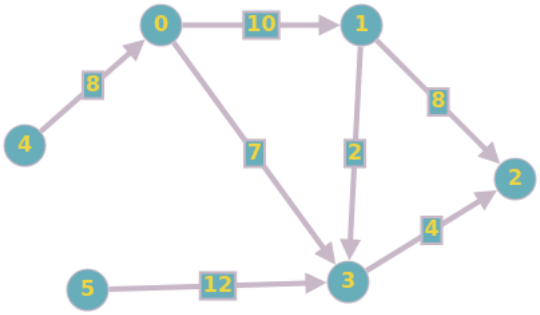
2023-2024 Université Paris 8
Algorithmique avancée Licence 3 Informatique

TP n°3 : Graphes: parcours en largeur et algorithme de Dijkstra.

Vous testerez toutes vos fonctions sur le graphe suivant. Quels sont les accessibles depuis le sommet 0, les plus courts chemins depuis 0 ?



Vous trouverez dans l'archive *Graphes.zip* sur Moodle les fichiers :

- *fifo.c* qui contient les fonctions de base sur la structure de file;
- *affichage.c* qui contient les fonctions d'affichage du tableau des accessibles ainsi que des plus courtes distances;
- *matriceadj.c* qui contient les fonctions de base sur les graphes représentés par des matrices d'adjacence, que l'on complètera;
- *heap.c* qui contient les fonctions de base sur la structure de file de priorité/tas;
- *listeadj.c* qui contient les fonctions de base sur les graphes représentés par des matrices d'adjacence, que l'on complètera;
- les fichiers header associés ainsi qu'un makefile permettant de compiler le projet, la fonction *main* se trouvant dans *main.c*.

1 Matrices d'adjacence

Vous trouverez la structure de graphe utilisant une matrice d'adjacence dans le fichier *matriceadj.h*.

1. En utilisant la définition et les fonctions sur les files, écrire une fonction *accessibles_mat* qui parcourt un graphe en largeur et renvoie un tableau indexé par tous les sommets, contenant 1 si le sommet est accessible depuis un sommet s, et 0 sinon.

2. En utilisant la fonction *minDistance* permettant de trouver le sommet non-encore parcouru situé à plus courte distance, écrivez une fonction *dijkstra_mat* implémentant l'algorithme de Dijkstra avec la version matricielle.

3. Mesurer le temps de calcul de l'algorithme de Dijkstra (version matricielle) sur des échantillons de graphes avec 100, 1000 et 10000 sommets générés aléatoirement avec la fonction *remplir_grapheM*.

2 Listes d'adjacence

Vous trouverez la structure de graphe utilisant une liste d'adjacence dans le fichier *listeadj.h*.

1. Adapter la fonction *accessibles_mat* de l'exercice précédent pour écrire une fonction *accessibles_list* produisant le même résultat avec la structure de liste d'adjacence.

2. En utilisant la définition et les primitives de la structure de tas du fichier *heap.h*, implémenter l'algorithme de Dijkstra utilisant une file de priorité pour aller chercher l'élément de la liste d'adjacence avec le plus petit poids.

3. Écrire une fonction *dikjstra_list* qui implémente l'algorithme de Dijkstra en utilisant la fonction *minDistanceL*, permettant de trouver le sommet non-encore parcouru situé à plus courte distance.

4. Mesurer le temps de calcul de ces deux versions de l'algorithme de Dijkstra (version liste) sur des échantillons de graphes avec 100, 1000 et 10000 sommets générés aléatoirement avec la fonction *remplir_grapheL*.