项目说明文档

数据结构课程设计

——勇闯迷宫游戏

作 者 姓 名： 邓泉

学 号： 1953871

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1分析](#_Toc534119172)

[1.1 背景分析](#_Toc534119173)

[1.2 功能分析](#_Toc534119174)

[2 设计](#_Toc534119175)

[2.1 数据结构设计](#_Toc534119176)

[2.2 主程序设计](#_Toc534119181)

[3.实现](#_Toc534119182)

[3.1 LinkNode实现](#_Toc534119183)

[3.2 Stack类实现](#_Toc534119184)

3.3 主函数功能的实现................................................................................................

[4 测试](#_Toc534119185)

[4.1 功能测试](#_Toc534119186)

[4.1.1 测试1](#_Toc534119187)

[4.1.2 测试2](#_Toc534119188)

4.1.3 测试3..........................................................................................................

[4.2 错误测试](#_Toc534119189)

[4.2.1 没有通路](#_Toc534119190)

4.2.2 入口不合法.................................................................................................

4.2.3 出口不合法..................................................................................................

# **1 分析**

## **1.1 背景分析**

迷宫问题是一类常见的搜索问题，解决方法常分为深度优先和广度优先两种。本系统是使用回溯法求解迷宫的一条通路。回溯法本质上是一个递归过程，所以本题使用递归及背后的递归工作栈完成题目。

## **1.2 功能分析**

迷宫问题的求解过程可以采用回溯法即在一定的约束条件下试探地搜索前进，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。在求解过程中，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，当找到出口时试探过程就结束了。

实际就是一个深度优先搜索（BFS）,只不过不是在图中,而是在数组中,所有只要注意数组边界就好。同时，需要存储迷宫的路径，所以还需要一个栈。

例如：

输入：一个7\*7的矩阵，#表示墙壁，0表示可以走的路。

输出：输出迷宫通路在原迷宫上的反映及从左上角（1，1）到右下角（5，5）的所有迷宫通路中的一条路径并且输出标出找出路径后的迷宫。

# **2 设计**

## **2.1 数据结构设计**

栈具有先进后出（FILO）的特点，是一种使用广泛，特点鲜明的线性存储结构。本题使用递归来解决，需要用到递归工作栈。思路为回溯法，即从终点向起点方向走，若一条路走不通则回溯到上一节点。最终找到迷宫通路后反向输出迷宫通路，便是从起点到终点的通路。

## **2.1.1 LinkNode设计**

LinkNode节点保存一个模板类型的元素data,并且有一个指针link指向下一个元素,y用struct描述以便于访问。

## **2.1.2 Stack设计**

底层存储Node类头节点以及栈的元素个数top（也可视为栈顶“指针”）。为外界提供公有函数push（），pop（），getTop（），isEmpty（）。提供栈的基本功能。

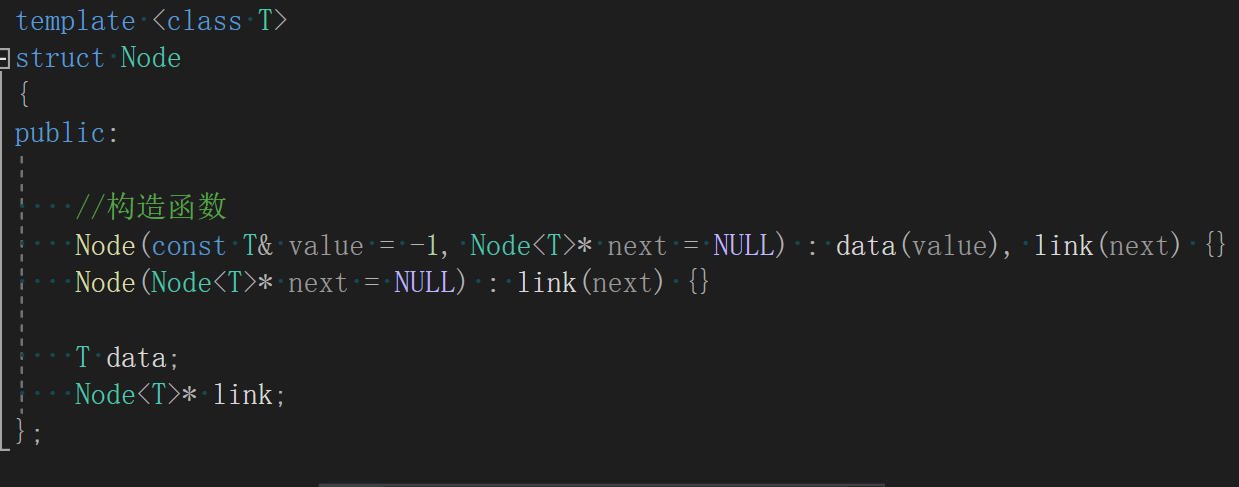
## **2.3 主程序设计**

由用户输入迷宫的规模以及迷宫具体地形，再让用户输入起始位置及目标位置。并且程序判断位置是否合理。展示迷宫原来的样子，再通过DFS（）递归寻找出路，并且用一个栈存储路径，如果找到了从栈中取出路径（颠倒一次栈），如果没找到给出没有路径的提示。

# 

# **3 实现**

## **3.1 LinkNode类实现**



## **3.2 Stack类实现**

### **3.2.1 push功能实现**

新加入一个Node元素，并且将之指向原来的头节点，原来的头节点指向现在的头节点，将存储的元素个数N+1。

### **1609164606(1)**

### **3.2.2 pop功能实现**

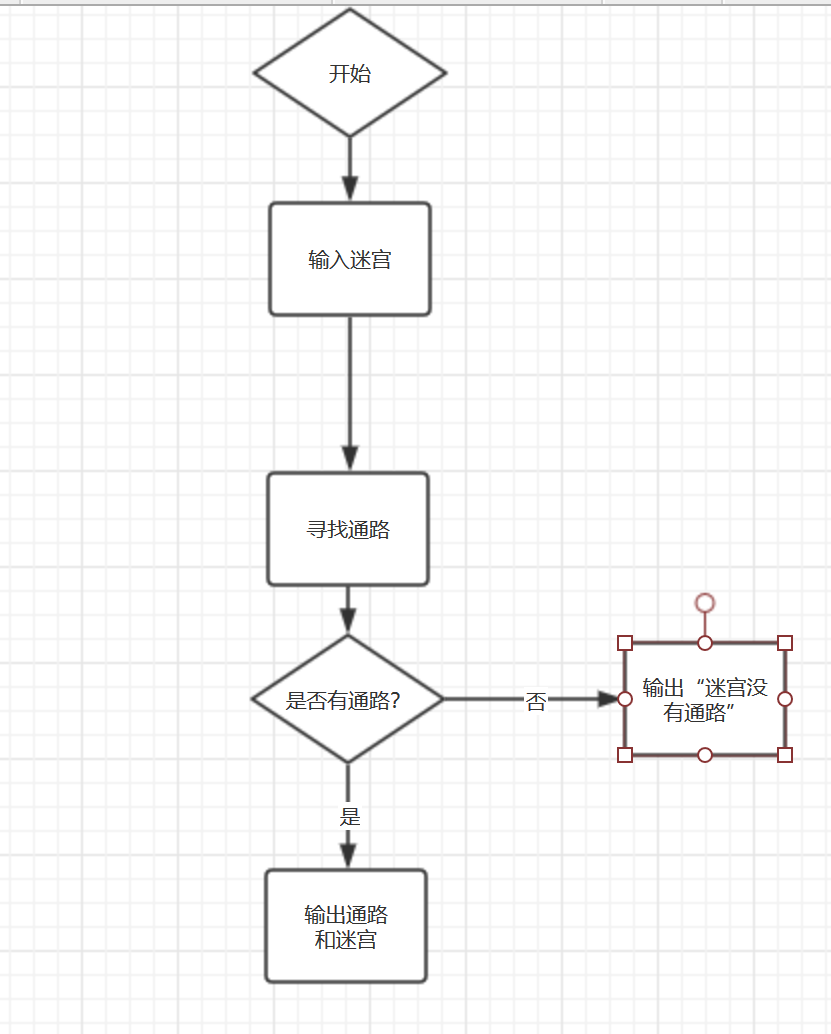
判断是否为空，不为空就让头节点指向下一个元素，然后delete之前的元素，释放内存，--top。

### **1609163676(1)**

### **3.2.3 isEmpty，getTop功能实现**

## **1609163626(1)**

## **3.3 主函数功能的实现**



### 

### **3.3.1 输入的处理**

先根据输入,赋值给M,N,然后创建M\*N大小的二维数组,再根据输入的内容创建迷宫地形。

### **1609162945(1)**

### **3.3.2 isValid（）函数**

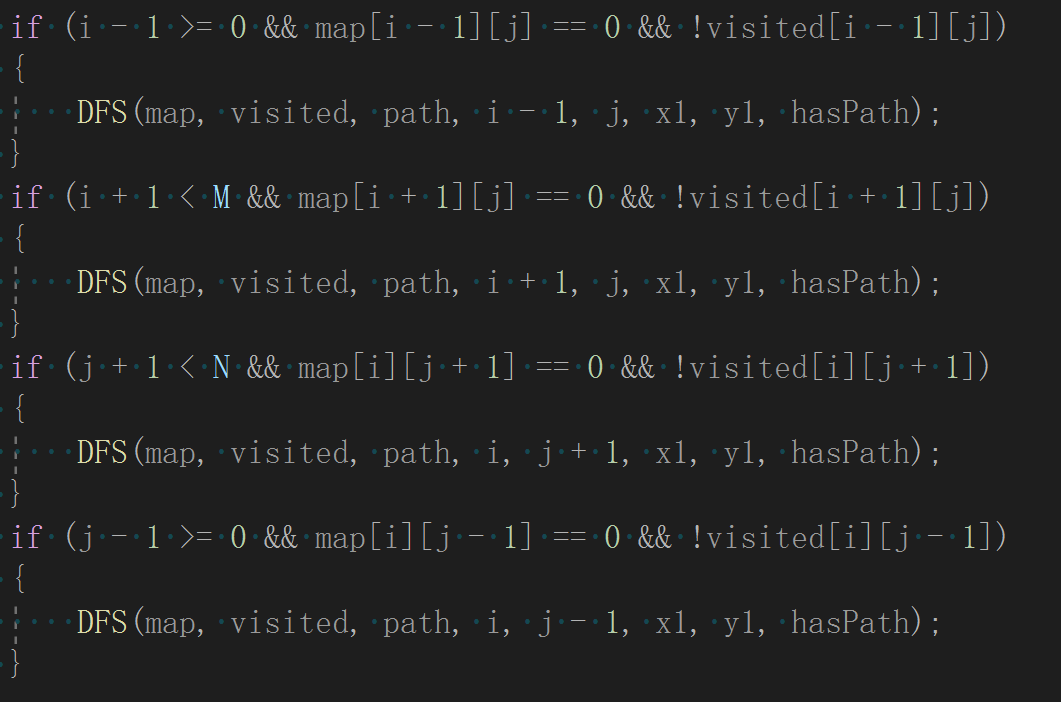
用于检查起始，终止位置合理性，是否越界，是否为墙，并且给相应的提示，如果不合法，就直接结束程序。

### **1609162992(1)**

### **3.3.3 DFS()函数初步实现**

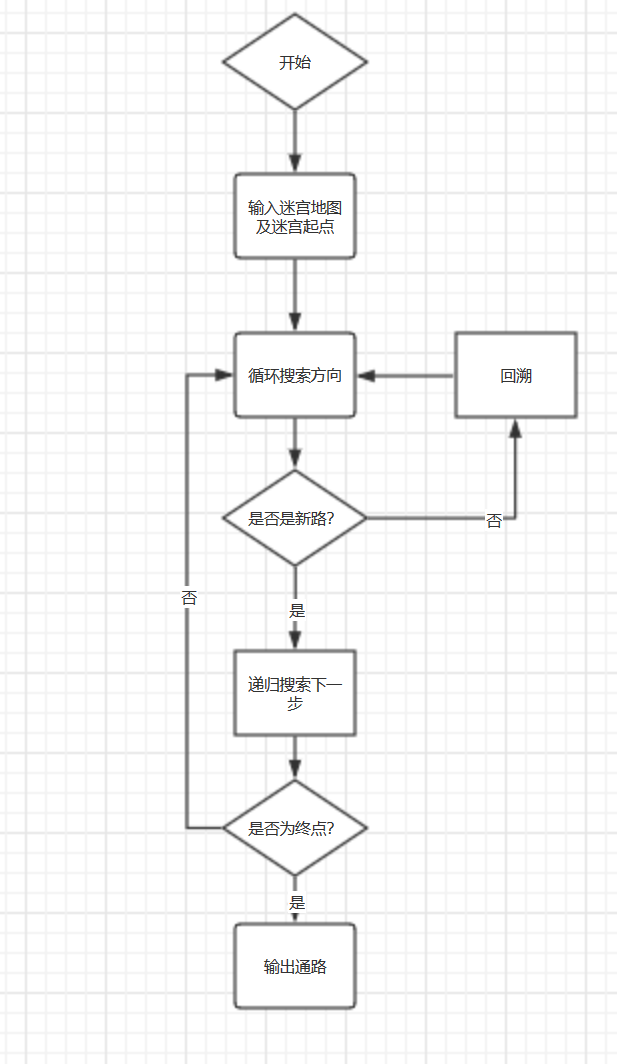
从x1,y1开始，在没有找到x2,y2之前，不懂调用递归往四个方向的，没有标记为访问过的，没有越界的，是路径的位置走，而且是深度优先搜索，即如果进入了一个方向，则直到找完这个方向的所有路径后，才走另一个方向。

初步实现的关键地方的代码（因为是直接截取完整程序，所以参数会带上了之后所用到的栈）：



### 

### **3.3.4 DFS完整实现**

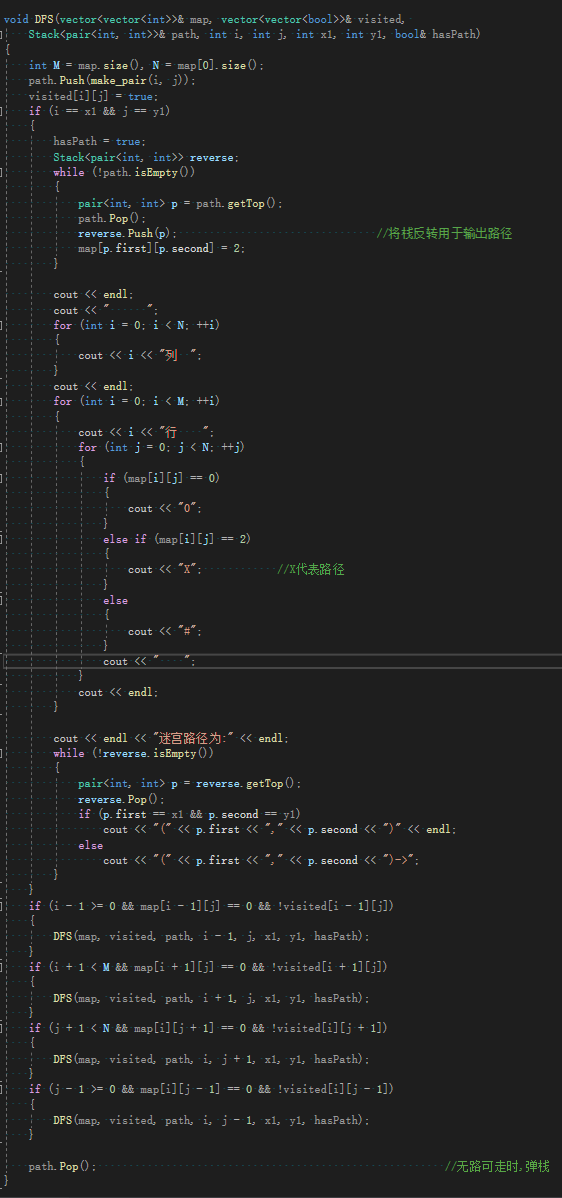


初步实现后，为了增加显示迷宫路径功能以及没有路径时给出提示的功能，还用到了一个自己实现的栈，以及bool类型的hasPath,用于当没有路径时才输出没有路径的提示信息。

每递归调用一次，就将这个坐标加入栈中（用pair存储了坐标）。而当四个if都没进入时，即无路可走时，就弹出栈顶元素，这意味着，这个栈总是存储着当前能走的路径。

然后，如果递归到了终点位置，即有路可走，就将hasPath设为true，并且用一个栈存储当前栈的反转（这样才是从入口到出口）。并且还将map在这个路径上的元素设置为2，即为路径。

最后，输出迷宫，迷宫上的路径用（X）表示，并且在下方通过遍历那个反转栈，输出路径坐标，从入口到出口。结束DFS（）函数。



**4 测试**

## **4.1 功能测试**

### **4.1.1测试1**

****测试用例****：

7 7

0 0 1 1 1 1 1

1 0 0 0 1 0 1

0 0 1 0 1 1 0

1 0 0 1 1 1 0

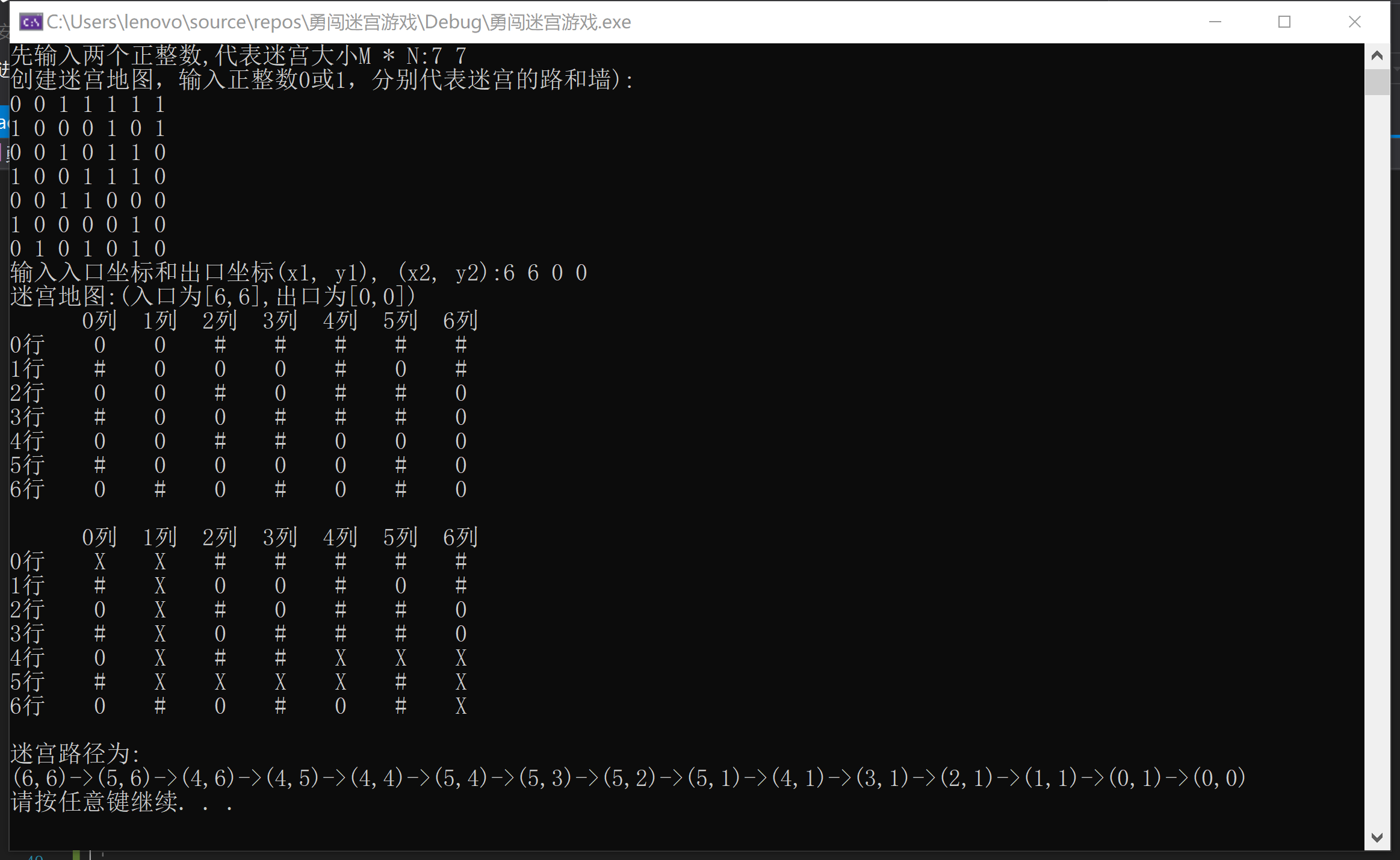
0 0 1 1 0 0 0

1 0 0 0 0 1 0

0 1 0 1 0 1 0

6 6 0 0

****实验结果:****

********

### **4.1.2 测试2**

****测试用例：****

7 7

0 0 1 1 1 1 1

1 0 0 0 1 0 1

0 0 1 0 1 1 0

1 0 0 1 1 1 0

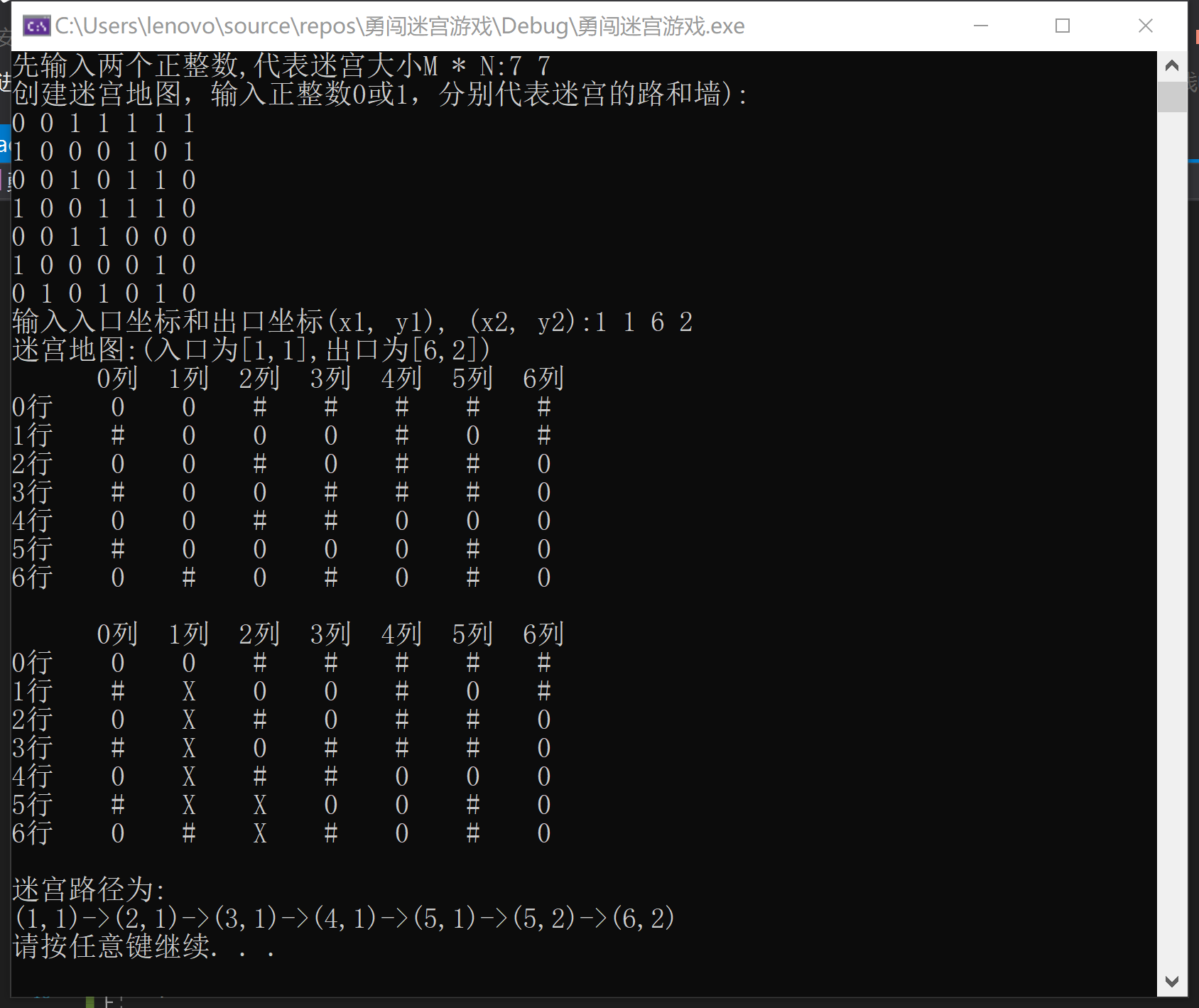
0 0 1 1 0 0 0

1 0 0 0 0 1 0

0 1 0 1 0 1 0

1 1 6 2

****实验结果：****

********

### **4.1.3 测试3**

****测试用例：****

8 9

0 0 1 1 1 1 1 0 1

0 0 0 0 0 0 1 0 1

0 0 1 0 1 0 0 0 0

1 0 0 1 1 1 1 1 0

0 0 1 1 0 0 0 1 0

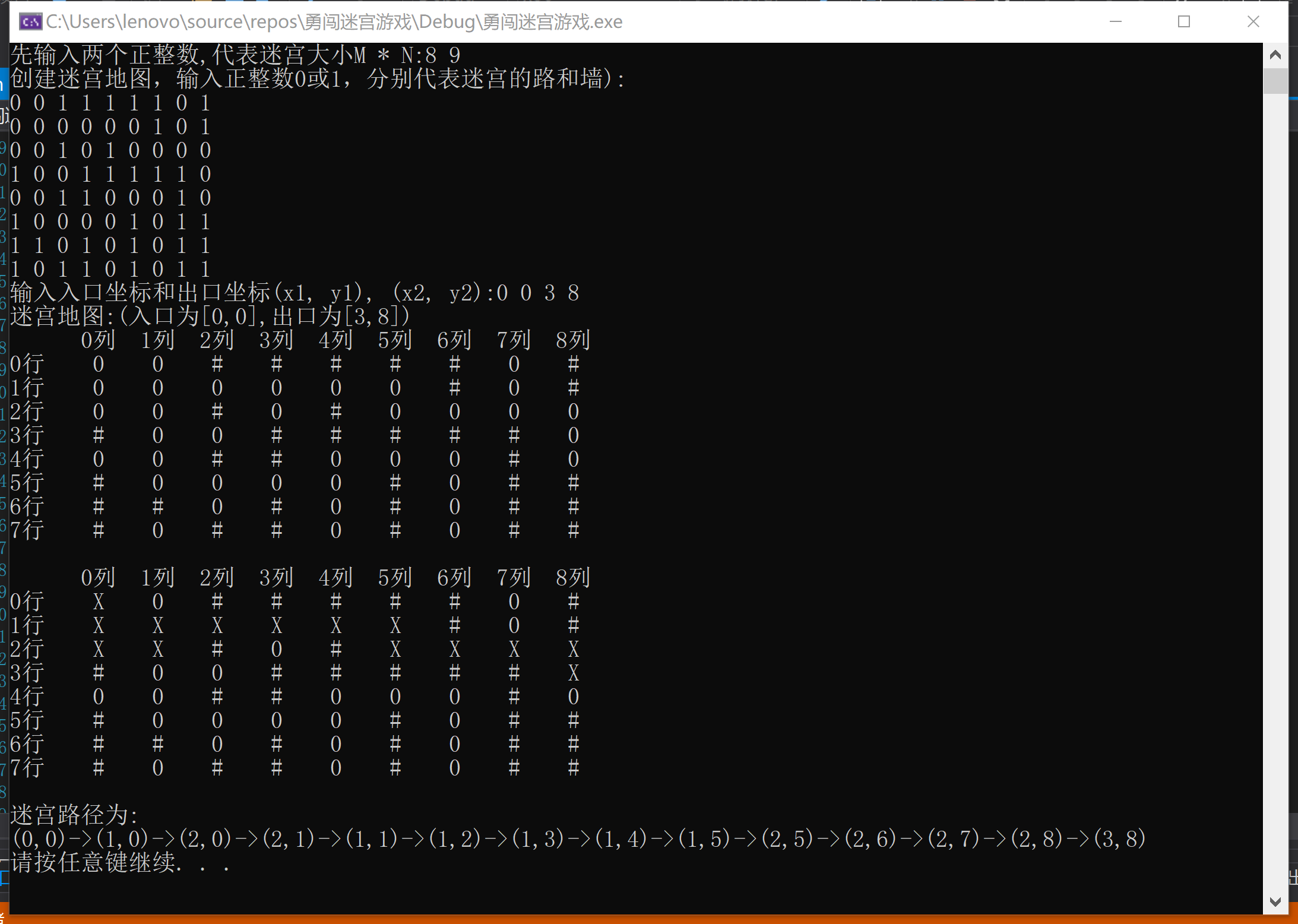
1 0 0 0 0 1 0 1 1

1 1 0 1 0 1 0 1 1

1 0 1 1 0 1 0 1 1

0 0 3 8

****实验结果：****

********

****(从测试用例可以看出来,虽然找到了路径,但是不是最短路径)****

## 

## **4.2 错误测试**

### **4.2.1 没有从入口到出口的路径**

****测试用例：****

7 8

0 0 1 1 1 1 1 0

0 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0

1 0 0 1 1 1 1 1

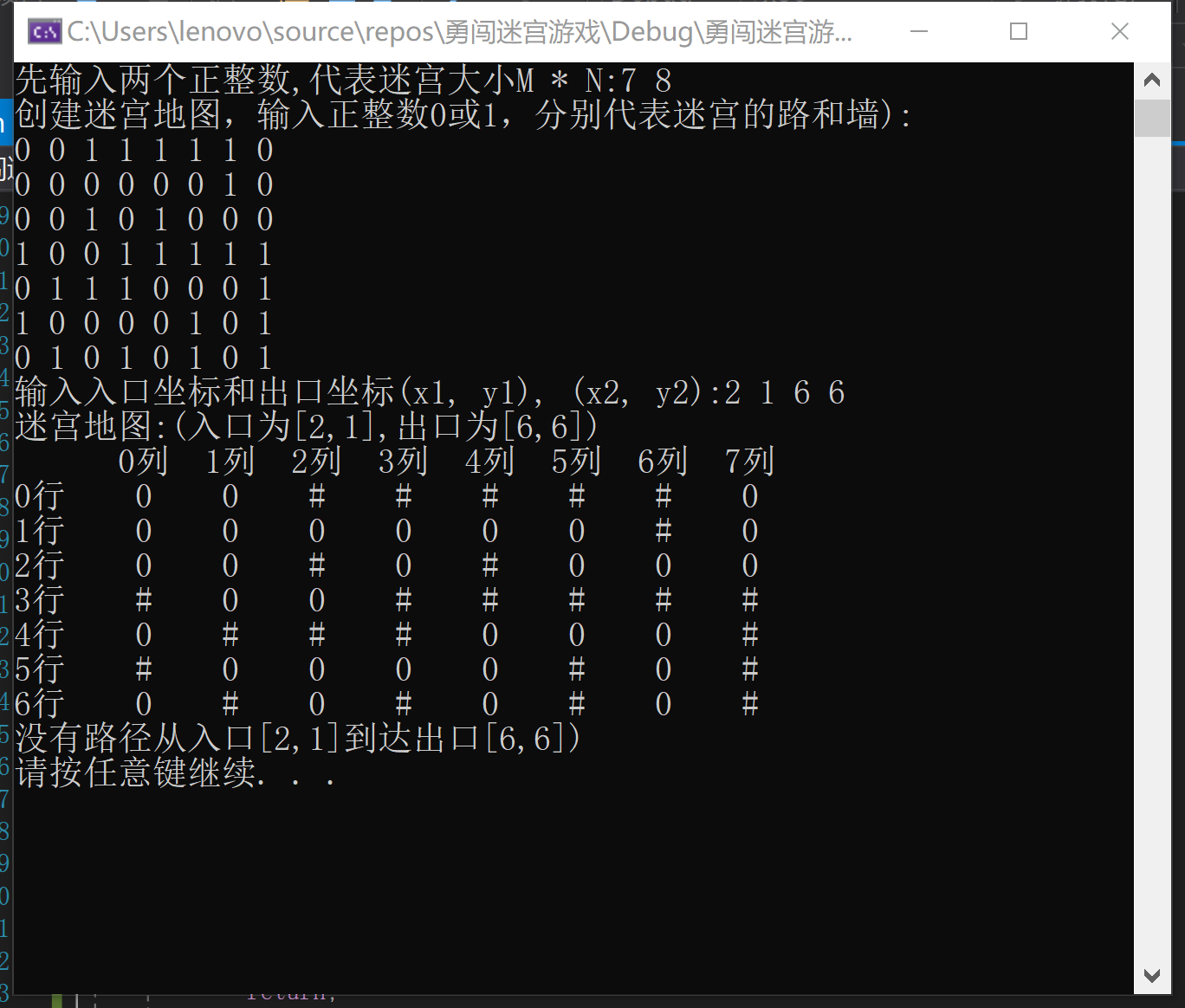
0 1 1 1 0 0 0 1

1 0 0 0 0 1 0 1

0 1 0 1 0 1 0 1

2 1 6 6

****实验结果：****

********

### **4.2.2 入口不在数组中或者为墙**

****测试用例：****

7 8

0 0 1 1 1 1 1 0

0 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0

1 0 0 1 1 1 1 1

0 1 1 1 0 0 0 1

1 0 0 0 0 1 0 1

0 1 0 1 0 1 0 1

0 3 6 6

****实验结果：****

********

****测试用例：****

7 8

0 0 1 1 1 1 1 0

0 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0

1 0 0 1 1 1 1 1

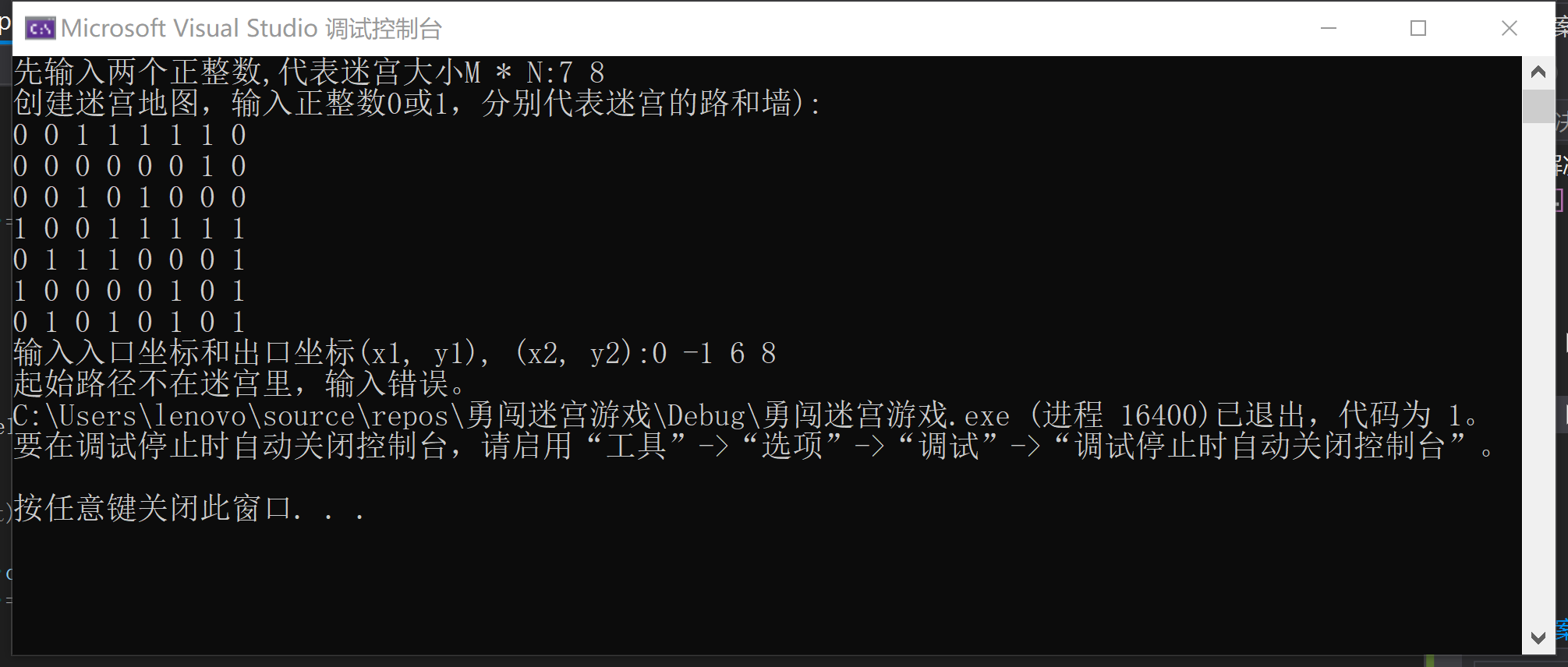
0 1 1 1 0 0 0 1

1 0 0 0 0 1 0 1

0 1 0 1 0 1 0 1

0 -1 6 8

****实验结果：****

********

### **4.2.3 出口不在迷宫中或者出口为墙**

****测试用例：****

7 8

0 0 1 1 1 1 1 0

0 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0

1 0 0 1 1 1 1 1

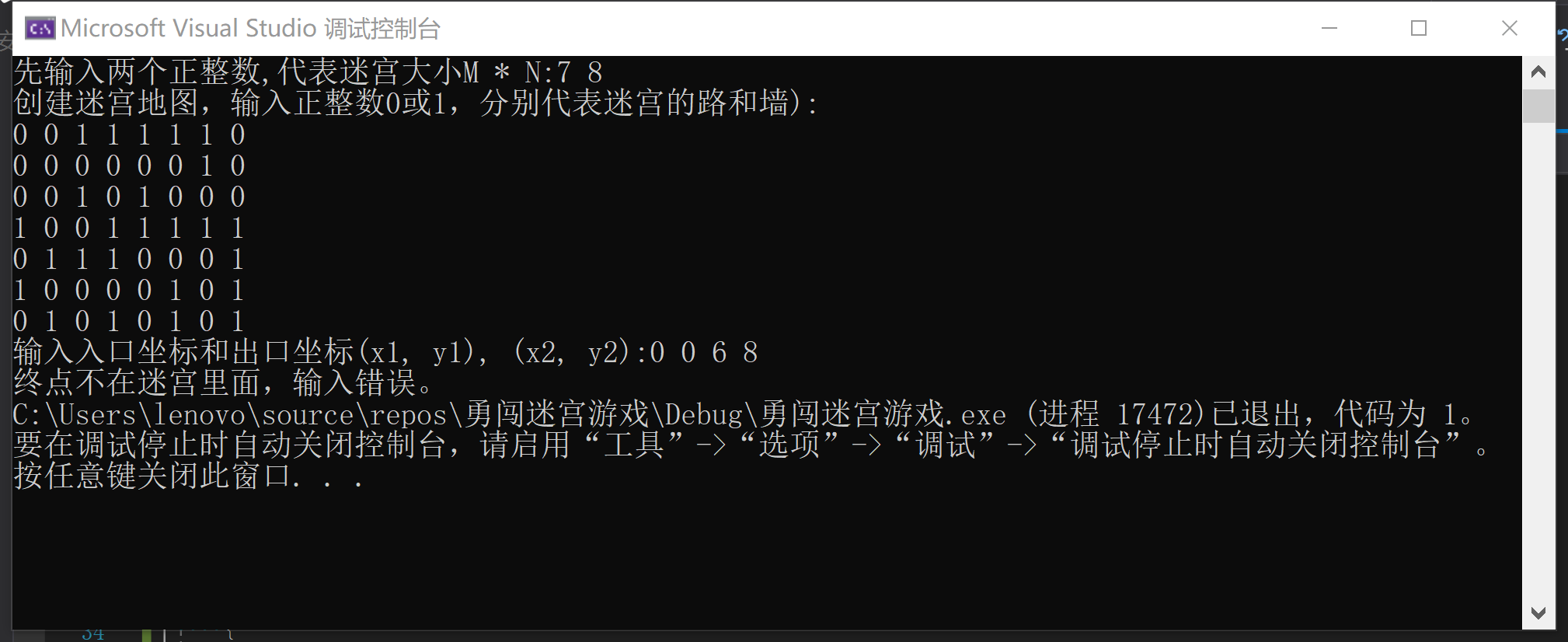
0 1 1 1 0 0 0 1

1 0 0 0 0 1 0 1

0 1 0 1 0 1 0 1

0 0 6 8

****实验结果：****



****测试用例：****

7 8

0 0 1 1 1 1 1 0

0 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0

1 0 0 1 1 1 1 1

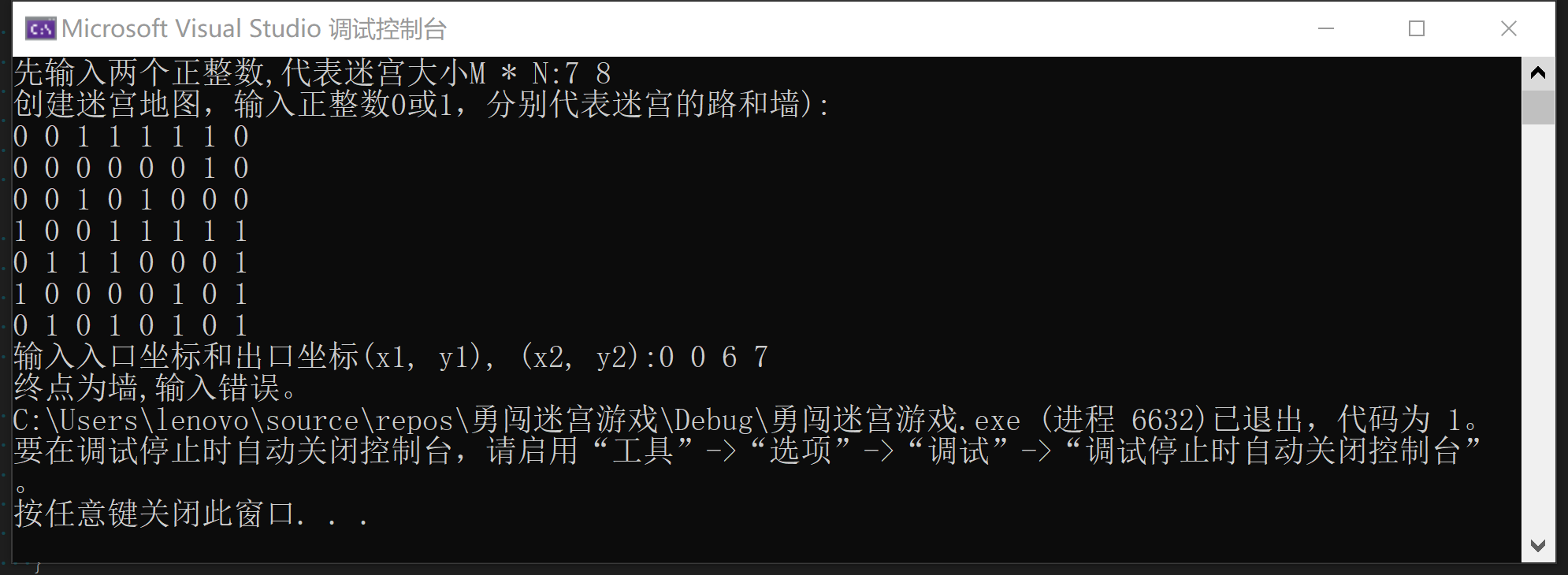
0 1 1 1 0 0 0 1

1 0 0 0 0 1 0 1

0 1 0 1 0 1 0 1

0 0 6 7

****实验结果：****

********