Aymane AMESSEGHER
Dila MEMIL
Aya Guermouh

Rendu TP3 ALGO

Lien vers le code source: https://gricad-gitlab.univ-grenoble-alpes.fr/amessega/tp3-ALGO

Idées de fonctions principaux:

dijkstra :on commence par initialiser les distances des sommets à une valeur infinie. Ensuite, on initialise la distance du sommet de départ à zéro et on le met dans une file de priorité. On boucle tant que la file n'est pas vide, en extrayant chaque fois le sommet avec la plus petite distance actuelle. Pour chaque arc sortant de ce sommet, si le sommet de destination n'a pas encore été visité ou si une distance plus courte a été trouvée, on met à jour sa distance et on l'ajoute à la file. Enfin, on affiche les distances les plus courtes trouvées pour chaque sommet.

Elementaire :on cherche d'abord le premier sommet du chemin dans le graphe. Si ce sommet n'est pas trouvé, on affiche un message d'erreur et on retourne 0. Ensuite, on parcourt les arcs du chemin pour vérifier s'il y a des répétitions de sommets. Si on trouve un sommet répété, on affiche un message d'erreur et on retourne 0. Sinon, on retourne 1, indiquant que le chemin est élémentaire.

Simple :on alloue dynamiquement un tableau pour suivre les poids des arcs déjà parcourus. Ensuite, on parcourt les arcs du chemin donné. Si on trouve un poids d'arc déjà présent dans le tableau, on libère la mémoire allouée et on retourne 0. Sinon, on ajoute le poids de l'arc au tableau et on continue. Une fois la vérification terminée, on libère la mémoire allouée pour le tableau et on retourne 1, indiquant que le chemin est simple.

Eulerien :on commence par déterminer le nombre total d'arcs dans le graphe. Ensuite, on compte le nombre d'arcs dans le chemin donné. Si le nombre d'arcs dans le chemin est égal au nombre d'arcs dans le graphe, on retourne vrai, sinon on retourne faux.

Graphe_eulerien :on parcourt chaque sommet du graphe. Pour chaque sommet, on calcule son degré sortant. Si on trouve un sommet avec un degré sortant impair, on retourne faux, indiquant que le graphe n'est pas eulerien. Sinon, si tous les sommets ont un degré sortant pair, on retourne vrai, signifiant que le graphe est eulerien.

Graphe_hamiltonien :
on commence par vérifier si le graphe est vide. Ensuite, on parcourt chaque sommet du graphe. Pour chaque sommet, on initialise les couleurs des sommets. Si on trouve un chemin hamiltonien à partir du sommet actuel, on retourne vrai. Sinon, si aucun chemin hamiltonien n'est trouvé, on retourne faux.

Distance :, on cherche d'abord les sommets x et y dans le graphe. Si l'un des sommets n'est pas trouvé, on affiche un message d'erreur et on retourne -1. Ensuite, on initialise les couleurs des sommets et les distances à partir du sommet de départ x en utilisant l'algorithme de Dijkstra. On parcourt le graphe en mettant à jour les distances jusqu'à ce qu'on atteigne le sommet y ou que tous les chemins possibles aient été explorés. Si le sommet y est atteint, on retourne la distance jusqu'à ce sommet. Sinon, si aucun chemin n'est trouvé, on affiche un message d'erreur et on retourne -1.

Excentricite :on cherche d'abord le sommet n dans le graphe. Si le sommet n'est pas trouvé, on affiche un message d'erreur et on retourne -1. Ensuite, on initialise les couleurs des sommets et les distances à partir du sommet n en utilisant l'algorithme de Dijkstra. On parcourt le graphe en mettant à jour les distances jusqu'à ce que la file de priorité soit vide. Pendant ce processus, on met à jour l'excentricité maximale si une distance plus grande est trouvée. Enfin, on retourne l'excentricité maximale trouvée.

Diametre :on parcourt chaque sommet du graphe. Pour chaque sommet, on calcule son excentricité en utilisant la fonction « excentricite ». On met à jour le diamètre maximal si une excentricité plus grande est trouvée. Enfin, on retourne le diamètre maximal trouvé parmi tous les sommets du graphe.

Afficher_profondeur :On initialise d'abord les couleurs des sommets, puis on cherche le sommet de départ, et enfin on lance récursivement le parcours en profondeur, affichant les sommets visités dans l'ordre.

Afficher_graphe_largeur: On a laissé cette fonction à la fin pour l’implémenter, avec le parcours profondeur, sous prétexte qu’elle était facile, mais on a bien galéré à la coder. On a essayé avec une file à priorité au début, comme on a utilisé une file à priorité dans la plupart des fonctions, et aussi avec une file normale, mais on n'a pas réussi à la coder comme il le faut. Il existe un petit bug de pointeur qu’on n'a pas réussi à trouver.