**数据结构课程设计**

**项目说明文档**

**勇闯迷宫游戏**

作 者 姓 名： 苏家铭

学 号： 2151299

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 项目分析 1](#_Toc27853)

[1.1 项目背景 1](#_Toc14737)

[1.2项目要求 1](#_Toc27006)

[1.2.1 功能要求 1](#_Toc25846)

[1.2.2 项目实例 1](#_Toc9983)

[2 项目设计及实现 2](#_Toc258)

[2.1 数据结构设计思路 2](#_Toc6736)

[2.2 类设计 2](#_Toc342)

[2.2.1 链表 2](#_Toc30886)

[2.2.2 链式栈 4](#_Toc32256)

[2.2.3 队列 5](#_Toc6847)

[2.2.4 堆 5](#_Toc12815)

[2.2.5 优先级队列 7](#_Toc1965)

[2.3 项目算法 8](#_Toc18328)

[2.3.1 实现思路 8](#_Toc19254)

[2.3.2 代码实现 8](#_Toc9519)

[3 项目测试 14](#_Toc7163)

[3.1 一般情况 14](#_Toc20448)

[3.2 较大数组 15](#_Toc14336)

[3.3 边界情况：起点与终点重合 16](#_Toc20008)

[3.4 没有可行路径情况 17](#_Toc21067)

[3.5 广度、深度、A星路径不一样情况（展示特点） 18](#_Toc7040)

[4 算法性能分析 18](#_Toc696)

[4.1 正确性 18](#_Toc31095)

[4.2 可使用性 18](#_Toc28983)

[4.3 可读性 19](#_Toc4721)

[4.4 效率 19](#_Toc1)

[4.5 健壮性 19](#_Toc17284)

# 1 项目分析

## 项目背景

迷宫只有两个门，一个门叫入口，另一个门叫出口。一个骑士骑马从入口进入迷宫，迷宫设置很多障碍，骑士需要在迷宫中寻找通路以到达出口。在现实生活中总会遇到求路径的问题，或是求最短路径，或是仅求可行路径，而从迷宫起点通向迷宫终点有很多条路，寻求一条合适的道路是一个值得深思的问题。

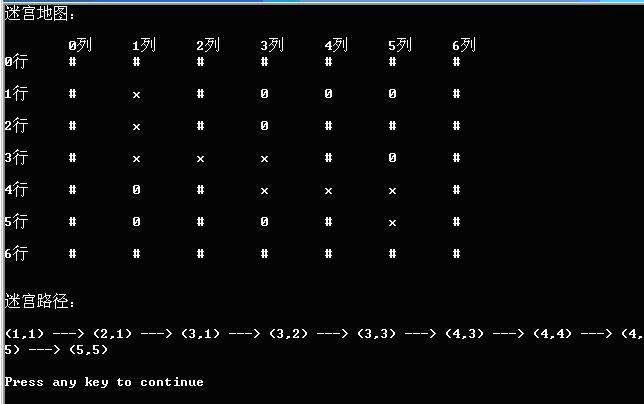
本项目依托此例为背景，让我们探究寻求路径的基本算法，为往后编写更深的算法打牢根基。

## 1.2项目要求

### 1.2.1 功能要求

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。

### 1.2.2 项目实例



# 2 **项目设计及实现**

## 2.1 **数据结构设计思路**

根据项目要求，本例可以采用多种算法实现寻路过程，其中需要使用到的数据结构有链表、栈、优先级队列等。同时在实现优先级队列时使用了最小堆，以下是对各种数据结构的类实现说明。

## 2.2 **类设计**

### 2.2.1 链表

虽然带头结点的双向循环链表结构复杂，但使用代码后会发现许多优势。该链表一般包括两个抽象数据类型（ADT）：链表结点类（DblNode）与链表类（DblList），而两个类之间的耦合关系可以采用嵌套、复合等多种关系。若结点类定义为class那么需要在类内定义友元函数链表类才能访问，为了实现代码的复用性，本人使用结构体来定义链表的结点，此方法虽然使得DblNode类失去了封装性，但链表类可以直接访问结点中的数据成员。

节点类数据成员：

template<class T>

T data;//链表结点数据

DblNode<T>\* lLink,\*rLink;//链指针域,前驱（左链）、后继（右链）

链表类数据成员：

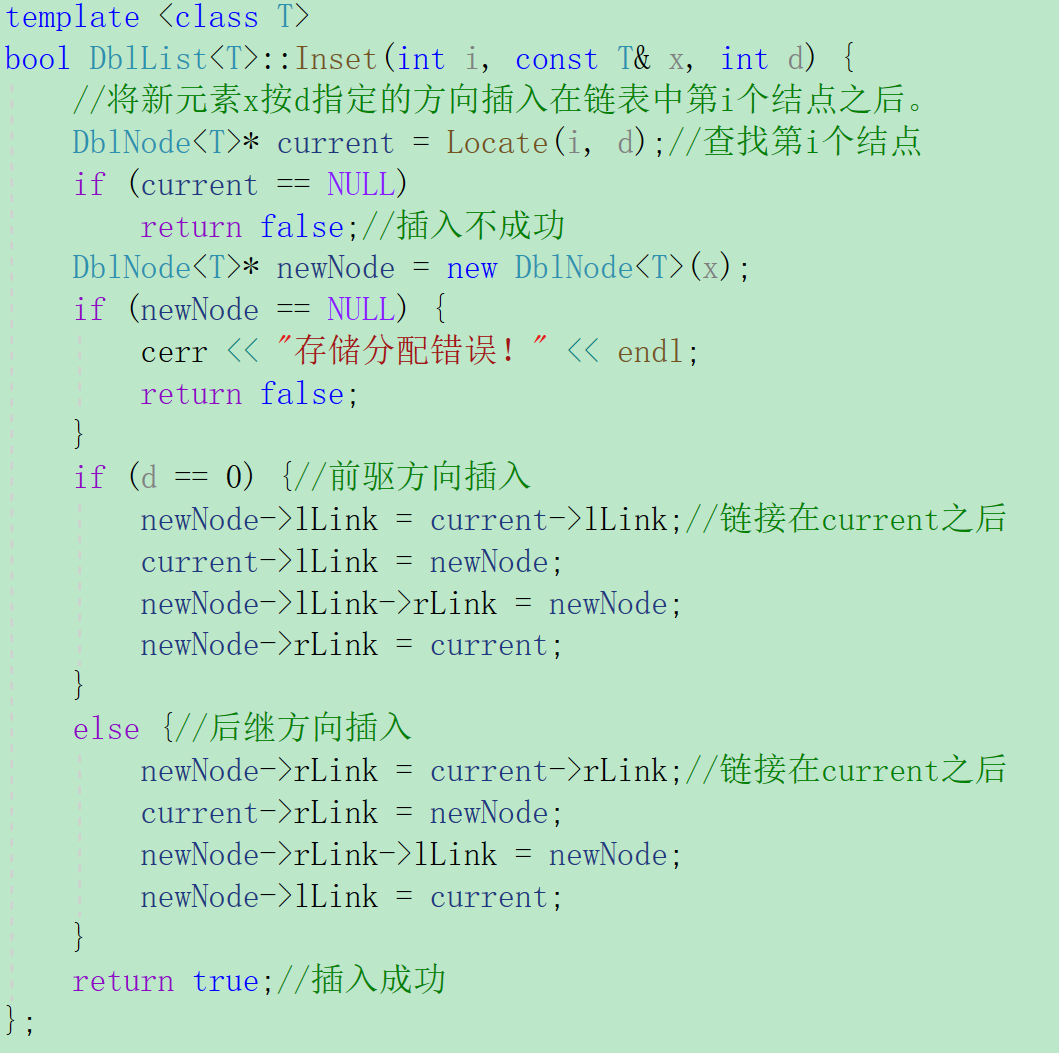
DblNode<T>\* first;//链表头结点

该链表已经实现19种功能：置空、计算长度、返回头结点、设置头结点、搜索、定位、返回元素地址、修改元素的值、插入、删除、置空、判空、判满、尾部插入、头部删除、向后插入i个元素、向后插入若干元素（直到遇到endTag）、输出链表、等号重载。

具体函数为：

* void makeEmpty();//将链表置为空表
* int Length()const;//计算双链表的长度
* void setHead(DblNode<T>\* ptr)//设置附加头结点地址
* DblNode<T>\* getHead()const//返回附加头结点地址
* DblNode<T>\* Search(const T& x);//在链表中沿后继寻找等于数据x的元素
* DblNode<T>\* Locate(int i,int d);//在链表中定位序号为i（>=0）的结点，d=0按前驱方向，d!=0按后继方向
* bool getDate(int i, T& x,int d);//取出按d方向第i个元素的地址
* void setData(int i, T& x,int d);//用x修改按d方向第i个元素的值
* bool Inset(int i, const T& x,int d);//在第i个元素后插入x,d=0按前驱方向，d!=0按后继方向
* bool Remove(int i, T& x,int d);//删除第i个元素,x返回该元素的值,d=0按前驱方向，d!=0按后继方向
* bool IsEmpty()const//判表空否?空则返回true
* bool IsFull()const {return false;}//判表满否？不满则返回false
* void push\_back(T data);//在尾部插入一个数据
* void push\_front(T data);//在前部插入一个数据
* void input\_num(int i);//输入i个元素
* void input\_endTag(T endTag);//输入若干元素，直到遇到endTag
* void output(int d);//输出
* DblList<T>& operator= (const DblList<T>&L);//重载函数：赋值

插入算法的代码实现：



主要算法的时间复杂度：

* 插入：O(n)
* 删除：O(n)
* 搜索：O(n)
* 遍历：O(n)
* 修改：O(n)

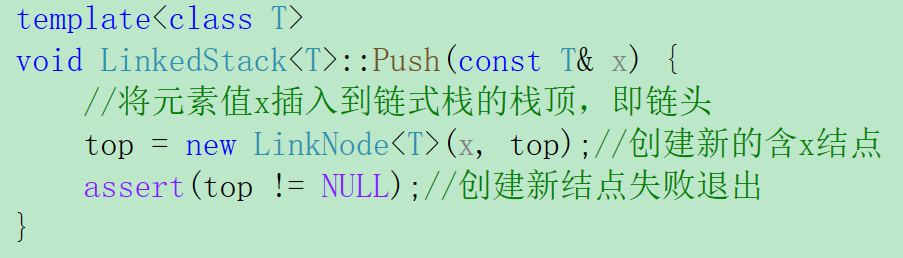
### 2.2.2 链式栈

[栈](https://so.csdn.net/so/search?q=%E6%A0%88%E5%92%8C%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41575507/article/details/_blank)是重要的线性结构，从数据结构的角度看，栈也是线性表，其特殊性在于栈的基本操作是线性表的子集。这是一种是操作受限的线性表，因此，可称为限定性的数据结构。但从数据类型角度看，他是和[线性](https://so.csdn.net/so/search?q=%E7%BA%BF%E6%80%A7&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41575507/article/details/_blank)表大不相同的重要的抽象数据类型。栈(stack)是限定仅在表尾进行插入或者删除的线性表。对于栈来说，表尾端称为栈顶(top)，表头端称为栈低(bottom)。因为栈限定在一端进行插入或者删除，所以栈又被称为后进先出的线性表（LIFO）。

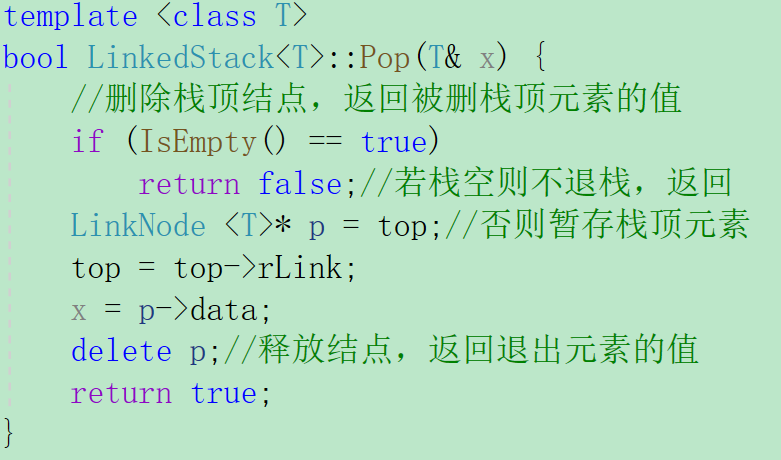
本人基于链表实现了链式栈的进栈、退栈、返回栈顶元素、返回栈内元素个数、置空、判空以及左移运算符的重载操作

* void Push(const T& x);//一个元素进栈
* bool Pop(T& x);//一个元素退栈
* bool getTop(T& x)const;//返回栈顶元素
* bool IsEmpty()const //判空
* int getSize()const;//返回栈的元素个数
* void makeEmpty();//清空栈
* friend ostream& operator << (ostream& os, LinkedStack<T>& s);//输出

进栈算法代码实现：



出栈算法代码实现：



主要算法的时间复杂度：

* 进栈：O(1)
* 退栈：O(1)
* 返回栈顶元素：O(1)
* 返回元素个数：O(n)

### 2.2.3 队列

队列(Queue)是只允许在一端进行插入，而在另一端进行删除的运算受限的线性表，允许插入的一端叫做队尾(back)，允许删除的一端叫做队头(front)。队列也叫做先进先出的线性表，即FIFO(First In/First Out).

而本项目基于已经实现了的双向循环链表，实现了链式队列。每次入队时在链表的末尾插入一个元素，出队时则删除链表的第一个元素。出队入队都可以通过链表的first指针的左右指针找到队首和队尾，则无需再定义last指针，也无需遍历一遍链表找队尾元素，即出队、入队的时间复杂度可以降为O(1).

队列类数据成员：

template<class T>

DblList<T> list;

该链式队列已经实现6种功能：入队、出队、置空、判空、返回队首元素、返回队列长度。

具体函数声明为：

* void push(const T& val);//入队
* void pop();//出队
* void makeEmpty();//置空
* bool empty()const;//判断空否
* int size()const;//返回队列大小
* const T& front();//返回队首元素

主要算法的时间复杂度：

* 入队：O(1)
* 出队：O(1)
* 置空：O(n)
* 判断空否：O(1)
* 返回队首元素：O(1)
* 返回队列大小：O(n)

### 2.2.4 堆

堆(heap):若有一个关键码的集合K = {k0，k1，k2，…，kn-1}，把它的所有元素按[完全二叉树](https://so.csdn.net/so/search?q=%E5%AE%8C%E5%85%A8%E4%BA%8C%E5%8F%89%E6%A0%91&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/zgdlxs/article/details/_blank)的顺序存储方式存储在一个一维数组中，并满足：Ki <= K2i+1 且 Ki <= K2i+2，则称为小堆(或大堆)。将根节点最大的堆叫做最大堆或大根堆，根节点最小的堆叫做最小堆或小根堆。

本项目中堆基于一维数值来实现，通过定义建堆、调整堆来完成了一个堆。

最小堆类数据成员：

template <class T>

T\* pheap;//存放小根堆中元素的数组

int current\_size;//小根堆中当前元素个数

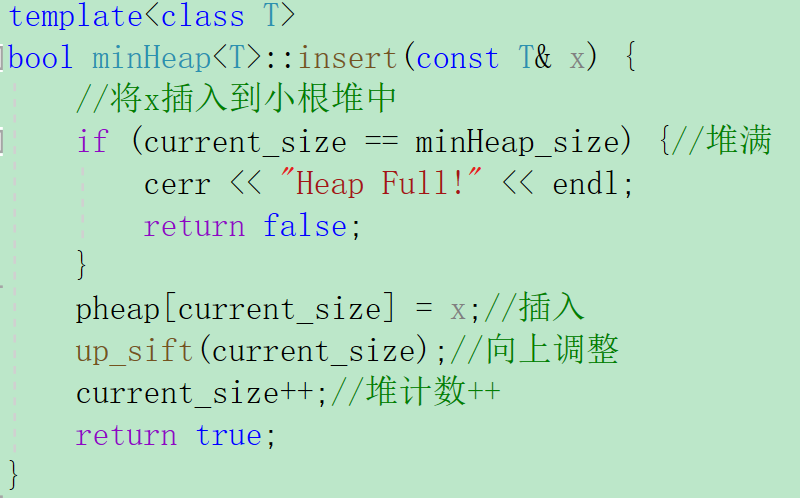
int minHeap\_size;//小根堆最多允许元素个数

该最小堆已经实现8种功能：向上调整、向下调整、插入元素、删除堆顶元素、判空、判满、置空堆、堆排序。

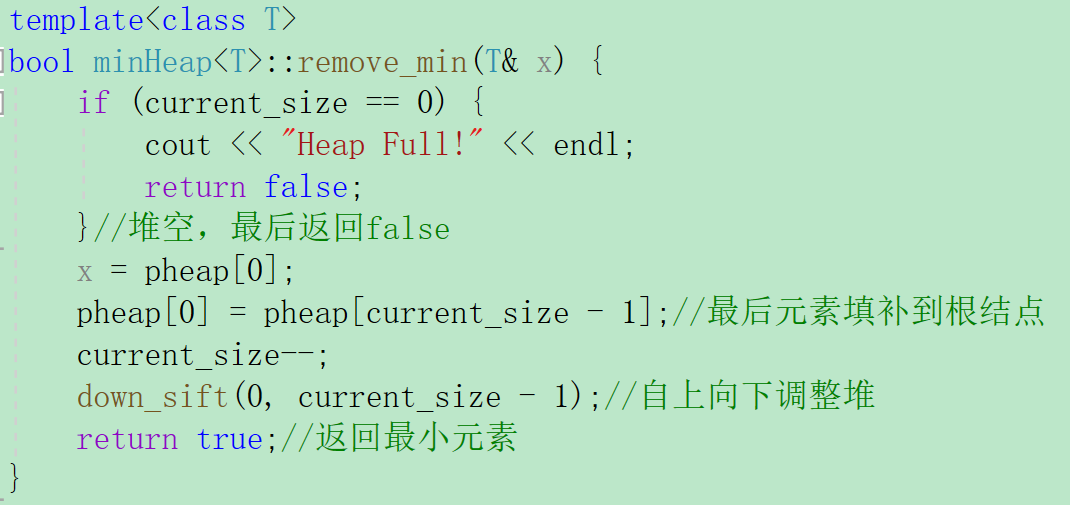
具体函数声明为：

* void down\_sift(int start, int m);//自上向下调整
* void up\_sift(int start);//自下向上调整
* void makeEmpty();//置空
* bool insert(const T& x);//将x插入到小根堆中
* bool remove\_min(T& x);//删除小根堆上的最小元素
* bool empty()const ;//判断堆是否为空
* bool full()const ;//判断堆是否达到最大限制个数
* void make\_empty();//置空堆
* void rank\_heap(T rank[]);//堆排序

插入算法代码实现：



删除算法代码实现：



主要算法的时间复杂度：

* 插入：O(log2n)
* 删除：O(log2n)
* 置空：O(n)
* 判断空否：O(1)

在最后为了能在优先级队列中访问堆的数据成员和成员函数，将优先级队列声明为友元函数：

//友元类

template <class E>

friend class minPQueue;

### 2.2.5 优先级队列

一般的队列是一种特征为FIFO的特殊线性表，但是在实际情况中通常需要选取优先级最高的元素先出队列。为此，本项目实现了一种每次从队列中取出的应该是具有最高优先权的元素的队列，也即优先级队列(Priority Queue)

本项目中最小优先级队列基于小根堆来实现，每次出队则返回选取小根堆的堆顶元素并删除调整，完成了优先级队列的出队过程。

最小优先级队列类数据成员：

template <class T>

minHeap<T> heap;

与一般队列类似，该最小优先级队列也实现6种功能：入队、出队、置空、判空、返回队首元素、返回队列长度。

具体函数声明为：

* void push(const T& val);//入队
* void pop();//出队
* void makeEmpty();//置空
* bool empty()const;//判断空否
* int size()const;//返回队列大小
* const T& front();//返回队首元素

主要算法的时间复杂度：

* 入队：O(log2n)
* 出队：O(log2n)
* 置空：O(n)
* 判断空否：O(1)
* 返回队首元素：O(1)
* 返回队列大小：O(1)

## 2.3 **项目算法**

### 2.3.1 实现思路

对于迷宫算法的求解通常有广度优先搜索(BFS)和深度优先搜索(DFS)以及A星算法，其中广度优先搜索通常使用队列或寻路结构体数组作为辅助工具来实现，而深度优先搜索算法通常可以分为用递归实现和用栈实现两种实现方式。

* BFS算法过程：

1.取出队列首元素作为当前节点，检查是否为出口结点。

2.对当前结点的邻接顶点遍历，选取未遍历过的顶点，依次进入队列。

3.当队列为空时，表示所有路径已经探索完毕。

优点：获得最短路径（探索深度而不是权值）

缺点：极为巨大的内存需求（能够高达几十MB）造成程序堆栈溢出。

* DFS算法过程：

1.访问当前节点，若无法访问则退回父节点，并且出栈。

2.检查邻接结点中是否有出口结点。

3.依次进入子结点，邻接结点入栈。

优点：节省空间，得到路径。

缺点：时间复杂度高，不一定是最佳的结果。

* A星算法过程：

1.利用曼哈顿距离作为关键码，取出最小优先级队列首元素作为当前节点，检查是否为出口结点。

2.对当前结点的邻接顶点遍历，选取未遍历过的顶点，依次进入队列。

3.当队列为空时，表示所有路径已经探索完毕。

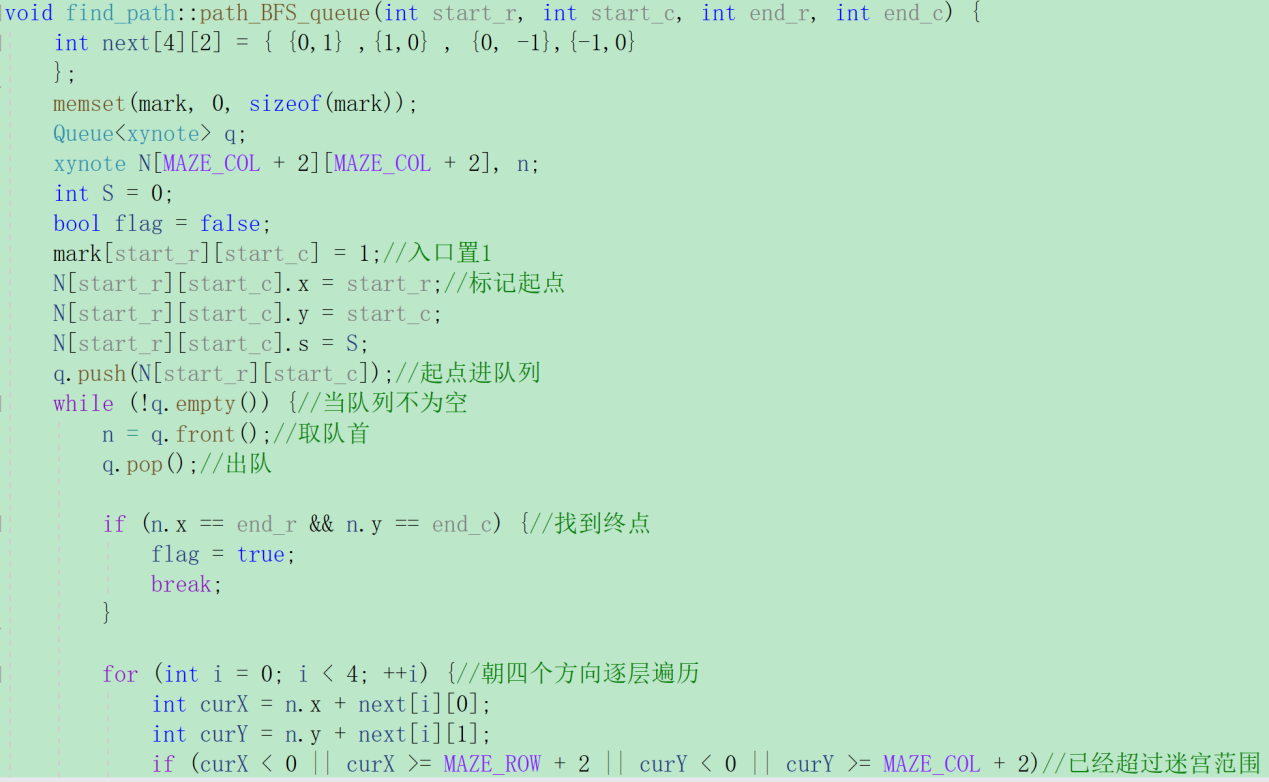
优点：效率较高

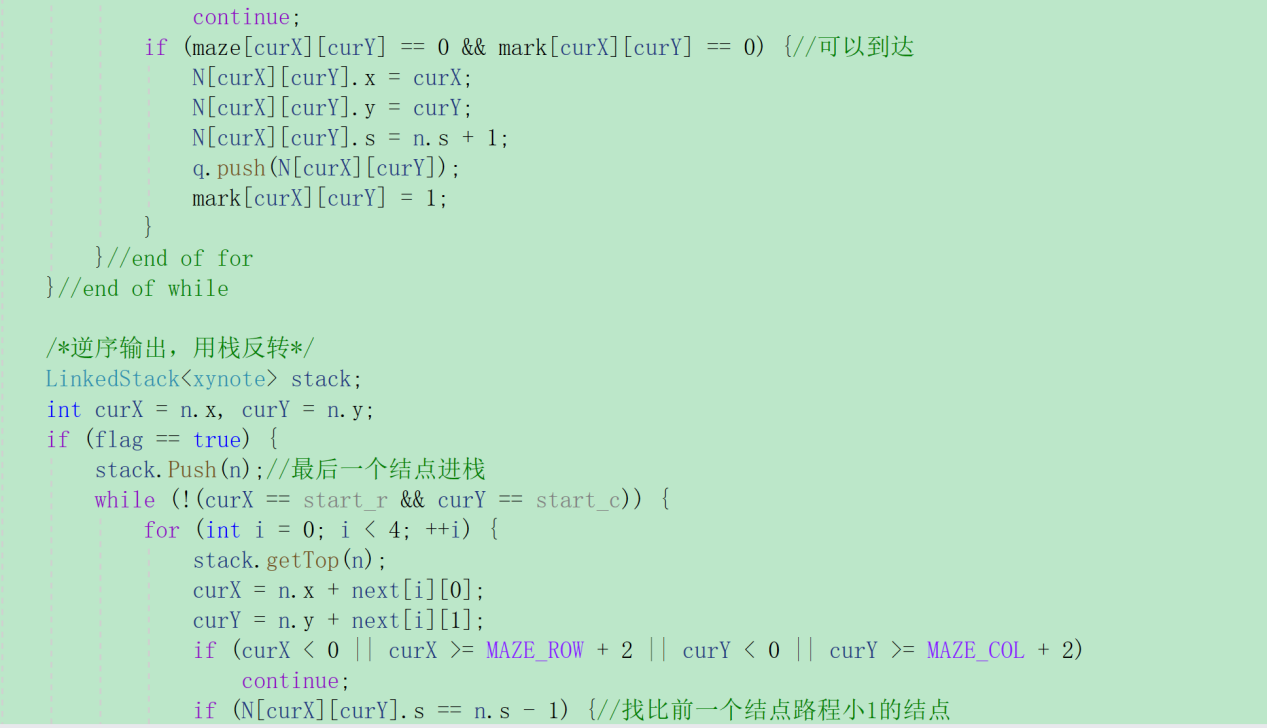
缺点：不一定是最佳结果

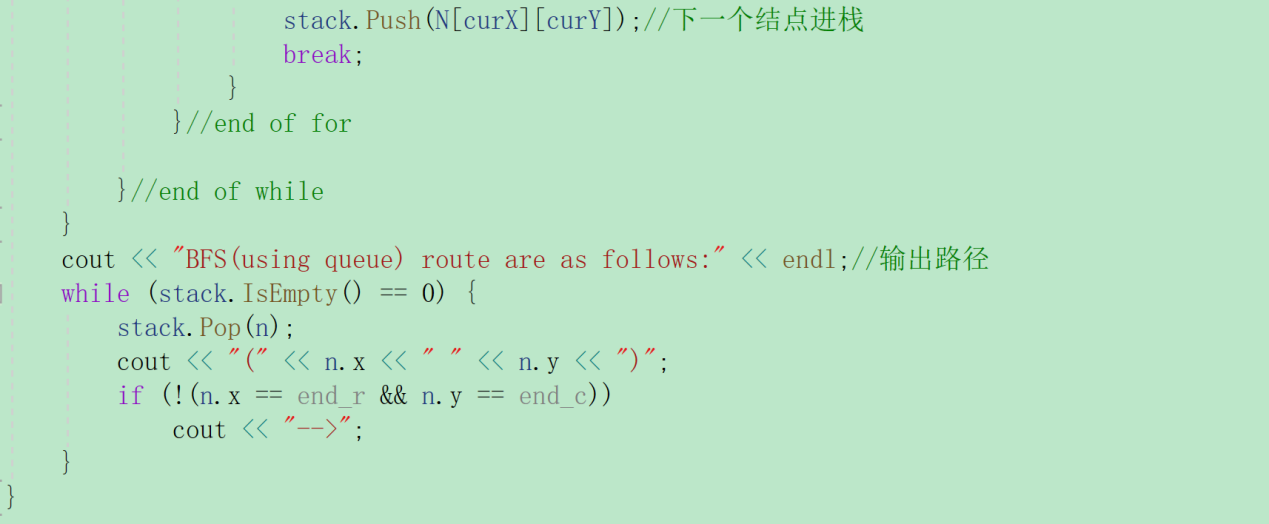
本项目基于以上思路，完成了五种求解迷宫路径的算法。

### 2.3.2 代**码实**现

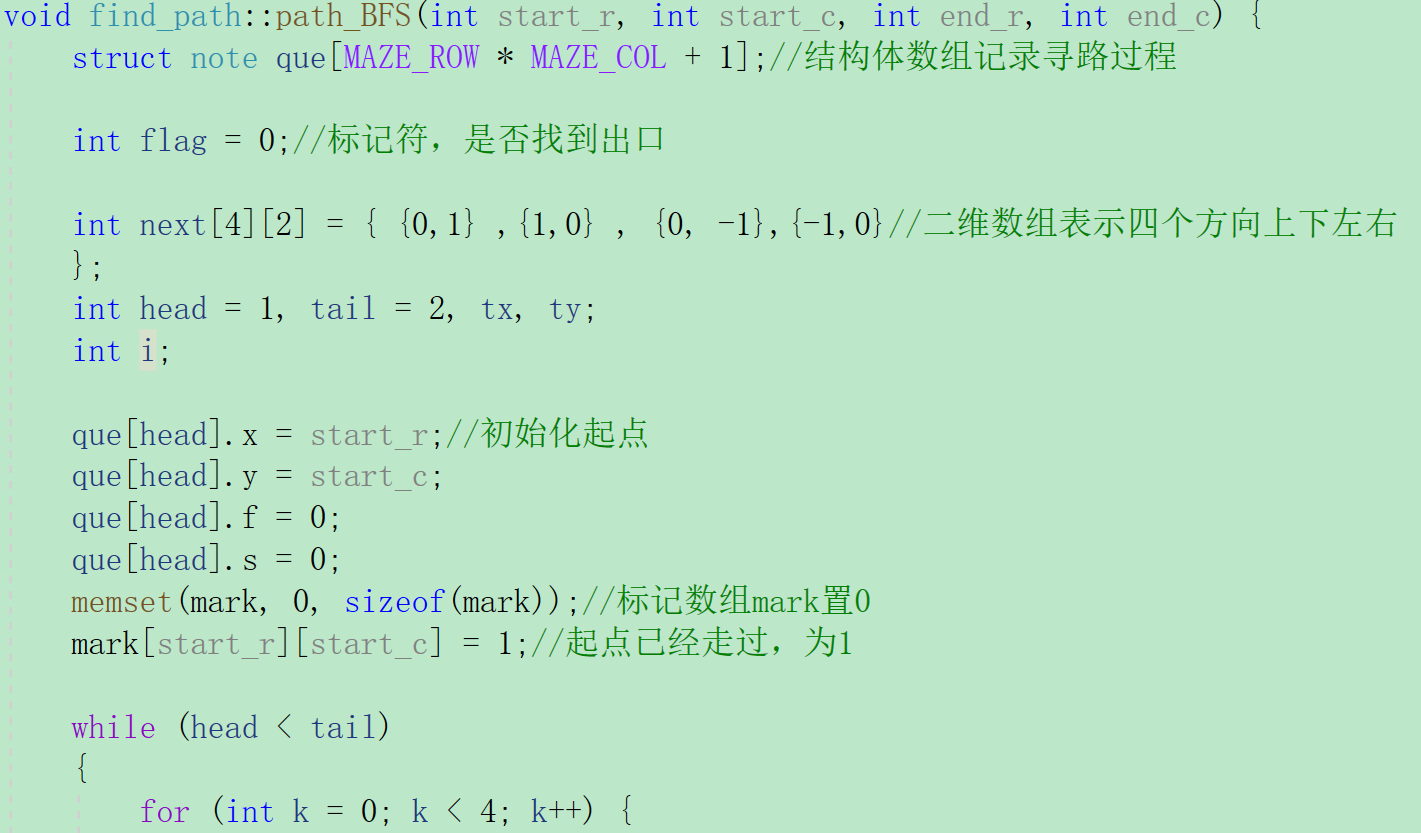
* BFS（借助队列）:

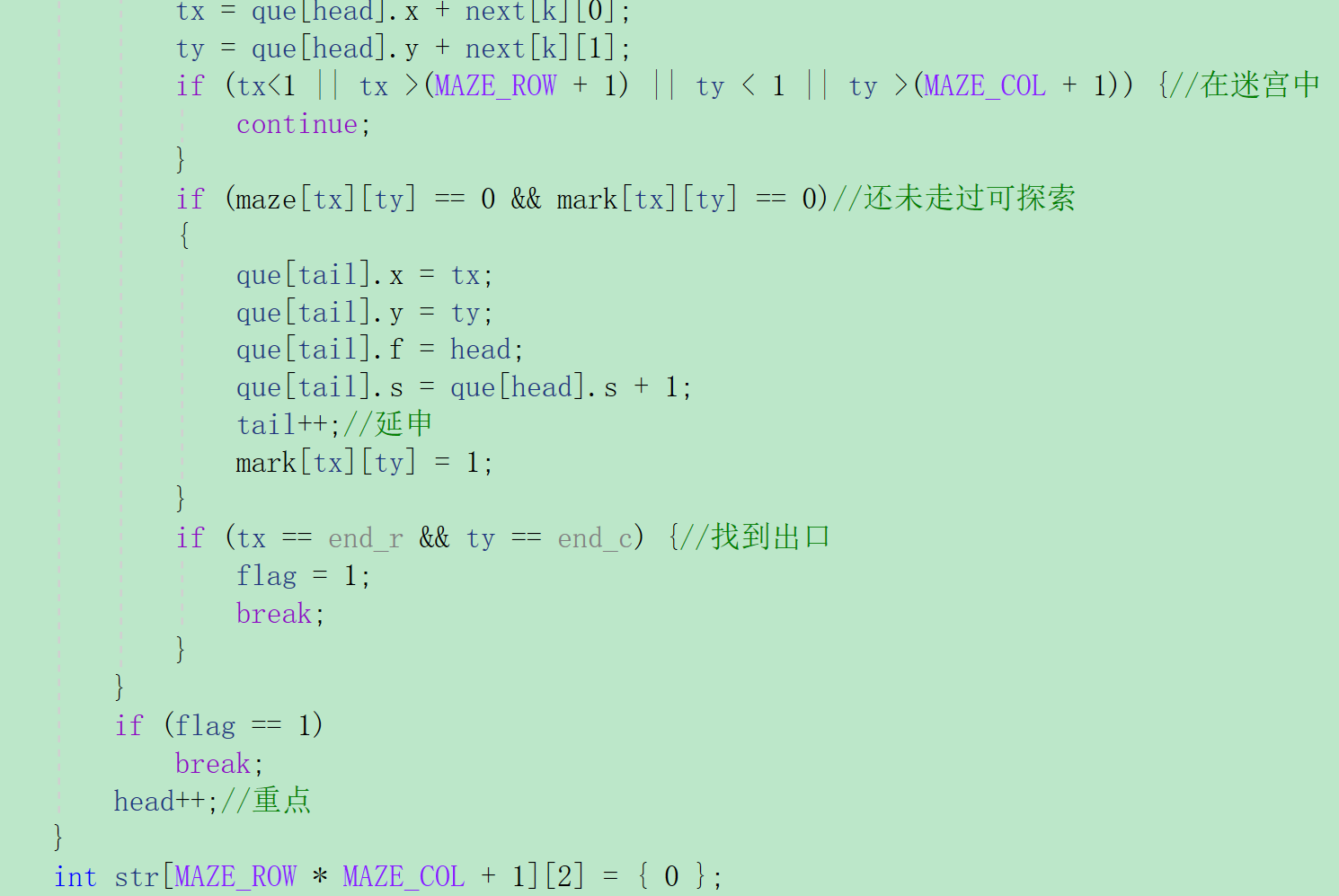


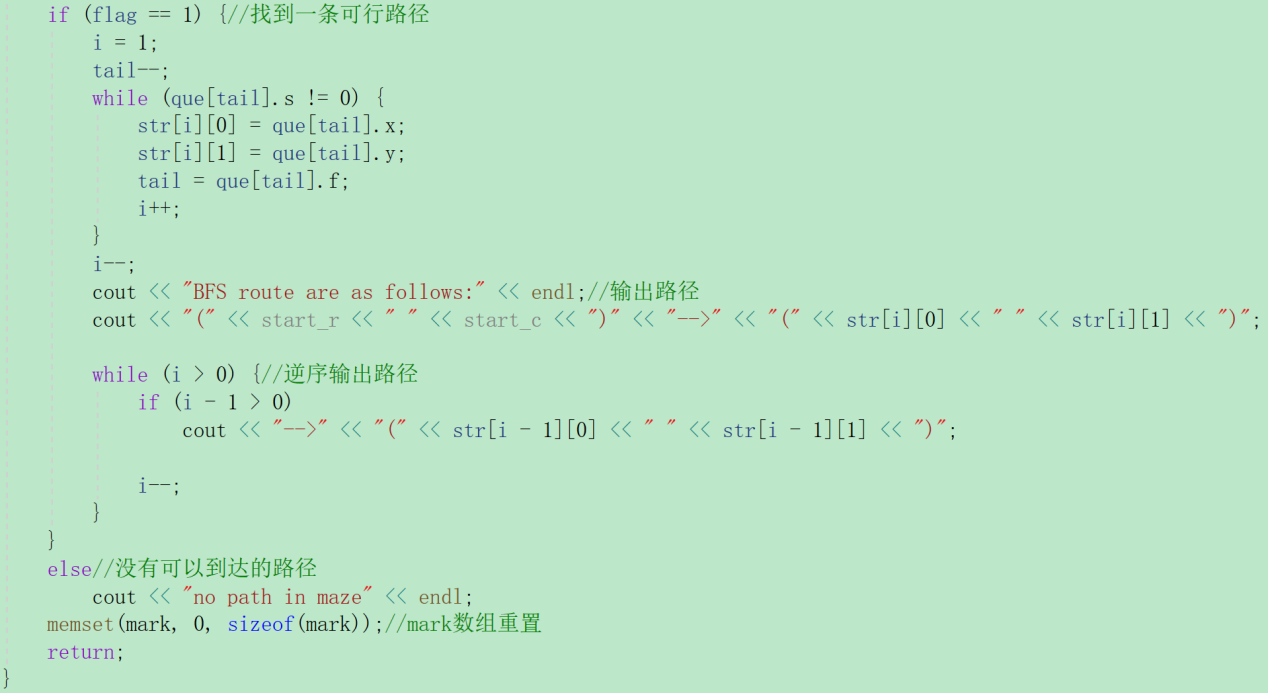




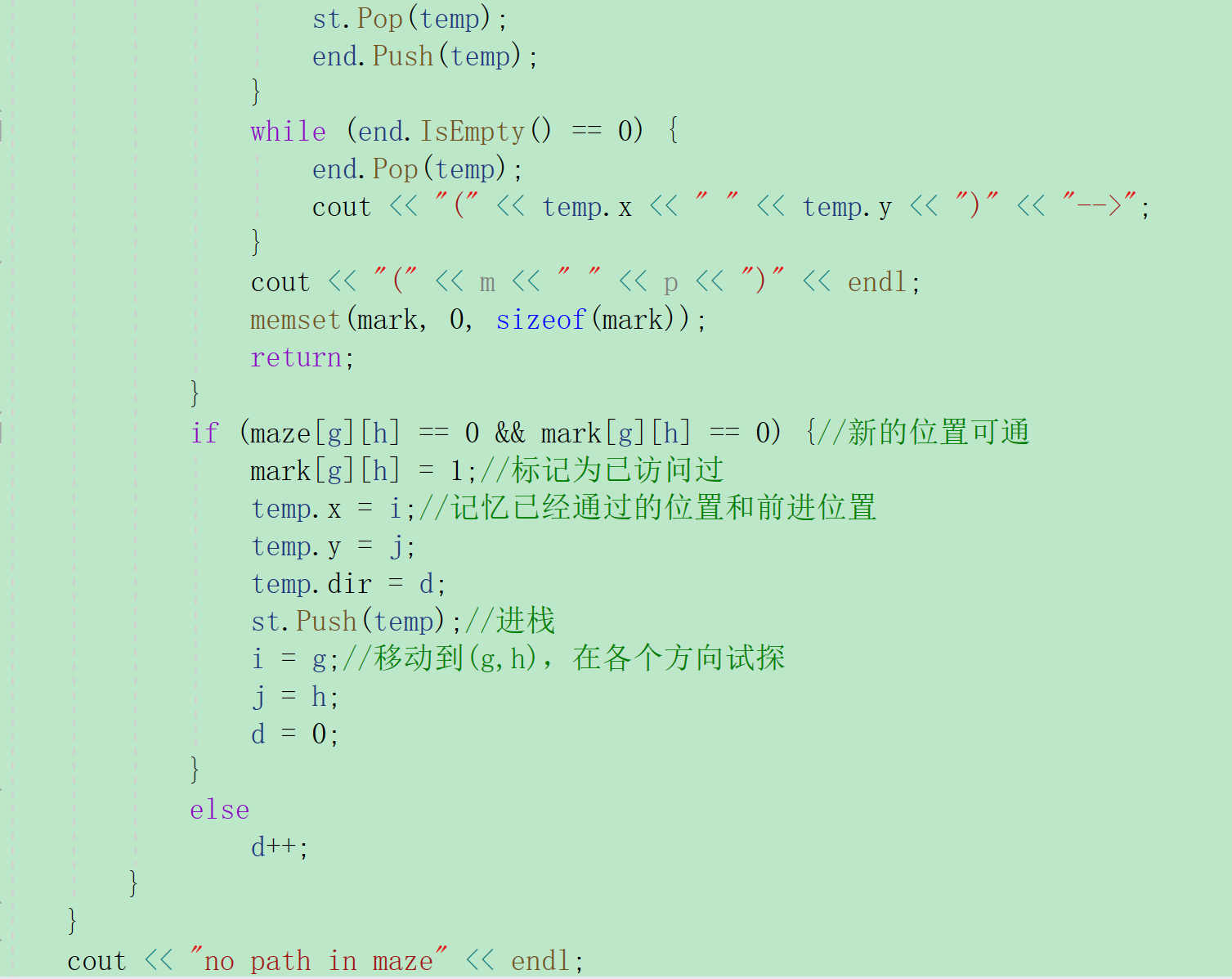
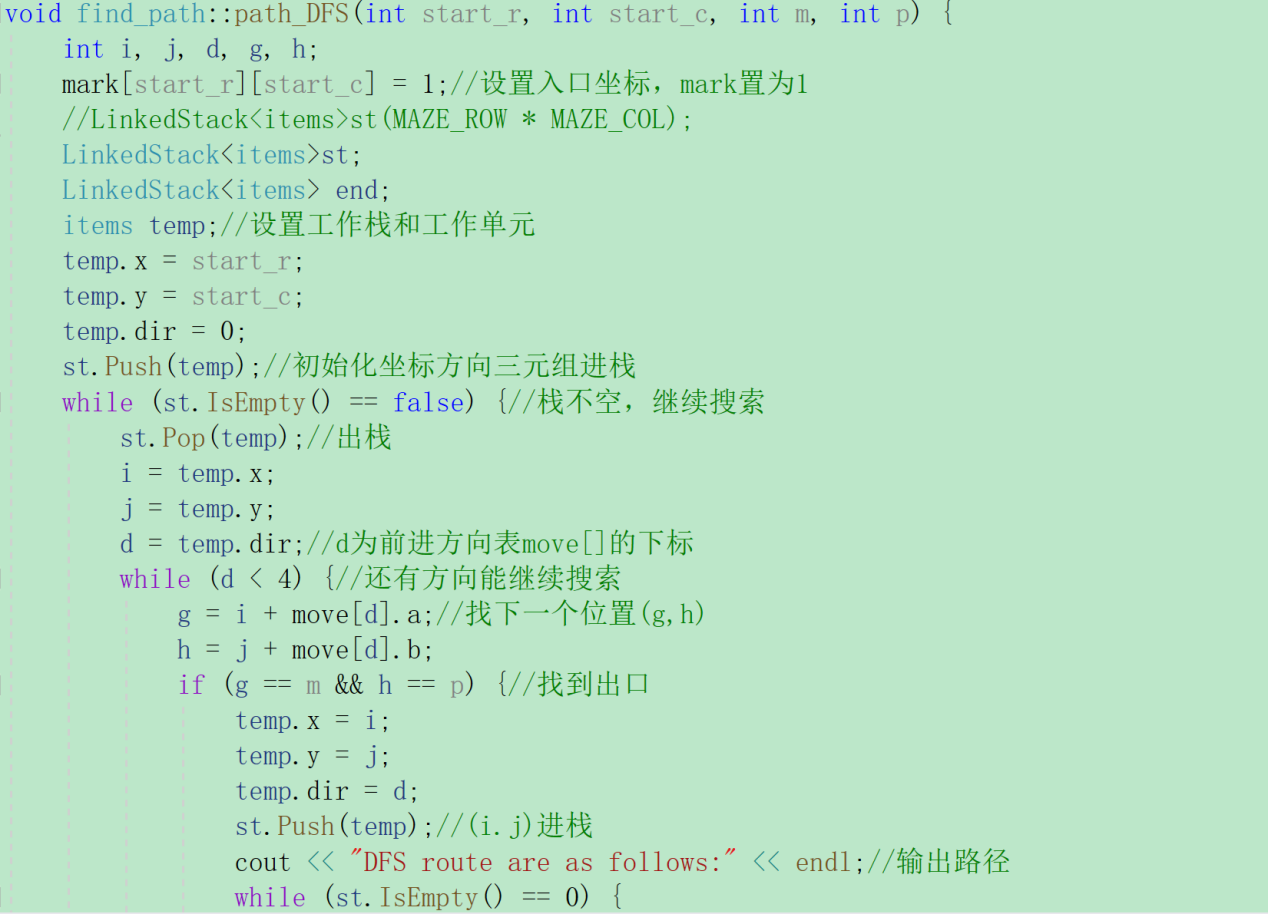
* BFS（借助结构体数组记录路径）:



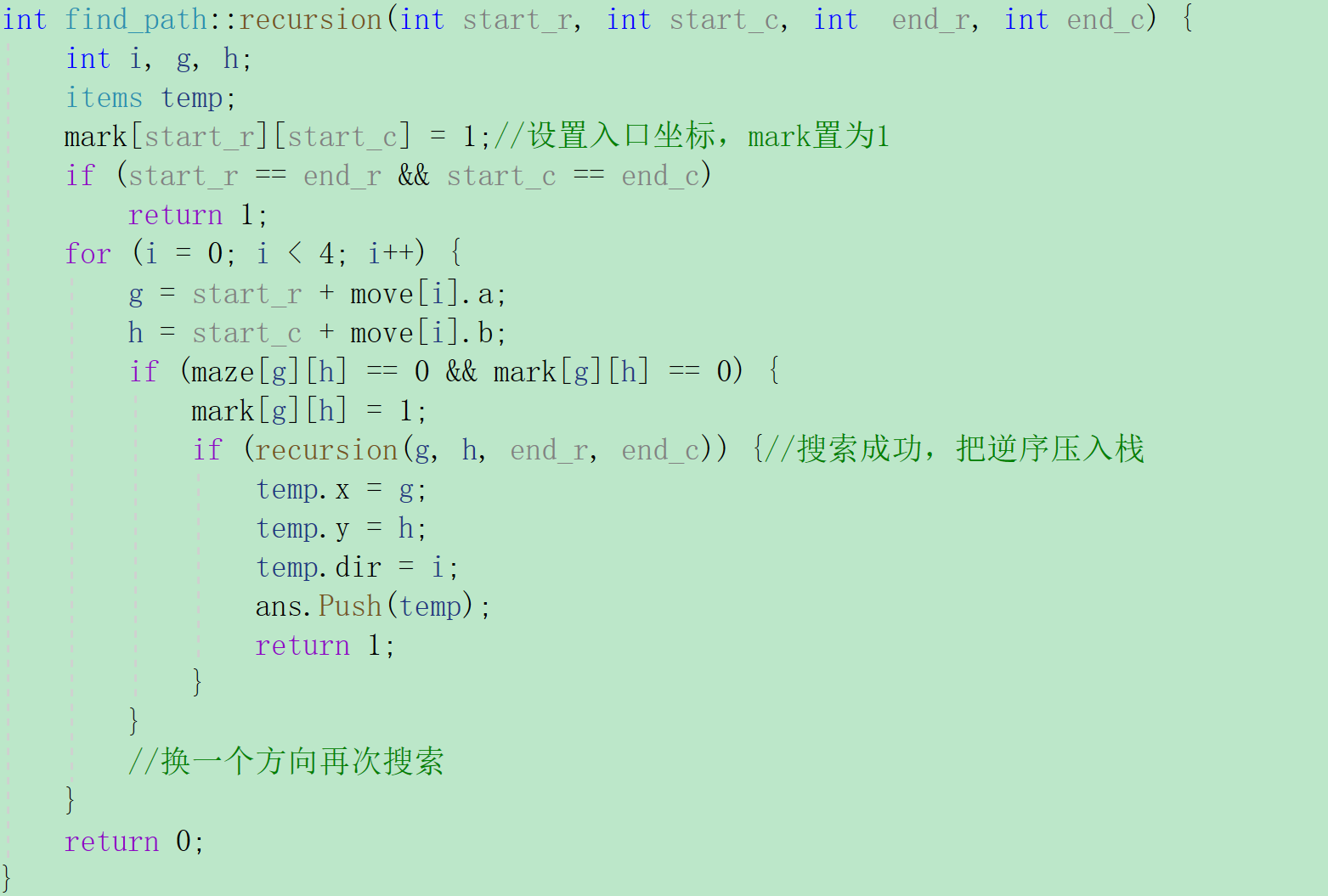




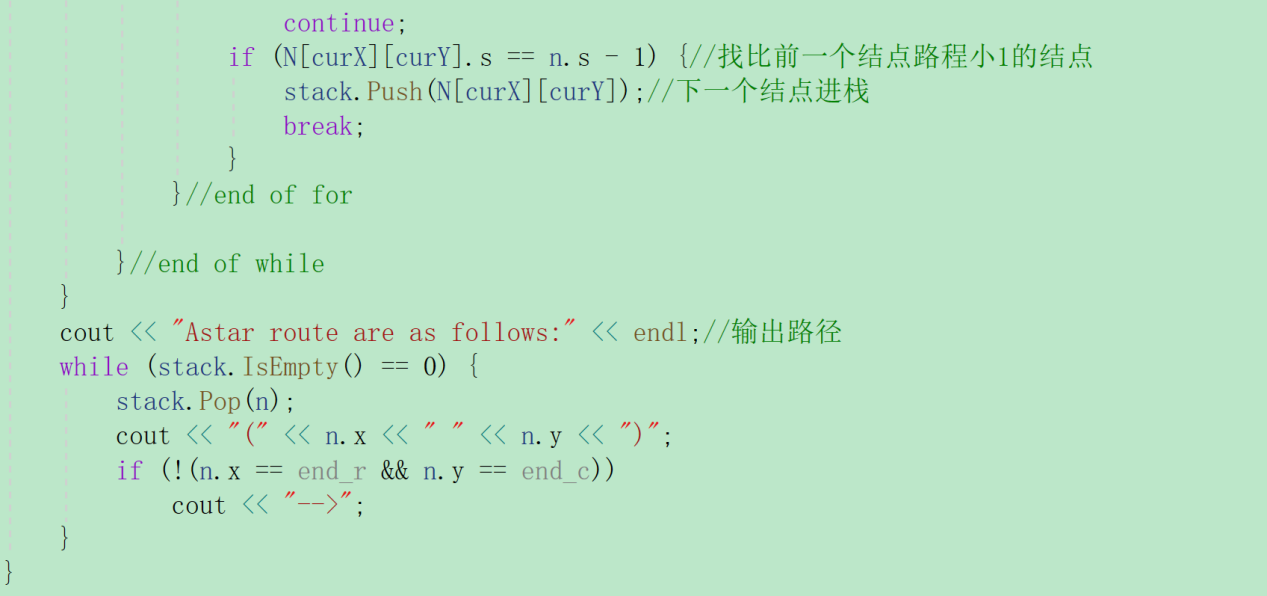
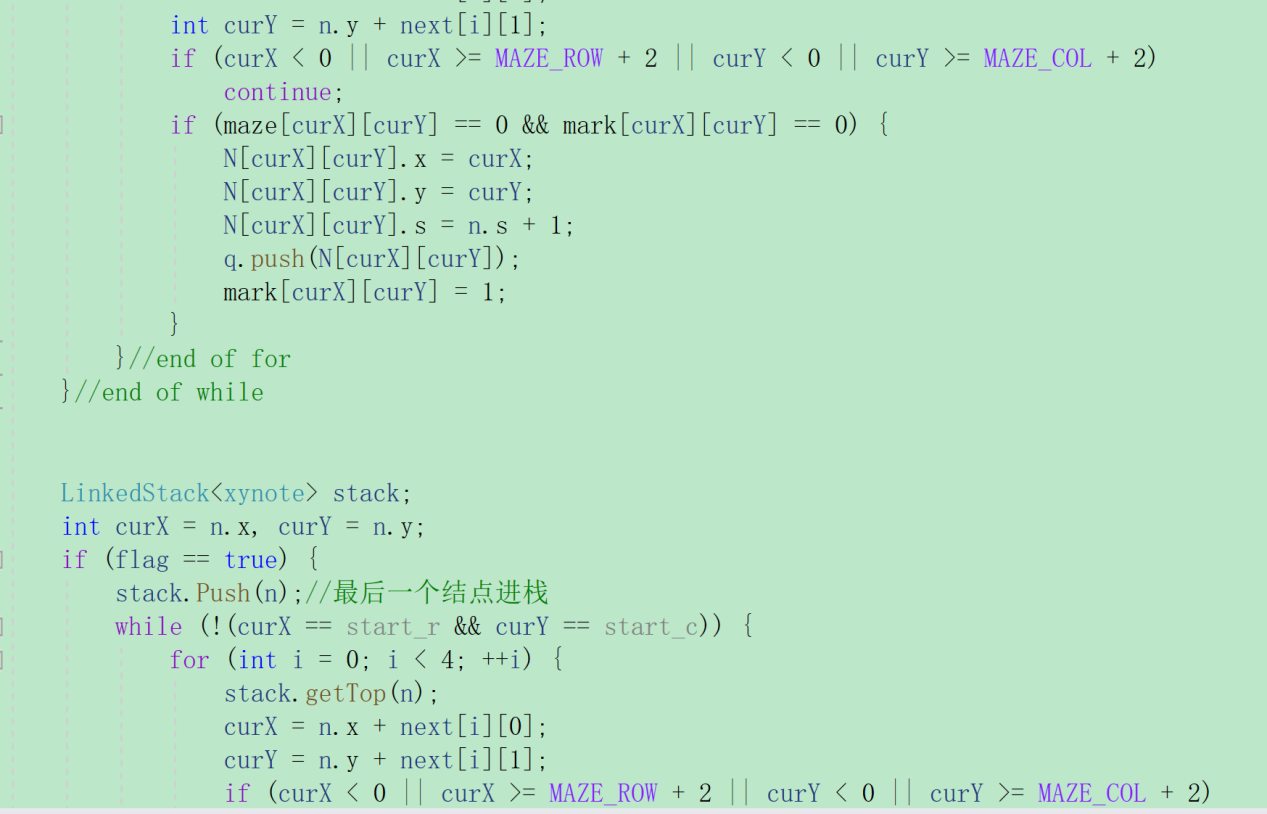
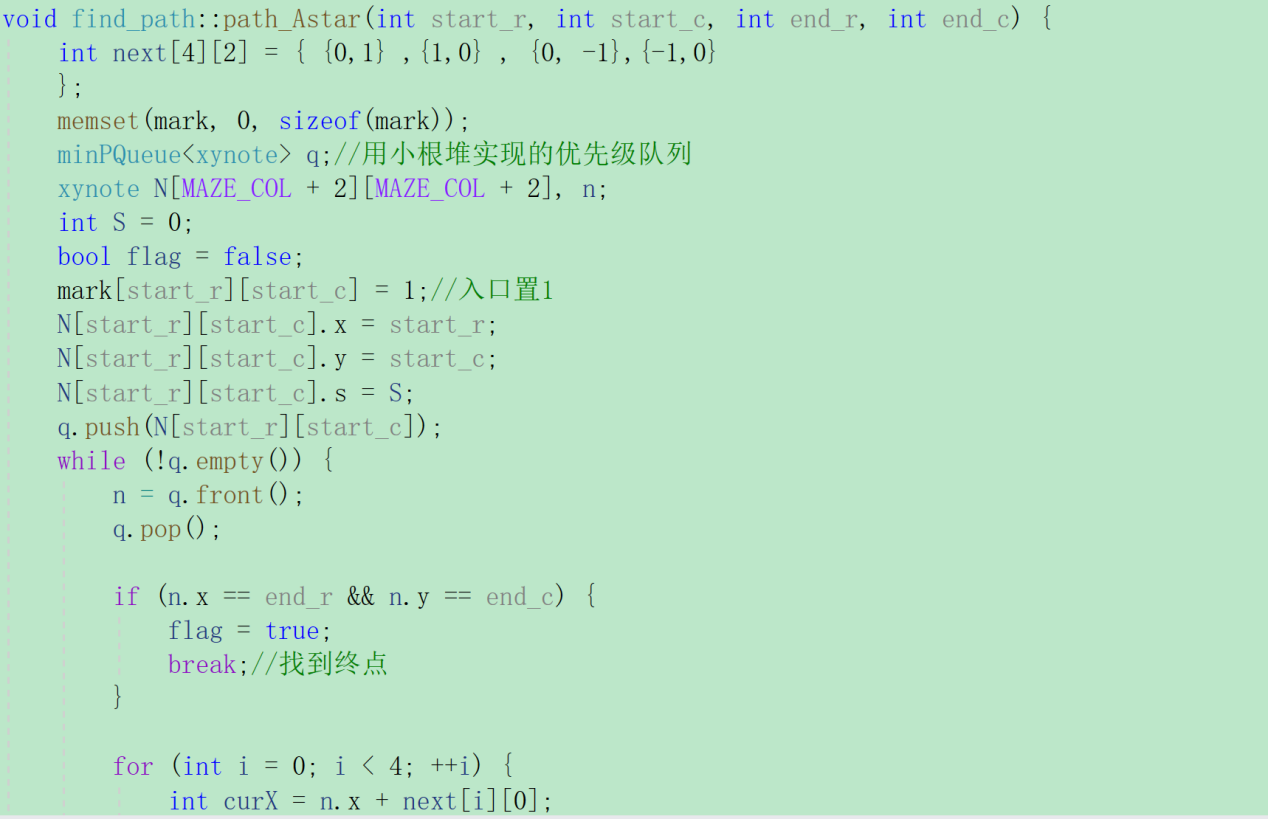
* DFS(借助栈)：



* DFS（递归实现）：



* A\*算法（BFS下运用最小优先级队列）：



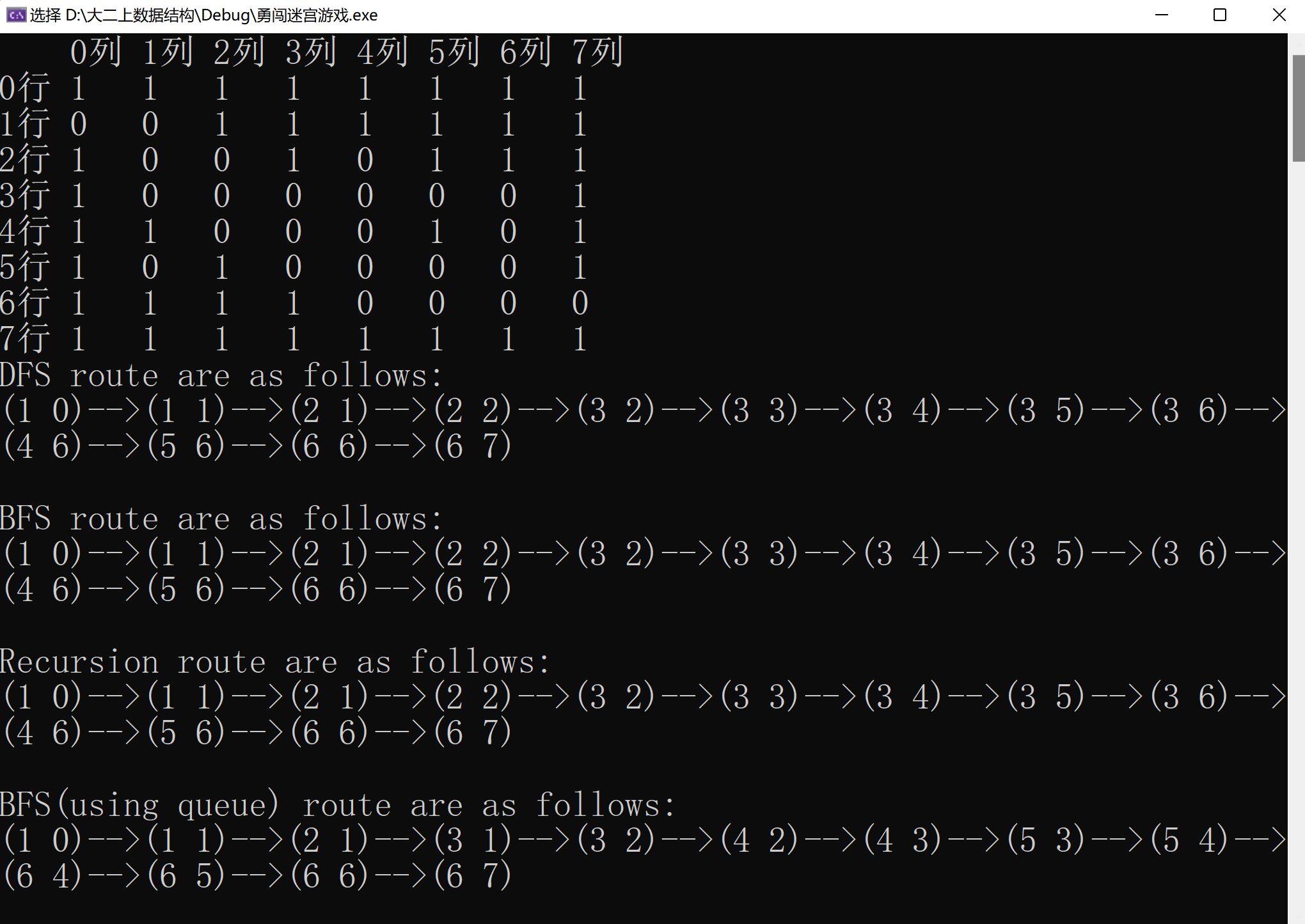
# 3 项目测试

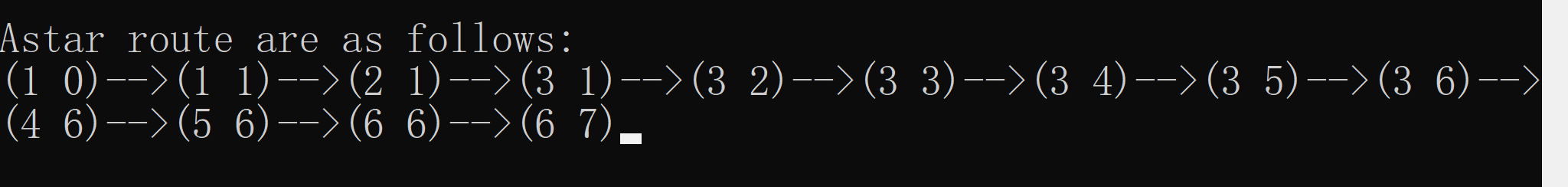
## 3.1 **一般情况**

测试数组：8行8列

预期输出：8X8的迷宫中从入口到出口的路径

实验结果：





## 3.2 较大数组

测试数组：12行12列

预期输出：12X12的迷宫中从入口到出口的路径

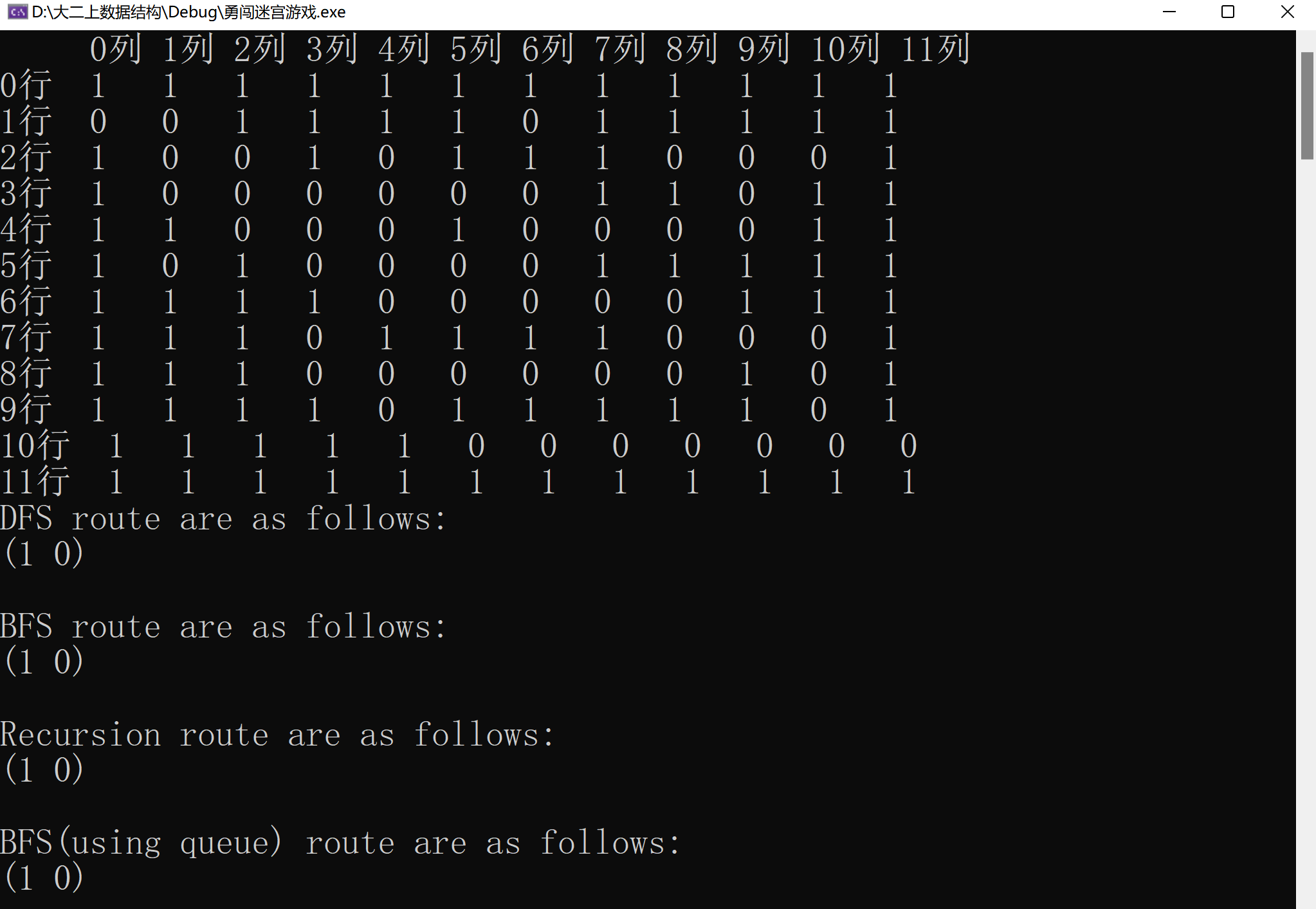
实验结果：

## 3.3 边界情况：起点与终点重合

测试数组：起点和终点都是(1, 0)

预期输出：只输出一个点

实验结果：

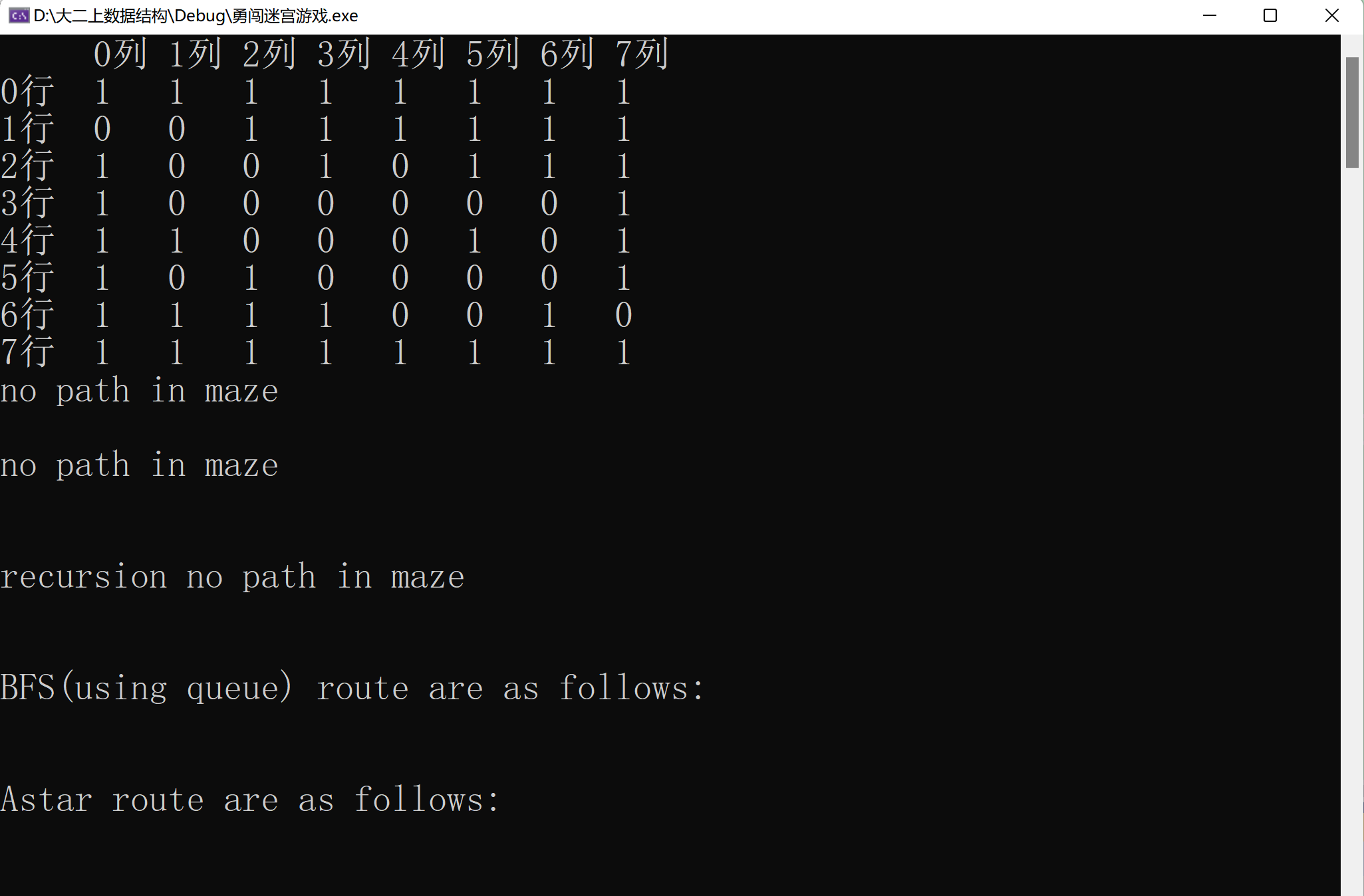


## 3.4 没有可行路径情况

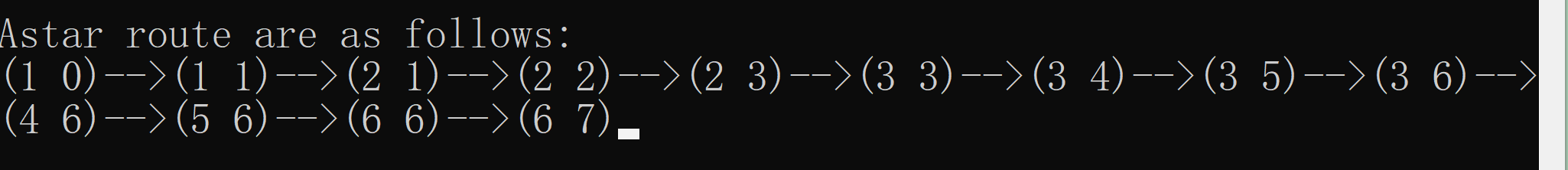
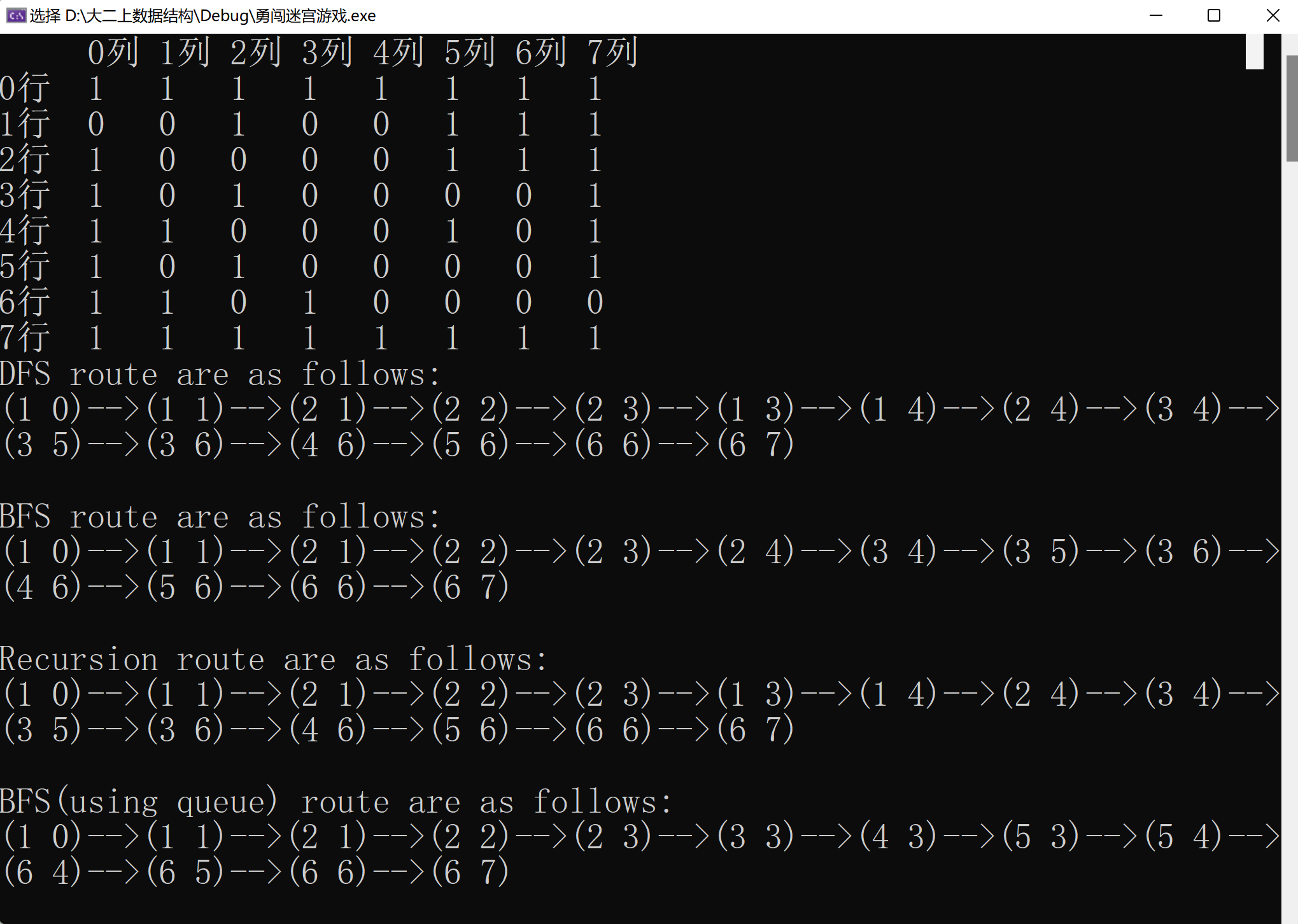
测试数组：无可行路径

预期输出：有相应提示

实验结果：



## 3.5 广度、深度、A星路径不一样情况（展示特点）



# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，各个算法都能得到一条从起点到终点的路径，且可以相互验证都为正确。

## 4.2 可使用性

本算法可以很方便地使用，五类求解算法都封装在各自的函数中，而且参数为起始位置和终止位置，可移植性强，复用性好。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 效率

广度优先搜索虽然一定可以找到最短路径，但搜索的效率没有深度优先搜索高，但在广度优先搜索使用队列的基础上优化为使用曼哈顿距离为权值的最小优先级队列，则可以实现更高效率的A星算法。

## 4.5 健壮性

本算法对于边界条件，诸如起点与终点重合或者没有可行路径的情况有具体显示。