哈尔滨工业大学(深圳)

《数据结构》实验报告

实验二

深度优先和广度优先搜索

学　院: 计算机科学与技术

姓　名: 李秋阳

学　号: 180110527

专　业: 计算机科学与技术

日　期: 2019-04-24

一、问题分析

1.1 迷宫计数

实验原题：输入一个任意大小的迷宫，用深度优先遍历(用栈的方法实现)的方法求出从起点开始能走到的格数(不包含起点)，并输出所有能到达的点。

问题分析：要通过栈的方式实现深度优先遍历，可以建立栈来存储迷宫的已访问点，在判断结点是否符合要求后，通过出栈、入栈等操作以深度优先遍历的方法遍历迷宫可达点并输出。

1.2 迷宫问题求解

实验原题：输入一个任意大小的迷宫，用广度优先遍历(用队列的方法实现)的方法求出走出迷宫的 最短路径长度(步数)，并将路径输出。

问题分析：要通过队列的方式实现广度优先遍历，可以建立队列来存储迷宫的已访问点，在判断结点是否符合要求后，通过入队列、出队列等操作以广度优先遍历的方式找到将迷宫的最短路径并存储在队列中，然后输出。

二、详细设计

2.1 设计思想

2.1.1 迷宫计数

要通过栈的方式实现深度优先遍历，可以（1）建立二维字符数组来存储迷宫，找出合法的起点位置；（2）将合法起点入栈；（3）取栈顶元素，按设定顺序求邻接的未被访问的可通过结点。如果有，将其入栈并记录为已经访问。如果没有则通过将栈顶元素出栈回溯上一结点 ；（4）重复步骤（3）直到栈空，将标记的所有结点数目输出并在迷宫上标记。

2.1.2 迷宫问题求解

要通过队列的方式实现广度优先遍历，可以（1）建立二维字符数组来存储迷宫，找出合法的起点位置；（2）从起点元素开始，判断它的邻接元素结点是否可通过，如果可以就入队列；（3）取队首元素并出队列，寻找可通过邻接元素结点入队列并标记其前驱节点为队首元素；（4）重复步骤（3）直到队列为空或者找到终点。然后从终点开始根据结点的前驱结点输出一条最短可行路径。

2.2 存储结构及操作

(1) 存储结构

一、迷宫计数

```
char MAZE[maxn][maxn];
int vis[maxn][maxn];
int dir[4][2] = {{0,1},{1,0},{0,-1},{-1,0}};
int n,m;
```

栈

```
typedef struct snode {
    /* 栈中存储的节点 */
    int x; // 行坐标
    int y; // 列坐标
    int flag; // 当前方向
    struct snode *prev; // 栈中上一个节点的指针
    /* 可自由添加需要用的变量 */

} StackNode;

typedef struct {
    /* 栈 */
    StackNode *top; // 栈顶指针
    int size; // 栈中节点个数
    /* 可自由添加需要用的变量 */

} Stack;
```

二、迷宫问题求解

```
char MAZE[maxn][maxn];
int vis[maxn][maxn];
int dir[4][2] = {{0,1},{1,0},{0,-1},{-1,0}};
int n,m;
int Judge;
```

队列

```
typedef struct qnode {
    /* 队列中存储的节点 */
    int x; // 行坐标
    int y; // 列坐标
    int step; // 当前点与起点的距离
    struct qnode *pre;//前一个节点
    struct qnode *pab; //队列中下一个节点的指针
    /* 可自由添加需要用的变量 */

} QueueNode;

typedef struct {
    /* 队列 */
    QueueNode *front; //队首指针
    QueueNode *back; //队尾指针
    int size; // 队列中节点个数
    /* 可自由添加需要用的变量 */

} Queue;
```

(2) 涉及的操作

主函数 int main()

一、迷宫计数

① 判断位置合法性 int LegalPosition(int x, int y)
② 初始化栈 Stack *InitStack()
③ 判断栈是否为空 int isEmpty_S(Stack *sta)
④ 将一个节点压入栈中 void Push(Stack *sta, StackNode now)
⑤ 将一个节点弹出栈 StackNode *Pop(Stack *sta)
⑥ 深度优先遍历 void DFS(char MAZE[maxn][maxn], int n, int m)
⑦ 检查访问点数量 int Check_Visit()

二、迷宫问题求解

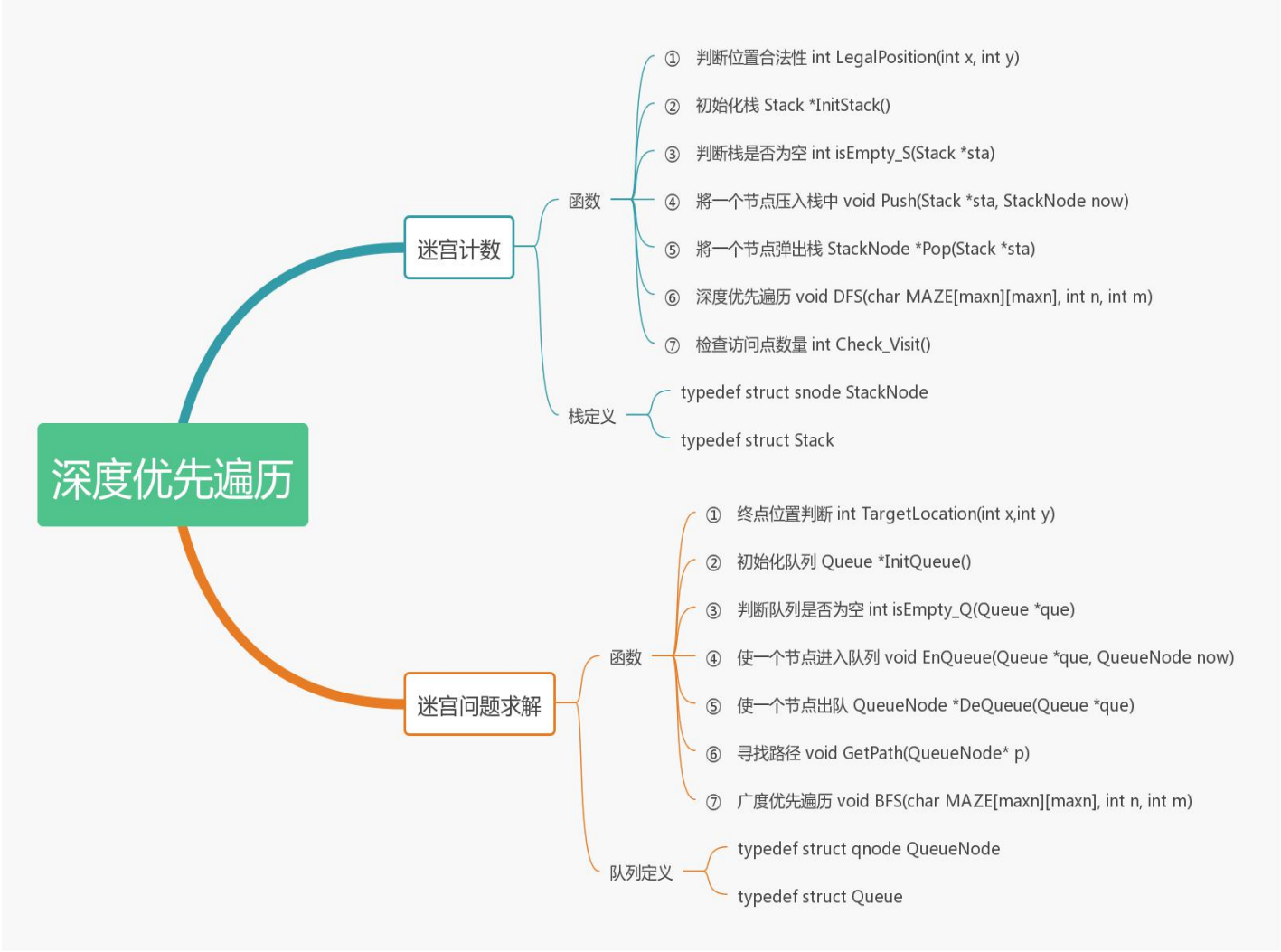
① 终点位置判断 int TargetLocation(int x,int y)
② 初始化队列 Queue *InitQueue()
③ 判断队列是否为空 int isEmpty_Q(Queue *que)
④ 使一个节点进入队列 void EnQueue(Queue *que, QueueNode now)

⑤ 使一个节点出队 QueueNode *DeQueue(Queue *que)

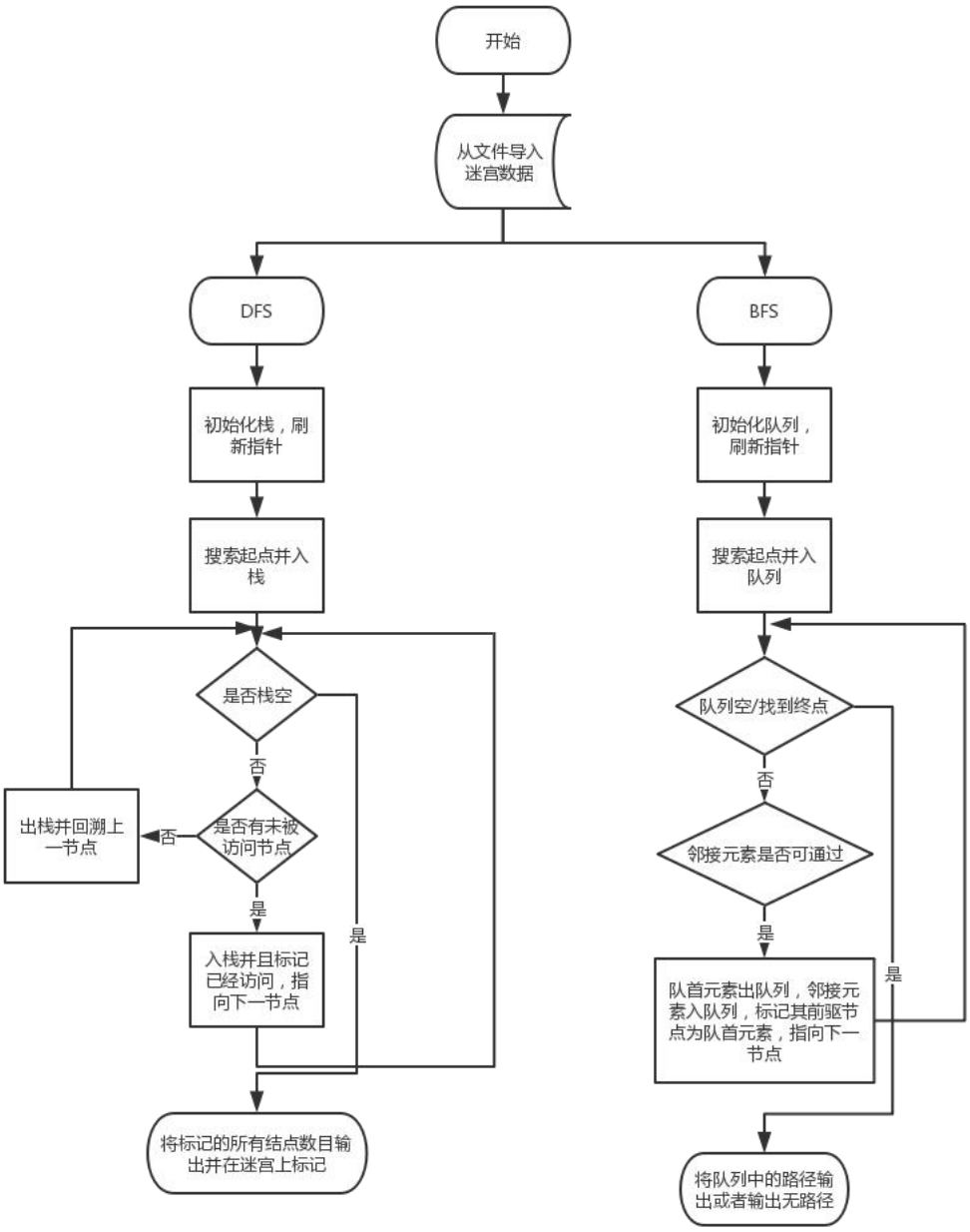
⑥ 寻找路径 void GetPath(QueueNode* p)

⑦ 广度优先遍历 void BFS(char MAZE[maxn][maxn], int n, int m)

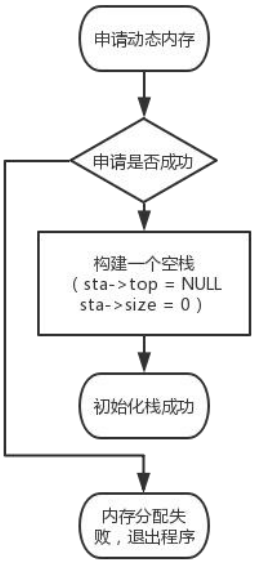
2.3 程序整体流程



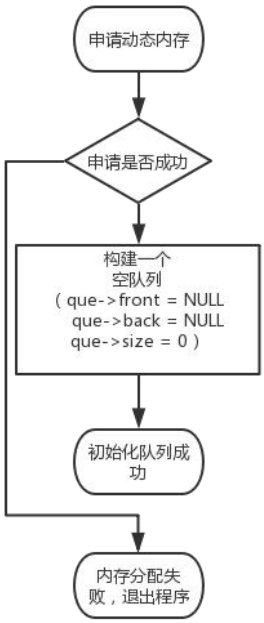
核心算法流程


开始
从文件导入迷宫数据
DFS
初始化栈，刷新指针
搜索起点并入栈
是否栈空
否
是否有未被访问节点
否
出栈并回溯上一节点
是
入栈并且标记已经访问，指向下一节点
是
将标记的所有结点数目输出并在迷宫上标记
BFS
初始化队列，刷新指针
搜索起点并入队列
队列空/找到终点
否
邻接元素是否可通过
是
队首元素出队列，邻接元素入队列，标记其前驱节点为队首元素，指向下一节点
是
将队列中的路径输出或者输出无路径

存储结构初始化

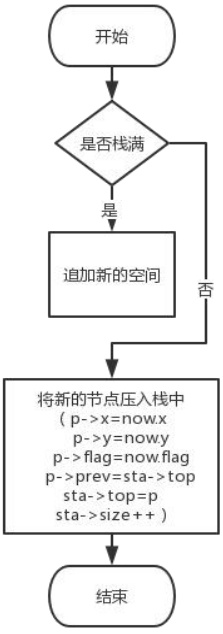


〔 初始化栈 〕

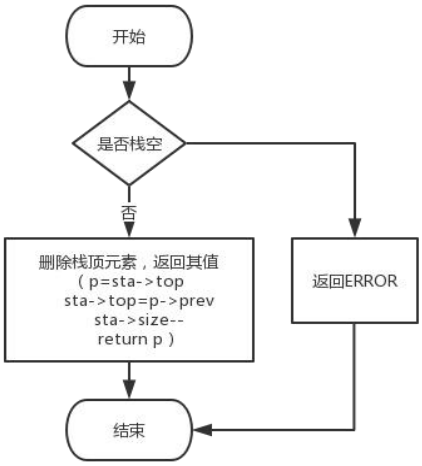


〔 初始化队列 〕

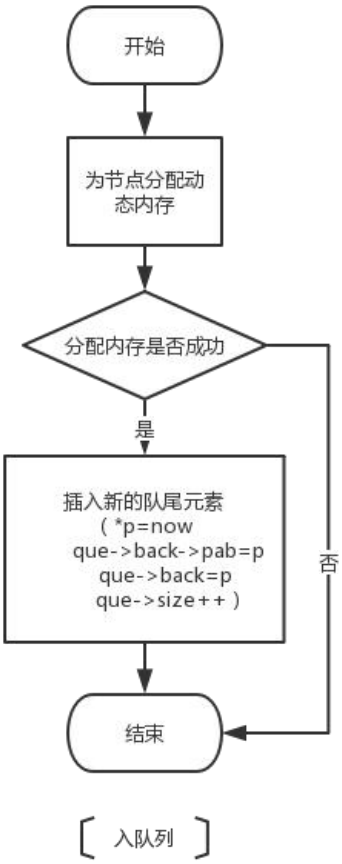
基本操作



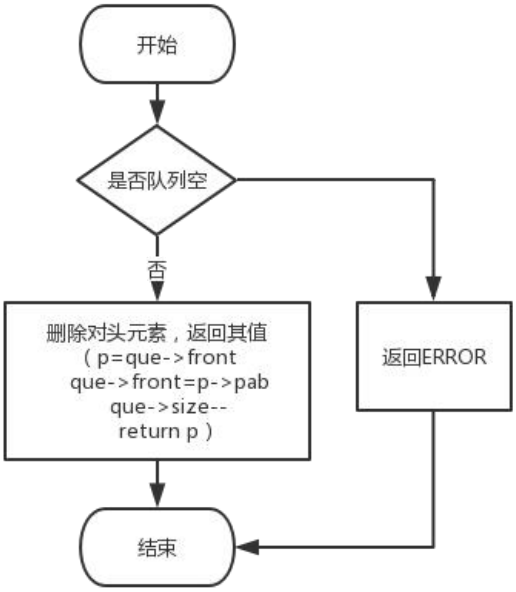
〔 入栈 〕



〔 出栈 〕


开始
为节点分配动态内存
分配内存是否成功
是
否
插入新的队尾元素
(*p=now
que->back->pab=p
que->back=p
que->size++)
结束

入队列


开始
是否队列空
否
删除对头元素，返回其值
(p=que->front
que->front=p->pab
que->size--
return p)
返回ERROR
结束

出队列

三、用户手册

用户通过改变“maze.in”文件内容即可将迷宫内容替换为自定义迷宫

格式如下：

Row（行数） Column（列数）

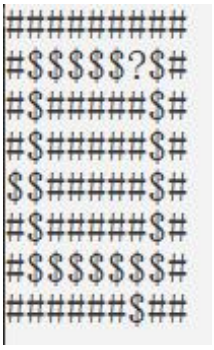
#代表障碍，0 代表空地，?代表起点，走出迷宫视为抵达终点

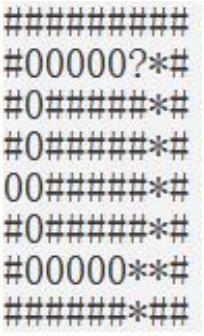
输出的可达点位置用“$”表示，最短路径用“*”覆盖

范例：

8 9

#########
#00000?0#
#0#####0#
#0#####0#
00#####0#
#0#####0#
#0000000#
######0##



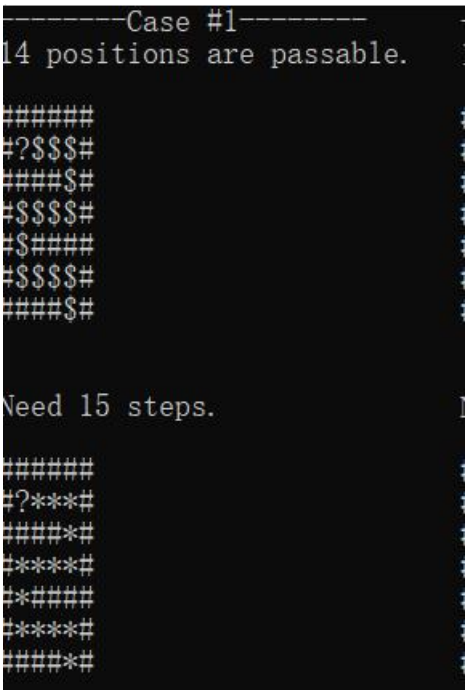


四、结果

迷宫一：

7 6

######
#?000#
####0#
#0000#
#0####
#0000#
####0#


--------Case #1--------
14 positions are passable.
######
#?$$$#
####$#
#$$$$#
#$####
#$$$$#
####$#
Need 15 steps.
######
#?***#
####*#
#****#
#*####
#****#
####*#

迷宫二：

5 5

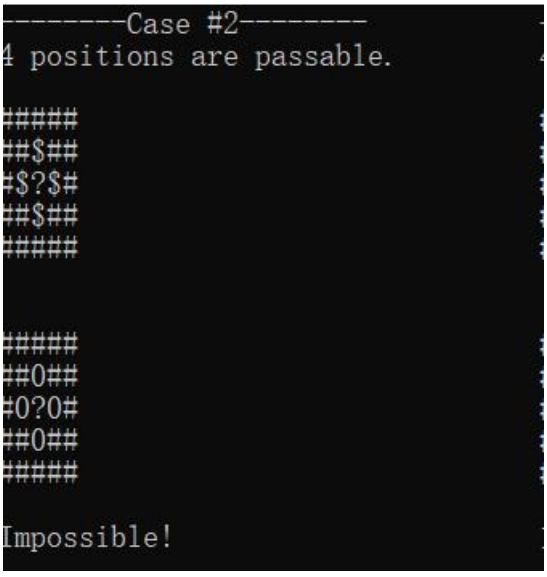
#####

##0##

#0?0#

##0##

#####


--------Case #2--------
4 positions are passable.
#####
##$##
#$?$#
##$##
#####
#####
##0##
#0?0#
##0##
#####
Impossible!

迷宫三：

5 5

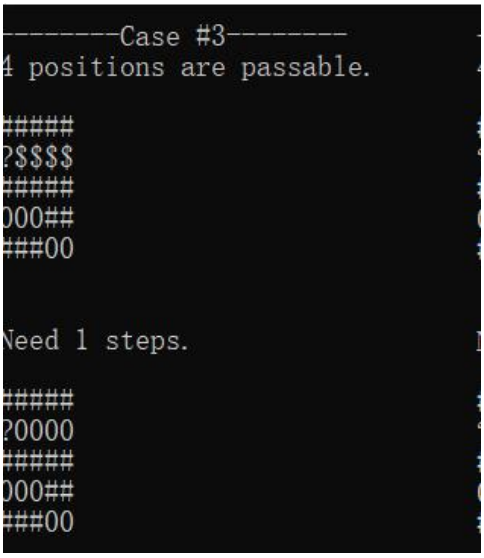
#####

?0000

#####

000##

###00



迷宫四：

7 10

##########

#?0#000#0#

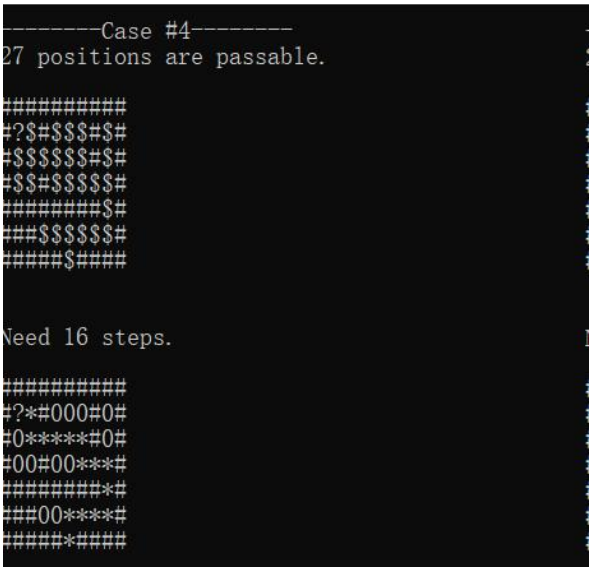
#000000#0#

#00#00000#

########0#

###000000#

#####0####



迷宫五：

8 9

#########

#00000?0#

#0#####0#

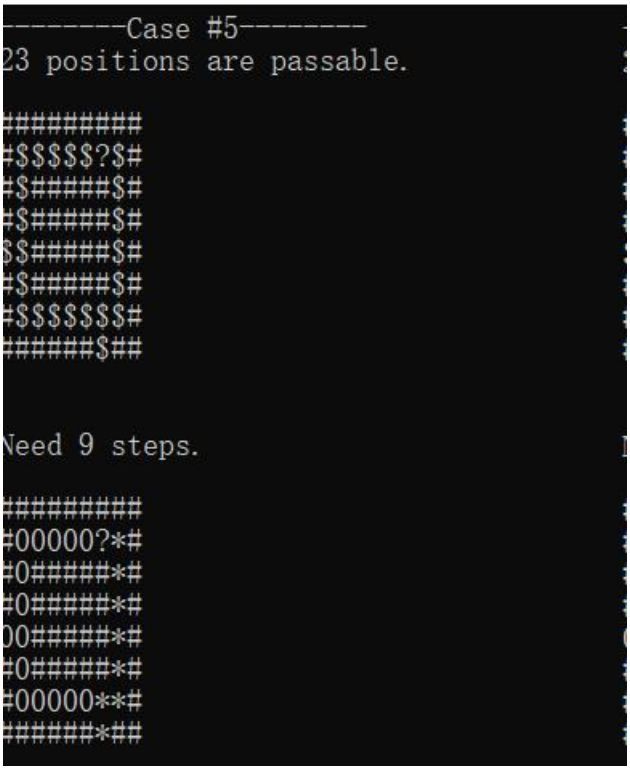
#0#####0#

00#####0#

#0#####0#

#0000000#

######0##



迷宫六：

10 10

#0#######0

##0####0##

#0###000##

##0##00#0#

##00##0#0#

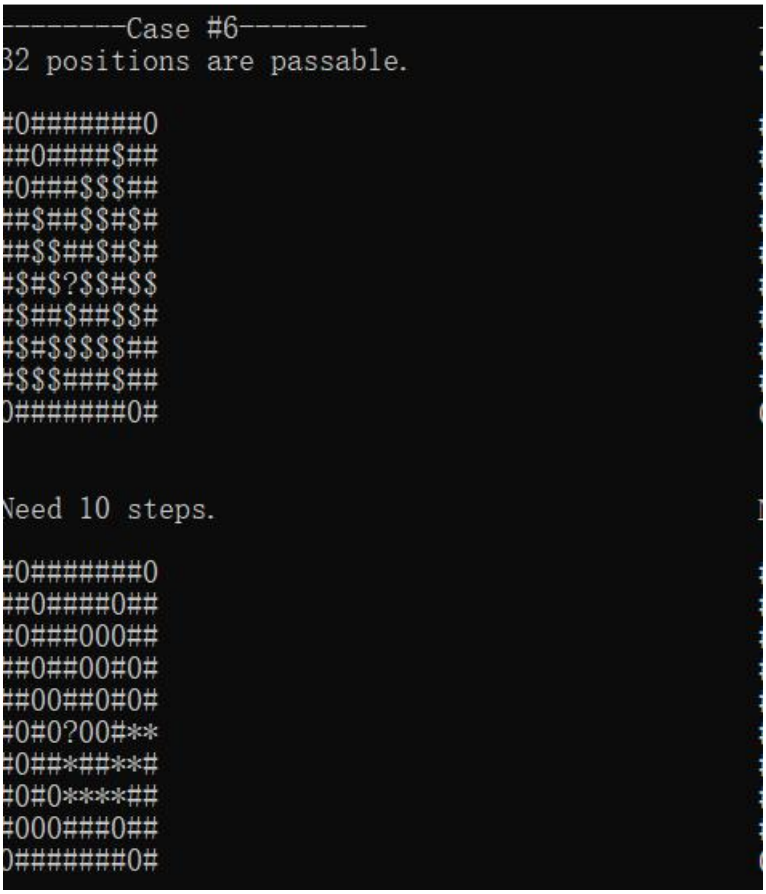
#0#0?00#00

#0##0##00#

#0#00000##

#000###0##

0#######0#



五、总结

实验一、二分别运用了栈和队列两种存储结构，在此基础上又分别实现了DFS（深度优先搜索）和 BFS（广度优先搜索）两种算法来实现对迷宫的遍历和路径的寻找。解决迷宫问题如果使用递归方法的话完全可以不用到栈和队列，相对比较简单。但如果使用非递归的方法，实现起来就要难上许多，不是简简单单懂得算法原理就可以解决的。在实现过程中，对进行方向选择的数组和通过对指针的使用实现回溯一直让我很困扰，一个判断条件没有设置清楚都会让指针的使用完全错乱。经过反复调试后终于将已经的算法实现为计算机语言，不仅帮助我更好地理解了两种存储结构和两种算法，更提高了编程实践能力。