Conservatoire National des Arts et Métiers
École SITI - Département Informatique

Année 2013–2014

UE NFP 136 (VARI 2)

Séances de TP 7 à 9
Mini-projet : l'algorithme de Kruskal

Le but de ce mini-projet est d'implémenter en JAVA l'algorithme de Kruskal. Le programme JAVA ainsi obtenu sera capable de déterminer un arbre couvrant de poids minimum d'un graphe (décrit sous la forme d'un fichier texte), dont les arêtes sont pondérées par des entiers positifs. Ce document a pour objectif de vous guider dans les différentes étapes de la réalisation de ce mini-projet, et est à lire attentivement. Pour une description détaillée de l'algorithme de Kruskal, se référer au cours et à l'ED correspondants.

Exercice 1 : classe Arete

On supposera que les n sommets du graphe considéré sont représentés par les n entiers de 1 à n. On se propose, dans le cadre de ce mini-projet, de définir une classe JAVA **Arete**, qui permettra de représenter les arêtes d'un graphe pondéré. Ainsi, chaque objet de cette classe sera décrit par trois entiers (variables d'instance) : le premier représente le numéro du sommet qui constitue la première extrémité de l'arête, le deuxième représente le numéro du sommet qui constitue la deuxième extrémité de l'arête, et le dernier représente le poids de l'arête. De cette façon-là, on peut très facilement représenter, par exemple, une arête de poids 56 entre les sommets 3 et 19, à l'aide de trois informations (3, 19, et 56).

On vous demande d'implémenter une telle classe JAVA. On n'oubliera pas de définir des méthodes permettant de modifier les valeurs des trois variables d'instance et d'y accéder : d'abord, un constructeur permettant d'initialiser les valeurs des trois variables d'instance, puis trois méthodes permettant d'accéder aux valeurs de ces trois variables, et enfin trois méthodes pour modifier leurs valeurs. On pourra éventuellement écrire d'autres méthodes (par exemple, une méthode permettant de modifier les valeurs de ces trois variables à la fois). Une fois la classe **Arete** définie, on pourra représenter un graphe sous la forme d'un tableau d'objets de cette classe.

Exercice 2 : lecture du graphe à traiter dans un fichier texte

Les graphes à traiter par l'algorithme seront stockés dans des fichiers texte qui vous seront fournis. Avant de pouvoir appliquer l'algorithme à un graphe,

il faut donc commencer par lire les données dans le fichier correspondant, et les stocker ensuite dans une structure facilement "exploitable" (ici, un tableau d'objets de la classe **Arete**).

On vous demande donc ici d'implémenter une méthode JAVA permettant d'effectuer cette opération. Pour cela, il faut commencer par lire les données du fichier. Les fichiers représentant les graphes auront le format suivant :

- La première ligne contiendra une seule information, le nombre n de sommets du graphe.
- La deuxième ligne contiendra également une seule information, le nombre m d'arêtes du graphe.
- Chacune des m lignes suivantes contiendra la description d'une arête du graphe : sa première extrémité, sa deuxième extrémité, et son poids.

Sur chacune des m dernières lignes, ces trois informations sont séparées par des virgules. Un exemple de graphe à quatre sommets et cinq arêtes est donné par le fichier texte suivant, de nom "toto.txt" :

```
4
5
1,2,10
1,3,41
1,4,3
2,3,15
3,4,7
```

Le graphe ainsi décrit contient cinq arêtes : une arête entre les sommets 1 et 2, une arête entre les sommets 1 et 3, une arête entre les sommets 1 et 4, une arête entre les sommets 2 et 3, et une arête entre les sommets 3 et 4. Ces arêtes ont pour poids respectifs 10, 41, 3, 15, et 7.

Pour aller lire des informations dans un fichier, il faudra utiliser un objet de la classe **BufferedReader**. Ainsi, la ligne suivante initialise un objet *lecteur* de cette classe, qui permet d'accéder aux informations stockées dans le fichier de nom "toto.txt" :

```
BufferedReader lecteur = new BufferedReader(new FileReader("toto.txt"));
```

En appelant la méthode *readLine* (qui renvoie un objet de la classe **String**) sur cet objet, on lira le contenu de la ligne suivante du fichier de nom "toto.txt" (la lecture d'un fichier étant séquentielle). Ainsi, après le premier appel à cette méthode, la variable *ligne* contiendra la valeur "4" :

```
String ligne=lecteur.readLine();
```

Après un deuxième appel, la variable *ligne* contiendra la valeur "5" :

```
ligne=lecteur.readLine();
```

Enfin, après un troisième appel, identique au deuxième, la variable *ligne* contiendra la valeur "1,2,10" :

```
ligne=lecteur.readLine();
```

Lorsque la fin du fichier est atteinte et qu'il n'y a plus rien à lire dedans, l'appel `lecteur.readLine()` renvoie la valeur `null`. La construction d'un objet de la classe **BufferedReader** est susceptible de générer une exception (si le fichier de nom "toto.txt" n'existe pas, par exemple), et il serait donc préférable de gérer toutes ces instructions dans un bloc `try{...} catch(...){...}`. Enfin, après utilisation, cet objet *lecteur* doit être fermé, à l'aide de l'instruction suivante :

```
lecteur.close();
```

Une fois les données d'une ligne récupérées, il faudra créer un objet de la classe **Arete** associé. Pour cela, on vous rappelle l'existence de :

- la méthode *split* de la classe **String**, qui permet de "découper" l'objet de la classe **String** sur lequel la méthode est appelée en fonction du séparateur passé en paramètre, et qui stocke les objets de la classe **String** qui résultent de ce découpage dans un tableau.
- la méthode de classe *parseInt()* de la classe **Integer**, qui permet de convertir en entier (variable de type **int**) l'objet de la classe **String** passé en paramètre (par exemple, "2" sera converti en l'entier 2).

Pour plus d'information sur ces deux méthodes, se référer à l'énoncé du premier TP.

Exercice 3 : tri des arêtes

On rappelle qu'une étape préliminaire dans l'algorithme de Kruskal est de trier les arêtes du graphe par ordre de poids croissants. Implémenter (en réutilisant les algorithmes implémentés durant les séances de TP précédentes) une méthode permettant de trier, par ordre de poids croissants, un tableau d'objets de la classe **Arete**, qui contiendra toutes les arêtes du graphe initial. Un petit bonus sera octroyé si cette méthode est implémentée à l'aide de l'algorithme du tri par tas, ou à l'aide de l'algorithme du tri fusion.

Exercice 4 : détection des cycles

La dernière étape dans l'implémentation de l'algorithme de Kruskal consiste à parcourir les arêtes dans l'ordre des poids croissants (obtenu à l'aide de la

méthode de l'exercice précédent), et à ajouter chaque arête à la solution en construction si elle ne crée pas de cycle avec les arêtes déjà sélectionnées.

Écrire une méthode qui, étant donné un tableau d'objets de la classe **Arete** triés par ordre de poids croissants, parcourt ce tableau et sélectionne les arêtes de l'arbre couvrant de poids minimum. Cela pourra se faire, par exemple, en construisant au fur et à mesure un nouveau tableau contenant les arêtes sélectionnées. Pour détecter les cycles éventuels créés par l'ajout d'une arête, on pourra utiliser l'approche vue en ED (un tableau stockant, pour chaque sommet, l'identifiant de la composante connnexe à laquelle appartient ce sommet). On n'oubliera pas non plus de gérer le fait qu'ajouter une arête à la solution en construction entraîne la fusion des deux composantes connexes contenant ces deux extrémités, ni de vérifier que la méthode implémentée prend bien en compte le cas où le graphe n'est pas connexe (la méthode doit s'arrêter dans ce cas, et l'indiquer par le biais de la valeur de retour).

Exercice 5 : dernières touches

Pour compléter le projet, on écrira un programme principal qui récupérera en paramètre, au moment de l'appel du programme, le nom du fichier contenant les arêtes du graphe considéré. Puis, il ira lire les arêtes du graphe décrit dans ce fichier, et appliquera l'algorithme de Kruskal dessus. Si le graphe n'est pas connexe, il faudra en informer l'utilisateur. Dans le cas contraire, il faudra afficher la liste des arêtes de l'arbre couvrant de poids minimum, ainsi que son poids total.

Lors du rendu du projet, il faudra bien évidemment en fournir le code source commenté (fichiers *.java*), le bytecode (fichiers *.class*), ainsi que les arbres couvrants de poids minimum (résultants de l'application de votre programme), et leur poids, pour les graphes dont les fichiers seront fournis.