Bauby Gaspard S2A
Bernard Julien

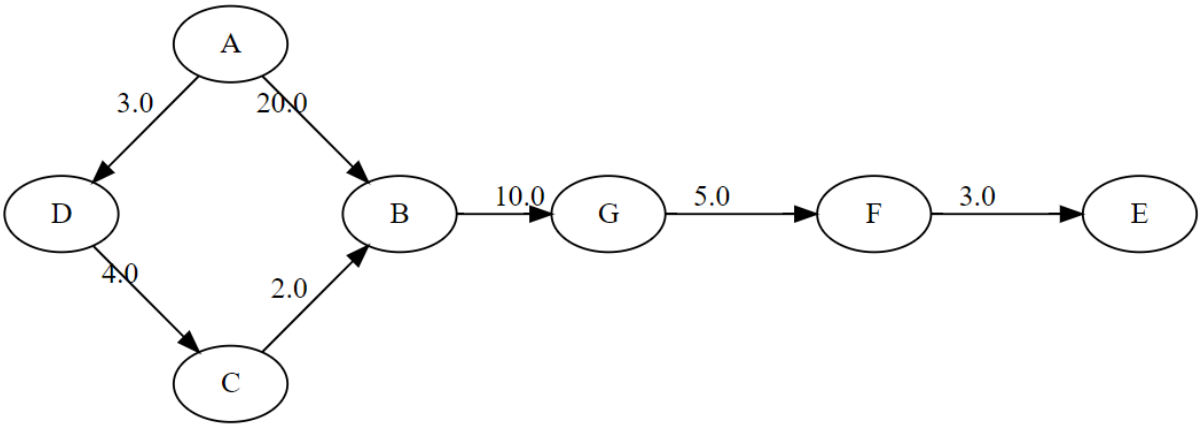
SAE 2.2 - Exploration Algorithmique
Recherche de plus court chemin dans un graphe

I. Représentation d'un graphe

Classes ajoutées:
Classe Noeud
Classe Arc
Interface Graphe
Classe GrapheListe
Main MainGrapheListe
Dans la classe testGraphe :

- testAffichageGraphe : verification de la bonne construction du graphe et de son affichage
- testAffichageGraphviz : verification de la bonne construction du graphe et de son affichage au format graphviz.
- testConstructeurFichier : Verification de la construction du graphe à l'aide d'un nom de fichier.

Question 10:



II. Calcul du plus court chemin par point fixe

Classes ajoutées:
L'algorithme Bellman Ford
Classe Valeur
Main MainBellmanFord
Différents test unitaires

III. Calcul du meilleur chemin par Dijkstra:

Classes ajoutées:
classe Dijkstra

Différents tests unitaires
Main MainDijkstra

IV. Validation et expérimentation:

Question 21 : Afficher le contenu de l'objet valeur après chaque itération de chacun des algorithmes. Expliquer la différence de comportement entre les deux algorithmes.

Algo de Bellman Ford	Algo de Dijkstra
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 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 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	A -> V:0.0 p:null B -> V:20.0 p:A C -> V:Infinity p:null D -> V:Infinity p:null E -> V:Infinity p:null F -> V:Infinity p:null G -> V:Infinity p:null A -> V:0.0 p:null B -> V:20.0 p:A C -> V:Infinity p:null D -> V:3.0 p:A E -> V:Infinity p:null F -> V:Infinity p:null G -> V:Infinity p:null 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:Infinity p:null F -> V:Infinity p:null G -> V:Infinity p:null 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:Infinity p:null F -> V:Infinity p:null G -> V:Infinity p:null 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:Infinity p:null F -> V:Infinity p:null G -> V:19.0 p:B 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:Infinity p:null F -> V:24.0 p:G G -> V:19.0 p:B 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

L'algorithme de Dijkstra fait plus d'itération avec le graphe boucle que l'algorithme de Bellman Ford. En effet le premier valide les valeurs une par une, à chaque itération il en valide une seule, tandis que le second essaye à chaque fois d'en valider le plus possible, bien souvent plusieurs d'un coup.

Question 22 : Tirer des conclusions à partir de vos observations.
L'algorithme de Bellman Ford semble plus rapide pour les petits graphes, il aura moins d'itération, mais cette observation pourrait s'inverser pour les grands graphes.

Question 23 : Lequel des deux algorithmes semble le plus efficace et pourquoi selon vous ?
L'algorithme de Dijkstra semble plus rapide puisqu'il ne recalcule pas les valeurs dont les valeurs sont certaines, tandis que l'algorithme de Bellman Ford calcule chaque valeur a chaque itération, ce qui demandera plus de ressources et plus de temps. En effet, il a fallu 66571 ms a l'algorithme de Bellman Ford pour résoudre tous les graphes, tandis qu'il n'aura fallu que 30660 ms pour l'algorithme de Dijkstra. L'algorithme de Dijkstra a mis 2 fois moins de temps pour résoudre tous les graphe, il semble donc plus optimisé/ plus efficace que l'algorithme de Bellman Ford

Question 26: A l'aide de votre générateur de graphes, comparez les résultats obtenus en fonction de la taille du graphe. Quel est l'algorithme le plus efficace selon ces résultats ?

Nbr Graphes	NbArc	NbNoeuds	Belman	Dij	Ratio Belman/Dij	Ratio Arc/Nœuds
100	10	10	1333	582	2,29	1
100	20	10	1161	454	2,56	2
100	40	10	2407	702	3,43	4
100	60	10	2358	555	4,25	6
100	50	50	4210	1866	2,26	1
100	100	50	6078	2767	2,20	2
100	200	50	8313	3628	2,29	4
100	300	50	9781	3234	3,02	6
100	250	250	17965	31231	0,58	1
100	500	250	61582	31642	1,95	2
100	1000	250	106649	30524	3,49	4
100	1500	250	138489	34161	4,05	6
100	500	500	62819	101015	0,62	1
100	1000	500	190990	89518	2,13	2
100	2000	500	376431	109928	3,42	4
100	3000	500	441234	133700	3,30	6
100	1000	1000	181872	442167	0,41	1
100	2000	1000	775785	332814	2,33	2
100	4000	1000	1233304	422153	2,92	4
100	6000	1000	2596854	529510	4,90	6

On constate que plus le graphe est grand (nombre de nœuds et d'arc important), plus l'algorithme de Dijkstra est efficace.
Cependant le ratio de temps n'évolue que peu lorsque le nombre de noeud et arc augmente du même nombre/ proportionnellement (500 arc, 500 noeuds → 1000 arcs, 1000 noeuds / 1000 arc, 500 noeuds → 2000 arc, 1000 noeuds): le ratio 0,62 est proche de 0,41 et 0,58, le

ratio 2,13 est proche de 2,33 et 1,95.Ce dernier augmente légèrement lorsque la taille du graphe augmente.

Question 27: Quel est le ratio de performance entre les deux algorithmes en fonction du nombre de nœuds ? Ce rapport est-il constant ou dépend-il du nombre de nœuds ?

On constate que pour des graphes de plus de 250 nœuds, plus le nombre d'arc est important, plus le ratio Bellman/ Dijkstra croit en faveur de l'algorithme Dijkstra. Cependant, on constate que ce ratio s'inverse lorsque le nombre d'arcs est faible par rapport au nombre de nœuds.

Question 28: Quelle conclusion pouvez-vous tirer de cette étude ?

Suite à cette étude, on peut conclure que l'algorithme de Dijkstra est plus efficace en condition réelle puisque les graphes seront très probablement conséquent: nombre de noeuds et nombre d'arc important, alors l'algorithme de Dijkstra nécessite moins de temps pour réaliser les résolution de graphes. Cependant si on sait à l'avance que le nombre d'arc ne sera jamais important alors il faudra mieux utiliser l'algorithme de Bellman.

Question 30-31

Validation labyrinthe

Première solution

Écriture de la méthode genererGraphe dans la classe Labyrinthe.
Calcule d'un chemin le plus court à l'aide de la classe Valeur, et de la méthode calculerChemin de la classe valeur.

Pour la deuxième solution, nous avons créé la classe GrapheLabyrinthe qui implémente l'interface graphe. Cette classe possède un constructeur qui prend en paramètre un labyrinthe.
Nous pouvons donc par la suite calculer le chemin le plus court avec la méthode cheminCourt qui utilise la classe Valeur fournie.

Conclusion

Appris:
Nous avons découvert un nouvel algorithme qui permet de résoudre des graphes, que nous avons implémenté de différentes manières afin de pouvoir les utiliser au final sur des labyrinthes afin d'avoir un résultat concret. Cela nous permet de calculer les chemins minimaux d'un point à un autre dans le labyrinthe.
Les difficultés:
Nous avons eu quelques difficultés à faire l'algorithme de BellmanFord, mais nous n'avons pas eu de difficultés à le retranscrire en langage java par la suite.
Bilan:
Nous avons découvert l'algorithme de Dijkstra qui se trouve être beaucoup plus rapide dans la plupart des cas des graphiques à résoudre,