**2019秋季学期**

**数据结构实验报告**

**实验四 图搜索实验**

**班级：\_\_\_\_计实验19\_\_\_\_\_\_\_\_**

**学号：\_\_\_\_\_19101130122\_\_\_\_\_\_\_**

**姓名：\_ 陈旻杰\_\_\_\_\_\_\_\_\_\_**

|  |
| --- |
| **评语：**  **日期：** |

**实验目的：**

熟悉非线性结构的特点 , 掌握非线性结构的存储方式及各种操作的实现方法，同时对自顶向下的程序设计方法、应用程序界面的设计、非线性结构的文件存储方法等方面的辑程技术进行训练。

1、迷宫问题

**问题描述**：

一只老鼠走进了一个迷宫，这个迷宫是由M行N列的方格构成的，相邻方格之间可能是相通的，也可能有墙相隔，各方格位置由其对应坐标确定，如图所示为5行5列的迷宫，矩阵中1代表是墙走不通，0表示可以通行。迷宫在（1，1）处有一个入口，在（M,N）处有一个出口。问题是让你帮助老鼠找出从入口到出口一共有多少条路径，最短路径需多少步到达出口。

**基本要求:**

采用深度优先搜索计算路径数，用广度优先搜索找最短路径。

**Input**

输入第一行为2个整数M和N（M、N不超过10），表示迷宫为M行N列，从第二行开始共有M行，每行N个整数（0或1），各数之间由空格分隔。

**Output**

输出包括2行，第一行为路径个数，第二行为最快到达出口的步数。

**Sample Input**

**5 5**

**0 1 0 0 0**

**0 1 0 1 0**

**0 0 0 0 0**

**0 1 0 1 0**

**0 0 0 1 0**

**Sample Output**

**4**

**8**

**需求分析：**（包括对问题的理解，解决问题的策略、方法描述）

问题的理解：

这题就是一个简单的搜索题，还不算是图论题。既然是搜索题 ，那自然是用搜索来解决了。迷宫问题是典型的矩阵类搜索问题。

解决问题的策略：

1.位置信息结构体赋予初始值（起点位置，步数为0）

2.从这个位置信息处分别作dfs和bfs找到目标点后就结束。

3.根据最后的位置信息可以知道最小步数，也可以推出路径（从最终点找它的前置点，直到起点）。

4.dfs求可行解个数，而bfs求最优解个数。

方法描述：

1.dfs 和 bfs。dfs求可行解个数，而bfs求最优解个数。

2.方向数组

int p[4][2]={{0,1},{1,0},{0,-1},{-1,0}};

3.a用来存储是否可走， vis用来存储是否走过

int vis[1000][1000];

**系统设计：**（包括数据结构定义、抽象出基本操作描述、主程序模块处理过程描述）

***数据结构定义：***

struct pos

**{**

int x**,**y**,**step**;**

pos**(**int x1**,**int y1**,**int s1**)**

**{**

x **=** x1**;**

y **=** y1**;**

step **=** s1**;**

**}**

**};**

上述是对于迷宫每个位置的数据结构定义。

***抽象出基本操作描述：***

预处理：

int minn **=** 0x7fffffff**;**

int sum**;**

int m**,**n**;**

int p**[**4**][**2**]={{**0**,**1**},{**1**,**0**},{**0**,-**1**},{-**1**,**0**}};**

int a**[**1000**][**1000**];**

int vis**[**1000**][**1000**];**

queue**<**struct pos**>** q**;**

dfs求可行解个数：

void dfs**(**int x**,**int y**)**；

bfs求最优解**;**

void bfs**();**

***主程序模块处理过程描述:***

int main**()**

**{**

cin**>>**m**>>**n**;**

**for** **(**int i **=** 1**;** i **<=** m**;**i**++)**

**{**

**for** **(**int j **=** 1**;** j **<=** n**;**j**++)**

**{**

cin**>>**a**[**i**][**j**];**

**}**

**}**

vis**[**1**][**1**]** **=** 1**;**

dfs**(**1**,**1**);**

bfs**();**

printf**(**"%d\n"**,**sum**);**

**if(**sum**)**

**{**

printf**(**"%d"**,**minn**);**

**}**

**else**

**{**

printf**(**"NoPath"**);**

**}**

**return** 0**;**

**}**

**基本操作的实现：**（对各基本操作实现的描述）

***dfs求可行解个数：***

void dfs**(**int x**,**int y**)**

**{**

**if(**x**==**m**&&**y**==**n**)**

**{**

sum**++;**

**return** **;**

**}**

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

**{**

int tx **=** x **+** p**[**i**][**0**];**

int ty **=** y **+** p**[**i**][**1**];**

**if(**tx**<**1**||**tx**>**m**||**ty**<**1**||**ty**>**n**)**

**{**

**continue;**

**}**

**if(!**a**[**tx**][**ty**]&&!**vis**[**tx**][**ty**])**

**{**

vis**[**tx**][**ty**]** **=** 1**;**

dfs**(**tx**,**ty**);**

vis**[**tx**][**ty**]** **=** 0**;**

**}**

**}**

**return** **;**

**}**

描述：借助递归可以实现走找到存在多少个解的情况，其中主要是额外判断边界和墙的情况

***bfs求最优解***

void bfs**()**

**{**

memset**(**vis**,**0**,sizeof(**vis**));**

struct pos t **=** pos**(**1**,**1**,**0**);**

q**.**push**(**t**);**

vis**[**1**][**1**]** **=** 1**;**

bool flag **=** 0**;**

**while(!**q**.**empty**())**

**{**

t **=** q**.**front**()** **;**

int xx **=** t**.**x **;**

int yy **=** t**.**y **;**

int s **=** t**.**step **;**

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

**{**

int qx **=** xx **+** p**[**i**][**0**];**

int qy **=** yy **+** p**[**i**][**1**];**

**if(**qx**<**1**||**qx**>**m**||**qy**<**1**||**qy**>**n**)**

**continue;**

**if(!**a**[**qx**][**qy**]** **&&** **!**vis**[**qx**][**qy**])**

**{**

vis**[**qx**][**qy**]** **=** 1**;**

t **=** pos**(**qx**,**qy**,**s**+**1**);**

q**.**push**(**t**);**

**}**

**if(**qx **==** m **&&** qy **==** n**)**

**{**

flag **=** 1**;**

**}**

**}**

q**.**pop**()** **;**

**if(**flag**)** **break;**

**}**

**while(**q**.**size**()** **>** 1**)**

**{**

q**.**pop**()** **;**

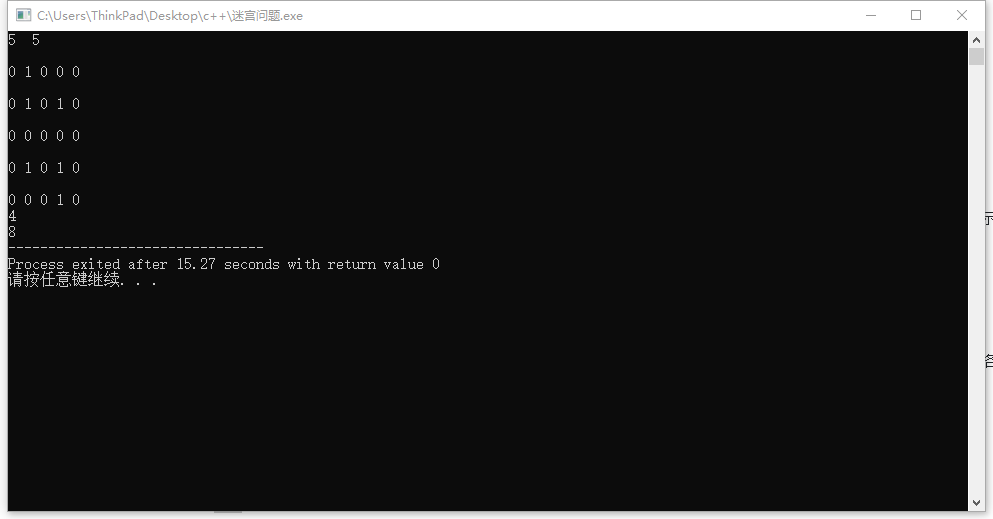
**}**

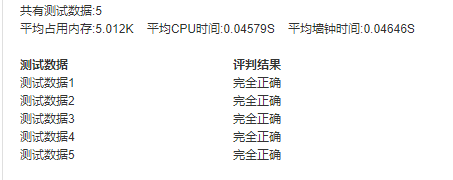
minn **=** q**.**front**().**step**;**

**}**

描述：借助queue数据结构能够在走到的每个点进行操作，以此每走一步计数最后找到最优解情况

**测试结果：**（输入的测试数据及运行结果、正确性、在线测试情况）





**调试分析：**（包括调试过程中对原设计的修改，以及遇到的问题和解决的方法）

1. 位移数组不建议使用move命名，会与库函数重名报错
2. 不建议使用getchar和gets函数，可能会出现意想不到的错误
3. 根据题意dfs求解可行解，bfs求解最优解。但是这题从求解角度来说不需要用bfs。dfs用递归，代码短，且两者复杂度均为O(V+E)。V是点数，E是连接点的边总数。既然dfs可以求出所有可行解，那么最优解自然也是可行解的一种。所以我们可以在用dfs求可行解时，顺便记录一下最优解。即在dfs剪枝中加上一句; minl = min(minl,step);即可求出最优解。即minl = min(minl,step);这一句话可代替整个bfs。求不消耗额外的空间，时间复杂度也是线性的。

***附上：dfs解题***

#include <iostream>

#include <algorithm>

**using** **namespace** std**;**

const int maxn **=** 1010**;**

int a**[**maxn**][**maxn**]** **=** **{**0**};**

bool vis**[**maxn**][**maxn**]** **=** **{**0**};**

int p**[**4**][**2**]** **=** **{{-**1**,**0**},{**1**,**0**},{**0**,-**1**},{**0**,**1**}};**

int ans **=** 0**;**

int minl **=** 1e9**;**

int m**,**n**;**

void dfs**(**int x**,**int y**,**int step**)**

**{**

**if(**x **==** m**&&**y **==** n**)**

**{**

ans**++;**

minl **=** min**(**minl**,**step**);**

**return** **;**

**}**

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

**{**

int tx **=** x**+**p**[**i**][**0**];**

int ty **=** y**+**p**[**i**][**1**];**

**if(**tx**<**1**||**tx**>**m**||**ty**<**1**||**ty**>**n**)**

**{**

**continue;**

**}**

**if(!**a**[**tx**][**ty**]&&!**vis**[**tx**][**ty**])**

**{**

vis**[**tx**][**ty**]** **=** 1**;**

dfs**(**tx**,**ty**,**step**+**1**);**

vis**[**tx**][**ty**]** **=** 0**;**

**}**

**}**

**return** **;**

**}**

int main**()**

**{**

cin **>>** m **>>** n**;**

**for(**int i **=** 1**;**i **<=** m**;**i**++)**

**{**

**for(**int j **=** 1**;**j **<=** n**;**j**++)**

**{**

cin **>>** a**[**i**][**j**];**

**}**

**}**

vis**[**1**][**1**]** **=** 1**;**

dfs**(**1**,**1**,**0**);**

cout **<<** ans **<<** endl**;**

**if(**ans**)** cout **<<** minl **<<** endl**;**

**else** cout **<<** "NoPath" **<<** endl**;**

**return** 0**;**

**}**

int main**()**

**{**

cin **>>** m **>>** n**;**

**for(**int i **=** 1**;**i **<=** m**;**i**++)**

**{**

**for(**int j **=** 1**;**j **<=** n**;**j**++)**

**{**

cin **>>** a**[**i**][**j**];**

**}**

**}**

vis**[**1**][**1**]** **=** 1**;**

dfs**(**1**,**1**,**0**);**

cout **<<** ans **<<** endl**;**

**if(**ans**)** cout **<<** minl **<<** endl**;**

**else** cout **<<** "NoPath" **<<** endl**;**

**return** 0**;**

**}**