目录

[DAY 1 2](#_Toc27649)

[一、课程设计概述： 2](#_Toc5943)

[二、问题分析： 2](#_Toc715)

[DAY 2 3](#_Toc13913)

[一、采用的数据结构与算法 3](#_Toc16124)

[二、算法设计： 3](#_Toc26781)

[三、算法分析： 3](#_Toc22732)

[DAY 3 4](#_Toc23034)

[一、所学知识： 4](#_Toc22126)

[1.BFS算法寻路打印路径思路： 4](#_Toc16778)

[2.随机生成迷宫： 4](#_Toc32126)

[二、代码实现: 4](#_Toc13752)

[1、设计每个成员函数： 4](#_Toc23798)

[（1）生成迷宫 4](#_Toc31469)

[（2）dfs寻找最短路函数 5](#_Toc7074)

[（3）bfs寻找最短路函数 6](#_Toc26788)

[（4）bfs记录迷宫路径函数 6](#_Toc22384)

[2、设计主函数 7](#_Toc8195)

[DAY4： 9](#_Toc1044)

[1. 选择dfs迷宫寻路： 10](#_Toc26741)

[2. 选择bfs迷宫寻路 11](#_Toc27593)

[DAY5: 13](#_Toc27094)

**DAY 1**

**一、课程设计概述：**

迷宫求解是一个经典的程序设计问题，在复杂工程问题的设计和程序设计竞赛中都有其应用或变形。迷宫问题主要利用bfs广度优先搜索，或dfs深度优先搜索进行解决，能否写好迷宫问题是对我们是否准确掌握搜索算法的一个判断途径，也是对我们学好数据结构的检测。

**二、问题分析：**

迷宫问题采用试探法求解，即从入口出发，在当前位置任选一个方向向前试探，若下一位置可通（即该位置为从未走过的通道块）则继续试探行进；若当前位置的所有方向均不通（为墙体或已走过的通道块）时，则按原路返回上一位置，重新选择其他方向继续试探前进。这样不断试探前进，知道到达迷宫出口则求解了一种可行的路径；若沿入口的所有方向都不能到达出口，则说明此迷宫不存在可行路径。这种经试探可行则行进，不可行则返回重新试探的方法，则可以作为该问题的求解方法。

**DAY 2**

**一、采用的数据结构与算法**

1.采用的数据结构：队列结构

2.采用的算法：DFS深度优先搜素与BFS广度优先搜索

3.本设计部分采用的队列的存储结构，因为在进行探索是否有通路的过程中，常进行查找，比较，为了保证在任何位置上都能找全其东、西、南、北四个方向的路径，我选择用“先进先出”的队列结构来保存从入口到当前位置的路径。同时对于此次课程设计，队列的存入、弹出数据的方式比数组更为方便，且不用限定存储空间的大小，这个优点对设计很有利，所以采用了队列的存储结构。

**二、算法设计：**

迷宫问题通常用的是“穷举求解”的方法。为了保证在任何位置上都能原路退回，显然需要用递归来探寻迷宫的路径。因此，在求解迷宫通路的第一个方法中我选择了dfs算法，即深度优先搜素算法。

而为了保证在任何位置上都能找全其东、西、南、北四个方向的路径，我选择用“先进先出”的队列结构来保存从入口到当前位置的路径。因此，在求解通路的第二种方法中我应用“队列”的思想，选择了bfs算法，即广度优先搜索算法求解。

**三、算法分析：**

DFS——采用回溯法实现该问题的求解。回溯法是一种不断试探及时纠正错误的搜索方法。从入口出发，按某一方向向前探索，若能走通(未走过的)，即某处可以到达，则到达新点，否则试探下一方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的通路都搜索到，或找到一条通路，或无路可走又返回到入口点。

BFS——用队列存储路径，当队列中还有值时，继续寻路，从此时的队头开始，寻找其所有方向可走的方块并将可走方块入队后，将其出队。**一层层向外扩展查找可走的方块，直到找到出口为止，最先找到的这个答案就必然是最短的。**

****DAY 3****

****一、所学知识：****

1.BFS算法寻路打印路径思路：

搜索方法使用BFS，即队列方法搜索，广度优先寻找最短路径，但是需要找出最短路径，使用的方法是另开结构体存路径。即 用结构体记录每个点的父节点。找到终点后，从终点开始递归，由最后一个结点依次向前找它们的父节点，最后回溯以打印出最终路径。

2.随机生成迷宫：

思路：用随机数种子任意生成墙和通路，最后用dfs/bfs检验该迷宫是否有解，无解则重新生成。

不难，但是感觉很暴力很巧妙，没学之前想的太复杂了，完全想不到怎么用现有的知识随机生成迷宫。

**二、代码实现:**

**1、设计每个成员函数：**

struct node { //用结构体同时存储该点坐标和父结点坐标

int x,y; //当前位置的坐标

int fromx,fromy; //当前位置坐标由上一个坐标的转移

}

**（1）生成迷宫**

void Make() {

system("cls");

cout<<"请输入制作随机迷宫边长：\n";

cin>>n;

system("cls"); //清屏

printf("迷宫制作中请等待1~3秒");

while (1) {

srand(time(NULL)); //以当前时间为种子，生成随机数

for (int i=1;i<=n;i++) {

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

int num=rand()%3; //根据生成的随机数设置墙/路

if (num==0) {

MAP[i][j]='X'; //墙

} else {

MAP[i][j]='O'; //路

}

}

}

Again(); //把 MAP中的值赋给 MAPbfs和 MAPdfs中

bfs(); //判断是否通路，是则退出函数，否则继续生成迷宫

if (flag==1) break;

}

}

**（2）dfs寻找最短路函数**

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

if (flag) return;

if (x==n&&y==n) { //找到终点

flag=1;

Sleep(3000);

Print\_dfs(step); //打印路径

Again(); //把 MAP中的值赋给 MAPbfs和 MAPdfs中

return;

}

for (int i=0;i<4;i++) { //遍历该点的东西南北四个方向

int nextx=x+turnx[i];

int nexty=y+turny[i];

if (nextx>=1&&nextx<=n&&nexty>=1&&nexty<=n&&(MAPdfs[nextx][nexty]=='S'||MAPdfs[nextx][nexty]=='O'||MAPdfs[nextx][nexty]=='E')) { //边界判断与通路判断

MAPdfs[nextx][nexty]='o';

if (nextx!=n||nexty!=n) Print\_dfs(step); //打印路径

dfs(nextx,nexty,step+1); //从该点继续寻找路径

if (flag) return; //若找到通路，则直接回溯

MAPdfs[nextx][nexty]='x'; //否则 将该店设为未访问状态

Print\_dfs(step); //打印路径

}

}

}

**（3）bfs寻找最短路函数**

void bfs() {

int nx,ny; //下一个坐标的 x，y值

path[1][1].x=path[1][1].y=1;

Q.push(path[1][1]);

while (!Q.empty()) { //队列中还有数时，继续寻路

NOW=Q.front(); //将此时的队头（即当前位置）取出

Q.pop(); //队头出队

if (NOW.x ==n && NOW.y ==n) { //找到出口

flag=1; //做个标记，方便后续判断该迷宫是否有通路

return;

}

for (int i=0;i<4;i++) { //遍历该点的东西南北四个方向

nx=NOW.x+turnx[i];

ny=NOW.y+turny[i];

if (nx>=1 && nx<=n && ny>=1 && ny<=n && MAPbfs[nx][ny]!='X') { //越界和通路判定

path[nx][ny].x =nx;

path[nx][ny].y =ny;

path[nx][ny].fromx =NOW.x ;

path[nx][ny].fromy =NOW.y ; //记录路径

MAPbfs[nx][ny]='X'; //标记已找过的路

Q.push(path[nx][ny]) ; //将该点入队

}

}

}

}

**（4）bfs记录迷宫路径函数**

void vispath(int x,int y) { //运用用递归、回溯，做到寻找、标记、打印路径

memset(vis,0,sizeof(vis)); //初始化

if (x==1&&y==1) {

vis[1][1]=1;

Print\_bfs(); //从起点开始打印路径

return; //找到起点后，开始回溯，输出最终路径

}

int fx=path[x][y].fromx ;

int fy=path[x][y].fromy ;

vispath(fx,fy); //不断找该点的父结点

vis[path[x][y].x ][path[x][y].y ]=1; //回溯后将该点做上标记

Print\_bfs(); //打印路径

}

**2、设计主函数**

int main() {

while (1) {

init();

flag=0;

string choice;

{

system("cls");

system("color 0F");

cout<<"请选择您的操作：\n";

cout<<"1.开始游戏\n";

cout<<"2.退出程序\n";

cin>>choice; //用户选择操作

while (choice!="1"&&choice!="2") { //错误特判

cout<<"操作有误，请重新选择！\n";

cin>>choice;

}

if (choice == "2") return 0;

Make(); //制作迷宫

}

while (1) {

Print\_maze();

cout<<"\n请选择寻找出口的方式：\n";

cout<<"1.dfs深度优先搜索\n";

cout<<"2.bfs广度优先搜索\n";

cout<<"3.返回上一界面\n";

flag=0;

cin>>choice; //用户选择操作

while (choice!="1"&&choice!="2"&&choice!="3") { //错误特判

cout<<"操作有误，请重新选择！\n";

cin>>choice;

}

if (choice == "1") flag=0, dfs(1,1,0);

else if (choice == "2") vispath(n,n), Print\_bfs();

else if (choice == "3") break;

cout<<"\n按 1 返回上一界面（ ￣ ▽ ￣ ）\n";

cin>>choice;

while (choice!="1") { //错误特判

cout<<"输错了哦，请重新输入（ ￣ ▽ ￣ ）\n";

cin>>choice;

}

system("cls");

}

}

return 0;

}

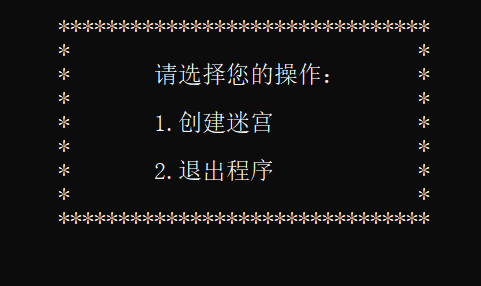
**DAY4：**

**程序使用说明和运行结果**

程序运行后进入初始页面：



按任意键后进入到操作页面：



选择1.创建迷宫后程序会提示出入迷宫的边长:



输入后，程序开始制作迷宫：





1. 选择dfs迷宫寻路：

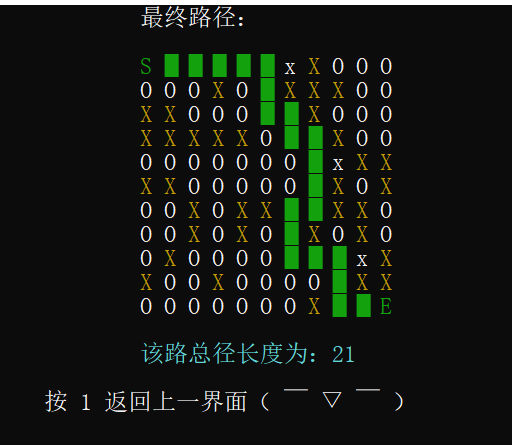
程序会打印出寻路过程，正确的路径用绿色方块表示，错误的路径用红色x表示，

且在迷宫下方有此时的路径长度。



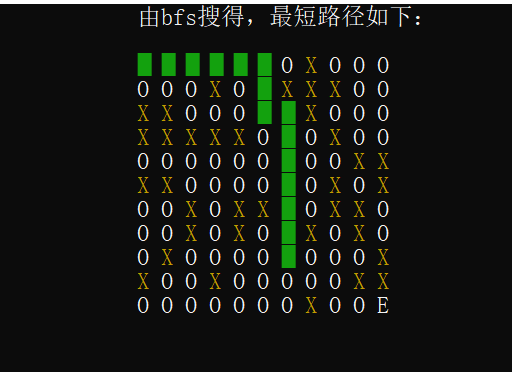


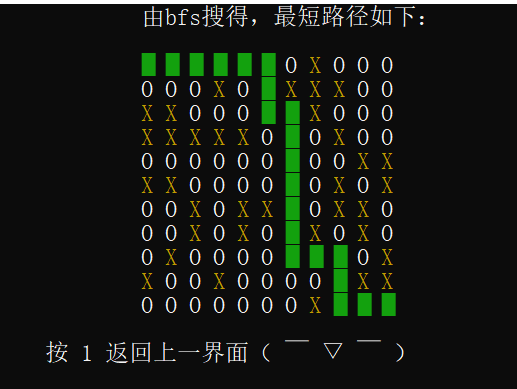
探寻到终点，停留3秒后输出最终结果：



1. 选择bfs迷宫寻路

程序会打印出最短路径探寻过程，最短路径用绿色方块表示。





**DAY5:**

**课设总结报告:**

在压缩包中(｀・ω・´)