

# SAE Graphe

BAUDOUIN Mathieu  
GRECO Lucas  
S2A

## 2) Représentation d'un graphe

Dans cette partie nous avons créé la classe Arc, l'interface graphique et la classe GrapheListe.

Nous avons fait des test qui porte sur :

- Création d'un Arc
- Ajout d'un Arc à une liste de type Arcs
- Ajout d'un Arc qui n'existait pas à une GraphListe
- Récupérer l'Arc suivant dans une GrapheListe

## 3) Calcul du plus court chemin par point fixe

Dans cette partie nous avons fait la class pour l'algorithm de Bellman Ford et le main pour l'exécuter

Question 8)

Algo logique :

fonction pointFixe(Graphe g, Noeud depart)

Valeur valeurs

liste\_noeud <- g.getNoeud()

pour n dans noeud faire

L(n)<-infini

fpour

L(depart)<-0

boolean iter\_changed <- vrai

tant que iter\_changed

```

iter_changed <- false

pour n dans liste_noeud faire

    pour arc dans g.suivant(n).getArcs // arc de type Arc

        distance <- arc.get Cout
        dest <- arc.get Dest

        old_distance <- L(dest)
        new_distance <- L(n)+distance

        si (new_distance < old_distance) alors

            parent(dest)<-n
            L(dest)<-new_distance

            iter_changed <- true
        fsi
    fpour
fpour

```

Les tests que nous avons fait était les suivant :

- Assigner un parent à un sommet dans Valeur
- Retrouver le parent d'un sommet dans une Valeur
- Test sur l'algorithme de Bellman

## 4) Calcul du meilleur chemin par Dijkstra

Dans cette partie nous avons fait la classe pour l'algorithme de Dijkstra

Nous avons fait un test sur l'algorithme de Dijkstra

## 5) Validation et expérimentation

Dans la classe Comparaison.java, nous pouvons comparer les 2 algorithmes.

Voici les résultats sur une de nos machines.

```

Graphe61.txt      298100.0      210200.0      27/33
Graphe62.txt      231900.0      220900.0      27/34
Graphe63.txt      341500.0      202000.0      27/35
Graphe64.txt      290200.0      216300.0      27/36
Graphe65.txt      218200.0      223600.0      28/36
Graphe701.txt     4.00287E7      9.07287E7      29/36
Graphe702.txt     4.71761E7      1.001968E8      30/36
Graphe703.txt     4.44982E7      9.39674E7      31/36
Graphe704.txt     4.01227E7      8.95945E7      32/36
Graphe705.txt     4.72131E7      1.030002E8      33/36
Graphe71.txt      513300.0      298500.0      33/37
Graphe72.txt      402800.0      291800.0      33/38
Graphe73.txt      530200.0      325200.0      33/39
Graphe74.txt      316000.0      316700.0      34/39
Graphe75.txt      390100.0      283600.0      34/40
Graphe801.txt     5.88524E7      1.328454E8      35/40
Graphe802.txt     5.48215E7      1.324121E8      36/40
Graphe803.txt     5.44523E7      1.307701E8      37/40
Graphe804.txt     5.33164E7      1.350542E8      38/40
Graphe805.txt     5.04002E7      1.300368E8      39/40
Graphe81.txt      412400.0      380200.0      39/41
Graphe82.txt      911100.0      374900.0      39/42
Graphe83.txt      541200.0      370700.0      39/43
Graphe84.txt      551000.0      384700.0      39/44
Graphe85.txt      534200.0      368300.0      39/45
Graphe901.txt     6.78101E7      1.832591E8      40/45
Graphe902.txt     6.67077E7      1.833367E8      41/45
Graphe903.txt     6.59162E7      1.819708E8      42/45
Graphe904.txt     6.48725E7      1.83311E8      43/45
Graphe905.txt     7.8908E7       2.100112E8      44/45
Graphe91.txt      982100.0      515000.0      44/46
Graphe92.txt      531000.0      479700.0      44/47
Graphe93.txt      546400.0      544700.0      44/48
Graphe94.txt      707800.0      510400.0      44/49
Graphe95.txt      700300.0      522100.0      44/50
Bellman-ford : 1.3952810638297873E7      44
Dijkstra : 1.3952810638297873E7 50

C:\Users\greco\SAE_graph\src>|

```

On observe que, sur les graphes donnés, les deux algorithmes se valent au niveau de la moyenne des temps d'exécution. Cependant, l'algorithme de Dijkstra est plus performant dans 50 cas sur 94. Il a aussi un écart-type plus petit.

## Conclusion )

Dans cette SAE, nous avons appris à implémenter des algorithmes de recherche de chemin et à les comparer.