****

**《人工智能》实验报告**

|  |  |
| --- | --- |
| 学生姓名 | 李栩雅 |
| 学 号 | 8208201116 |
| 专业班级 | 计科2002 |
| 指导教师 | 黄芳 |
| 学 院 | 计算机学院 |
| 完成时间 | 2022.5.24 |

计算机学院

2022年5月

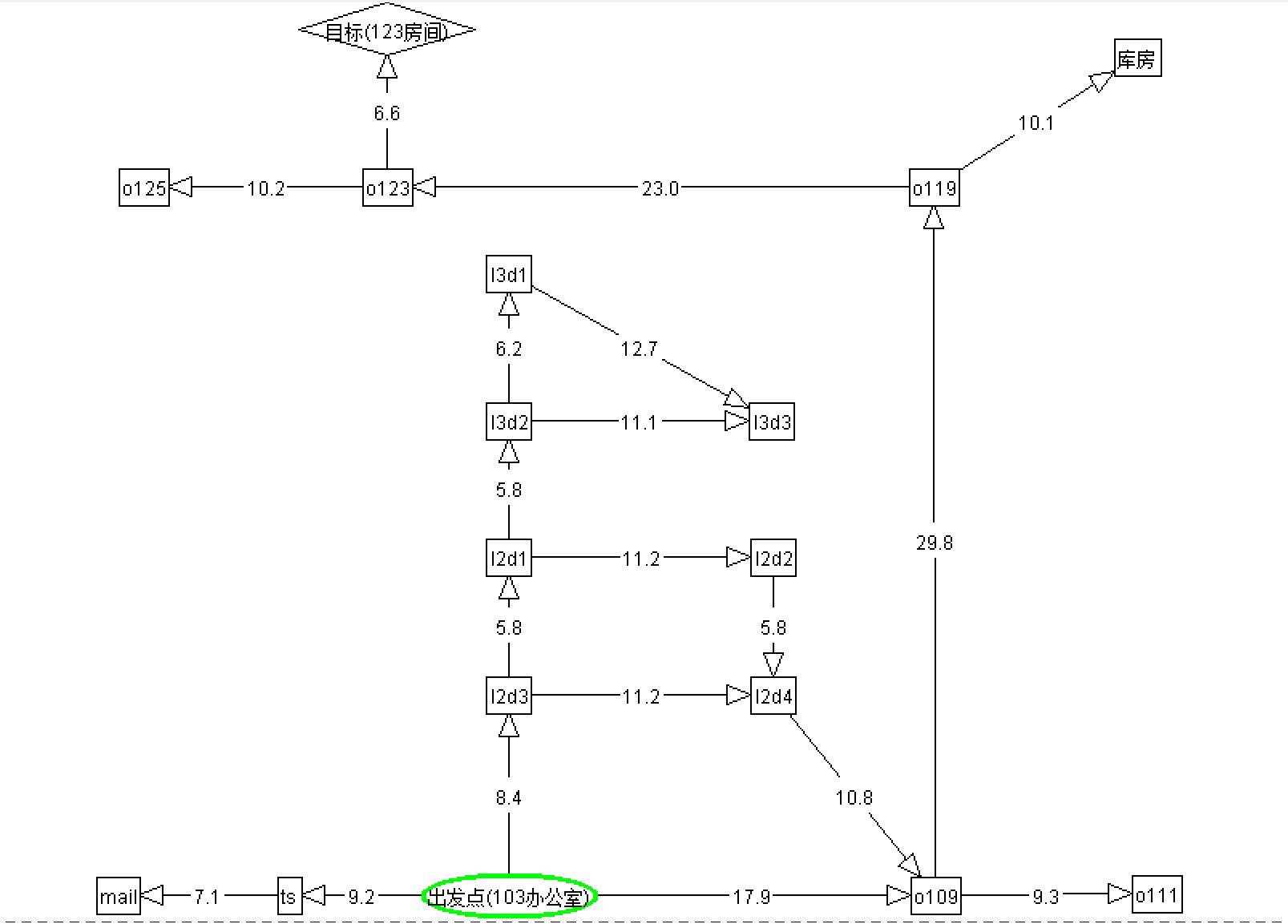
实验一 搜索策略

一、实验目的

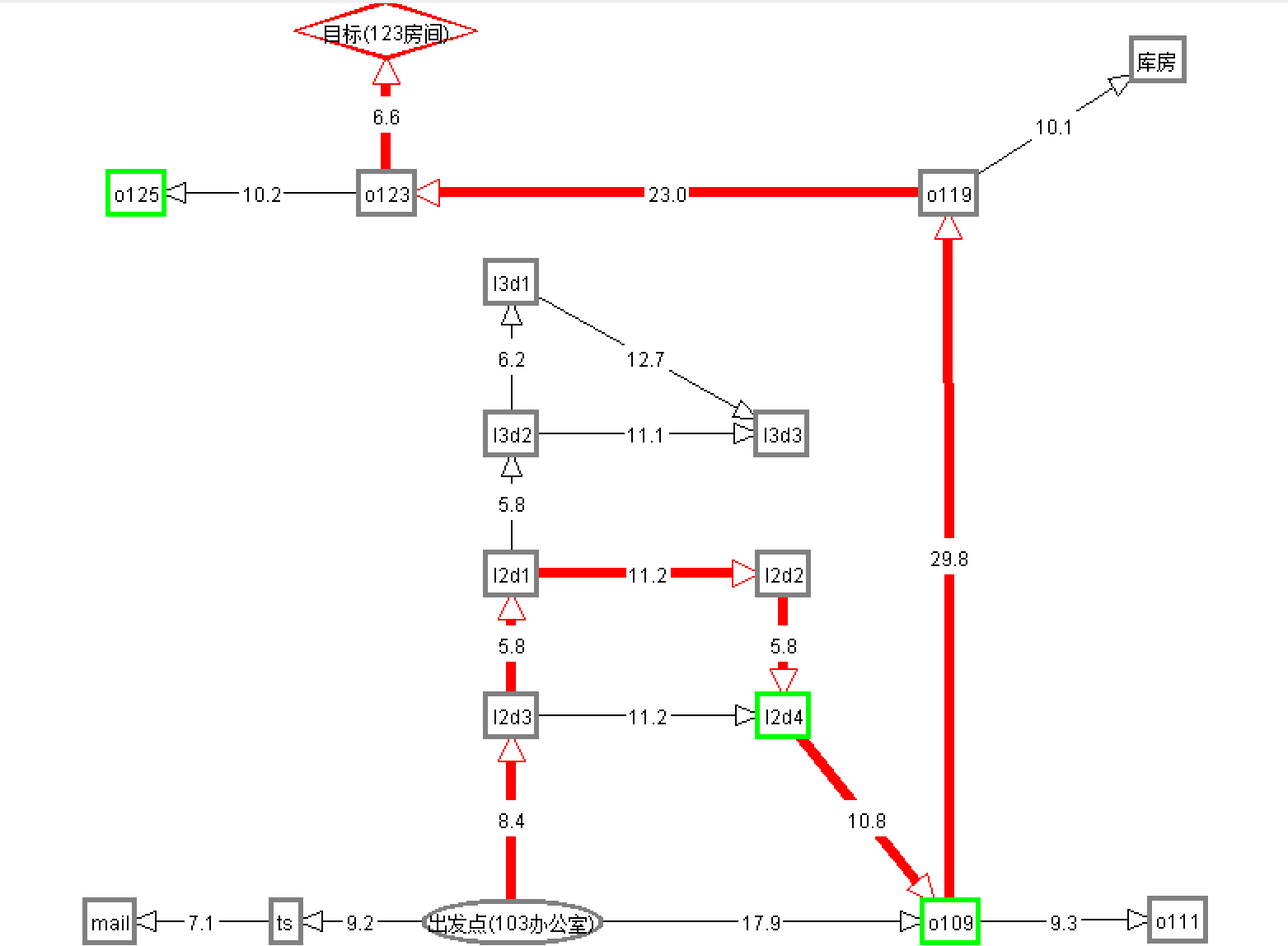
熟悉和掌握各种启发式搜索策略的思想，掌握A\*算法的定义、估价函数和算法过程，理解求解流程和搜索顺序。

二、算法比较

图的搜索



1. DFS



表格如下：

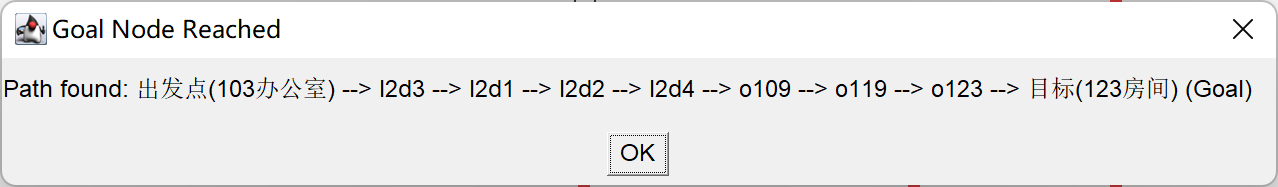
|  |  |  |
| --- | --- | --- |
| Open表 | Close表 | 估价函数 |
| {出发点}  {ts,o109,l2d3}  {mail,o109,l2d3}  { o109,l2d3}  {o109,l2d1,l2d4}  { o109,l2d4,l3d2,l2d2}  { o109,l2d4,l2d2,l3d1,  l3d3}  { o109,l2d4,l2d2,l3d3}  { o109,l2d4,l2d2}  { o109,l2d4,l2d2}  {l2d4,o109}  {o109}  { o111,o119}  {o119}  {o123,库房}  { o123}  {o125,123房间}  GOAL | { }  {出发点}  {出发点,ts}  {出发点,ts,mail}  {出发点,ts,mail ,l2d3}  {出发点,ts,mail ,l2d3,  l2d1}  {出发点,ts,mail ,l2d3,  l2d1, l3d2}  {出发点,ts,mail ,l2d3,  l2d1, l3d2,l3d1}  {出发点,ts,mail ,l2d3,  l2d1, l3d2,l3d1,l3d3}  {出发点,ts,mail ,l2d3,  l2d1, l3d2,l3d1,l3d3}  {出发点,ts,mail ,l2d3,  l2d1, l3d2,l3d1,l3d3}  {出发点,ts,mail ,l2d3,  l2d1, l3d2,l3d1,l3d3,l2d2,l2d4}  {出发点,ts,mail ,l2d3,  l2d1, l3d2,l3d1,l3d3,l2d2,  ,o109,l2d4}  {出发点,ts,mail ,l2d3,  l2d1,l3d2,l3d1,l3d3,l2d2,l2d4}  {出发点,ts,mail ,l2d3,l2d4  l2d1,l3d2,l3d1,l3d3,l2d2.o119}  {出发点,ts,mail ,l2d3,l2d4  l2d1,l3d2,l3d1,l3d3,l2d2.o119,库房}  {出发点,ts,mail ,l2d3,l2d4  l2d1,l3d2,l3d1,l3d3,l2d2.o119,库房,o123} | 无 |

搜索路径：

出发点->ts->mail->l2d3->l2d1->l3d2->l3d1->l3d3->l2d2->l2d4

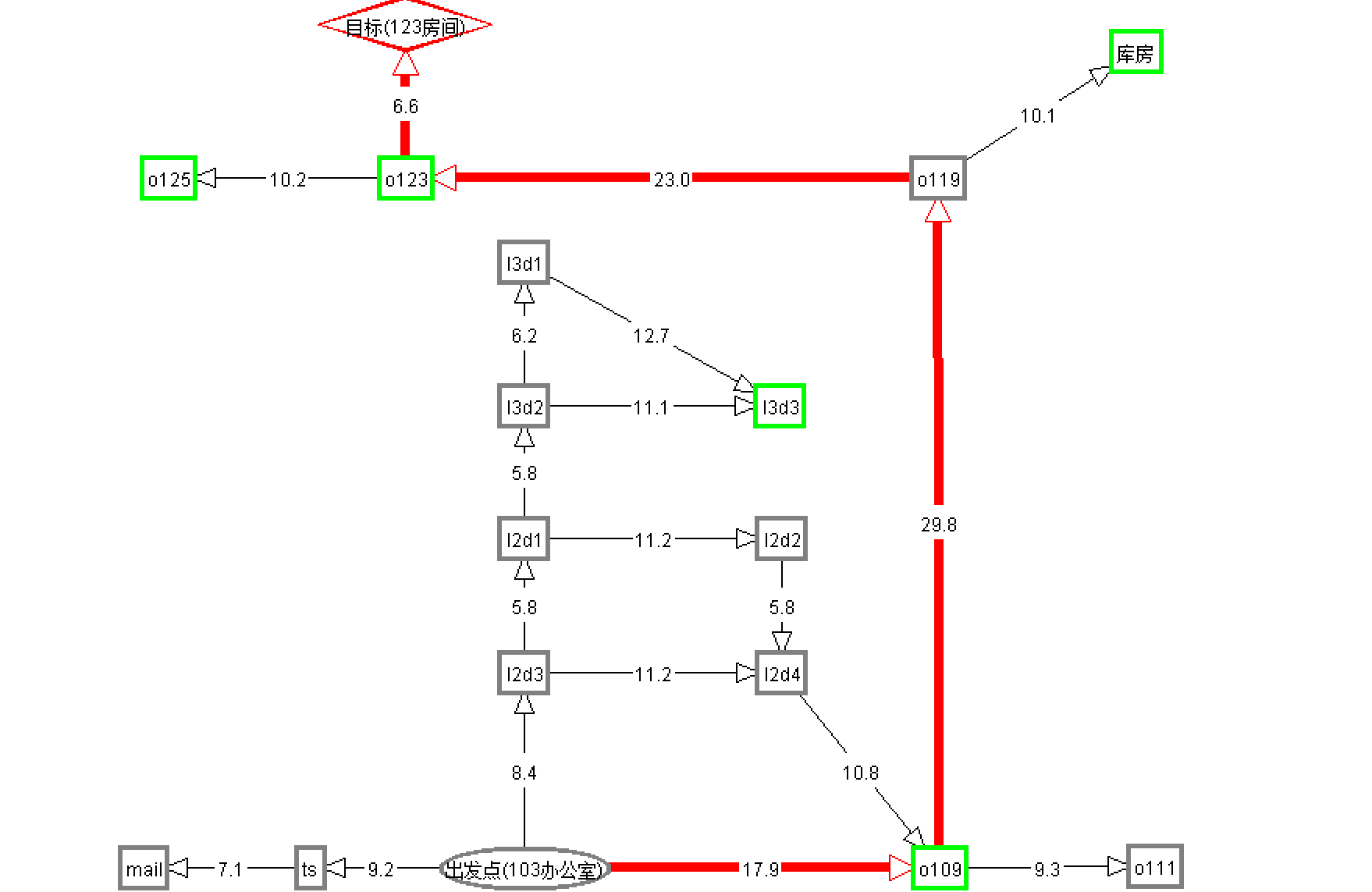
->o109->o111->o119->库房->o123->GOAL

最终路径如下：



学生结论：**DFS使用递归，并且一般使用类似栈的数据结构依次访问所有节点，俗称“一条路走到黑”。当搜索空间很大的时候，搜索效率和图的储存结构有一定关系，如果陷入一条不能通行的分支会导致效率变差；如果图中有环，容易陷入循环无法继续；并且找到的路径不一定是最优路径。**

1. BFS



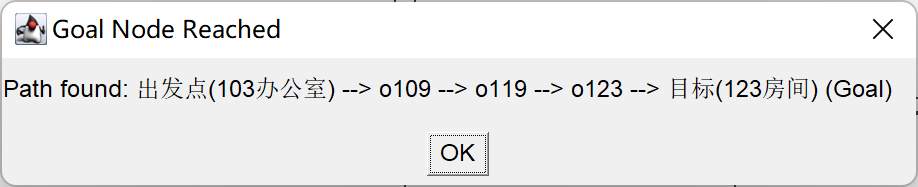
表格如下：

|  |  |  |
| --- | --- | --- |
| Open | Close | 估价函数 |
| {出发点}  {ts,o109,l2d3}  {o109,l2d3,mail}  { l2d3,mail,o111,o119}  { mail,o111,o119,l2d1,l2d4}  { o111,o119,l2d1,l2d4}  { o119,l2d1,l2d4}  { l2d1,l2d4,o123,库房}  { l2d4,o123,库房,l3d2,l2d2}  { o123,库房,l3d2,l2d2,o109}  {库房,l3d2,l2d2,o109,目标,o125}  {l3d2,l2d2,o109,目标,o125}  {l2d2,o109,目标,o125,l3d1,l3d3}  {o109,目标,o125,l3d1,l3d3,l2d4}  {目标,o125,l3d1,l3d3,l2d4,o111,o119}  GOAL | { }  {出发点}  {出发点,ts}  {出发点,ts,o109 }  {出发点,ts,o109,l2d3}  {出发点,ts,o109,l2d3,mail}  {出发点,ts,o109,l2d3,mail,o111}  {出发点,ts,o109,l2d3,mail,o111,o119}  {出发点,ts,o109,l2d3,mail,o111,o119,l2d1}  {出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4}  {出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,0123}  {出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,0123,库房}  {出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,0123,库房,l3d2}  {出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,0123,库房,l3d2,l2d2}  {出发点,ts,o109,l2d3,mail,o111,o119,l2d1,l2d4,0123,库房,l3d2,l2d2} | 无 |

搜索路径：

出发点 -> ts -> o109 -> l2d3 -> mail -> o111 -> o119 -> l2d1 -> l2d4 -> 0123 ->库房-> l3d2 -> l2d2 -> o109 ->目标

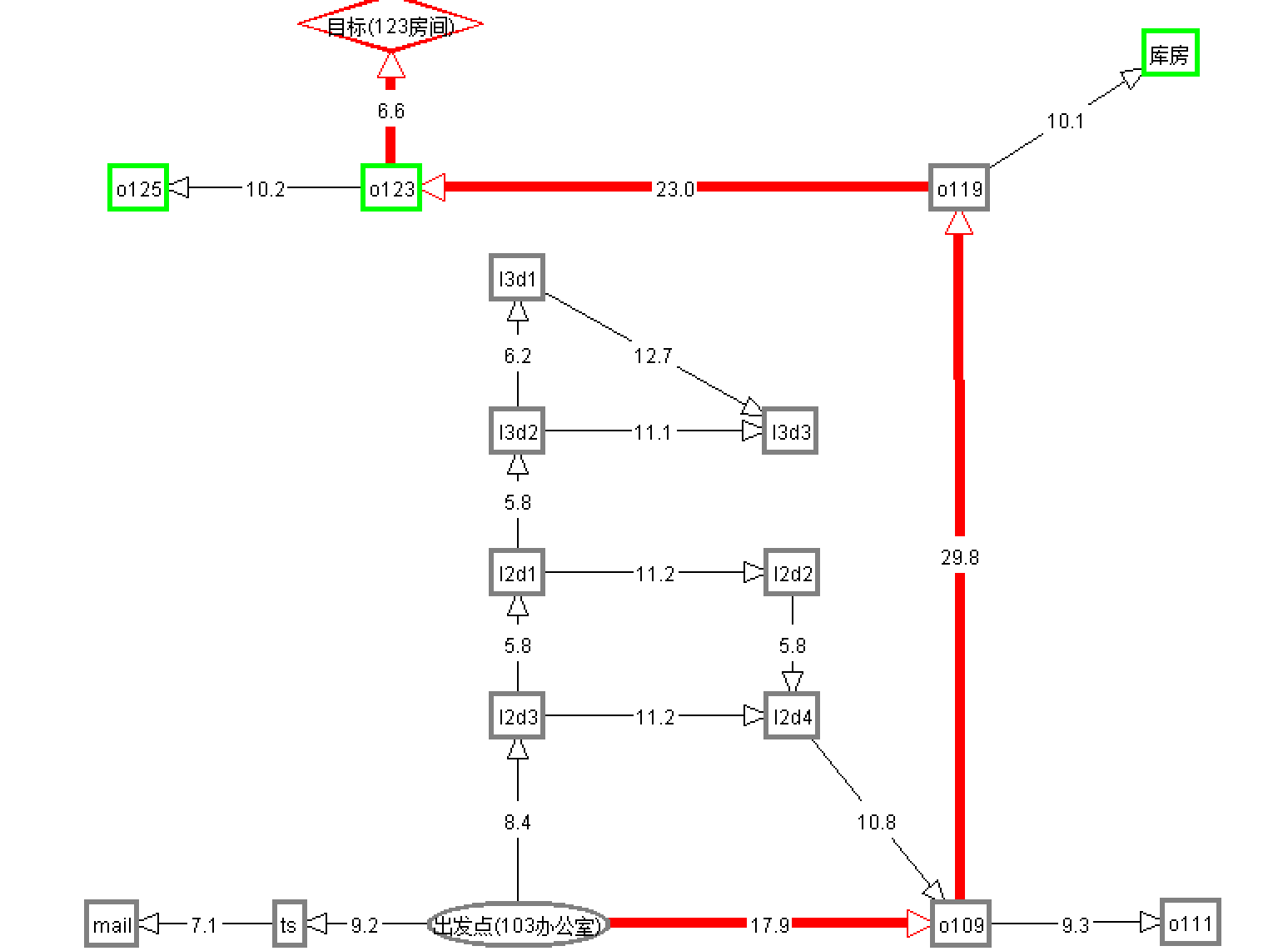
最终路径如下：



学生结论：

**BFS使用队列的数据结构访问所有节点，也是属于穷举类型的盲目搜索，当有循环时依旧可以进行搜索，但是当图的节点较多时搜索速度会减慢，效率较低**。

1. lowest cost(Dijkstra算法)



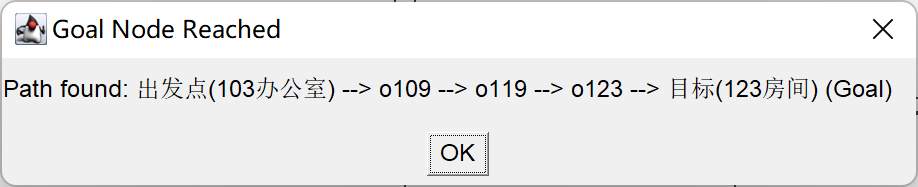
表格如下：

|  |  |  |
| --- | --- | --- |
| Open | Close | 估价函数 |
| {出发点}  {ts:9.2,o109:17.9,l2d3:8.4}  {ts:9.2,o109:17.9,l2d1:14.2,l2d4:19.6}  {o109:17.9,l2d1:14.2,l2d4:19.6,mail:16.3}  {o109:17.9,l2d4:19.6,mail:16.3,l3d2:20.0,l2d2:25.4}  {o109:17.9,l2d4:19.6,l3d2:20.0,l2d2:25.4}  {l2d4:19.6,l3d2:20.0,l2d2:25.4,o111:27.2,o119:47.7}  {l3d2:20.0,l2d2:25.4,o111:27.2,o119:47.7,o109:30.4}  {l2d2:25.4,o111:27.2,o119:47.7,o109:30.4,l3d1:26.2,l3d3:31.1}  {o111:27.2,o119:47.7,o109:30.4,l3d1:26.2,l3d3:31.1,l2d4:31.2,l3d3:38.9}  {o111:27.2,o119:47.7,o109:30.4,l3d3:31.1,l2d4:31.2,l3d3:38.9}  {o119:47.7,o109:30.4,l3d3:31.1,l2d4:31.2,l3d3:38.9}  {o119:47.7,l3d3:31.1,l2d4:31.2,l3d3:38.9,o111:39.7}  {o119:47.7,l2d4:31.2,l3d3:38.9,o111:39.7}  {o119:47.7,o109:42.0,l3d3:38.9,o111:39.7,o109:42.0}  {o119:47.7,o111:39.7,l3d3:38.9,o119:60.2,o109:42.0,o119:71.8}  {o111:39.7,l3d3:38.9,o119:60.2,o109:42.0,o119:71.8}  {o111:39.7,o119:60.2,,o109:42.0,o119:71.8,库房:57.8,o123:70.7}  {o119:60.2,o109:42.0,o119:71.8,库房:57.8,o123:70.7,o111:51.3}  {o119:60.2,o119:71.8,库房:57.8,o123:70.7,o111:51.3}  {o119:60.2,o119:71.8,库房:57.8,o123:70.7}  {o119:60.2,o119:71.8,o123:70.7}  {o119:71.8,o123:70.7,库房:70.3,o123:83.2}  {o119:71.8,o123:70.7,o123:83.2}  {o119:71.8,o123:83.2,目标:77.3,0125:80.9}  {o123:83.2,目标:77.3,0125:80.9,库房:81.9}  GOAL | { }  {出发点}  {出发点,l2d3}  {出发点,l2d3,ts}  {出发点,l2d3,ts,l2d1}  {出发点,l2d3,ts,l2d1,mail}  {出发点,l2d3,ts,l2d1,mail,o109}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119,库房}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119,库房}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119,库房}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119,库房,o123}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119,库房,o123}  {出发点,l2d3,ts,l2d1,mail,o109,l2d4,l3d2,l2d2,l3d1,o111,l3d3,0119,库房,o123} | f(n)=g(n) |

搜索路径：

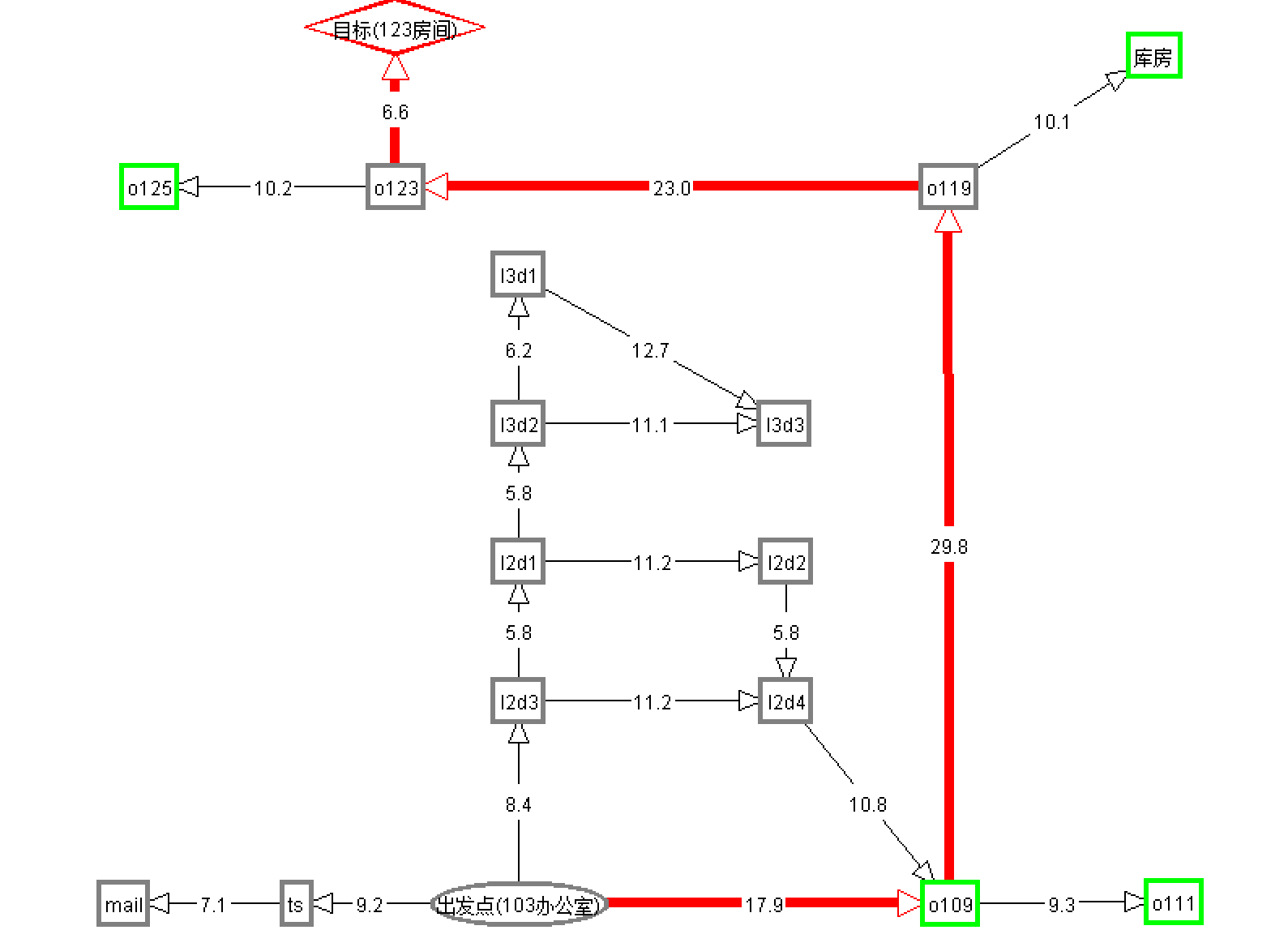
出发点 -> l2d3 -> ts -> l2d1 -> mail -> o109 -> l2d4 -> l3d2 -> l2d2 -> l3d1 -> o111 -> o109 -> l3d3 -> l2d4 -> l3d3 -> o111 -> o109 -> o119 -> o111 -> 库房 -> o119 -> 库房 -> o123 -> o119 ->目标

最终路径如下：



学生结论：**最短路径搜索是在每次记录起点到此刻的搜索代价，寻找代价最小的节点往下搜索。这样可以保证最终找到的路径是最短的。但是当搜索路径复杂或是有环时，搜索容易绕圈并且降低，算法效率会大大降低。**

1. Best First(贪婪)

O

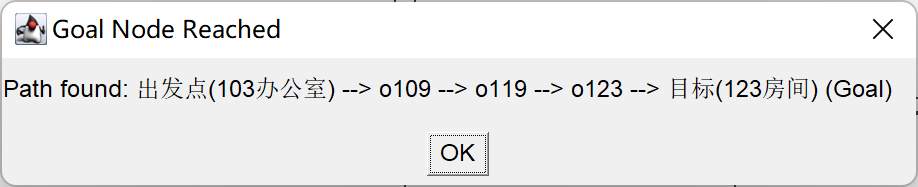
表格如下：

|  |  |  |
| --- | --- | --- |
| Open | Close | 估价函数 |
| {出发点}  {ts:36.6,o109:43.1,l2d3:28.4}  {ts:36.6,o109:43.1,l2d1:22.8,l2d4:32.4}  {ts:36.6,o109:43.1,l2d4:32.4,l3d2:17.2,l2d2:27.5}  {ts:36.6,o109:43.1,l2d4:32.4,l2d2:27.5,l3d1:11.4,l3d3:23.0}  {ts:36.6,o109:43.1,l2d4:32.4,l2d2:27.5,l3d3:23.0,l3d3:23.0}  {ts:36.6,o109:43.1,l2d4:32.4,l2d2:27.5,l3d3:23.0}  {ts:36.6,o109:43.1,l2d4:32.4,l2d2:27.5}  {ts:36.6,o109:43.1,l2d4:32.4,l2d4:32.4}  {ts:36.6,o109:43.1,l2d4:32.4}  {ts:36.6,o109:43.1}  {o109:43.1,mail:38.1}  {o109:43.1}  {o111:48.6,0119:23.9}  {o111:48.6,库房:31.5,o123:6.6}  {o111:48.6,库房:31.5,目标:0}  GOAL | { }  {出发点}  {出发点,l2d3}  {出发点,l2d3,l2d1}  {出发点,l2d3,l2d1,l3d2}  {出发点,l2d3,l2d1,l3d2,l3d1}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3,l2d2}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3,l2d2,l2d4}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3,l2d2,l2d4}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3,l2d2,l2d4,ts}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3,l2d2,l2d4,ts,mail}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3,l2d2,l2d4,ts,mail,o109}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3,l2d2,l2d4,ts,mail,o109,o119}  {出发点,l2d3,l2d1,l3d2,l3d1,l3d3,l2d2,l2d4,ts,mail,o109,o119,o123} | f(n)=h(n) |

搜索路径：

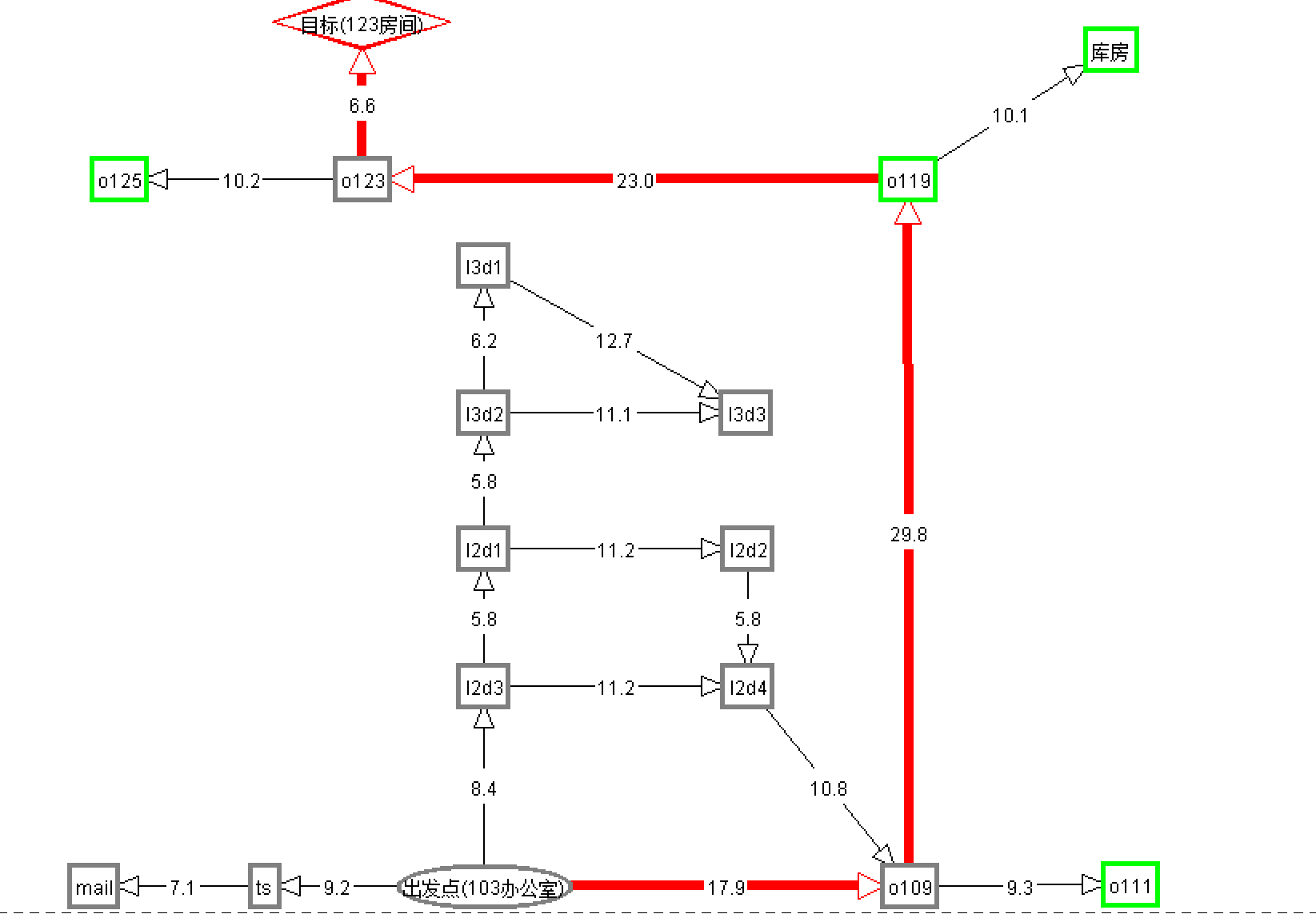
出发点--> l2d3--> l2d1--> l3d2--> l3d1--> l3d3--> l2d2--> l2d4--> ts --> mail--> o109--> o119--> o123

最终路径如下：



学生结论：**贪婪搜索，即只扩展当前代价最小的节点(或者说离当前节点最近的点)，也是一种启发式搜索**。**这样做的缺点就是，目前代价小，之后的代价不一定小，如果解在代价最大的点，那么按照贪婪最佳优先算法，可能就找不到这个解，然后就会陷入死循环。**

1. A\*



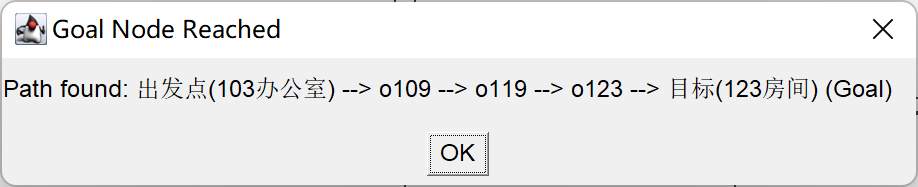
表格如下：

|  |  |  |
| --- | --- | --- |
| Open | Close | 估价函数 |
| {出发点}  {ts:45.8,o109:61.0,l2d3:36.8}  {ts:45.8,o109:61.0,l2d1:37.0,l2d4:52.0}  {ts:45.8,o109:61.0,l2d4:52.0,l3d2:37.2,l2d2:52.9}  {ts:45.8,o109:61.0,l2d4:52.0,l2d2:52.9,l3d1:37.6,l3d3:54.1}  {ts:45.8,o109:61.0,l2d4:52.0,l2d2:52.9,l3d3:54.1,l3d3:61.9}  {mail:54.4,o109:61.0,l2d4:52.0,l2d2:52.9,l3d3:54.1,l3d3:61.9}  {mail:54.4,o109:61.0,l2d2:52.9,l3d3:54.1,l3d3:61.9,o109:73.5}  {mail:54.4,o109:61.0,l3d3:54.1,l3d3:61.9,o109:73.5,l2d4:63.6}  {mail:54.4,o109:61.0,l3d3:61.9,o109:73.5,l2d4:63.6}  {o109:61.0,l3d3:61.9,o109:73.5,l2d4:63.6}  {l3d3:61.9,o109:73.5,l2d4:63.6,o111:75.8,o119:71.6}  {o109:73.5,l2d4:63.6,o111:75.8,o119:71.6}  {o109:73.5,o111:75.8,o119:71.6,o109:85.1}  {o109:73.5,o111:75.8,o109:85.1,库房:89.3,o123:77.3}  {o111:75.8,o109:85.1,库房:89.3,o123:77.3,o111:88.3,o119:84.1}  {o109:85.1,库房:89.3,o123:77.3,o111:88.3,o119:84.1}  {o109:85.1,库房:89.3,o111:88.3,o119:84.1,o125:93.0,目标:77.3}  GOAL | { }  {出发点}  {出发点,l2d3}  {出发点,l2d3,l2d1}  {出发点,l2d3,l2d1,l3d2}  {出发点,l2d3,l2d1,l3d2,l3d1}  {出发点,l2d3,l2d1,l3d2,l3d1,ts}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3,mail}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3,mail,o109}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3,mail,o109}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3,mail,o109}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3,mail,o109,o119}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3,mail,o109,o119}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3,mail,o109,o119}  {出发点,l2d3,l2d1,l3d2,l3d1,ts,l2d4,l2d2,l3d3,mail,o109,o119,o123} | f(n)=g(n)+h(n) |

搜索路径：

出发点--> l2d3--> l2d1--> l3d2--> l3d1--> ts--> l2d4--> l2d2--> l3d3 --> mail--> o109--> l3d3--> l2d4-->o119-->o109-->o111-->o123

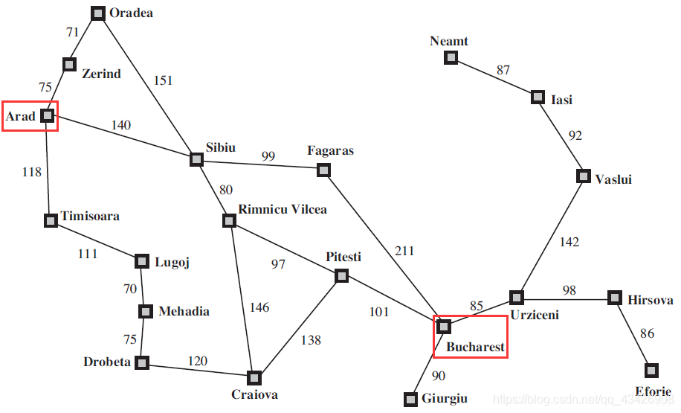
最终路径如下：



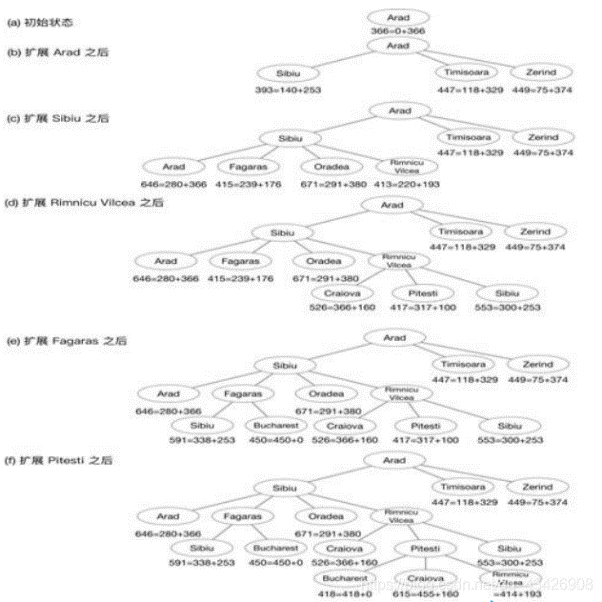
学生结论：**A\*是一种启发式搜索算法、一种静态路网中求解**[**最短路径**](https://baike.baidu.com/item/%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84/6334920)**最有效的直接搜索方法，也是解决许多搜索问题的**[**有效**](https://baike.baidu.com/item/%E6%9C%89%E6%95%88)**算法。它兼顾了搜索效率和可靠性，可以根据不同的情况使用不同的启发函数来改变搜索策略，针对每个图找到最优的搜索方式。**

三、修改启发函数的比较

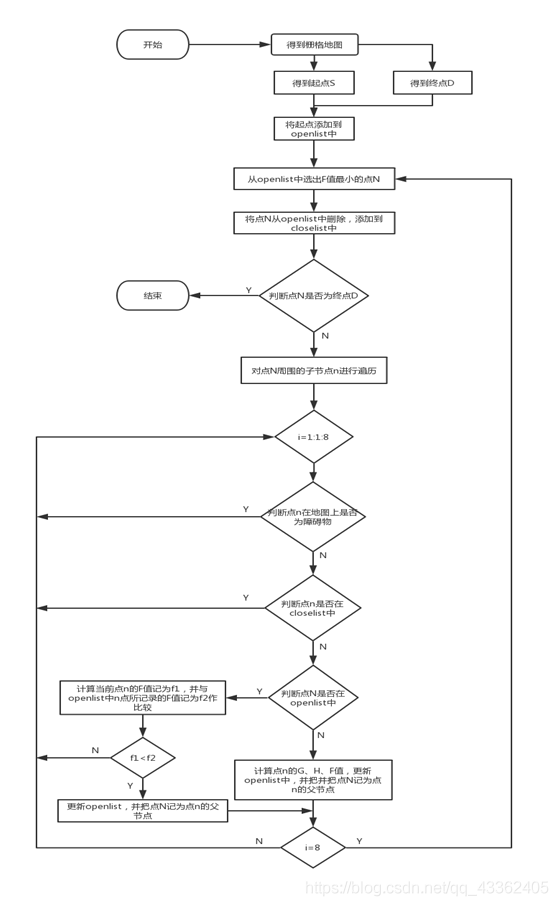
**本实验我选择了路径规划进行启发函数的比较，参考了一篇博客的代码，即解决罗马尼亚问题，并修改了启发函数进行测试，比较了不同启发函数对搜索路径的影响。**



**△上图为路径规划代码的可视化图。**



**△上图为A\*算法的可视化图**



**△流程图如上**

结论分析：

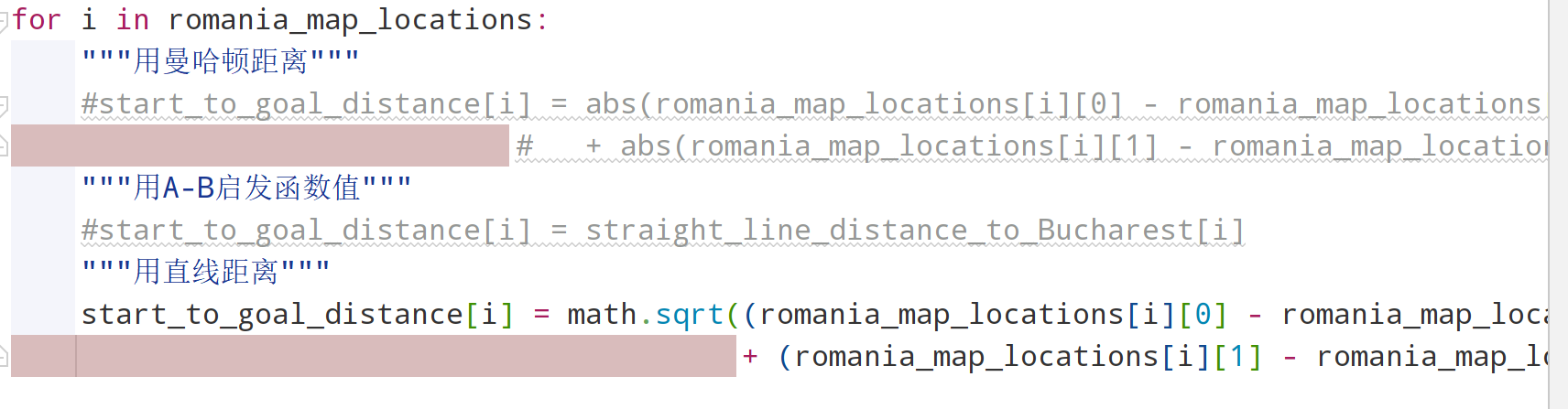
**BFS和DFS属于盲目搜索，适用于无权路径规划问题。对于BFS,，如果路径代价是基于结点深度的非递减函数，则算法是最优的，否则不具备最优性。**

**在盲目搜索策略中，往往不考虑结点离目标有多远，也就是说，在搜索的过程中，并不考虑目标的状态对当前搜索行为的影响，而仅仅依赖于当前的状态；比如在罗马尼亚问题中，从城市 Arad 到城市 Bucuresti 的搜索，仅仅只考虑在当前状态下可能导致的代价，该策略虽然能搜索到最优解，但是在搜索过程中可能会有很多不必要的操作，因为搜索的结果只关注那条最优的路径，但是实际在搜索的过程中可能会搜索到与最优路径无关的结点。而启发式搜索策略则不一样，他会考虑目标状态对当前状态的影响，这样做使得在搜索的过程中更大的可能搜索和最优路径相关的状态，减少不必要的搜索，增加搜索的效率，比如说在罗马尼亚问题中，可以在考虑代价的同时，还考虑当前结点离目标结点的直线距离来选取下一个要拓展的结点，而这个直线距离就作为启发式的信息来帮助搜索的决策，以此更快地搜索到最优路径。**

**在启发式搜索中，通常用 g(n) 表示结点 n 当前的代价，用 h(n) 表示启发函数，一般来说 h(n) = 结点 n 到目标结点的最小代价路径的代价估计值（比如在罗马尼亚问题中可以用 Arad 到 Bucuresti 的直线距离来作为从 Arad 到 Bucuresti 的最小代价路径的代价估计值），用 f(n) 作为选择结点的代价评估值，f(n)=g(n)+h(n) ，当 f(n) 越小，则代表结点越优；启发式搜索对扩展结点的选择同时依赖于 g(n) 和 h(n)，对于 h(n) 较小的结点，我们可以认为该结点会让我们离目标结点更近，所以 h(n) 作为启发式的信息能帮助我们在搜索的时候做出决策。**

详细分析：

**在 BF 算法中，仅仅采用启发式作为评估值，可能导致不能求到最优解，A\* 搜索算法同时考虑到代价，以期解决 BF 算法中的问题，因此 A\* 搜索采用的代价评估值 f(n) 为 h(n) 与 g(n) 的和，针对不同的情况采用不同的估价函数，参考代码中有以下几种：**

****

1. **曼哈顿距离（最终路径450）**
2. **A-B启发函数值（418）**
3. **直线距离（418）**