**数据结构上机实习报告**

**第四次**

**院系：机械与电子信息学院**

**姓名：洪世杰 李业敏**

**班号：071183**

目录

[一、 题目 3](#_Toc55757475)

[二、问题描述 3](#_Toc55757476)

[三、基本要求 3](#_Toc55757477)

[四、测试数据 3](#_Toc55757478)

[五、算法思想 4](#_Toc55757479)

[六、模块划分 4](#_Toc55757480)

[七、数据结构 4](#_Toc55757481)

[八、测试情况 5](#_Toc55757482)

[九、补充说明 5](#_Toc55757483)

[1、若不需要输出最短路径 5](#_Toc55757484)

[2、不使用堆栈，仅采用回溯算法 6](#_Toc55757485)

[十、源代码 7](#_Toc55757486)

[1、堆栈实现并输出最短路径 7](#_Toc55757487)

[2、堆栈实现，输出任意一条路径 13](#_Toc55757488)

[3、不使用堆栈 17](#_Toc55757489)

# 一、 题目

迷宫问题

# 二、问题描述

要求机器人导航迷宫。它被放置在迷宫中的某个位置（起始位置），并被要求尝试到达另一个位置（目标位置）。迷宫中的置将被打开或被障碍物挡住。 位置由（x，y）坐标标识。在任何给定时刻，机器人只能在4个方向中的一个方向上移动1步。 有效的举措是：  
 北：（x，y） - >（x，y-1）  
 东：（x，y） - >（x + 1，y）  
 南：（x，y） - >（x，y + 1）  
 西：（x，y） - >（x-1，y）  
 注意，位置是在从零开始的坐标中指定的（即0,1,...size-1，其中size是相应维度中迷宫的大小）。  
 机器人只能移动到没有障碍物的位置，并且必须留在迷宫内。  
 机器人应该搜索从起始位置到目标位置（解决方案路径）的路径，直到找到一个或直到它耗尽所有可能性。  
 此外，它应标记它在迷宫中找到的路径（如果有的话）。  
 为了使这个问题具体，让我们考虑一个由字符矩阵表示的迷宫。

# 三、基本要求

1、使用堆栈实现迷宫中寻找路径算法。

2、提供测试代码，给出测试结果。

# 四、测试数据

测试如下矩阵模拟的迷宫，其中‘1’代表墙，‘0’代表通路  
 {0, 0, 1, 1, 1, 1, 1, 1, 1, 1},  
 {1, 0, 0, 0, 1, 0, 1, 0, 0, 1},  
 {1, 0, 0, 0, 1, 0, 1, 0, 1, 1},  
 {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},  
 {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},  
 {1, 1, 1, 0, 0, 0, 0, 0, 0, 1},  
 {1, 0, 1, 0, 0, 1, 1, 1, 1, 1},  
 {1, 0, 1, 0, 0, 0, 1, 0, 1, 1},  
 {1, 0, 1, 0, 0, 0, 0, 0, 0, 1},  
 {1, 1, 1, 1, 1, 1, 1, 1, 0, 0}

# 五、算法思想

1、void setcolor(int color)；

该函数是用来控制输出字体颜色。在printf（）语句前使用setcolor(i)可以控制输出不同的颜色。

2、void printfroute()；

该函数是用来输出迷宫对应的路径。其思想是在程序中将走到的路径置为-1，此时二维数组里面有三个数：0代表为未走的路，1代表墙，-1代表走过的路。此时根据数的不同对应输出不同的字符，并加上颜色，以达到使路径清晰明了展示的目的。

3、void findroute(int x1, int y1, int x2, int y2)；

该函数是用来查找迷宫路径。其思想是建立两个不同的堆栈，将当前寻找路径的数据存储在一个栈中，将找到的有效路径保存到另一个栈path[]中，并定义变量minlen来保存最短路径长度。当 while循环结束的时候输出path[]中的路径。

用C语言的思路来求解迷宫问题（从（1,1）到（M，N）的路径）算法：先将入口位置的方块进栈（初始方块的位置设置为-1），在栈不空的时候进循环：读栈顶方块（不退栈），如果该方块是出口，则输出栈中的方块位置即为路径；否则就在相邻的方块中寻找下一个可走的方块，若不存在可走的方块，则退栈（即向后退一步）；若存在这样的方块，则将其位置存在栈中，并将这个可走的相邻方块入栈（并将新方块的初始位置设置为1）。 为了保证试探的可走的相邻方块不是已经走过的方块，我们可以在一个方块入栈后将其对应的迷宫数组（maze矩阵）元素改为-1（变为不可走的方块），在退栈时（当没有可以走的方块的时候）再将其恢复为0。例如方块（i, j）已经入栈，在试探（i +1 ,j）的时候，又会试探到（i, j），这样可能会引起死循环，采用上述方法之后就会避免这种情况发生。

当所有路径全部查找完毕后，对第二个堆栈进行出栈，即将最短路径输出；若第二个堆栈里面没有数据元素，则证明迷宫无通路。

4、int main()；

该函数时是测试函数。运行该函数时，需要用户对应输出迷宫的起点坐标和终点坐标。

# 六、模块划分

1、main.c文件。包括以下函数：

void setcolor(int color)；该函数是用来控制输出字体颜色。

void printfroute()；该函数是用来输出迷宫对应的路径。

void findroute(int x1, int y1, int x2, int y2)；该函数是用来查找迷宫路径。

# 七、数据结构

struct

{

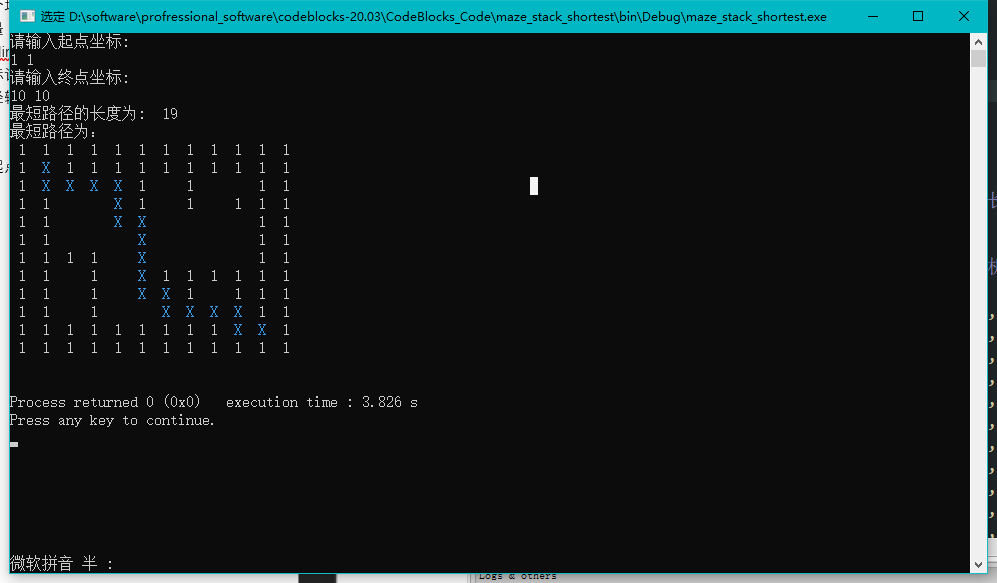
int i; //当前行

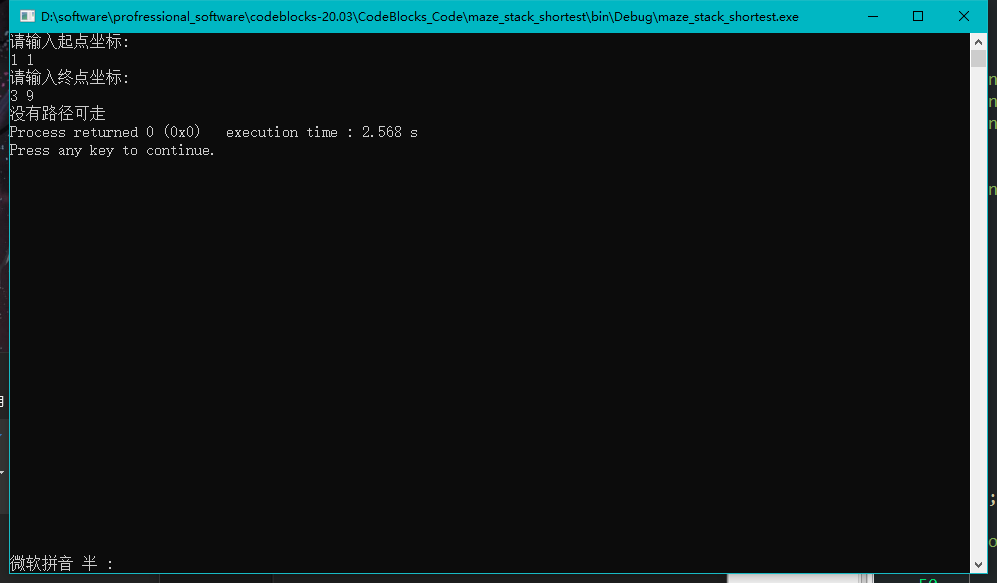
int j; //当前列

int dir; //下一步方向

} st[1000], path[1000]; //定义栈

# 八、测试情况

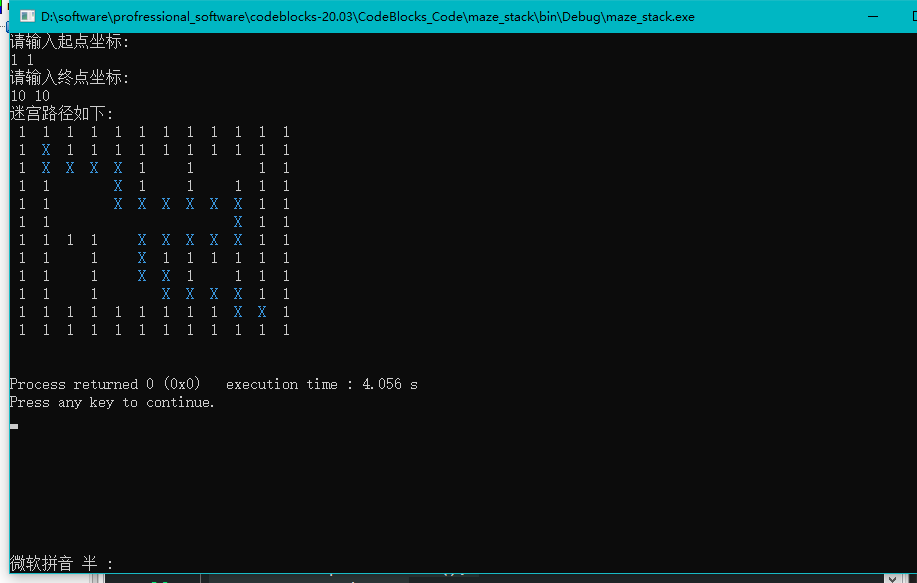




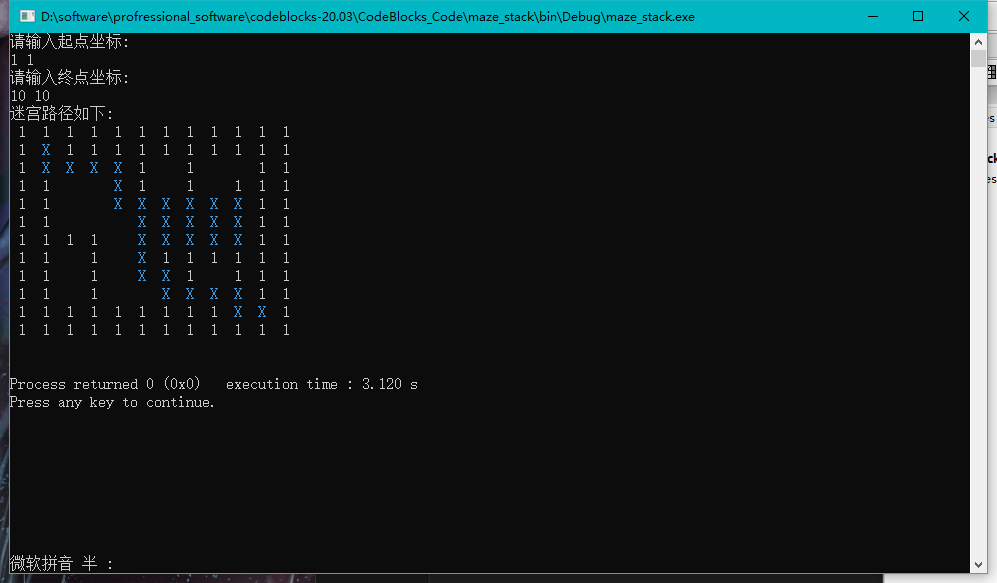
# 九、补充说明

## 1、若不需要输出最短路径

与本程序的实现相比，若不需要输出最短路径，则不需要建立第二个堆栈。寻找路径的算法与本程序相同，但是只需要找到一条路径，将其对应输出就可以。该思路的输出路径会随着程序判断方向的先后顺序不同而不同，其实现效果如下。

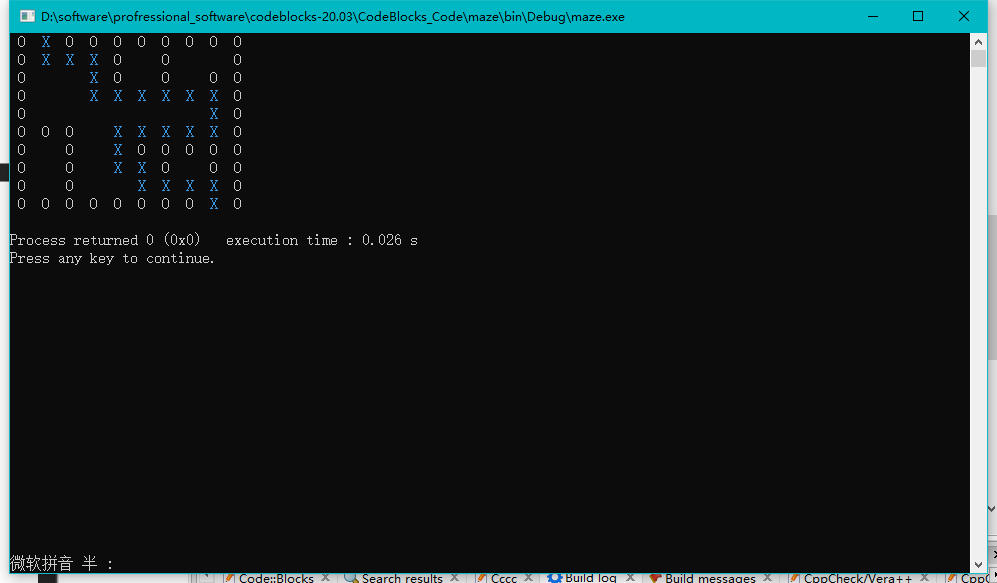
（1）方向判断顺序为右下左上

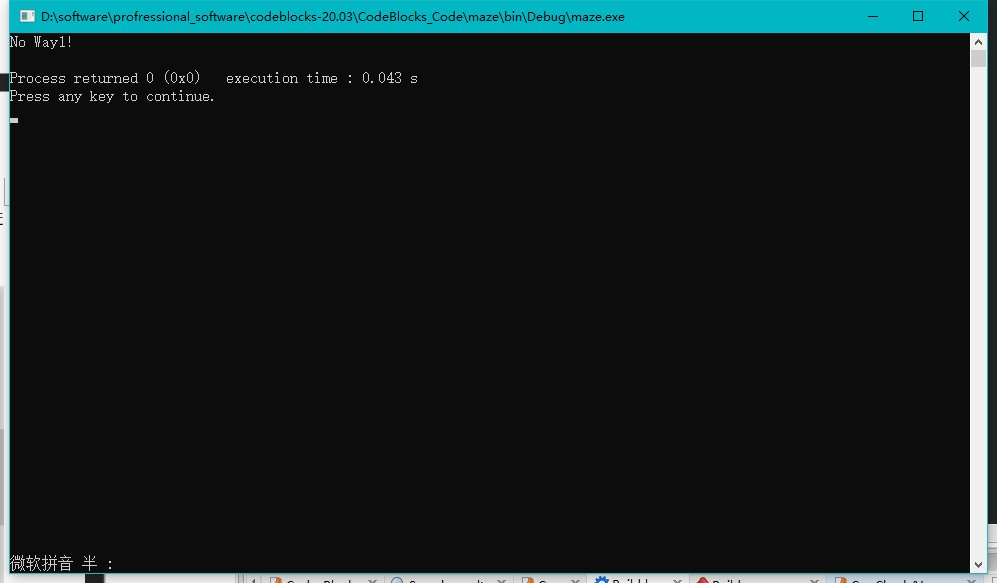
（2）方向判断顺序为右下上左



## 2、不使用堆栈，仅采用回溯算法

与输出任意一条路径的程序相比，该程序不需要建立堆栈，仅使用回溯算法思想，在程序的运行过程中将数组元素进行重新赋值（墙、未经过的坐标、经过的坐标），最终将路径进行输出。该程序的输出路径同样会随着方向判断先后的不同而不同。当方向判断顺序为右下左上时，输出如下：





# 十、源代码

## 1、堆栈实现并输出最短路径

#include<stdio.h>

#include <windows.h>

#define M 10 //10行

#define N 10 //10列

//控制颜色

void setcolor(int color)

{

HANDLE

hc=GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hc,0|color);

}

struct

{

int i; //当前行

int j; //当前列

int dir; //下一步方向

} st[1000], path[1000]; //定义栈

int top = -1; //初始化栈顶指针

int count = 1; //初始化计数器

int minlen = 1000; //初始化最短路径长度

int maze[M + 2][N + 2] = //要求：机器人只能移动到没有障碍物的位置，并且必须留在迷宫内。

{

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1},

{1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1},

{1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},

{1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},

{1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1},

{1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1},

{1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1},

{1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

};

void printfroute()

{

for(int i=0; i<12; i++)

{

for(int j=0; j<12; j++)

{

if(maze[i][j]==1)

{

setcolor(7);

printf(" 1 ");

}

if(maze[i][j]==0)

{

setcolor(7);

printf(" ");

}

if(maze[i][j]==-1)

{

setcolor(3);

printf(" X ");

}

}

printf("\n");

}

printf("\n");

}

//0是路,1是墙

void findroute(int x1, int y1, int x2, int y2)

{

int i, j, dir, find, k;

top++; //起点位置进栈

st[top].i = x1; //行号

st[top].j = y1; //列号

st[top].dir = -1; //下一块的方位

maze[x1][y1] = -1; //起点置为-1，避免之后再走一次；当作输出的标志

while(top > -1) //栈不空时，循环继续

{

i = st[top].i;

j = st[top].j;

dir = st[top].dir;

if(i == x2 && j == y2) //说明已经找到了出口，直接输出路径

{

if(top + 1 < minlen)

{

for(k = 0; k <= top; k++)

path[k] = st[k]; //当前栈中元素依次压入临时栈中

minlen = top + 1; //栈顶指针对应

}

maze[st[top].i][st[top].j] = 0; //让该位置变为其他路径可走

top--;

i = st[top].i;

j = st[top].j;

dir = st[top].dir;

}

find = 0;

while(find == 0 && dir < 4) //寻找下一个可走的方向

{

dir++;

switch(dir)

{

case 0:

i = st[top].i - 1;

j = st[top].j;

break; //向上走

case 1:

i = st[top].i;

j = st[top].j + 1;

break; //向右走

case 2:

i = st[top].i + 1;

j = st[top].j;

break; //向下走

case 3:

i = st[top].i;

j = st[top].j - 1;

break; //向左走

}

if(maze[i][j] == 0) //如果找到了通路，就不用再进行上面的循环了

find = 1;

}

if(find == 1) //找到了下一个可走的方向

{

st[top].dir = dir; //修改之前栈顶元素中dir的值

top++; //下一个可以走的方向进栈,栈顶指针上移

st[top].i = i;

st[top].j = j;

st[top].dir = -1; //不修改之后会无法判断全部方向

maze[i][j] = -1; //将走过的路径设为-1，避免之后再走一次;当作输出的标志

}

else //没有路径就退栈（回溯）

{

maze[st[top].i][st[top].j] = 0; //让该位置变为其他路径可走

top--; //栈顶指针下移

}

}

if(minlen == 1000 && st[top].i == 0 && st[top].j == 0) //如果输出的元素均为0，则说明没有路径可走

printf("没有路径可走");

else

{

printf("最短路径的长度为: %d\n", minlen);

printf("最短路径为：\n");

for(k = 0; k < minlen; k++)

{

int m, n;

m = path[k].i;

n = path[k].j;

maze[m][n]=-1;

}

printfroute();

}

}

int main()

{

int x1, y1, x2, y2;

printf("请输入起点坐标:\n");

scanf("%d%d", &x1, &y1);

printf("请输入终点坐标:\n");

scanf("%d%d", &x2, &y2);

findroute(x1, y1, x2, y2);

return 0;

}

## 2、堆栈实现，输出任意一条路径

#include<stdio.h>

#include <windows.h>

#define M 10 //10行

#define N 10 //10列

struct

{

int i; //当前行

int j; //当前列

int dir; //下一步方向(0123-右下左上)

} st[1000]; //定义栈

int top = -1; //初始化栈顶指针

static int maze[M + 2][N + 2] = //要求：机器人只能移动到没有障碍物的位置，并且必须留在迷宫内。

{

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1},

{1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1},

{1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},

{1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},

{1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1},

{1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1},

{1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1},

{1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

};

//控制颜色

void setcolor(int color)

{

HANDLE

hc=GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hc,0|color);

}

void printfroute()

{

for(int i=0; i<12; i++)

{

for(int j=0; j<12; j++)

{

if(maze[i][j]==1)

{

setcolor(7);

printf(" 1 ");

}

if(maze[i][j]==0)

{

setcolor(7);

printf(" ");

}

if(maze[i][j]==-1)

{

setcolor(3);

printf(" X ");

}

}

printf("\n");

}

printf("\n");

}

//0是路,1是墙

void findroute(int x1, int y1, int x2, int y2)

{

int i, j, dir, find;

top++; //初始化方块进栈

st[top].i = x1; //行号

st[top].j = y1; //列号

st[top].dir = -1; //下一块的方位

maze[x1][y1] = -1; //起点置为-1,避免之后再走一次;当作输出的标志

while(top > -1) //栈不空时，循环继续

{

i = st[top].i;

j = st[top].j; //起点开始

dir = st[top].dir;

if(i == x2 && j == y2) //说明已经找到了出口，直接输出路径

{

printf("迷宫路径如下:\n");

printfroute();

return;

}

find = 0;

while(find == 0 && dir < 4) //判断下一步方向

{

dir++; //初值为-1

switch(dir)

{

case 0:

i = st[top].i;

j = st[top].j + 1;

break; //向右走

case 1:

i = st[top].i + 1;

j = st[top].j;

break; //向下走

case 2:

i = st[top].i;

j = st[top].j - 1;

break; //向左走

case 3:

i = st[top].i - 1;

j = st[top].j;

break; //向上走

}

if(maze[i][j] == 0) //如果找到了通路，就不用再进行上面的循环了

find = 1;

}

if(find == 1) //找到方向

{

st[top].dir = dir; //修改之前栈顶元素中di的值

top++; //下一步坐标进栈,栈顶指针上移

st[top].i = i;

st[top].j = j;

st[top].dir = -1; //不修改之后会无法判断全部方向

maze[i][j] = -1; //将走过的路径设为-1，避免之后再走一次;当作输出的标志

}

else //没有路径就退栈,即回溯

{

maze[st[top].i][st[top].j] = 0; //让该位置变为其他路径可走的方块

top--; //栈顶指针下移

}

}

printf("没有可走路径");

}

int main()

{

int x1, y1, x2, y2;

printf("请输入起点坐标:\n");

scanf("%d%d", &x1, &y1);

printf("请输入终点坐标:\n");

scanf("%d%d", &x2, &y2);

findroute(x1, y1, x2, y2);

return 0;

}

## 3、不使用堆栈

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#if(0)

static int maze[10][10]=

{

{1, 0, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 0, 0, 0, 1, 0, 1, 0, 0, 1},

{1, 0, 0, 0, 1, 0, 1, 0, 1, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 1, 1, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 1, 0, 0, 1, 1, 1, 1, 1},

{1, 0, 1, 0, 0, 0, 1, 0, 1, 1},

{1, 0, 1, 0, 0, 0, 0, 0, 0, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 0, 1}

};

#endif

#if(1)

static int maze[10][10]=

{

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

};

#endif

//控制颜色

void setcolor(int color)

{

HANDLE

hc=GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hc,0|color);

}

//输出线路（结果）

void printroute()

{

for(int i=0; i<10; i++)

{

for(int j=0; j<10; j++)

{

if(maze[i][j]==1)

{

setcolor(7);

printf(" O ");

}

if(maze[i][j]==0)

{

setcolor(7);

printf(" ");

}

if(maze[i][j]==2)

{

setcolor(3);

printf(" X ");

}

}

printf("\n");

}

}

//寻找线路：右下左上(右下上左)

void findroute(int i,int j)

{

if(i==10&&j==8)//找到终点

{

printroute();

}

else if(maze[1][0]==1&&maze[0][1]==1)//入口无法进入

{

printf("No Way1!\n");

}

else

{

if(maze[i][j+1]!=1&&maze[i][j+1]!=2)//右

{

maze[i][j]=2;//将2作为线路的标志

j++;

findroute(i,j);//递归

j--;//回溯

maze[i][j]=0;

}

else if(maze[i+1][j]!=1&&maze[i+1][j]!=2)//下

{

maze[i][j]=2;

i++;

findroute(i,j);

i--;

maze[i][j]=0;

}

else if(maze[i][j-1]!=1&&maze[i][j-1]!=2)//左

{

maze[i][j]=2;

j--;

findroute(i,j);

j++;

maze[i][j]=0;

}

else if(maze[i-1][j]!=1&&maze[i-1][j]!=2)//上

{

maze[i][j]=2;

i--;

findroute(i,j);

i++;

maze[i][j]=0;

}

else

printf("No Way2!\n");

}

}

int main()

{

findroute(0,1);

}