链滴

5 X 5 方阵引出的寻路算法 之 路径遍历（完结）

作者：someone45333
原文链接：https://ld246.com/article/1563617394912
来源网站：链滴

  此篇文章源自对一个有趣问题的思考，在我的另一篇博文《个有趣的 5 X 5 方阵一笔画问题》中有详细介绍。在已知其结论的情况下，作为程序员的我，还是想用该问题当做出发点，写一个可以遍历所有“可能路线”的寻路算法，当做学习“图”相关算法的练。如果对那个原始问题有兴趣，点击上面的文章链接，出门右转便是。

一、问题回顾

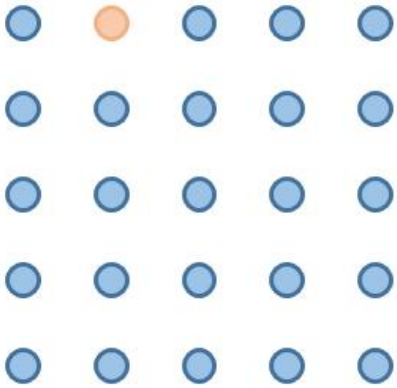
  还是要简单描述一下问题：有一个 5 X 5 的点方阵，如下图，要想用一笔画所有的蓝色点连起来，是否有可行路线。需要满足3点要求：

  1、笔画必须水平或垂直。

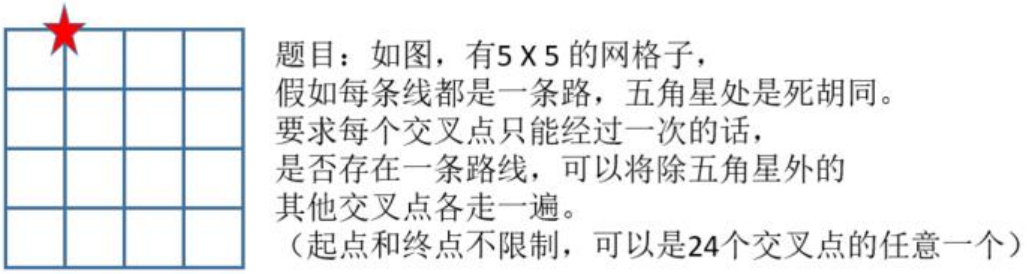
  2、笔画不可以超出方阵边界之外，

  3、每个点只经过一次，且不可以经过黄色点。

  这个问题的数学证明已经在开头的文章中给出，在此不做赘述。这里只考虑的算法实现。



  在思考算法之前，需要先对问题进行抽象分析，我们可以将上面的问题理解下图这样：



二、抽象分析

  由于题目已经被数学证明没有一条路线能够满足题目要求，所以算法的目的是为了找到所谓符合要求的路线，而是遍历从出发点向网格中所有点可形成的所有路线。可以假设上的网格是一个地图，小明同学按照地图在网格中行走，目的是找出从出发点到所有点的所有路线。小先在地图中规定坐标，网格底边作为x轴，左侧的竖边作为y轴，那么红色五角星坐标就是（1,4）。明从原点开始出发，刚开始时地图中所有的交叉点都被标记“未走过”标识。小明随机选择下一个点假如依次经过（0,0）、（1,0）、（1,1）、（2,1）、（2,2）、（3,2）、（4,2）、（4,1）、（4,0、（3,0）、（3,1），每往前走一个点，他都会在地图的相应位置标记已经走过。走到（3,1）时，由四周的（2,1）、（3,2）、（4,1）都已经走到过，所以往前是无路可走的，只能逐步回退，回退之前明为了下次不再重新走这个线路，他将这个线路记到了本子上，并标记本路线为历史路线，然后回退每回退一步小明都会在离开的那个位置将“已走过”的标记涂抹掉并改为“未走过”标识，因为这样下次从其他路线走时还可以走那个位置。当回退到（3,0）时，（3,1）位置就会由标记“已走过”改标记“未走过”的状态，并发现临近的（2,0）还没有走过，于是不再回退，转向（2,0）。到达（2,0时，又出现了无路可走的情况，于是记录当下的路线，然后回退并标记地图。当回退到（3,0）时，然会发现（3,1）是“未走过”状态，但由于如果走（3,1）的话就会发现这条路线在笔记本的历史路中已经出现过，所以依然不能走（3,1），继续回退，按照这个原则一直走下去，直到遍历完从起点发到达地图上所有点的所有路线，小明最终一定会回到起点。此时他笔记本中的历史路线列表，就是起点达地图上所有点的所有路线。

三、算法分析

  考虑到算法的实现，这显然是利用了数据结构中的“图”。每一次路线尝试都是对图的深度遍历，算法要做的事情，无非就是随机选择起点，然后不断地深度遍历新的路线。每前走一步，就将新位置放进栈。这里要注意的是，针对每一次尝试，当走到无路可走时，并不意味着2个点全部已经遍历到了，因为根据规则，无路可走的原因可能是当下点的四周位置（下面统称为“邻”）已经全部走过了，即它们已经在栈中了，或者是走到了边角的位置。那么此时无路可走该怎么办？是重新从起点开始呢，还是怎样。理论上都可行，不过考虑到效率，即让算法尽量少做重复的事情我选择在无路可走时，将当下的路线（一个由点序列组成的数组）做记录（存入一个集合中，这个集记录着所有历史路线），然后回退并将离开的位置从栈弹出，每次回退都将当下路线记录为历史路线直到周围存在“没有在栈中的点”时不再回退，而是转向新的位置。此时，最近一次回退的那个历史线决定了下次不会再走这个路线的方向。（假如连续回退了N步，那么最近一次回退的那个历史路线是临近前N-1次回退历史路线的子集）。不断以上面的方式进行下去，由于“地图”是有限的，当所可能路线全部遍历过时，最终一定会回到起点。此时历史路线集合列表，记录了从起点达地图上所有的所有路线。

四、算法描述：

  1、输入：一个二维数组M[26][5]。（第一维度角标1到4分别代表上右下左个方向，第二维度角标1到25分别代表图的每个点，一次从地图的左上到右下为1到25。数组中的每值，比如M[20][1]=15 表示地图编号为20的点的上边是编号为15的点，M[20][2]=0表示地图编号为2的点的右方没有通路。其中M[i][0]=0,M[0][j]=0）

  2、初始化：定义栈Q = {1} 和 集合H = {} .（Q中记录了当下路径中依次走的所有点，H记录着所有的历史路径。）

  3、重复以下动作，直到 Q = {}:

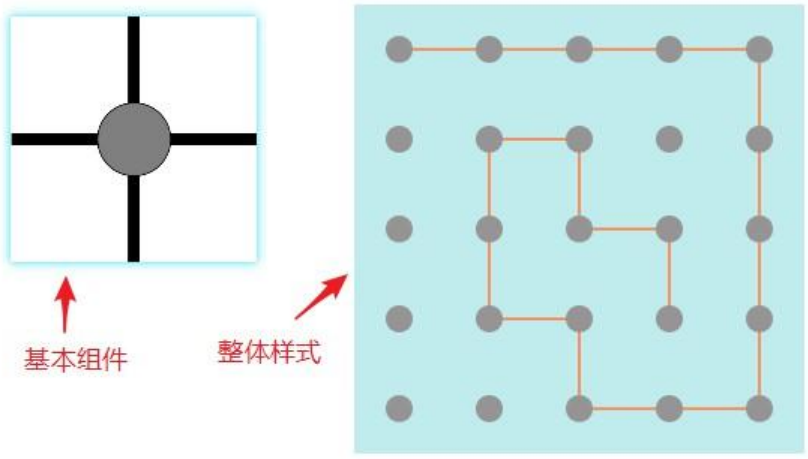
  根据Q的栈顶元素d，和邻接表M找出d的所有临近点元素，从其中随机选择个元素，且Q中不存在该元素，若找到，将该元素放入Q中。若找不到，则将Q的副本存入H中，并将从Q中弹出。

  4、输出：H。

五、算法实现

  到这里，对原始问题的抽象思考过程和算法的理论分析过程已经描述完毕。面就是用实际的代码实现该算法并做到可视化。如果不考虑可视化，那么随便一个编程语言都可以很便的实现它，我之前用java写过，将结果输出在控制台那种。不过感觉没啥意思，还是想要可视化的觉。当时很长一段时间我都不知道如何将它可视化，直到我遇到了React，深入了解后，发现用React完成该算法的可视化，真是再合适不过了。

  由于React是面向组件开发的模式，并且可以很容易根据状态来自动渲染页，所以可视化部分的设计，就变成对原始“地图”的拆解和状态的定义。我设计的最基础的组件和整样式如下图这样：



  每个位置（点）都有上下左右四个方向，点与点之间，如果存在连线，则将应方向的线条用CSS渲染成“block”，其他渲染成“none”.这个过程在定义好CSS样式后，根据数状态，React会自动帮助界面渲染，不需要反复用js写渲染逻辑。

  最终效果请点击这个 程序链接1 或程序链接2 查看。以下带部分代码。另外关于实现过程中对性能的优化，我已经做了一些努力，以后也会不断对其优化，欢有兴趣的朋友提出高见。

```
import React, { Component } from 'react';
import './AppDemo.css';
import Grid from './Grid';

class AppDemo extends Component {
    constructor(proprs) {
        super(proprs);
        var width = 5;
        var height = 5;
        var matrix = this.init(width,height);
        var x = 50;
        var start = Math.floor(Math.random() * width * height + 1);
        this.state = {
            matrix: matrix,
            start: start,
            arr: [start],
            historyPath: [],
            width: width,
            height: height,
            timeID: 0,
            speed: 5,
            random: true,
            x: x,
```

```
        time: 0
    }
}
init(width,height){
    var matrix = [[0, 1, 2, 3, 4]];
    for (var numb = 1; numb <= width * height; numb++) {
        var up = numb > width ? numb - width : 0;
        var right = (numb % width) !== 0 ? numb + 1 : 0;
        var down = numb <= width * (height - 1) ? numb + width : 0;
        var left = ((numb - 1) % width) !== 0 ? numb - 1 : 0;
        var arr = [numb];
        arr.push(up);
        arr.push(right);
        arr.push(down);
        arr.push(left);
        matrix.push(arr);
    }
    return matrix;
}
handle() {
    //var beginTime1=0;
    //var beginTime2=0;
    //beginTime1 = new Date().getTime();
    var nowRow = this.state.arr[this.state.arr.length - 1];//获取当下的位置编号
    var arr = this.state.arr;//路径编号
    var matrix = this.state.matrix;//矩阵存储结构
    var historyPath = this.state.historyPath;//历史路径
    if (arr.length > 0) {//如果路径长度>0
        var next = false;//默认找不到路径
        var ran = 1
        if (this.state.random) {
            ran = Math.floor(Math.random() * 4 + 1);
        }
        //var beginTime4 = new Date().getTime();
        for (var i = 0; i < 4; i++) {
            var nextNumb = matrix[nowRow][ran];
            if (nextNumb !== 0 && !this.containNowPath(nextNumb)) {//找到路径
                arr.push(nextNumb);//将新元素入栈
                if (!this.containHistoryPath(arr)) {//若新路径没有在历史路径中出现过，则走该路径
                    this.setState({
                        arr: arr
                    });
                    next = true;
                    break;
                } else {//若新路径在历史路径中出现过，则跳过该路径
                    arr.pop();//放弃该位置
                    ran = ran + 1 > 4 ? 1 : ran + 1;
                }
            } else {
                ran = ran + 1 > 4 ? 1 : ran + 1;
            }
        }

        //var beginTime5 = new Date().getTime();
```

```
            if (!next) {//如果无路可走
                //判断当下路径（未退步之前）是否包含于历史记录。
                if (!this.containHistoryPath(arr)) {
                    historyPath.push(arr.slice());//若没有包含与历史中，则将新的尝试路径保存进历史路
集中
                }
                arr.pop();//将最后一个元素弹出，相当于后退一步
                this.setState({//修改当前改变了的状态
                    arr: arr,
                    historyPath: historyPath
                });
            }
        } else {//若路径遍历结束，则换一个起点继续遍历。
            this.stop();
            // this.setState({
            //     start: this.state.start + 1,
            //     arr: [this.state.start + 1],
            //     historyPath: [],
            //     len: 5
            // });
        }
        // beginTime2 = new Date().getTime();
        // var time = beginTime2 - beginTime1 ;
        // if(time> this.state.time){
        //     this.setState({//修改当前改变了的状态
        //         time: time
        //     });
        // }
        //alert(time);
    }

    containNowPath(row) {//判断下一个位置是否已经存在当下路径中。
        var r = false;
        for (var i = 0; i < this.state.arr.length; i++) {
            r = this.state.arr[i] === row;
            if (r) {
                break;
            }
        }
        return r;
    }
    containHistoryPath(arr) {//从历史路径中查找是否已经存在下一步要走的路径
        var r = false;
        var historyPath = this.state.historyPath;
        for (var i = historyPath.length - 1; i >= 0; i--) {
            r = historyPath[i].toString().indexOf(arr.toString()) !== -1;
            if (r) {
                break;
            }
        }
        return r;
    }
    render() {
        return (
```

```
          <div>
            <div style={{ margin: "50px auto 0px auto", width: (this.state.width * this.state.x) + "
x", minWidth: "700px" }}>
                <div className="control">
                  <button type="button" onClick={() => this.start()}>开始</button>
                  <button type="button" onClick={() => this.stop()}>暂停</button>
                  <button type="button" onClick={() => this.start()}>继续</button>
                  <button type="button" onClick={() => this.step()}>单步</button>
                  <apan style={{ margin: "0px auto 0px 10px", width: "120px", display: "inline-bl
ck" }}>尝试次数：{this.state.historyPath.length}</apan>
                  <apan style={{ margin: "0px auto 0px 10px" }}>速度：</apan>
                  <button style={this.state.speed === 1000 ? { backgroundColor: "#61dafb" } : {}
 type="button" onClick={() => this.speed(1000)}>极慢</button>
                  <button style={this.state.speed === 500 ? { backgroundColor: "#61dafb" } : {}}
ype="button" onClick={() => this.speed(500)}>慢</button>
                  <button style={this.state.speed === 100 ? { backgroundColor: "#61dafb" } : {}}
ype="button" onClick={() => this.speed(100)}>中</button>
                  <button style={this.state.speed === 50 ? { backgroundColor: "#61dafb" } : {}} t
pe="button" onClick={() => this.speed(50)}>快</button>
                  <button style={this.state.speed === 5 ? { backgroundColor: "#61dafb" } : {}} ty
e="button" onClick={() => this.speed(5)}>极快</button>
                  <apan style={{ margin: "0px auto 0px 10px", width: "50px", display: "inline-blo
k" }}></apan>
                  <button style={this.state.random ? { backgroundColor: "#61dafb" } : {}} type="
utton" onClick={() => this.random()}>随机</button><br />
                </div>
                <div className="control2">
                  <button style={{ width: "80px" }} type="button" onClick={() => this.addheight(
)}>增加行+</button>
                  <button style={{ width: "80px" }} type="button" onClick={() => this.addwidth(1
}>增加列+</button>
                  <button style={{ width: "80px" }} type="button" onClick={() => this.big(1)}>放
+</button><br />
                  <button style={{ width: "80px" }} type="button" onClick={() => this.addheight(
1)}>减少行-</button>
                  <button style={{ width: "80px" }} type="button" onClick={() => this.addwidth(-
)}>减少列-</button>
                  <button style={{ width: "80px" }} type="button" onClick={() => this.big(-1)}>
小-</button>
                </div>
                <Grid x={this.state.x} width={this.state.width} height={this.state.height} arr={this.s
ate.arr} />

            </div>
          </div >

      )
    }
    addwidth(n) {
      this.stop();
      var width = this.state.width;
      width = width + n;
      if (width > 0 && width * this.state.x <= 1000) {
        var matrix = this.init(width,this.state.height);
```

```
            var start = Math.floor(Math.random() * width * this.state.height + 1);
            this.setState({
                matrix : matrix,
                start: start,
                arr: [start],
                historyPath: [],
                width: width
            });
        } else {
            this.setState({
                width: Math.floor(1000 / this.state.x),
            });
        }
    }
    addheight(n) {
        this.stop();
        var height = this.state.height;
        height = height + n;
        if (height > 0 && height * this.state.x <= 500) {
            var matrix = this.init(this.state.width,height);
            var start = Math.floor(Math.random() * this.state.width * height + 1);
            this.setState({
                matrix:matrix,
                start: start,
                arr: [start],
                historyPath: [],
                height: height
            });
        } else {
            this.setState({
                height: Math.floor(500 / this.state.x),
            });
        }
    }
    big(n) {
        var x = this.state.x;
        x = x + n;
        if (x > 0 && ((x * this.state.width <= 1000) || (x*this.state.height<=500)) ){
            this.setState({
                x: x
            });
        } else {
            this.setState({
                x: Math.floor(x * this.state.width > x*this.state.height?1000/this.state.width:500/this.
tate.height)
            });
        }
    }
    // shouldComponentUpdate(nextProps, nextState){
    //     return nextState.arr.length>100;
    // }
    step() {
        this.stop();
        this.handle();
```

```
}
start() {
    var timeID = this.state.timeID;
    if (timeID === 0) {
        timeID = setInterval(
            () => this.handle(),
            this.state.speed
        );
    }
    this.setState({
        timeID: timeID
    });
}
stop() {
    var timeID = this.state.timeID;
    if (timeID !== 0) {
        clearInterval(timeID);
    }
    this.setState({
        timeID: 0
    });
}
componentDidMount() {
    var matrix = [[0, 1, 2, 3, 4]];
    var width = this.state.width;
    var height = this.state.height;
    for (var numb = 1; numb <= width * height; numb++) {
        var up = numb > width ? numb - width : 0;
        var right = (numb % width) !== 0 ? numb + 1 : 0;
        var down = numb <= width * (height - 1) ? numb + width : 0;
        var left = ((numb - 1) % width) !== 0 ? numb - 1 : 0;
        var arr = [numb];
        arr.push(up);
        arr.push(right);
        arr.push(down);
        arr.push(left);
        matrix.push(arr);
    }
}
speed(n) {
    var timeID = this.state.timeID;
    clearInterval(timeID);
    timeID = 0;
    timeID = setInterval(
        () => this.handle(),
        n
    );
    this.setState({
        timeID: timeID,
        speed: n
    });
}
random() {
    this.setState({
```

```
            random: !this.state.random
        });
    }

    sleep(numberMillis) {
        var now = new Date();
        var exitTime = now.getTime() + numberMillis;
        while (true) {
            now = new Date();
            if (now.getTime() > exitTime)
                return true;
        }
    }
}

export default AppDemo;
```

原文链接：5 X 5 方阵引出的寻路算法 之 路径遍历（完结）