**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。**

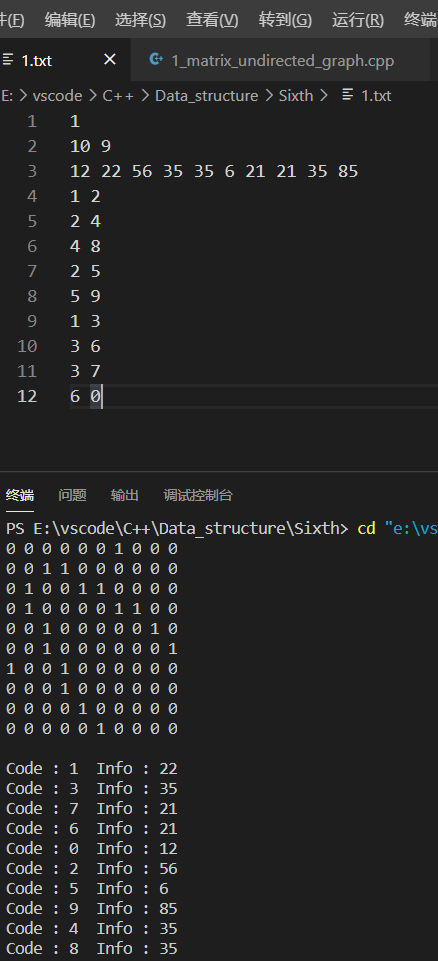
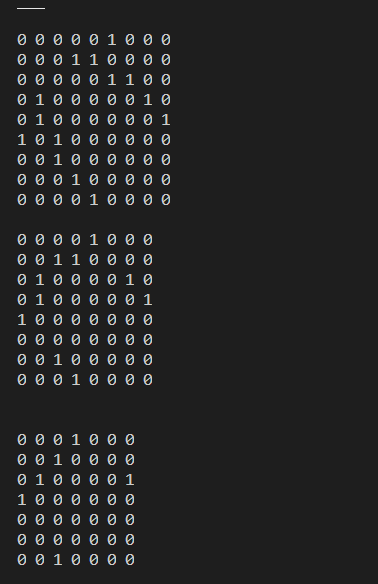
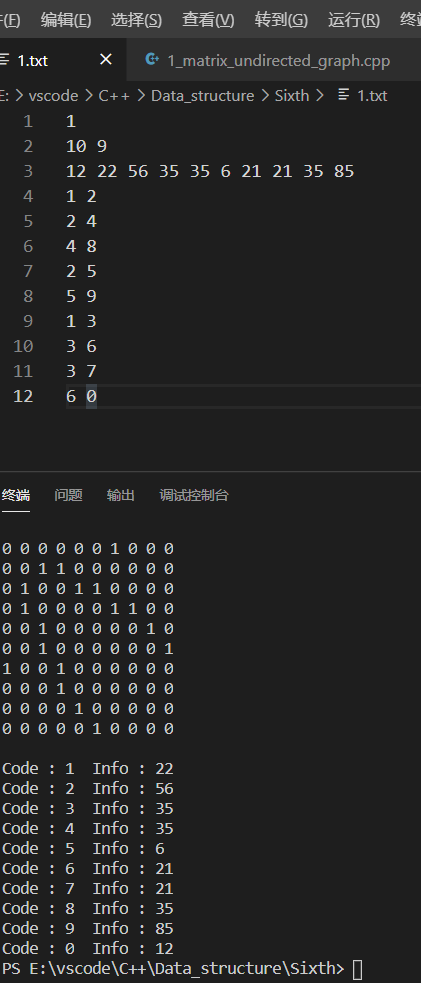
一、调试成功程序及说明

1、

题目：利用邻接矩阵储存方式实现图的基本操作

算法思想：作出基本操作

运行结果：

结果分析：运行结果均正确

附源程序。

#include <iostream>

#include <string>

#include <fstream>

#include <stack>

#include <queue>

using namespace std;

#define ElemType int

#define INFINITY INT\_MAX //定义正无穷

#define MAX\_V 20 //最大定点数目

#define GraphKind int //声明图的标志 1无向图 2有向图 3有向网 4无向网

typedef struct

{

int code; //结点编号0~G.vernum

ElemType info; //结点其他信息，这里是点权

}VertexType;

typedef struct

{

int arcs[MAX\_V][MAX\_V]; //邻接矩阵

int vexnum, arcnum; //顶点个数和边的个数

VertexType vexs[MAX\_V]; //存放定点信息

GraphKind type;

}MGraph; //图的邻接矩阵类型

bool CreateGraph(MGraph &G, string FileName)

{

fstream file(FileName.c\_str(),ios::in);

if(!file)

{

cout << "Floating TIP File " << FileName << " can't be opened !"<<endl;

return false;

}

file >> G.type; //输入图的类型

file >> G.vexnum >> G.arcnum ;

for(int i = 0; i<G.vexnum; i++)

{

G.vexs[i].code = i;

file >> G.vexs[i].info;

}

for(int i = 0; i < G.vexnum; i++)

{

for(int j = 0; j < G.vexnum; j++)

{

G.arcs[i][j] = 0;

}

}

int x = 0, y = 0;

for(int i = 0; i < G.arcnum; i++)

{

file >> x >> y;

G.arcs[x][y] = 1;

G.arcs[y][x] = 1;

}

return true;

}

//建立无向图

void DestroyGraph(MGraph &G)

{

for(int i = 0; i < G.vexnum; i++)

{

for(int j = 0; j < G.vexnum; j++)

{

G.arcs[i][j] = 0;

}

}

for(int i = 0; i<G.vexnum; i++)

{

G.vexs[i].code = 0;

G.vexs[i].info = 0;

}

G.arcnum = 0;

G.vexnum = 0;

G.type = -1;

return;

}

//销毁无向图

VertexType GetVex(MGraph G, int v) //v是编号，返回元素值

{

VertexType e;

if(v < 0 || v >= G.vexnum)

{

e.code = -1;

e.info = -1;

return e;

}

else

{

e = G.vexs[v];

return e;

}

}

//返回下标为v的元素值

int FirstAdjVex(MGraph G, int v)

{

if(v < 0 || v >= G.vexnum)

{

return -1;

}

bool flag = false;

for(int j = 0; j < G.vexnum; j++)

{

if(G.arcs[v][j] == 1)

{

flag = true;

return j;

}

}

if(!flag)

{

return -1; //找不到邻接点

}

}

int NextAdjVex(MGraph G, int v, int w)

{

if(v < 0 || v >= G.vexnum)

{

return -1;

}

bool flag = false;

for(int j = w + 1; j < G.vexnum; j++)

{

if(G.arcs[v][j] == 1)

{

flag = true;

return j;

}

}

if(!flag)

{

return -1;

}

}

void VisitVertex(VertexType Ver)

{

cout << "Code : "<<Ver.code << " Info : " << Ver.info << " "<<endl;

return;

}

//访问的函数

void DFSTraverse(MGraph G, int v) //v记录当前遍历到的结点

{

bool flag[G.vexnum];

stack <int> S;

S.push(v); //从v开始进行遍历

int now;

while(!S.empty())

{

now = S.top();

VisitVertex(G.vexs[now]);

flag[now] = true;

S.pop();

for(int j = 0; j < G.vexnum; j++)

{

if(G.arcs[now][j] == 1 && !flag[j]) //找到一个连接的点，并且没有访问过

{

S.push(j);

}

}

}

return ;

}

//进行深度优先遍历

void BFSTraverse(MGraph G, int v)

{

bool flag[G.vexnum];

queue <int> Q;

int now;

VisitVertex(G.vexs[v]);

flag[v] = true;

Q.push(v);

while(!Q.empty())

{

now = Q.front();

Q.pop();

for(int j = 0; j <G.vexnum; j++)

{

if(G.arcs[now][j] == 1 && !flag[j])

{

VisitVertex(G.vexs[j]);

flag[j] = true;

Q.push(j);

}

}

}

return;

}

//广度优先遍历

void InsertVex(MGraph &G, VertexType ver)

{

if(G.vexnum == MAX\_V)

{

cout << "Memory is overflowed, vertex can't insert !"<<endl;

}

ver.code = G.vexnum;

G.vexs[G.vexnum] = ver; //储存顶点信息

G.vexnum++;

for(int i = 0; i < G.vexnum; i++)

{

G.arcs[ver.code][i] = 0;

G.arcs[i][ver.code] = 0;

}

return;

}

//插入一个顶点

void InsertArc(MGraph &G, int v, int w)

{

if(v < 0 || v >= G.vexnum || w < 0 || w >= G.vexnum)

{

cout << "can't add edge"<<endl;

return ;

}

G.arcs[v][w] = 1;

G.arcs[w][v] = 1;

G.arcnum ++;

return;

}

//插入一条边

void DeleteVex(MGraph &G, int v)

{

if(v < 0 || v >= G.vexnum)

{

cout << v <<" vertex is not exist !"<<endl;

return ;

}

for(int i = 0; i < G.vexnum; i++)

{

if(G.arcs[i][v] == 1 || G.arcs[v][i] == 1)

{

G.arcnum--;

}

G.arcs[v][i] = 0;

G.arcs[i][v] = 0;

}

//直接全扫一遍

for(int i = v; i < G.vexnum - 1; i++)

{

for(int j = 0; j < G.vexnum; j++)

{

G.arcs[i][j] = G.arcs[i+1][j];

}

}

for(int i = 0; i < G.vexnum - 1; i++)

{

for(int j = v; j < G.vexnum - 1; j++)

{

G.arcs[i][j] = G.arcs[i][j+1];

}

}

for(int i = v; i < G.vexnum - 1; i++)

{

G.vexs[i] = G.vexs[i+1];

G.vexs[i].code--;

}

//对点进行移位

G.vexnum--;

return;

}

void DeleteArc(MGraph &G, int v, int w)

{

if(v < 0 || v >= G.vexnum || w < 0 || w >= G.vexnum)

{

cout << "Can't delete edge"<<endl;

return ;

}

if(G.arcs[v][w] == 1 || G.arcs[w][v] == 1)

{

G.arcs[v][w] = 0;

G.arcs[w][v] = 0;

cout << "Floating TIP Delete the arc betewwn " << v << " and "<< w << " successfully ! "<<endl;

}

else

{

cout << "Floating TIP The arc between "<< v << " and "<< w<< " is now exist."<<endl;

}

return;

}

void DisplayMGraph(MGraph G)

{

if(G.vexnum == 0 && G.vexnum ==0)

{

cout << "NULL Graph"<<endl;

}

for(int i = 0; i < G.vexnum; i++)

{

for(int j = 0; j < G.vexnum; j++)

{

cout << G.arcs[i][j]<< " ";

}

cout << endl;

}

}

int main()

{

string f = "1.txt";

MGraph G;

VertexType V;

V.code =100;

V.info = 99;

CreateGraph(G,f);

DisplayMGraph(G);

cout << endl;

DFSTraverse(G,1);

cout << endl;

DisplayMGraph(G);

cout << endl;

BFSTraverse(G,1);

//DestroyGraph(G);

//VisitVertex(V);

//V = GetVex(G,1);

//VisitVertex(V);

//DisplayMGraph(G);

// DisplayMGraph(G);

// cout << FirstAdjVex(G,2)<<endl;;

// cout << NextAdjVex(G,2,1);

DeleteVex(G,1);

DisplayMGraph(G);

cout <<endl;

DeleteVex(G,2);

DisplayMGraph(G);

cout <<endl;

cout <<endl;

DeleteVex(G,2);

DisplayMGraph(G);

cout <<endl;

return 0;

}

2、

题目：

算法思想：

运行结果：

结果分析：

附源程序。

3、

题目：

算法思想：

运行结果：

结果分析：

附源程序。

4、

题目：

算法思想：

运行结果：

结果分析：

附源程序。

5、

题目：

CSP题目

问题描述：俄罗斯方块是俄罗斯人阿列克谢·帕基特诺夫发明的一款休闲游戏。

　　游戏在一个15行10列的方格图上进行，方格图上的每一个格子可能已经放置了方块，或者没有放置方块。每一轮，都会有一个新的由4个小方块组成的板块从方格图的上方落下，玩家可以操作板块左右移动放到合适的位置，当板块中某一个方块的下边缘与方格图上的方块上边缘重合或者达到下边界时，板块不再移动，如果此时方格图的某一行全放满了方块，则该行被消除并得分。

　　在这个问题中，你需要写一个程序来模拟板块下落，你不需要处理玩家的操作，也不需要处理消行和得分。

　　具体的，给定一个初始的方格图，以及一个板块的形状和它下落的初始位置，你要给出最终的方格图。

输入格式：

　　输入的前15行包含初始的方格图，每行包含10个数字，相邻的数字用空格分隔。如果一个数字是0，表示对应的方格中没有方块，如果数字是1，则表示初始的时候有方块。输入保证前4行中的数字都是0。

　　输入的第16至第19行包含新加入的板块的形状，每行包含4个数字，组成了板块图案，同样0表示没方块，1表示有方块。输入保证板块的图案中正好包含4个方块，且4个方块是连在一起的（准确的说，4个方块是四连通的，即给定的板块是俄罗斯方块的标准板块）。

　　第20行包含一个1到7之间的整数，表示板块图案最左边开始的时候是在方格图的哪一列中。注意，这里的板块图案指的是16至19行所输入的板块图案，如果板块图案的最左边一列全是0，则它的左边和实际所表示的板块的左边是不一致的（见样例）

输出格式：

输出15行，每行10个数字，相邻的数字之间用一个空格分隔，表示板块下落后的方格图。注意，你不需要处理最终的消行。

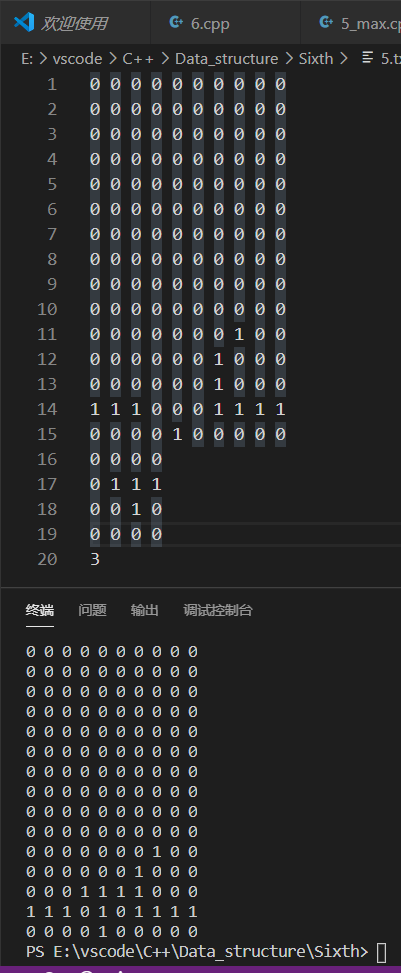
算法思想：

这道题目我尝试了三种方法。第一种是首先计算碰撞点的坐标，然后把图案复制过去，第二种是进行连通性判断，利用深度优先搜索，找到连通部分的数值，如果是4的话，那么物块下落。第三种是最简单有效的方法，判断每一列物块下边是不是为空，是的话就将物块下落。为了达到上述目的，我们为了方便搜索，将下落物块进行染色处理，其标记为2，原有物块标记为1，空的部分标记为0。根据题意，前四行数据全为零，即可以将4\*4的物块复制到原图形上来，顺便将其染色。

首先我们根据需求，实现物块下落的功能，对于每一列，按照行数，从最后一行向上搜索，搜索到标记为2的块，就将其下边的一个变为2，当前格子变为其上方的数据。

实现判断物块是否可以继续下落的函数，如果所有列的下方都为零，那么物块可以下落。（这里我进行计数，如果物体占有的列数，等于下方为零的数目，那么认为物块可以继续下落）

运行结果：



结果分析：运行结果正确，通过CSP测试。



附源程序。

#include <iostream>

#include <fstream>

#include <string.h>

using namespace std;

int map[20][20];

int object[10][10];

int position = 0;

int max\_col[20];

int min\_col[10];

void print()

{

for (int i = 1; i <= 15; i++)

{

for (int j = 1; j <= 10; j++)

{

if(map[i][j] != 0)

{

cout << "1";

}

else

{

cout << "0";

}

cout <<" ";

}

cout << endl;

}

return ;

}

void test\_print()

{

for (int i = 1; i <= 15; i++)

{

for (int j = 1; j <= 10; j++)

{

cout <<map[i][j]<< " ";

}

cout << endl;

}

return ;

}

int read()

{

fstream file("5.txt",ios::in);

if(!file)

{

cout << "file can't be opened !"<<endl;

exit(0);

}

for (int i = 1; i <= 15; i++)

{

for (int j = 1; j <= 10; j++)

{

file >> map[i][j];

}

}

for (int j = 1; j <= 10; j++)

{

map[16][j] = 1;

}

for (int i = 1; i <= 4; i++)

{

for(int j = 1; j <= 4; j ++)

{

file >> object[i][j];

}

}

file >> position ;

}

void start()

{

for(int i = 1; i <= 4; i++)

{

for(int j = 1; j <= 4; j++)

{

if(object[i][j] != 0)

{

map[i][position+j-1] = 2;

}

}

}

return ;

}

void drop( )

{

for (int j = 0; j < 4; j++)

{

for(int i = 15; i >= 1; i--)

{

if(map[i][j+position] == 2)

{

map[i+1][j+position] = map[i][j+position];

map[i][j+position] = map[i-1][j+position];

}

}

}

return;

}

int cnt = 0;

void dfs(int x, int y)

{

if(map[x][y] == 0)

{

cout<<"yes"<<endl;

return;

}

else

{

cnt++;

map[x][y] = 0;

if(x + 1 < 16 && map[x+1][y] != 0)

{

dfs(x+1,y);

}

if(x - 1 > 0 && map[x-1][y] != 0)

{

dfs(x-1,y);

}

if(y + 1 < 11 && map[x][y+1] != 0)

{

dfs(x,y+1);

}

if(y - 1 > 0 && map[x][y-1] != 0)

{

dfs(x,y-1);

}

map[x][y] = 2;

}

}

void FindFirst(int &x, int &y)

{

for(int i = 1; i <= 15; i++)

{

for(int j = 1; j <= 10; j++)

{

if(map[i][j] == 2)

{

x = i;

y = j;

return ;

}

}

}

return;

}

bool judge()

{

int sum = 0;

int cnt\_1 = 0;

bool flag = false;

for (int j = 0; j < 4; j++)

{

flag = false;

for(int i = 15; i >= 1; i--)

{

if(map[i][j+position] == 2)

{

flag = true;

if(map[i+1][j+position] == 0)

{

cnt\_1++;

}

}

}

if(flag)

{

sum ++;

}

}

if(sum == cnt\_1)

{

return true;

}

else

{

return false;

}

}

int main()

{

int x = 1, y =1;

read();

start();

do

{

if(cnt == 4)

{

drop();

}

cnt = 0;

FindFirst(x,y);

cout << "x = "<<x<<" y = "<<y << endl;

dfs(x,y);

}while(cnt == 4);

while(judge())

{

drop();

}

cout << endl;

print();

return 0;

}

6、

题目：

算法思想：

运行结果：

结果分析：

附源程序。

......

二、未调试成功程序及说明

1、

题目：

算法思想：

错误原因：

附源程序。

2、

题目：

算法思想：

错误原因：

附源程序。

......

三、代码行数及小结