**实验4：迷宫最短路径求解（4学时）**

**[问题描述]**

求迷宫中从入口到出口的所有路径中最短路径问题。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  | 0 |  |  |  |  |
|  |  | 0 |  | 0 |  |  |  |  |  |
|  | 0 |  | 0 | 0 | 0 | 0 | 0 |  |  |
|  | 0 |  |  |  | 0 |  |  |  |  |
|  |  | 0 | 0 |  |  | 0 | 0 | 0 |  |
|  | 0 |  |  | 0 | 0 |  |  | 0 |  |
|  |  |  |  |  |  |  |  |  |  |

**[实验目的]**

（1） 深入理解队列的数据结构和基本操作。

（2） 掌握队列的应用方法。

**[实验内容及要求]**

1. 输入迷宫的分布（可自行定义，随机生成，不必如右图），通过队列的基本操作求取最短路径。
2. 迷宫内可探测堂方向为8，探测顺序以水平开始，顺时针方向。。
3. 输出最短路径的长度以及路径。

**[测试数据]**

/\*

 \* @Author: Weidows

 \* @Date: 2020-11-27 11:33:20

 \* @LastEditors: Weidows

 \* @LastEditTime: 2020-12-31 19:36:40

 \* @FilePath: \Weidows\C++\homework\ing\大二上\实验四\1.cpp

 \* @Description:迷宫最短路径

 \*/

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h>

**#define** STACK\_INIT\_SIZE 20 //存储空间初始分配量

**#define** STACKINCREMENT 10  //存储空间配增量

**typedef** ***struct***

{

***int*** i, j;

} *PosType*; //通道块在迷宫中的坐标位置

**typedef** ***struct***

{

***int*** ord;      //通道在路径上的序号

*PosType* seat; //通道块在迷宫中的坐标位置

***int*** di;       //从此通道块走向下一通道块的方向

} *SElemType*;    //栈的元素类型

**typedef** ***struct***

{

*SElemType* **\***base; //在栈构造之前和销毁之后，base值为NULL

*SElemType* **\***top;  //栈顶指针

***int*** stacksize;   //当前已分配的存储空间，以元素为单位

} *SqStack*;         //栈

*SqStack* InitStack(*SqStack* *S*)

{ //创建栈

*S*.base **=** (*SElemType* **\***)malloc(STACK\_INIT\_SIZE **\*** (**sizeof**(*SElemType*)));

**if** (**!***S*.base) //存储空间分配失败

    exit(0);

*S*.top **=** *S*.base;

*S*.stacksize **=** STACK\_INIT\_SIZE;

**return** *S*;

} //InitStack

//若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR

***int*** IsEmpty(*SqStack* *S*)

{

**if** (*S*.top **==** *S*.base) //如果空栈，报错

**return** 1;

**return** 0;

}

***void*** DestroyStack(*SqStack* *S*)

{ //销毁栈

**if** (**!**IsEmpty(*S*))

    free(*S*.base);

}

*SqStack* Push(*SqStack* *S*, *SElemType* *e*)

{ //插入元素e为新的栈顶元素

**if** (*S*.top **-** *S*.base **>=** *S*.stacksize)

  { //栈满，追加存储空间

*S*.base **=** (*SElemType* **\***)realloc(*S*.base, (*S*.stacksize **+** STACKINCREMENT) **\*** **sizeof**(*SElemType*));

**if** (**!***S*.base)

      exit(0);

*S*.top **=** *S*.base **+** *S*.stacksize;

*S*.stacksize **+=** STACKINCREMENT;

  } //if

**\***(*S*.top) = *e*;

*S*.top**++**;

**return** *S*;

} //Push

*SqStack* Pop(*SqStack* *S*)

{ //出栈

  //考虑到返回栈顶元素有专门的函数，所以该函数返回栈S，并非真正意义上的出栈函数，仅仅修改栈顶指针

**if** (*S*.top **==** *S*.base)

    exit(0);

*S*.top**--**;

**return** *S*;

} //Pop

*SElemType* GetTop(*SqStack* *S*)

{ //返回栈顶元素

**if** (*S*.top **==** *S*.base)

    exit(0);

**return** **\***(*S*.top **-** 1);

} //GetTop

//创建迷宫，随机设置障碍物

***void*** Create\_Maze(***int*** \*\**maze*, ***int*** *line*, ***int*** *row*)

{

***int*** Fi, Fj; //用于随机产生有障碍的位置

***int*** i, j;

**for** (i **=** 0; i **<** *line*; i**++**)

**for** (j **=** 0; j **<** *row*; j**++**)

**if** (i **==** 0 **||** j **==** 0 **||** i **==** *line* **-** 1 **||** j **==** *row* **-** 1) //围墙

*maze*[i][j] **=** 0;

**else**

*maze*[i][j] **=** 1; //可通为1，有障碍为0

  /\*随机产生障碍物\*/

  srand((***unsigned***)time(NULL));

**for** (i **=** (*line* **+** *row*) **\*** 2; i **>** 0; i**--**)

  { //随机生成line+row次

    Fi **=** rand() **%** *line*;

    Fj **=** rand() **%** *row*;

*maze*[Fi][Fj] **=** 0;

  } //for

} //Create\_Maze

***void*** Print\_Maze(***int*** \*\**maze*, ***int*** *line*, ***int*** *row*)

{ //打印迷宫

***int*** i, j;

  //列坐标提示

  printf("  ");

**for** (i **=** 0; i **<** *row*; i**++**)

    printf("%-2d", i);

  printf("\n");

**for** (i **=** 0; i **<** *line*; i**++**)

  {

    printf("%-2d", i); //行坐标提示

**for** (j **=** 0; j **<** *row*; j**++**)

    {

**switch** (*maze*[i][j])

      {

**case** 0:

        printf("\* ");

**break**; //0是障碍物

**case** 3:  //3是指走过，但不通的地方

**case** 1:

        printf("  ");

**break**; //1是通的地方

**case** 2:

        printf("+ ");

**break**; //2是通路

**case** 4:

        printf("S ");

**break**; //4是起点

**case** 5:

        printf("E ");

**break**; //5是终点

      }        //switch

    }          //for

    printf("\n");

  } //for

} //Print\_Maze

*PosType* NextPos(*PosType* *pos*, ***int*** *di*)

{ //下一个要走的位置

**switch** (*di*)

  {

**case** 1:

*pos*.j**++**;

**break**;

**case** 2:

*pos*.i**++**;

**break**;

**case** 3:

*pos*.j**--**;

**break**;

**case** 4:

*pos*.i**--**;

**break**;

  } //switch

**return** *pos*;

} //NextPos

***void*** FindOut(***int*** \*\**maze*, *PosType* *Start*, *PosType* *End*)

{ //寻找出路

*SqStack* S;

  S = InitStack(S);

*PosType* curpos; //当前位置

  curpos.i **=** *Start*.i;

  curpos.j **=** *Start*.j;

*SElemType* e; //当前通道块

***int*** curstep **=** 1; //探索第一步；

**do**

  {

**if** (*maze*[curpos.i][curpos.j] **==** 1)

    {

*maze*[curpos.i][curpos.j] **=** 2; //表示留下足迹

      e.di **=** 1;

      e.seat.i **=** curpos.i;

      e.seat.j **=** curpos.j;

      e.ord **=** curstep;

      S = Push(S, e); //加入路径

**if** (curpos.i **==** *End*.i **&&** curpos.j **==** *End*.j) //到达终点

**break**;

      curpos = NextPos(curpos, 1);

      curstep**++**; //探索下一步

    }            //if

**else**

    {

**if** (**!**IsEmpty(S))

      {

        e = GetTop(S); //由于Pop的不完善，所以这两步加起来才是真正的Pop功能，下同

        S = Pop(S);

        curstep**--**;

**while** (e.di **==** 4 **&&** **!**IsEmpty(S))

        {

*maze*[e.seat.i][e.seat.j] **=** 3; //3表示不能通过的标记

          e = GetTop(S);

          S = Pop(S);

          curstep**--**;

        }

**if** (e.di **<** 4)

        {

          e.di**++**;

          S = Push(S, e);

          curstep**++**;

          curpos = NextPos(e.seat, e.di);

        } //if

      }   //if

    }     //else

  } **while** (**!**IsEmpty(S));

**if** (IsEmpty(S))

  {

*maze*[*Start*.i][*Start*.j] **=** 1; //因为无论如何起点都会留下足迹

    printf("无解\n");

  } //if

**else**

  { //打印路径

*PosType* Path[curstep];

***int*** i;

**while** (**!**IsEmpty(S))

    {

      e = GetTop(S);

      S = Pop(S);

      Path[e.ord **-** 1].i **=** e.seat.i; //ord是从1开始计算的，而数组是从0开始

      Path[e.ord **-** 1].j **=** e.seat.j;

    }                                 //while

**for** (i **=** 0; i **<** curstep **-** 1; i**++**) //最后一个单独打印

      printf("(%d,%d) -> ", Path[i].i, Path[i].j);

    printf("(%d,%d)\n", Path[i].i, Path[i].j);

  } //else

  DestroyStack(S);

} //FindOut

***int*** main()

{

***int*** Si, Sj, Ei, Ej, i; //分别是起点终点坐标，和计数器

***int*** **\*\***maze;            //迷宫二维数组

***int*** LSize, RSize;      //迷宫大小，可以自定义行和列

*SqStack* S;

  //自定义迷宫大小

  printf("请输入迷宫大小(行数 列数):");

  scanf("%d %d", **&**LSize, **&**RSize);

  //动态分配二维数组内存

**do**

  { //确保开辟成功

    maze **=** (***int*** **\*\***)malloc(**sizeof**(***int*** **\***) **\*** LSize);

  } **while** (maze **==** NULL);

**for** (i **=** 0; i **<** LSize; i**++**)

  {

**do**

    {

**\***(maze **+** i) **=** (***int*** **\***)malloc(**sizeof**(***int***) **\*** RSize);

    } **while** (maze **+** i **==** NULL);

  }

  //创建并打印迷宫

  Create\_Maze(maze, LSize, RSize);

  Print\_Maze(maze, LSize, RSize);

  //输入起点、终点

**while** (Si **>** LSize **-** 1 **||** Sj **>** RSize **-** 1 **||** maze[Si][Sj] **==** 0)

  { //最后一项表达式不能放到前面，不然会越界

    printf("输入起点:");

    scanf("%d %d", **&**Si, **&**Sj);

  }

**while** (Ei **>** LSize **-** 1 **||** Ej **>** RSize **-** 1 **||** maze[Ei][Ej] **==** 0)

  {

    printf("输入终点:");

    scanf("%d %d", **&**Ei, **&**Ej);

  }

  //创建起点终点

*PosType* Start **=** {Si, Sj};

*PosType* End **=** {Ei, Ej};

  FindOut(maze, Start, End); //路径

  //起点终点标记符

  maze[Start.i][Start.j] **=** 4;

  maze[End.i][End.j] **=** 5;

  Print\_Maze(maze, LSize, RSize);

  free(maze); //释放内存

**return** 0;

} // main

