TP Option Info MP/MP* : Graphes

On rappelle les deux implémentations possibles des graphes non pondérés (orientés ou non):
- Par une matrice d'adjacence `m` : `m.(i).(j)` est un booléen indiquant si `i` et `j` sont connectés (par convention `i` et `i` ne sont pas connectés).
- Par un tableau `g` de listes d'adjacence : `g.(i)` contient la liste des voisins de `i`.

On verra dans l'exercice 3 une implémentation des graphes pondérés.

Exercice 1 : Passage d'une implémentation à l'autre

1. Écrire une fonction `matadj` qui convertit un graphe donné sous forme de listes d'adjacence en matrice d'adjacence. Ecrire de même une fonction `listadj` qui fait le travail inverse.

2. Écrire des fonctions `desorient1` et `desorient2` qui prennent en entrée un graphe orienté (avec les deux implémentations possibles) et qui renvoient sa version non orientée.

Exercice 2 : Parcours

En choisissant l'implémentation la mieux adaptée, écrire des fonctions `largeur` et `profondeur` prenant en entrée un graphe et un sommet initial et qui réalisent le parcours en largeur/profondeur du graphe.

Exercice 3 : Chemin de poids minimal

Dans cette partie on travaille avec des graphes pondérés, on modifie donc nos implémentations:
- Matrice d'adjacence `m` : `m.(i).(j)` vaut le poids de l'arête entre `i` et `j` si elle existe et `max_int` sinon (avec la convention `m.(i).(i)`=0).
- Listes d'adjacence `g` : `g.(i)` contient une liste de couples (sommet, poids).

1. Que se passe-t-il si l'on calcule `max_int +1` ? Pouvez-vous l'expliquer ? Ecrire une fonction `somme i j` qui étend la somme usuelle sur les entiers en intégrant `max_int` avec la convention naturelle.

2. En choisissant l'implémentation la mieux adaptée, écrire une fonction `floyd_warshall` qui réalise l'algorithme de Floyd-Warshall. La fonction devra renvoyer la matrice des distances les plus courtes sans modifier l'entrée et en ne créant qu'une seule matrice intermédiaire. Quelle est sa complexité ?

3. En choisissant l'implémentation la mieux adaptée, écrire une fonction `dijkstra` qui réalise l'algorithme de Dijkstra et renvoyant le tableau des distances au sommet d'indice 0. On utilisera un tableau de booléens pour mémoriser les sommets déjà visités et on pourra faire appel à une fonction auxiliaire récursive qui parcourt la liste d'adjacence et met à jour les différents tableaux. Quelle est sa complexité ?

   Pour les plus motivés: On souhaite améliorer l'algorithme de Dijkstra pour les graphes "peu denses" en utilisant une file de priorité (implémentée par un tas) plutôt qu'un tableau de booléens. Reprendre le cours et les TP sur les tas pour réaliser cette partie. Attention,

les tas seront ici constitués de couples (élément, priorité), la priorité étant la distance actuelle à 0 (qui sera mise à jour). On pourra utiliser les fonctions `fst` et `snd` pour accéder au premier/ deuxième élément du couple. Attention aussi au fait qu'on travaille avec un tas-min et non un tas-max : l'élément en sommet de tas est celui de priorité (distance) minimale.

4. Écrire une fonction `creer_tas n` qui crée un tas de longueur n comportant tous les entiers de 0 à $n-1$ avec des priorités initiales à 0 pour le sommet 0 et `max_int` pour tous les autres.

5. On souhaite écrire les fonctions de modification du tas. On suppose disposer d'un tableau `pos` répertoriant les positions de chaque sommet dans le tas (au début, chaque sommet numéroté i dans le graphe est à la position i dans le tas). Écrire une fonction `swap t i j pos` d'échange des deux éléments `i` et `j` dans le tas, qui met également à jour le tableau `pos`.

6. Écrire les fonctions `monte t k pos` de remontée de l'élément numéroté `k` dans le tas, et `descend t n k pos` de descente de cet élément dans le tas, l'indice `n` indiquant la fin du tas (qui n'est pas forcément égale à la taille du tableau). Les fonctions mettront également à jour le tableau des positions `pos`. Attention on travaille avec un tas-min !

7. Écrire une fonction `dijkstra_file` qui réalise l'algorithme de Dijkstra en utilisant une file de priorité. Attention à mettre à jour la longueur du tas (qui diminue au fur et à mesure). On ne touchera pas à la case d'indice 0 qui restera à (0,0) tout au long de l'algorithme (pour être cohérent avec les indices, voir TP sur les tas). En supposant le graphe peu dense (nombre d'arêtes du même ordre de grandeur que le nombre de sommets), évaluer sa complexité et la comparer à la version précédente.