TP Graphes

I Première séance : plusieurs représentations d'un graphe.

Pour implémenter les graphes orientés, on considère la classe **`Graphe`** suivante :

```
public class Graphe{

    int NS; //nombre de sommets
    int[][] dico; //dico qui contient les successeurs

}
```

Attention, les sommets seront nommés $1, 2, \ldots, NS$. Vous testerez vos méthodes avec le graphe donné en TD.

Q.1) - Écrire un constructeur qui prend en paramètre le dictionnaire du graphe.

Q.2) - Écrire une méthode qui prend en paramètre un sommet et affiche sur la sortie standard les successeurs de ce sommet, sous la forme `1 : 2 4 5`.

Q.3) - Écrire une méthode `affiche()` qui affiche sur la sortie standard le dictionnaire du graphe.

Q.4) - Écrire une méthode qui prend en paramètre un sommet et affiche sur la sortie standard les prédécesseurs de ce sommet.

Q.5) - Écrire une méthode qui calcule le nombre d'arcs et initialise un nouvel attribut `NA` représentant le nombre d'arcs.

Q.6) - On ajoute comme attributs deux tableaux `orig` et `extr` pour les origines et les extrémités des arcs. Écrire une méthode qui initialise ces nouveaux attributs à partir du tableau `dico` représentant le dictionnaire.

Q.7) - Compléter votre constructeur et votre méthode `affiche()`.

Q.8) - Écrire un constructeur `Graphe(`**`int`** `NS, `**`int[]`** `orig, `**`int[]`** `extr)` qui prend en paramètres le nombre de sommets et les tableaux représentants les arcs et qui initialise également le dictionnaire (pour cela, on pourra passer par la matrice d'incidence).

II Seconde séance : parcours dans un graphe orienté.

Le but de cette séance est de coder en JAVA l'algorithme 1 étudié en TD.

Q.1) - Ajouter un tableau de booléens `marque` à la classe **`Graphe`** comme attribut.

Q.2) - Écrire une méthode qui initialise `marque` en un tableau de taille `NS` remplit de `false`.

Q.3) - Écrire une méthode qui marque un sommet passé en paramètres.

Q.4) - Écrire une méthode **`void`** `parcours(`**`int`** `deb)` qui fait le parcours décrit par l'algorithme 1 du TD :
- On pourra utiliser un objet de type **`Vector<Integer>`** pour représenter la file, ainsi que les méthodes suivantes de la classe **`Vector`** : `isEmpty()`, `firstElement()`, `remove(`**`int`** `index)` et `add(`**`Integer`** `t)`.
- On affichera sur la sortie standard la suite des sommets parcourus.

Q.5) - Ajouter à votre classe un tableau `distance` de `double` pour représenter des distances.

Q.6) - Écrire une méthode `initDistance(int deb)` qui initialise `distance` en un tableau de taille `NS`, met 0 à l'indice `deb-1` de ce tableau et `Double.POSITIVE_INFINITY` aux autres indices.

Q.7) - À l'aide du tableau `distance`, compléter votre méthode `parcours`, afin que celle-ci affiche au fur et à mesure la distance des sommets parcourus au sommet de départ `deb`.

Q.8) - À l'aide d'un tableau d'entiers `peres` et éventuellement d'autres méthodes auxiliaires, compléter votre méthode `parcours` afin que celle-ci affiche également au fur et à mesure les chemin de longueur minimale du sommet de départ `deb` aux sommets parcourus.

III Troisième séance : distances dans un graphe orienté valué.

L'objectif de cette partie est de coder l'algorithme de Ford étudié en TD. On testera cet algorithme sur le graphe valué donné en TP.

Q.1) - Créer une classe fille **`GrapheValue`** de la classe **`Graphe`**.

Q.2) - Écrire un constructeur `Graphe(int NS, int[] orig, int[] extr, double[] w)` qui prend en paramètres le nombre de sommets et les tableaux représentants les origines, les extrémités et les longueurs des arcs.

Q.3) - Coder dans une méthode `ford(int s)` l'algorithme de Ford. On pourra utiliser le tableau `distance` de la classe mère.

Voici une description de l'algorithme :

Donnée : s : le sommet source
début
  initDistances(s) // initialise le tableau distance (cf partie précédente),
  continuer ← VRAI
  tant que *continuer* **faire**
    continuer ← FAUX // a priori c'est le dernier tour
    pour *chaque arc* $(t, v) \in U$ **faire**
      si $d_s(t) + w(t, v) < d_s(v)$ **alors**
        $d_s(v) \leftarrow d_s(t) + w(t, v)$
        continuer ← VRAI //il y a eu un changement
// il faudra refaire un parcours des arcs
fin

Algorithme 2 : Ford(Sommet s)

Q.4) - Écrire une méthode qui affiche sur la sortie standard les distances des sommets au sommet passé en paramètres.

Q.5) - Compléter vos méthodes pour afficher également sur la sortie standard un plus court chemin vers chaque sommet accessible à partir du sommet passé en paramètres.