Ford.

Question 24:

Ajout méthode static GrapheListe genererGraphe(int nbNoeuds) dans la classe GrapheListe qui permet de générer un graphe automatiquement en précisant le nombre de nœuds. L'idée de cette méthode consiste à relier chaque nœud au nœud suivant et il y a 50% de chances pour que ce nœud se lie aussi à un autre nœud aléatoire.

Question 25:

digraph {
n1 -> n2 [label = 48]
n2 -> n3 [label = 40]
n2 -> n2 [label = 18]
n3 -> n4 [label = 61]
n3 -> n6 [label = 75]
n4 -> n5 [label = 19]
n6 -> n7 [label = 9]
n5 -> n6 [label = 5]
n7 -> n8 [label = 19]

```
n8 -> n9 [label = 61]
n9 -> n10 [label = 7]
}

digraph {
n1 -> n2 [label = 50]
n1 -> n1 [label = 39]
n2 -> n3 [label = 72]
n3 -> n4 [label = 84]
n3 -> n6 [label = 76]
n4 -> n5 [label = 46]
n6 -> n7 [label = 86]
n5 -> n6 [label = 33]
n7 -> n8 [label = 87]
n8 -> n9 [label = 11]
n9 -> n10 [label = 44]
}

digraph {
n1 -> n2 [label = 88]
n1 -> n8 [label = 84]
n2 -> n3 [label = 4]
n2 -> n4 [label = 79]
n8 -> n9 [label = 48]
n8 -> n4 [label = 9]
n3 -> n4 [label = 87]
n3 -> n6 [label = 6]
n4 -> n5 [label = 35]
n4 -> n1 [label = 84]
n6 -> n7 [label = 72]
n6 -> n9 [label = 79]
n5 -> n6 [label = 95]
n5 -> n2 [label = 13]
n7 -> n8 [label = 14]
n7 -> n3 [label = 5]
n9 -> n10 [label = 55]
}
```

Question 26:

| Nb itérations BellMan-Ford | Nb itérations Dijkstra | Noeuds | Arcs | Temps BellMan-Ford (nanosecondes) | Temps Dijkstra (nanosecondes) |
|---|---|---|---|---|---|
| 2 | 10 | 10 | 16 | 16899700 | 509100 |
| 2 | 10 | 10 | 13 | 17237100 | 501900 |
| 2 | 10 | 10 | 14 | 16832500 | 447400 |
| 4 | 100 | 100 | 147 | 36512900 | 3999500 |
| 4 | 100 | 100 | 146 | 39001200 | 6161700 |
| 6 | 100 | 100 | 157 | 39724600 | 3414200 |
| 8 | 1000 | 1000 | 1493 | 477588800 | 120856900 |
| 9 | 1000 | 1000 | 1497 | 509431300 | 114067200 |
| 7 | 1000 | 1000 | 1527 | 488236200 | 118297000 |

Question 27:

L'algorithme de Dijkstra est 33 fois plus rapide que l'algorithme de BellMan-Ford avec 10 noeuds, 10 fois plus rapide avec 100 noeuds et 4 fois plus rapide avec 1000 noeuds

Question 28:

On peut en conclure que la méthode Dijkstra est plus rapide, mais il peut y avoir un très grand nombre d'itérations, ce qui peut être humainement difficile à utiliser comparé à la méthode BellMan-Ford qui a moins d'itérations en dépit d'être moins rapide.

## 6. Extension : Intelligence Artificielle et labyrinthe

Question 29:

Ajout méthode GrapheListe genererGraphe() dans Labyrinthe situé dans src/ qui permet de générer un graphe à partir du labyrinthe.

Question 30:

Labyrinthe utilisé : labytest.txt (le labyrinthe donné dans le sujet, figure 11)
Départ : (0, 0), Arrivée : (4, 0)
Chemin le plus court : [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (3, 2), (4, 2), (4, 1), (4, 0)]

Question 31:

Question effectuée dans la classe Question31
Labyrinthe utilisé : labytest.txt (le labyrinthe donné dans le sujet, figure 11)
Départ : (0, 0), Arrivée : (4, 2)
Chemin le plus court : [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (3, 2), (4, 2)]

**Bilan:**

Cette SAÉ nous à permis de mettre en pratique les algorithmes vus en cours donc cela nous a permis de mieux appréhender la notion de recherche de chemin le plus court.
Nous avons appris que l'algorithme de Dijkstra est bien plus rapide que celui du point fixe.