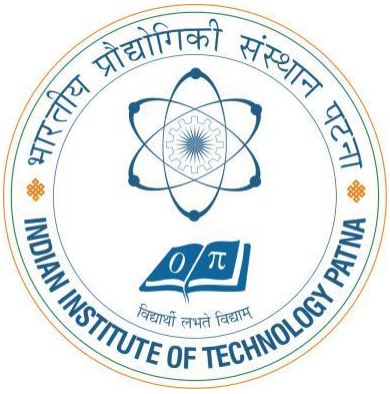
CS571: Artificial Intelligence Lab

Indian Institute of Technology Patna



ASSIGNMENT 4

Hill Climb

Group Members

- Siddhant Kumar (2001CS70)
- Rohit Ranjan (2001CS56)
- Kartik Kailas Mouli (2001CS35)

Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.

Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem.

This solution may not be the global optimal maximum. Here we need to maximize or minimize a given real function by choosing values from the given inputs.

Features of Hill Climbing

1. Variant of generate and test algorithm: It is a variant of generating and test algorithm. The generate and test algorithm is as follows:

- Generate possible solutions.
- Test to see if this is the expected solution.
- If the solution has been found, quite else go to step 1.

Hence, we call Hill climbing a variant of generating and test algorithm as it takes the feedback from the test procedure. Then this feedback is utilized by the generator in deciding the next move in the search space.

2. Uses the Greedy approach: At any point in state space, the search moves in that direction
only which optimizes the cost of function with the hope of finding the optimal solution at the
end.

Algorithms for Hill Climbing:

Step 1: Evaluate the initial state. If it is a goal state, then stop and return success. Otherwise, make the initial state the current state.

Step 2: Loop until the solution state is found or there are no new operators present which can be applied to the current state.

a) Select a state that has not been yet applied to the current state and apply it to produce a new state.
b) Perform these to evaluate new state

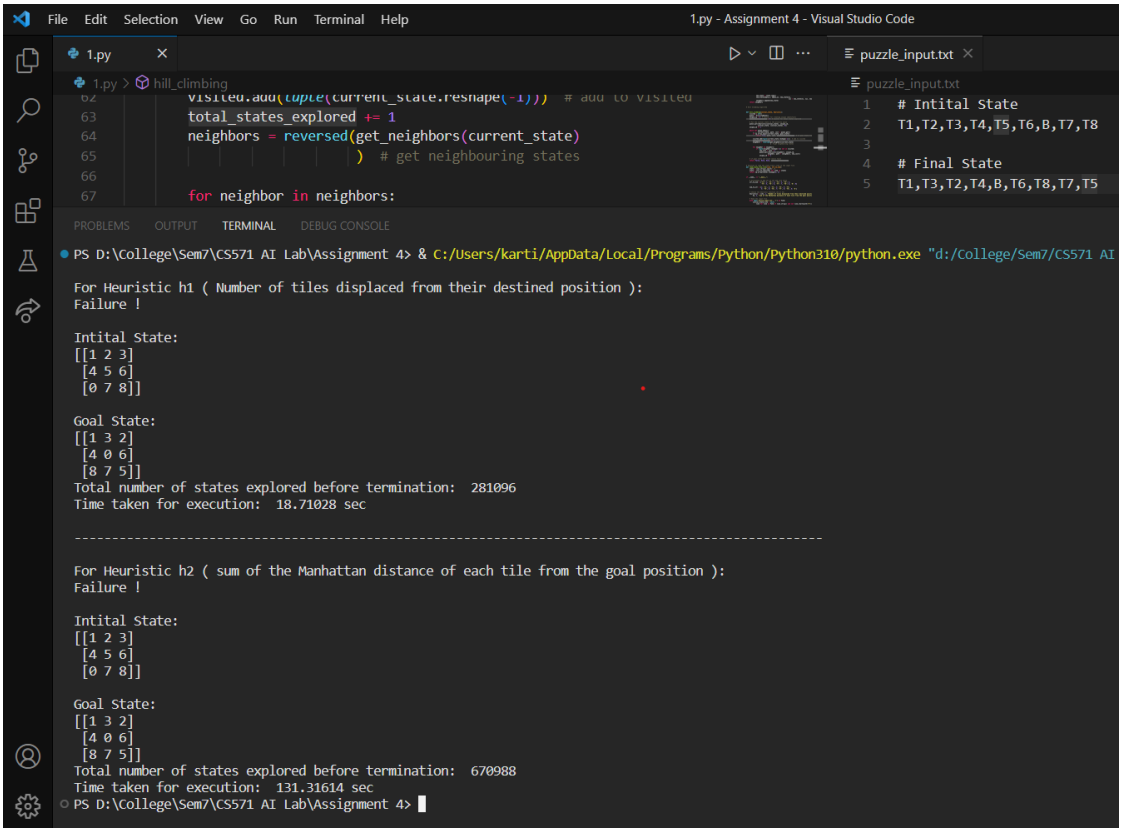
i. If the current state is a goal state, then stop and return to success.
ii. If it is better than the current state, then make it the current state and proceed further.
iii. If it is not better than the current state, then continue in the loop until a solution is found.

Step 3: Exit.

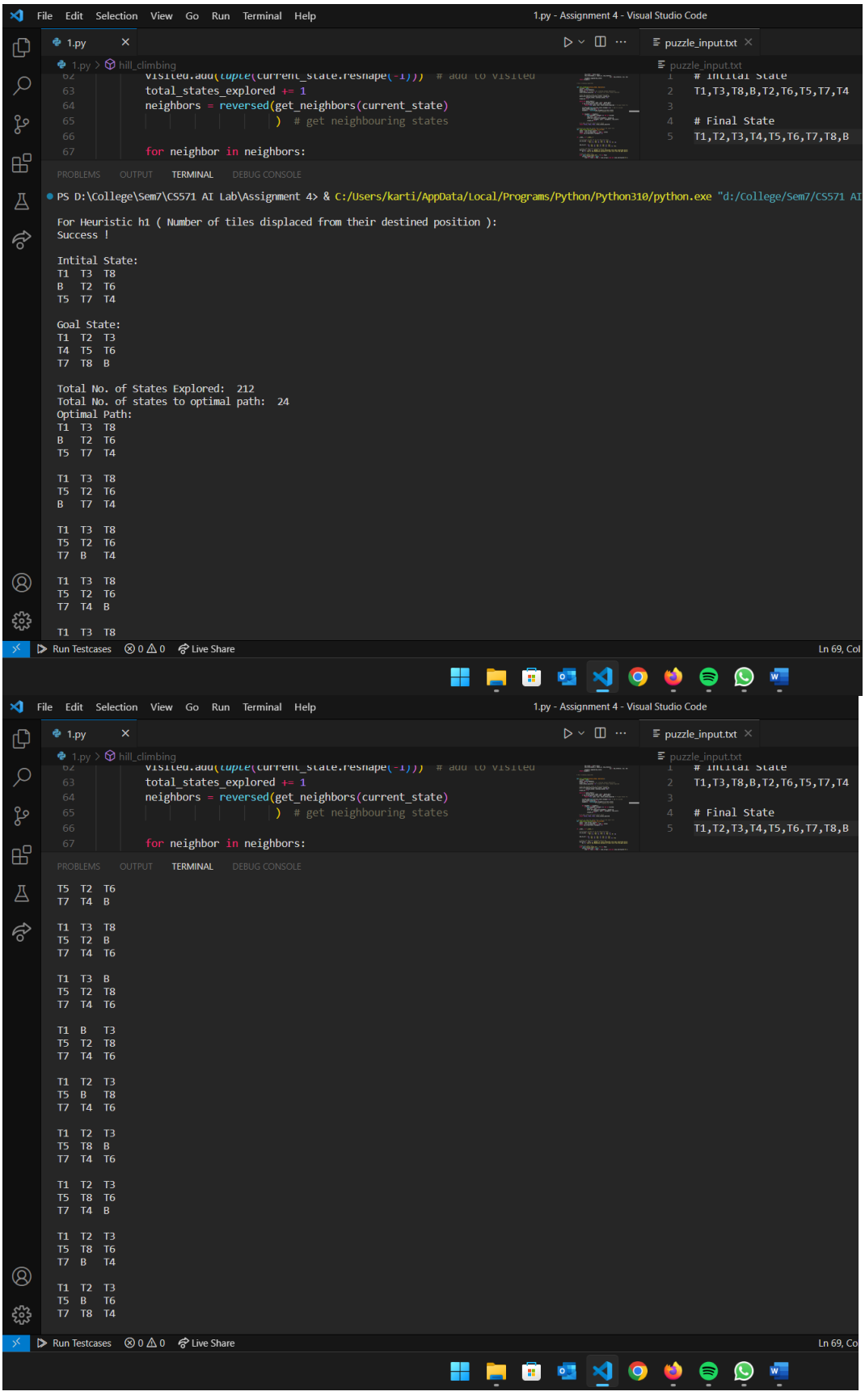
H1(n): number of tiles displaced from their destined position.
It counts no values in the matrix as displaced from their destination position.

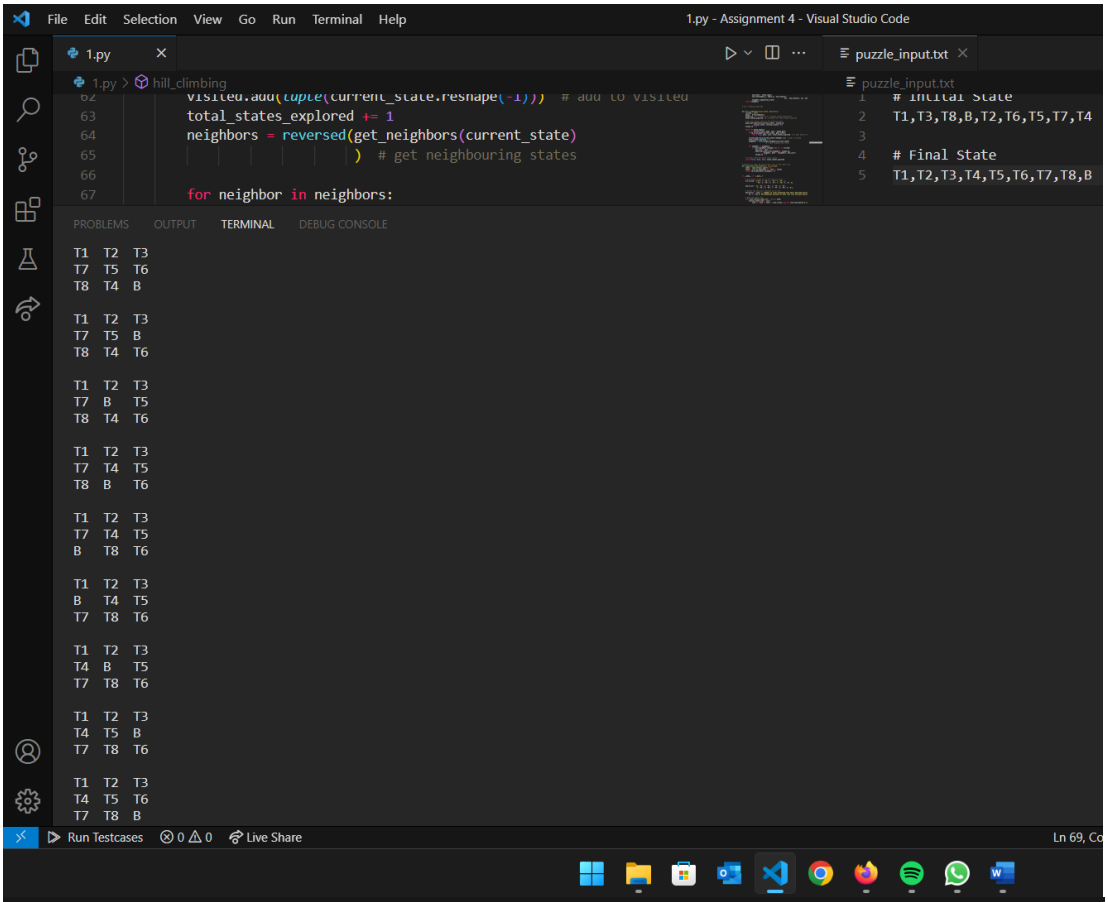
H2(n): sum of the Manhattan distance of each tile from the goal position.
It counts the sum of all values in the matrix and how much is displaced from their destination position.

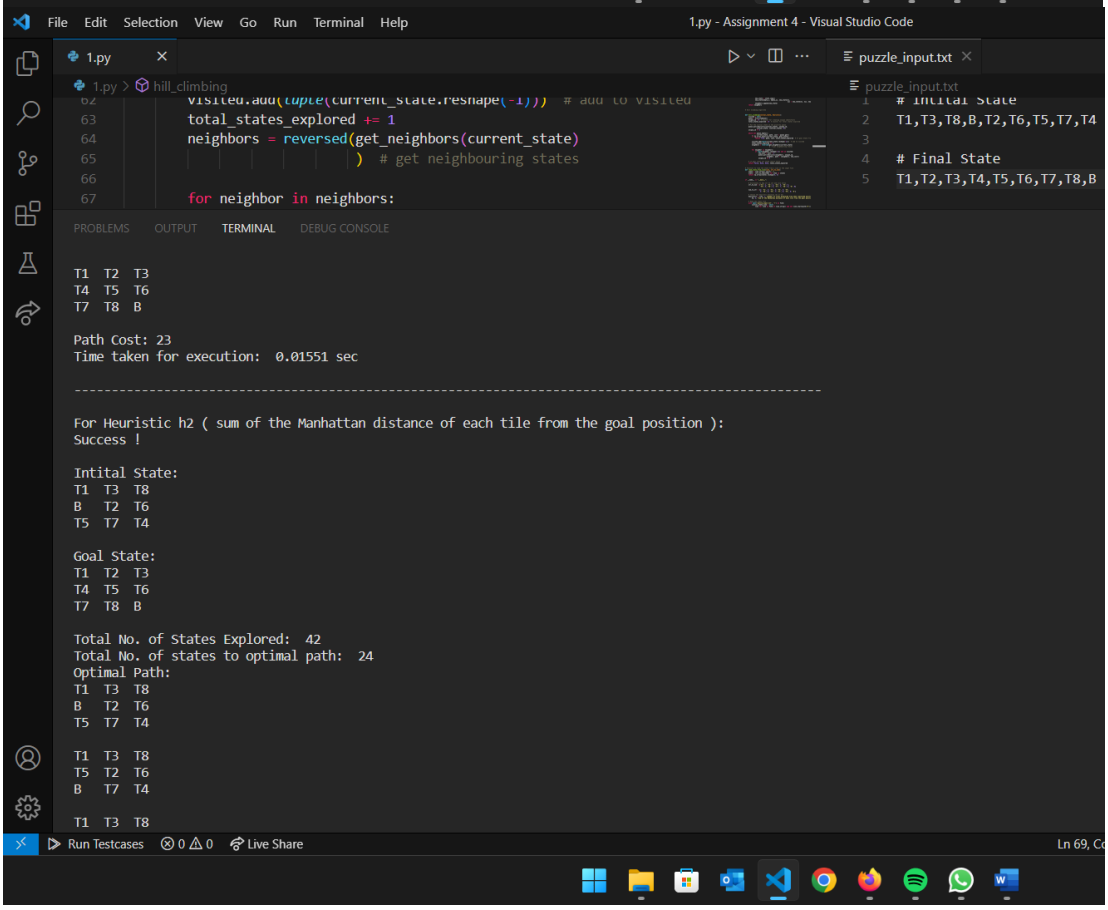
Case 1 (Failure):

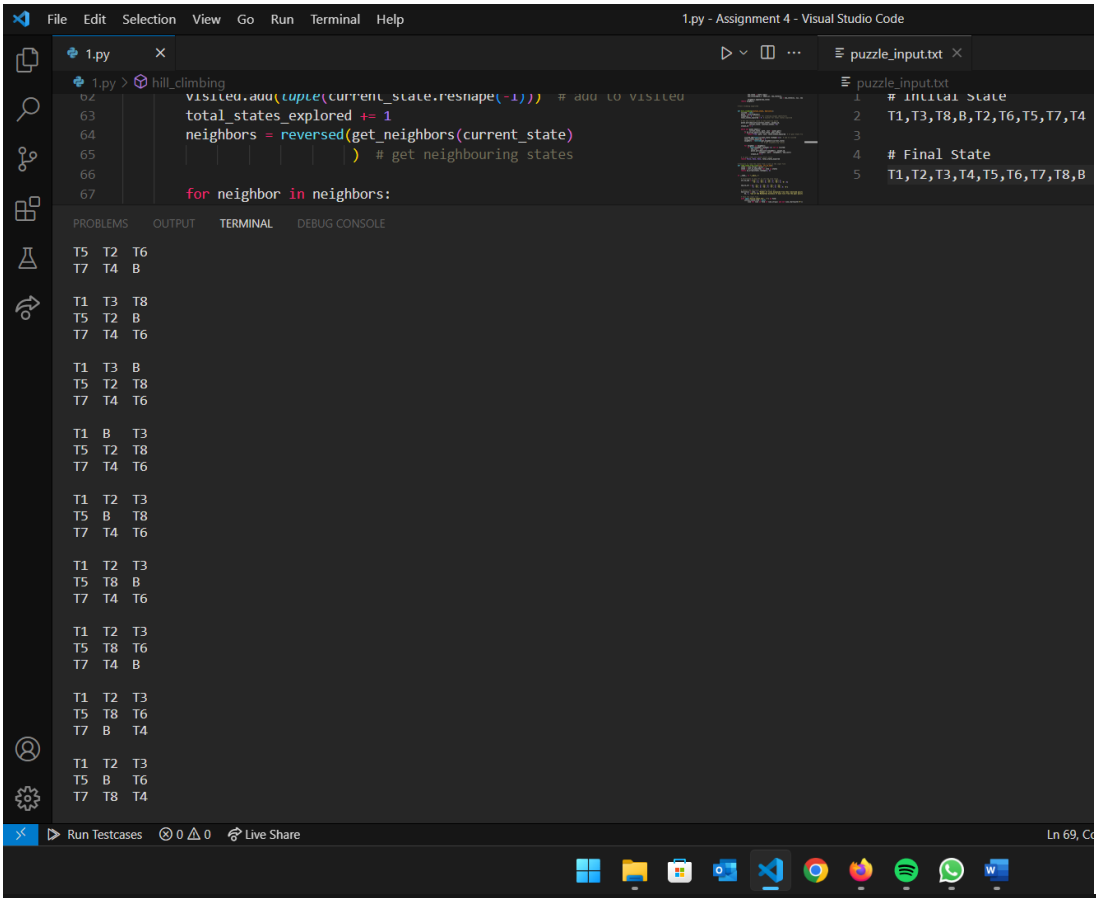


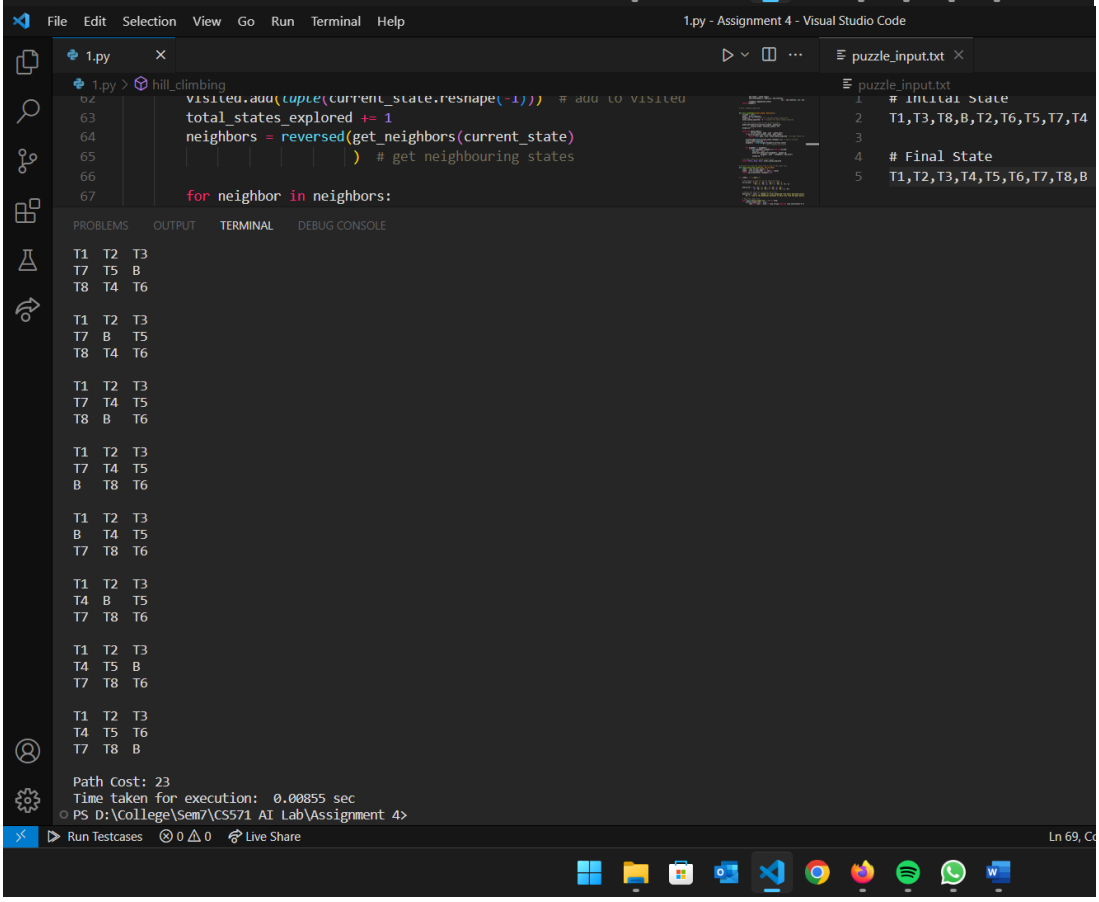
Case 2 (Success):




File Edit Selection View Go Run Terminal Help
1.py - Assignment 4 - Visual Studio Code
1.py
puzzle_input.txt
1.py > hill_climbing
visited.add(tuple(current_state.reshape(-1))) # add to visited
total_states_explored += 1
neighbors = reversed(get_neighbors(current_state)
) # get neighbouring states
for neighbor in neighbors:
puzzle_input.txt
Initial State
T1,T3,T8,B,T2,T6,T5,T7,T4
Final State
T1,T2,T3,T4,T5,T6,T7,T8,B
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
T1 T2 T3
T7 T5 T6
T8 T4 B
T1 T2 T3
T7 T5 B
T8 T4 T6
T1 T2 T3
T7 B T5
T8 T4 T6
T1 T2 T3
T7 T4 T5
T8 B T6
T1 T2 T3
T7 T4 T5
B T8 T6
T1 T2 T3
B T4 T5
T7 T8 T6
T1 T2 T3
T4 B T5
T7 T8 T6
T1 T2 T3
T4 T5 B
T7 T8 T6
T1 T2 T3
T4 T5 T6
T7 T8 B
Run Testcases
0 0
Live Share
Ln 69, Co


File Edit Selection View Go Run Terminal Help
1.py - Assignment 4 - Visual Studio Code
1.py
puzzle_input.txt
1.py > hill_climbing
visited.add(tuple(current_state.reshape(-1))) # add to visited
total_states_explored += 1
neighbors = reversed(get_neighbors(current_state)
) # get neighbouring states
for neighbor in neighbors:
puzzle_input.txt
Initial State
T1,T3,T8,B,T2,T6,T5,T7,T4
Final State
T1,T2,T3,T4,T5,T6,T7,T8,B
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
T1 T2 T3
T4 T5 T6
T7 T8 B
Path Cost: 23
Time taken for execution: 0.01551 sec
For Heuristic h2 (sum of the Manhattan distance of each tile from the goal position):
Success !
Intital State:
T1 T3 T8
B T2 T6
T5 T7 T4
Goal State:
T1 T2 T3
T4 T5 T6
T7 T8 B
Total No. of States Explored: 42
Total No. of states to optimal path: 24
Optimal Path:
T1 T3 T8
B T2 T6
T5 T7 T4
T1 T3 T8
T5 T2 T6
B T7 T4
T1 T3 T8
Run Testcases
0 0
Live Share
Ln 69, Co


File Edit Selection View Go Run Terminal Help
1.py - Assignment 4 - Visual Studio Code
1.py
puzzle_input.txt
1.py > hill_climbing
visited.add(tuple(current_state.reshape(-1))) # add to visited
total_states_explored += 1
neighbors = reversed(get_neighbors(current_state)
) # get neighbouring states
for neighbor in neighbors:
# Initial State
T1,T3,T8,B,T2,T6,T5,T7,T4
Final State
T1,T2,T3,T4,T5,T6,T7,T8,B
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
T5 T2 T6
T7 T4 B
T1 T3 T8
T5 T2 B
T7 T4 T6
T1 T3 B
T5 T2 T8
T7 T4 T6
T1 B T3
T5 T2 T8
T7 T4 T6
T1 T2 T3
T5 B T8
T7 T4 T6
T1 T2 T3
T5 T8 B
T7 T4 T6
T1 T2 T3
T5 T8 T6
T7 T4 B
T1 T2 T3
T5 T8 T6
T7 B T4
T1 T2 T3
T5 B T6
T7 T8 T4
Run Testcases
0 0
Live Share
Ln 69, Co


File Edit Selection View Go Run Terminal Help
1.py - Assignment 4 - Visual Studio Code
1.py
puzzle_input.txt
1.py > hill_climbing
visited.add(tuple(current_state.reshape(-1))) # add to visited
total_states_explored += 1
neighbors = reversed(get_neighbors(current_state)
) # get neighbouring states
for neighbor in neighbors:
# Initial State
T1,T3,T8,B,T2,T6,T5,T7,T4
Final State
T1,T2,T3,T4,T5,T6,T7,T8,B
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
T1 T2 T3
T7 T5 B
T8 T4 T6
T1 T2 T3
T7 B T5
T8 T4 T6
T1 T2 T3
T7 T4 T5
T8 B T6
T1 T2 T3
T7 T4 T5
B T8 T6
T1 T2 T3
B T4 T5
T7 T8 T6
T1 T2 T3
T4 B T5
T7 T8 T6
T1 T2 T3
T4 T5 B
T7 T8 T6
T1 T2 T3
T4 T5 T6
T7 T8 B
Path Cost: 23
Time taken for execution: 0.00855 sec
PS D:\College\Sem7\CS571 AI Lab\Assignment 4>
Run Testcases
0 0
Live Share
Ln 69, Co

Conclusion:

The effectiveness of the Hill Climbing algorithm depends on the choice of heuristic function. H1, which counts the number of displaced tiles, performed well in finding optimal solutions, while H2, which calculates the Manhattan distance, was less successful in some cases.

Hill Climbing is a local search algorithm, and its success in finding optimal solutions is highly dependent on the initial state and the chosen heuristic. Further exploration and experimentation may be necessary to determine the optimal heuristic for specific problem domains.