Robert Antonin
Noel Alexandre
Trieu Zhi-Sheng

Rapport SAE 2.2 :

Table des matières :

I. Représentation d'un graphe

Première partie de la SAE qui à servi à poser les bases du projet. Nous avons implémenté les arcs et les nœuds des graphes et leurs méthodes qui perm ettent de les manipuler.

Nous avons donc écrit une classe Arc et Noeud ainsi que l'interface Graphe. Nous avons fait une classe GrapheListe qui implémente l'interface Graphe qui stock les nœuds d'un graphe sous forme de liste. Nous avons testé toutes ces classes dans un main, dans une classe de tests et grâce à la méthode toGraphViz qui permet d'afficher le graphe visuellement.

II. Point fixe

Nous avons d'abord écrit l'algorithme puis nous l'avons implémenté un tout petit peu différemment de ce que nous avions prévu. (Un boolean au lieu de comparer la ligne d'avant à chaque fois). Pour l'initialisation des valeurs nous avons décidé de créer une interface Algorithme qui va contenir une méthode initialisation et qui va se faire implémenter par la classe BellmanFord et Dijkstra.

Nous avons testé, dans la classe TestAlgorithme, les cas où :

- nous utilisons un graphe simple pour le cas de base
- nous utilisons un graphe plus complexe
- nous utilisons le graphe "boucle" (cf. figure 10 du sujet)
- nous utilisons un graphe plus "compliqué"

III. Dijkstra

Nous avons repris l'algorithme donné en cours et nous l'avons implémenté dans la classe Dijkstra qui implémente l'interface Algorithme.

Nous avons décidé de tester, dans la classe TestAlgorithme, les cas où :

- nous utilisons un graphe normal avec les bons départ et destination
- le graphe ne contient pas le noeud de destination
- nous utilisons le graphe "boucle" (cf. figure 10 du sujet)

IV. Validation et expérimentation

900 noeuds :

Temps d'exécution avec BellmanFord: 157655800 ns
Temps d'exécution avec Dijkstra: 122509700 ns

Dijkstra est plus rapide de : 35146100 ns

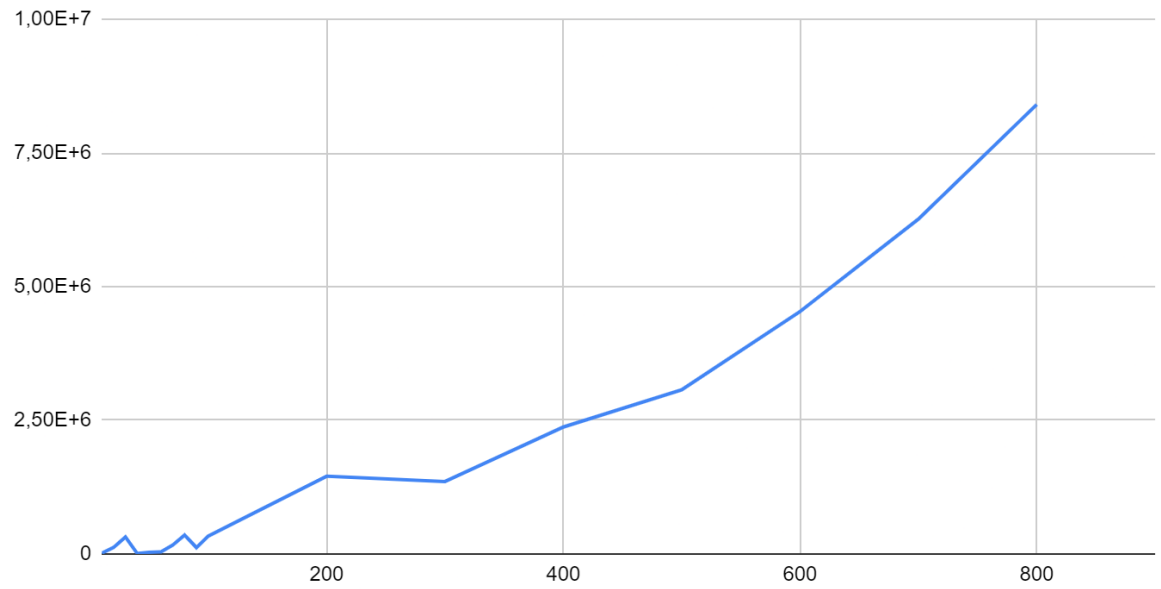
30 noeuds :
Temps d'exécution avec BellmanFord: 2804400 ns
Temps d'exécution avec Dijkstra: 726200 ns

Dijkstra est plus rapide de : ns

100 noeuds :
Temps d'exécution avec BellmanFord: 21315000 ns
Temps d'exécution avec Dijkstra: 4114000 ns

Dijkstra est plus rapide de : 17201000 ns

Évolution de la différence de temps entre les deux algorithmes en fonction du nombre de nœuds.



Graphique d'une moyenne de 1000 écarts de temps entre les deux sur toutes les nombres de points possible dans le sujet.

On peut remarquer qu'entre 40 et 50 nœuds le temps est minime et il est arrivé que Bellman Ford ait été plus rapide que Dijkstra. Mais que en grande majorité et plus le nombre de points augmente, plus l'algorithme de dijkstra est le plus efficace en termes de temps.

Question 21 :

Pour expliquer la différence de comportement entre l'algorithme de Bellman Ford et de Dijkstra, nous avons pris l'exemple du graphe "boucle" du sujet (figure 10).

Etat de l'objet valeur pour chaque itération pour la méthode du point fixe :

Noeuds	A	B	C	D	E	F	G
suivants	B(20) D(3)	G(10)	B(2)	C(4)		E(3)	F(5)
initialisation	0	+∞	+∞	+∞	+∞	+∞	+∞
1ère itération	0	9(C)	7(D)	3(A)	38(F)	35(G)	30(B)
2ème itération	0	9(C)	7(D)	3(A)	27(F)	24(G)	19(B)
Point Fixe	0	9(C)	7(D)	3(A)	27(F)	24(G)	19(B)

Etat de l'objet valeur pour chaque itération pour la méthode de Djikstra :

Noeuds	A	B	C	D	E	F	G
suivants	B(20) D(3)	G(10)	B(2)	C(4)		E(3)	F(5)
initialisation	0	+∞	+∞	+∞	+∞	+∞	+∞
1ère itération	0	20(A)	+∞	3(A)	+∞	+∞	+∞
2ème itération	0	20(A)	7(D)	3(A)	+∞	+∞	+∞
3ème itération	0	9(C)	7(D)	3(A)	+∞	+∞	+∞
4ème itération	0	9(C)	7(D)	3(A)	+∞	+∞	19(B)
5ème itération	0	9(C)	7(D)	3(A)	+∞	24(G)	19(B)
6ème itération	0	9(C)	7(D)	3(A)	27(F)	24(G)	19(B)

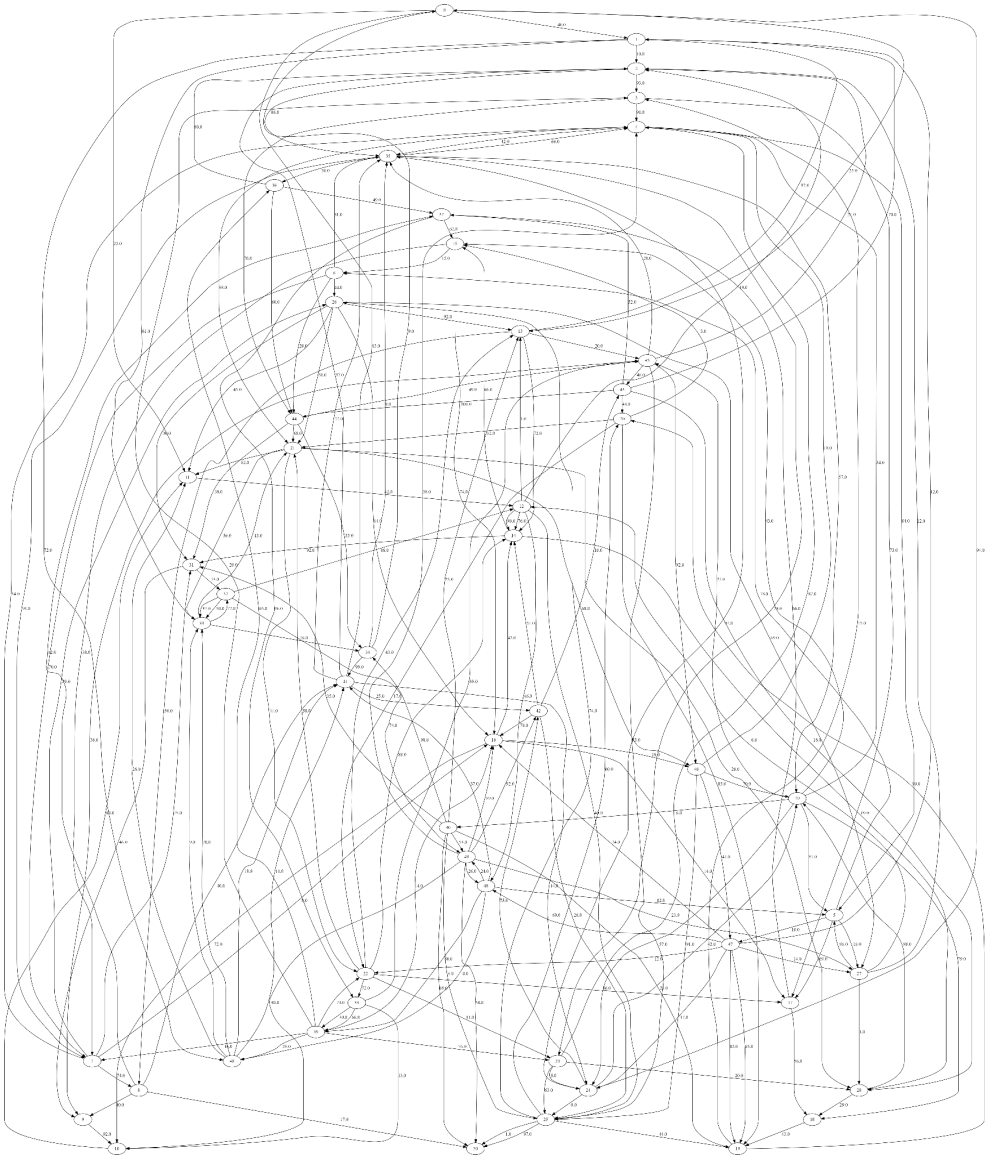
Dans l'algorithme du Point fixe, à chaque itération on va parcourir tous les sommets du graphe pour calculer les chemins minimums.

Alors que dans l’algorithme de Dijkstra, à chaque itération on va sélectionner un seul sommet et calculer la longueur des arcs adjacents.

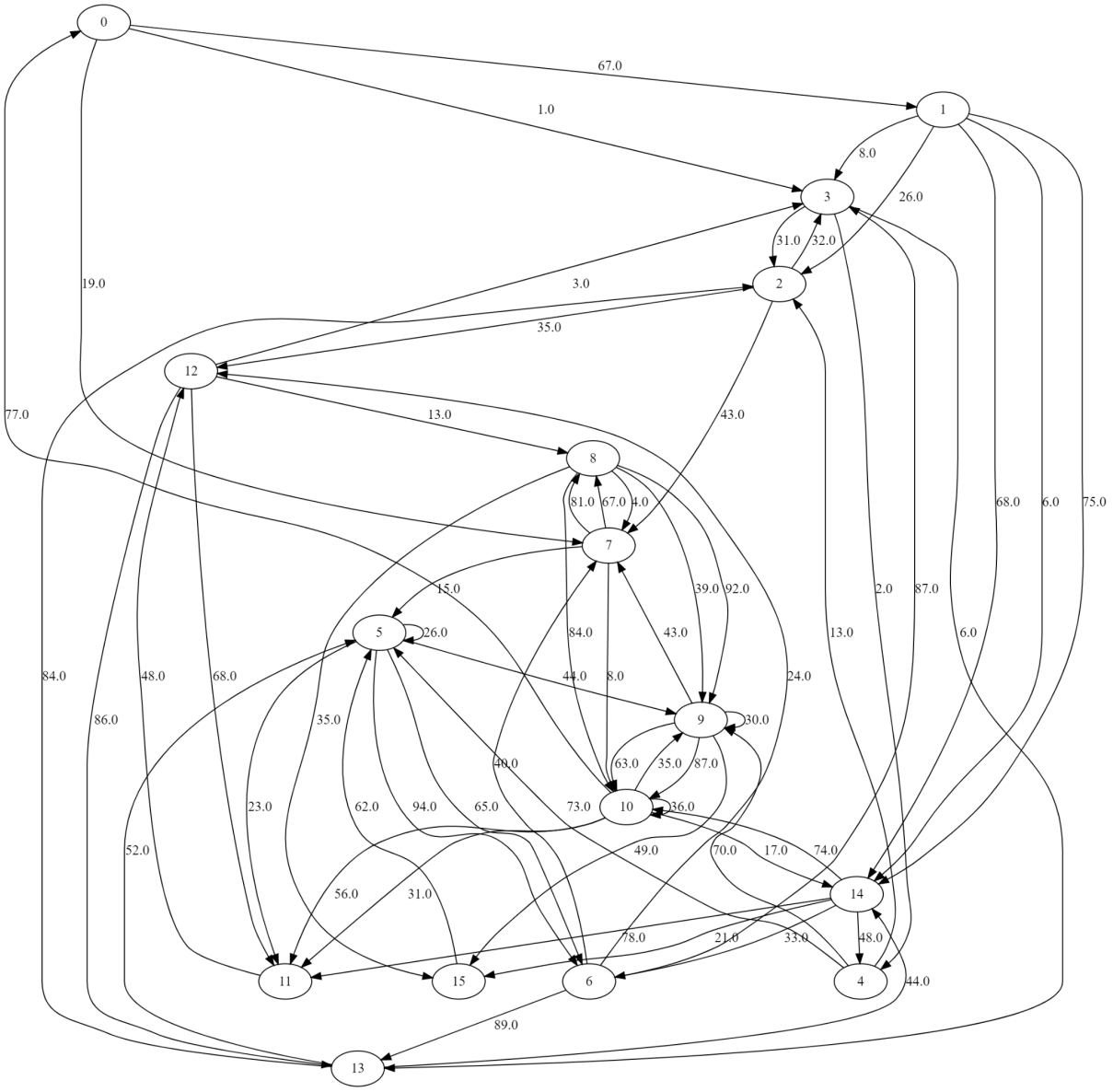
Question 22 :
On remarque que la méthode de Dijkstra doit faire plus d’itération que la méthode du point fixe de Bellman Ford.

Question 25 :

50 noeuds

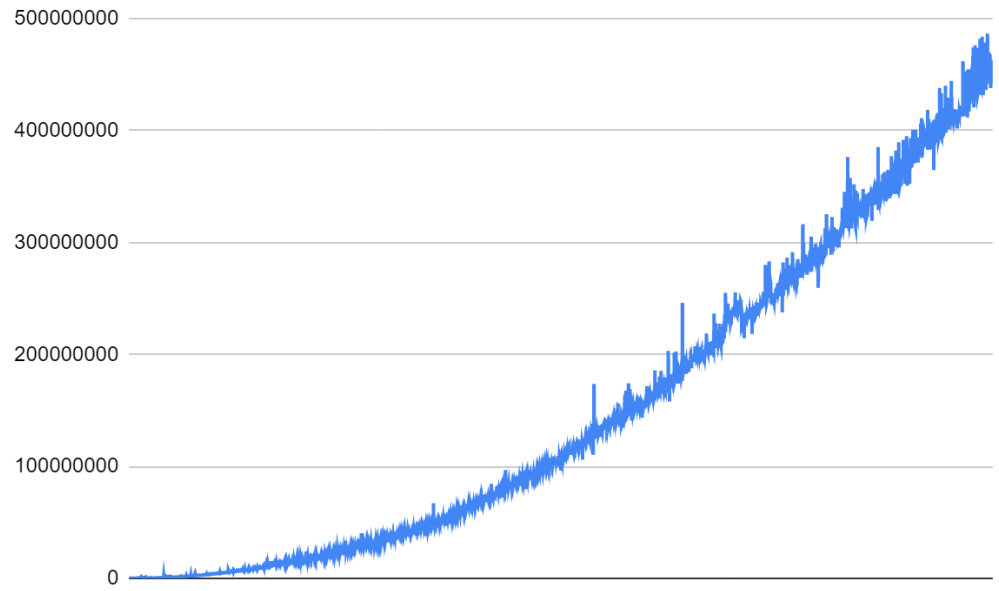


15 noeuds :

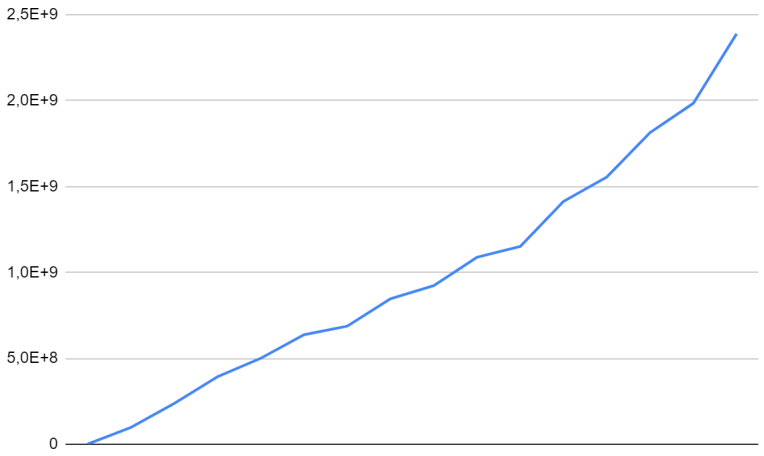


on est parti sur un format de génération avec nombre d’arc = 3*nombre de nœuds.

Question 26 :



La différence de temps entre les deux algorithmes, on voit que dijkstra est de plus en plus efficace en fonction du nombre de nœuds sachant que le calcul normal de performance se fait aussi en fonction du nombre d'arc et du nombre de nœuds.
Nous avons pris la liberté pour ce graphe de donner un nombre d'arc non aléatoire (3n) car nous avons remarqué que ceux donnés dans le sujet ont aussi été générés avec ($n^2/4$).
Donc nous avons regardé l'efficacité en fonction du nombre d'arc pour un même nombre de noeuds :



Ci-dessus l'évolution de l'écart de temps entre les deux algorithmes pour un nombre d'arc évoluant et un nombre de nœuds constant.

Question 27 :

D’après la courbe de 5000 tests, on peut dire que plus le nombre de nœuds augmente, plus l’écart est important. On peut peut-être faire ressembler la courbe du graphique avec une courbe logarithmique.
Le ratio entre les deux algorithmes dépend du nombre de nœuds et d’arc.

Question 28 :

Nous avons conclu que l'algorithme de Dijkstra est de plus en plus efficace par rapport à BellmanFord au fur et à mesure que le nombre de nœuds et d’arc augmente.

V. Labyrinthe

Pour cette partie, nous avons décidé de créer la méthode voisinsValides(int x, int y) qui renvoie la liste de la position de chaque cases adjacentes qui ne sont ni des murs, ni hors de la bordure. Cette méthode est utilisée pour genererGraphe de la classe Labyrinthe et la classe GrapheLabyrinthe. Ensuite, nous avons créé la classe MainLaby pour aplliquer l’algorithme de Dijkstra sur les graphes générés par les classes Labyrinthe et GrapheLabyrinthe.