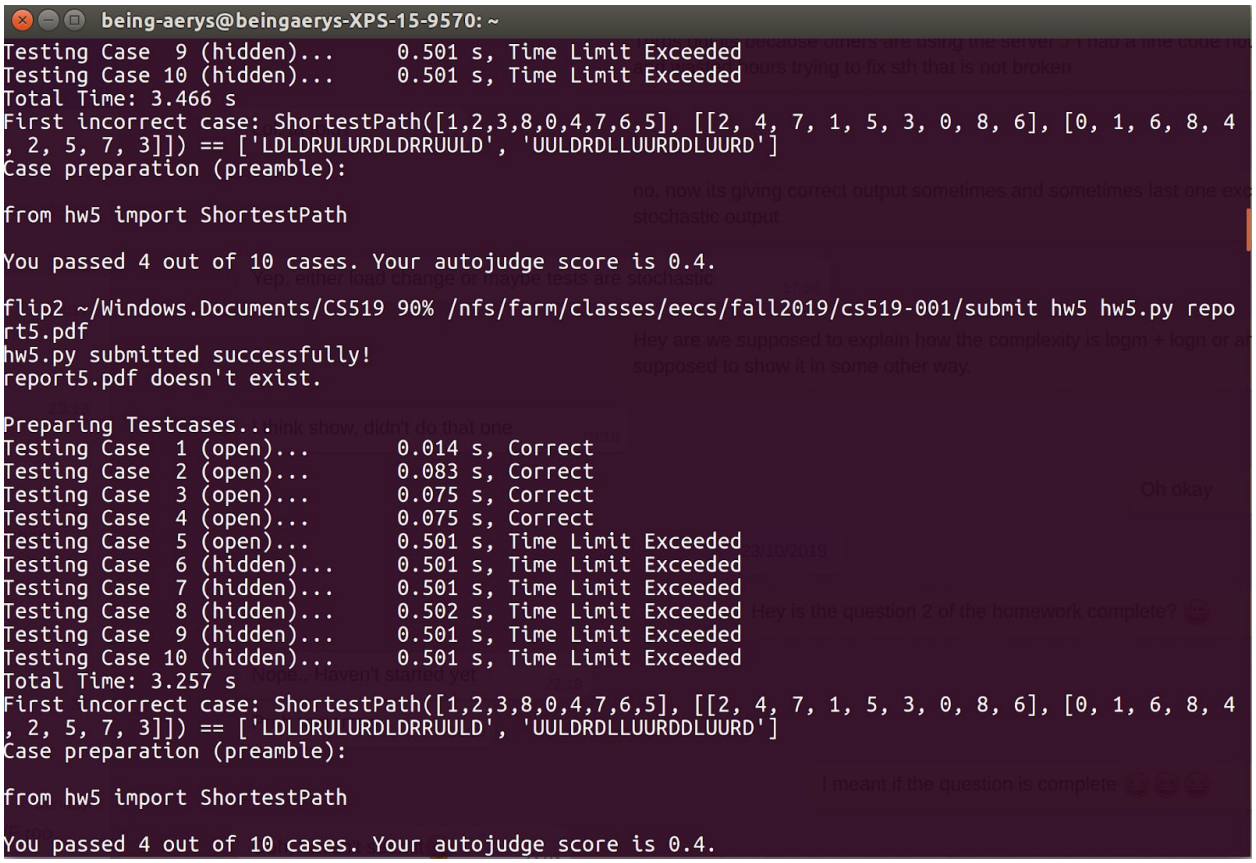
CS 519 Design and Analysis of Algorithms
Fall 2019
Aashish Adhikari

1.(7 points) You will be implementing breadth first search to solve a sliding tile puzzle called 8-puzzle (smaller version of 15-puzzle).

Solution
Implemented in code.
Note: Not sure if server is busy or what, there is a very small margin of 0.001 left for some test cases but I believe my code is correct for all cases given the margin.



2. (1 point) Suppose that the only negative edges are those that leave the starting node s. Does Dijkstra's algorithm find the shortest path from s to every other node in this case? Justify your answer carefully.

Solution:
Dijskstra's algorithm is dependent upon the fact that if we assign the correct distance to a vertex x, then we can assign the correct distance to a vertex y connected to x using this distance from source s to x.

For the sake of simplicity, let us assume that there are 2 negative edges e1 and e2 going out of the source node s. e1 connects s to v1 while e2 connects s to v2. The idea is that if we assign correct shortest paths to the "next-to-source" nodes, Dijkstra's algorithm's proof holds true for the rest of the nodes in the graph.

So let's try to prove that we can assign correct values for the vertices connected to s. Let's assume that the shortest path from s to v1 is not through e1.
i.e., 0 > e1 > e2 + p , where p is a path that leads from v2 to v1 and does not include e1
0 + e1 > e1 + e1 > e1 + e2 + p

This means that there is a negative cycle in the graph which allows for the case that the shortest path from s to v1 could be a path without e1. Hence, in such a scenario, we cannot assign correct values to edges adjoining s and thus the proof for Dijkstra's algorithm does not hold.

However, if the graph had no negative cycles, this would contradict the result above allowing Dijkstra's algorithm to assign correct values to the nodes adjoining the source s.

3.(1 point) Give an $O(n^3)$ algorithm that takes a directed graph as input and returns the length of the shortest cycle in the graph where n is the number of nodes.

Solution:

```
shortest_cycle_len = sys.max
For u in Vertices :
shortest_dist_from_u[ ] = dijkstra(Graph, len_of_edges, u)
        For v in Vertices :
                if (v, u) belongs to E:
                        Shortest_cycle_len =
                                min[shortest_cycle_len, shortest_dist_from_u[v]  + len(v, u)]
if Shortest_cycle_len == sys.max:
        print("The graph has no cycles.")
Else: return Shortest_cycle_len
```

Since we have n nodes and for each node, we are running Dijkstra's algorithm which give a time complexity of $O(n^2)$ in its simplest implementation, time complexity for the above algorithm is $O(n^3)$.

4.(1 point) You are given a strongly connected directed graph G = (V,E) with positive edge weights. Give an efficient algorithm for finding the shortest paths between all pairs of nodes with the restriction that they all must pass through the node A.

Solution

The given restriction is that all the shortest paths should go through A. Thus, the only option we are left with is to divide this problem into 2 shortest-path problems. For any two vertices V1 and V2, the shortest path acceptable must go through A. Note that the graph is strongly-connected. First, we find the shortest path from vertex A to all other vertices using Dijkstra's algorithm. Let this shortest path from vertex A to a vertex V[i] be P[i].Now, we reverse the direction of the directed graph and find the shortest path from all vertices to vertex A. Let this shortest path from a vertex V[j] to vertex A be Q[j]. Then, the shortest path from a vertex V[i] to another vertex V[j] = P[i] + Q[j]. Efficient implementation of this method will give a time complexity O(n + e) * log n) where n is the number of strongly connected vertices and e is the number of edges present.