作业4.1 – 1133730117 - 张实唯

## 实验结果

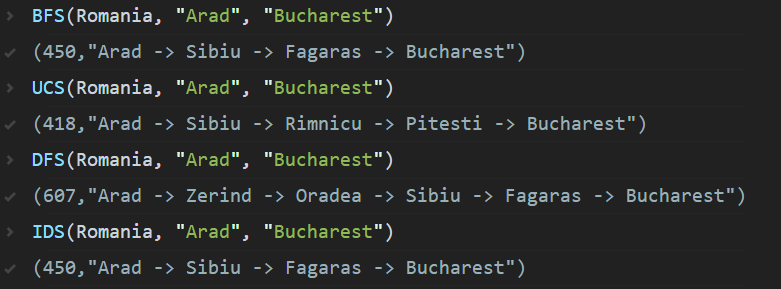
1. **哈工大数据集**

实际最佳路线为 "正心楼 -> 3G体验店 -> 工商银行 -> 诚意楼"



1. **罗马尼亚数据集**

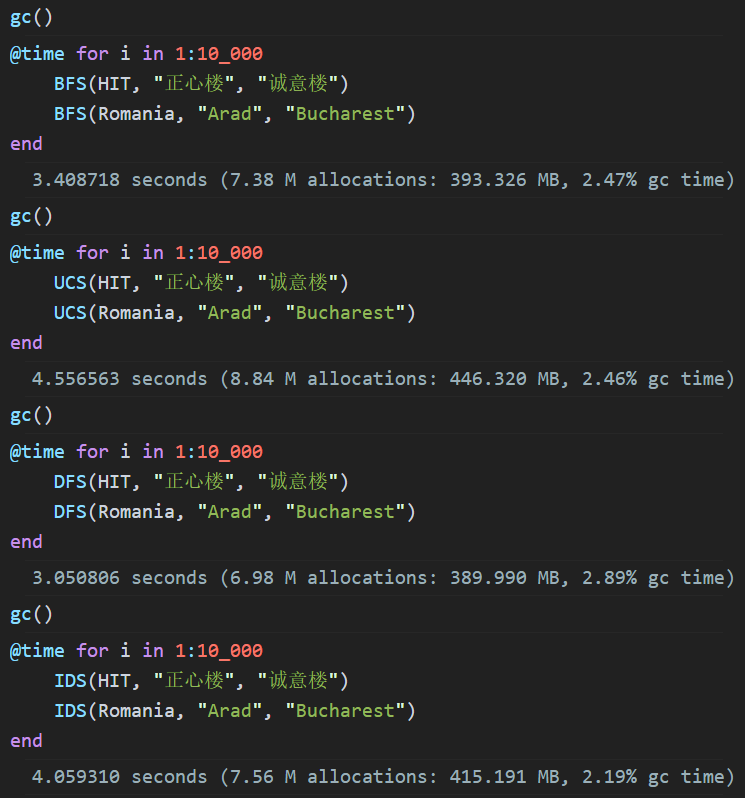
实际最佳路线为 "Arad -> Sibiu -> Rimnicu -> Pitesti -> Bucharest"



1. **运行速度**

使用不同算法运行上述两个测试各10000次，记录总时间。

由于使用的图比较小，因此各个算法的优劣不能很好的从这两个测试中体现出来。比如UCS扩展的节点数最少，但是由于优先队列算法较为复杂，运行时间反而较长。



1. **小结**

|  |  |  |  |
| --- | --- | --- | --- |
| 算法 | 完备性\* | 最优性\*\* | 时空开销\*\*\* |
| BFS | 是 | 否 | 3.4 |
| UCS | 是 | 是 | 4.6 |
| DFS | 是 | 否 | 3.1 |
| IDS | 是 | 否 | 4.1 |

\*两个测试中均能找到正确路径

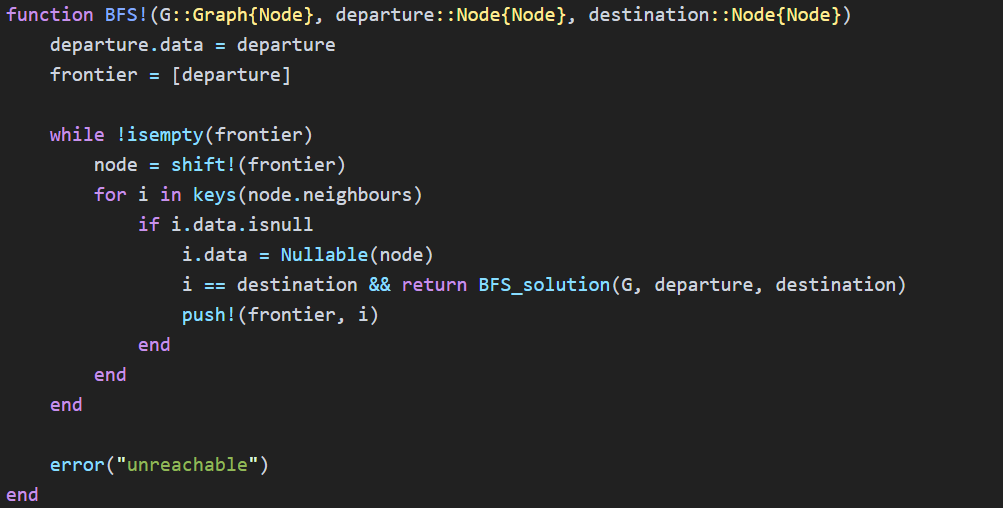
\*\*两个测试中均能找到最优路径

\*\*\*运行两个测试10000次所需时间(秒)，由于语言使用GC管理内存，难以测定空间开销

## 代码解释

1. **BFS**

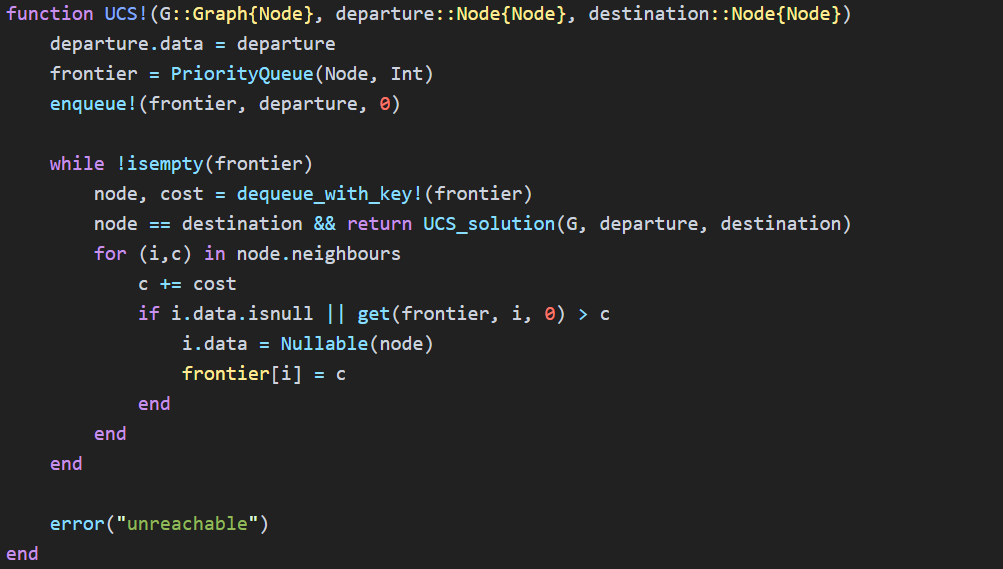
相比课件上的算法，我在节点的数据结构中增加了一个data字段，在扩展子节点的时候将子节点的data指针指向它的父节点，这样检查每个节点的data指针是否为null即可知道该节点是否已经被访问。这样可以省去遍历frontier和explored的时间，explored集合也不再需要。此外在算法结束时只要从目的节点开始跟随data指针一路追回到出发点即可生成路径，不需要在搜索时单独记录了。



1. **UCS**

作业里没有要求UCS，但是其它几个算法都不能查找到最优路径，所以我实现了这个算法作为对比。

实现方法和BFS几乎相同，只需要将FIFO队列换成PriorityQueue即可。



1. **DFS**

使用递归实现的DFS方法，通过limit == 0 来判断深度限制，这样在一开始将limit设置为-1即可不限深度。stack集合用来保存栈中的节点，避免在不限深度时无限循环。这个开销只有O(m)，不增加整个算法的空间复杂度。



1. **IDS**

直接调用DFS，逐渐提高深度限制。

