**南京航空航天大学**

**数据结构课程设计报告**

|  |  |
| --- | --- |
| 学生姓名 |  |
| 学 号 |  |
| 学 院 | 计算机科学与技术学院 |
| 专 业 |  |
| 班 级 |  |
| 指导教师 |  |

二〇一八年一月

**目 录**

[一、 XXXXXXXX 4](#_Toc16524)

[1.1 题目简介 4](#_Toc24720)

[1.2数据结构 4](#_Toc28370)

[1.3 算设法计思想 5](#_Toc9562)

[1.4测试数据和结果 7](#_Toc31152)

[1.5算法时间复杂度 9](#_Toc32400)

[1.6源代码 10](#_Toc27957)

[二、 XXXXXXXX 24](#_Toc24745)

[2.1 题目简介 24](#_Toc25235)

[2.2 数据结构 25](#_Toc4239)

[2.3 算法设计思想 25](#_Toc28770)

[2.4测试数据和结果 25](#_Toc30801)

[2.5算法时间复杂度 27](#_Toc4610)

[2.6源代码 27](#_Toc31079)

[三、 XXXXXXXX 33](#_Toc2816)

[3.1 题目简介 33](#_Toc3912)

[3.2 数据结构 33](#_Toc19359)

[3.3 算法设计思想 33](#_Toc25064)

[3.4测试数据和结果 34](#_Toc31772)

[3.5算法时间复杂度 35](#_Toc15368)

[3.6源代码 36](#_Toc8095)

[四、 XXXXXXXX 49](#_Toc24057)

[4.1 题目简介 49](#_Toc11772)

[4.2 数据结构 49](#_Toc3759)

[4.3 算法设计思想 50](#_Toc2661)

[4.4测试数据和结果 50](#_Toc15911)

[4.56算法时间复杂度 51](#_Toc4576)

[4.6源代码 51](#_Toc24227)

[五、XXXXXXXX 58](#_Toc8639)

[5.1 题目简介 58](#_Toc6319)

[5.2 数据结构 58](#_Toc4286)

[5.3 算法设计思想 58](#_Toc29013)

[5.4测试数据和结果 59](#_Toc8829)

[5.5算法时间复杂度 60](#_Toc30701)

[5.6源代码 60](#_Toc22481)

[六、 XXXXXXXX 68](#_Toc1727)

[6.1 题目简介 68](#_Toc150)

[6.2 数据结构 68](#_Toc14755)

[6.3 算法设计思想 68](#_Toc23480)

[6.4测试数据和结果 70](#_Toc25757)

[6.5算法时间复杂度 70](#_Toc7219)

[6.6源代码 71](#_Toc24469)

[七、XXXXXXXX 85](#_Toc29741)

[7.1 题目简介 85](#_Toc664)

[7.2 数据结构 85](#_Toc23174)

[7.3 算法设计思想 85](#_Toc8852)

[7.4测试数据和结果 86](#_Toc5111)

[7.5算法时间复杂度 86](#_Toc220)

[7.6源代码 87](#_Toc24053)

[八、 XXXXXXXX 97](#_Toc26239)

[8.1 题目简介 97](#_Toc3026)

[8.2 数据结构 97](#_Toc13925)

[8.3 算法设计思想 97](#_Toc6513)

[8.4测试数据和结果 98](#_Toc24690)

[8.5算法时间复杂度 109](#_Toc5814)

[8.6源代码 109](#_Toc32518)

[九、 XXXXXXXX 115](#_Toc24048)

[9.1 题目简介 115](#_Toc3603)

[9.2 数据结构 115](#_Toc23185)

[9.3 算法设计思想 115](#_Toc1243)

[9.4测试数据和结果 116](#_Toc14056)

[9.5算法时间复杂度 117](#_Toc26546)

[9.6源代码 117](#_Toc13172)

十、 [课程设计总结 1](#_Toc24048)18

[10.1 完成情况（代码行数） 1](#_Toc3603)18

[10.2 心得体会 1](#_Toc6584)18

# 九、迷宫问题

9.1 题目简介

11、迷宫问题 (选做)（栈，深度搜索，广度搜索）

[问题描述]

利用栈操作实现迷宫问题求解。

[基本要求]

（1）随机生成模拟迷宫地图，不少于10行10列，存在文件中。

（2）动态显示每一步的结果 。

（3）可在此基础上有改进方法。

9.2 数据结构

使用栈操作，与深度搜索相结合

typedef int AdjMatrix[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

typedef struct {

AdjMatrix arcs; //邻接矩阵

int rows; //行

int columns; //列

}Labyrinth; //定义迷宫

void Show(Labyrinth L);

typedef struct{

int x;

int y;

}location;

typedef struct{

int ord; //路径标号

location seat; //坐标位置

int di; //走向下一块方向

}SELemType;

typedef struct{

SELemType \*base;

SELemType \*top;

int stacksize;

}SqStack;

9.3 算法设计思想

本题有2种方法解决，栈与递归

栈：

将迷宫转化为一个int型矩阵，若有障碍则设为1，没有障碍设为0

在搜索过程中的某一时刻所在图中某个方块位置”，则求迷宫中一条路径的算法的基本思想是：

若当前位置“可通”，则纳入“当前路径”，并继续朝“下一位置”探索，即切换“下一位置”为“当前位置”，如此重复直至到达出口；

若当前位置“不可通”，则应顺着“来向”退回到“前一通道块”，然后朝着除“来向”之外的其他方向继续探索；

若该通道块的四周4个方块均“不可通”，则应从“当前路径”上删除该通道块。

所谓“下一位置”指的是当前位置四周4个方向（东、南、西、北）上相邻的方块。假设以栈S记录“当前路径”，则栈顶中存放的是“当前路径上最后一个通道块”。

由此，“纳入路径”的操作即为“当前位置入栈”；“从当前路径上删除前一通道块”的操作即为“出栈”。

递归：

将迷宫转化为一个int型矩阵，若有障碍则设为1，没有障碍设为0

在搜索过程中的某一时刻所在图中某个方块位置”，则求迷宫中一条路径的算法的基本思想是：

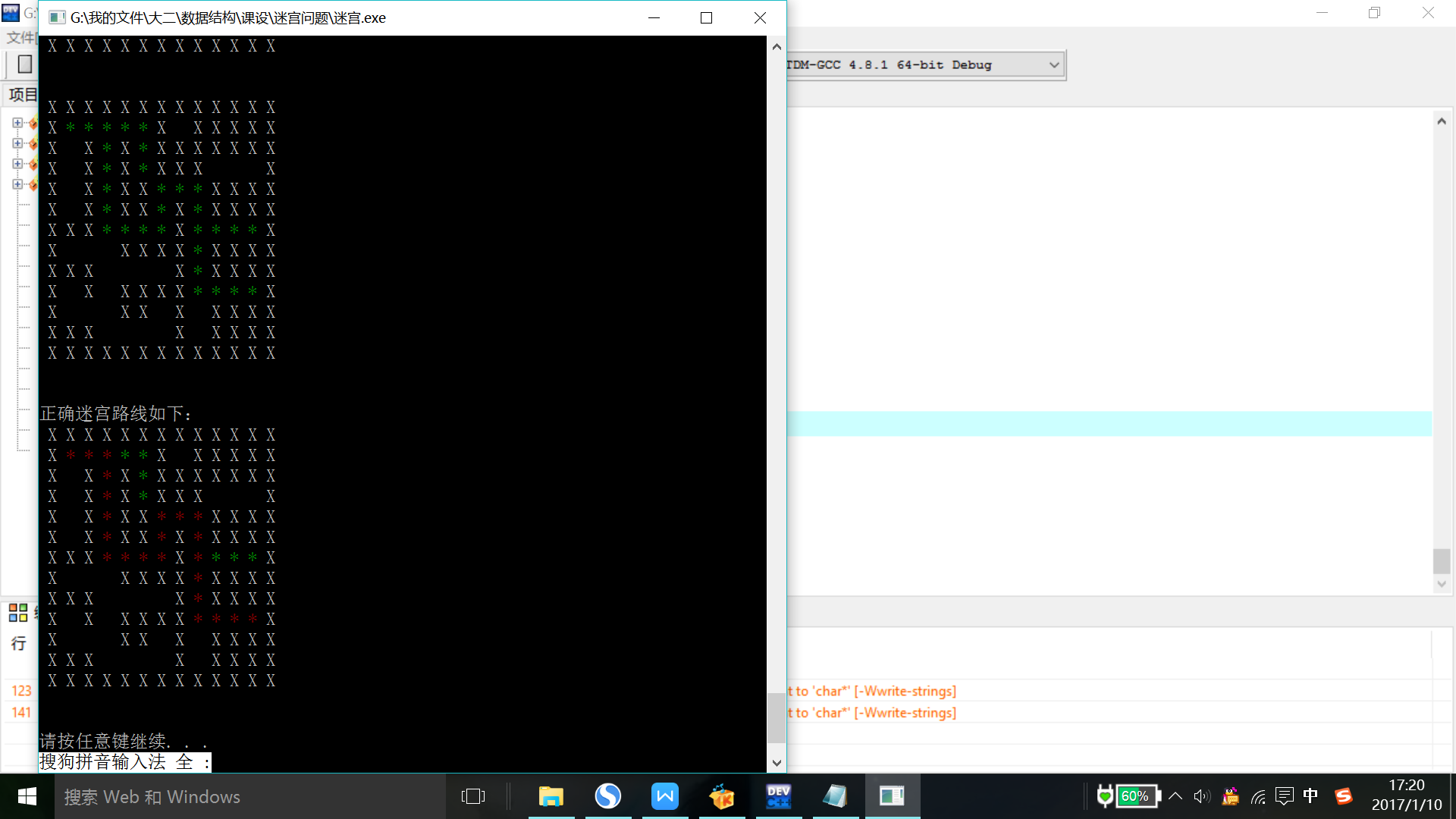
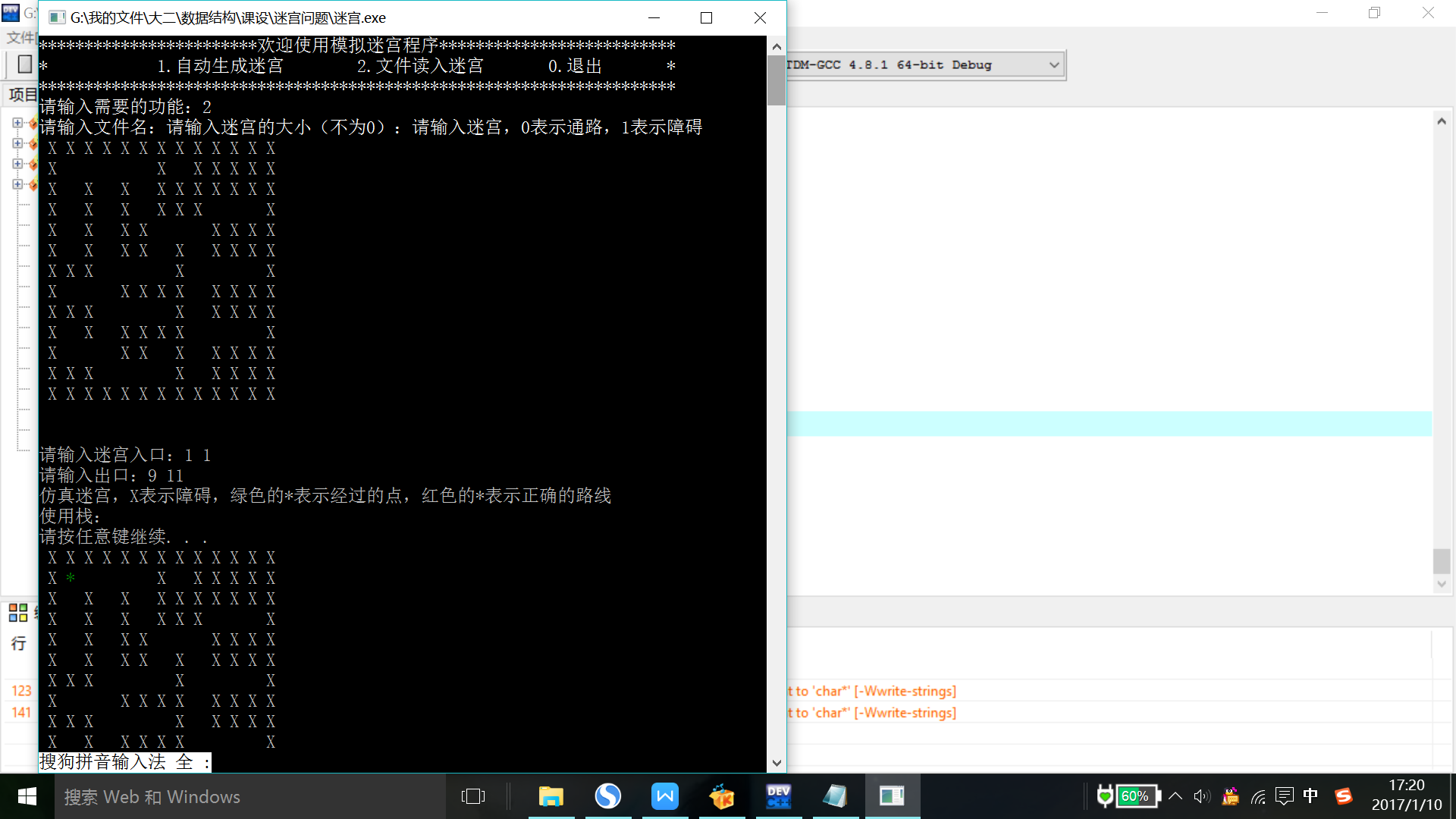
若当前位置“可通”，则纳入“当前路径”，并继续朝“下一位置”探索，即切换“下一位置”为“当前位置”，如此重复直至到达出口；

若当前位置“不可通”，则应顺着“来向”退回到“前一通道块”，然后朝着除“来向”之外的其他方向继续探索；

若该通道块的四周4个方块均“不可通”，则应从“当前路径”上删除该通道块。

所谓“下一位置”指的是当前位置四周4个方向（东、南、西、北）上相邻的方块。若该位置可通返回true，若不可通，返回false；

9.4测试数据和结果



9.5算法时间复杂度

n为矩阵的长。

exitLabyrinth\_D(Labyrinth &L,int row,int column,int outrow,int outcolum)

递归时间复杂度O(n²)

exitLabyrinth\_F(SqStack &S,Labyrinth &L,location start,location end)

栈时间复杂度O(n²)

9.6源代码

#include<iostream>

#include<fstream>

#include <iomanip>

#include<cstdlib>

#include<time.h>

#include <windows.h>

using namespace std;

#define MAX\_VERTEX\_NUM 100

#define STACK\_INIT\_SIZE 1000

typedef int AdjMatrix[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

typedef struct {

AdjMatrix arcs; //邻接矩阵

int rows; //行

int columns; //列

}Labyrinth; //定义迷宫

void Show(Labyrinth L);

typedef struct{

int x;

int y;

}location;

typedef struct{

int ord; //路径标号

location seat; //坐标位置

int di; //走向下一块方向

}SELemType;

typedef struct{

SELemType \*base;

SELemType \*top;

int stacksize;

}SqStack;

void miniPath(SqStack S,Labyrinth &L);

bool NoPass(Labyrinth L,int i,int j);

void initLabyrinth1(Labyrinth &L);

void initLabyrinth2(Labyrinth &L,char \*name);

void Show(Labyrinth L);

bool exitLabyrinth\_D(Labyrinth &L,int row,int column,int outrow,int outcolum);

void InitStack(SqStack &S);

bool DestroyStack(SqStack &S);

bool StackEmpty(SqStack S);

bool Push(SqStack &S,SELemType e);

bool Pop(SqStack &S,SELemType &e);

bool exitLabyrinth\_F(SqStack &S,Labyrinth &L,location start,location end);

void miniPath(SqStack S,Labyrinth &L);

bool exitLabyrinth\_D(Labyrinth &L,int row,int column,int outrow,int outcolum) { //递归 深度优先搜索

bool done = false; // 如果位置(row,column)可通行，标记之，并依次向四周搜索着前进

if(NoPass(L,row,column)==true)

{

// 走过的位置标记为8

L.arcs[row][column]=8;

Sleep(500);

Show(L);

// 抵达终点时表示穿越完成

if(row==outrow&&column==outcolum)

done=true;

。。。。。