BELAYACHI Ibrahim
BENSALEM Nadji

Rapport TP04 - NF16
Les Arbres Binaires de Recherche

N.B :

- n : le nombre de nœuds. Le pire cas est que l'arbre est seulement un chemin, d'où la hauteur h = n.
- M : la longueur maximale des chaînes de caractères.

• Fonctions de base

T_Element *ajouterInscription(T_Element *liste, char* code)

Cette fonction crée un nouvel élément avec le code fourni en argument et l'insère de manière triée dans la liste fourni en argument, évitant les doublons.
La complexité est de O(n), car dans le pire cas, la fonction parcourt tous les éléments et insère le nouvel élément en queue de la liste.

T_Arbre inscrire(T_Arbre abr, char* nom, char* prenom, char* code)

Cette fonction inscrit un étudiant à une UV, elle traite les cas où l'arbre est vide, oùl'étudiant est déjà présent dans liste, et où l'inscription doit se faire dans le sous-arbre droit ou gauche d' une manière récursive.

Cette fonction insère donc un nœud dans un arbre binaire de recherche et met à jour la liste d'inscriptions.Sa complexité est O(h + m), où h est la hauteur de l'arbre et m est la longueur de la liste d'inscriptions pour la gestion des UVs.

T_Arbre chargerFichier(T_Arbre abr, char *filename)

Cette fonction a pour objectif de lire un fichier spécifié par le paramètre "filename", où chaque ligne représente une entrée au format "Nom/Prénom/UVXX". À partir de chaque ligne lue, la fonction crée un nœud avec la fonction "inscrire" dans l'arbre binaire de recherche (abr) en utilisant les informations extraites du nom, prénom et code d'UV. En notant k le nombre de ligne dans le fichier et O(h+m) la complexité de "inscrire" ainsi la complexité de chargerFichier est O(m*(h+m)).

void afficherInscriptions(T_Arbre abr)

Cette fonction a pour but d'afficher les détails des inscriptions des étudiants présents dans un arbre binaire de recherche (T_Arbre abr). Elle réalise un parcours récursif de l'arbre, affichant le nom et le prénom de chaque étudiant, ainsi que la liste de leurs inscriptions (UVs) en utilisant la fonction "*afficherListe*" qui est de complexité O(m) alors la complexité totale de la fonction "*afficherInscriptions*" est O(n*m).

void afficherInscriptionsUV(T_Arbre abr)

Cette fonction permet d'afficher tous les étudiants inscrits à une UV, elle parcourt tous les nœuds et dans le pire cas, elle parcourt les m éléments de la liste UV de l'étudiant à chaque nœud. Donc la complexité est O(n*m).

T_Arbre supprimerInscription(T_Arbre abr, char *nom, char *prenom, char *code)

Le pire cas est qu'il faut supprimer un nœud qui a deux fils ainsi la fonction fait d'abord appel à rechercherEtu, ensuite les fonctions strcpy sont de complexité O(M), ensuite supprimerUV de complexité O(1) (car le noeud a supprimé a exactement une seule UV), et *supprimerNoeud.* Donc la fonction est de complexité O(n*M + M + 1 + n*M) = O(n*M).

• Fonctions supplémentaires

T_Element *creerElement(char* code)

Cette fonction alloue dynamiquement de la mémoire pour créer un nouvel élément de type T_Element. Elle copie la chaîne de caractères passée en argument dans cet élément, initialise le champ suivant à NULL, et renvoie un pointeur vers le nouvel élément créé. La fonction gère également les éventuels problèmes d'allocation de mémoire en vérifiant si l'allocation a réussi avant de continuer.

L'allocation mémoire se fait dans une complexité temporelle O(1). La copie de code est de complexité O(n) où n est la longueur de la chaîne « code », or, la longueur de la chaîne est constante, elle vaut toujours 4, par conséquent, la complexité de la copie est de O(1).
Donc, la complexité totale est de O(1).

Noeud *creerNoeud(char* nom, char* prenom, T_Element* listeInscriptions)

Cette fonction alloue dynamiquement de la mémoire pour créer un nouveau nœud de type Noeud. Elle copie les chaînes de caractères nom et prenom dans les champs correspondants du nœud, initialise le champ listeInscriptions avec une liste d'éléments fournie en argument, et initialise les champs filsGauche et filsDroit à NULL. La fonction gère également les éventuels problèmes d'allocation de mémoire en vérifiant si l'allocation a réussi avant de continuer. Enfin, elle renvoie un pointeur vers le nouveau nœud créé.
Le copie de nom est de complexité O(n), où n correspond à la longueur de nom. La copie de prénom est de complexité O(m), où m représente le nombre de caractères de prénom.
Donc la complexité totale vaut O(N) où N= max(n,m).

void libererListeInscriptions (T_Element *listeInscription)

Cette fonction a pour objectif de libérer la mémoire d'une liste d'UVs (T_Element). Elle procède en libérant chaque élément de la liste de manière itérative, y compris le code

d'UV associé. Cette fonction est conçue comme une étape intermédiaire pour libérer la mémoire d'un arbre. En notant n le nombre d'éléments (T_Element) dans la liste, la complexité de cette fonction est O(n).

void libererArbre(T_Arbre *abr)

Cette fonction a pour objectif de libérer la mémoire associée à un arbre binaire de recherche (T_Arbre). Elle procède récursivement, parcourant chaque nœud de l'arbre et libérant les ressources allouées pour le nom, le prénom, et la liste d'inscriptions. La fonction garantit une libération complète de la mémoire de l'arbre, en terminant avec un arbre vide et en définissant le pointeur initial (*abr) comme NULL. Si on note m comme le nombre total d'éléments dans toutes les listes d'inscriptions de l'arbre, la complexité totale de la fonction libererArbre serait O(m+n), où n est le nombre de nœuds dans l'arbre.

void viderBuffer()

Cette fonction a pour but de vider le tampon d'entrée standard (stdin). Elle est utilisée pour nettoyer tout caractère restant dans le tampon. Soit m le nombre de caractères. Dans le pire des cas, la complexité est O(m), car la fonction lit chaque caractère un par un jusqu'à ce qu'elle atteigne une nouvelle ligne ('\n') ou la fin de fichier (EOF).

char* ToMajuscule(char* nom)

Cette fonction permet de convertir les minuscules en majuscule. Elle est de complexité O(M).

void afficherListe(T_Element* liste)

Cette fonction a pour but d'afficher tous les codes d'UV présents dans une liste de type T_Element fournie en tant qu'argument. La complexité de cette opération est linéaire, notée O(m), où m représente le nombre de codes d'UV dans la liste.

Noeud* minimum(T_Arbre abr)

Cette fonction renvoie un pointeur sur le premier étudiant (dans l'ordre alphabétique) de l'arbre abr fourni en argument.

Dans le pire cas, l'arbre n'est qu'un chemin et la fonction parcourt tout ce chemin.
À chaque itération de la boucle while, nous comparons les chaînes avec une complexité de O(M). Donc la complexité totale est de O(n*M).

int rechercheEtu(T_Arbre abr, char *nom, char *prenom)

Cette fonction affirme si l'étudiant est présent dans l'arbre ou non. Le pire cas est que l'étudiant n'est pas présent dans l'arbre abr, donc la fonction parcourt les n noeuds de l'arbre, et à chaque itération, compare les chaînes dans une complexité O(M), donc la fonction est de complexité O(n*M).

T_Element* supprimerUV(T_Element* liste, char* code)

Cette fonction permet de supprimer une UV d'une liste d'UVs d'un étudiant.
La pire cas est que l'UV est classée dernière dans l'ordre alphabétique, ainsi la fonction parcourt tous les éléments de la liste. Donc la complexité est de O(u) avec u le nombre d'éléments.

Noeud* supprimerNoeud(Noeud* racine, char *nom, char *prenom)

Cette fonction permet de supprimer un étudiant de l'arbre, elle est utilisée lorsque l'étudiant n'est inscrit à aucune UV.
Dans le pire cas la fonction parcourt les n nœuds d'une manière récursive, dans ce cas nous sommes dans une complexité de O(n). De plus, dans le pire cas le noeud a 2 fils, il recherche le minimum dans une complexité de O(n*M) et les fonctions strdup sont de complexité O(M), ensuite la fonction réalise un appel récursif pour le minimum. Le minimum est soit une feuille, soit le nœud a un fils droit, alors sa suppression se fait en O(1). Donc la complexité totale est de O(n*M)

char* Saisie(char* chaine)

Cette fonction permet de récupérer une chaîne de caractères. La complexité est de O(t) où t correspond au nombre de tentatives de l'utilisateur jusqu'à ce qu'il rentre une bonne saisie.

char* SaisieCode(char* code)

Cette fonction permet de récupérer un code UV. La complexité est de O(t) où t correspond au nombre de tentatives de l'utilisateur jusqu'à ce qu'il rentre une bonne saisie.