**TP3 :** Les Listes

Laborie Léo

Playez Romain

Linéaires Chaînées

Objectifs du TP :

L’objectif de ce TP est de créer un menu permettant à l’utilisateur de manipuler et analyser des graphes.

Fichier 1 : main.c

Ce fichier est celui qui contient la fonction main() est constitue l’interface, pour que l’utilisateur puisse intérargir avec le menu du projet.

Il est important de prendre en compte des cas extrêmes, pour assurer la robustesse du programme :

1. Suppression propre du graphe existant avant création d'un nouveau graphe vide
2. Vérification de la validité du nombre saisi de sommets (doit être un entier positif)
3. Vérification de l'existence du graphe avant d'ajouter un sommet
4. Vérification de l'existence du sommet à supprimer
5. Vérification de l'existence des sommets avant d'ajouter l'arête
6. Vérification de l'existence du graphe avant affichage
7. Vérification de l'existence du graphe avant de rechercher le degré maximal
8. Vérification de l'existence du graphe avant de rechercher une boucle
9. Vérification de l'existence des deux sommets à fusionner

Fichier 2 : tp3 .h

Ce fichier est divisé en deux parties, l’une pour déclarer les structures nécessaires pour représenter un graphe, et l’autre pour déclarer les prototypes des fonctions ,nécessaires pour manipuler et analyser ce graphe.

1. Structures :

* Graphe : représente le graphe et contient un pointeur vers le premier sommet
* Sommet : représente un sommet dans le graphe et contient un indice, un pointeur vers le sommet suivant et un pointeur vers ses voisins
* Voisin : représente un voisin d’un sommet et contient un indice et un pointeur vers le voisin suivant

1. Prototypes des fonctions :

* creerGraphe() : créer un graphe vide
* créerSommet(graphe \*g, int id) : créer un sommet
* rechercherSommet(graphe g, int id) : rechercher un sommet
* ajouterArete(graphe \*g, int id1, int id2) : ajouter une arête
* construireGraphe(int N) : construire un graphe avec N sommets
* afficherGraphe(graphe g) : afficher le graphe actuel
* rechercherDegre(graphe g) : recherche et donne le degré maximal
* supprimerSommet(graphe \*g, int id) : supprimer un sommet
* contientboucle(graphe g) : savoir si le graphe contient une boucle
* fusionnerSommet(graphe \*g, int idSommet1, int idSommet2) :fusionner deux sommets
* is\_in\_voisin(sommet s, int id\_sommet) : vérifier si un sommet est voisin d’un autre
* liberer\_proprement\_sommets(sommet \*s) : libérer la mémoire allouée aux sommets
* liberer\_proprement\_voisins(voisin \*vois) : libérer la mémoire allouée aux voisins d’un sommet

Fichier 3 : tp3.c

Ce fichier contient la définition des fonctions uilisées dans le projet.

1. « graphe\* creerGraphe()  »

Entrée : pas d’entrée

Sortie : un pointeur sur la structure graphe

Complexité : O(1) constante

Justification : Cette fonction alloue de la mémoire la structure graphe et initialise le pointeur de sommet à NULL.

1. « void creerSommet(graphe \*g, int id) »

Entrée : un pointeur vers la structure graphe et un entier id

Sortie : pas de sortie

Complexité : O(n) linéaire avec n le nombre de sommets

Justification : Avant d’allouer de la mémoire pour un nouveau sommet, cette fonction parcourt la liste des sommets du graphe pour vérifier si le sommet existe déjà.

1. « sommet\* rechercherSommet(graphe g, int id) »

Entrée : la structure graphe est un entier id

Sortie : un pointeur vers le sommet correspond à l’ID ou NULL

Complexité : O(n) linéaire avec n le nombre de sommets

Justification : Cette fonction parcourt la liste des sommets du graphe jusqu'à trouver le sommet avec l'ID ou atteindre la fin de la liste.

1. « void ajouterArete(graphe \*g, int id1, int id2) »

Entrée : un pointeur vers la structure graphe est deux entiers id1 et id2

Sortie : pas de sortie

Complexité : O(n) linéaire avec n le nombre de sommets

Justification : Cette fonction parcourt la liste des sommets du graphe jusqu'à trouver les sommets correspondant aux Ids et ajoute une arête entre les deux.

1. « graphe\* construireGraphe(int N) »

Entrée : un entier N

Sortie : un pointeur vers la structure graphe

Complexité : O(n2) quadratique avec n le nombre de sommets

Justification : Cette fonction effectue N itérations pour créer N sommets, et à chaque itération, elle parcourt la liste des sommets pour vérifier qu’il n’existe pas encore.

1. « void afficherGraphe(graphe g) »

Entrée : la structure graphe

Sortie : pas de sortie

Complexité : O(n\*m) avec n le nombre de sommets et m d’arêtes

Justification : Cette fonction parcourt chaque sommet et pour chaque sommet, elle parcourt sa liste de ses voisins pour les afficher.

1. « int rechercherDegre(graphe g) »

Entrée : la structure graphe

Sortie : un entier pour le degré maximal

Complexité : O(n\*m) avec n le nombre de sommets et m d’arêtes

Justification : Cette fonction parcourt chaque sommet et pour chaque sommet, elle compte le nombre de ses voisins.

1. « void supprimerSommet(graphe \*g, int id) »

Entrée : un pointeur vers la structure graphe et un entier id

Sortie : pas de sortie

Complexité : O(n\*m) avec n le nombre de sommets et m d’arêtes

Justification : Cette fonction parcourt la liste des sommets pour trouver le sommet à supprimer, puis pour chaque sommet, parcourt la liste de ses voisins pour supprimer le sommet voulu.

1. « int contientBoucle(graphe g) »

Entrée : la structure graphe

Sortie : un entier pour savoir si le graphe contient une boucle (1 ou 0)

Complexité : O(n\*m) avec n le nombre de sommets et m d’arêtes

Justification : Cette fonction parcourt chaque sommet et pour chaque sommet, parcourt sa liste de voisins pour vérifier s'il existe une arête vers ce même sommet.

1. « void fusionnerSommet(graphe \*g, int idSommet1, int idSommet2) »

Entrée : un pointeur vers la structure graphe deux entiers idsommet1 et 2

Sortie : pas de sortie

Complexité : O(m) linéaire avec m le nombre d’arêtes

Justification : Cette fonction recherche les sommets correspondant aux IDs, puis parcourt la liste des voisins du deuxième sommet pour ajouter ses voisins au premier sommet.

1. « int is\_in\_voisin(sommet s, int id\_sommet) »

Entrée : la structure sommet et un entier id\_sommet

Sortie : un entier pour savoir si l’ID est voisin du sommet (1 ou 0)

Complexité : O(m) linéaire avec m le nombre d’arêtes

Justification : Cette fonction parcourt la liste de voisins du sommet pour vérifier si l'ID donné est présent.

1. « void liberer\_proprement\_sommets(sommet \*s)  »

Entrée : pointeur vers la liste des sommets

Sortie : pas de sortie

Complexité : O(n) linéaire avec n le nombre de sommets

Justification : Cette fonction est récursive et parcours la liste des sommets afin de libérer la mémoire allouée a chaque sommet.

1. « void liberer\_proprement\_voisins(voisin \*vois) »

Entrée : pointeur vers la liste des voisins

Sortie : pas de sortie

Complexité : O(m) linéaire avec m le nombre d’arêtes

Justification : Cette fonction est récursive et parcours la liste des voisins afin de libérer la mémoire allouée a chaque voisin.