**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

**一、调试成功程序及说明**

**1、**

**题目：**输入稀疏矩阵，建立稀疏矩阵三元组顺序结构，实现矩阵的列序遍历转置和快速转置算法。

**算法思想：**类似顺序表的方法建立稀疏矩阵三元组顺序结构，结合书本可知列序遍历转置在于遍历比较，找出对应元素并改变；快速转置借助两个辅助数组，直接将各三元组放到转置后的新位置。

**运行结果：**输入三元组结构稀疏矩阵，输出逆置后的矩阵。

**结果分析：**正确

**附源程序。**

//列序遍历转置

#include<iostream>

using namespace std;

#include<stdio.h>

#include<stdlib.h>

#define OK 1

#define MAXSIZE 256

typedef int Status;

typedef int ElemType;

typedef struct

{

int i,j;

ElemType v;

}Triple;

typedef struct

{

Triple arr[MAXSIZE];

int Rows,Cols,Nums;

}SqSMatrix;

//创建矩阵

Status CreateTSMatrix(SqSMatrix &M)

{

int i,m,n;

ElemType e;

cout<<"请输入矩阵的行数,列数,非零元素数："<<endl;

cin>>M.Rows>>M.Cols>>M.Nums;

for(i=0;i<M.Nums;i++)

{

cout<<"请依次输入第"<<i+1<<"个非零元的行下标、列下标和值"<<endl;

cin>>m>>n>>e;

M.arr[i].i=m;

M.arr[i].j=n;

M.arr[i].v=e;

}

}

//输出矩阵

Status PrintTSMatrix(SqSMatrix M)

{

if(M.Rows==0||M.Cols==0)

{

printf("矩阵为空！\n");

return OK;

}

cout<<"行数为"<<M.Rows <<"，列数为"<<M.Cols<<endl;

if(M.Nums==0)

{

printf("矩阵为零矩阵！\n");

return OK;

}

for(int k=0;k<M.Nums;k++)

cout<<M.arr[k].i<<" "<<M.arr[k].j<<" "<<M.arr[k].v<<endl;

return OK;

}

Status TransposeTSMatrix(SqSMatrix A,SqSMatrix &B)

{

B.Rows = A.Cols ;

B.Cols = A.Rows ;

B.Nums = A.Nums ;

if(A.Nums >0)

{

int q=0;

for(int k=0;k<A.Cols ;k++)

{

for(int p=0;p<A.Nums ;p++)

{

if(A.arr[p].j == k )

{

B.arr[q].i = A.arr[p].j ;

B.arr[q].j = A.arr[p].i ;

B.arr[q].v = A.arr[p].v ;

q++;

}

}

}

}

return OK;

}

int main()

{

SqSMatrix M,T;

CreateTSMatrix(M);

cout<<"原矩阵为："<<endl;

PrintTSMatrix(M);

TransposeTSMatrix(M,T);

cout<<"转置后的矩阵为："<<endl;

PrintTSMatrix(T);

return 0;

}

//快速转置算法

#include<iostream>

using namespace std;

#include<stdio.h>

#include<stdlib.h>

#define OK 1

#define MAXSIZE 256

typedef int Status;

typedef int ElemType;

typedef struct

{

int i,j;

ElemType v;

}Triple;

typedef struct

{

Triple arr[MAXSIZE];

int Rows,Cols,Nums;

}SqSMatrix;

//创建矩阵

Status CreateTSMatrix(SqSMatrix &M)

{

int i,m,n;

ElemType e;

cout<<"请输入矩阵的行数,列数,非零元素数："<<endl;

cin>>M.Rows>>M.Cols>>M.Nums;

for(i=0;i<M.Nums;i++)

{

cout<<"请依次输入第"<<i+1<<"个非零元的行下标、列下标和值"<<endl;

cin>>m>>n>>e;

M.arr[i].i=m;

M.arr[i].j=n;

M.arr[i].v=e;

}

}

//输出矩阵

Status PrintTSMatrix(SqSMatrix M)

{

if(M.Rows==0||M.Cols==0)

{

printf("矩阵为空！\n");

return OK;

}

cout<<"行数为"<<M.Rows <<"，列数为"<<M.Cols<<endl;

if(M.Nums==0)

{

printf("矩阵为零矩阵！\n");

return OK;

}

for(int k=0;k<M.Nums;k++)

cout<<M.arr[k].i<<" "<<M.arr[k].j<<" "<<M.arr[k].v<<endl;

return OK;

}

Status FastTransposeSMatrix(SqSMatrix A, SqSMatrix &B)

{

int rowNum[MAXSIZE],rowStart[MAXSIZE];

B.Rows = A.Cols ;

B.Cols = A.Rows ;

B.Nums = A.Nums ;

if(A.Nums >0)

{

for(int k=0;k<A.Cols ;k++)

rowNum[k]=0;

for(int p=0;p<A.Nums ;p++)

rowNum[A.arr[p].j ]++;

rowStart[0]=0;

for(int k=1;k<A.Cols ;k++)

rowStart[k]=rowStart[k-1]+rowNum[k-1];

for(int p=0;p<A.Nums ;p++)

{

int q=rowStart[A.arr[p].j ];

B.arr[q].i = A.arr[p].j ;

B.arr[q].j = A.arr[p].i ;

B.arr[q].v = A.arr[p].v ;

rowStart[A.arr[p].j ]++;

}

}

return OK;

}

int main()

{

SqSMatrix M,T;

CreateTSMatrix(M);

cout<<"原矩阵为："<<endl;

PrintTSMatrix(M);

FastTransposeSMatrix(M,T);

cout<<"转置后的矩阵为："<<endl;

PrintTSMatrix(T);

return 0;

}

**2、**

**题目：**求矩阵的马鞍点。（书P69 7）

**算法思想：**二维数组存储矩阵，遍历先比较出每行的最小值，再在最小值所处列找出列最大值，若行最小值与列最大值相等，则为鞍点。

**运行结果：**输入矩阵，得到矩阵的鞍点位置或无鞍点。

**结果分析：**正确

**附源程序。**

#include <iostream>

using namespace std;

#include<stdio.h>

#include<string.h>

int main()

{

int a[100][100];

int m,n;

int row = 0,col = 0;

cout<<"请输入矩阵的长和宽（行\*列）：";

cin>>m>>n;

cout<<endl;

cout<<"请输入矩阵的元素："<<endl;

for(int i = 0; i < m; i++)

{

for(int j = 0; j < n;j++)

{

cin>>a[i][j];

}

}

int max,min;

for(int i = 0; i < m; i++)

{

row=0;

col=0;

min = a[i][0];

for(int j = 0; j <n;j++)

{

if(a[i][j] < min)

{

min = a[i][j];

col = j;

}

}

max = a[0][col];

for(int k=0; k < m; k++)

{

if( a[k][col] > max )

{

max = a[k][col];

row = k;

}

}

if( min == max )

{

cout<<"此矩阵鞍点在第"<<row+1<<"行，第"<<col+1<<"列"<<endl;

break;

}

}

if(min!=max)

cout<<"无鞍点"<<endl;

}

**3、**

**题目：**CSP题目 小明回家

**算法思想：**累加计算所用时长，关键在于lasttime对每次到达红绿灯时只需计算最后一轮的时间，用取余来实现。取余之后只需分析出所处的红绿灯区间，在各区间上只需计算一轮红绿灯运行时长，较为方便。

**运行结果：**输入路段与红绿灯状态及时长，得出回家所用时间。

**结果分析：**正确

**附源程序。**

#include <iostream>

using namespace std;

int main()

{

int r,y,g;

int n,k,t;

int time=0,lasttime;

cin>>r>>y>>g;

cin>>n;

while(n)

{

cin>>k>>t;

if(k==0)

time+=t;

if(k==1)

{

lasttime=time%(r+g+y);

if(lasttime<t)

time+=t-lasttime;

if(lasttime >= t && (lasttime< t+g))

{

}

if(lasttime >= t+g && lasttime < t+g+y)

time += t+g+y-lasttime+r;

if(lasttime >= t+g+y && lasttime < r+g+y)

time += r+g+y+t-lasttime;

}

if(k==2)

{

lasttime=time%(r+g+y);

if(lasttime<t)

time+=t-lasttime+r;

if(lasttime >= t && (lasttime< t+r))

time += t+r-lasttime;

if(lasttime >= t+r && lasttime < t+r+g)

{

}

if(lasttime >= t+r+g && lasttime < r+g+y)

time += r+g+y+t-lasttime+r;

}

if(k==3)

{

lasttime=time%(r+g+y);

if(lasttime<t)

{

}

if(lasttime >= t && (lasttime< t+y))

time += t+y-lasttime+r;

if(lasttime >= t+y && lasttime < t+r+y)

time += t+r+y-lasttime;

if(lasttime >= t+r+y && lasttime < r+g+y)

{

}

}

n--;

}

cout<<time;

return 0;

}

**二、未调试成功程序及说明**

**1、**

**题目：**CSP题目 简单座位分配

**算法思想：**利用顺序表结构存储座位信息，每5个座位用Group区分开来用seat数组存储，用遍历按编号从小到大分配座位，关键在于remainder信息，为分配座位提供极大便利。

**错误原因：**做题量较少，没有解题思路，尤其是remainder记录剩余座位这一点没有想到。

**正确代码：**

#include <stdio.h>

#define N 20

#define m 5

typedef struct seat{

int num;//编号

int val;//状态

}Seat;

typedef struct Group{

Seat seat[m];

int remainder;//组内剩余座位

}Seat\_Group;

void Initial\_Seat(Seat\_Group G[],int GroupNum, int seat\_num )//初始化座位

{

int i,j;

for( i = 0; i < GroupNum; i ++)

{

G[i].remainder = 5;

for( j = 0; j < seat\_num; j ++)

{

G[i].seat[j].num = 5\*i+j+1;

G[i].seat[j].val = 0;

}

}

}

int main()

{

int n,i,j,k;

int mark = 0; //标记已已经排完的组号

Seat\_Group G[N];

Initial\_Seat(G,N,m);

scanf("%d",&n);

int a[n];

for( i = 0; i < n; i ++)

{

scanf("%d",&a[i]);

}

for( i = 0; i < n; i ++)

{

int flag = 1; //指示是否找到组

for( j = 0; j < N && flag; j ++) //寻找相邻的座位

{

if(a[i] <= G[j].remainder)

{

flag = 0;

int cnt = 0;

for( k = 0; k < m; k++)

{

if(G[j].seat[k].val == 0)

{

cnt ++;

G[j].seat[k].val = 1;

if(cnt == 1)

{

printf("%d",G[j].seat[k].num);

}

else{

printf(" %d",G[j].seat[k].num);

}

}

if(cnt == a[i])

{

a[i] = 0; //表示分配完了

G[j].remainder -= cnt;

break;

}

}

}

}

if(a[i] != 0) //无法相邻的情况

{

for( j = mark; j < N && a[i]; j ++)

{

if(G[j].remainder != 0)

{

for(k = 0; k < m && a[i]; k ++)

{

if(G[j].seat[k].val == 0)

{

G[j].seat[k].val = 1;

G[j].remainder --;

printf("%d ",G[j].seat[k].num);

a[i] --;

}

}

}

}

mark = j - 1;

}

if(i != n )

{

printf("\n");

}

}

return 0;

}

**三、代码行数及小结**

行数：合计400行左右。

小结：此次上机重点在于矩阵，与之前的线性表、数组有很多联系之处，有一定的书本给出的代码，要求我们实现。两道CSP题比较绕，预示着我需要增加做题量。