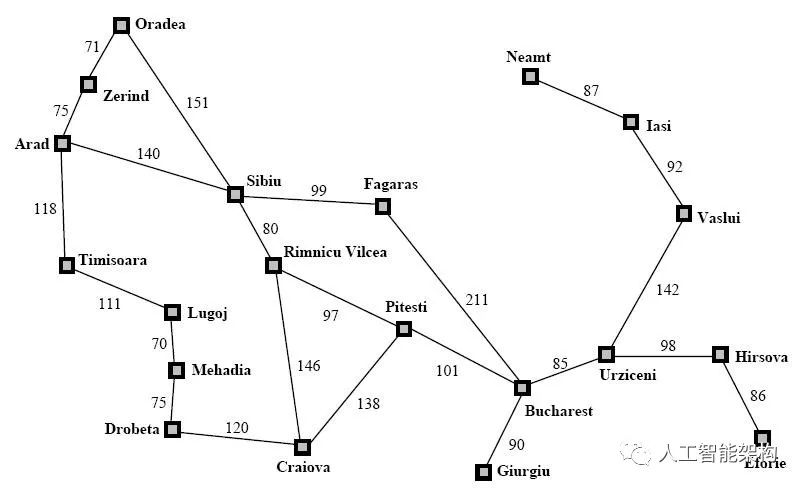
# Uniformed Search

## 算法描述

**通过使用Uniformed Search 得到从起始点Arad到目标点Bucharest的一条路径，即为罗马尼亚问题的一个解，在求解的过程中记录每种算法得到的解，即输出每种解得到的条路径。罗马尼亚地图：**



## 实验代码

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Stack;

**public** **class** aaa {

**int** result;

**int** xiabiao[]=**null**;//访问的下标

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

// **TODO** Auto-generated method stub

Graph graph=**new** Graph();

@SuppressWarnings("unused")

UCS ucs=**new** UCS(graph,0,12);

}

@SuppressWarnings("rawtypes")

**public** **void** show(Graph g,Stack stack){

**if**(stack.size()==0){

System.***out***.println("路径搜索失败");

}

result=0;

System.***out***.print("访问的下标： ");

**for**(**int** i =0; i < stack.size(); i++){

System.***out***.print("-->"+stack.get(i));

}

System.***out***.print("\n访问过程： ");

xiabiao=**new** **int**[stack.size()];

**if**(stack.isEmpty()){

System.***out***.println("搜索失败");

}

**else**

{

**for**(**int** i =0; i < stack.size(); i++){

System.***out***.print("-->"+g.cities[(Integer) stack.get(i)]);

}

**for**(**int** i =0; i < stack.size()-1; i++){

result+=g.path[(Integer) stack.get(i)][(Integer) stack.get(i+1)];

System.***out***.println("\n总长度为："+result+"\n");

g.markInit();//清空访问

}

}

}

**public** **static** **class** Graph{

**public** **int** path[][]=**new** **int**[][]{

{0,75,10000,118,140,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000},

{75,0,71,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000},

{10000,71,0,10000,151,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000},

{118,10000,10000,0,10000,111,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000},

{140,10000,151,10000,0,10000,80,99,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,111,10000,0,10000,10000,70,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,80,10000,0,10000,10000,10000,146,97,10000,10000,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,99,10000,10000,0,10000,10000,10000,10000,211,10000,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,10000,70,10000,10000,0,75,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,10000,10000,10000,10000,75,0,120,10000,10000,10000,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,10000,10000,146,10000,10000,120,0,138,10000,10000,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,10000,10000,97,10000,10000,10000,138,0,101,10000,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,10000,10000,10000,211,10000,10000,10000,101,0,90,85,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,90,0,10000,10000,10000,10000,10000,10000},

{10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,85,10000,0,98,10000,142,10000,10000},

{10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,98,0,86,10000,10000,10000},

{10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,86,0,10000,10000,10000},

{10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,142,10000,10000,0,92,10000},

{10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,92,0,87},

{10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,10000,87,0}

};

**public** String[] cities=**new** String[]{"Arad","Zerind","Oradea","Timisoara",

"Sibiu","Lugoj","Rimnicu Vilcea","Fagaras",

"Mehadia","Drobeta","Craiova","Pitesti",

"Bucharest","Giurgiu","Urziceni","Hirsova",

"Eforie","Vaslui","Isi","Neamt"};//城市名

**public** **int**[]mark=**new** **int**[20];//访问标记

**public** Graph(){//得到数据

markInit();

} /\*\* \* 访问标志初始化 \*/

**public** **void** markInit(){

**for**(**int** i =0; i < mark.length; i++){

mark[i]=0;

}

}/\*\*第一个孩子 \* **@param** g \* **@param** start \* **@return** -1表示一个孩子都没有 \*/

**public** **int** getFirstVex(**int** start){

**if**(start>=0&&start<path.length){

**for**(**int** j =0; j < path.length; j++)

**if**(path[start][j]<10000&&path[start][j]>0)//有关系

**return** j;

}

**return**-1;

} /\*\*下一个孩子 \* **@param** start \* **@param** w \* **@return** 表示图G中顶点i的第j个邻接顶点的下一个邻接顶点 \* 返回-1，表示后面没有邻接点了 \*/

**public** **int** getNextVex(**int** start,**int** w){

**if**(start>=0&&start<path.length&&w>=0&&w<path.length){

**for**(**int** i = w+1; i < path.length; i++)

**if**(path[start][i]<10000&&path[start][i]>0)**return** i;

}

**return**-1;

}

**public** **int** getNumber(){

**return** path.length;

}

}

**public** **static** **class** UCS {

**public** UCS(Graph g,**int** start,**int** end){

**int**[] pre =**new** **int**[20];// 保存各个结点的前驱结点

**int**[] dist =**new** **int**[20];// 用于保存当前结点到起始结点的实际路径长度

**for**(**int** i =0; i < pre.length; i++){

pre[i]=-1; dist[i]=10000;}// 调用一致搜索算法搜索路径

UC\_search(g,start, end, dist, pre);// 打印路径显示函数

displayPath(start, end, pre,g);

}/\*\* \* **@param** start：开始 \* **@param** goal：目的 \* **@param** prev：前驱节点 \* **@param** g：图 \*/

**public** **void** displayPath(**int** start,**int** goal,**int**[] prev,Graph g){

Stack<Integer> stack =**new** Stack<Integer>();

stack.push(goal);**while**(prev[goal]!= start){

stack.push(prev[goal]);

goal = prev[goal];

}

stack.push(start);

System.***out***.print("访问的下标: ");

**for**(**int** i = stack.size()-1; i >=0; i--){

System.***out***.print("-->"+stack.get(i));

}

System.***out***.print("\n访问过程: ");

**for**(**int** i = stack.size()-1; i >=0; i--){

System.***out***.print("-->"+ g.cities[stack.get(i)]);

}

System.***out***.print("\n总长度为： ");

**int** result=0;

**for**(**int** i =0; i < stack.size()-1; i++){

result+=g.path[stack.get(i)][stack.get(i+1)];

}

System.***out***.print(result);System.***out***.println("\n");

g.markInit();

}/\*\* \* **@param** g：图 \* **@param** start：开始 \* **@param** goal：目的 \* **@param** prev：前驱节点 \* \*/

**public** **void** UC\_search(Graph g,**int** start,**int** goal,**int**[] dist,**int**[] pre){

List<Integer> list =**new** ArrayList<Integer>();

list.add(start);**while**(!list.isEmpty()){

*moveMinToTop*(list, dist);// 将dist数组中最小值所对应的结点，移至list队首

**int** current = list.remove(0);// 将list队首的结点出队，并展开

g.mark[current]=1;

**if**(current == goal){

**return**;

}

**for**(**int** j =0; j < g.path[current].length; j++){

**if**(g.path[current][j]<10000&& g.mark[j]==0){

**if**(!*isInList*(j, list))// 结点j不在队列里{

list.add(j);

pre[j]= current;

dist[j]= dist[current]+ g.path[current][j];

}

**else** **if**((dist[current]+ g.path[current][j])< dist[j]){

pre[j]= current;

dist[j]= dist[current]+ g.path[current][j];

}

}

}

**if**(list.isEmpty()){System.***out***.println("搜索不成功！");

}

}

}/\*\* \* 检查结点a，是否在队列list里 \*/

**public** **static** **boolean** isInList(**int** a,List<Integer> list){

**for**(**int** i =0; i < list.size(); i++){

**if**(list.get(i)== a){

**return** **true**;

}

}

**return** **false**;

}/\*\* \* 将dist数组中的最小值所对应的结点，从list队列中移至队列头 \*/

**public** **static** **void** moveMinToTop(List<Integer> list,**int**[] dist){

**int** index =0;**int** min = dist[index];

**for**(**int** i =0; i < list.size(); i++){

**int** a = list.get(i);

**if**(dist[a]< min){

index = i;

min = dist[a];

}

}

**int** temp = list.get(index);

**for**(**int** i = index; i >0; i--){

list.set(i, list.get(i -1));

}

list.set(0, temp);

}

}

实验结果

