《数据结构》课程实践报告

院、系	计算机学院	年级专业	21 计算机科学与技术	姓名	赵鹏	学号	2127405037
实验布置日期	2022.9.5		提交日期	2022.10.20		成绩	

课程实践实验 5.迷宫求解

一、问题描述及要求

一般的迷宫可表示为一个二维平面图形，将迷宫的左上角作为入口，右下角作为出口。迷 宫问题求解的目标是寻找一条从入口点到出口点的通路。 例如，可设计一个 8×8 矩阵 maze[8][8]来表示迷宫，如下所示

0 1 0 0 0 0 1 1
0 0 0 1 0 0 1 0
1 0 1 0 1 0 1 1
1 0 1 0 1 1 0 1
0 1 1 1 1 1 1 0
1 0 0 1 1 0 0 0
1 0 1 0 0 0 1 1
1 0 1 1 0 1 0 0

左上角 maze[0][0]为起点，右下角 maze[7][7]为终点；设“0”为通路，“1”为墙。假设 一只老鼠从起点出发，目的为右下角的终点，可向“上、下、左、右、左上、左下、右 上、右下”8 个方向行走。 设计一个程序，能自动生成或手动生成一个 8×8 矩阵，针对这个矩阵，程序判断能否从起 点经过迷宫走到终点。如果能，请给出一种走出迷宫的路径。

二、概要设计

(1)本实验的本质是在一张二维网格上找到一条从起点到达终点的路径，使用基于队列的宽度优先搜索算法便可以快速的找到一条起点到达终点的最短路线，记录每个点的前驱节点，最后倒推出路径即可。

（2）本程序主要设计了以下功能：

1. 初始化：

1. 由用户手动输入行数列数以及整个网格

2. 文件读入网格

3. 随机生成网格

2．自定义起点和终点

3. 寻找一条起点到达终点的最短路径并从起点开始依次输出路径上点的坐标。

4. 寻找从起点到终点的可行路径数量

（3）程序运行的界面设计

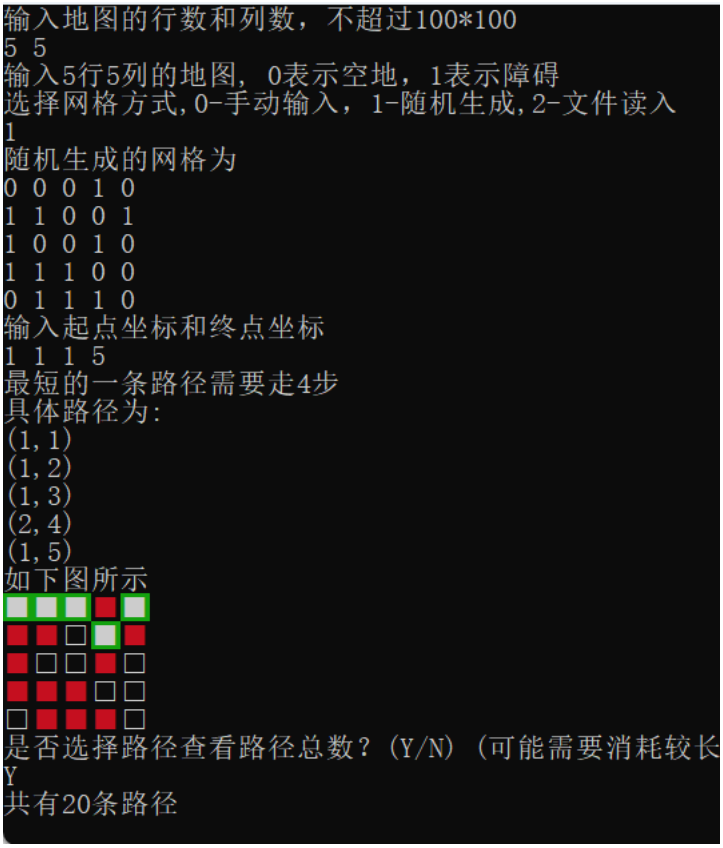
1. 首先提示用户输入地图的行数和列数

2. 随后由用户选择输入方式，0-手动输入，1-随机生成,2-文件读取

3. 然后由用户输入起点终点的坐标。

4. 随后开始寻找路径，若无法找到路径则进行提示，否则输出最短距离和路径上的点坐标，并以图形的方式直观的显示路径。

5. 询问用户是或否需要计算所有可行路径数量。



（4） 项目在设计的过程中采用了队列、栈等数据结构。其中，队列用于实现宽度优先搜索找到一条最短路径。栈用于记录前驱节点后讲路径上的点按正序输出，同时调用系统栈用于 dfs 算法实现统计路径数目。

项目在实现的过程中设计并使用了以下的类：

1. stack 类

函数	功能
push	把元素压入栈中
pop	弹出栈顶元素
Empty	判断栈是否为空
Top	返回栈顶元素

2. queue 类

函数	功能
Push	把元素入队
pop	把元素出队
Empty	判断队列是否为空
Get_front	返回队头元素
Get_size	返回队列中元素数量

3. Maze 类

函数	功能
Maze	构造函数，把网格初始化
Init	获得网格数据
find_path	寻找起点到达终点的最短路径
count_path	计算路径总数
dfs	使用 DFS 算法求解路径总数

其中，stack 类和 queue 类均用于实现 Maze 类中的 find_path 函数，queue 类用于实现宽度优先搜索寻找最短路径，stack 类用于实现将路径正序输出。

(5) 在程序实现的过程中，首先使用到了在课后已经完成的 stack 类和 queue 类，将其简单修改成 queue.h 和 stack.h 并调用，自主设计了 utility.h 用于包含常用的头文件和 Maze.h、Maze.cpp,main.cpp，其中，Maze.h 用于声明迷宫类，包括网格数据，到达每个点的距离等成员。Maze.cpp 用于实现 Maze 类声明的各种函数。

在 Maze 类中,使用了二维数组存储地图,二维数组与二维的地图有着很好的对应关系，很适合用于存储二维网格。

对于最关键的 find_path 函数，在实现过程中使用了宽度优先搜索算法，因为每一步的

距离都为 1，所以使用宽度优先搜索算法按层次遍历，第一次遍历到某一点即为该点到达起到起点的最短路径，在遍历网格的过程中只需要保存每个点第一次是由哪个点走来即可从终点倒推出到达起点的路径。为了正序输出路径，只需要将将路径上的点依次压入栈中，最后从栈顶依次输出并弹出即可正序输出路径。

除了 find_path 函数，count_path 函数也是终点。在实现过程中使用了深度优先搜索算法。通过 count_path 函数调用 Maze 类的私有成员函数 dfs 函数。在 dfs 函数中，枚举八个方向，如果可行则递归调用 dfs 函数，若到达终点则将计数器+1 即可。需要注意的是在递归过程中需要对走过的点进行标记，防止来回走，否则将不断递归直至爆栈。

三、详细设计

本程序的主函数较为简洁，主要是输入地图的行数列数，对地图初始化以及输入起点终点。

程序的重点在于 find_path 函数的实现。首先，可以使用 pair<int,int>存储坐标，可以一定程度的简化程序。实现 find_path 函数，首先需要把初始化所有距离为-1，代表为访问过，其次把起点入队并记录起点距离为 0，可以使用两个长度为 8 一维数组记录方向，随后开始循环，每次取出队头元素，遍历从队头元素开始走的八个方向，如果下一步在网格内切不是障碍切距离为-1，则把下一个点入队，并记录下一个点的前驱节点取出的队头的节点。当队列为空时即完成了遍历。

count_path 函数是程序的另一个重点，它使用了基于递归的 DFS 算法，通过系统栈枚举每一个点分别向八个方向走的方案，如果可行就递归调用并将参数中的坐标修改为移动后的值并传递到下一层。但由于每一个点都需要枚举八个方向，因此时间复杂度为指数级别，对于地图较大时需要较长的时间进行计算。使用深度优先搜索算法求解路径的时间可能极长也是在寻找路径时使用基于队列的宽度优先搜索的算法的原因。

四、实验结果

测试输入：

```
8 8
0
0 1 0 0 0 0 1 1
0 0 0 1 0 0 1 0
1 0 1 0 1 0 1 1
1 0 1 0 1 1 0 1
0 1 1 1 1 1 1 0
1 0 0 1 1 0 0 0
1 0 1 0 0 0 1 1
1 0 1 1 0 1 0 0
1 1 8 8
```

Y

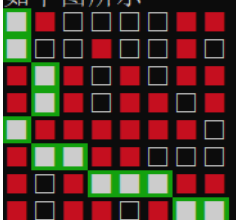
测试目的：正常输出路径

正确输出：

(1,1)
(2,1)
(3,2)
(4,2)
(5,1)
(6,2)
(6,3)
(7,4)
(7,5)
(7,6)
(8,7)
(8,8)

实际输出：

```
输入地图的行数和列数，不超过100*100
8 8
输入8行8列的地图，0表示空地，1表示障碍
选择网格方式,0-手动输入，1-随机生成,2-文件读入
0
请手动输入8行8列的网格
0 1 0 0 0 0 1 1
0 0 0 1 0 0 1 0
1 0 1 0 1 0 1 1
1 0 1 0 1 1 0 1
0 1 1 1 1 1 1 0
1 0 0 1 1 0 0 0
1 0 1 0 0 0 1 1
1 0 1 1 0 1 0 0
输入起点坐标和终点坐标
1 1 8  8
最短的一条路径需要走11步
具体路径为:
(1,1)
(2,1)
(3,2)
(4,2)
(5,1)
(6,2)
(6,3)
(7,4)
(7,5)
(7,6)
(8,7)
(8,8)
如下图所示
```



```
是否选择路径查看路径总数？(Y/N) (可能需要消耗较长时间求解)
Y
共有6624条路径
```

测试结论：通过

测试输入：

10 10

1

1 1 10 10

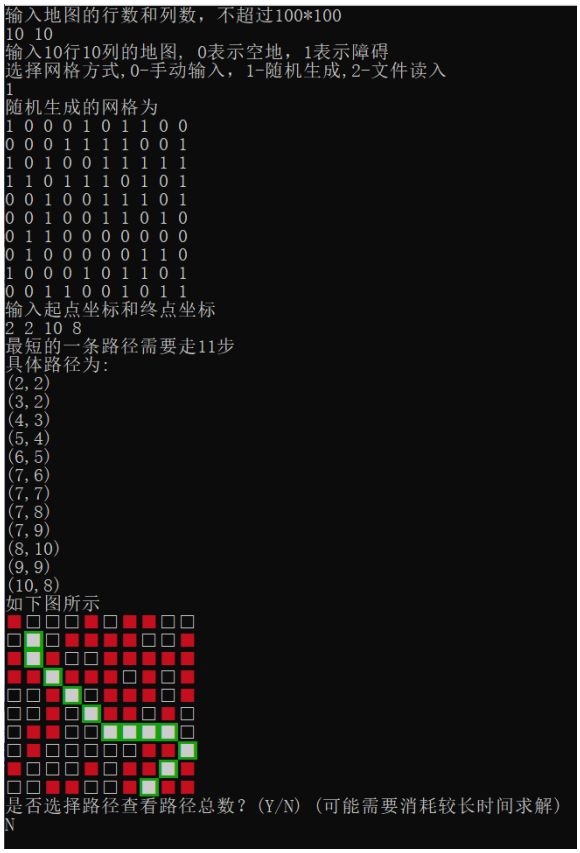
2 2 10 8

N

测试目的：测试能否正确随机生成地图并寻找路径

正确输出：随机生成的网格，正确的路径/无法找到路径

实际输出：



测试结论：通过

测试输入：

5 5

2

D:/test.txt

1 1 5 5

Y

测试目的：测试能否正确从文件读入网格

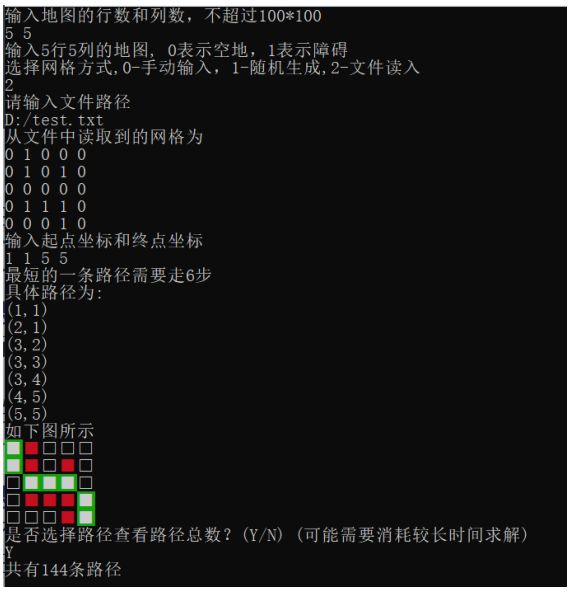
正确输出：

(1,1)

(2,1)

(3,2)
(3,3)
(3,4)
(4,5)
(5,5)

实际输出：



测试结论：通过

测试输入：

5 5

0

0 1 0 0 0

0 1 0 1 0

0 0 0 0 0

0 1 1 1 0

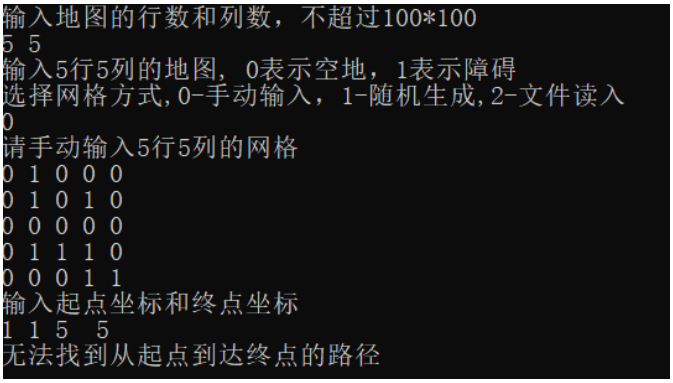
0 0 0 1 1

1 1 5 5

测试目的：是否能正确判断无法找到路径的情况

正确输出：无法找到从起点到达终点的路径

实际输出：



测试结论：通过

五、实验分析与探讨

本次测试测试了程序的各种三种初始化方式，包括手动输入、随机生成、文件读入，以及存在路径和不存在路径的情况。程序均通过了测试。由于使用了宽度优先搜索算法，最坏情况是把整张网格遍历一遍，时间复杂度为 O(Rows*Cols)，由于多次存储网格，空间复杂度同样为 O(Rows*Cols)，时空复杂度均较为优秀。

对于寻找路径问题，除了宽度优先搜索算法以外，还可以使用深度优先搜索算法，深度优先算法可以找到所有的路径，但时间复杂度较不稳定，最坏为指数级别，除了宽度优先搜索算法和深度优先搜索算法，还可以使用 A*算法等启发式搜索算法，通常将当前点到达终点的曼哈顿距离作为估价函数。除了搜索类算法外，还可以使用图论相关的最短路算法，如 dijkstra 算法，bellman_ford 算法、floyd 算法等，在此不多赘述。

在实现本程序的过程中，遇到了以下问题：

1.

问题：使用基于栈的深度优先搜索算法寻找最短路径时间复杂度太高

解决办法：使用基于队列的宽度优先搜索算法。

2.

问题：寻找最短路径时存储路径不容易实现

解决办法：记录每个点的前驱节点，最后倒推出路径即可。

3.

问题：计算路径总数时来回走导致无限递归

解决办法：对已经走过的点进行标记。

六、小结

通过本次的实验，我的 C++代码编写能力有所提升，同时，我仍然需要进一步学习面向对象的思想以及编程方法，对数据结构的掌握程度也仍然需要加强。

在本次实验中，我实现了手动输入、文件读入、随机生成以及自定义终点和打印路径等多种功能。但程序仍然有完善空间，比如可以可视化的输出路径，用箭头表示地图上点的行走方向等。

附录: 源代码

1 实验环境： Visual Studio 2022

2、

1. Maze.h

```
#pragma once
```

```
class Maze
{
private:
	static const int max_size = 105;
	int map[max_size][max_size];
	int dist[max_size][max_size];
	int st[max_size][max_size];
	int in_path[max_size][max_size];
	int Row, Col;
	int pathNum = 0;
	int dx[8] = { 1,-1,0,0 ,1,1,-1,-1 };
	int dy[8] = { 0,0,1,-1 ,1,-1,1,-1 };
	void dfs(int x,int y,int tx,int ty);
public:
	Maze(int Rows, int Cols);
	void find_path(int sx, int sy,int tx,int ty);
	void init(int Rows, int Cols);
	int count_path(int sx,int sy,int tx,int ty);

};
```

2. queue.h

```
#pragma once
template <typename Data>
class Queue
{

private:
	Data data_queue[10000];
	int front = 0;
	int rear = -1;
	int siz = 0;

public:
	void push(Data u)
	{
		data_queue[++rear] = u;
		siz++;
	}
	void pop()
	{
		data_queue[front++];
		siz--;
	}
	bool empty()
```

```
    {
        return siz == 0;
    }
    Data get_front()
    {
        return data_queue[front];
    }
    int get_size()
    {
        return siz;
    }
};
```

3. stack.h

```
#pragma once
template<typename T1>
class stack
{
private:
 T1 stk[10000];
 int Bottom = 0;
 int Top = -1;
public:
 void push(T1 u)
 {
     stk[++Top] = u;
 }
 void pop()
 {
     Top--;
 }
 bool empty()
 {
     return Top < Bottom;
 }
 T1 top()
 {
     return stk[Top];
 }

};
```

4. utility.h

```
#pragma once
#include<iostream>
```

```
#include<cstring>
#include<ctime>
#include<fstream>
using namespace std;
```

5. Maze.cpp

```
#include "Maze.h"
#include "queue.h"
#include "stack.h"
#include "utility.h"
#define CONSOLE_RED "\033[31m"
#define CONSOLE_RESET "\033[0m"
#define CONSOLE_GREEN "\033[42m"
Maze::Maze(int Rows, int Cols)
{

        Row = Rows;
        Col = Cols;
        memset(in_path, 0, sizeof in_path);
    for (int i = 0; i <= Rows; i++)
        for (int j = 0; j <= Cols; j++)
            map[i][j] = 1;


}
void Maze::init(int Rows, int Cols)
{
 cout << "输入"<<Rows<<"行"<<Cols<<"列的地图, 0 表示空地，1 表示障碍" <<
endl;
    int choice;
    cout << "选择网格方式,0-手动输入，1-随机生成,2-文件读入" << endl;

    cin >> choice;
    if (choice == 0)
    {
        cout << "请手动输入" << Rows << "行" << Cols << "列的网格" << endl;

        for (int i = 1; i <= Rows; i++)
            for (int j = 1; j <= Cols; j++)
                cin >> map[i][j];
    }
    else if (choice == 1)
    {
        srand(time(NULL));
```

```
            for (int i = 1; i <= Rows; i++)
                for (int j = 1; j <= Cols; j++)
                    map[i][j]=rand()%2;
            cout << "随机生成的网格为"<<endl;
            for (int i = 1; i <= Rows; i++)
            {
                for (int j = 1; j <= Cols; j++)
                {
                    cout << map[i][j] << " ";
                }
                cout << endl;
            }
        }
        else if (choice == 2)
        {
            string path, line;
            cout << "请输入文件路径" << endl;

            cin >> path;
            ifstream iFile;
            iFile.open(path);
            while (!iFile.is_open())
            {
                cout << "路径错误，请重新输入" << endl;
                cin >> path;
                iFile.open(path);
            }
            for (int i = 1; i <= Rows; i++)
                for (int j = 1; j <= Cols; j++)
                    iFile >> map[i][j];

            cout << "从文件中读取到的网格为" << endl;
            for (int i = 1; i <= Rows; i++)
            {
                for (int j = 1; j <= Cols; j++)
                {
                    cout << map[i][j] << " ";
                }
                cout << endl;
            }
        }

}
void Maze::find_path(int sx, int sy,int tx,int ty)
```

```
    {
        if (map[sx][sy] == 1)
        {
            cout << "无法找到从起点到达终点的路径" << endl;
            return;
        }
        typedef pair<int, int> pii;
        pii pre[105][105];
        Queue<pii>q;
        stack<pii>stk;
        memset(dist, -1, sizeof dist);
        q.push({ sx,sy });
        pre[sx][sy] = { 0,0 };
        dist[sx][sy] = 0;
        int dx[8] = { 1,-1,0,0 ,1,1,-1,-1};
        int dy[8] = { 0,0,1,-1 ,1,-1,1,-1};
        while (q.get_size())
        {
            auto t = q.get_front(); q.pop();
            for (int k = 0; k < 8; k++)
            {
                int xx = t.first + dx[k], yy = t.second + dy[k];
                if (xx >= 1 && xx <= Row && yy >= 1 && yy <= Col && dist[xx][yy]
== -1 && map[xx][yy] == 0)
                {
                    dist[xx][yy] = dist[t.first][t.second] + 1;
                    pre[xx][yy] = { t.first,t.second };

                    q.push({ xx,yy });
                }
                if (dist[tx][ty] != -1)break;
            }
        }
        if (dist[tx][ty] == -1)
        {
            cout << "无法找到从起点到达终点的路径" << endl;
            return;
        }
        cout <<"最短的一条路径需要走" << dist[tx][ty]<<"步" << endl;
        pii now_palce1 = { tx,ty };
        in_path[tx][ty] = 1;
        pii temp = { 0,0 };
        while (now_palce1 != temp)
        {
```

```
            stk.push(now_palce1);
            now_palce1 = pre[now_palce1.first][now_palce1.second];
            in_path[now_palce1.first][now_palce1.second] = 1;
        }

        pii now_place = { 1,1 };
        pii target_palce = { tx,ty };
        cout << "具体路径为:" << endl;

        while (!stk.empty())
        {
            auto temp = stk.top(); stk.pop();
            cout <<"(" << temp.first << "," << temp.second<<")" << endl;
        }

        cout << "如下图所示" << endl;

        for (int i = 1; i <= Row; i++)
        {
            for (int j = 1; j <= Col; j++)
            {
                if (in_path[i][j])
                    cout << CONSOLE_GREEN <<"■"<<CONSOLE_RESET;
                else
                {
                    if (map[i][j] == 0)
                        cout << "□";
                    else
                        cout<< CONSOLE_RED << "■"<< CONSOLE_RESET;
                }
            }
            cout << endl;
        }
        cout << "是否选择路径查看路径总数？(Y/N) (可能需要消耗较长时间求解)"<<endl;
        string choice;
        cin >> choice;
        if(choice=="y"||choice=="Y")
            cout << "共有" << count_path(sx, sy, tx, ty) << "条路径" << endl;


        return;

    }
```

```
int Maze::count_path(int sx,int sy,int tx,int ty)
{
    memset(st, 0, sizeof st);
    if (map[sx][sy] == 1)
        return 0;
    dfs(sx, sy, tx, ty);
    return pathNum;
}
void Maze::dfs(int x, int y, int tx, int ty)
{
    if (x == tx && y == ty)
    {
        pathNum++;
        return;
    }

    for (int i = 0; i < 8; i++)
    {
        int xx = x + dx[i];
        int yy = y + dy[i];
        if (xx >= 1 && xx <= Row && yy >= 1 && yy <= Col && !st[xx][yy] &&
map[xx][yy] == 0)
        {
            st[xx][yy] = 1;
            dfs(xx, yy,tx,ty);
            st[xx][yy] = 0;
        }
    }
}
```

6. main.cpp

```
#include "utility.h"
#include "stack.h"
#include "queue.h"
#include "Maze.h"
int main()
{
    int n, m;
    cout << "输入地图的行数和列数，不超过 100*100" << endl;
    while (cin >> n >> m)
    {
        if (n >= 0 && n <= 100 && m >= 0 && m <= 100)break;
        else cout << "数据不合法，请重新输入" << endl;
    }
```

```
    Maze maze(n, m);
    maze.init(n,m);
    cout << "输入起点坐标和终点坐标" << endl;
    int sx, sy, tx, ty;
    cin >> sx >> sy >>tx >> ty;
    maze.find_path(sx,sy,tx,ty);

    return 0;
}
```