稀疏矩阵的加法和乘法

班级：2022217803
姓名：密言
学号：2022210064

2024.4.16

1 题目分析

矩阵 A 和 B 均以三元组表作为储存结构，写出矩阵相加和相乘的算法，另设三元组表 C 存放结果矩阵。

要求：

1. 从键盘输入矩阵 A 和 B。
2. 检测 A 和 B 是否能够相加/相乘。
3. 如能，做矩阵的相加/相乘运算，并打印运算结果。
4. 如不能，应显示出原因。

本题的输入为矩阵 A 和矩阵 B，输出为它们的相加和相乘结果（如果可以进行相加/相乘运算），或者输出无法进行相加/相乘运算的原因。

下面是两个输入输出的示例：

```
//eg1
3 3
5 0 0
0 0 1
0 0 0
3 2
2 0
0 0
4 0
Can not sum!
Multiplication:
10 0
4 0
0 0

//eg2
3 3
1 2 3
```

```
3 2 1
1 3 2
3 3
1 1 1
2 2 2
3 3 3
Sum:
2 3 4
5 4 3
4 6 5
Multiplication:
14 14 14
10 10 10
13 13 13
```

2 概要设计

首先需要定义一个结构体用来存放矩阵中的元素，由于存放的是稀疏矩阵，因此可以用三元组来存放矩阵中的元素。

需要定义函数用于在堆区开辟用于存放矩阵的空间，设计一个函数 TSMatrix* create(int n,int m) 在开辟空间的同时可以存入矩阵中的非零元素，用于存放 A、B 矩阵；同时设计一个函数 TSMatrix* createNULL(int n,int m) 来开辟一个空的矩阵空间。

函数 int getnum(TSMatrix *M,int i,int j) 来返回矩阵 M 中位于 (i, j) 位置的元素值。

函数 TSMatrix *Sum(TSMatrix* A,TSMatrix *B) 和 TSMatrix *Multiply(TSMatrix *A,TSMatrix* B) 来返回矩阵 A 和 B 相加/相乘的结果。

函数 void PrintMatrix(TSMatrix *M) 用于打印结果。

3 详细设计

3.1 矩阵的三元组储存形式

一个矩阵由行数 n、列数 m、非零元素个数 t 和 t 个非零元素组成，其中 t 个非零元素由数组顺序存储。数组的每个元素是结构体类型，包含该元素在数组中的位置 i、j 以及该元素的值 v。矩阵的三元组定义代码如下：

```
typedef struct tuple{
    int i;
    int j;
    int v;
}TupleNode;

```

```
typedef struct matrix{
    int n,m,t;
    TupleNode* data;
}TSMatrix;
```

3.2 在堆区开辟空间存放矩阵

将矩阵的相关数据存放在堆区，可以避免函数结束时数据被清除。函数 TSMatrix* create(int n,int m) 在堆区开辟一个可以存放 $n*m$ 矩阵的空间，并且存放其中的非零元素；函数 TSMatrix* createNULL(int n,int m) 在堆区开辟一个可以存放 $n*m$ 矩阵的空间，不存放元素，等待运算结果的存入。实现代码如下：

```
TSMatrix* create(int n,int m){
    TSMatrix* matr= malloc(sizeof(TSMatrix));
    matr->data= malloc(sizeof(TupleNode)*n*m);
    matr->n=n;
    matr->m=m;
    int cnt=0;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            int temp;
            scanf("%d",&temp);
            if(temp!=0){
                matr->data[cnt].i=i;
                matr->data[cnt].j=j;
                matr->data[cnt].v=temp;
                cnt++;
            }
        }
    }
    matr->t=cnt;
    return matr;
}

TSMatrix* createNULL(int n,int m){
    TSMatrix* matr= malloc(sizeof(TSMatrix));
    matr->data= malloc(sizeof(TupleNode)*n*m);
    matr->n=n;
    matr->m=m;
    return matr;
}
```

3.3 获取矩阵中位于 (i, j) 的元素值

当我们进行矩阵的加法或乘法时，需要逐个元素进行计算，因此需要获取矩阵在某个位置的元素值。查找方法为: 遍历三元组表，若找到了对应的 i 和 j，则返回相应的 v；否则返回 0。代码实现如下：

```
int getnum(TSMatrix *M,int i,int j){
    for(int k=0;k<M->t;k++){
        if(M->data[k].i==i&&M->data[k].j==j){
            return M->data[k].v;
        }
    }
    return 0;
}
```

3.4 矩阵加法

矩阵加法即相同位置的元素相加，遍历矩阵的每一个位置，获取该位置的元素值并相加，如果结果非零，则存入结果矩阵对应的三元组表中即可。实现代码如下：

```
TSMatrix *Sum(TSMatrix* A,TSMatrix *B){
    TSMatrix *C= createNULL(A->n,A->m);
    int cnt=0;
    for(int i=1;i<=A->n;i++){
        for(int j=1;j<=B->m;j++){
            int temp=getnum(A,i,j)+getnum(B,i,j);
            if(temp!=0){
                C->data[cnt].i=i;
                C->data[cnt].j=j;
                C->data[cnt].v=temp;
                cnt++;
            }
        }
    }
    C->t=cnt;
    return C;
}
```

3.5 矩阵乘法

模拟矩阵相乘的过程，如果对应结果矩阵位置的元素非 0，则存入结果矩阵的三元组表中。实现代码如下：

```
TSMatrix *Multiply(TSMatrix *A,TSMatrix* B){
```

```
    TSMatrix *D= createNULL(A->n,B->m);
    int cnt=0;
    for(int i=1;i<=A->n;i++){
        for(int j=1;j<=B->m;j++){
            int temp=0;
            for(int k=1;k<=A->m;k++){
                temp+= getnum(A,i,k)* getnum(B,k,j);
            }
            if(temp!=0){
                D->data[cnt].i=i;
                D->data[cnt].j=j;
                D->data[cnt].v=temp;
                cnt++;
            }
        }
    }
    D->t=cnt;
    return D;
}
```

4 调试分析报告

4.1 问题及解决方法

针对三元组表只存储矩阵中非 0 元素的特点，不方便直接获取位于 (i, j) 位置的元素，故设计 int getnum(TSMatrix *M,int i,int j) 来获取元素值。

4.2 时空分析

在时间上，获取位于 (i, j) 位置的元素需要 $O(n)$ 的复杂度，矩阵加法的时间复杂度为 $O(n^3)$，矩阵乘法的时间复杂度为 $O(n^4)$。
在空间上，需要两个结构体用于存放矩阵 A 和 B，其中每个结构体存放的数据包括矩阵的行、列、非 0 元素个数和储存非 0 元素的三元组表，每个三元组由元素在矩阵中的行、列以及该元素的值构成。

4.3 改进设想

矩阵乘法的时间复杂度较高，在牺牲存储空间的情况下，可以被优化为 $O(n^3)$。但是当矩阵维数较高且足够稀疏的情况下，本算法在存储上更有优势。

5 用户使用说明

将附录中的代码复制到 IDE 中运行即可。

6 测试结果

6.1 test 1

```
3 3
1 2 3
3 1 2
2 1 3
3 3
1 1 1
2 2 2
3 3 3
Sum:
2 3 4
5 3 4
5 4 6
Multiplication:
14 14 14
11 11 11
13 13 13
```

6.2 test 2

```
1 3
1 2 3
3 2
1 2
3 4
5 6
Can not sum!
Multiplication:
22 28
```

6.3 test 3

```
2 3
1 2 3
```

```
4 5 6
1 2
1 2
Can not sum!
Can not multiply!
```

A 附录：完整代码

```c
//
// Created by 密言 on 2024/4/16.
//
#include <stdio.h>
#include <stdlib.h>

//定义矩阵的三元组存储形式
typedef struct tuple {
    int i;
    int j;
    int v;
} TupleNode;

typedef struct matrix {
    int n, m, t;
    TupleNode *data;
} TSMatrix;

//创建矩阵并存储数据
TSMatrix *create(int n, int m) {
    TSMatrix *matr = malloc(sizeof(TSMatrix));
    matr->data = malloc(sizeof(TupleNode) * n * m);
    matr->n = n;
    matr->m = m;
    int cnt = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            int temp;
            scanf("%d", &temp);
            if (temp != 0) {
                matr->data[cnt].i = i;
                matr->data[cnt].j = j;
```

```
33                  matr->data[cnt].v = temp;
34                  cnt++;
35              }
36          }
37      }
38      matr->t = cnt;
39      return matr;
40  }
41
42  //创建空矩阵
43  TSMatrix *createNULL(int n, int m) {
44      TSMatrix *matr = malloc(sizeof(TSMatrix));
45      matr->data = malloc(sizeof(TupleNode) * n * m);
46      matr->n = n;
47      matr->m = m;
48      return matr;
49  }
50
51  //获取位于(i,j)的数据
52  int getnum(TSMatrix *M, int i, int j) {
53      for (int k = 0; k < M->t; k++) {
54          if (M->data[k].i == i && M->data[k].j == j) {
55              return M->data[k].v;
56          }
57      }
58      return 0;
59  }
60
61  //矩阵加法
62  TSMatrix *Sum(TSMatrix *A, TSMatrix *B) {
63      TSMatrix *C = createNULL(A->n, A->m);
64      int cnt = 0;
65      for (int i = 1; i <= A->n; i++) {
66          for (int j = 1; j <= B->m; j++) {
67              int temp = getnum(A, i, j) + getnum(B, i, j);
68              if (temp != 0) {
69                  C->data[cnt].i = i;
70                  C->data[cnt].j = j;
71                  C->data[cnt].v = temp;
72                  cnt++;
73              }
```

```
74          }
75      }
76      C->t = cnt;
77      return C;
78  }
79
80  //矩阵乘法
81  TSMatrix *Multiply(TSMatrix *A, TSMatrix *B) {
82      TSMatrix *D = createNULL(A->n, B->m);
83      int cnt = 0;
84      for (int i = 1; i <= A->n; i++) {
85          for (int j = 1; j <= B->m; j++) {
86              int temp = 0;
87              for (int k = 1; k <= A->m; k++) {
88                  temp += getnum(A, i, k) * getnum(B, k, j);
89              }
90              if (temp != 0) {
91                  D->data[cnt].i = i;
92                  D->data[cnt].j = j;
93                  D->data[cnt].v = temp;
94                  cnt++;
95              }
96          }
97      }
98      D->t = cnt;
99      return D;
100 }
101
102 //打印矩阵
103 void PrintMatrix(TSMatrix *M) {
104     int cnt = 0;
105     for (int i = 1; i <= M->n; i++) {
106         for (int j = 1; j <= M->m; j++) {
107             if (M->data[cnt].i == i && M->data[cnt].j == j && cnt <
                    M->t) {
108                 printf("%d ", M->data[cnt].v);
109                 cnt++;
110             } else {
111                 printf("0 ");
112             }
113         }
```

```
114             printf("\n");
115         }
116     }
117
118     int main() {
119         int n1, m1;
120         scanf("%d %d", &n1, &m1);
121         TSMatrix *A = create(n1, m1);
122         int n2, m2;
123         scanf("%d %d", &n2, &m2);
124         TSMatrix *B = create(n2, m2);
125
126         if (n1 == n2 && m1 == m2) {
127             printf("Sum:\n");
128             TSMatrix *C = Sum(A, B);
129             PrintMatrix(C);
130         } else {
131             printf("Can not sum!\n");
132         }
133
134         if (m1 == n2) {
135             printf("Multiplication:\n");
136             TSMatrix *D = Multiply(A, B);
137             PrintMatrix(D);
138         } else {
139             printf("Can not multiply!\n");
140         }
141     }
```