**人工智能和图论等相关算法总结**

自己想的例子:

例子1：给定一个地图，具有出发点和终点，其中0代表空白可以通行，1代表障碍物，寻找一个从起点到终点的最短路径，返回这条路径,运动方向上下左右,起点和终点保证不被墙所包围

要求：输入数据：一个0,1数组, int[] start,int[] end

返回信息:List<int[]> path;且是最短路径

函数原型:

List<int[]> getPath(int[][] map,int[] start,int[] end)

Dfs方法得到的代码:

**package** algorithms;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** bruteForcePath {

**boolean**[][] visited=**null**;

List<**int**[]> finalPath=**null**;

**int**[][] direction={

{-1,0},

{1,0},

{0,-1},

{0,1}

};

**public** List<**int**[]> getPath(**int**[][] map,**int** starti,**int** startj,**int** endi,**int** endj){

//first we judge the valid of start and end

**if**(map==**null**||map.length==0||map[0].length==0||map[starti][startj]==1||map[endi][endj]==1)**return** **null**;

**int** m=map.length;

**int** n=map[0].length;

visited=**new** **boolean**[m][n];

List<**int**[]> path=**new** ArrayList<>();

dfsHelper(map,path,starti,startj,endi,endj);

/\* for(int[] pos:finalPath)

System.out.println(pos[0]+","+pos[1]);\*/

**return** finalPath;

}

**private** **void** dfsHelper(**int**[][] map,List<**int**[]> path,**int** curi,**int** curj,**int** endi,**int** endj){

//if invalid position of seaching point

**if**(curi<0||curi>=map.length||curj<0||curj>=map[0].length)**return**;

//if cur point is within the map but it is visited previously and it is a wall return

**if**(visited[curi][curj]||map[curi][curj]==1)**return**;

//if path find then

**if**(curi==endi&&curj==endj){

//judge if it is the first time find the path,if yes then add it

**int**[] temp=**new** **int**[2];

temp[0]=endi;

temp[1]=endj;

path.add(temp);

System.***out***.print("path found:");

**for**(**int**[] pos:path){

System.***out***.print("("+pos[0]+","+pos[1]+") ");

}

System.***out***.println();

**if**(finalPath==**null**){

finalPath=**new** ArrayList<**int**[]>(path);

}

//if no,then judege the new length with it

**else**{

**if**(path.size()<finalPath.size()){

finalPath.clear();

finalPath=**new** ArrayList<**int**[]>(path);

}

}

**path.remove(path.size()-1);**

**return**;

}

//if it is still on the way

**int**[] coor=**new** **int**[2];

coor[0]=curi;

coor[1]=curj;

path.add(coor);

visited[curi][curj]=**true**;

**for**(**int** i=0;i<4;i++){

dfsHelper(map,path,curi+direction[i][0],curj+direction[i][1],endi,endj);

}

visited[curi][curj]=**false**;

path.remove(path.size()-1);

**return**;

}

}

测试部分:

**package** algorithms;

**public** **class** Test {

**public** **static** **void** main(String[] args){

**int**[][] map={

{0,0,0},

{0,1,0},

{0,1,0},

{0,0,0}

};

**new** bruteForcePath().getPath(map, 0, 0, 2,2);

}

}

我标红的部分，在这个测试用例里面加不加都对，但是却是必须加的，这个例子里对于终点附近的那个点来说，它只有一个方向可走，所以其他方向不会产生对path的扩充，也就是说我把终点加进去如果在退出的时候没有删除没有什么大影响，但是假如换成另一个就会产生很大的影响例如输入用例是:也就是说当你找到一条到终点路径时候，有可能回到上一个点还要从当前点继续探索路径，这时候原来的终点还在path里面，继续增加点，这样显然就不对了.整个结果都会出现问题.

**int**[][] map={

{0,0,0,0},

{0,1,0,0},

{0,1,0,0},

{0,0,0,0}

};

这时候你要是把那个注释掉就变成了:

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (3,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (3,2) (2,2) (3,3) (2,3) (1,3) (0,3) (0,2) (1,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (3,2) (2,2) (3,3) (2,3) (1,3) (0,3) (1,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (3,2) (2,2) (3,3) (2,3) (1,3) (0,3) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (0,1) (0,2) (1,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (0,1) (0,2) (1,2) (2,2) (1,3) (2,3) (3,3) (3,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (0,1) (0,2) (1,2) (2,2) (1,3) (2,3) (3,3) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (0,1) (0,2) (1,2) (2,2) (1,3) (0,3) (1,3) (2,3) (3,3) (3,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (0,1) (0,2) (1,2) (2,2) (1,3) (0,3) (1,3) (2,3) (3,3) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (0,1) (0,2) (1,2) (2,2) (1,3) (0,3) (1,3) (2,3) (3,3) (1,2) (2,2)

0,0

1,0

2,0

3,0

3,1

3,2

2,2

而正确的输出结果应该是下边这个:

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (3,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (3,2) (3,3) (2,3) (1,3) (0,3) (0,2) (1,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (3,2) (3,3) (2,3) (1,3) (1,2) (2,2)

path found:(0,0) (1,0) (2,0) (3,0) (3,1) (3,2) (3,3) (2,3) (2,2)

path found:(0,0) (0,1) (0,2) (1,2) (2,2)

path found:(0,0) (0,1) (0,2) (1,2) (1,3) (2,3) (3,3) (3,2) (2,2)

path found:(0,0) (0,1) (0,2) (1,2) (1,3) (2,3) (2,2)

path found:(0,0) (0,1) (0,2) (0,3) (1,3) (2,3) (3,3) (3,2) (2,2)

path found:(0,0) (0,1) (0,2) (0,3) (1,3) (2,3) (2,2)

path found:(0,0) (0,1) (0,2) (0,3) (1,3) (1,2) (2,2)

0,0

0,1

0,2

1,2

2,2

所以这种问题一定要求你每一步想的很清楚，要不结果一定是错的，下边再写一个bfs的思路:然后我们根据这个复杂度进行改进引出人工智能的算法.