



武汉纺织大学  
WUHAN TEXTILE UNIVERSITY

数学与计算机学院

# 迷宫问题的求解报告

杨鑫

计算机 11902 班

2021 年 1 月 25 日

本作品采用 CC-BY-NC-SA 协议进行许可



# 1 需求分析

## 1.1 问题描述

以一个  $m \times n$  的长方阵表示迷宫，0 和 1 分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

## 1.2 涉及知识点

求迷宫中从入口到出口的所有路径是一个经典的程序设计问题。由于计算机解迷宫时，通常用的是“穷举求解”的方法，即从入口出发，顺某一方向向前探索，若能走通，则继续往前走；否则沿原路径退回，换一个方向再继续探索，直至所有可能的通路都探索到为止。

为了保证在任何位置上都能沿原路退回，显然需要用一个后进先出的结构来保存从入口到当前位置的路径。因此，在求迷宫通路的算法中应用“栈”也就是自然而然的事了。

## 1.3 基本要求

- (1) 首先实现一个以链表作存储结构的栈类型，然后编写一个求解迷宫的非递归程序。求得的通路以三元组  $(i, j, d)$  的形式输出。其中： $(i, j)$  指示迷宫中的一个坐标， $d$  表示走到下一坐标的方向。

如，对于图 1 所示的迷宫，输出一条通路为：

$(1, 1, 0), (1, 2, 1), (2, 2, 1), (3, 2, 2), (3, 1, 1), \dots$

- (2) 编写递归形式的算法，求得迷宫中所有可能的通路。
- (3) 以方阵形式输出迷宫及其通路。

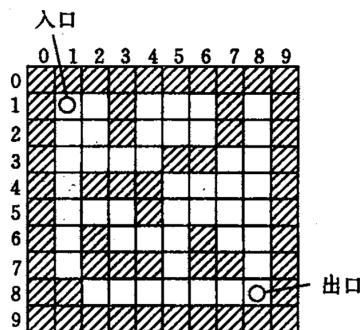


图 1: 迷宫示例

## 2 概要设计

### 2.1 数据结构

#### 2.1.1 坐标位置类

坐标是一个重要的数据结构，它记录了方块的位置信息，是计算机探索路径的关键依据。

Pos
+ i : int
+ j : int
+ Pos()
+ Pos(int i, int j)
+ operator==(const Pos& right) : bool
+ to_String(): string

坐标类数据成员：(i, j) 表示一个坐标，i 和 j 均为 int 类型。

坐标类函数成员：

- Pos(), 构造函数，初始化一个坐标
- operator==( ), 重载 “==” 运算符，用于判断两个位置是否相等
- to\_String(), 将坐标信息转换为字符串

#### 2.1.2 通道块类

假设“当前位置”指的是“在搜索过程中某一时刻所在图中某个方块位置”，那么应该有一个辅助变量指示计算机下次要达到的位置。“下一位置”可以是当前位置的东、南、西、北方向相邻的方块，我们只记录即将转向的方向值就可以了。

于是，将坐标位置和“从此通道块走向下一通道块的方向”封装成一个通道块类。

Block
+ ord : int
+ seat : Pos
+ di : int
+ Block()
+ Block(int ord, Pos seat)
+ setBlock(int ord, Pos seat, int di)
+ to_String() : string
+ operator<<(ostream& out, Block block) : ostream&

通道块类数据成员：

- ord, 通道块在路径上的“序号”
- seat, 通道块在迷宫中的“坐标”
- di, 从此通道块走向下一通道块的方向, 规定其值取 0、1、2、3, 分别代表东、南、西、北

通道块类函数成员：

- Block(), 构造函数, 初始化一个通道块
- setBlock(), 设置通道块
- to\_String(), 将通道块信息转换为字符串
- operator«(), 重载“«”运算符, 用于输出通道块类信息到输出流中

### 2.1.3 迷宫类

将迷宫的行数、列数、迷宫矩阵, 封装成一个迷宫类。此处规定迷宫矩阵中障碍的值为“@”, 通路的值为“.”, 计算机探索后, 经过的路径的值为“\*”。

Maze
- r : int - c : int - map : char[] []
+ Maze() + Maze(int r, int c, char map[] []) + getRow() : int + getCol() : int + setRow() + setCol() + setMapValue(Pos pos, char value) + getMapValue(Pos pos) : char + canPass(Pos pos) : bool

迷宫类数据成员：

- r, 行数
- c, 列数
- map, 迷宫矩阵

迷宫类函数成员：

- Maze(), 构造函数, 初始化迷宫
- getRow(), 获取行数
- getCol(), 获取列数
- setRow(), 设置行数
- setCol(), 设置列数
- setMapValue(), 设置迷宫矩阵某坐标元素值
- getMapValue(), 获取迷宫矩阵某坐标元素值
- canPass(), 某坐标是否通路

#### 2.1.4 栈节点类

栈节点类, 组成栈结构的基本单元, 此处选用链栈结构。

Node<T>
- data: T
- next: Node<T>*
+ Node()
+ Node(T data, Node<T>* next = nullptr)
+ getData() : T
+ setNext(Node<T>* next)
+ getNext() : Node<T>*

栈节点类的数据成员:

- data, 数据域
- next, 指针域, 指向下一节点

栈节点类的函数成员:

- Node(), 构造函数, 初始化一个节点
- getData(), 获取数据域
- setNext(), 设置下一节点
- getNext(), 获取下一节点

### 2.1.5 栈类

栈 (Stack)，是限定仅在表尾进行插入或删除的线性数据结构，具有“后进先出”的特点。这符合迷宫问题求解时需在任何位置上都能沿原路退回的要求。

这里使用栈的链式结构。

Stack<T>
- top: Node<T>*
- size: int
+ Stack()
~ Stack()
+ getSize() : int
+ isEmpty() : bool
+ push(T data)
+ pop()
+ pop(T &e)
+ getTop() : Node<T>&
+ print()

栈类的数据成员：

- top，栈顶，链栈头指针
- size，栈大小 (节点个数)

栈类的函数成员：

- Stack()，构造函数，初始化一个栈
- ~Stack()，析构函数，释放栈
- getSize()，获取栈长
- isEmpty()，判断栈是否为空
- push()，入栈
- pop()，出栈
- getTop()，获取栈顶
- print()，将栈输出到 cout 对象中

## 2.2 程序模块

### 1. 输入模块

从标准输入或文件中输入迷宫矩阵。

### 2. 非递归搜索算法模块

使用栈这种数据结构进行路径的非递归搜索，求出一条路径或判定是否存在通路。

该算法的设计见算法1。

---

#### 算法 1: 非递归搜索

---

**输入:** 入口位置 *start*, 出口位置 *end*

**输出:** 迷宫矩阵的一条通路路径, 或不存在通路

```

1 Stack S;
2 当前位置 curpos  $\leftarrow$  start;
3 do
4   if curpos 可通 then
5     curpos 入栈 (纳入路径);
6     if curpos = end then
7       STOP;
8     else
9       curpos  $\leftarrow$  东邻居方块;
10    end
11  else
12    if S 不空 then
13      if 栈顶位置尚有其他方向未经探索 then
14        curpos  $\leftarrow$  沿顺时针方向旋转找到的栈顶位置的下一相邻方块;
15      end
16      if 栈顶位置四周均不可通 then
17        S 栈顶出栈 (从路径中删去该通道块);
18        if S 不空 then
19          重新测试新的栈顶位置, 直至找到一个可通的相邻块或出栈至空栈;
20        end
21      end
22    end
23  end
24 while S 不空;

```

---

### 3. 递归搜索算法模块

使用栈这种数据结构进行路径的递归搜索，求出所有通路路径。

该算法的设计见算法2。

---

#### 算法 2: 递归搜索

---

输入: 入口位置 *start*, 出口位置 *end*

输出: 迷宫矩阵的所有通路路径

```

1 Stack S;
2 Function MazePath(Pos start, Pos end) is
3   if start = end then
4     end 入栈;
5     Print 当前通路路径;
6     end 出栈;
7   else
8     if start 可通 then
9       while start 相邻方块未全部探索 do
10        start 入栈;
11        Pos next  $\leftarrow$  沿顺时针方向的下一相邻方块;
12        MazePath(next, end);
13        start 出栈;
14      end
15    end
16  end
17 end

```

---

## 3 详细设计

### 3.1 类的函数实现

#### 3.1.1 坐标位置类

坐标位置类的函数实现如代码 1 所示。

```

1 // 重载 "==" 运算符, 用于判断两个位置是否相等
2 bool Pos::operator==(const Pos &right) const {
3     return (i == right.i && j == right.j);
4 }
5 // 构造函数
6 Pos::Pos() : Pos(0, 0) {}
7 Pos::Pos(int i, int j) {
8     this->i = i;
9     this->j = j;

```



```

10 }
11 // 将坐标信息转换为字符串
12 std::string Pos::to_String() {
13     return ( std::to_string(i) + "," + std::to_string(j));
14 }

```

代码 1: Pos 类的实现

### 3.1.2 通道块类

通道块类的函数实现如代码 2 所示。

```

1 // 构造函数
2 Block::Block() = default;
3 Block::Block(int ord, Pos seat, int di) {
4     setBlock(ord, seat, di);
5 }
6 // 设置通道块
7 void Block::setBlock(int ord, Pos seat, int di){
8     this->ord = ord;
9     this->seat = seat;
10    this->di = di;
11 }
12 // 将通道块信息转换为字符串
13 std::string Block::to_String() {
14     return "(" + seat.to_String() + "," + std::to_string(di) + "
15         )";
16 }
17 // 重载 "<<" 运算符, 用于输出通道块类信息到输出流中
18 std::ostream &operator<<(std::ostream &output, Block block) {
19     output << block.to_String();
20     return output;
21 }

```

代码 2: Block 类的实现

### 3.1.3 迷宫类

迷宫类的函数实现如代码 3 所示。

```

1 // 构造函数

```

```
2 Maze::Maze() = default;
3 Maze::Maze(int r, int c, char map[MAXLEN][MAXLEN]) {
4     this->r = r;
5     this->c = c;
6     // 将map拷贝到this->map
7     for (int i = 0; i < r; i++) {
8         for (int j = 0; j < c; j++) {
9             this->map[i][j] = map[i][j];
10        }
11    }
12}
13// 获取行数
14int Maze::getRow() {
15    return r;
16}
17// 获取列数
18int Maze::getCol() {
19    return c;
20}
21// 设置行数
22void Maze::setRow(int row){
23    this->r = row;
24}
25// 设置列数
26void Maze::setCol(int col){
27    this->c = col;
28}
29// 设置迷宫矩阵某坐标元素值
30void Maze::setMapValue(Pos pos, char value) {
31    this->map[pos.i][pos.j] = value;
32}
33// 获取迷宫矩阵某坐标元素值
34char Maze::getMapValue(Pos pos) {
35    return map[pos.i][pos.j];
36}
37// 某坐标是否通路
38bool Maze::canPass(Pos pos) {
39    return getMapValue(pos) == '.';
40}
```

代码 3: Maze 类的实现

### 3.1.4 栈节点类

栈节点类的函数实现如代码 4 所示。

```
1 // 构造函数
2 template<typename T>
3 Node<T>::Node() = default;
4 // 有参构造函数
5 template<typename T>
6 Node<T>::Node(T data, Node<T> *next) {
7     this->data = data;
8     this->next = next;
9 }
10 // 获取数据域
11 template<typename T>
12 T Node<T>::getData() {
13     return data;
14 }
15 // 设置下一节点
16 template<typename T>
17 void Node<T>::setNext(Node<T> *next) {
18     this->next = next;
19 }
20 // 获取下一节点
21 template<typename T>
22 Node<T> *Node<T>::getNext() {
23     return next;
24 }
```

代码 4: Node 类的实现

### 3.1.5 栈类

栈类的函数实现如代码 5 所示。

```
1 // 构造函数
2 template<typename T>
3 Stack<T>::Stack() {
```

```
4     top = nullptr;
5     // 初始时, 没有节点
6     size = 0;
7 }
8 // 析构函数
9 template<typename T>
10 Stack<T>::~~Stack() {
11     while (top) { // 如果栈顶存在
12         Node<T> *p = top;
13         top = top->getNext();
14         // 释放栈顶
15         delete p;
16     }
17 }
18 // 判断栈是否为空
19 template<typename T>
20 bool Stack<T>::isEmpty() {
21     return top == nullptr;
22 }
23 // 获取栈长
24 template<typename T>
25 int Stack<T>::getSize(){
26     return size;
27 }
28 // 入栈
29 template<typename T>
30 void Stack<T>::push(T data) {
31     // 生成新栈顶, 并指向当前栈顶
32     auto newTop = new Node<T>{data, top};
33     // top指向新栈顶
34     top = newTop;
35     // 更新size
36     size++;
37 }
38 // 出栈
39 template<typename T>
40 void Stack<T>::pop() {
41     if (isEmpty()) {
42         std::cerr << "栈空! 无法出栈." << std::endl;
```

```
43         exit(1);
44     }
45     Node<T> *p = top;
46     // top变为原栈顶的下一节点
47     top = top->getNext();
48     // 删除原栈顶
49     delete p;
50     // 更新size
51     size--;
52 }
53 // 出栈并将栈顶节点的data值赋给参数e
54 template<typename T>
55 void Stack<T>::pop(T &e) {
56     if (isEmpty()) {
57         std::cerr << "栈空! 无法出栈." << std::endl;
58         exit(1);
59     }
60     // 将栈顶赋值给e
61     e = (*top).getData();
62     Node<T> *p = top;
63     // top变为原栈顶的下一节点
64     top = top->getNext();
65     // 删除原栈顶
66     delete p;
67     // 更新size
68     size--;
69 }
70 // 获取栈顶的引用
71 template<typename T>
72 Node<T> &Stack<T>::getTop() {
73     if (isEmpty()) {
74         std::cerr << "栈空! 没有栈顶." << std::endl;
75         exit(1);
76     }
77     return *top;
78 }
79 // 打印栈
80 template<typename T>
81 void Stack<T>::print() {
```

```

82     if (isEmpty()) {
83         std::cout << "null";
84     }
85     Node<T> *p = top;
86     auto cnt{1};
87     while (p) {
88         std::string ch = (cnt % 5 ? " " : "\n"); // 一行五个
89         if (cnt == size) // 如果是最后一个
90             ch = "\n";
91         cnt++;
92         std::cout << p->getData() << ch;
93         p = p->getNext();
94     }
95 }

```

代码 5: Stack 类的实现

## 3.2 程序模块

### 3.2.1 输入模块

输入模块的函数代码实现如代码 6 所示。

```

1  /**
2   * @brief 输入迷宫
3   * @param maze 迷宫对象，将输入到此对象中
4   * @param In 流对象，从此流对象输入到 maze 中
5   */
6  void InputMaze(Maze &maze, std::istream &In) {
7      auto row{0}, col{0};
8      In >> row >> col;
9      maze.setRow(row);
10     maze.setCol(col);
11     char value;
12     for (int i = 0; i < maze.getRow(); i++) {
13         for (int j = 0; j < maze.getRow(); j++) {
14             In >> value;
15             maze.setMapValue(Pos{i, j}, value);
16         }
17     }
18 }

```

代码 6: 输入模块

### 3.2.2 非递归搜索算法模块

根据算法 1 的设计, 编写出了相应的 C++ 语言代码, 见代码 7。

```

1 Status MazePath(Maze &maze, Pos start, Pos end) {
2     Stack<Block> S;          // 实例化一个栈对象
3     Pos curpos = start;     // 当前坐标
4     Block e;                // 实例化一个通道块对象
5     int curstep = 1;        // 探索步骤
6
7     do {
8         if (maze.canPass(curpos)) {          // 当前位置可通过
9             maze.setMapValue(curpos, '*');    // 留下足迹
10            e.setBlock(curstep, curpos, 0);    // 设置待入栈的通道块
11
12            S.push(e); // 入栈
13            if (curpos == end) { // 到达终点(出口)
14                S.print();
15                return true;
16            }
17            // 下一位置是当前位置的东邻
18            curpos = NextPos(curpos, 0);
19            // 探索下一步
20            curstep++;
21        } else { // 当前不能通过
22            if (!S.isEmpty()) {
23                S.pop(e); // 出栈
24                while (e.di == 3 && !S.isEmpty()) {
25                    if (maze.getMapValue(e.seat) != '@')
26                        maze.setMapValue(e.seat, '.');
27                    S.pop(e); // 出栈, 回退一步
28                }
29                if (e.di < 3) {
30                    e.di++; // 换下一个方向探索
31                    S.push(e);
32                    // 设定当前位置是该新方向上的相邻块
33                    curpos = NextPos(e.seat, e.di);

```

```

34         }
35     }
36 }
37 } while (!S.isEmpty());
38 return false;
39 }

```

代码 7: 非递归搜索算法模块代码

### 3.2.3 递归搜索算法模块

根据算法 2 的设计, 编写出了相应的 C++ 语言代码, 见代码 8。

```

1  Maze maze;
2  Stack<Block> S;
3  int count = 0; // 路径计数
4
5  void ReMazepath(Pos start, Pos end) {
6      if (start == end) {
7          // 放入终点
8          maze.setMapValue(start, '*');
9          S.push(Block{S.getSize() + 1, start, 1});
10         cout << "找到第" << ++count << "条通路:" << endl;
11         S.print();
12         PrintMaze(maze);
13         // 退出栈, 找下一条路径
14         S.pop();
15         // 恢复原值
16         maze.setMapValue(start, '.');
17     } else {
18         if (maze.canPass(start)) {
19             int di = 0;
20             while (di < 4) {
21                 // 加入当前方块
22                 S.push(Block{S.getSize() + 1, start, di});
23                 // 下一位置
24                 Pos next = NextPos(start, di);
25                 // 避免来回走动
26                 maze.setMapValue(start, '*');
27                 // 递归
28                 ReMazepath(next, end);

```



```

29         // 退出栈，找其他元素
30         S.pop();
31         // 恢复原值
32         maze.setMapValue(start, '.');
33         di++;
34     }
35 }
36 }
37 }

```

代码 8: 递归搜索算法模块代码

### 3.3 其他函数

*NextPos* 函数，用于确定下一相邻方块：

```

1  int dir[][2] = {{0, 1},    // 东
2                  {1, 0},    // 南
3                  {0, -1},   // 西
4                  {-1, 0}};   // 北
5
6  Pos NextPos(Pos curpos, int i) {
7      Pos ret = curpos;
8      // 新坐标
9      ret.i += dir[i][0];
10     ret.j += dir[i][1];
11     return ret;
12 }

```

代码 9: NextPos 函数

*PrintMaze* 函数，打印迷宫路程图：

```

1  void PrintMaze(Maze &maze) {
2      std::cout << "迷宫路程图：\n";
3      for (int i = 0; i < maze.getRow(); i++) {
4          std::cout << "      ";
5          for (int j = 0; j < maze.getCol(); j++) {
6              std::cout << std::setw(2)
7                  << maze.getMapValue(Pos{i, j});
8          }
9          std::cout << std::endl;

```

```

10     }
11     std::cout << std::endl;
12 }

```

代码 10: PrintMaze 函数

*main* 函数、*menu* 函数、重载的 *InputMaze* 函数，用于组织程序结构：

```

1  int main() {
2      while (true) {
3          switch (menu()) {
4              case 1: {
5                  InputMaze();
6                  Pos start, end;
7                  cout << "输入入口位置:";
8                  cin >> start.i >> start.j;
9                  cout << "输入出口位置:";
10                 cin >> end.i >> end.j;
11                 auto flag = MazePath(maze, start, end);
12                 if (flag) {
13                     PrintMaze(maze);
14                 } else {
15                     cout << "*****    此迷宫无法从起点走到终点。
16                         *****\n";
17                 }
18             }
19             break;
20             case 2: {
21                 InputMaze();
22                 Pos start, end;
23                 cout << "输入入口位置:";
24                 cin >> start.i >> start.j;
25                 cout << "输入出口位置:";
26                 cin >> end.i >> end.j;
27                 ReMazepath(start, end);
28             }
29             break;
30             case 0:
31                 cout << "程序结束，谢谢使用！" << endl;
32                 exit(0);
33         }
34     }
35 }

```

```

33     }
34     return 0;
35 }
36
37 int menu() {
38     auto sn{0};
39     cout << endl
40         << "<-----显示菜单-----" << endl
41         << "1. 非递归搜索迷宫通路路径" << endl
42         << "2. 递归搜搜迷宫通路路径" << endl
43         << "0. 结束程序" << endl
44         << "-----显示菜单----->" << endl
45         << "输入0-2:";
46     while (true) {
47         cin >> sn;
48         if (sn < 0 || sn > 3)
49             cout << "输入错误, 重选0-2:" << endl;
50         else
51             break;
52     }
53     return sn;
54 }
55
56 void InputMaze() {
57     auto way{0};
58     cout << "0手动输入,1从文件导入:";
59     while (true) {
60         cin >> way;
61         if (way < 0 || way > 1)
62             cout << "输入错误, 重新输入0或1:" << endl;
63         else
64             break;
65     }
66     if (way) {
67         std::string filename;
68         cout << "输入文件名:";
69         cin >> filename;
70         std::ifstream InputFile;
71         InputFile.open(filename);

```

```

72         if (!InputFile.is_open()) {
73             std::cerr << "没有找到文件" << filename << endl;
74             exit(1);
75         }
76         InputMaze(maze, InputFile);
77         InputFile.close();
78     } else {
79         cout << "输入行数、列数、迷宫矩阵:" << endl;
80         InputMaze(maze, std::cin);
81     }
82 }

```

代码 11: 其他函数

## 4 测试分析

使用 G++ 8.1.0 编译器编译本实例，运行测试。

测试文件为 test.in，其内容见图 2：

```

10 10
@ @ @ @ @ @ @ @ @ @
@ . . @ . . . @ . @
@ . . @ . . . @ . @
@ . . . . @ @ . . @
@ . @ @ @ . . . . @
@ . . . @ . . . . @
@ . @ . . . @ . . @
@ . @ @ @ @ @ @ . @
@ @ . . . . . @ . @
@ @ @ @ @ @ @ @ @ @

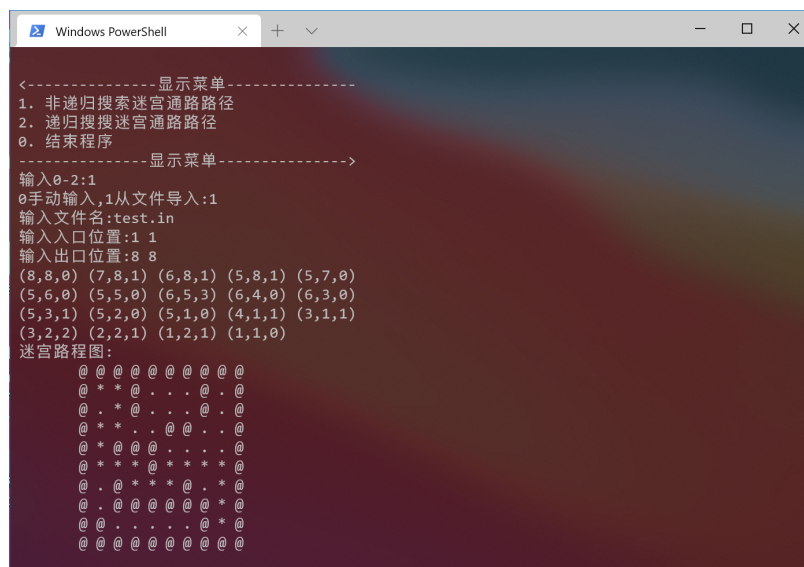
```

图 2: 测试迷宫

### 4.1 非递归搜索

#### 4.1.1 搜索成功

入口：(1,1)，出口 (8,8)，结果见图3。



```

Windows PowerShell

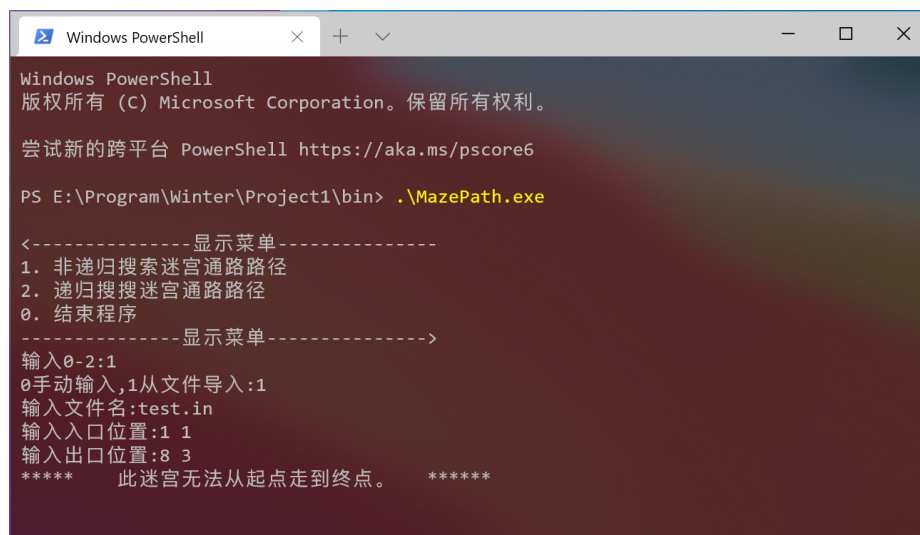
<-----显示菜单----->
1. 非递归搜索迷宫通路路径
2. 递归搜索迷宫通路路径
0. 结束程序
-----显示菜单----->
输入0-2:1
0手动输入,1从文件导入:1
输入文件名:test.in
输入入口位置:1 1
输入出口位置:8 8
(8,8,0) (7,8,1) (6,8,1) (5,8,1) (5,7,0)
(5,6,0) (5,5,0) (6,5,3) (6,4,0) (6,3,0)
(5,3,1) (5,2,0) (5,1,0) (4,1,1) (3,1,1)
(3,2,2) (2,2,1) (1,2,1) (1,1,0)
迷宫路程图:
  @ @ @ @ @ @ @ @ @
  @ * @ . . . @ . @
  @ . * @ . . . @ . @
  @ * * . @ @ . . @
  @ * @ @ @ . . . @
  @ * * * * * * * @
  @ . @ * * * @ . * @
  @ . @ @ @ @ @ * @
  @ @ . . . . @ * @
  @ @ @ @ @ @ @ @ @

```

图 3: 非递归搜索 - 搜索成功

### 4.1.2 搜索失败

入口: (1,1), 出口 (8,3), 结果见图4。



```

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS E:\Program\Winter\Project1\bin> .\MazePath.exe

<-----显示菜单----->
1. 非递归搜索迷宫通路路径
2. 递归搜索迷宫通路路径
0. 结束程序
-----显示菜单----->
输入0-2:1
0手动输入,1从文件导入:1
输入文件名:test.in
输入入口位置:1 1
输入出口位置:8 3
***** 此迷宫无法从起点走到终点。 *****

```

图 4: 非递归搜索 - 搜索失败

## 5 递归搜索

入口: (1,1), 出口 (6,4), 共有 4 条通路路径, 结果见图 5 和图 6。

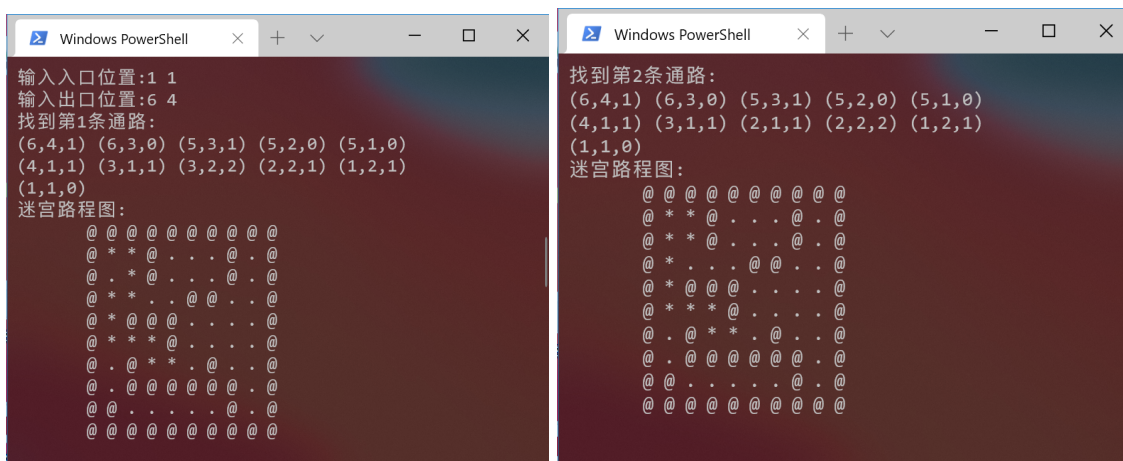


图 5: 递归搜索 - 第一条和第二条通路路径

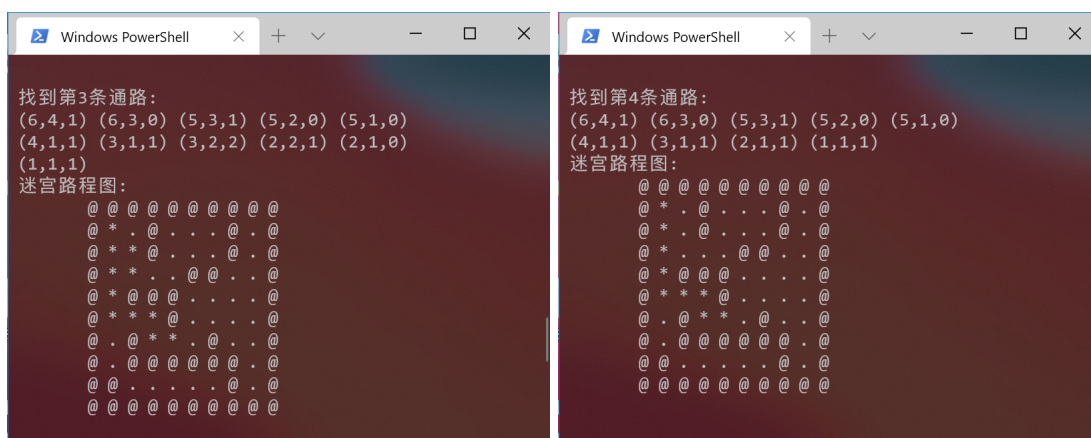


图 6: 递归搜索 - 第三条和第四条通路路径

## 6 总结

通过本次项目实践，熟练地掌握栈了这种数据结构，并实现了链栈的编写。

使用计算机进行迷宫通路路径的探索，使得栈这种数据结构得到实际的应用。在实践过程中，使用了非递归搜索算法，对计算机如何模拟实际问题有了较好的感性理解。在使用递归搜索算法中，感受到了递归所带来的便利之处，能够将复杂的问题简单化，并且也易于人理解。

## 7 附录

本项目实例使用 CMake 构建，并要求编译器为 G++ 8.1.0，使用的操作系统是 Windows。

项目主要文件清单:

```

/.....项目根目录
├── bin.....输出文件夹
│   └── MazePath.exe .....已经编译的可执行文件

```

└─ test.in.....	测试输入文件
└─ CMakeLists.txt.....	CMake 项目配置文件
└─ docs.....	项目文档目录
└─ images.....	文档使用的图片资源
└─ project1.pdf.....	项目文档
└─ project1.tex.....	项目文档 L <sup>A</sup> T <sub>E</sub> X 源文件
└─ src.....	源代码
└─ includes.....	头文件包含目录
└─ Stack.h.....	Stack 类头文件
└─ main.cpp.....	主程序
└─ Maze.cpp.....	Maze 类的实现
└─ Maze.h.....	Maze 类的声明

这些源文件可以在 <https://github.com/DianDengJun/Course-Design/tree/main/Winter/Project1> 中查看。

构建本实例的命令 (Bash 或 Powershell) 如下:

进入根目录, 执行:

```
mkdir build
cd build
cmake -G "MinGW Makefiles" ..
make # 或者是 mingw32-make
```

然后进入 bin 目录运行 MazePath.exe。

```
cd ../bin
./MazePath
```