

实验报告 (week-06)
2020111235 马靖淳

一. 实验任务

1. C 语言可变参数
2. 三元组顺序表
3. 采用三元组顺序表求转置矩阵
4. 改进的稀疏矩阵转置算法

二. 实验上机时间

三. 知识点

1. C 语言可变参数
 - 1) stdarg.h
 - 2) va_list
 - 3) va_start
 - 4) va_arg
 - 5) va_end
2. 三元组顺序表
稀疏矩阵的压缩存储
3. 采用三元组顺序表求转置矩阵
4. 改进的稀疏矩阵转置算法

四. 源代码及结果截屏

1. C 语言可变参数

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdarg.h>
#include <stdio.h>
#include "String.h"

int main()
{
    int mm;
    mm = max(6, 20, 40, 10, 50, 30, 40);
    printf("最大值为: %d\n", mm);


    system("pause");
    return 0;
}

int max(int n, ...)
{
    //定参n表示后面变参数量, 定界用
    va_list ap; //定义一个va_list 指针来访问参数表
    va_start(ap, n); //初始化ap, 指向第一个变参
    int maximum = 0;
    int temp = 0;
    for (int i = 0; i < n; i++)
```

```

{
    //获取一个int型参数，并且ap指向下一个参数
    temp = va_arg(ap, int);
    if (maximum < temp)
        maximum = temp;
}
va_end(ap); //结束关闭ap
return maximum;
}

```

 E:\大二上\数据结构\代码保存\va_start\Debug\va_start.exe

最大值为：50
请按任意键继续. . .

2. 三元组顺序表

1) SparseMatrix.h

```

#ifndef SPARSEMATRIX_H
#define SPARSEMATRIX_H

#include<string.h>
#include<malloc.h> // malloc()等
#include<stdio.h> // EOF(=^Z或F6), NULL
#include<stdlib.h> // atoi()
#include<math.h> // floor(), ceil(), abs()

#define MAXSIZE 10000

//三元组类型
typedef int Status;
typedef int ElemType;
typedef struct
{
    int i, j; //该非零元的行下标和列下标
    ElemType e; //该非零元的值
}Triple;

//稀疏矩阵类型
typedef struct
{
    Triple data[MAXSIZE + 1]; //data[0]未用 非零元三元组表
    int mu, nu, tu; //矩阵行数mu，列数nu，非零元个数tu
}TSMatrix;

Status CreateSMatrix(TSMatrix* M); //创建稀疏矩阵M
Status DestroySMatrix(TSMatrix* M); //销毁稀疏矩阵M

```

```

Status PrintSMatrix(TSMatrix M); //输出稀疏矩阵M
Status CopySMatrix(TSMatrix M, TSMatrix* T); //由稀疏矩阵M复制得到T
int comp(int c1, int c2);
Status AddSMatrix(TSMatrix M, TSMatrix N, TSMatrix* Q); //求稀疏矩阵的和Q=M+N
Status SubSMatrix(TSMatrix M, TSMatrix N, TSMatrix* Q); //求稀疏矩阵的差Q=M-N
Status MultSMatrix(TSMatrix M, TSMatrix N, TSMatrix* Q); //求稀疏矩阵的乘积Q=M*N
Status TransposeSMatrix(TSMatrix M, TSMatrix* T); //求稀疏矩阵M的转置矩阵T

#endif

```

2) SparseMatrix.c

```

#define _CRT_SECURE_NO_WARNINGS
#include "SparseMatrix.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

Status CreateSMatrix(TSMatrix* M)
{
    printf("请输入行数、列数和非零元的个数: ");
    scanf("%d%d%d", &(*M).mu, &(*M).nu, &(*M).tu);
    int k;
    printf("请依次输入%d个非零元的行下标、列下标和值: \n", (*M).tu);
    for (k = 1; k <= (*M).tu; k++)
        scanf("%d%d%d", &((*M).data[k].i), &((*M).data[k].j), &((*M).data[k].e));
    printf("\n");
    return OK;
}

Status DestroySMatrix(TSMatrix* M)
{
    (*M).mu = 0;
    (*M).nu = 0;
    (*M).tu = 0;
    return OK;
}

Status PrintSMatrix(TSMatrix M)
{
    // 输出稀疏矩阵M
    int i;
    printf("%d行%d列%d个非零元素\n", M.mu, M.nu, M.tu);
    printf("行 列 元素值\n");
    for (i = 1; i <= M.tu; i++)

```

```

        printf("%2d%4d%8d\n", M.data[i].i, M.data[i].j, M.data[i].e);
    printf(" \n");
    return OK;
}

Status CopySMatrix(TSMatrix M, TSMatrix* T)
{
    // 由稀疏矩阵M复制得到T
    (*T) = M;
    return OK;
}

int comp(int c1, int c2)
{ // AddSMatrix函数要用到
    int i;
    if (c1 < c2)
        i = 1;
    else if (c1 == c2)
        i = 0;
    else
        i = -1;
    return i;
}

Status AddSMatrix(TSMatrix M, TSMatrix N, TSMatrix* Q)
{
    // 求稀疏矩阵的和Q=M+N
    Triple* Mp, * Me, * Np, * Ne, * Qh, * Qe;
    if (M.mu != N.mu)
        return ERROR;
    if (M.nu != N.nu)
        return ERROR;
    (*Q).mu = M.mu;
    (*Q).nu = M.nu;
    Mp = &M.data[1]; // Mp的初值指向矩阵M的非零元素首地址
    Np = &N.data[1]; // Np的初值指向矩阵N的非零元素首地址
    Me = &M.data[M.tu]; // Me指向矩阵M的非零元素尾地址
    Ne = &N.data[N.tu]; // Ne指向矩阵N的非零元素尾地址
    Qh = Qe = (*Q).data; // Qh、Qe的初值指向矩阵Q的非零元素首地址的前一地址
    while (Mp <= Me && Np <= Ne)
    {
        Qe++;
        switch (comp(Mp->i, Np->i))
        {

```

```

case 1:
    *Qe = *Mp;
    Mp++;
    break;
case 0:
    switch (comp(Mp->j, Np->j)) // M、N矩阵当前非零元素的行相等, 继续比较列
    {
        case 1:
            *Qe = *Mp;
            Mp++;
            break;
        case 0:
            *Qe = *Mp;
            Qe->e += Np->e;
            if (!Qe->e) // 元素值为0, 不存入压缩矩阵
                Qe--;
            Mp++;
            Np++;
            break;
        case -1:
            *Qe = *Np;
            Np++;
    }
    break;
case -1:
    *Qe = *Np;
    Np++;
    }
}

if (Mp > Me) // 矩阵M的元素全部处理完毕
    while (Np <= Ne)
    {
        Qe++;
        *Qe = *Np;
        Np++;
    }

if (Np > Ne) // 矩阵N的元素全部处理完毕
    while (Mp <= Me)
    {
        Qe++;
        *Qe = *Mp;
        Mp++;
    }

(*Q).tu = Qe - Qh; // 矩阵Q的非零元素个数

```

```

        return OK;
    }

Status SubSMatrix(TSMatrix M, TSMatrix N, TSMatrix* Q)
{
    // 求稀疏矩阵的差Q=M-N
    int i;
    for (i = 1; i <= N.tu; i++)
        N.data[i].e *= -1;
    AddSMatrix(M, N, Q);
    return OK;
}

Status MultSMatrix(TSMatrix M, TSMatrix N, TSMatrix* Q)
{
    // 求稀疏矩阵的乘积Q=M*N
    int i, j;
    int h = M.mu;
    int l = N.nu; // h, l分别为矩阵Q的行、列值
    int Qn = 0; // Qn为矩阵Q的非零元素个数，初值为0
    ElemType* Qe;
    if (M.nu != N.mu)
        return ERROR;
    (*Q).mu = M.mu;
    (*Q).nu = N.nu;
    Qe = (ElemType*)malloc(h * l * sizeof(ElemType)); // Qe为矩阵Q的临时数组
    // 矩阵Q的第i行j列的元素值存于*(Qe+(i-1)*l+j-1)中，初值为0
    for (i = 0; i < h * l; i++)
        *(Qe + i) = 0; // 赋初值0
    for (i = 1; i <= M.tu; i++) // 矩阵元素相乘，结果累加到Qe
        for (j = 1; j <= N.tu; j++)
            if (M.data[i].j == N.data[j].i)
                *(Qe + (M.data[i].i - 1) * l + N.data[j].j - 1) += M.data[i].e *
N.data[j].e;
    for (i = 1; i <= M.mu; i++)
        for (j = 1; j <= N.nu; j++)
            if (*(Qe + (i - 1) * l + j - 1) != 0)
            {
                Qn++;
                (*Q).data[Qn].e = *(Qe + (i - 1) * l + j - 1);
                (*Q).data[Qn].i = i;
                (*Q).data[Qn].j = j;
            }
    free(Qe);
}

```

```

        (*Q).tu = Qn;
    return OK;
}

Status TransposeSMatrix(TSMatrix M, TSMatrix* T)
{
    //按照矩阵M的列序进行转置
    int p, q, col;
    (*T).mu = M.nu;
    (*T).nu = M.mu;
    (*T).tu = M.tu;
    if ((*T).tu)
    {
        q = 1;
        for(col=1;col<M.nu;++col)
            for(p=1;p<=M.tu;++p)
                if (M.data[p].j == col)
                {
                    (*T).data[q].i = M.data[p].j;
                    (*T).data[q].j = M.data[p].i;
                    (*T).data[q].e = M.data[p].e;
                    ++q;
                }
    }
    return OK;
}

```

3) test.c

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "SparseMatrix.h"

int main()
{
    TSMatrix A, B, C;
    //CreateSMatrix test
    printf("创建矩阵A: ");
    CreateSMatrix(&A);
    //PrintSMatrix test
    printf("A矩阵为:");
    PrintSMatrix(A);
    //CopySMatrix test
    printf("由矩阵A复制矩阵B: ");
    CopySMatrix(A, &B);
    PrintSMatrix(B);
}

```

```

//DestroySMatrix test
DestroySMatrix(&B);
printf("销毁矩阵B后:\n");
PrintSMatrix(B);
//AddSMatrix test
printf("创建矩阵B2: (与矩阵A的行、列数相同, 行、列分别为%d,%d)\n", A.mu, A.nu);
CreateSMatrix(&B);
PrintSMatrix(B);
printf("矩阵C1 (A+B): ");
AddSMatrix(A, B, &C);
PrintSMatrix(C);
DestroySMatrix(&C);
//SubtSMatrix test
printf("矩阵C2 (A-B): ");
SubSMatrix(A, B, &C);
PrintSMatrix(C);
DestroySMatrix(&C);
//TransposeSMatrix test
printf("矩阵C3 (A的转置): ");
TransposeSMatrix(A, &C);
PrintSMatrix(C);
//MultSMatrix test
DestroySMatrix(&A);
DestroySMatrix(&B);
DestroySMatrix(&C);
printf("创建矩阵A2: ");
CreateSMatrix(&A);
PrintSMatrix(A);
printf("创建矩阵B3: (行数应与矩阵A2的列数相同=%d)\n", A.nu);
CreateSMatrix(&B);
PrintSMatrix(B);
printf("矩阵C5 (A*B): ");
MultSMatrix(A, B, &C);
PrintSMatrix(C);
DestroySMatrix(&A);
DestroySMatrix(&B);
DestroySMatrix(&C);

system("pause");
return 0;
}

```


E:\大二上\数据结构\代码保存\SparseMatrix\Debug\SparseMatrix.exe

创建矩阵A: 请输入行数、列数和非零元的个数: 3 4 2
请依次输入2个非零元的行下标、列下标和值:

1 1 12
2 3 9

A矩阵为:3行4列2个非零元素

行	列	元素值
1	1	12
2	3	9

由矩阵A复制矩阵B: 3行4列2个非零元素

行	列	元素值
1	1	12
2	3	9

销毁矩阵B后:

0行0列0个非零元素

行	列	元素值
---	---	-----

创建矩阵B2: (与矩阵A的行、列数相同, 行、列分别为3, 4)

请输入行数、列数和非零元的个数: 3 4 4

请依次输入4个非零元的行下标、列下标和值:

1 1 4
2 2 5
3 3 7
1 4 5

3行4列4个非零元素

行	列	元素值
1	1	4

E:\大二上\数据结构\代码保存\SparseMatrix\Debug\SparseMatrix.exe

2	2	5
3	3	7
1	4	5

矩阵C1 (A+B): 3行4列5个非零元素

行	列	元素值
1	1	16
2	2	5
2	3	9
3	3	7
1	4	5

矩阵C2 (A-B): 3行4列5个非零元素

行	列	元素值
1	1	8
2	2	-5
2	3	9
3	3	-7
1	4	-5

矩阵C3 (A的转置): 4行3列2个非零元素

行	列	元素值
1	1	12
3	2	9

创建矩阵A2: 请输入行数、列数和非零元的个数: 3 4 4

请依次输入4个非零元的行下标、列下标和值:

1 1 4
2 3 6
3 3 1

```

E:\大二上\数据结构\代码保存\SparseMatrix\Debug\Sparse1
2 4 5

3行4列4个非零元素
行 列 元素值
1 1 4
2 3 6
3 3 1
2 4 5

创建矩阵B3: (行数应与矩阵A2的列数相同=4)
请输入行数、列数和非零元的个数: 4 2 2
请依次输入2个非零元的行下标、列下标和值:
3 2 2
4 2 1

4行2列2个非零元素
行 列 元素值
3 2 2
4 2 1

矩阵C5 (A*B): 3行2列2个非零元素
行 列 元素值
2 2 17
3 2 2

请按任意键继续. . .

```

3. 采用三元组顺序表求转置矩阵

```

Status TransposeSMatrix(TSMatrix M, TSMatrix* T)
{
    //按照矩阵M的列序进行转置
    int p, q, col;
    (*T).mu = M.nu;
    (*T).nu = M.mu;
    (*T).tu = M.tu;
    if ((*T).tu)
    {
        q = 1;
        for(col=1; col<M.nu; ++col)
            for(p=1; p<=M.tu; ++p)
                if (M.data[p].j == col)
                {
                    (*T).data[q].i = M.data[p].j;
                    (*T).data[q].j = M.data[p].i;
                    (*T).data[q].e = M.data[p].e;
                    ++q;
                }
    }
    return OK;
}

```

在 2.中已经检验过了，把截图又剪辑了一下见下方，结果是正确的

E:\大二上\数据结构\代码保存\SparseMatrix\Debug\SparseMatrix.exe

创建矩阵A: 请输入行数、列数和非零元的个数: 3 4 2
请依次输入2个非零元的行下标、列下标和值:

1 1 12
2 3 9

矩阵C3(A的转置): 4行3列2个非零元素

行	列	元素值
1	1	12
3	2	9

4. 改进的稀疏矩阵转置算法

```
Status FastTransposeSMatrix(TSMatrix M, TSMatrix* T)
{
    //采用三元组顺序表存储表示, 求稀疏矩阵M的转置矩阵T
    int col, t, p, q;
    int num[20], cpot[20];
    (*T).mu = M.nu;
    (*T).nu = M.mu;
    (*T).tu = M.tu;
    if ((*T).tu)
    {
        for (col = 1; col <= M.nu; ++col)
            num[col] = 0;
        for (t = 1; t <= M.tu; ++t) //求M中每一列非零元个数
            ++num[M.data[t].j];
        cpot[1] = 1; //求M中每列第一个非零元在T.data中的序号
        for (col = 2; col <= M.nu; ++col)
            cpot[col] = cpot[col - 1] + num[col - 1];
        for (p = 1; p <= M.tu; ++p)
        {
            col = M.data[p].j;
            q = cpot[col];
            (*T).data[q].i = M.data[p].j;
            (*T).data[q].j = M.data[p].i;
            (*T).data[q].e = M.data[p].e;
            ++cpot[col];
        } //for
    } //if
    return OK;
} //FastTransposeSMatrix
//时间复杂度为: O(nu+tu)

test.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "SparseMatrix.h"
```

```

int main()
{
    TSMatrix A, B, C;
    //CreateSMatrix test
    printf("创建矩阵A: ");
    CreateSMatrix(&A);
    //PrintSMatrix test
    printf("A矩阵为:");
    PrintSMatrix(A);
    //TransposeSMatrix test
    printf("TransposeSMatrix矩阵B(A的转置): ");
    TransposeSMatrix(A, &B);
    PrintSMatrix(B);
    //FastTransposeSMatrix test
    printf("FastTransposeSMatrix test矩阵C(A的转置): ");
    FastTransposeSMatrix(A, &C);
    PrintSMatrix(C);

    DestroySMatrix(&A);
    DestroySMatrix(&B);
    DestroySMatrix(&C);

    system("pause");
    return 0;
}

```

C:\E:\大二上\数据结构\代码保存\SparseMatrix\Debug\SparseMatrix.exe

```

创建矩阵A: 请输入行数、列数和非零元的个数: 3 4 2
请依次输入2个非零元的行下标、列下标和值:
1 1 9
2 3 6

A矩阵为:3行4列2个非零元素
行 列 元素值
1 1 9
2 3 6

TransposeSMatrix矩阵B(A的转置): 4行3列2个非零元素
行 列 元素值
1 1 9
3 2 6

FastTransposeSMatrix test矩阵C(A的转置): 4行3列2个非零元素
行 列 元素值
1 1 9
3 2 6

请按任意键继续. . .

```

上面截图体现两种算法均可实现，下面列举不同情况说明各算法优势，并计算程序运行时间

test.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "SparseMatrix.h"
#include <windows.h>
#include <time.h>

int main()
{
    TSMatrix A, B, C;
    //CreateSMatrix test
    printf("创建矩阵A: ");
    CreateSMatrix(&A);
    //PrintSMatrix test
    printf("A矩阵为:");
    PrintSMatrix(A);

    //TransposeSMatrix test
    printf("TransposeSMatrix矩阵B(A的转置): ");
    long op, ed; //定义开始时间和结束时间
    op = clock(); //开始计时
    for (int i = 0; i < 10000; i++)
    {
        TransposeSMatrix(A, &B);
    }
    Sleep(1000); //静态方法，控制当前正在运行的线程
    ed = clock(); //结束计时
    printf("运行时间为%ldms\n", ed - op); //运行时间
    PrintSMatrix(B);

    //FastTransposeSMatrix test
    printf("FastTransposeSMatrix test矩阵C(A的转置): ");

    op = clock(); //开始计时
    for (int i = 0; i < 10000; i++)
    {
        FastTransposeSMatrix(A, &C);
    }
    Sleep(1000); //静态方法，控制当前正在运行的线程
    ed = clock(); //结束计时
    printf("运行时间为%ldms\n", ed - op); //运行时间
```

```
PrintSMatrix(C);
```

```
DestroySMatrix(&A);
```

```
DestroySMatrix(&B);
```

```
DestroySMatrix(&C);
```

```
system("pause");
```

```
return 0;
```

```
}
```

E:\大二上\数据结构\代码保存\SparseMatrix\Debug\SparseMatrix.exe

创建矩阵A: 请输入行数、列数和非零元的个数: 3 4 2
请依次输入2个非零元的行下标、列下标和值:

1 1 9

3 3 7

A矩阵为:3行4列2个非零元素

行	列	元素值
1	1	9
3	3	7

TransposeSMatrix矩阵B(A的转置): 运行时间为1030ms

4行3列2个非零元素

行	列	元素值
1	1	9
3	3	7

FastTransposeSMatrix test矩阵C(A的转置): 运行时间为1028ms

4行3列2个非零元素

行	列	元素值
1	1	9
3	3	7

请按任意键继续. . .

当矩阵中元素个数较少时, fast 算法优势体现

E:\大二上\数据结构\代码保存\SparseMatrix\Debug\SparseMatrix.exe

创建矩阵A: 请输入行数、列数和非零元的个数: 1000 500 2

请依次输入2个非零元的行下标、列下标和值:

798 213 7

3 7 1

A矩阵为:1000行500列2个非零元素

行	列	元素值
798	213	7
3	7	1

TransposeSMatrix矩阵B(A的转置): 运行时间为1043ms

500行1000列2个非零元素

行	列	元素值
7	3	1
213	798	7

FastTransposeSMatrix test矩阵C(A的转置):

出

五. 实验总结

1. #include <stdarg.h>

1) stdarg.h

是 C 语言中 C 标准函数库的头文件, stdarg 是由 standard (标准) arguments (参数) 简化而来, 主要目的为让函数能够接收可变参数。C++ 的 cstdarg 头文件中也提供这样的功能; 虽然与 C 的头文件是兼容的, 但是也有冲突存在。

2) va_list

是在 C 语言中解决变参问题的一组宏, 所在头文件: #include <stdarg.h>, 用于获取不确定个数的参数, va_list 是一个堆栈结构体, 准备接收"...接收到的参数!

3) va_start

函数名称, 读取可变参数的过程其实就是在堆栈中, 使用指针, 遍历堆栈段中的参数列表, 从低地址到高地址一个一个地把参数内容读出来的过程, va_start(va_list 类型变量, 参数个数); 初始化 va_list 结构体堆栈的深度!

4) va_arg

va_arg (va_list 类型变量, 数据类型); 出栈符合该数据类型的下一个参数!

5) va_end

va_end (va_list 类型变量); 释放 va_list 类型的堆栈空间!

6) 记住 va_arg 这个函数, 这个 stdarg.h 是解决函数传递不确定个参数而生的, 我们知道可以为 main 函数传递不确定个数的参数, 存放在 argv[] 里面, 个数就是 argc, 但是普通的函数于 main 函数不一样, 如果需要传递不确定个参数进去, 需要这个头文件, 里面就只有三个函数, 一个是初始化堆栈的深度(用来存储传递进来的不确定个数的参数), 函数是 va_start; 另一个是出栈 va_arg; 最后一个是释放堆栈 va_end

2. 警告 C6262: 函数使用了堆栈中的 <constant> 个字节: 超过了 /analyze:stacksize<constant>。请考虑将某些数据移到堆中

此警告指出在函数内检测到了超出预设阈值的堆栈使用率。默认情况下, 当堆栈大小超过 16K 字节时会生成此警告。堆栈是有限的, 甚至在用户模式下也是如此, 如果无法提交堆栈页, 会导致堆栈溢出异常。_resetstkoflw 函数可以将系统从堆栈溢出的情况恢复为正常, 从而使程序得以继续运行, 而不会由于出现异常错误而失败。如果未调用 _resetstkoflw 函数, 则在上一个异常后不会显示保护页。当下次发生堆栈溢出时, 根本不会显示异常, 进程将在没有任何警告的情况下终止。

若要更正此警告, 1. 既可以将一些数据移至堆, 2. 也可以增加堆栈大小。无论采取哪一种更正措施, 在对代码进行更改前均应考虑到所用方法的利弊

3. 0x00693037 处有未经处理的异常(在 SparseMatrix.exe 中): 0xC00000FD: Stack

overflow (参数: 0x00000000, 0x01062000)。

按照错误提示我们可以知道该错误是“栈溢出”，回头一看可能出错的地方也就是在定义数组的地方了，当把预定义的数组大小改成 10000 之后，程序居然可以运行了，由此看来是我们定义的数组太大了，所以在定义数组的时候一定要谨记千万别太大，如果非要用容量比较大的数组的话建议使用 new 进行分配，然后在函数返回时记得 delete 就行了。