项目说明文档

数据结构课程设计

——**勇闯迷宫游戏**

作 者 姓 名： 肖杨

学 号： 1950430

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 分析 3

2 设计 3

3 实现 8

4 用于测试的迷宫 15

1 分析

1.1 项目要求分析

在一个迷宫中有一个入口，有一个出口，从出口开始有很多个可以通过的位置，也有很多障碍， 现在需要从中找出一条通路能够从入口到达出口，走出迷宫。

1.2 功能分析

已经有一个地图、起点和终点的位置，需要从一个地图中找到一条从出口到入口的通路，这需要程序能保存路径和进行迷宫搜索。

2 设计

2.1 数据结构设计

设计时考虑输入任意迷宫，额外设计了输入功能，迷宫会根据实际大小储存在对应大小的二维数组中。其次需要能进行迷宫的搜索工作，本项目采取的是用栈实现的深度优先搜索(DFS)。最后还需要保存路径，由于路径长度无法在求解之前确定，为了减少总体代码量，选择用另一个栈将最后一次DFS所储存的路径反向保存，再依次出栈，即可实现输出路径。

2.2 类结构设计

为了保证设计的数据结构的泛用性，本项目选择将stack类模板类。为了让模板类能实际应用于本项目，还设计了Position类来储存迷宫中的位置信息。最后将所有功能需求和迷宫打包成一个类Maze(具体在下文介绍)。

2.3 成员与操作设计

栈类

类定义：

template <class T>

class stack

{

public:

stack(int size);

~stack();

bool push(T elem);

void expand();

int size();

T pop();

bool empty();

T peek();

private:

T\* m\_buffer;

int m\_size;

int m\_element;

};

私有成员：

T\* m\_buffer;

//储存数据元素的地址

int m\_size;

//栈当前大小

int m\_element;

//栈当前元素数量

公有操作：

stack(int size);

//构造函数

bool push(T elem);

//栈顶增加元素

void expand();

//push中自动调用，扩大栈大小

int size();

//返回栈的大小

T pop();

//出栈并返回出栈元素

bool empty();

//判栈是否为空

T peek();

//返回栈顶元素

Position类

Position类是在Maze类内定义的类

类定义：

class Position

{

public:

int x = -1;

int y = -1;

Position(int column, int row)//x表示纵坐标，y表示横坐标

{

x = column;

y = row;

}

Position() = default;

bool operator ==(Position pos)

{

return pos.x == this->x && pos.y == this->y;

}

};//重载等号

Maze类

类定义：

class Maze

{

private:

//已省略Position类

stack<Position> m\_way;

stack<Position> m\_outPrint;

Position m\_position;

Position m\_exit;

Position m\_entry;

Position m\_nullposition;

const int m\_column;

const int m\_row;

string\* p\_maze;

bool\*\* isVisited;

bool isInBound(Position pos);//判断位置是否在迷宫内

bool isAvaiable(Position pos);//判断位置是否可行走

void printMaze();

void getWay();

public:

Maze(int column, int row);

~Maze();

void getOut();//出迷宫

};

私有成员：

stack<Position> m\_way;

//可通路径

stack<Position> m\_outPrint;

//用于输出的栈

Position m\_position;

//当前位置

Position m\_exit;

//出口

Position m\_entry;

//入口

Position m\_nullposition;

//用于判断位置是否为空或不存在

const int m\_column;

//列长度

const int m\_row;

//行长度

string\* p\_maze;

//迷宫本身

bool\*\* isVisited;

//判断是否已访问

私有操作：

bool isInBound(Position pos);

//判断位置是否在迷宫内

bool isAvaiable(Position pos);

//判断位置是否可行走

void printMaze();

//输出迷宫

void getWay();

//已经找到路径，将路径输入用于输出的栈

公有操作：

Maze(int column, int row);

//构造函数

void getOut();

//初始化完毕后，随时调用即可输出迷宫路径

2.4 总体设计

打开程序后，首先输入行列长度，随后输入迷宫，此时会有输入检测保证输入的长度合法并且字符合法。随后会调用getout函数，若存在合法路径则输出通路，否则给出提示。（注：本程序允许迷宫无墙壁，同时允许不存在出入口，但不允许多个出入口）

本程序使用了宏定义如下：

#define BLOCK '#'

#define BLANK 'O'

#define WAY '1'

#define EXIT '$'

#define ENTRY '@'

#define VISITED true

#define UNVISITED false

3 实现

3.1 迷宫展示的实现

3.1.1 核心代码

void Maze::printMaze()

{

cout << " ";

for (int i = 0; i < m\_row; i++)

{

cout << "第" << i << "列" << " ";

}

cout << endl;

for (int i = 0; i < m\_column; i++)

{

for (int j = -1; j < m\_row; j++)

{

if (j == -1)

{

cout << "第" << i << "行" << " ";

}

else

{

cout << " " << p\_maze[i][j] << " ";

}

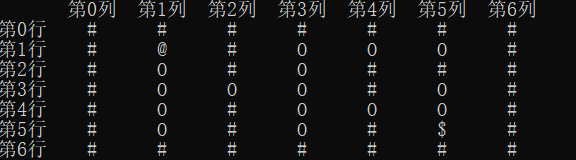
}

cout << endl;

}

}

3.1.2 实例迷宫展示



3.2 迷宫输入的实现

3.2.1 迷宫输入流程

首先，先输入迷宫行列长度，之后在构造函数中持续写入迷宫并储存。内部含有各类型的输入判断，可以保证输入合法的情况下进行储存。

3.2.2 核心代码

Maze::Maze(int column, int row) :m\_column(column), m\_row(row), m\_way(1000), m\_outPrint(1000)

{

p\_maze = new string[column];

cout << "请按行输入迷宫内容，#表示墙壁，大写字母O表示通路，@表示入口，$表示出口" << endl;

string buffer;

bool isAllowed = true;

bool haveEntry = false, haveExit = false;

Position tempentry, tempexit;

int i = 0, j = 0;

while (i < m\_column)

{

isAllowed = true;

cin >> buffer;

if (buffer.size() != m\_row)

{

cout << "请检查输入元素数量是否与行长度相等并重新输入" << endl;

continue;

}

else

{

for (j = 0; j < buffer.size(); j++)

{

if (buffer[j] == BLANK || buffer[j] == BLOCK)

{

continue;

}

else if (buffer[j] == ENTRY)

{

if (!haveEntry&&tempentry == m\_nullposition)

{

tempentry = Position(i, j);

}

else

{

isAllowed = false;

break;

}

}

else if (buffer[j] == EXIT)

{

if (!haveExit&&tempexit == m\_nullposition)

{

tempexit = Position(i, j);

}

else

{

isAllowed = false;

break;

}

}

else

{

isAllowed = false;

break;

}

}

if (isAllowed)

{

p\_maze[i] = buffer;

if (!haveEntry && !(tempentry == m\_nullposition))

{

m\_entry = tempentry;

haveEntry = true;

}

if (!haveExit && !(tempexit == m\_nullposition))

{

m\_exit = tempexit;

haveExit = true;

}

cout << "本行输入成功" << endl;

i++;

}

else

{

tempentry = m\_nullposition;

tempexit = m\_nullposition;

cout << "输入不合法字符或出现多个出入口" << endl;

continue;

}

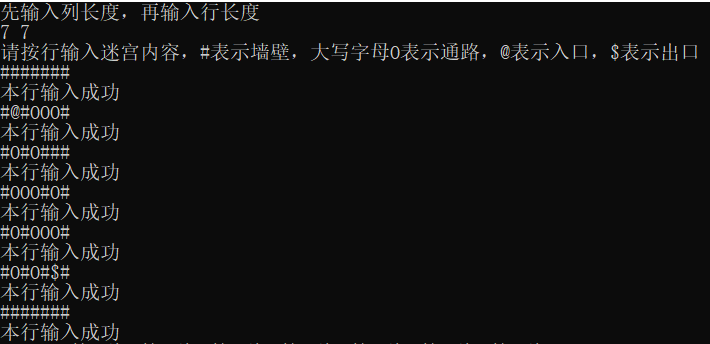
}

}

printMaze();

}

3.2.3 实例迷宫输入



3.3 路径求解的实现

3.3.1 路径求解的流程

首先找到出入口，当都存在时才进行求解。

之后从入口开始，将找到的可通路径存入栈，并标记为已访问，随后从新位置出发，直到无路可走或找到终点。当无路可走时，将栈顶出栈，从新栈顶寻找通路。

最后若有通路，将得到一个存放路线所有位置的栈，否则得到空栈并提示。若有通路，另外输出走出迷宫的路径与经过的路线。

3.3.2 核心代码

void Maze::getOut()

{

if (m\_entry == m\_nullposition || m\_exit == m\_nullposition)

{

cout << "没有出口/入口，无法给出路线" << endl;

return;

}

m\_position = m\_entry;

isVisited = new bool\*[m\_column];

int row, column;

Position temp;

for (int i = 0; i < m\_column; i++)

{

isVisited[i] = new bool[m\_row];

for (int j = 0; j < m\_row; j++)

{

isVisited[i][j] = false;

}

}

m\_way.push(m\_position);

isVisited[m\_position.x][m\_position.y] = VISITED;

while (!(m\_position == m\_exit))

{

row = m\_position.y;

column = m\_position.x;

temp = m\_position;

for (int i = 0; i < 4; i++)

{

if (isAvaiable(Position(column + DIRECTION[i][0], row + DIRECTION[i][1])))

{

temp = Position(column + DIRECTION[i][0], row + DIRECTION[i][1]);

isVisited[temp.x][temp.y] = VISITED;

m\_way.push(temp);

p\_maze[temp.x][temp.y] = WAY;

break;

}

}

if (m\_position == temp)

{

m\_way.pop();//此路不通

p\_maze[m\_position.x][m\_position.y] = BLANK;

if (m\_way.empty())

{

cout << "无可通路径" << endl;

return;

}

else

{

m\_position = m\_way.peek();//获取栈顶

}

}

else

{

m\_position = temp;

}

}//出循环，说明已经有一条通路

getWay();

p\_maze[m\_exit.x][m\_exit.y] = EXIT;

printMaze();

cout << endl;

cout << "迷宫路径：" << endl << endl;

while (!m\_outPrint.empty())

{

temp = m\_outPrint.pop();

cout << '<' << temp.x << ',' << temp.y << '>' << ' ';

if (!m\_outPrint.empty())

{

cout << "---> ";

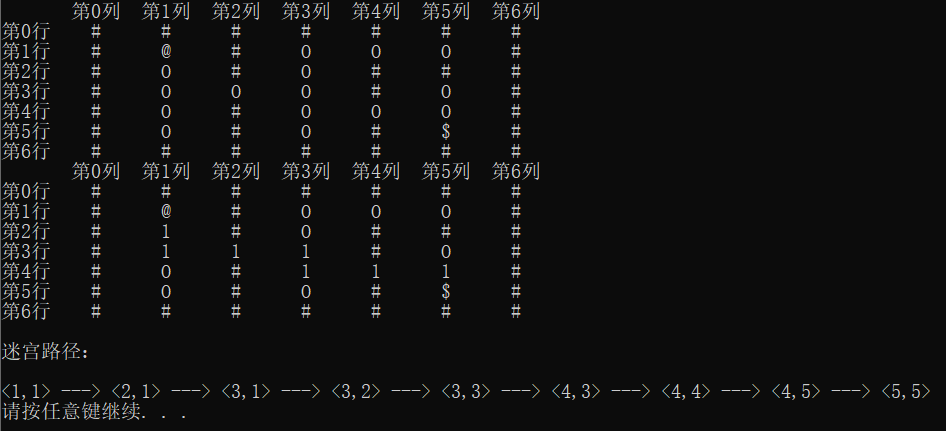
}

}

cout << endl;

}

3.3.3 实例迷宫通路与输出



3.3.4 其他辅助函数的实现

以下给出用于判断和输出的辅助函数

bool Maze::isInBound(Position pos)

{

return pos.x >= 0 && pos.y >= 0 && pos.x < m\_column&&pos.y < m\_row;

}

bool Maze::isAvaiable(Position pos)

{

return isInBound(pos) && (!isVisited[pos.x][pos.y]) && (p\_maze[pos.x][pos.y] == BLANK || p\_maze[pos.x][pos.y] == EXIT);

}

void Maze::getWay()

{

int size = m\_way.size();

for (int i = 0; i < size; i++)

{

m\_outPrint.push(m\_way.pop());

}

}

4 用于测试的迷宫实例

4.1 小迷宫

7 7

#######

#@#OOO#

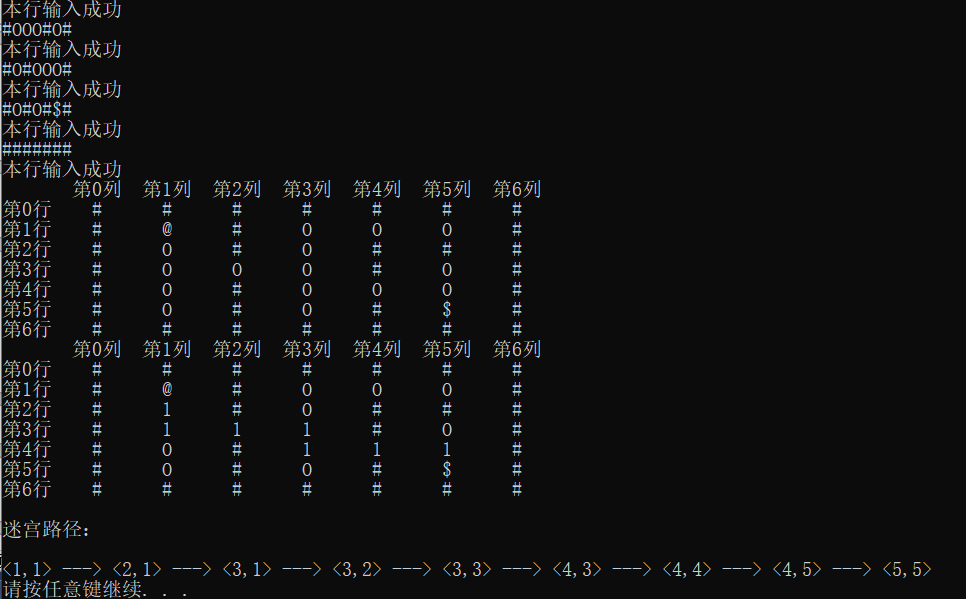
#O#O###

#OOO#O#

#O#OOO#

#O#O#$#

#######



4.2 大迷宫

25 26

##########################

#@#O##OO#OOOO#O#OOO##OO#O#

#O#OOOO#OO##OOOOO#OOO#OOO#

#OO#O#OO##OO###O#O##OO#O##

##O#OO#OOO#OOO#O#O#OO#OO##

#OO##OO##OOO#O#O#OO#O##OO#

#O#OOO#O##O##OOO##O#OOO#O#

#OO#O#OOOO#OO#O#O#OO##O#O#

#O#OO#O##OO#OO##OOO#OOO#O#

#OOO#OOOO#OOO#OOO#OO#O#OO#

#O##O##O#$#O#OO##O#O#OO#O#

##OOOO#OOO#OO#O#OO#OO#OOO#

#OO##OO#O#OO##OOO#O#OO##O#

#O#OO#OO#OO#OO#O#OOOO#OOO#

#O##OO#OOO#OO#OOO##O#OO###

#OOO#OO##OO#O#O##OOOO#O#O#

#O#OO#OOOO#OOOO##O##O#OOO#

#OO#OO####OO###OOOO#O###O#

###O#O#OOOO#OOOO###OOOO#O#

#OOOOO#O###O#O##OOOO#O#OO#

#O#####OOO#O#OOO#O##OO#O##

#O#OOOO##OOOO##OO##OO#OOO#

#OOO##OO##O##OO#OOO#O#O#O#

#O###O#OOOOOOO#O##OO#OOO##

#OOOOOO#O##O#OOOOO#OOO#OO#

##########################,

迷宫路径：

<1,1> ---> <2,1> ---> <3,1> ---> <3,2> ---> <4,2> ---> <5,2> ---> <5,1> ---> <6,1> ---> <7,1> ---> <8,1> ---> <9,1> ---> <9,2> ---> <9,3> ---> <8,3> ---> <8,4> ---> <7,4> ---> <6,4> ---> <6,5> ---> <5,5> ---> <4,5> ---> <4,4> ---> <3,4> ---> <2,4> ---> <2,5> ---> <2,6> ---> <3,6> ---> <3,7> ---> <4,7> ---> <4,8> ---> <4,9> ---> <5,9> ---> <5,10> ---> <5,11> ---> <4,11> ---> <4,12> ---> <4,13> ---> <5,13> ---> <6,13> ---> <6,14> ---> <6,15> ---> <5,15> ---> <4,15> ---> <3,15> ---> <2,15> ---> <2,16> ---> <1,16> ---> <1,17> ---> <1,18> ---> <2,18> ---> <2,19> ---> <2,20> ---> <3,20> ---> <4,20> ---> <5,20> ---> <6,20> ---> <6,21> ---> <6,22> ---> <7,22> ---> <8,22> ---> <8,21> ---> <9,21> ---> <10,21> ---> <10,22> ---> <11,22> ---> <11,23> ---> <11,24> ---> <12,24> ---> <13,24> ---> <13,23> ---> <13,22> ---> <14,22> ---> <15,22> ---> <16,22> ---> <16,23> ---> <16,24> ---> <17,24> ---> <18,24> ---> <19,24> ---> <19,23> ---> <20,23> ---> <21,23> ---> <21,22> ---> <22,22> ---> <23,22> ---> <23,21> ---> <24,21> ---> <24,20> ---> <24,19> ---> <23,19> ---> <23,18> ---> <22,18> ---> <22,17> ---> <22,16> ---> <21,16> ---> <21,15> ---> <20,15> ---> <20,14> ---> <20,13> ---> <19,13> ---> <18,13> ---> <18,14> ---> <18,15> ---> <17,15> ---> <17,16> ---> <17,17> ---> <16,17> ---> <15,17> ---> <15,18> ---> <15,19> ---> <14,19> ---> <13,19> ---> <13,20> ---> <12,20> ---> <11,20> ---> <11,19> ---> <10,19> ---> <9,19> ---> <9,18> ---> <8,18> ---> <8,17> ---> <8,16> ---> <9,16> ---> <9,15> ---> <9,14> ---> <10,14> ---> <11,14> ---> <12,14> ---> <12,15> ---> <13,15> ---> <14,15> ---> <14,14> ---> <15,14> ---> <16,14> ---> <16,13> ---> <16,12> ---> <16,11> ---> <17,11> ---> <17,10> ---> <18,10> ---> <18,9> ---> <18,8> ---> <18,7> ---> <19,7> ---> <20,7> ---> <20,8> ---> <20,9> ---> <21,9> ---> <21,10> ---> <22,10> ---> <23,10> ---> <23,9> ---> <23,8> ---> <23,7> ---> <22,7> ---> <22,6> ---> <21,6> ---> <21,5> ---> <21,4> ---> <21,3> ---> <22,3> ---> <22,2> ---> <22,1> ---> <21,1> ---> <20,1> ---> <19,1> ---> <19,2> ---> <19,3> ---> <19,4> ---> <19,5> ---> <18,5> ---> <17,5> ---> <17,4> ---> <16,4> ---> <16,3> ---> <15,3> ---> <15,2> ---> <15,1> ---> <14,1> ---> <13,1> ---> <12,1> ---> <12,2> ---> <11,2> ---> <11,3> ---> <11,4> ---> <11,5> ---> <12,5> ---> <12,6> ---> <13,6> ---> <13,7> ---> <14,7> ---> <14,8> ---> <14,9> ---> <13,9> ---> <13,10> ---> <12,10> ---> <12,11> ---> <11,11> ---> <10,11> ---> <9,11> ---> <9,10> ---> <8,10> ---> <8,9> ---> <7,9> ---> <7,8> ---> <7,7> ---> <7,6> ---> <8,6> ---> <9,6> ---> <9,7> ---> <10,7> ---> <11,7> ---> <11,8> ---> <11,9> ---> <10,9>