TP 4 Algorithmique et structures des données
Implémentation des Graphes et Algorithme de Dijkstra

Le but de ce TP est coder l'algorithme de Dijkstra pour déterminer le chemin plus court dans un graphe orienté ou pas, lorsque les poids des arcs sont positifs.
Nous utiliserons les graphes pondérés représentés par des listes d'adjacences. Vous utiliserez les Classes dans le fichier **ClassGRAPHES_LISTES** (importer ou copier)

Noter:

- les sommets du graphe sont numérotés de 0 à nbsommets-1
- le graphe est de type `typeG`, dont les valeurs sont `ORIENTED` ou `NON_ORIENTED`
- l'attribut `listadj` est le **tableau/liste** des listes d'adjacences : `mygraphe.listadj[i]` contient la liste d'adjacence du sommet `i` dans le graphe `mygraphe`

A. Adaptation ou refonte de files de priorité (TP2).

L'algorithme de Dijkstra utilise *une file de priorite de type min* qui doit fournier aussi la méthode *update*: cette méthode mets à jour la priorité d'un élément identifié par une clé, si la nouvelle prioirité est *meilleure* (= inférieure dans ce cas) de l'ancienne.

Vous avez deux solutions au choix:

Solution1: facile, mais *inefficace*: vous ***ajoutez*** aux méthodes du TP2 une méthode ***update*** pour mettre à jour la priorité d'un élément d'une file identifié par une clé. Solution inefficace: O(n), étant n la taille de la file, car il faut chercher dans toute la file la ***position*** d'un élément avec une certaine clé.

Solution2: plus difficile, mais *efficace*: redéfinir la classe File en ajoutant un tableau/liste **`position`** pour *mémoriser* la *position* d'un élément dans le *tas*: un indice de **`position`** est une clé et sa valeur est la position. Cette solution présuppose que la file ne gère qu'un nombre maximal prédefini d'élements et que les clés soient des entiers entre 0 et une valeur max (une restriction acceptable dans ce cas: le nombre de sommets d'un graphe est connu d'avance).

Exemple: si `tas[7] = (11, 200)`, où 11 est la clé et 200 la priorité, alors il sera : `position[11] = 7`.

Si une clé k n'est pas présente dans le tas, il sera `position[k] = -1` (une position inexistente). Les valeurs de `position` doivent être modifiées lors des opérations d'insertion, extraction du min et update. Avec cet implémentation, *on peut accéder directement à la position de l'élément à mettre à jour en connaissant sa clé, sans le chercher* (complexité O(logn)).

Un conseil: vous pouvez coder l'algorithme de Dijkstra d'abord avec la Solution 1 et si vous avez le temps essayer la solution 2.

B. Implémenter l'algorithme de Dijkstra :

La fonction:

```
dijkstra(graphe, source,...,autres paramètres)
```

elle implémente l'algorithme de Dijkstra .

- La fonction prend en entrée un graphe orienté ou non-orienté dans lequel les poids des arcs sont non-négatif (condition *qui n'est pas vérifiée* par l'algorithme)
- la fonction calcule *le poids d'un plus court chemin* depuis `source` à chaque sommet et le *parent* de chaque sommet dans ce chemin.
- Ces informations sont enregistrées par un ou plusieurs paramètres passés à la fonction.