CHAZELAS Clément
PASZTOR Amélie

Rapport TP3

Fonction creerProduit :

Fonctionnement : La fonction creerProduit prend en paramètre une désignation, un prix et une quantité et retourne un pointeur vers une structure de type T_Produit initialisée avec ces valeurs. Elle utilise malloc pour allouer dynamiquement de la mémoire pour le pointeur et pour la désignation du produit. Elle copie également la désignation en utilisant strcpy.

Complexité : La fonction ne contient pas de boucle, ni de structure de contrôle qui s'exécutent un nombre de fois dépendant de la taille des entrées, et les opérations effectuées sont toutes de complexité constante (allocation de mémoire, copie de chaîne de caractères, assignation de valeurs à une structure). Sa complexité est donc O(1).

Fonction creerRayon :

Fonctionnement : La fonction creerRayon prend en paramètre un nom et retourne un pointeur vers une structure de type T_Rayon initialisée avec ce nom et des valeurs par défaut. Elle utilise malloc pour allouer dynamiquement de la mémoire pour le pointeur et pour le nom du rayon.

Complexité : La fonction ne contient pas de boucle, ni de structure de contrôle qui s'exécutent un nombre de fois dépendant de la taille des entrées, et les opérations effectuées sont toutes de complexité constante (allocation de mémoire, copie de chaîne de caractères, assignation de valeurs à une structure). Sa complexité est donc O(1).

Fonction creerMagasin :

Fonctionnement : La fonction creerMagasin prend en paramètre un nom et retourne un pointeur vers une structure de type T_Magasin initialisée avec ce nom et des valeurs par défaut. Elle utilise malloc pour allouer dynamiquement de la mémoire pour le pointeur et pour le nom du magasin.

Complexité : Elle ne contient pas de boucle ni de structure de contrôle qui dépendent de la taille des entrées. Les opérations effectuées sont toutes de complexité constante, à savoir l'allocation de mémoire, la copie de chaîne de caractères et l'assignation de valeurs à une structure. Ainsi, le temps d'exécution de cette fonction ne dépend pas de la taille de la chaîne de caractères pour le nom du magasin. Sa complexité est donc O(1)

Fonction ajouterRayon :

Fonctionnement : La fonction ajouterRayon prend en paramètre un pointeur vers une structure T_Magasin et un nom de rayon. Elle ajoute un rayon au magasin trié par ordre alphabétique en fonction de son nom. Si le rayon existe déjà, il ne sera pas ajouté. Cette fonction utilise une boucle pour parcourir un par un les rayons du magasin et trouve l'emplacement pour ajouter le nouveau rayon. Si l'ajout doit se faire en tête de liste, un nouveau rayon est créé et il devient la nouvelle tête de liste. Sinon, un nouveau rayon est créé et inséré au bon endroit. C’est une liste chaînée donc pour insérer, on modifie le

pointeur du rayon situé avant dans la liste pour qu'il pointe vers notre nouveau rayon, et notre nouveau rayon pointe vers celui qui était auparavant pointé par le rayon situé avant.

Complexité : La fonction utilise une boucle while pour parcourir la liste des rayons du magasin, jusqu'à trouver l'emplacement où insérer le nouveau rayon. Dans le pire des cas, la boucle doit parcourir tous les rayons de la liste, donc la complexité est de O(n). De plus, la fonction appelle la fonction creerRayon qui a une complexité de O(1) pour chaque rayon à insérer,ce qui ne change pas sa complexité totale. Enfin, les autres opérations effectuées dans la fonction, comme la comparaison de chaînes de caractères et l'assignation de valeurs à des pointeurs, ont également une complexité de O(1). La complexité de cette fonction est donc O(n), où n est le nombre de rayons déjà présents dans la liste.

Fonction ajouterProduit :

Fonctionnement : La fonction ajouterProduit prend en paramètre un pointeur vers une structure T_Rayon, une désignation, un prix et une quantité. Elle ajoute un produit au rayon s'il n'existe pas déjà un produit ayant la même désignation. Cette fonction utilise une boucle pour parcourir les produits du rayon et vérifie si un produit existe déjà avec la même désignation. Si le produit existe déjà, il ne sera pas ajouté. Sinon, la fonction trouve l'emplacement pour ajouter le nouveau produit. Si l'ajout doit se faire en tête de liste, un nouveau produit est créé et il devient la nouvelle tête de liste. Sinon, un nouveau produit est créé et inséré au bon endroit. La liste des produits est également une liste chaînée donc l'insertion de rayons fonctionne de la même manière.

Complexité : La complexité de cette fonction dépend de deux boucles. La première boucle while parcourt la liste des produits du rayon pour vérifier s'il existe déjà un produit avec la même désignation. Dans le pire des cas, cette boucle doit parcourir tous les produits de la liste, ce qui donne une complexité de O(n). La deuxième boucle while est utilisée pour trouver l'emplacement où insérer le nouveau produit en fonction de son prix. Dans le pire des cas, cette boucle va également parcourir tous les produits de la liste, donnant une complexité de O(n). Les autres opérations effectuées dans la fonction, comme la comparaison de chaînes de caractères, l'allocation de mémoire, l'assignation de valeurs à des pointeurs ainsi que l'appel à la fonction creerProduit, ont toutes une complexité de O(1). La complexité de cette fonction sera donc O(n), où n est le nombre total de produits déjà présents dans le rayon.

Fonction afficherMagasin :

Fonctionnement : La fonction afficherMagasin prend en paramètre un pointeur vers une structure T_Magasin et affiche le contenu du magasin, c'est-à-dire le nom du magasin et la liste de ses rayons et de leurs produits. Cette fonction utilise des boucles pour parcourir les rayons et les produits du magasin, et affiche les informations correspondantes.

Complexité : La complexité de cette fonction dépend de deux boucles imbriquées. La boucle externe parcourt tous les rayons du magasin, donc elle s'exécute n fois. À l'intérieur de cette boucle, on a une boucle interne qui compte le nombre de produits dans chaque rayon. Pour cela, elle va parcourir tous les produits d'un rayon donné, donc elle s'exécute m fois. La complexité de cette fonction est donc O(n*m) où n est le nombre de rayons dans le magasin et m est le nombre de produits dans chaque rayon.

Fonction afficherRayon :

Fonctionnement : La fonction afficherMagasin prend en paramètre un pointeur vers une structure T_Magasin et affiche tous les produits d'un rayon donné. Elle parcourt chaque élément de la liste de produits du rayon et affiche les détails de chaque produit grâce à un print.

Complexité : La complexité de cette fonction dépend de la boucle while qui parcourt tous les produits de la liste une fois pour afficher leurs informations. Le reste des opérations dans la fonction est en O(1). La complexité de cette fonction est donc en O(n), où n est le nombre de produits dans le rayon.

Fonction supprimerProduit:

Fonctionnement : Cette fonction supprime un produit spécifié par sa désignation d'un rayon donné. Elle parcourt la liste des produits du rayon jusqu'à trouver le produit spécifié. Si le produit n'est pas trouvé, la fonction retourne 0, sinon elle supprime le produit et retourne 1. La liste des produits est une liste chaînée donc pour supprimer un produit, il faut libérer la mémoire, mais aussi supprimer l'objet dans la liste, donc modifier le pointeur rayon.suivant du rayon précédant le rayon supprimé pour qu'il pointe vers le rayon suivant le rayon supprimé.

Complexité : Dans cette fonction, on cherche d'abord le produit à supprimer en parcourant la liste de produits. Cette recherche a une complexité linéaire O(n) dans le pire des cas, car on doit parcourir tous les produits pour trouver celui à supprimer. Ensuite, si le produit est trouvé, on le supprime de la liste en mettant à jour les pointeurs de la liste. Cela se fait en temps constant, O(1), car on ne fait que mettre à jour les pointeurs, sans avoir besoin de parcourir la liste. Enfin, on libère la mémoire allouée pour la désignation du produit et pour le produit lui-même, ce qui se fait également en temps constant, O(1). La complexité totale de cette fonction est O(n) dans le pire des cas, où n est le nombre de produits dans le rayon

Fonction supprimerRayon :

Fonctionnement : Cette fonction supprime un rayon spécifié par son nom d'un magasin donné. Elle parcourt la liste des rayons du magasin jusqu'à trouver le rayon spécifié. Si le rayon n'est pas trouvé, la fonction affiche un message d'erreur, sinon elle supprime le rayon ainsi que ses produits un par un. De la même manière que pour les produits, il faut supprimer le rayon dans la liste et donc le rayon précédant celui qu'on veut supprimer pointera vers le suivant.

Complexité : Dans le pire des cas, si le rayon à supprimer se trouve à la fin de la liste des rayons, la fonction devra parcourir tous les rayons jusqu'à trouver le rayon à supprimer. La complexité est de donc l'ordre de O(n), où n est le nombre de rayons dans le magasin.
De plus, dans le cas où le rayon à supprimer est trouvé, la fonction doit également supprimer tous les produits associés à ce rayon. Pour chaque produit, il y a une boucle de libération de mémoire qui parcourt la liste des produits. La complexité de cette partie est donc de l'ordre de O(m), où m est le nombre de produits à supprimer. La complexité de cette fonction est donc de l'ordre de O(n + m), où n est le nombre de rayons dans le magasin et m est le nombre de produits dans le rayon à supprimer.

Fonction rechercheProduits :

Fonctionnement : Cette fonction recherche les produits d'un magasin qui se situent dans une fourchette de prix spécifiée. Elle parcourt la liste de rayons du magasin et pour chaque rayon, elle parcourt la liste des produits et vérifie si le prix du produit est compris entre le prix minimum et maximum spécifié. Si un produit correspond à ces critères, elle l'affiche.

Complexité : Le premier bloc if vérifie si le magasin est vide en temps constant, donc O(1).
Ensuite, il y a une boucle while qui parcourt chaque rayon dans le magasin. Pour chaque rayon, il y a une autre boucle while qui parcourt chaque produit dans ce rayon. À l'intérieur de la deuxième boucle while, il y a une condition if qui vérifie si le prix du produit est compris entre le prix minimum et le prix maximum. Si tel est le cas, le produit est ajouté au rayon temporaire en utilisant la fonction ajouterProduit, qui a une complexité de O(1). Si des produits ont été trouvés, la fonction afficherRayon est appelée, qui a une complexité de O(m) pour chaque rayon (puisque nous ne parcourons que les produits dans le rayon temporaire). Ainsi, la complexité totale de cette fonction est O(n * m).

Fonction fusionnerRayons :

Fonctionnement : Dans cette fonction, on commence par parcourir la liste des rayons pour afficher les rayons existants et proposer à l'utilisateur de choisir parmi eux. Il est bien sûr impossible d'utiliser la fonction s'il y a moins de 2 rayons, on vérifie cela à l'aide d'un if. Ensuite on récupère le choix de l'utilisateur à l'aide de deux scanf. On cherche ces deux rayons et on initialise deux pointeurs qui pointent sur chacun d'entre eux.
Ensuite on demande à l'utilisateur d'entrer le nom du nouveau rayon qu'on ajoute au magasin grâce à la fonction ajouterRayon. Pour remplir ce nouveau rayon, on parcourt les produits des deux anciens un par un et on ajoute chaque produit dans le nouveau rayon à l'aide de la fonction ajouterProduit. Enfin, on supprime les deux anciens rayons à l'aide de la fonction supprimerRayon.

Complexité : On a en premier une boucle while qui parcourt la liste des rayons pour afficher leur nom, ce qui a une complexité linéaire O(n), où n est le nombre de rayons dans le magasin. On a ensuite trois boucles while qui cherchent les rayons dans le magasin en fonction de leur nom, chacune ayant également une complexité linéaire O(n).
On a ensuite deux boucles while, la première parcourt la liste de produits du premier rayon, qui contient n1 produits et fait appel à la fonction ajouterProduit qui a une complexité de O(m), m étant le nombre de produit dans le rayon. La deuxième boucle fait la même chose avec le deuxième rayon qui contient n2 produits. Les deux boucles ont donc respectivement une complexité de O(n1 * m1) et O(n2 * m2), où n1 et n2 sont respectivement la taille des listes de produits des deux rayons et m1 et m2 le nombre de produits dans chacun des deux rayons. On va ensuite faire deux appels à la fonction supprimerRayon qui a une complexité de O(n + m), où n est le nombre de rayons dans le magasin et m est le nombre de produits dans le rayon à supprimer. Enfin, il y a quelques opérations de lecture et d'écriture qui ont une complexité constante. On a donc une complexité de O(7n +m1*n1 + m2*n2 + m1 + m2), où n est le nombre de rayons dans le magasin et n1 et n2 sont respectivement la taille des listes de produits des deux rayons et m1 et m2 le nombre de produits dans chacun des deux rayons.

CHAZELAS Clément
PASZTOR Amélie