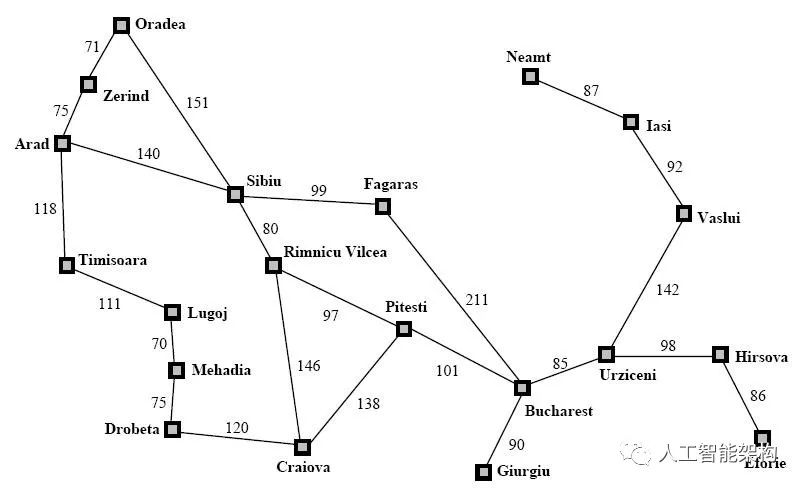
# 迭代加深搜索

## 算法描述：

广度优先搜索可以用迭代加深搜索代替。迭代加深搜索实质是限定下界的深度优先搜索，即首先允许深度优先搜索搜索 *k*层搜索树，若没有发现可行解，再将 *k+1* 后再进行一次以上步骤，直到搜索到可行解。这个算法模仿广度优先搜索法比起广搜是牺牲了时间，但节约了空间。

## 二．罗马尼亚问题：



**从起始点Arad到目标点Bucharest。**

## 三．算法代码：

Graph类（地图）：

**package** IDS;

**public** **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** **int**[]H=**new** **int**[]{516,524,530,479,403,394,343,326,391,392,310,160,150,155,100,0};//启发式函数

**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;

}

}

IDS：

**package** IDS;

**import** java.util.Stack;

**public** **class** IDS {

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

/\*\*迭代加深搜索

\* **@param** g：图

\* **@param** v0：开始节点

\* **@param** vg：目的节点

\* **@param** depthMax：depthMax

\*/

**public** **void** IDS\_Search(Graph g,**int** v0,**int** vg,**int** depthMax){

**for**(**int** i =2; i <=depthMax; i++){//迭代depthMax次

**if**(dfsearch(g, v0, vg,i)==1){

**break**;

}

}

}

/\*\*深度搜索

\* **@param** g：图

\* **@param** v0：开始节点

\* **@param** vg：目的节点

\* **@param** depthMax：depthMax

\* **@return**

\*/

**public** **int** dfsearch(Graph g,**int** v0,**int** vg,**int** depthMax){

**int** x;

**int** w;//v0的第一个邻接点

stack.push(v0);//入栈

g.mark[v0]=1;//v0被访问

**while**(**true**){

x=(Integer) stack.peek();//查看栈顶元素

w=g.getFirstVex(x);

**while**(g.mark[w]==1){//被访问，则寻找下一个邻接点

w=g.getNextVex(x, w);

**if**(w==-1){

**break**;

}

}

**while**(w==-1){//没有找到下一个邻接点

stack.pop();

**if**(stack.size()==0){//清空了栈里的元素

g.markInit();//访问初始化

**return** 0;

}

x=(Integer) stack.peek();

w=g.getFirstVex(x);

**while**(g.mark[w]==1){

w=g.getNextVex(x, w);

**if**(w==-1){

**break**;

}

}

}

stack.push(w);

g.mark[w]=1;

**if**(w==vg)

{

**break**;

}//检查是否达到终点

**if**(stack.size()>=depthMax){//重新迭代则重新初始化值

stack.pop();

}

}

**new** Main().show(g, stack);

**return** 1;

}

}

Main函数：

**package** IDS;

**import** java.util.Stack;

**public** **class** Main{

**int** result;

**int** xiabiao[]=**null**;//访问的下标

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

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

System.***out***.println("----------------罗马尼亚问题---------------");

System.***out***.println("迭代加深的搜索");

IDS ids=**new** IDS();

ids.IDS\_Search(graph,0,12,15);}//深度设15

/\*\*打印

\* **@param** g:图

\* **@param** stack：栈

\*/

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

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

System.***out***.println("路径搜索失败");

**return**;

}

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

}

}

}

## 四．运行结果：

