4.1 题 稀疏矩阵运算器
实习报告

题目：实现一个能进行稀疏矩阵基本运算的运算器。
班级：02
成员：张磊（2017K8009922027）赵鑫浩（2017K8009922032）姜小平（2017K8009922014）
完成日期：2019.06.29

一、 需求分析

1. 稀疏矩阵是指那些多数元素为 0 的矩阵。为节省存储空间，提高计算效率，我们需要利用其“稀疏”特点编写程序。
2. 稀疏矩阵的表示方法采用书中的以“带行逻辑连接信息”的三元组顺序表。演示程序采用文件输入输出的方法，输入文件 datain.in 中每行代表一个计算表达式，输出文件 dataout.out 中对于每个计算式包括序号、两个运算矩阵和一个结果矩阵。
3. 输入文件中，每个大括号{}内表示一个稀疏矩阵，里面包括矩阵的行数和列数，而中括号[]内表示稀疏矩阵所有点的信息，小括号()表示稀疏矩阵一个点的行号、列号和值，用分号隔开。
4. 程序执行的命令：
   （1） 读入一行数据直到读到文件结束符 EOF；
   （2） 判断运算符及其位置，建立两个运算稀疏矩阵，输出；
   （3） 判断能否进行运算并输出相关信息，如果能运算，则调用相关运算函数得到结果矩阵，输出。
5. 测试数据（三个）：
   （1）输入：

```
{3;3;[(1,1,10);(2,3,9);(3,1,-1)]}+{3;3;[(2,3,-1);(3,1,1);(3,3,-3)]}
{3;2;[(1,1,10);(2,2,9);(3,1,-1)]}-{3;2;[(2,2,-1);(3,1,1);(3,2,-3)]}
{4;5;[(1,1,4);(1,2,-3);(1,5,1);(2,4,8);(3,3,1);(4,5,70)]}*{5;3;[(1,1,3);(2,1,4);(2,2,2);(3,2,1);(4,1,1)]}
```

   （2）输出：

```
第1个矩阵计算式:
10   0    0
0    0    9
-1   0    0

0    0    0
0    0    -1
1    0    -3

10   0    0
0    0    8
0    0    -3

第2个矩阵计算式:
10   0
0    9
-1   0

0    0
0    -1
1    -3

10   0
0    10
-2   3

第3个矩阵计算式:
4    -3   0    0    1
0    0    0    8    0
0    0    1    0    0
0    0    0    0    70

3    0    0
4    2    0
0    1    0
1    0    0
0    0    0

0    -6   0
8    0    0
0    1    0
0    0    0
```

二、 概要设计

1. 抽象数据类型稀疏矩阵的定义如下：使用以“带行逻辑连接信息”的三元组顺序表。

```
ADT RLSMatrix {
    数据对象：D={}
    数据关系：
    基本操作：
        InitRLSMatrix(&M);
            初始条件：稀疏矩阵 M 存在。
            操作结果：初始化稀疏矩阵 M。
        GetRLSMatrix(&M);
            操作结果：根据输入得到稀疏矩阵 M。
        PrintRLSMatrix(RLSMatrix M);
            初始条件：稀疏矩阵 M 存在。
            操作结果：输出稀疏矩阵 M。
        addMatrix(M, N, &Q);
            初始条件：稀疏矩阵 M 和 N 的行数和列数对应相等。
            操作结果：求稀疏矩阵的和 Q=M+N。
        subMatrix(M, N, &Q);
            初始条件：稀疏矩阵 M 和 N 的行数和列数对应相等。
            操作结果：求稀疏矩阵的差 Q=M-N。
        multiplyMatrix(M, N, &Q);
            初始条件：稀疏矩阵 M 的列数等于 N 的行数。
            操作结果：求稀疏矩阵的和 Q=M*N。
} ADT RLSMatrix
```

2. 主程序

```
int main(void){
    while(读入的一行不为 EOF){
        初始化 M 和 N；
        判断运算符及其位置；
        得到稀疏矩阵 M 和 N，并输出；
        判断运算能否进行，若能进行则输出结果矩阵 Q；
    }
    return 0;
}
```

三、 详细设计

1. 稀疏矩阵的“带行逻辑连接信息”的三元组存储类型

```
typedef int ElemType;  //矩阵元素类型
```

```
#define MAXSIZE 400  //稀疏矩阵非零元的最大个数
#define MAXRC 20     //稀疏矩阵的最大行数
typedef struct Triple{
    int i, j;        //该非零元的行下标和列下标
    ElemType e;      //该非零元的值
}Triple;             //非零元结点的类型定义
typedef struct{
    Triple data[MAXSIZE+1];  //非零元三元组表
    int rpos[MAXRC];         //各行第一个非零元的位置表
    int mu,nu,tu;            //矩阵的行数、列数和非零元个数
}RLSMatrix;                  //稀疏矩阵的类型定义
```

2. 稀疏矩阵的基本操作设定为：

```
int InitRLSMatrix(RLSMatrix *M);
int GetRLSMatrix(RLSMatrix *M, char *line, int start);
int PrintRLSMatrix(RLSMatrix M);
RLSMatrix addMatrix(RLSMatrix M, RLSMatrix N, int *error);
RLSMatrix subMatrix(RLSMatrix M, RLSMatrix N, int *error);
RLSMatrix multiplyMatrix(RLSMatrix M, RLSMatrix N, int *error);
```

其中部分操作的代码算法如下：

(1) int InitRLSMatrix(RLSMatrix *M);

```
int InitRLSMatrix(RLSMatrix *M)
{
// 初始化以"行逻辑链接的顺序表"表示的稀疏矩阵M
    int i;
    Triple init;                  //init表示行、列、值都为0的元素
    init.i = init.j = 0;
    init.e = 0;
    if(!M) return ERROR;          //若没分配空间，返回ERROR
    for(i=0; i<MAXSIZE+1; i++){   //M元素全部初始化为零元
        M->data[i] = init;
    }
    for(i=0; i<MAXRC+1; i++){     //各行第一个非零元位置表初始化为0
        M->rpos[i] = 0;
    }
    M->mu = M->nu = M->tu = 0;    //矩阵行数、列数、非零元数初始化0

    return OK;
}//InitRLSMatrix
```

(2) int GetRLSMatrix(RLSMatrix *M, char *line, int start);

```
int GetRLSMatrix(RLSMatrix *M, char *line, int start)
{
//根据读入计算表达式line和主函数中确定的矩阵起始位置start
//得到稀疏矩阵M，并返回

    int k;  //k表示当前字符在line中的下标
    int i,j,e,mu,nu,tu;
    int ispositive; //ispositive判断值的符号
    int s;  //s表示当前得到数据在三元组()的位置
    int isrpos;
    char c;

    k = start;

    //得到矩阵的行数和列数
    mu = 0; k++;
    while((c=*(line+k)) != ';'){     //mu读入结束
        if(c>='0' && c<='9') { mu = mu*10+c-'0'; }
        k++;
    }
    nu = 0; k++;
    while((c=*(line+k)) != ';'){     //nu读入结束
        if(c>='0' && c<='9') { nu = nu*10+c-'0'; }
        k++;
    }


    // 得到矩阵非零元的个数，和稀疏矩阵的元素（行逻辑排列）
    tu = 0; k = k+2;
    while((c=*(line+k)) != ']'){     //矩阵读入结束
        if(c == '('){            //一个三元组读入开始，初始化
            tu++;
            i = j = e = 0;
            s = 1;
            ispositive = 1;
        }else if(c == '-'){      //读到值为负的元素
            ispositive = -1;
        }else if(c == ','){      //i和j读入
            if(s==1)       M->data[tu].i = i;
            else if(s==2) M->data[tu].j = j;
            s++;
        }else if(c == ')'){      //e读入
            M->data[tu].e = ispositive*e;
        }else{                   //计算i,j,e
            if(s==1)       i = i*10+c-'0';
            else if(s==2) j = j*10+c-'0';
            else if(s==3) e = e*10+c-'0';
        }
        k++;
    }

    //将mu,nu,tu放入稀疏矩阵M中
    M->mu = mu; M->nu = nu; M->tu = tu;
```

```
    //得到稀疏矩阵M中的rpos
    int rposi[MAXRC+1];      //rposi表示每一行中非零元的个数
    for(k=0; k<=MAXRC; k++) //初始化rposi
        rposi[k] = 0;
    for(k=1; k<=tu; k++)     //计算每一行中非零元的个数
        rposi[M->data[k].i]++;

    for(i=1; i<=mu; i++)     //找到第一个非零元的行号
        if(rposi[i]){
            M->rpos[i]=1;
            break;
        }
    for(k=i+1; k<=mu; k++)   //根据rposi判断之后让rpos
        M->rpos[k] = M->rpos[k-1] + rposi[k-1];

    return OK;
}//GetRLSMatrix
```

(3) int PrintRLSMatrix(RLSMatrix M);

```
int PrintRLSMatrix(RLSMatrix M)
{
//输出稀疏矩阵
    int i,j,k;
    k = 1;
    //遍历稀疏矩阵M的每一行和列
    for(i=1; i<=M.mu; i++){
        for(j=1; j<=M.nu; j++){
            //判断当前位置是否有非零元
            if(M.data[k].i == i && M.data[k].j == j){
                fprintf(fout, "%d\t", M.data[k].e);
                k++;
            }else
                fprintf(fout, "%d\t", 0);
        }                    FILE* fout
        fprintf(fout, "\n");
    }

    fprintf(fout, "\n");
    return 0;
}//PrintRLSMatrix
```

(4) RLSMatrix addMatrix(RLSMatrix M, RLSMatrix N, int *error);

```
//输出矩阵的和
RLSMatrix addMatrix(RLSMatrix M, RLSMatrix N, int *error)
{
    int m=1, n=1, c=1, i
    RLSMatrix Q;
    InitRLSMatrix(&Q);

    //判断是否符合加法要求
    if(M.mu!=N.mu || M.nu!=N.nu)
        error=ADD_ERROR;

    Q.mu=M.mu;
    Q.nu=M.nu;
    //根据M和N中非零元，比较其行号和列号
    //判断M当前元和N当前元是否处于同一位置
    //若不同，则将M和N中靠前的元素赋值给Q
    //若相同，则进行加法运算
    while(m<=M.tu){
        while(n<=N.tu){
            if(M.data[m].i<N.data[n].i){
                Q.data[c].i = M.data[m].i;
                Q.data[c].j = M.data[m].j;
                Q.data[c].e = M.data[m].e;
                c++;m++;
                break;
            }else if(M.data[m].i>N.data[n].i){
                Q.data Triple struct5_RLSMatrix::data
                Q.data[c].j = N.data[n].j;
                Q.data[c].e = N.data[n].e;
                c++;n++;

            }else{
                if(M.data[m].j<N.data[n].j){
                    Q.data[c].i = M.data[m].i;
                    Q.data[c].j = M.data[m].j;
                    Q.data[c].e = M.data[m].e;
                    c++;m++;
                    break;
                }else if(M.data[m].j>N.data[n].j){
                    Q.data[c].i = N.data[n].i;
                    Q.data[c].j = N.data[n].j;
                    Q.data[c].e = N.data[n].e;
                    c++;n++;
                }else{
                    //将对应位置上的数相加，并将结果赋值给Q
                    Q.data[c].i = N.data[n].i;
                    Q.data[c].j = N.data[n].j;
                    Q.data[c].e = M.data[m].e+N.data[n].e;
                    //判断得到的数是否为0，若不为0，则记录
                    if(Q.data[c].e!=0){
                        c++;
                    }
                    m++;n++;
                    break;
                }
            }
        }
    }
```

```
    //若M或N中有剩余的元素，将其全部给Q
    if(m>M.tu&&n<=N.tu){
        for(i=n;i<=N.tu;i++){
            Q.data[c].i = N.data[n].i;
            Q.data[c].j = N.data[n].j;
            Q.data[c].e = N.data[n].e;
            c++;
        }
    }
    if(n>N.tu&&m<=M.tu){
        for(i=m;m<=M.tu;i++){
            Q.data[c].i = M.data[m].i;
            Q.data[c].j = M.data[m].j;
            Q.data[c].e = M.data[m].e;
            c++;
        }
    }

    //矩阵Q中元素的个数
    Q.tu = c-1;

    return Q;
}
```

(5) RLSMatrix subMatrix(RLSMatrix M, RLSMatrix N, int *error);
只需将加法操作转换为对应的减法操作。

(6) RLSMatrix multiplyMatrix(RLSMatrix M, RLSMatrix N, int *error);

```
//输出矩阵的成积
RLSMatrix multiplyMatrix(RLSMatrix M, RLSMatrix N, int *error)
{
    RLSMatrix Q;
                //行列不符合要求，返回错误
    if(M.nu != N.mu)
    {
        error = MULT_ERROR;
        return Q;
    }
                //Q的初始化
    Q.mu = M.mu;
    Q.nu = N.nu;
    Q.tu = 0;

    int ctemp[Q.nu];
    int tp,t,p,q,arow,brow,ccol;

    if(M.tu * N.tu != 0)//Q不是非零矩阵
    {
        for(arow = 1;arow <= M.mu;++arow)//处理M的每一行
        {
            for(int i = 0;i <= Q.nu;i++)//当前各行元素累加器清零
                ctemp[i]=0;

            Q.rpos[arow] = Q.tu + 1;

            if(arow < M.mu)//求出每一行非零元素个数的上界
                tp = M.rpos[arow + 1];
            else
                tp = M.tu+1;

            for(p = M.rpos[arow];p < tp;++p)//对当前行中每一个非零元找到对应元在N中的行号
            {
                brow = M.data[p].j;

                if(brow < N.mu)//求出矩阵N中第brow行非零元素个数的上界
                    t = N.rpos[brow + 1];
                else
                    t = N.tu+1;


                for(q = N.rpos[brow];q < t;++q)
                {
                    ccol = N.data[q].j;//乘积元素在Q中的列号

                    ctemp[ccol] += M.data[p].e * N.data[q].e;
                }//for q

            }//求得Q中第crow(=arow)行的非零元

            for(ccol = 1;ccol < Q.nu;++ccol)//压缩存储该行非零元
            {
                if(ctemp[ccol])
                {
                    if(++Q.tu > MAXSIZE)//如果Q中非零元个数超过MASIZE，则出错
                    {
                        error = MULT_ERROR;
                        return Q;
                    }

                    Q.data[Q.tu].i = arow;
                    Q.data[Q.tu].j = ccol;
                    Q.data[Q.tu].e = ctemp[ccol];

                }//if
            }
        }//for arow
    }//if
    return Q;
}//matrix_mult
```

3. 主函数和其他函数

```
//读入一行计算式
int mygetline(char *str, int lim)
{
    int len=0,c;
    while((--lim>0) && ((c=getc(fin))!=EOF) && (c!='\n'))
        str[len++]=c;
    if(c=='\n')
        str[len++]='\n';
    str[len]='\0';
    return len;
}
```

```
int main(void){
    fin = fopen("data.in","rb");
    fout = fopen("data.out","wb");

    int len, symposi, num;
    int error;
    char line[MAXLINE], symbol;
    RLSMatrix M,N,Q;

    num = 0;
    while((len=mygetline(line, MAXLINE))>2){//读入一行line，根据长度len判断是否是运算式
        num++;
        fprintf(fout, "第%d个矩阵计算式:\n", num); //输出计算式序号

        InitRLSMatrix(&M);      //初始化M和N
        InitRLSMatrix(&N);

        for(symposi=0; symposi<len; symposi++){ //判断运算符的位置symposi
            if(line[symposi]=='+' || line[symposi]=='-' || line[symposi]=='*')
                if(line[symposi-1]=='}' && line[symposi+1]=='{')    //判断不是负号
                    break;
        }
        symbol = line[symposi];     //得到运算符

        GetRLSMatrix(&M, line, 0);  //根据起始位置得到M和N，并输出
        PrintRLSMatrix(M);
        GetRLSMatrix(&N, line, symposi+1);
        PrintRLSMatrix(N);

        error = 0;
        switch(symbol){             //根据运算符调用函数并得到结果
            case '+': Q = addMatrix(M,N,&error); break;
            case '-': Q = subMatrix(M,N,&error); break;
            case '*': Q = multiplyMatrix(M,N,&error); break;
            default:  Q = addMatrix(M,N,&error); break;
        }

        if(error==0) PrintRLSMatrix(Q);     //判断运算能否进行，输出结果
        else fprintf(fout, "Error!\n");
        fprintf(fout, "\n");
    }

    fclose(fin);
    fclose(fout);
    return 0;
}
```

4. 本程序函数调用关系简单，不再赘述。

四、 调试分析

1. 调试过程中，对设计做了一些修改

（1）在做乘法时对 rpos 数组的值进行了修改，使得矩阵乘法能够支持前几行都为 0 的情况

（2）在做加减法时按照顺序处理三元组的运算，使得输出更加方便

（3）对输入输出进行了优化，使得该程序可以通过文件输入输出

2. 从本实验中容易看出，数组和矩阵三元组的运用较为广泛，且使用时操作较为方便。
3. 主要算法的时空分析：

（1）矩阵加减法的时间复杂度 O(M.tu+N.tu),空间复杂度 O(1)

（2）矩阵乘法的时间复杂度 O(M.mu*N.nu+M.tu*N.tu/N.mu),空间复杂度 O(M.mu)

五、 用户手册

1. 本程序的运行环境为 code::Blocks 17.12，执行文件为 Matrix.cbp。
2. 用户可在 datain.in 中输入矩阵计算式，在 dataout.out 中得到输出结果。
3. 一个稀疏矩阵表示如下：{mu;nu;[(i1,j1,e1);(i2,j2,e3);…;(itu,jtu,etu)]}；两个矩阵之间用运算符+-*直接连接。

六、 测试结果

（1）输入文件：

```
{3;3;[(1,1,10);(2,3,9);(3,1,-1)]}+{3;3;[(2,3,-1);(3,1,1);(3,3,-3)]}
{3;2;[(1,1,10);(2,2,9);(3,1,-1)]}-{3;2;[(2,2,-1);(3,1,1);(3,2,-3)]}
{4;5;[(1,1,4);(1,2,-3);(1,5,1);(2,4,8);(3,3,1);(4,5,70)]}*{5;3;[(1,1,3);(2,1,4);(2,2,2);(3,2,1);(4,1,1)]}
```

（2）输出文件：

```
第1个矩阵计算式:
10  0   0
0   0   9
-1  0   0

0   0   0
0   0   -1
1   0   -3

10  0   0
0   0   8
0   0   -3

第2个矩阵计算式:
10  0
0   9
-1  0

0   0
0   -1
1   -3

10  0
0   10
-2  3

第3个矩阵计算式:
4   -3  0   0   1
0   0   0   8   0
0   0   1   0   0
0   0   0   0   70

3   0   0
4   2   0
0   1   0
1   0   0
0   0   0

0   -6  0
8   0   0
0   1   0
0   0   0
```

七、 附录

源程序文件名清单：

```
Head.h          //本程序头文件，包含数据结构定义个函数声明；
Matrix.c        //主程序文件
```