NF16 : RAPPORT TP3

LISTE CHAÎNÉES ET MATRICE CREUSE

NINA FORDE
ISMAIL KADIRI

1. OBJECTIF DU TP

L'objectif de ce TP est de nous familiariser avec l'utilisation des listes chaînées grâce à l'exemple des matrices creuses. Il nous faut donc manipuler des matrices creuses et proposer des algorithmes pour les fonctionnalités suivantes :

1- Remplir une matrice creuse.
2- Afficher une matrice creuse.
3- Donner la valeur d'un élément d'une matrice creuse.
4- Affecter une valeur à un élément d'une matrice creuse.
5- Additionner deux matrices creuses.
6- Calculer le gain en espace (par rapport à une matrice classique) en utilisant cette représentation pour une matrice donnée.

2. FONCTIONS PRINCIPALES

Remplir une matrice : *int remplirMat(matrice_creuse *m, int N, int M)* ;

Dans l'optique de rendre l'affichage de la matrice plus parlante nous avons fait le choix de renvoyer la taille du plus grand élément saisi et non pas de conserver *void* comme indiqué.

Complexité : Au départ nous avons des opérations élémentaires en **O(1).**
Ensuite nous allons chercher à parcourir toutes les colonnes de chaque ligne pour rentrer la valeur.
Dans tous les cas on aura M*N valeurs entrées. La complexité est donc en **O(M*N)**.

Afficher une matrice : *void afficherMat(matrice_creuse m, int longest_chiffre) ;*

Complexité : On va faire un travail similaire avec cette fonction en affichant les éléments entrés précédemment.
Donc on va parcourir toutes les lignes et colonnes. Dans le pire des cas toutes les cases sont non nulles et donc on a une complexité en **O(M*N).**

Additionner deux matrices : *void addMat(matrice_creuse m1, matrice_creuse m2) ;*

Complexité : Étant donné que les matrices ont les mêmes dimensions, nous allons parcourir M colonnes pour chacune des N lignes. La complexité est en **O(M*N)**.

Donner la valeur d'une matrice : *int getValue(matrice_creuse m, int i, int j) ;*

Complexité : On connaît l'emplacement de l'élément que l'on veut afficher. On se rend donc directement à l'adresse de la ligne concernée (O**(1)**). Puis dans le pire des cas on parcourt toutes les colonnes de cette ligne. On a donc une complexité en **O(M)**.

Modifier un élément d'une matrice : *void putValue(matrice_creuse m, int i, int j, int val) ;*

Complexité : Fonctionnement similaire à celui de *getValue*, on se rend à la ligne de l'élément que l'on veut modifier puis on part toutes les colonnes dans le pire des cas. La complexité est donc en **O(M)**.

Calculer le gain d'espace d'une matrice : *int nombreOctetsGagnes(matrice_creuse m1) ;*

Complexité : On parcourt chaque ligne et au pire des cas chaque colonne pour savoir combien d'éléments ont été enregistrés. On a donc une complexité **O(M*N).**

3. FONCTIONS SUPPLÉMENTAIRES

Nous avons fait le choix d'implémenter 5 fonctions supplémentaires dont 3 rendant l'interface utilisateur plus agréable avec l'affichage d'une matrice avec des cases et en centrant les valeurs.

int get_int_len(int value);
Cette fonction permet de connaître la taille d'un élément saisi par l'utilisateur. Elle réalise seulement des opérations élémentaires elle est donc en **O(1)**.

void PrintBorder(int j,int longestlength);
Cette fonction sert à afficher les bordures extérieures et intérieures de la matrice. Elle aussi comporte des affectations basiques donc sa complexité est en **O(1).**

void PrintRow(int num_espaces,int valeur) ;
Cette fonction permet de centrer la valeur dans la case en mettant la moitié des espaces à gauche et l'autre à droite. Complexité en **O(1).**

void freeMat(matrice_creuse *m) ;

Cette fonction permet de libérer l'espace alloué dynamiquement pour les matrices à la fin de notre programme. Elle libère l'espace initialement alloué à chaque élément de toutes les lignes en parcourant la liste chaînée.
On a donc une complexité en **O(M*N).**

element * newelement(element * suivant, int col,int valeur) ;
Cette fonction est utilisée pour créer un nouvel élément et lui allouer un espace mémoire. Cette fonctionnalité étant souvent utilisée, il nous a paru judicieux de créer une fonction à part fin d'alléger le code et le rendre plus lisible.
Elle réalise uniquement un travail d'affectatio elle est donc elle aussi en **O(1).**