INF2010
Structures de données et algorithmes
TP05 : Graphes

Automne 2019

Objectifs du TP:
Le but de ce TP est de retrouver s'initier aux notion de la théorie des graphes. Vous devez implémenter un graphe et utiliser l'algorithme de Dijkstra pour trouver le chemin le plus court entre deux nœuds d'un graphe.

Indication :

- Les sources qui vous sont remises incluent cinq classes: *Main.java*, *Node.java*, *Edge.java*, *Graph.java*, et *Dijkstra.java*. Main permet de construire un graph (*Graph.java*) et de tester l'algorithme de Dijkstra (*Dijkstra.java*), un graphe est un ensemble de nœuds (*Node.java*) et d'arcs (*Edges.java*).
- Vous avez le droit d'ajouter des attributs, mais pas de changer les signatures des classes ou des méthodes.
- Notez que votre code doit implémenter l'algorithme de Dijkstra en général, pas juste pour l'exemple de TP (d'autres graphes seront testés durant la correction).

1 Première partie : Création de graphe

Étant donnés un graphe $G(X, E, v)$

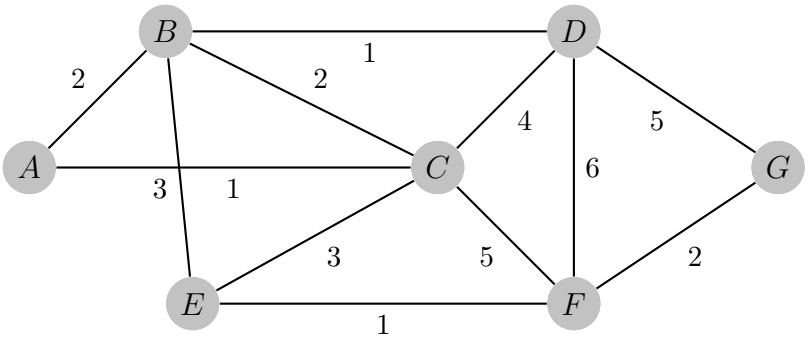

Figure 1: Graphe

A	→	B	C			
B	→	A	D	C	E	
C	→	A	B	D	E	F
D	→	B	C	F	G	
E	→	B	C	F		
F	→	C	D	E	G	
G	→	D	F			

Figure 2: Listes d'adjacence

1.1 Question :

Dans la classe « Graph », complétez le constructeur. Complétez la méthode « getEdgesGoingFrom » qui permet de retourner tous les chemins allant d'un nœud « source ». D'une manière similaire complétez la méthode « getEdgesGoingTo » qui permet de trouver toutles chemins allant vers un node « Dest ». Dans la fonction principale « main » de la classe « Main.java », utilisez l'objet graph et les classes « Graph », « Node », et « Edge » pour construire le graphe que représenté dans la Figure 1

2 Seconde partie :Implémentation de l'algorithme de Dijkstra

E. W. Dijkstra (1930-2002) a proposé en 1959 un algorithme qui permet de déterminer le plus court chemin entre deux sommets d'un graphe G connexe pondéré dont le poids lié aux arêtes est positif.

2.1 Principe de l'algorithme de Dijkstra

L'algorithme Dijkstra consiste à trouver le chemin le plus court entre un nœud de départ et un nœud d'arrivée. Pour implémenter l'algorithme, il faut suivre plusieurs itérations, dont chacune consiste à calculer la distance entre le nœud de départ et ses successeurs, ensuite ces derniers et leurs successeurs, en utilisant le Tableau 1.

1. **Itération 1 :** Initialiser le nœud de départ à une distance égale à 0 (pour aller du nœud de départ au même nœud, il faut une parcourir une distance égale à 0).
2. **Itération i :** Trouver la distance minimale de l'itération qui précède.

2.2 Exemple :

On désire exécuter l'algorithme de Dijkstra sur le graphe de la Figure 1

- **Itération 1) :** On initialise A à 0.
- **Itération 2) :** Dans le tableau 1, la distance minimale de l'itération qui précède est 0. Cette distance minimale est utilisée pour arriver au nœud A.

  Ensuite, il faut chercher les successeurs du nœud sélectionné (les successeurs de A sont B et C), et donner à chacun d'eux une distance, qui est la distance entre le nœud sélectionné et chacun des successeurs plus la distance nécessaire pour arriver au nœud sélectionné. Donc il faut compléter les valeurs de B et C avec la distance minimale sélectionnée précédemment (qui est 0), plus la distance entre le nœud et ses successeurs (qui sont 2 et 1 pour B et C respectivement). Il faut aussi noter dans le tableau les nœuds sources, par exemple, pour arriver au nœud C de l'itération 2, il faut une distance égale à 1, arrivant de A. Si la distance calculée pour arriver à un nœud donné est inférieure à la distance pour arriver au même nœud dans l'itération qui précède, alors il faut sélectionner le plus petit des deux.

- **Itération 3) :** on a besoin d'une distance de 3 pour arriver au nœud B à partir de C, mais on a besoin juste d'une distance de 2 arrivant de A pour atteindre le nœud B, donc il faut sélectionner la plus petite valeur des deux valeurs, donc « 2A ».

- **Itération 4) :** D'une manière similaire, il faut choisir le minimum entre 4C et 5B. Notez qu'une fois qu'on sélectionne une valeur minimale d'une colonne, on ne considère plus le nœud dans les itérations suivantes.

- **Itération 5, 6 et 7.**

  Remarques : à partir de l'itération 3, on ne considère plus le nœud C. On ne considère plus la colonne E après l'itération 5.

Le tableau final construit pour le graphe de la Figure 1 sera :

Itération	A	B	C	D	E	F	G
1	0A	.	.	.	.	.	.
2		2A	1A	.	.	.	.
3		2A/3C		5C	4C	6C	.
4				5C/3B	4C/5B	6C	.
5					4C	6C/9D	8D
6						6C/5E	8D
7							8D/7F

Table 1: L'exécution de l'algorithme de Dijkstra avec un tableau

2.3 Questions :

En suivant cet algorithme:

1. Compléter la méthode « findPath » de la classe « *Dijkstra.java* », qui consiste à remplir la table « dijkstraTable » dans les lignes correspondent aux itérations, et les colonnes représentent les nœuds.

2. Implémentez la méthode « getMinimum » qui retourne le minimum de deux arcs en se basant sur ses deux distances, faites attention aux objets null.

3. Complétez la fonction main en spécifiant le nœud « A » comme nœud source, et le nœud « G » comme destination.

Stockez le minimum de chaque itération dans la pile « path », dont le but est de l'utiliser pour afficher le chemin le plus court (Partie 3). Il faut empiler le minimum de chaque itération, en s'assurant que la destination de l'arc qui est en top de la pile, est bien la source du minimum que vous avez sélectionné. C'est à dire, il faut dépiler jusqu'à ce que cette condition soit satisfaite et ainsi insérer l'arc minimum. Par exemple, dans l'itération 1 et 2, il faut alimenter la pile avec « 0A » et « 1A » respectivement. Mais, dans l'itération 3 dont il faut insérer « 2A », il faut d'abord dépiler « 1A » vu que la destination de « 1A » est C, tant dis que la source de « 2A » est A (pas C). Après la dépilation, le nœud en top de la pile sera « 0A » dont la destination est « A », qui est aussi la source de « 2A ».

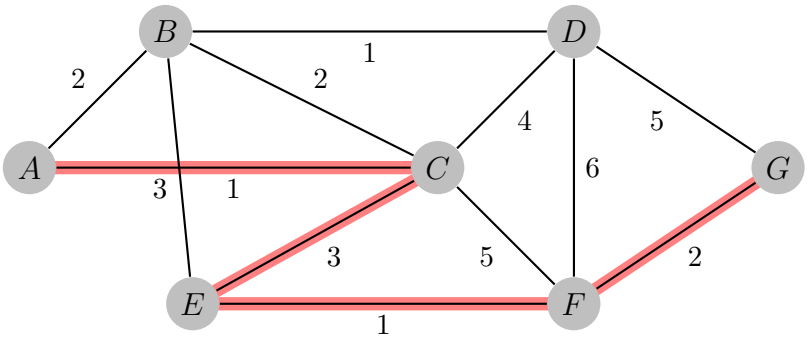

Figure 3: Plus court chemin entre deux sommets (A,G)

2.4 Questions :

Complétez la méthode « showTable » qui affiche la table « dijkstraTable » tel qu'il est montré dans l'exemple suivant. Notez que vous devez afficher juste le minimum de chaque case, par exemple, afficher juste 2A pour la colonne B de la ligne 3. (Pas besoin d'afficher la colonne « Itération »).

3 Troisième partie : Afficher le chemin le plus court

Une fois que vous avez complété le tableau, il faut afficher le chemin le plus court. Pour cela, il faut utiliser la pile « path » que vous avez alimentée dans la partie 2 .

3.1 Question :

Complétez la méthode « printShortPath » et modifiez le main pour afficher le plus court chemin entre A et G.

4 Instructions pour la remise

Le travail doit être remis via Moodle :

- 28 novembre avant 23h59 pour le groupe 01
- 21 novembre avant 23h59 pour le groupe 02
- 27 novembre avant 23h59 pour le groupe 03
- 20 novembre avant 23h59 pour le groupe 04
- 01 décembre avant 23h59 pour le groupe 05

Veuillez envoyer dans une archive de type $*$.zip qui portera le nom inf2010_lab5_MatriculeX_ MatriculeY (MatriculeX $<$ MatriculeY) :

- Vos fichiers .java

Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard.

4.0.1 Avertissement :

Pour ceux qui voudraient déposer leur laboratoire sur GitHub, assurez-vous que vos répertoires soient en mode **privée** afin d'éviter la copie et l'utilisation non autorisée de vos travaux.

4.1 Barème de correction

4 pts: Première partie

12 pts: Seconde partie

4 pts: Troisième partie