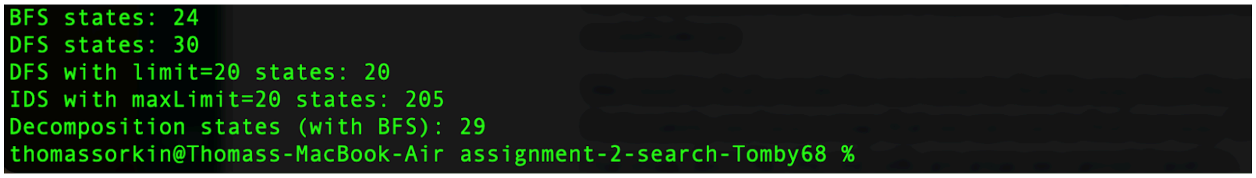
Thomas Sorkin
Question 1:

Algorithm	Time Complexity	Space Complexity	Complete?	Optimal?
BFS	O(b^d)	O(b^d)	Yes	Yes (Uniform cost only)
UCS	O(b^(1+C/ε)	O(b^(1+C/ε)	Yes	Yes
DFS	O(b^d)	O(bd)	No	No
DLS	O(b^L)	O(bL)	No	No
IDS	O(b^d)	O(bd)	Yes	Yes (Uniform cost only)
A*	O(b^d)	O(b^d)	Yes	Yes

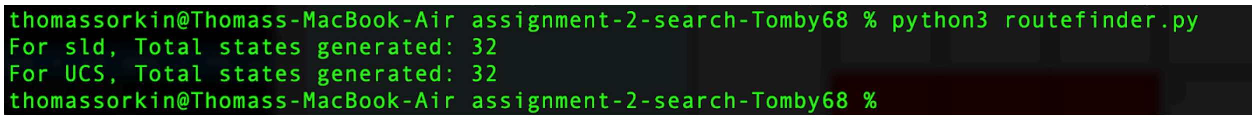
b: branching factor
d: depth (max depth for DFS)
C: Optimal solution cost
ε: Minimum cost
L: Depth limit

Question 2:
Search Algorithm State data:



```
BFS states: 24
DFS states: 30
DFS with limit=20 states: 20
IDS with maxLimit=20 states: 205
Decomposition states (with BFS): 29
thomassorkin@Thomass-MacBook-Air assignment-2-search-Tomby68 %
```

Question 3:
A* and UCS State data:



```
thomassorkin@Thomass-MacBook-Air assignment-2-search-Tomby68 % python3 routefinder.py
For sld, Total states generated: 32
For UCS, Total states generated: 32
thomassorkin@Thomass-MacBook-Air assignment-2-search-Tomby68 %
```

Question 5:

1. The engineering advances that gave Deep Blue an advantage over its contemporaries include: a complex evaluation function implemented in hardware, an increased number of chess chips in the system, allowing for massive parallelization, and the addition of evaluation and debugging tools to improve the system. The hardware-implemented evaluation function is very chess-specific, and makes these chess chips only usable to play chess (hence the name of the chips). The evaluation and debugging tools may have some applications elsewhere. Overall, however, the specific implementation of Deep Blue II is very chess-specific. This is not a tool that can play Go or Shogi, since it is hyper focused on chess and chess strategies. Still, the lessons are applicable in most computer science related problems. For example, one lesson is the effectiveness of mass parallelization, which is now a commonplace practice in computing. Another lesson is the power of a system-on-a-chip design. This is also now a commonplace practice in certain specialized computing environments such as distributed systems, where FPGAs or ASICs are used frequently (The summer research I did was partly on ASICs in switches used in supercomputing networks!)
2. The difference in moves, or possible states, that AlphaZero examines per second versus the number that StockFish examines is staggering. In the game of chess, AlphaZero searches through 80 thousand positions per second. Comparatively, StockFish searches 70 million moves per second. Despite this huge difference in the number of states searched per second, AlphaZero still outperforms StockFish. This is because AlphaZero prioritizes the nodes it evaluates. Where StockFish evaluates every possible move, AlphaZero looks only at the “most favorable” nodes in order to make its decision - This is closer to how a human would play chess, since there are objectively bad moves, like sacrificing a pawn for no gain, that do not need to be analyzed. AlphaZero likely ranks nodes using a priority queue and a heuristic function similar to our A* assignment, albeit much more complex. In this way, AlphaZero ensures it is only searching through the best states by popping from the head of the priority queue.