Alessio Mazzone

CS 1571

Project 1 Report

**1) Briefly describe your project setup: which version of Python? Basic computer configurations.**

Python Version: 2.7.10

Computer: Macbook Pro Early 2015

IDE: Visual Studio Code with Microsoft Python Extension

Operating System: macOS Mojave version 10.14

**2) If you discussed the project with someone, list their names and briefly describe what you talked about. Relatedly, if you consulted any sources outside of the class notes and the textbook, please give citations to them here.**

I discussed string/tuple/state/board/path representation with Hassan Syed.

No outside sources other than class notes, the textbook, and the Python 2 documentation (<https://docs.python.org/2/contents.html>) were used.

**3) Report any known bugs you have.**

There are no known bugs in my program.

**4)** **For each puzzle type:**

**a) Enumerate your action expansion order**

**b) Give the name of the heuristic function (so that the grader can supply it as an argument in the command line). If you made up any heuristic function(s) in addition to the ones we specified, explain what it is and why you designed it that way.**

**c) show a transcript of the expected run.**

Water Jugs:

1. My actions are added to the actions array in the following order:

If jug1 is empty, it is filled from the tap.

If jug1 is not empty but not at capacity, it is filled from the tap or dumped to ground.

If jug2 is empty or not at capacity, jug1 is dumped into jug2.

If jug2 is empty, it is filled from the tap.

If jug2 is not empty but not at capacity, it is filled from the tap or dumped to ground.

If jug1 is empty or not at capacity, jug2 is dumped into jug1.

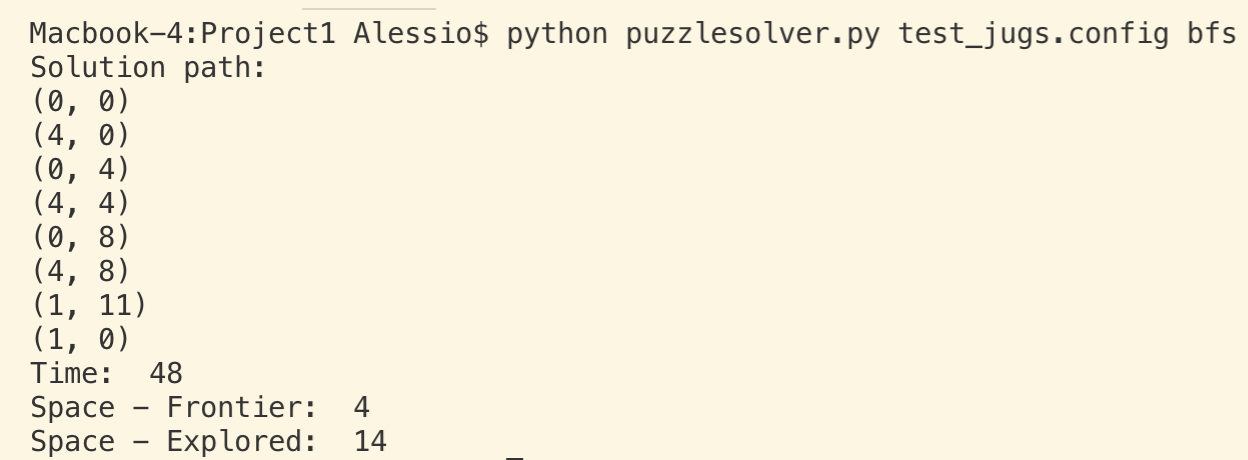
1. Heuristic function: “proximity”

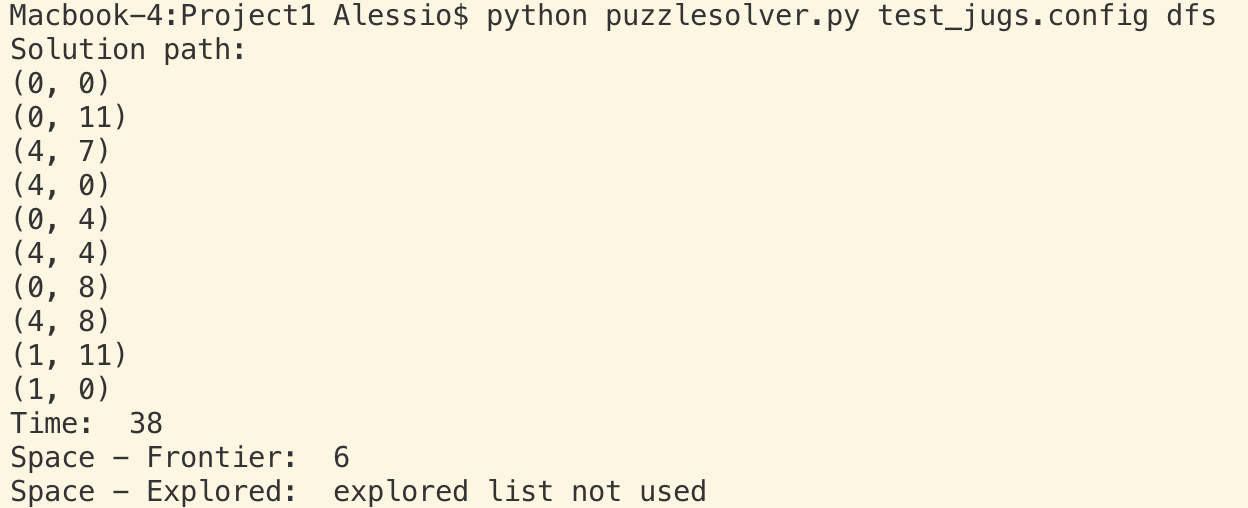
Proximity function checks the state for how close it is to the goal state. It checks how far away jug1 is from the goal and how far away jug2 is from the goal. For three jug problem, jug3 is also checked. The cost is calculated as:

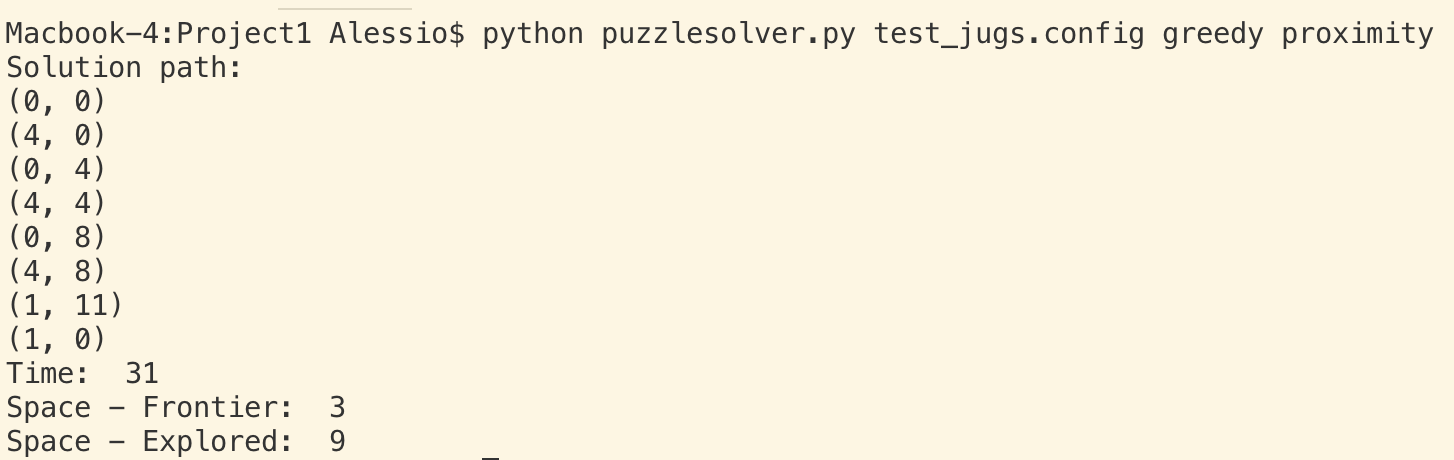
Cost two jugs = abs(curr\_jug1 - gloal\_jug1) + abs(curr\_jug2 - goal\_jug2)

Cost three jugs = abs(curr\_jug1 - gloal\_jug1) + abs(curr\_jug2 - goal\_jug2) + abs(curr\_jug3 – goal\_jug3)

c)

test\_jugs.config with bfs:

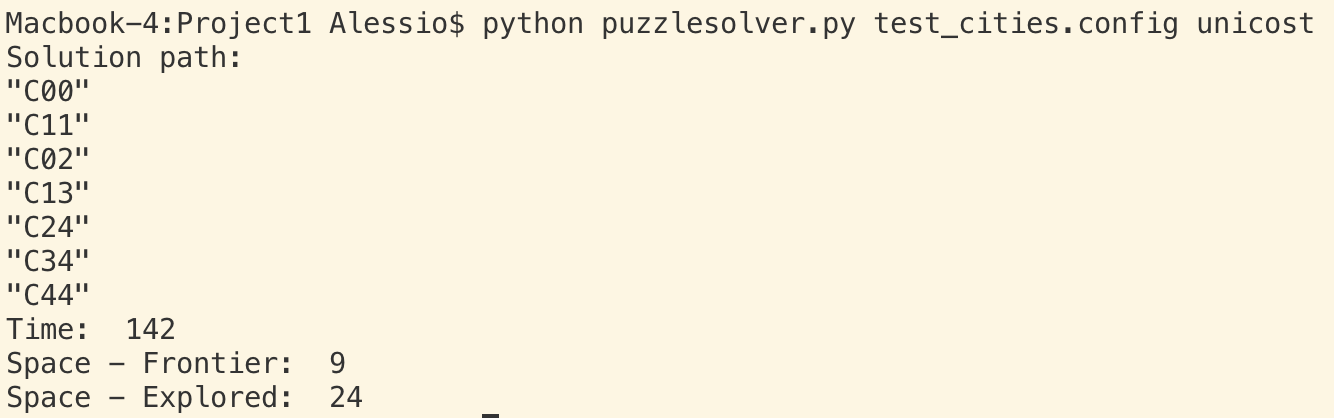
test\_jugs.config with dfs:

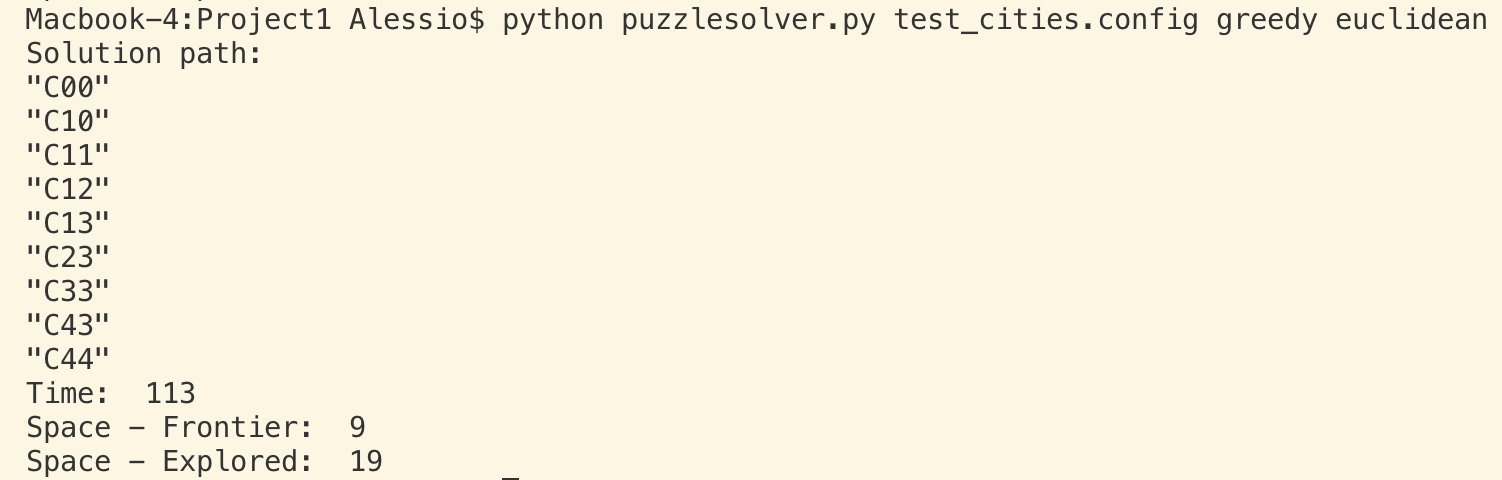
test\_jugs.config with greedy proximity:

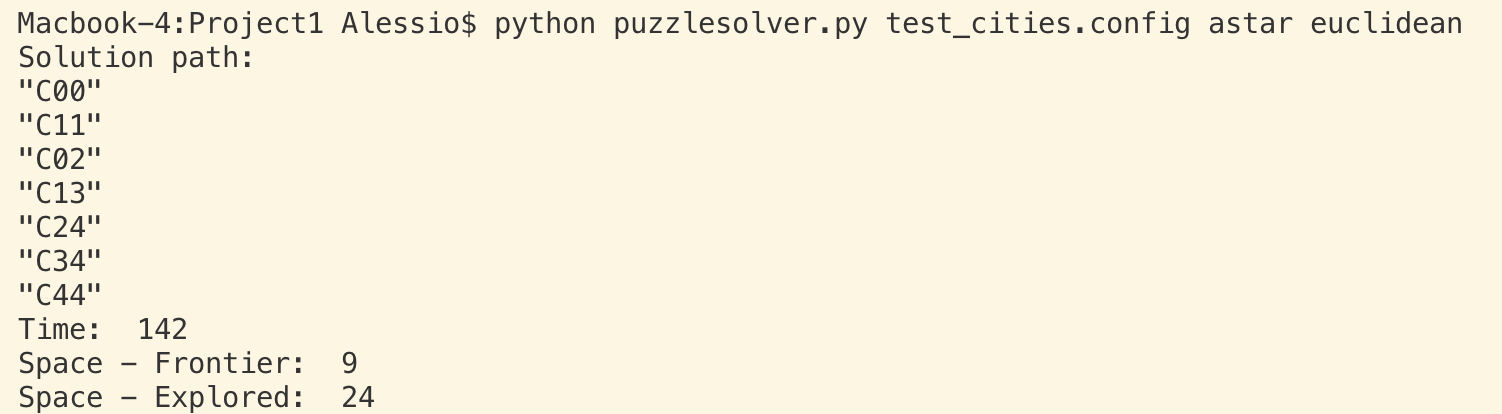
Path Finding Cities:

1. My actions are added to the actions array based on the order the current city appears in the list of edges. Starting from the first edge found on configuration file line 5 and moving down, if the current city is one of the two cities listed in the edge, then that edge is added to the actions array.
2. Heuristic function: “euclidean”

We are given the coordinates of the cities on a grid in the configuration file. The Euclidean heuristic function determines the distance between two cities based on the coordinates treated as (x,y). The distance between the cities is just the Pythagorean theorem, or the Euclidean distance.

test\_cities.config with unicost

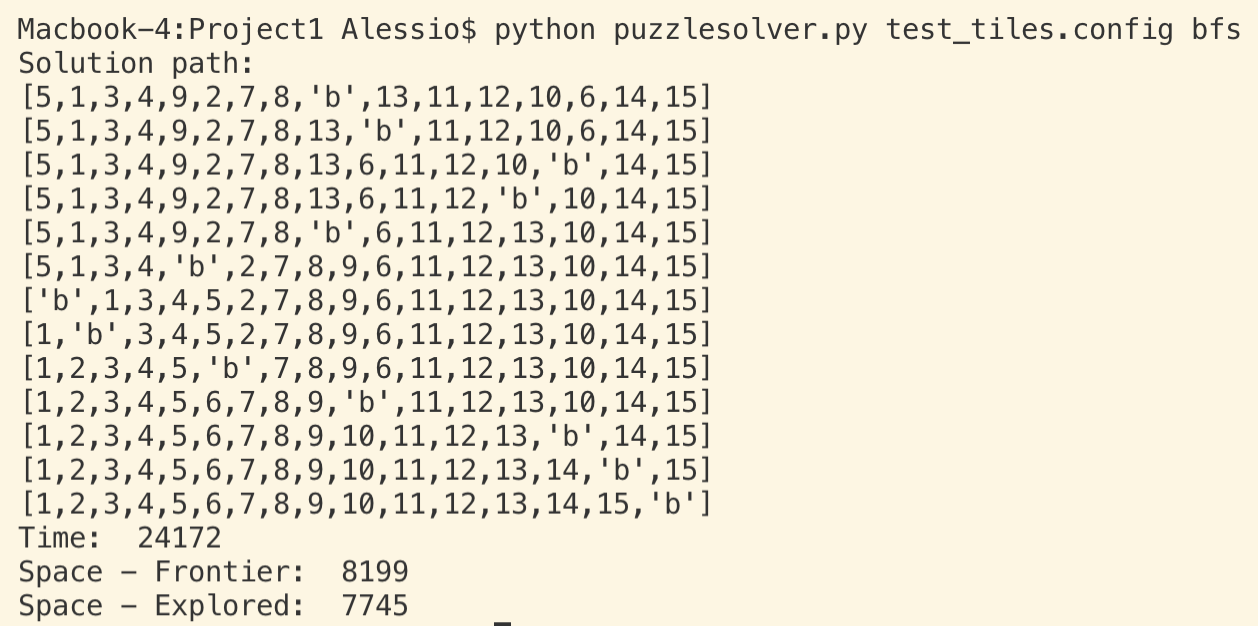
test\_cities.config with greedy euclidean:

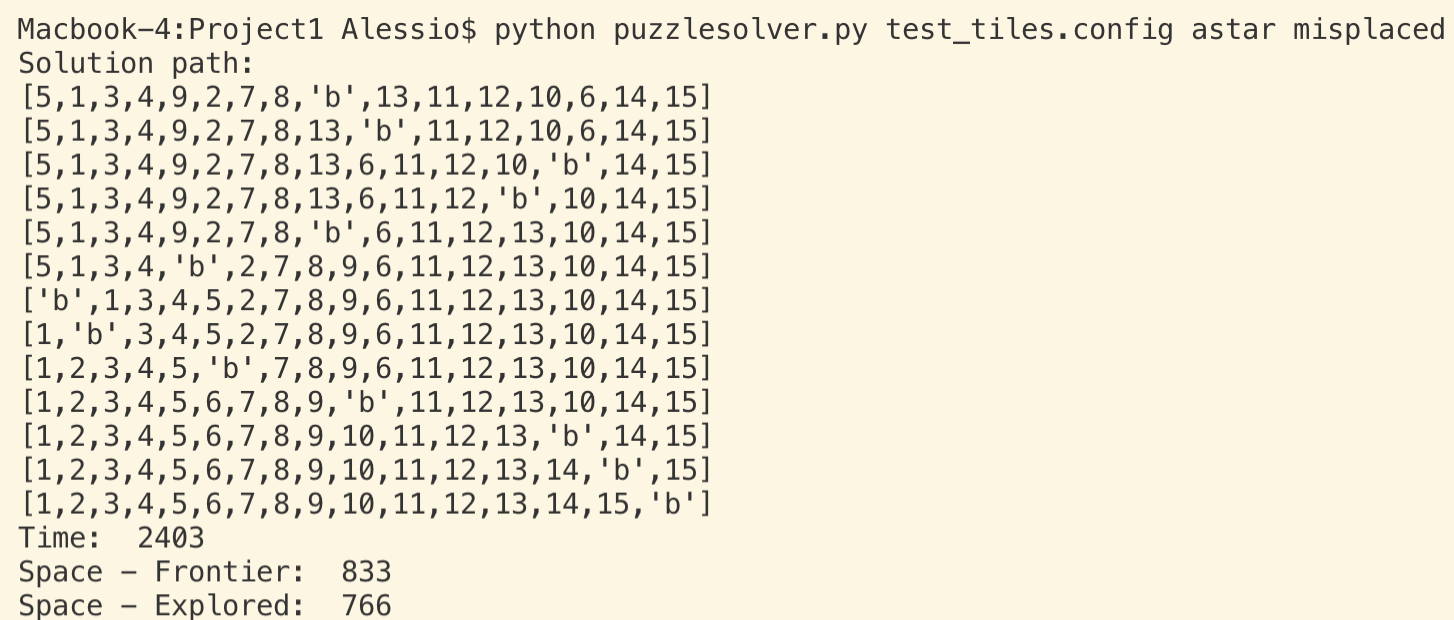
test\_cities.config with astar euclidean:

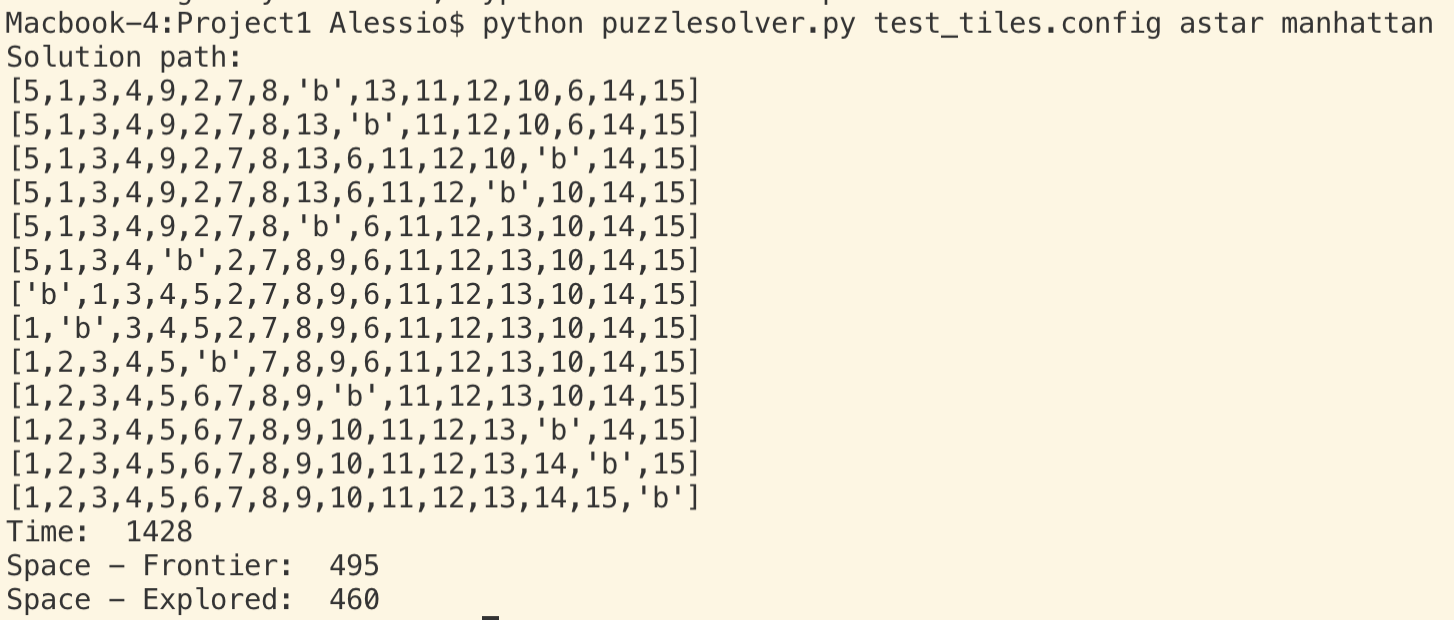
Tiles

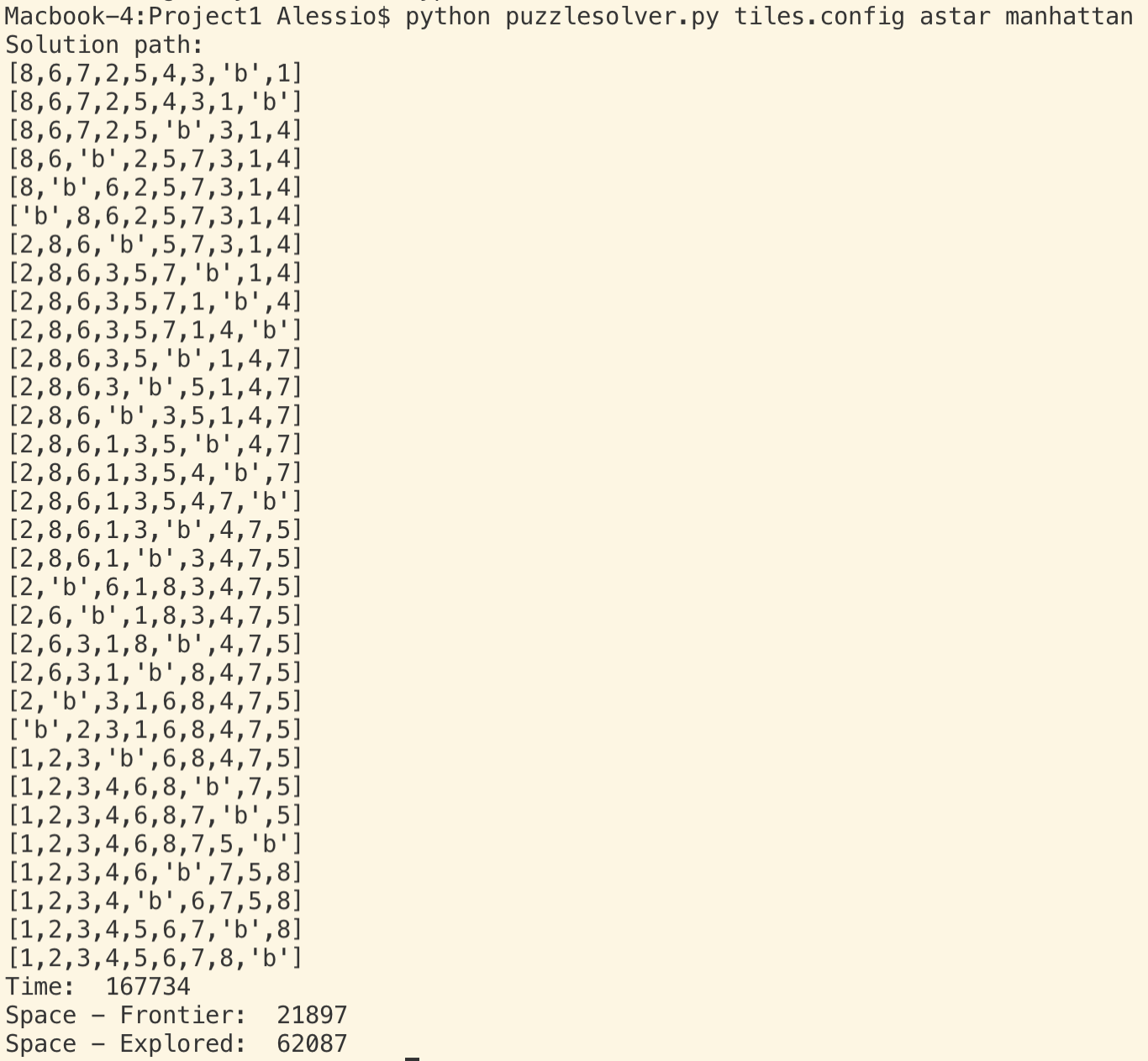
1. With respect to the blank, actions are added to the actions array by swapping blank with tile to its right, up, left, and down.
2. Two heuristic functions used: “misplaced” and “manhattan”

Both heuristics were implemented as explained in class. Misplaced heuristic counts the number of tiles that are not in the correct place with respect to goal. Manhattan heuristic counts how many slides away from where a tile is on goal a tile is currently and sums them together.

test\_tiles.config with bfs

test\_tiles.config with astar misplaced:

test\_tiles.config with astar manhattan:

BFS and astar with Misplaced Tiles Heuristic did not work for tiles.config file within the 30 minute time frame. Only astar manhattan distance was able to finish in under 30 minutes. Results are found below

**5) Additional discussion points: Here, you may want to run some test cases on more search strategies than those required for the transcript.  Address the following issues:**

1. **For each of the three puzzle types, what do you think is the best search strategy, and why? You should take all four factors into considerations (completeness, optimality, time complexity, space complexity).**
2. **For the water jugs problem, is your heuristic admissible and consistent? Explain.**
3. **You may also include additional discussions on any observations that you find surprising and/or interesting.**

a)

For the water jug problem using test\_jugs.config, Based on the data alone, DFS probably performed best. DFS is complete in this case because we have a finite state space, but it is not optimal. The path found by DFS was indeed longer than the all the algorithms by a few nodes, but it saved on space because it did not use an explored list, while the other algorithms did. Further, based on the data, the time it took was almost as good as the Greedy algorithm, and, considering again that it didn’t use an explored list, DFS is probably better still. The space complexity of DFS here is better than all the other algorithms.

For the city planning problem using test\_cities.config, the greedy algorithm using the Euclidean heuristic was the best performer. Greedy performed better than the other algorithms in terms of time and space