题目三 勇闯迷宫游戏

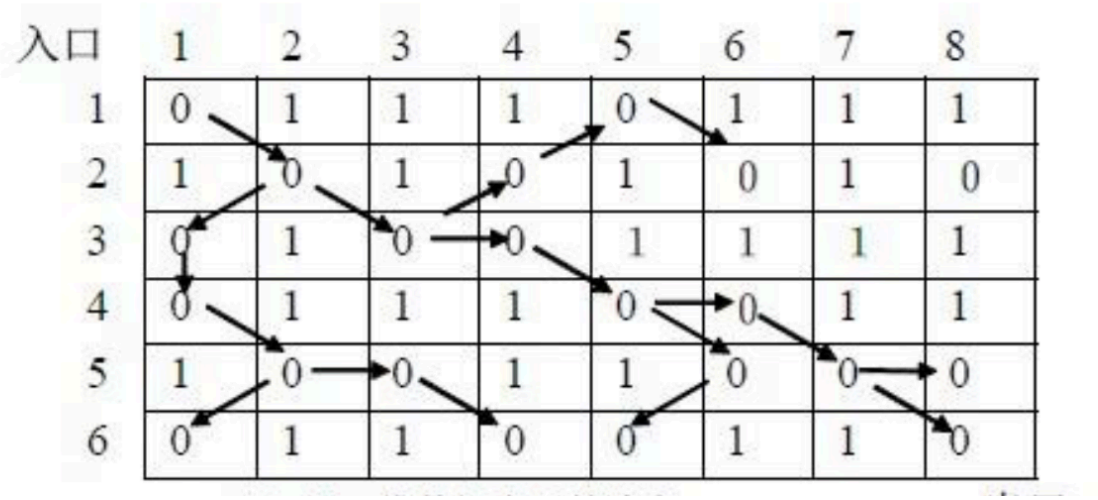
一.设计思路

勇闯迷宫游戏要求在一个迷宫中找到一条从入口位置到出口的路径。
迷宫中各个可走的位置可以看作一个个节点，这些节点连接起来，就相当于一个图。
该图具有如下特点：

- 该图为无向图
- 该图为无权图
- 可能为连通图，也可能不连通
- 图中可能存在回路

我们要做的是在这样一个图中给定的两个节点间找到一条通路。这样的通路可能不唯一。按照题目的说明，我们不需要找到最短的路径，也不需要求出所有的路线，所以不须要进行广度优先搜索（BFS）。为了达到目的，只需要从起点节点开始，进行深度优先搜索（DFS）即可。当搜索到出口节点时，搜索结束。

其原理可表示如下：



结合本题的特点，我们也不需要按照传统的方式新建一个图结构，只需要利用现有的地图信息即可。地图为可行的位置就是图的节点，两个节点间如果存在上下或左右相邻关系，则它们之间有边连接。
为了辅助进行深度优先搜索，需要建立一个栈。这个栈既作为搜索中保存先前路径的辅助数组，又作为最后的结果数组。
同时，为了方便函数的编写，需要新建立一个节点类型，在其上定义其自身的比较操作，输出操作等。

二.数据结构实现

1.节点类型 （PathPoint）

定义了结构体 `struct PathPoint` 作为树的结点，即家谱中每个成员。

1.1类成员

```
int x;       // 节点横坐标
int y;       // 节点纵坐标
```

定义了 `int` 类型的 `x, y` 记录该节点在迷宫地图中的坐标。

1.2构造函数

```
PathPoint():x(0),y(0){}      // 默认构造函数
```

定义了默认构造函数，用于一般新建成员结点。

```
PathPoint(int a = 0,int b = 0):x(a),y(b){}  // 传入坐标值的构造函数
```

定义了传入节点坐标构造节点的构造函数。简化后续新建节点操作。

1.3重载输出流

```
friend ostream& operator <<(ostream&, const PathPoint&);    // 自定义输出流
```

在 `struct PathPoint` 类中定义了友元函数，用以重载输出流，便于后续格式化输出节点信息。

运算符重载函数定义如下：

```
ostream& operator <<(ostream& out, const PathPoint& p){
    out<<'<'<<p.x<<','<<p.y<<'>';
   return out;
}
```

重载了 `<<` 运算符，以规定格式向输出流输出。

1.4重载比较运算符

```
bool operator == (const PathPoint& target){     // 自定义比较函数
    return (x==target.x&&y==target.y);
}
```

重载了比较运算符 `==` ，当两个节点横纵坐标均相同时，结果为真。
便于后续函数的编写。

2.路径栈（PathStack）

```
vector<PathPoint> PathStack;  // 栈存储路径
```

利用 C++ 的 `STL` 库中的 `vector` 建立一个路径栈。
栈中元素类型为为地图节点类型 `PathPoint`
既作为深度优先搜索辅助栈，又存储最终结果路线。

3.迷宫地图（map）

```
int map[MAXSIZE][MAXSIZE];
```

建立二维数组 `map` 存储地图信息，数组中元素下标代表节点位置下标。0 代表通路，1 代笔墙。同时地图最外围一圈有一层外墙。

4.访问标记（mark）

```
int mark[MAXSIZE][MAXSIZE]
```

元素下标对应节点在地图中的坐标。1 代表该元素已经被访问过，0 代表该元素还未被访问。

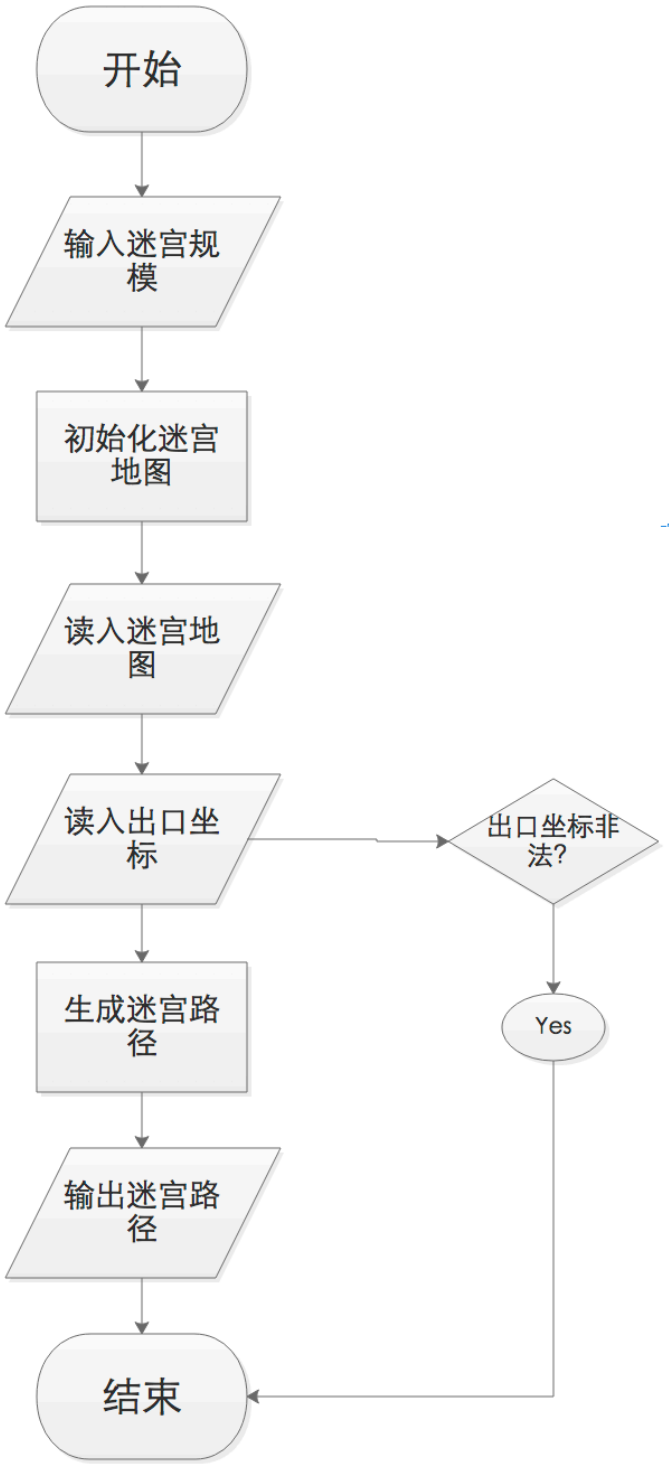
5.功能函数

```
    void initMap(int M,int N);       // 地图初始化函数
    void readMap(int M,int N);       // 地图信息读入函数
    void createPath(PathPoint e);    // 路径搜寻函数
    void print(int M,int N,PathPoint e);         // 路径打印函数
```

定义了一套功能函数，分别完成地图初始化，地图信息读入，地图中路径搜索以及路径打印功能。

三.系统实现

1.系统执行框架



首先要求用户输入迷宫的规模，即迷宫的长和宽。
（注意：这里的规模不包括迷宫最外一圈外墙）。

核心代码如下：

```
int m,n;
cout<<"请输入迷宫行数与列数(不包括外墙):\n";        // 读入迷宫规模
cin>>m>>n;
const int M = m;          // 定义迷宫规模常量
const int N = n;
```

程序执行情况如下：

随后调用 `void initMap(int M,int N)` 初始化迷宫地图，以便于后续输入。

核心代码如下：

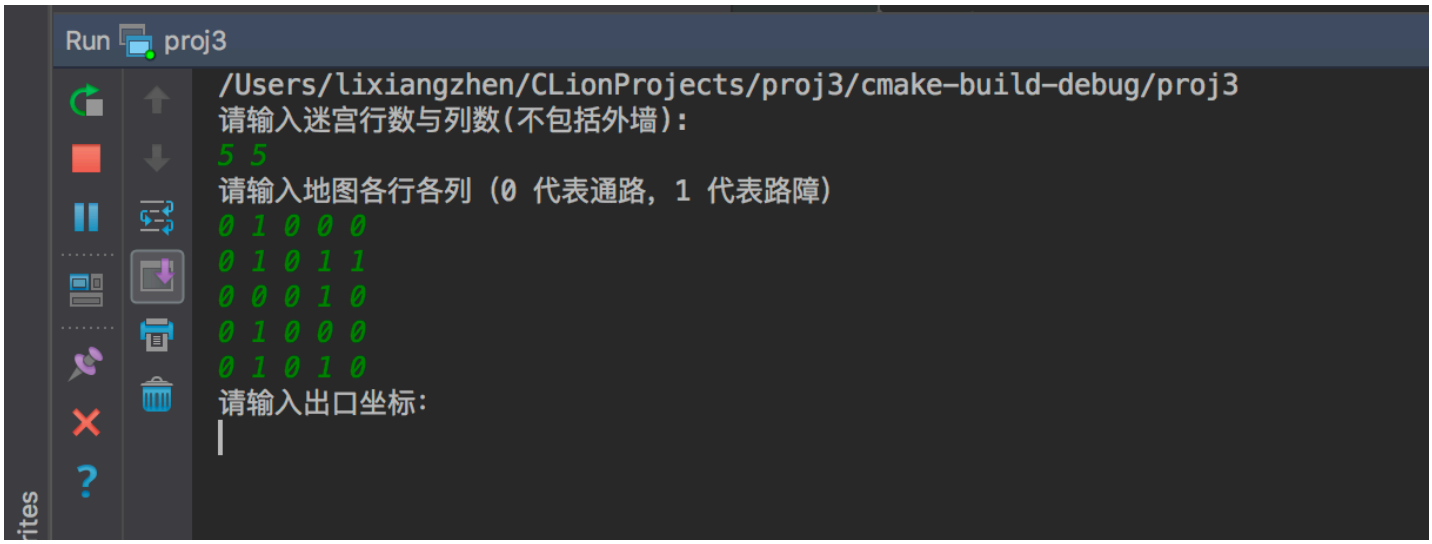
```
initMap(M,N);             // 初始化迷宫地图
```

接着调用 `void readMap(int M,int N)` 读入迷宫地图，地图由用户输入，0 代表通路，1 代表墙。

核心代码如下：

```
    readMap(M,N);            // 读入迷宫地图
```

程序执行情况如下：



然后要求用户输入出口位置。建立出口节点 `PathPoint exit` 。

核心代码如下：

```
    cout<<"请输入出口坐标: \n";
    cin>>m>>n;                      // 读入迷宫出口
    PathPoint exit(m,n);            // 建立出口节点
```

程序执行情况如下：

对输入的出口节点进行检查，排除节点处有墙以及节点处不在迷宫边缘的情况 。

核心代码如下：

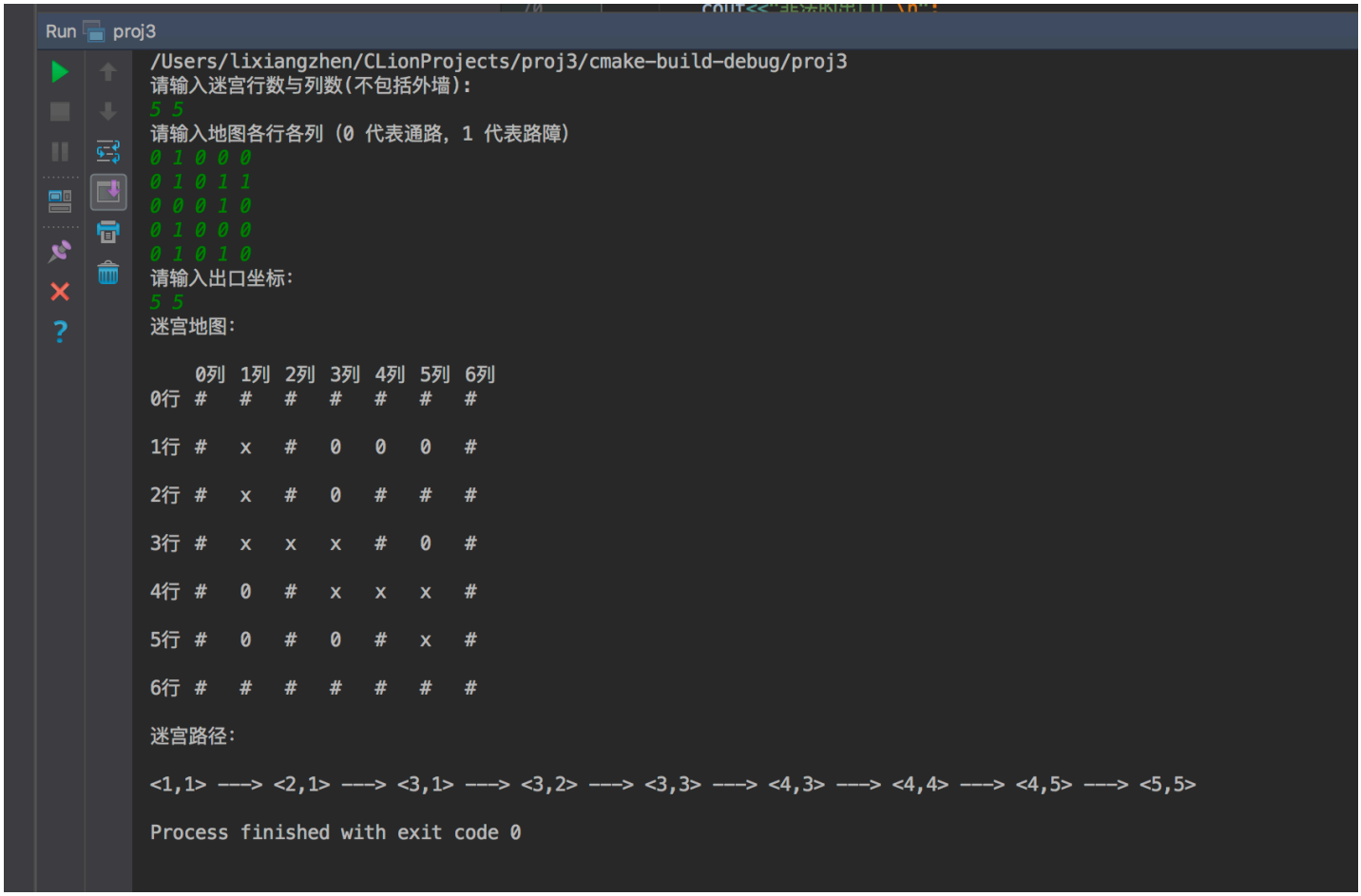
```
    //出口合法性检查
    if(map[exit.x][exit.y] == 1||exit.y != N){       // 如果出口处有墙或出口不在迷宫边缘则非法
        cout<<"非法的出口! \n";
        return 0;
    }
```

随后判断开始节点（1，1）处是否有墙。如果有墙，则报错。反之，调用`void createPath(PathPoint e)`函数在迷宫中搜路径，并调用`void print(int M,int N,PathPoint e)`函数将搜寻结果输出。

核心代码如下：

```
if(map[1][1] == 1)
    cout<<"（1，1）起始位置有墙，路径非法\n";
else{
    createPath(exit);        // 在迷宫中搜索路径

    print(M,N,exit);             // 打印路径
}
```

程序执行情况如下：



2.地图初始化功能

核心代码如下:

```
/*
 * 对地图进行初始化操作
 * 包括地图边缘、标记数组等
 * */
void initMap(int M,int N){
    //初始化map,mark
    memset(map,0,(M + 2)*(N + 2)*sizeof(int));       // 对地图清零
    memset(mark,0,(M + 2)*(N + 2)*sizeof(int));      // 对标记数组清零
    for(int i = 0;i < N + 2;++i)                     // 将地图边界外墙置1
        map[M + 1][i] = map[0][i] = 1;
    for(int i = 0;i < M + 2;++i)
        map[i][0] = map[i][N + 1] = 1;
}
```

先将迷宫地图数组以及访问标记数组清零。
然后将迷宫外围外墙位置置为1.

关联调用情况如下：

```
57      const int M = m;          // 定义迷宫规模常量
58      const int N = n;
59
60      initMap(M,N);             // 初始化迷宫地图
61
```

3.地图信息读入功能

核心代码如下：

```
/*
 * 读入地图信息
 * 0 代表通路，1 代表路障
 * 读入的迷宫地图不包括外墙
 * */
void readMap(int M,int N){
    //输入地图
    cout<<"请输入地图各行各列（0 代表通路，1 代表路障）\n";

    for(int i = 1;i <= M;++i)
        for(int j = 1;j <= N;++j){
            cin>>map[i][j];
        }

}
```

先告知用户地图输入规则
然后顺次向地图中读入信息

关联调用情况如下：

```
    readMap(M,N);              // 读入迷宫地图
    cout<<"请输入出口坐标：\n";
```

4.路径搜索功能

核心代码如下：

```
/*
 * 在迷宫中搜索路径
 * 通过栈来存储沿途节点
 * 采用DFS策略探寻路线
 * */
void createPath(PathPoint e){

    mark[1][1] = 1;      // 从（1，1）位置开始探寻
    PathStack.push_back(PathPoint(1,1));     // 将起始（1，1）位置压入栈中

    while(!PathStack.empty()){       // 栈非空时一直探寻路径
        PathPoint current = PathStack.back();    // 取出栈顶元素，以它为起点探寻路径
        if(current == e)         // 栈顶元素为出口，则找到路径，探寻结束
            break;
        if(mark[current.x][current.y - 1] == 0&&map[current.x][current.y - 1] == 0
){         //先探寻上方路径
            PathStack.push_back(PathPoint(current.x,current.y - 1));
            mark[current.x][current.y - 1] = 1;
        }
        else if(mark[current.x + 1][current.y] == 0&&map[current.x + 1][current.y]
 == 0){   //探寻右侧路径
            PathStack.push_back(PathPoint(current.x + 1,current.y));
            mark[current.x + 1][current.y] = 1;
        }
        else if(mark[current.x][current.y + 1] == 0&&map[current.x][current.y + 1]
 == 0){   //探寻下方路径
            PathStack.push_back(PathPoint(current.x,current.y + 1));
            mark[current.x][current.y + 1] = 1;
        }
        else if(mark[current.x - 1][current.y] == 0&&map[current.x - 1][current.y]
 == 0){   //探寻左侧路径
            PathStack.push_back(PathPoint(current.x + 1,current.y));
            mark[current.x - 1][current.y] = 1;
        }
        else
            PathStack.pop_back();        //四个方向均走不通，则退回前一步
    }

}
```

该函数为程序的核心功能函数，负责在迷宫中搜索一条从起点（1，1）到用户给定

终点的路径。
采用深度优先的搜索策略，不断探寻当前位置的下一个位置。当前位置的四周都走不通时，返回路径上的前一个位置。
利用辅助栈 `PathStack` 保存沿途经过的路径。
当探寻到出口节点或者栈为空则退出循环，搜索结束。

关联调用情况如下：

```
        cout<<" (1, 1) 起始位置有墙，路径非法\n";
    else{
        createPath(exit);        // 在迷宫中搜索路径
```

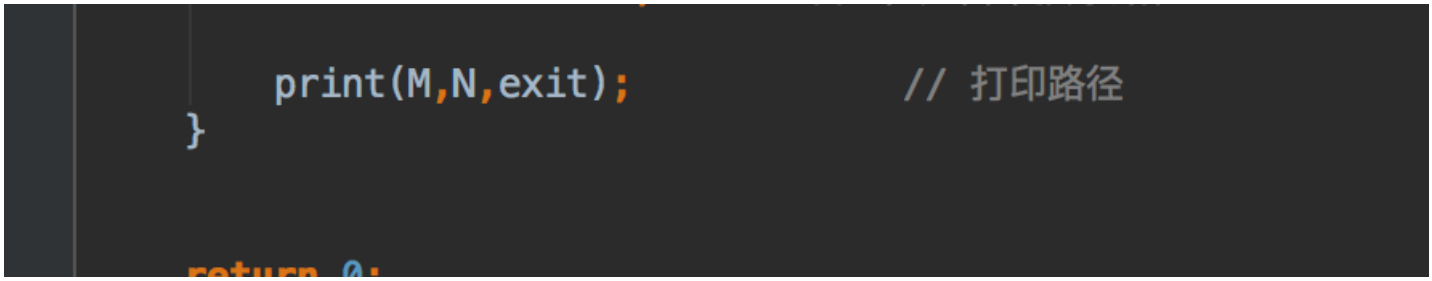
5.输出功能

核心代码如下：

```
/*
 * 格式化输出路线图以及路线
 * 内部创建了辅助数组printmap
 * */
void print(int M,int N,PathPoint e){
    //结果输出

    char printmap[M + 2][N + 2];         // 建立输出辅助数组

    if(PathStack.back() == e){           // 如果找到了出口，则打印路线地图

        for(int i = 0;i < M + 2;++i)         // 先将墙与路写入
            for(int j = 0;j < N + 2;++j){
                if(map[i][j] == 1)
                    printmap[i][j] = '#';
                else
                    printmap[i][j] = '0';
            }
        for(int i = 0;i < PathStack.size();++i){         // 写入搜索到的路径
            printmap[PathStack[i].x][PathStack[i].y] = 'x';
        }
        cout<<"迷宫地图：\n\n";       //格式化输出地图
        cout<<"     ";
        for(int i = 0;i < N + 2;++i)
            cout<<i<<"列 ";
        cout<<endl;
        for(int i = 0;i < M + 2;++i){
            cout<<i<<"行 ";
            for(int j = 0;j < N + 2;++j)
                cout<<printmap[i][j]<<"     ";
            cout<<"\n\n";
        }

        cout<<"迷宫路径：\n\n";       // 格式化输出路径
        for(int i = 0;i < PathStack.size() - 1;++i){
            cout<<PathStack[i]<<" ---> ";
        }
        cout<<e<<endl;
    }
    else
        cout<<"迷宫没有合法路径！\n";     // 如果没有找到出口则提示错误

}
```

该函数输出路径搜索的结果。
首先对结果进行判定，若果没有找到出口，则提示无合法路径。 反之则先输出路线地图，后输出路线。
为了方便地图的输出，建立了辅助数组 `char printmap[M + 2][N + 2]`

关联调用情况如下：



四.测试

1.基本功能测试

测试用例

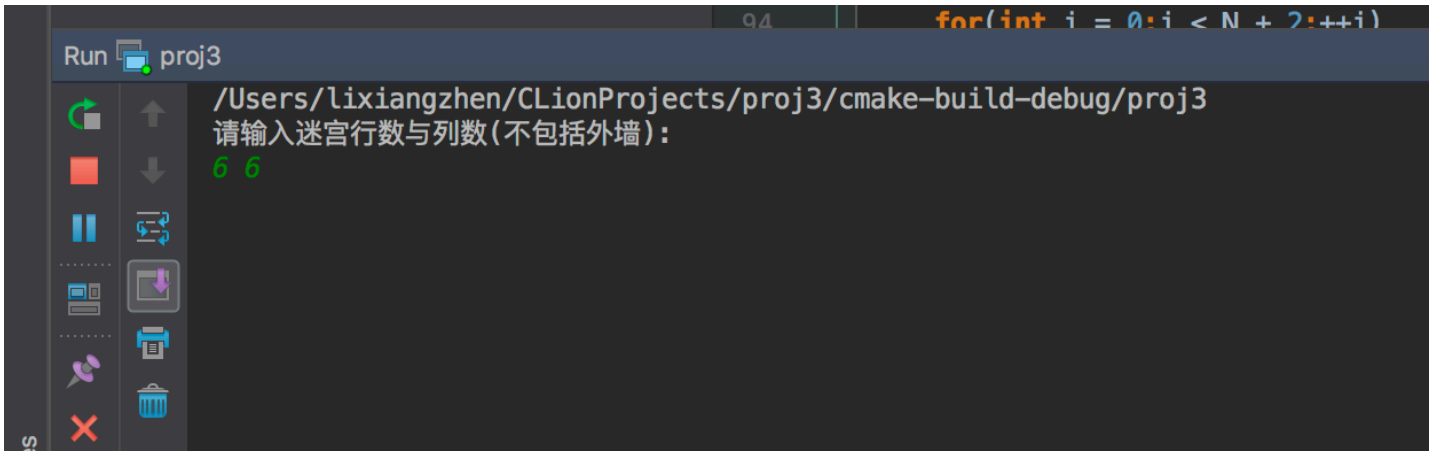
迷宫规模： `6 6`
出口位置： `6 6`

迷宫地图：

```
0 1 1 1 1 1
0 0 0 0 1 1
1 0 1 0 1 1
1 0 0 0 1 1
1 1 1 0 1 1
1 1 1 0 0 0
```

输入迷宫规模

程序执行情况如下：

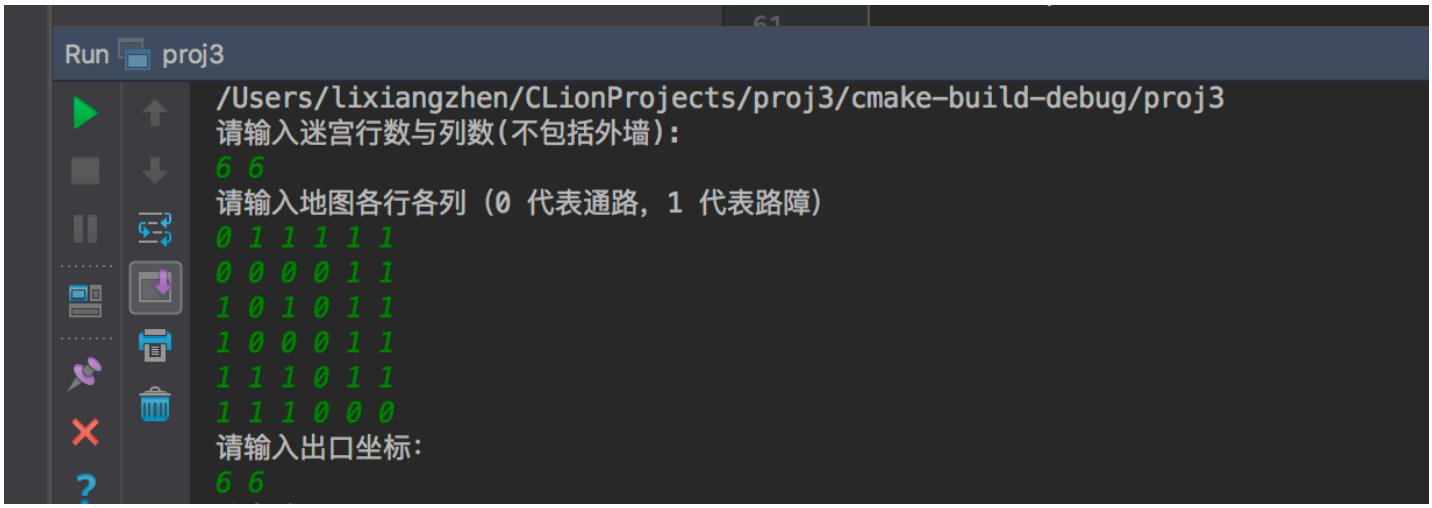


输入迷宫地图

程序执行情况如下：

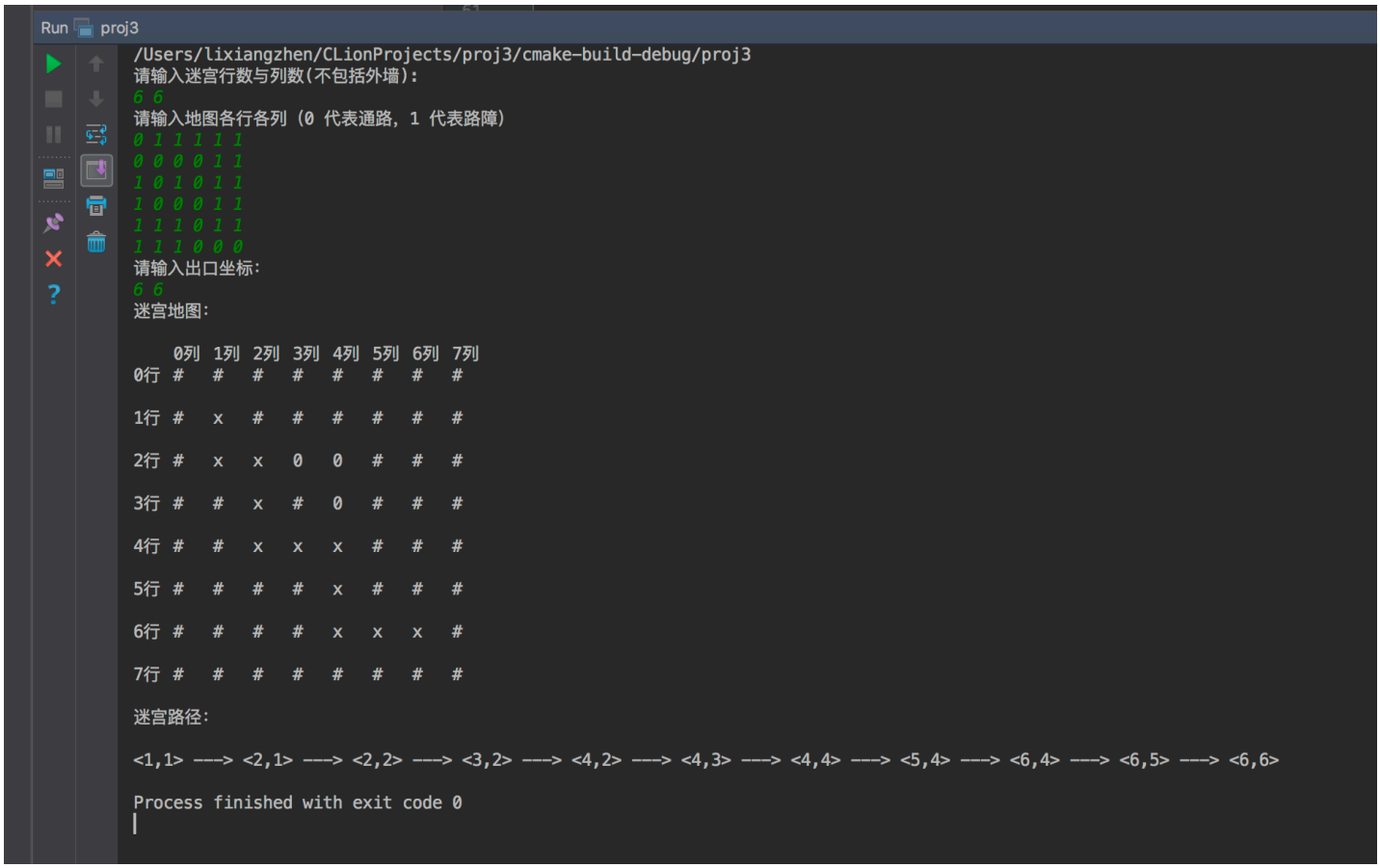
输入出口位置

程序执行情况如下：



路径搜索结果

程序执行情况如下：



2.边界测试

2.1出口非法

测试用例

迷宫规模：`4 4`
出口位置：`4 3`

迷宫地图：

```
0 1 1 0
1 0 1 1
1 0 0 0
1 1 1 0
```

程序执行情况如下：

由于（4，3）位置有墙，出口非法。

2.2入口非法

测试用例

迷宫规模：`4 4`
出口位置：`4 4`

迷宫地图：

```
1 1 1 0
1 0 1 1
1 0 0 0
1 1 1 0
```

程序执行情况如下：

由于（1，1）位置有墙，入口非法。

2.3路径不存在

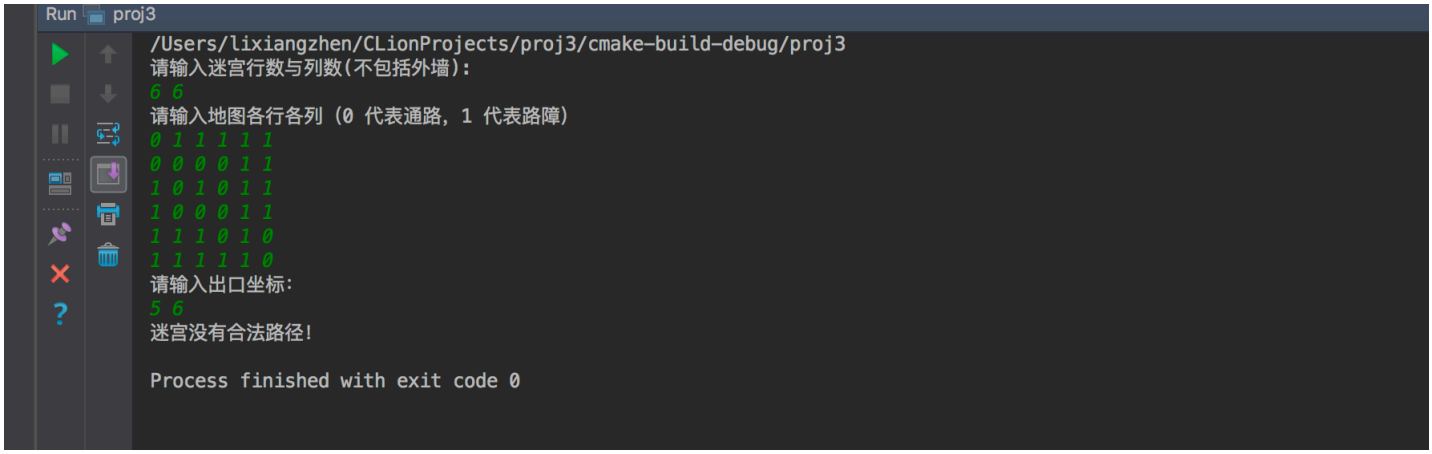
测试用例

迷宫规模：`6 6`
出口位置：`5 6`

迷宫地图：

```
0 1 1 1 1 1
0 0 0 0 1 1
1 0 1 0 1 1
1 0 0 0 1 1
1 1 1 0 1 0
1 1 1 1 1 0
```

程序执行情况如下：



迷宫中不存在从（1，1）到出口（5 6）的贯通路径。