项目说明文档

数据结构课程设计

——勇闯迷宫游戏

同济大学

Tongji University

目 录

[1 分析 1](#_Toc28973)

[1.1 背景分析 1](#_Toc32144)

[1.2 功能分析 1](#_Toc30178)

[2 设计 1](#_Toc21354)

[2.1 数据结构设计 1](#_Toc19079)

[2.2 类结构设计 1](#_Toc19591)

[2.3 成员与操作设计 2](#_Toc3377)

[2.4 系统设计 3](#_Toc9633)

[3 功能实现 3](#_Toc4244)

[3.1 迷宫初始化功能的实现 3](#_Toc31281)

[3.1.1 迷宫功能核心代码 3](#_Toc8942)

[3.1.2 迷宫功能截屏示例 3](#_Toc16758)

[3.2 栈功能的实现 4](#_Toc29239)

[3.2.1 栈功能核心代码 4](#_Toc17171)

[3.3 寻路功能的实现 6](#_Toc13698)

[3.3.1 寻路功能流程图 6](#_Toc31701)

[3.3.2 寻路功能核心代码 6](#_Toc12594)

[3.3.3 寻路功能截图示例 7](#_Toc13544)

[3.4总体系统实现 8](#_Toc17674)

[3.4.1 总体系统流程图 8](#_Toc3998)

[3.7.2总体系统核心代码 8](#_Toc32037)

[4 测试 10](#_Toc21048)

[4.1 功能测试 10](#_Toc10679)

[4.2 边界测试 11](#_Toc20484)

# 1 分析

## 背景分析

回溯法是计算机遍历的一种常用的方法，通过栈的先进后出的结构特点，进行先行点的储存，在某个路径无法继续进行下去的时候，将栈中的点排出，再进行另一个路径进行。可以通过递归的方法进行实现。

## 1.2 功能分析

作为一个简单的迷宫路径探索的问题，其解决问题的逻辑应当如下：

若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。

综上所述，为了保证在达到某一个点后不能向前继续行走时，能正确返回前一个以便从下一个方向向前试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向。该程序应有迷宫建立函数，栈建立函数，四个方向的寻路函数

# 

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，可以通过一个二维数组进行迷宫的建立，一个单链表来进行栈的实现，并在主函数中定义四个不同方向的寻路函数。

## 2.2 类结构设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（date）与链表类（list），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。为方便处理，本系统采用struct描述链表结点类(date)，这样使得链表结点类（list）可以访问链表结点。

## 2.3 成员与操作设计

**链表结点类（date）**

struct date

{

public :

int row; //计入在迷宫中的行

int col; //计入在迷宫中的列

date\* node;//链接点

};

**链表类（class）**

class stack //建立一个栈

{

private :

date \* top; //使用了定义的结构体

public:

stack() ;

void pushin(int n,int m) ;//压入函数

void decline();

void sendstack(int &n,int &m);//推栈函数

void showout();

## };

class palace //建立迷宫函数

{

public:

int movepath; //一个用于记录是否是空路，一个用于记录是否曾走过这条路

int move;

palace() { movepath = 1; move = 1; };

};

## 2.4 系统设计

系统首先在主函数中对二维数组赋初值实现对迷宫的初始化，再完成对栈的创建和输入数据工作，然后根据入口的位置，开始进行向四个方向寻路，不断压入点进入栈中，还有在进入死路时，退栈进行再次的尝试

# 3 功能实现

## 3.1 迷宫初始化功能的实现

### 3.1.1 迷宫功能核心代码

palace a[7][7];

a[1][1].move = a[2][1].move = a[2][2].move = a[2][3].move = a[3][3].move = a[4][3].move = a[4][4].move = a[4][5].move = a[5][5].move = 0;

a[1][1].movepath = a[2][1].movepath = a[2][2].movepath = a[2][3].movepath = a[3][3].movepath = a[4][3].movepath = a[4][4].movepath = a[4][5].movepath = a[5][5].movepath = 0;

a[3][1].move = a[3][1].movepath = 0;

a[4][1].move = a[4][1].movepath = 0;

a[2][4].move = a[2][4].movepath = 0;

a[2][5].move = a[2][5].movepath = 0;

for (int n = 0; n <= 6; n++)

{

for (int m = 0; m <= 6; m++)

{

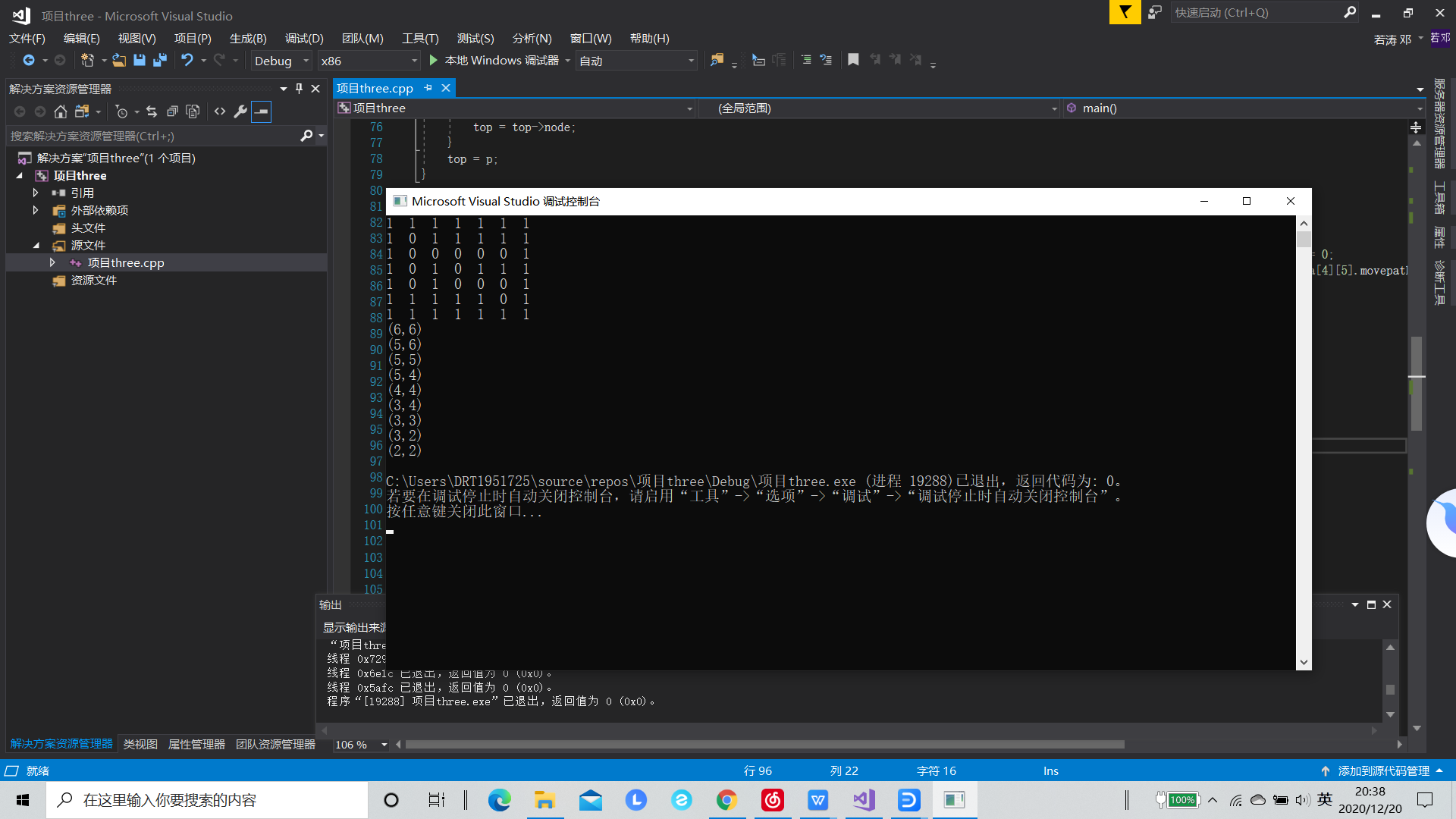
cout << a[n][m].move << " " ;

}

cout << endl;

### }

### 3.1.2 迷宫功能截屏示例



## 3.2 栈功能的实现

### 3.2.1 栈功能核心代码

class stack //建立一个栈

{

private :

date \* top;

public:

stack() ;

void pushin(int n,int m) ;//压入函数

void decline();

void sendstack(int &n,int &m);//推栈函数

void showout();

};

stack::stack() //初始化

{

top = new date;

top->col = 1;

top->row = 1;

top->node = NULL;

}

void stack::pushin(int n,int m) //压入

{

date \*p;

p = new date;

p->col = m;

p->row = n;

p->node = top;

top = p;

}

void stack::sendstack(int &n,int &m) //退出

{

n = top->row;

m = top->col;

}

void stack::decline() //解放空间

{

if (top->node != NULL)

{

date \*p;

p = top;

top = p->node;

delete p;

}

else

{

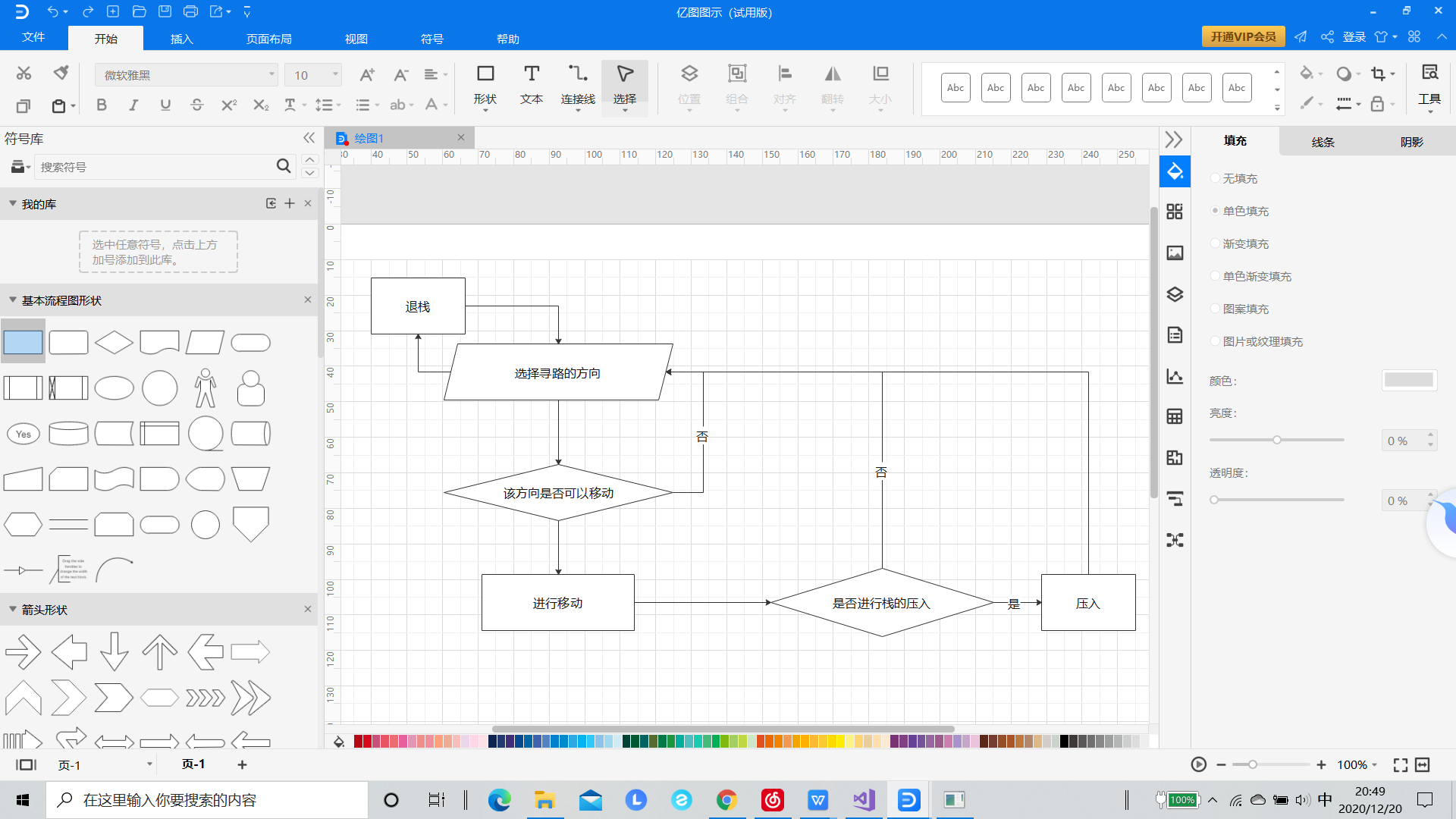
cout << "该栈中只有一个数据" << endl;

}

### }

## 3.3 寻路功能的实现

### 3.3.1 寻路功能流程图



### 3.3.2 寻路功能核心代码

bool rightseekload(palace a[7][7],stack &b, int &n, int &m)

{

if (m + 1 > 6) return false;

if (a[n][m + 1].movepath == 0) { b.pushin(n, m + 1); a[n][m + 1].movepath = 1; m++; return true; }

else return false;

}

bool downseekload(palace a[7][7], stack &b, int &n, int &m)

{

if (n + 1 > 6) return false;

if (a[n + 1][m].movepath == 0) { b.pushin(n + 1, m); a[n + 1][m].movepath = 1; n++; return true; }

else return false;

}

bool liftseekload(palace a[7][7], stack &b, int &n, int &m)

{

if (m - 1 < 0) return false;

if (a[n][m - 1].movepath == 0) { b.pushin(n, m - 1); a[n][m - 1].movepath = 1; m--; return true; }

else return false;

}

bool upseekload(palace a[7][7], stack &b, int &n, int &m)

{

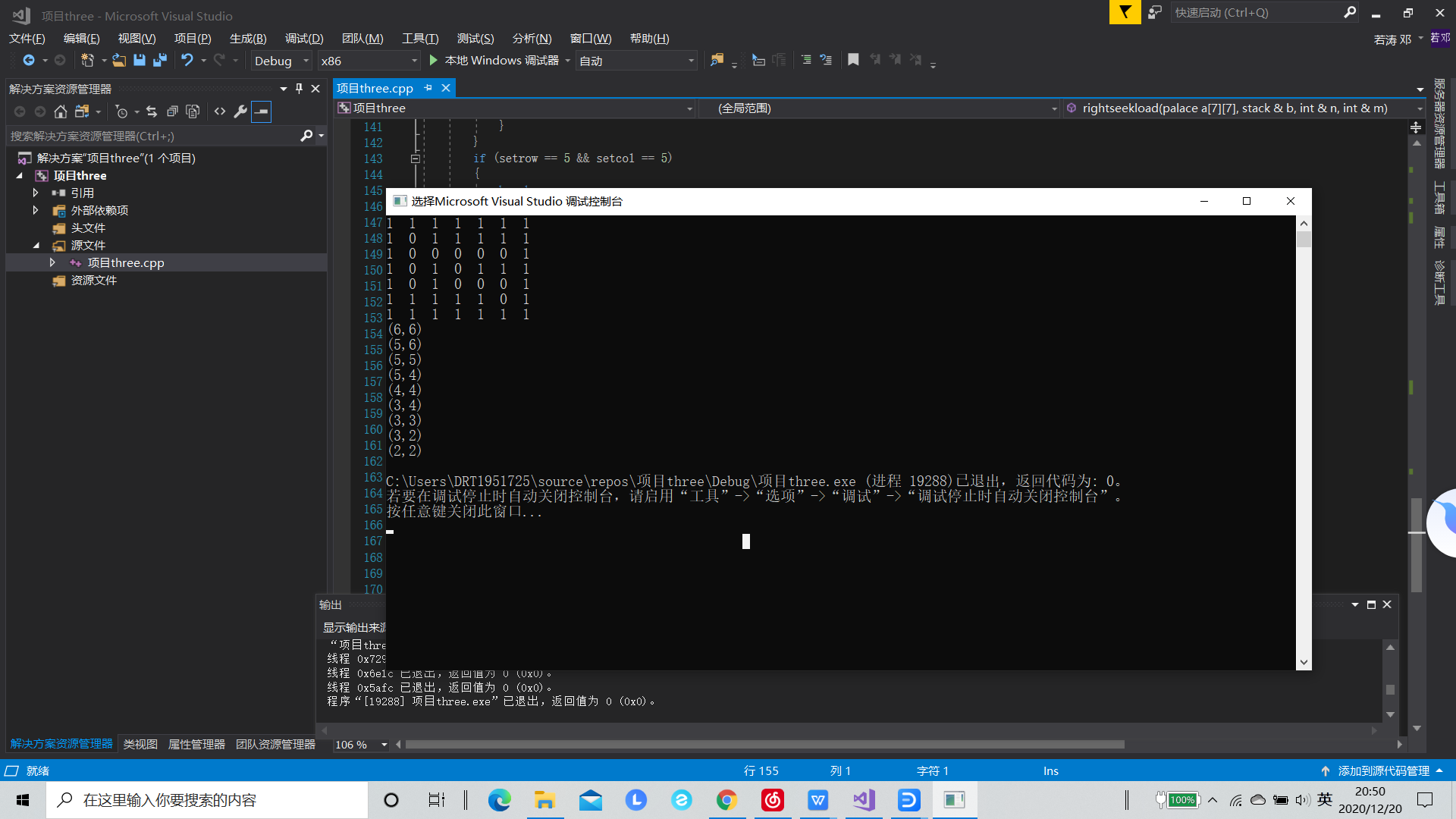
if (n - 1 < 0) return false;

if (a[n - 1][m].movepath == 0) { b.pushin(n - 1, m); a[n - 1][m].movepath = 1; n--; return true; }

else return false;

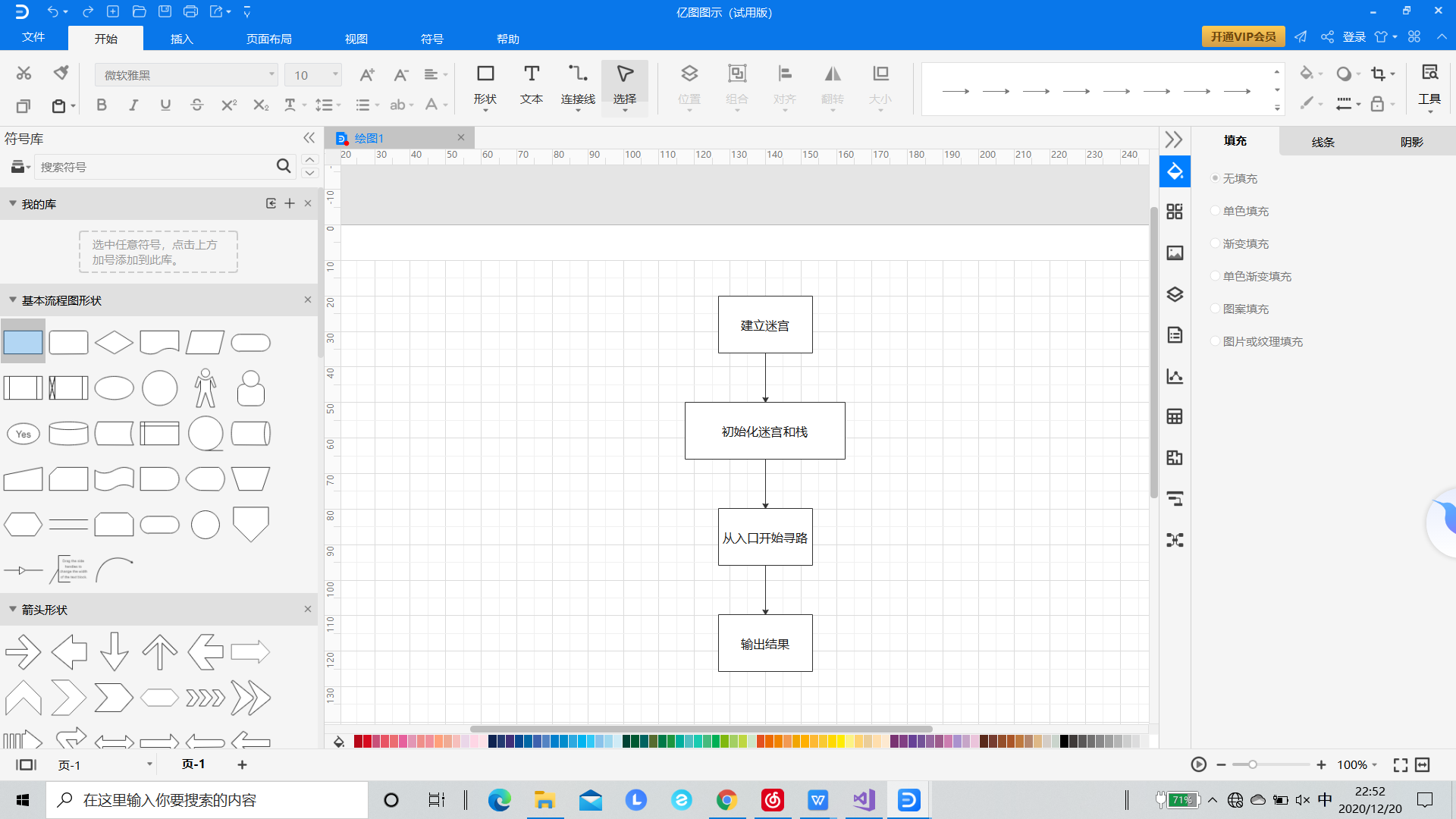
### }

### 3.3.3 寻路功能截图示例



# 3.4总体系统实现

### 3.4.1 总体系统流程图



### 3.7.2总体系统核心代码

for (int n = 0; n <= 6; n++)

{

for (int m = 0; m <= 6; m++)

{

cout << a[n][m].move << " " ;

}

cout << endl;

}

stack b;

bool rightseekload(palace a[7][7],stack &b,int &n,int &m);//右寻路

bool downseekload(palace a[7][7], stack &b, int &n, int &m);//下寻路

bool liftseekload(palace a[7][7], stack &b, int &n, int &m);//左寻路

bool upseekload(palace a[7][7], stack &b, int &n, int &m);//上寻路

int setrow; int setcol;

setrow = setcol=1;

for (;;)

{

for (;;)

{

if (rightseekload(a, b, setrow, setcol) == false)

{

if (setrow == 5 && setcol == 5)

{

break;

}

if (downseekload(a, b, setrow, setcol) == false)

{

if (setrow == 5 && setcol == 5)

{

break;

}

if (liftseekload(a, b, setrow, setcol) == false)

{

if (setrow == 5 && setcol == 5)

{

break;

}

if (upseekload(a, b, setrow, setcol) == false)

{

if (setrow == 5 && setcol == 5)

{

break;

}

b.decline();

b.sendstack(setrow, setcol);

//cout << setrow+1 << " " << setcol+1 << endl; 检验压入点的具体位置

break;

}

}

}

}

}

if (setrow == 5 && setcol == 5)

{

break;

}

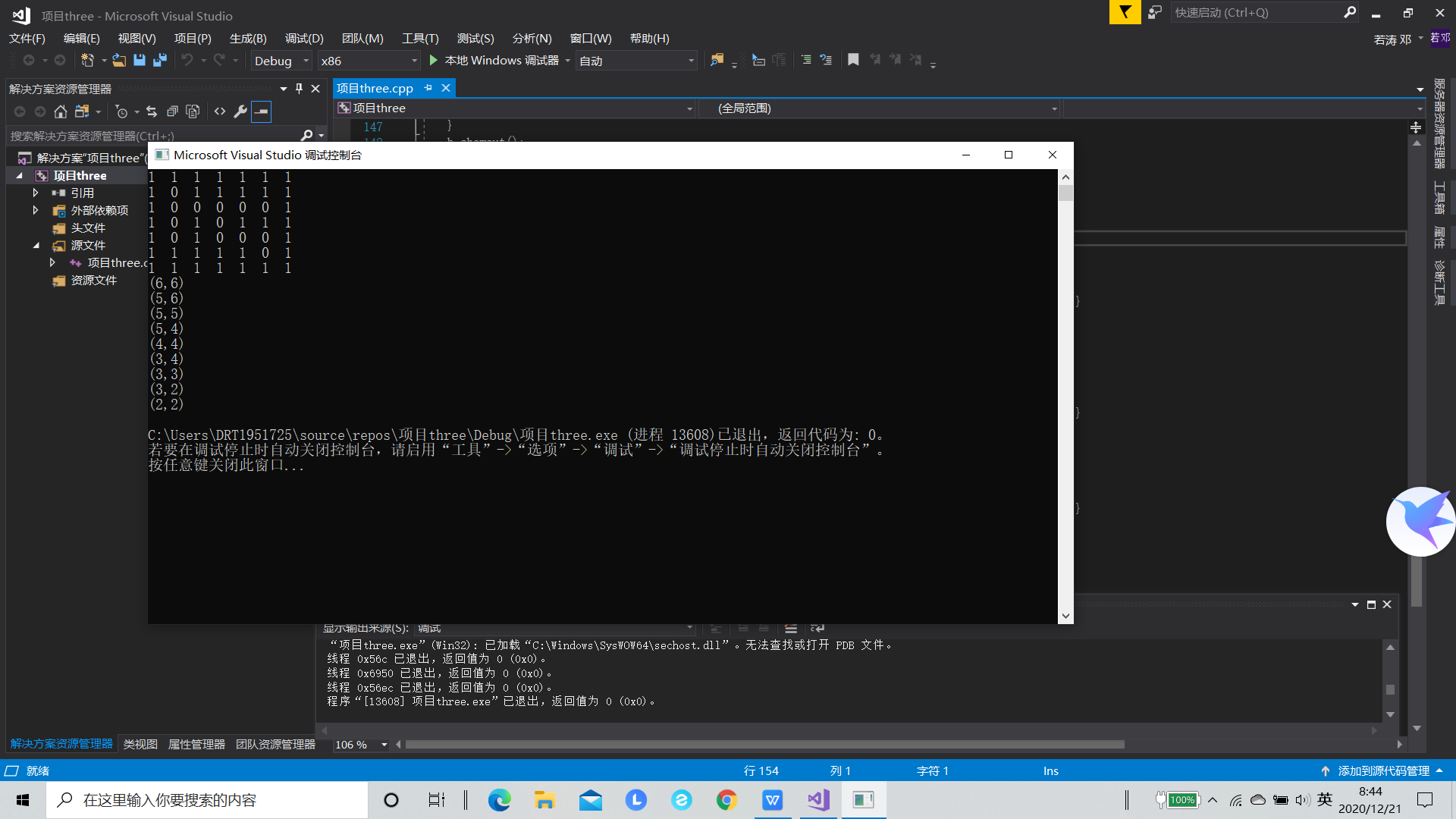
}

b.showout();

# 4 测试

## 4.1 功能测试

初始化一个迷宫，并通过回溯法进行路线的求解，输出可行的路程点。



## 4.2 边界测试

当迷宫的路径无法到达出口时

