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：

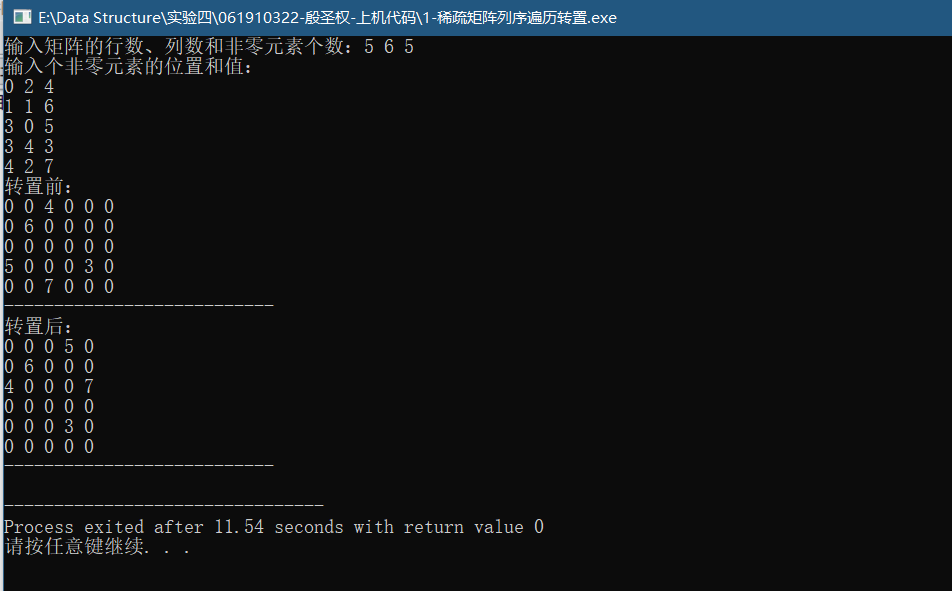
列序遍历：

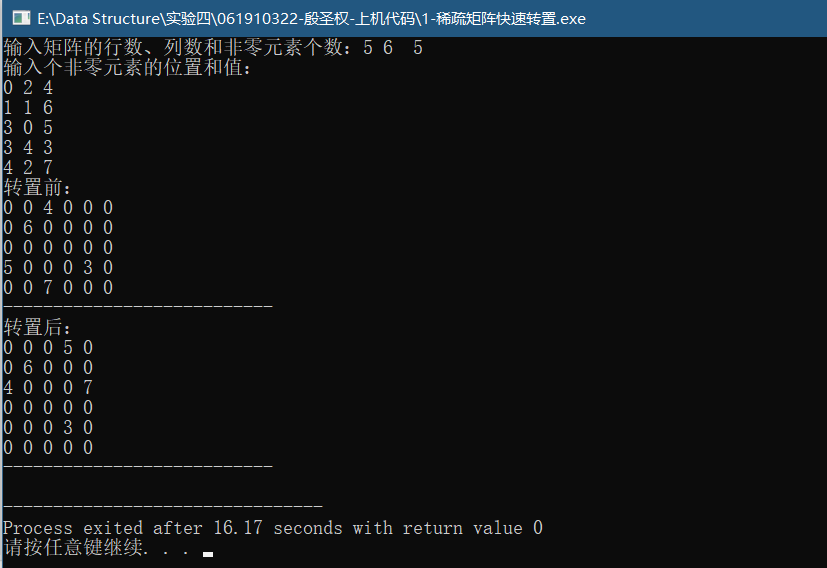
首先在A.arr中找到列号为0的三元组，将其转置并使之成为B.arr中行号为0的三元组，以此类推。即假设稀疏矩阵A有n列，则需要进行n躺扫描，第k躺是在A.arr的j中查找列号为k的三元组，若存在，则把此三元组存放在B.arr的第k行。

快速：

遍历两次A的三元组，第一次统计A中每列三元组的个数，即B中每行的个数，以便算出B每行三元组的起始位置，第二次遍历A时把三元组放在B中相应位置。

运行结果：





结果分析：

列序遍历：O（A.Cols \* A.Nums）；

快速转置法：O（A.Cols + A.Nums）。

附源程序。

2、

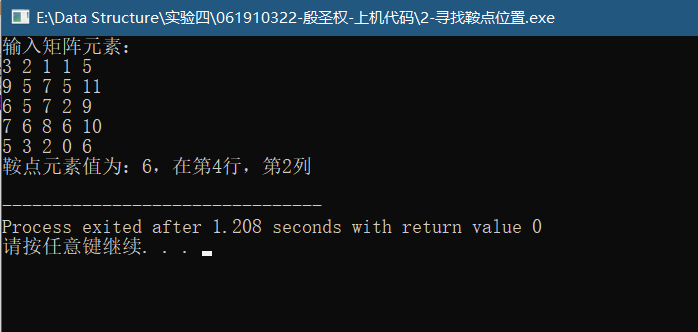
题目：

求矩阵的马鞍点。（书P69 7）

算法思想：

从第一行开始，找出每行的最小值，然后找出最小值所在列的最大值，若最大值依然是这个最小值，则输出此元素的位置。

运行结果：



结果分析：O（n）。

附源程序。

3、

题目：

CSP题目

题目背景：某地疫情爆发后，出于“应检尽检”的原则，我们想要通知所有近期经过该高危区域的居民参与核酸检测。

问题描述：想要找出经过高危区域的居民，分析位置记录是一种简单有效的方法。

具体来说，一位居民的位置记录包含t个平面坐标(x1,y1),(x2,y2),…,(xt,yt), 其中(xi,yi) 表示该居民i时刻所在位置。

高危区域则可以抽象为一个矩形区域（含边界），左下角和右上角的坐标分别为(xl,yd)和(xr,yu)，满足xl<xr且yd<yu。

考虑某位居民的位置记录，如果其中某个坐标位于矩形内（含边界），则说明该居民经过高危区域；进一步地，如果其中连续k个或更多坐标均位于矩形内（含边界），则认为该居民曾在高危区域逗留。需要注意的是，判定经过和逗留时我们只关心位置记录中的t个坐标，而无需考虑该居民在i到i+1时刻之间位于何处。

给定高危区域的范围和n位居民过去t个时刻的位置记录，试统计其中经过高危区域的人数和曾在高危区域逗留的人数。

输入格式：

输入共n+1行。

第一行包含用空格分隔的七个整数n、k、t、xl、yd、xr和yu，含义如上文所述。

接下来n行，每行包含用空格分隔的2t个整数，按顺序表示一位居民过去t个时刻的位置记录(x1,y1),(x2,y2),…,(xt,yt)。

输出格式：

输出共两行，每行一个整数，分别表示经过高危区域的人数和曾在高危区域逗留的人数。

样例输入1：

5 2 6 20 40 100 80

100 80 100 80 100 80 100 80 100 80 100 80

60 50 60 46 60 42 60 38 60 34 60 30

10 60 14 62 18 66 22 74 26 86 30 100

90 31 94 35 98 39 102 43 106 47 110 51

0 20 4 20 8 20 12 20 16 20 20 20

样例输出1：

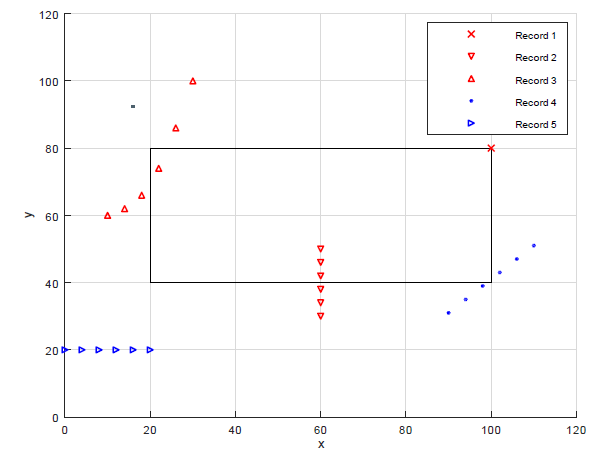
3

2

样例1说明：

如下图红色标记所示，前三条位置记录经过了高危区域；

但第三条位置记录（图中左上曲线）只有一个时刻位于高危区域内，不满足逗留条件。



样例输入2：

1 3 8 0 0 10 10

-1 -1 0 0 0 0 -1 -1 0 0 -1 -1 0 0 0 0

样例输出2：

1

0

样例2说明：

该位置记录经过了高危区域，但最多只有连续两个时刻位于其中，不满足逗留条件。

评测用例规模与约定：

全部的测试点满足1≤n≤20，1≤k≤t≤103，所有坐标均为整数且绝对值不超过106。

算法思想：

//判断是否经过或逗留

int pass = 0, stay = 0, p = 0, q;

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

{

q = 0;//判断是否经过

p = 0;//判断是否逗留

for(j=0; j<t; j++)

{

if(people[i][j].x>=x1 && people[i][j].x<=xr && people[i][j].y>=yd && people[i][j].y<=yu)

{

q = 1;

p++;

}

else

{

if(p >= k)

;

else

p = 0;//若没有连续经过，则p=0

}

}

if(q == 1)

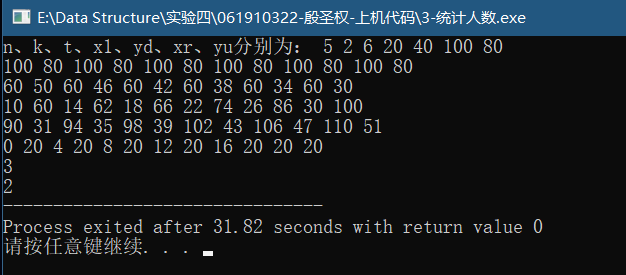
pass++;

if(p >= k)

stay++;

}

运行结果：



结果分析：O（n）。

附源程序。

4、

题目：

CSP题目

问题描述：请实现一个铁路购票系统的简单座位分配算法，来处理一节车厢的座位分配。  
　　假设一节车厢有20排、每一排5个座位。为方便起见，我们用1到100来给所有的座位编号，第一排是1到5号，第二排是6到10号，依次类推，第20排是96到100号。  
　　购票时，一个人可能购一张或多张票，最多不超过5张。如果这几张票可以安排在同一排编号相邻的座位，则应该安排在编号最小的相邻座位。否则应该安排在编号最小的几个空座位中（不考虑是否相邻）。  
　　假设初始时车票全部未被购买，现在给了一些购票指令，请你处理这些指令。

输入格式：对于所有评测用例，1 ≤ n ≤ 100，所有购票数量之和不超过100。

　　输入的第一行包含一个整数n，表示购票指令的数量。  
　　第二行包含n个整数，每个整数p在1到5之间，表示要购入的票数，相邻的两个数之间使用一个空格分隔。

输出格式

输出n行，每行对应一条指令的处理结果。  
　　对于购票指令p，输出p张车票的编号，按从小到大排序。

**问题分析：**这个问题可以用顺序结构或链式结构实现。

样例输入

4  
2 5 4 2

样例输出

1 2  
6 7 8 9 10  
11 12 13 14  
3 4

算法思想：

Status Arrangement(int p[ ],int n)

{

Seat s[100];

int i, k;

for(i=0; i<100; i++)

{

s[i].Num = i+1;

s[i].someone = 0;//0代表空座位，初始状态为空

}

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

{

int Bought = 0; //是否买了相邻座位的票，0代表没有

//1、有相邻座位

for(i=0; i<100; i++)

{

if(s[i].someone == 0 && s[i+p[k]-1].someone == 0 && i/5 == (i+p[k]-1)/5)//若有相邻座位的票

{

for(int j=i; j<i+p[k]; j++)

{

printf("%d ", s[j].Num);

s[j].someone = 1;//该座位已被分配，状态为1

}

printf("\n");

Bought = 1;//完成一次购买相邻座位的票

break;

}

}

//如果已经买了相邻座位的票，则进行下一次买票，否则去买不相邻的票

if(Bought == 1)

continue;

//2、没有相邻座位

if(96+p[k]>100 || 97+p[k]>100 || 98+p[k]>100 || 99+p[k]>100)

{

int vNum=0;

for(int v=0; v<100; v++)

{

if(s[v].someone == 0)

{

printf("%d ", s[v].Num);

s[v].someone = 1;

vNum++;

}

if(vNum == p[k])

{

printf("\n");

break;

}

}

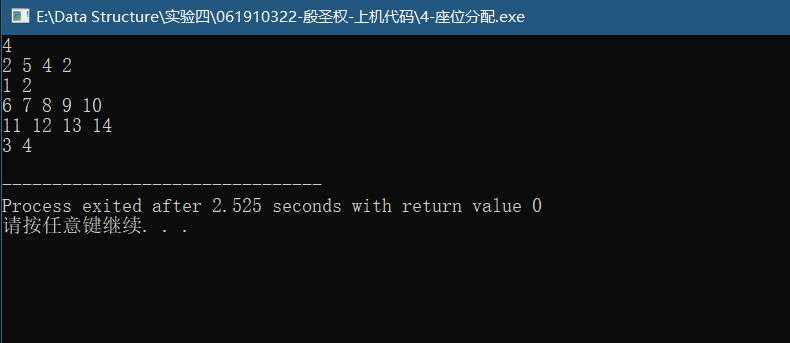
}

}

return OK;

}

运行结果：



结果分析：O（n\*i）

附源程序。

5、

题目：

3.20 假设以二维数组g(1…m, 1…n)表示一个图像区域，g[i,j]表示该区域中点(i,j)所具颜色，其值为从0到k的整数。编写算法置换点(i0,j0)所在区域的颜色。约定和(i0,j0)同色的上、下、左、右的邻接点为同色区域的点。

算法思想：

Status RegionFilling(point\_data g[M][N], point Cur, int NewColor)

{

Stack s;

InitStack(s);

point\_data e;

int OldColor = g[Cur.x][Cur.y].Color;//存储染色前的颜色

Push(s, g[Cur.x][Cur.y]);

while(!StackEmpty(s))//栈空时，停止循环

{

Pop(s,e);//弹栈并染色

Cur = e.seat;

g[Cur.x][Cur.y].Color = NewColor;//染色

g[Cur.x][Cur.y].Visited = 1;//标记为1，说明已被染色

//若相邻点存在且需要染色则压栈

if(Cur.x<M && !g[Cur.x+1][Cur.y].Visited && g[Cur.x+1][Cur.y].Color==OldColor)

Push(s ,g[Cur.x+1][Cur.y]);

if(Cur.x>0 && !g[Cur.x-1][Cur.y].Visited && g[Cur.x-1][Cur.y].Color==OldColor)

Push(s, g[Cur.x-1][Cur.y]);

if(Cur.y<N && !g[Cur.x][Cur.y+1].Visited && g[Cur.x][Cur.y+1].Color==OldColor)

Push(s, g[Cur.x][Cur.y+1]);

if(Cur.y>0 && !g[Cur.x][Cur.y-1].Visited && g[Cur.x][Cur.y-1].Color==OldColor)

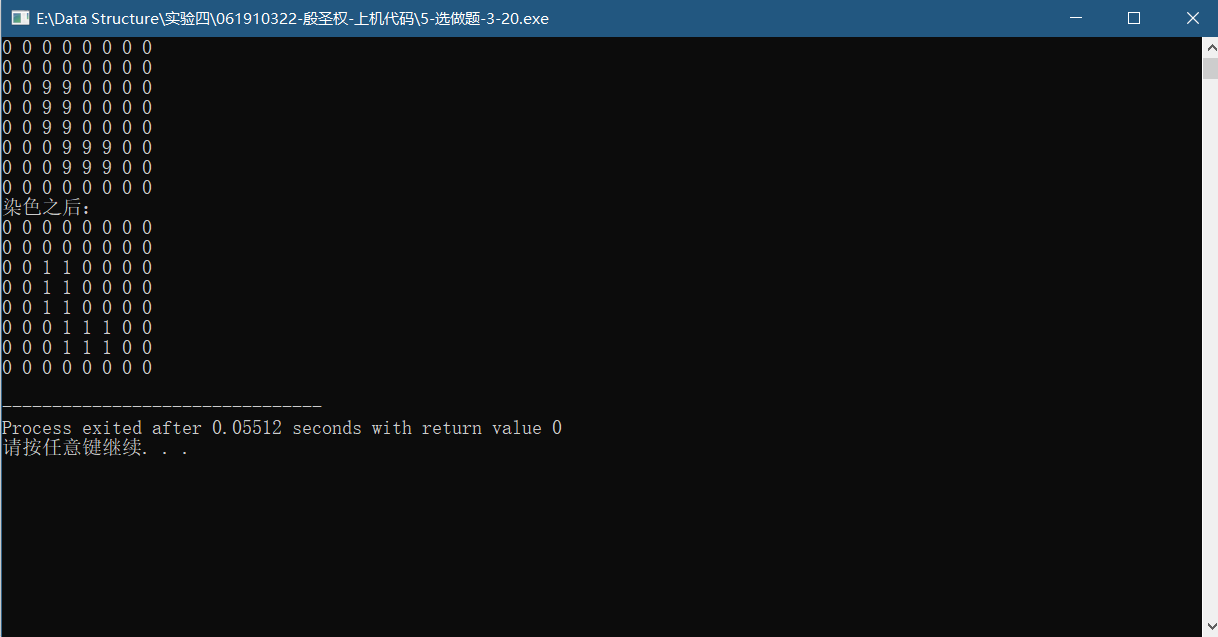
Push(s, g[Cur.x][Cur.y-1]);

}

return OK;

}

运行结果：



结果分析：O(n)。

附源程序。

6、

题目：

3.28 假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点（注意不设头指针），试编写相应的队列初始化、入队列和出队列的算法。

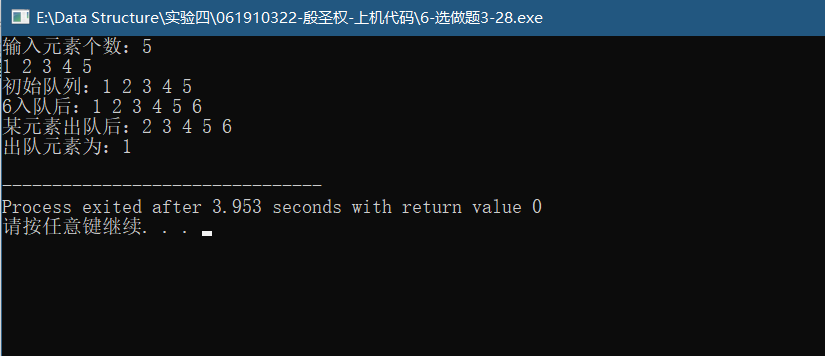
算法思想：

初始化：创建结点并接在队尾后

入队:把元素接在队尾，并更新队尾

出队：摘除头结点后第一个结点，若是表尾则另外判断

运行结果：



结果分析：O（1）。

附源程序。

7、

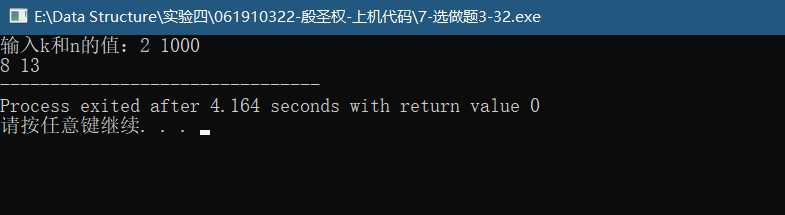
题目：

3.32 试利用循环队列编写求k阶菲波那契序列中前n+1项的算法，要求满足：而，其中max为某个约定的常数。（注意：本题所用循环队列的容量仅为k，则在算法执行结束时，留在循环队列中的元素应是所求k阶菲波那契序列中的最后k项）

算法思想：

前k项为0，第k项为1，以后的每一项为前面的元素之和。

运行结果：



结果分析：O（n）。

附源程序。

二、代码行数及小结

代码行数：

实验1：92+98

实验2：59

实验3：59

实验4：79

实验5：136

实验6：114

实验7：95

小结：

掌握了数组、字符串的基本操作和应用，复习了队列和栈的操作。