

算法与数据结构设计报告

（ 2020 / 2021学年 第 一 学期）

题 目： 无向图遍历问题

迷宫问题

|  |  |
| --- | --- |
| **专 业** | **计算机科学与技术** |
| **学 生 姓 名** | **梁佩欣** |
| **班 级 学 号** | **B18030409** |
| **指 导 教 师** | **徐小龙** |
| **指 导 单 位** | **计算机科学与技术系** |
| **日 期** | **2020.10.26-2020.11.08** |

|  |  |  |
| --- | --- | --- |
| **支撑指标点** | **评价准则** | **计分（每项10分）** |
| 课程目标1：文献调研与资料收集能力，问题发现、研究、分析与解决能力（20分） | 1、能够掌握算法与数据结构设计的相关基础知识，并能够针对求解的工程问题，收集资料进行合理的分析与设计 |  |
| 2、通过调研，能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行编程实现 |  |
| 课程目标2：通过课程设计，培养学生综合应用算法和数据结构等知识解决工程问题的实践能力（20分） | 3、能够给出数据结构和算法的设计描述，给出关键算法的流程图或伪代码，并给出各算法之间的结构关系描述 |  |
| 4、具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便 |  |
| 课程目标3：培养解决工程问题的开发工具运用能力，能够利用程序设计软件或系统对问题求解进行模拟和实现，能够设计测试数据验证问题解决方法的正确性，并能够对问题解决方法的性能和效率进行分析（40分） | 5、具备一定的算法与数据结构设计分析能力，能够完成课题要求的各项任务和指标 |  |
| 6、能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行有效建模和求解 |  |
| 7、具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能 |  |
| 8、掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试 |  |
| 课程目标4：选择同类课题的学生能够通过讨论和交流解决课程设计中的难题，能在实验报告中准确阐述课程设计的内容，能够清晰陈述观点和回答问题（20分） | 9、能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容，以及相关的工程基础知识具有较好的理解和掌握 |  |
| 10、具备一定的语言表达能力与文字处理能力，能够结合复杂工程问题撰写报告，报告内容和实验数据详实，格式规范 |  |
| 算法与数据结构设计能力测评总分 | |  |
| **指导教师： 年 月 日** | | |
| **备注：** | | |

**无向图遍历问题**

**一、课题内容和要求**

课题内容：给定无向图G，输入起始端点和结束端点，利用深度优先搜索或者广度优先搜索给出所有路径。

**二、数据结构说明**

遍历的结点用堆栈顺序存储：

struct StackNode

{

int data;//指的是栈顶元素的下标

StackNode \*next;//指向下一个边结点

};

**三、算法设计**

寻找路径的算法：

基于深度优先搜索算法：先把一个数组里的元素表示开始顶点的元素置1，然后比较输入的两个顶点，若两个顶点都相同那么调用输出函数，建立一个for循环，比较领接矩阵中的值与建立的数组的值比较有没有遍历过，如果没有遍历过就利用回溯算法进行遍历，最后要把数组的元素值置0，表示可以进行第二次路径遍历。算法流程图如下图所示。

**四、详细设计**

void push(StackNode \*&S,int e)

{

StackNode \*p=new StackNode;

p->data=e;//下一条边

p->next=S;//下一个结点

S=p;

}

int pop(StackNode \*&S)

{

int e=S->data;

StackNode \*p=S;

S=S->next;

delete p;

return e;

}

//邻接矩阵

void creat\_mg(Mgraph G) {

int i,j,k;

//输入图的相关信息

cout<<"请输入图的顶点数："<<endl;

cin>>n;

cout<<"请输入图的边数："<<endl;

cin>>e;

//将图中的所有值赋值为0，即任意两个点都是不相关的

//初始化 领接矩阵

//空出G[][0],便于记录

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

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

G[i][j] = 0;

}

//赋值

for(k = 1; k <= e; k++) {

cout<<"请输入之间有路径的两个顶点vi , vj"<<endl;

cin>>i>>j;

G[i][j] = 1;

G[j][i] = 1;

}

}

//输出邻接矩阵

void out\_mg(Mgraph G) {

int i ,j ,k;

//输出矩阵

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

cout<<endl;

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

//printf("%5d",G[i][j]);

cout<<G[i][j]<<" ";

}

//输出所在边

cout<<endl;

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

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

if(G[i][j] != 0)

cout<<"存在边<"<<i<<","<<j<<">"<<endl;

}

//深度优先搜索算法

void DFS(StackNode \*&S,int v,int r ,Mgraph G)

{

visited[v]=1;

push(S,v);

if(v==r)

{

show(S);

cout<<endl;

}

for(int w=0;w<=n;w++)//这里没有不是小于等于的话会显示不出最后一个顶点

if(G[v][w] && (!visited[w]))

DFS(S,w,r,G);

visited[v]=0; //回溯

pop(S);

}

**五、测试数据及其结果分析**

测试数据：

1. 有路径可走时：

输入：无向图的四个点1、2、3、4，点与点之间的边：<1,2>,<1,3>,<2,3>,<2,4>

其无向图如下：

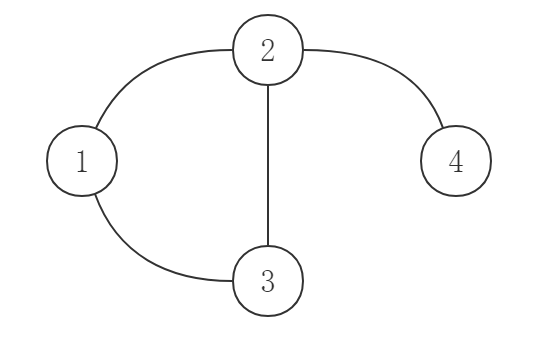


图1 测试无向图1

求无向图1到4的路径

理想输出：

路径1：

1­›2­›4

路径2：

1­›3­›2­›4

实际输出：

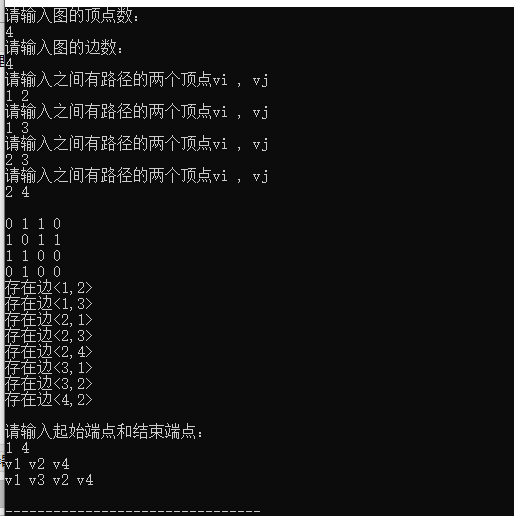


图2 无向图测试结果1

1. 无路径可走时：

输入：5个顶点：1、2、3、4、5，顶点之间的边：<1,2>,<1,3>,<2,4>

无向图如下：

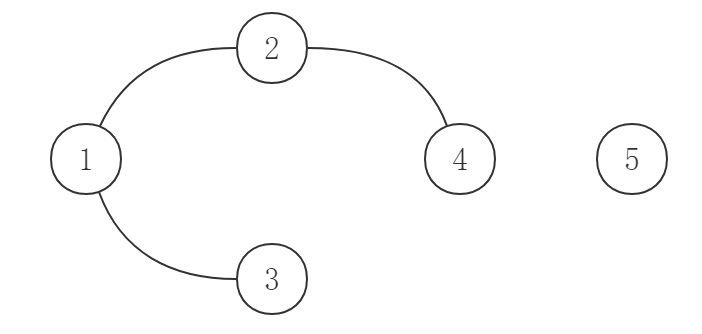


图3 测试无向图2

求顶点2到5之间的路径

理想输出：

无路可走

实际输出：

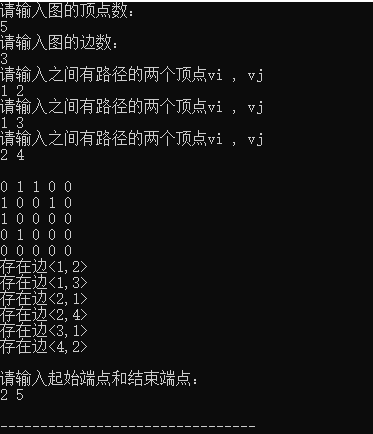


图4 无向图测试结果2

**六、算法设计和程序调试过程中的问题**

**问题1**：在寻找路径时，寻找的顶点要是最后一个点的话无法输出

**解决方法**：在深度优先搜索的函数中for循环的一个函数要是只是小于顶点个数的话，寻找路径时是找不到最后一个顶点的。For循环的判断函数要是小于等于顶点个数

**问题2：**在随机二维数组时怎么调用随机生成的二维数组

**解决方法：**将数组一开始就定义成全局变量，然后下面的随机数组函数就会把数组随机化，下一个函数可以直接调用，不需要随机数组函数返回数组。

# 迷宫问题

1. **课题内容和要求**
2. 课题内容

以一个m\*n的长方阵表示迷宫，0和1分别表示迷宫的通路和障碍。设计一个程序，对任意设定的迷宫，求出从入口到出口的通路，或得出没有通过的结论。

1. 课题要求
2. 基本要求
3. 随机生成m\*n的长方阵迷宫，m、n>8
4. 任意输入入口坐标点和出口坐标点，输出通行过程坐标点，或者不存在通信路径。
5. 可以使用二维数组表示，也可以使用图形化节点表示。
6. 其他要求
7. 变量、函数命名符合规范。
8. 注释详细；每个变量都要求有注释说明用途；函数有注释说明功能，对参数。返回值也要以注释的形式说明用途；关键的语句段要求有注释解释。
9. 程序的层次清晰，可读性强。
10. 界面美观，交互方便。
11. **数据结构说明**

1.随机二维数组的定义：

 var R = 13,//行列的值

        C = 13；

      var M = [//构建二维随机数组

        [0, 0],

        [0, 0],

      ];

 //定义与迷宫相同的二维数组，表示这条路有没有走，刚开始定义定义没走

      var t = new Array();

      for (var i = 0; i < R + 2; i++) {

        t[i] = new Array();

        for (var j = 0; j < C + 2; j++) {

          t[i][j] = 0;

        }

      }

//用于存放路径的栈

var stack=[

[0,0],

[0,0],

];

//var stack = new Array();

for (var i = 0; i < 100; i++) {

stack[i] = new Array();

for (var j = 0; j < 2; j++) {

stack[i][j]=0;

}

}

1. **算法设计**
2. 随机迷宫算法：先定义二维数组并利用随机函数Math.random给他们随机赋值0或者1.
3. 迷宫寻路算法：先判断点击的迷宫的入口和出口是否相同，如果相同输出路径，不相同进入循环，循环看迷宫走东南西北四个方向的哪一个，再判断这四个方向是否可以走，如果可以的话标记走过的这个点，然后回溯这个算法，继续下一个点，同时释放之前走过的路径的点的标记，直到最后的坐标与出口相同。
4. **详细设计**

//随机迷宫

function random() {

var i = 0,

j = 0;

for (i = 0; i < R + 2; i++) {

M[i] = new Array();

for (j = 0; j < C + 2; j++) {

M[i][j] = Math.round(Math.random());

if (i == 0 || j == 0 || i == R + 1 || j == C + 2) {

M[i][j] = 1;

}

}

}

}

//将数组变成页面的方框

function create() {

document.querySelector(".M").innerHTML = "";

for (var i = 0; i < M.length; i++) {

if (M[i].length > 0) {

for (var z = 0; z < M[i].length; z++) {

if (M[i][z] == 0) {

document.querySelector(

".M"

).innerHTML += `<span class="road" data-index="${i}-${z}" data-indexx="${i}" data-indexy="${z}"></span>`;

} else if (M[i][z] == 1) {

document.querySelector(".M").innerHTML +=

'<span class="wall" data-index="wall"></span>';

}

//换行

if (z == M[i].length - 1) {

document.querySelector(".M").innerHTML += "<br />";

}

}

}

}

}

//输出路径

function print() {

const info = document.querySelector(".info");

info.innerHTML = "迷宫的其中一条路径:";

for (var i = 0; i <= step; i++) {

info.innerHTML += "(" + stack[i][0]+","+stack[i][1] +")" + "->";

}

info.innerHTML += "到出口啦！";

}

//迷宫函数

function maze(x, y, m, n) {

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

//sum++;

print();

}

else

{

for (var i = 0; i < 4; i++)

{

var a ,b;

a= x - 0 + move[i][0];

b = y - 0 + move[i][1];//必须将字符串转换成数字，不然就只是直接相连接，直接-0就可以转换

if (!M[a][b] && !t[a][b])//如果都为0时,即有通路且没有遍历过

{

t[a][b] = 1;

M[a][b] = 1;

step++;

stack[step][0] = a;//?

stack[step][1] = b;

maze(a, b, m, n);

t[a][b] =0;

M[a][b] = 0;

step--;

}

}

}

}

1. **测试数据及其结果分析**

界面：点击创建迷宫之后会随机创建一个迷宫。

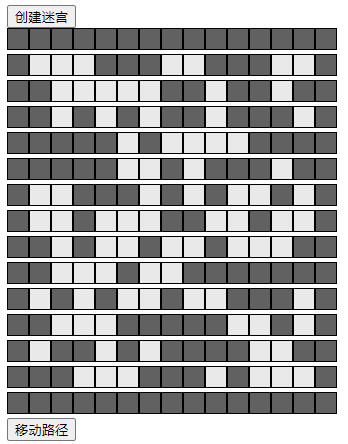


图5 迷宫界面

测试数据**：**

1.有路径可走时：

输入入口：(1,3)和出口坐标：(3,6)

理想输出：

迷宫的其中一条路径:

(1,3)->(1,2)->(2,2)->(2,3)->(2,4)->(2,5)->(2,6)->(3,6)->到出口啦！

实际输出：

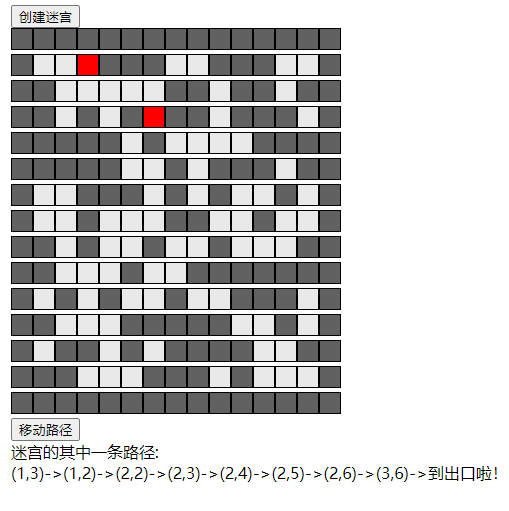


图6 迷宫测试1

2.无路可走时：

测试数据：入口坐标：（1,1）；出口坐标：（1,5）

理想输出：无路可走！

实际输出：

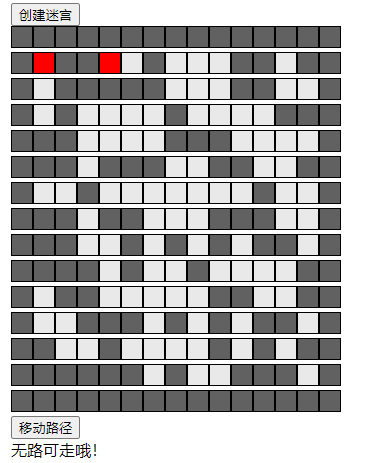


图7 迷宫测试图2

1. **算法设计和程序调试过程中的问题**

**问题1：**在建立二维数组时，用C语言的方法建立二维数组出错。

**解决方法**：在JS中建立二维数组不能用C语言的方式，只能是在一维数组中在建立，可以用以下两种方法：

法一：

var stack=[

[0,0],

[0,0],

];

法二：

var stack = new Array();

for (var i = 0; i < 100; i++) {

stack[i] = new Array();

for (var j = 0; j < 2; j++) {

stack[i][j]=0;

}

}

**问题2：**在二维数组如何在页面中生成可见的方框

**解决方法：**首先用document.querySelector(".M").innerHTML = ""取数组对象，并判断对应的数组元素的值是0还是1，表示通路或墙，并在0和1的元素处用一个标签标记，之后用CSS把这个标签实体化显示出来。

**问题3：**在生成的页面中如何点击相应的点从而获取到这个点的坐标

**解决方法：**在整一个迷宫的地图中绑定鼠标点击事件，并且会定位到你所点击的点，然后利用const value = e.target.dataset.index;可以定位到所点击的点的坐标然后直接获取坐标。

**问题4：**在页面中如何把所点击过的点变成另外一个颜色进行标记方便辨认入口和出口坐标

**解决方法：**给点击的点绑定一个事件e.target.classList.add('active');，之后再利用CSS将这个点的颜色进行更改。

**七、课程设计总结**

**算法进一步改进的设想：**

在随机迷宫的函数算法中，由于是随机给二维数组赋值，很多时候出来的迷宫图不是连通图，这样很容易造成障碍散乱。之后的算法可以改进成生成随机迷宫的时候迷宫就是一个连通图，可以试着利用普利姆算法进行操作。

**课程设计过程的收货和感受：**

在为期两周的程序设计中，我的编程能力有了很大的提高。首先在解决这两道题目时，要思考怎么实现以及他们之间的共同点，在这些思考的过程中，我对于深度优先搜索算法以及广度优先搜索算法的了解更加的深入了。在进行程序设计时遇到了很多问题，在解决这些问题的过程中，我对于一些语法知识也更加的深入了。在迷宫这道题用C语言解决了之后，我尝试用JS写，并且想写出界面，这个过程中我也更加了解了C语言和JS的一些语法上的区别。比如如何建立二维数组，这两者之间是不太一样的。还对于之前没注意的点函数不能直接返回数组有了了解。这次的程序设计周，极大地锻炼了我对于用JS编程的能力。