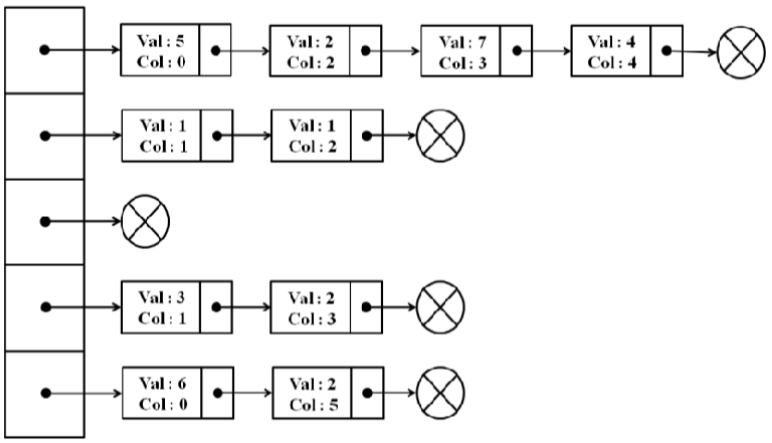
Rapport
TP3 - Matrice creuse

$$\begin{pmatrix} 5 & 0 & 2 & 7 & 4 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 2 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$



ZHANG Anxian
MIGUEU Bryan

26/11/2024

Déterminer les complexités algorithmiques des fonctionnalités

N.B. : Dans les explications ci-dessous, on s'intéressera aux complexités dans le pire des cas.

Fonctions se trouvant dans le fichier utile.h utilisé par SparseMatrix.h:

- trim: dans le pire cas, il y a une chaîne de caractère nulle de taille n qui est fournis a la fonction, les deux while parcours alors intégralement la chaîne de caractère. La complexité est donc de ***O(2n) soit O(n)***.
- coverExcessSpace: le cas le plus défavorable dans, c'est de fournir deux valeurs, et au milieu d'ajouter 'n' espaces. La Complicité sera alors de ***O(n-2), soit O(n)***.

Remplir une matrice:
Pour chaque ligne fournie, un traitement en amont est effectué avec les fonctions *trim* et *coverExcessSpace*, afin de retirer tout surplus de caractères vides dans les chaînes fournies. Cela implique déjà une complexité de ***O(n)*** pour l'utilisation de ces deux fonctions sur une ligne donnée.
Dans le pire des cas, l'utilisateur fournit une matrice de taille $N \times M$, avec tous les indices $(i, j) \in N \times M$ remplis. La complexité de création de la matrice sera alors de *N* lignes * *M* colonnes, soit $O(nm + n) = O(nm)$.

Afficher une matrice creuse sous forme de tableau:
Pour une matrice de *N* lignes et *M* colonnes, dans le pire des cas, toutes les lignes et colonnes sont remplies. Chaque élément de chaque élément des ligne est parcouru de manière récursive, avec une condition d'arrêt lorsqu'il n'y a plus d'éléments à parcourir dans la ligne fournie. La complexité est alors de ***O(nm + n) = O(nm)***.

Donner la valeur d'un élément d'une matrice:
Ici, le pire des cas correspond à la recherche d'une valeur qui se trouve à la dernière position d'une ligne. La recherche, effectuée de manière récursive, se fait alors dans l'ordre de ***O(m)***, où M correspond au nombre de colonnes.

Somme entre deux matrices de même taille N * M:
La fonction parcourt chaque ligne de la matrice, soit N lignes au total, ce qui contribue à une complexité de ***O(n)***.
Pour chacune des lignes. La complexité dépend du nombre de parcours d'éléments stockés dans les représentations creuses des matrices. Si m_1 et m_2 sont les nombres maximaux d'éléments stockés dans une ligne de m1 et m2, alors pour une ligne, la complexité est ***O(max(m_1, m_2))***.
Comme chaque ligne est traitée indépendamment, la complexité totale est alors de $O(N \times max(m_1, m_2))$.

Afficher une matrice creuse de taille n*m

Le pire cas est celui où la matrice ne contient aucune valeur nulle. Dans ce cas, on entre dans la boucle for n fois et à chaque fois, on entre dans le while m fois, ce qui donne une complexité dans le pire cas de ***O(m*n)***.

Supprimer un élément dans une matrice creuse de taille n*m:

Le pire cas est de supprimer la dernière valeur de n'importe quelle colonne. On vérifie d'abord si la valeur existe avec la fonction existence() de complexité ***O(m)***, puis on recherche la valeur dans un while avec une complexité de ***O(m)***. Au final, on a une complexité de ***O(m)***.

Ajouter une valeur dans une matrice creuse de taille n*m:

Le pire cas est celui où on ajoute à la dernière colonne de n'importe quelle ligne. On a deux cas :

1. **Valeur à ajouter non nulle** : On a l'indice de la ligne, donc on parcourt toute la colonne et on insère à la fin, d'où une complexité de ***O(m)***.
2. **Valeur à ajouter nulle** : On utilise la fonction delete_value() d'une complexité de ***O(m)***. À la fin, on obtient une complexité de ***O(m)***.

Trouver le nombre d'octets d'une matrice creuse de taille n*m:

Le pire cas est celui où il n'y a aucune valeur nulle dans la matrice. Dans ce cas, on parcourt toute la matrice en additionnant toutes les tailles de chaque élément, d'où une complexité de ***O(m*n)***.

Trouver le nombre d'octets d'une matrice où les éléments contenant 0 occupent un espace en mémoire:

Dans ce cas, le pire cas est équivalent au meilleur cas. On possède déjà la taille de la matrice, donc on fait simplement des multiplications et des additions, d'où une complexité de ***O(1).***

Libérer l'espace mémoire d'une matrice de taille n*m:

Le pire cas est celui où il n'y a aucune valeur nulle. Dans ce cas, on parcourt toute la matrice et on libère chaque élément, ce qui donne une complexité de ***O(m*n)***.

Libérer la mémoire d'un tableau de t matrices:

Le pire cas correspond à celui où aucune matrice ne contient de valeur nulle. Dans ce cas, on supprime chaque matrice n*i*mi avec une complexité de* ***O(mi*ni)***, et cela t fois, d'où une complexité de ***O(*$\sum$*i=0 .. t ni*mi).***