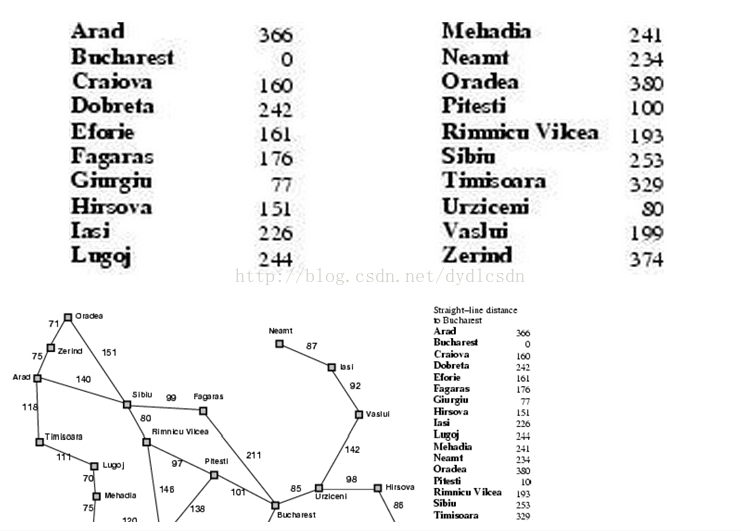
**贪婪搜索实现**

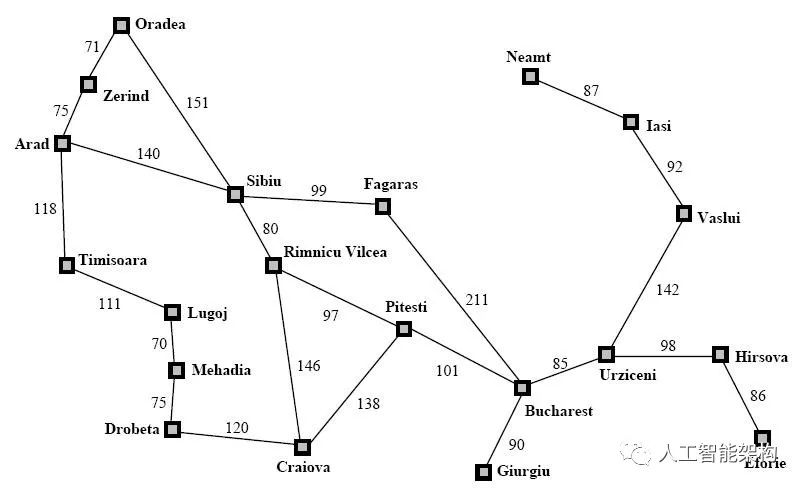
**171491125 吴俊**

1. **问题的提出：**

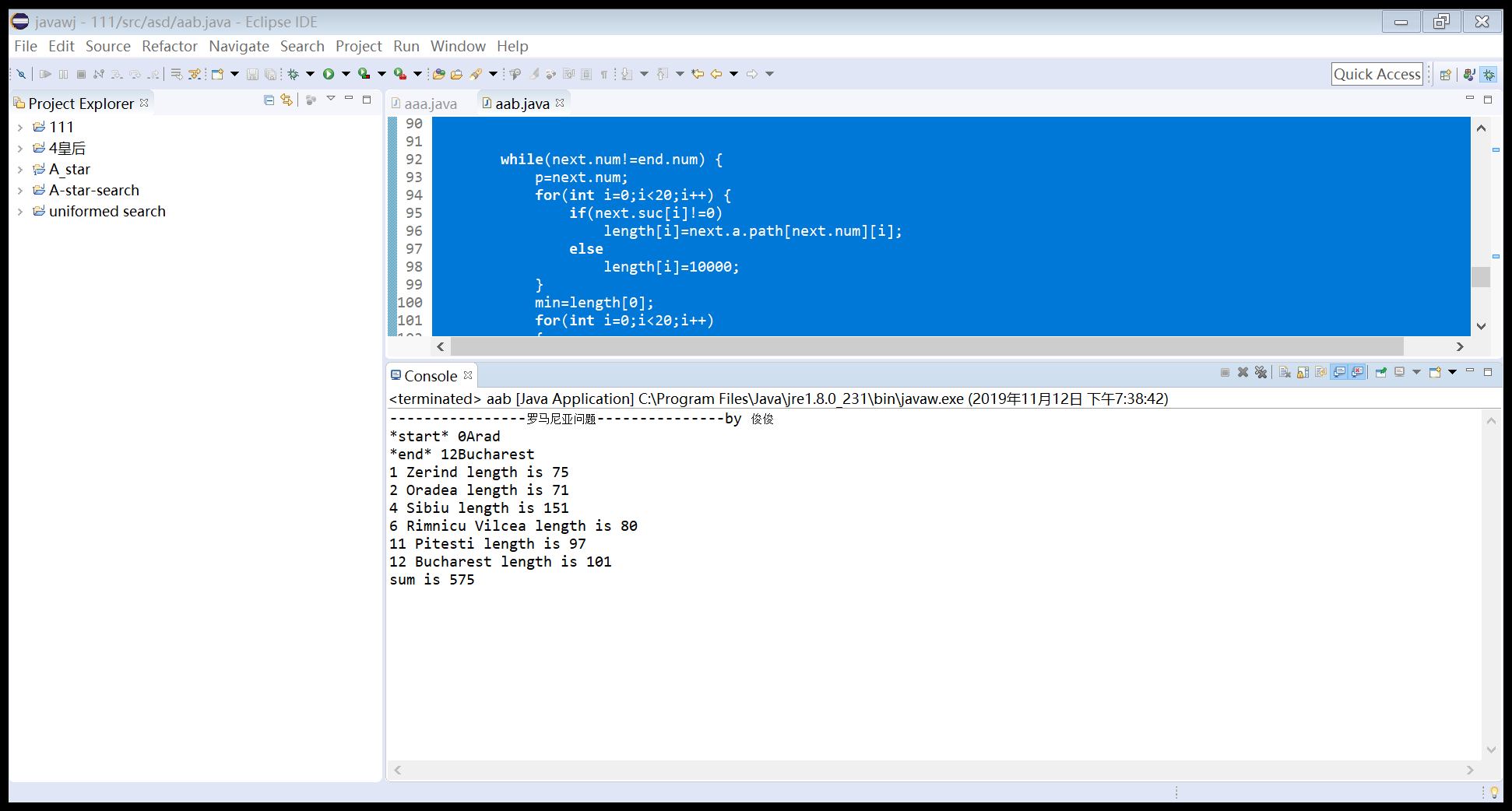
**通过使用贪婪搜索得到从起始点Arad到目标点Bucharest的一条路径，即为罗马尼亚问题的一个解，在求解的过程中记录每种算法得到的解，即输出每种解得到的条路径。**

1. **罗马尼亚地图：**

****

****

1. **实验结果**

****

1. **实验代码**

**package** asd;

**public** **class** aab {

**int** result;

**int** xiabiao[]=**null**;//访问的下标

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

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

node start=**new** node(0);

node end=**new** node(12);

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

*ucs*(start,end);

}

**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** **static** **class** node{

**int** num;

**int**[] suc=**new** **int**[20];

**int**[] pre =**new** **int**[20];

Graph a=**new** Graph();

node(**int** num)

{

**this**.num=num;

**for**(**int** i =0; i < 20; i++) {

**if**(a.path[num][i]>0&&a.path[num][i]<10000)

suc[i]=1;

**else**

suc[i]=0;

}

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

**if**(a.path[i][num]>0&&a.path[i][num]<10000)

pre[i]=1;

}

}

}

**public** **static** **void** ucs(node start, node end) {

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

**int** length[]=**new** **int** [20];

**int** min = 0;

**int** p=0;

**int** sum=0;

System.***out***.println("\*start\*"+" "+start.num+start.a.cities[start.num]);

System.***out***.println("\*end\*"+" "+end.num+end.a.cities[end.num]);

node next=start;

**while**(next.num!=end.num) {

p=next.num;

**for**(**int** i=0;i<20;i++) {

**if**(next.suc[i]!=0)

length[i]=next.a.path[next.num][i];

**else**

length[i]=10000;

}

min=length[0];

**for**(**int** i=0;i<20;i++)

{

**if**(length[i]<min) {

min=length[i];

next.num=i;

}

}

System.***out***.println(next.num+" "+next.a .cities [next.num]+" length is "+length[next.num]);

node aaa=**new** node(next.num);

next=aaa;

next.suc[p]=0;

sum=sum+min;

}

System.***out***.println("sum is "+sum);

}

}

1. **实验中遇到的问题**

开始时的结果，一直在循环。

**问题分析：**

这个代码运行出来的结果是在两个点之间循环。当我仅执行第一步的时候，不论是后继结点数组还是路径长度数组，都是正确的。原因就在于，对于某个结点来说，已经访问过的结点会继续访问造成死循环。

在原理课上曾经说过，贪婪搜索的完备性是不定的，就在这种存在死循环的情况下，就不具备完备性。

所以改进措施是将已经访问过的结点标记，以免造成死循环。

原本的代码：

**public** **static** **void** ucs(node start, node end) {

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

**int** length[]=**new** **int** [20];

**int** min = 0;

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

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

node next=start;

**while**(next.num!=end.num) {

**for**(**int** i=0;i<20;i++) {

**if**(next.suc[i]!=0)

length[i]=next.a.path[next.num][i];

**else**

length[i]=10000;

}

**for**(**int** i=0;i<20;i++)

System.***out***.print(length[i]+" ");

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

min=length[0];

**for**(**int** i=0;i<20;i++)

{

**if**(length[i]<min) {

min=length[i];

next.num=i;

}

}

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

//System.out.print(min);

//System.out.print(" ");

node aaa=**new** node(next.num);

next=aaa;

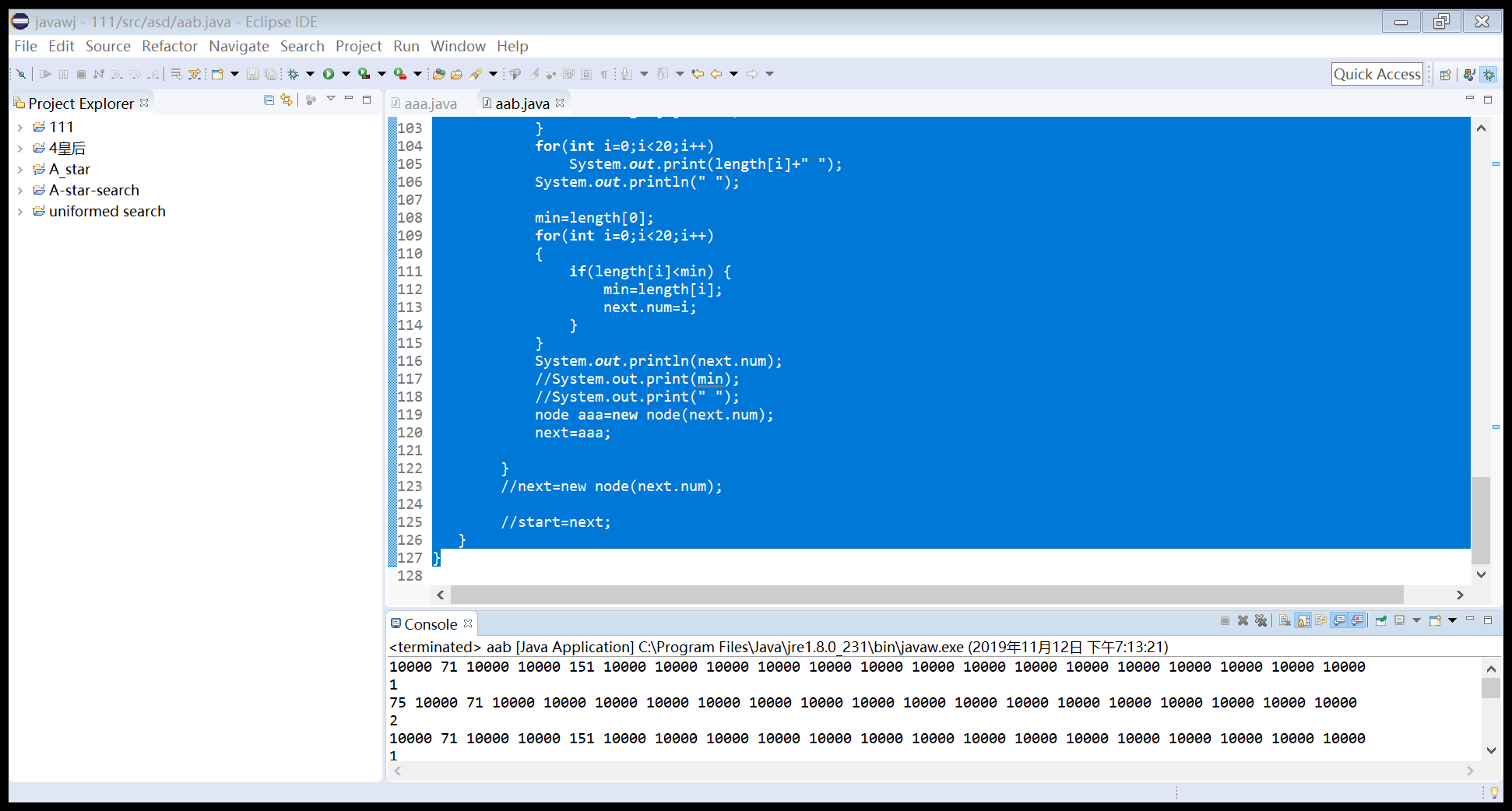
}

//next=new node(next.num);

//start=next;

}

}

输出结果：

改进后的代码在前面。