CSc445–HW#4–Spring18.   Due: Tues, April 2

Instructions.

1. You could submit a pdf of the homework, either printed or hand-written and scanned, as long as it is **easily** readable.
2. If your solution is not clearly written, it might not be graded.
3. Prove the correctness of your answer. A correct answer without a proof might not be accepted.
4. If you have discussed the solution with other students, mention their names clearly on the homework. These discussions are not forbidden and are actually **encouraged**. However, you must write your whole solution yourself.
5. All questions have same weight.
6. You could refer to a data structure studied in class, and just mention briefly their guaranties. For example "*It is known that a Red-Black tree could support the insert/delete/find operations on a set of n elements in time $O(\log n)$.* "
7. If your answer uses one of the data structures or algorithms that were studied in class, you could refer to it without having to repeat details studied in class. If your answer requires only modifications of one of the algorithm, it is enough to mention the required modifications, and what's the effect (if any) on the running time and on other operations that the algorithm performs.
8. In general, a complete answer should contain the following parts:
   (a) High level description of the data structures (if needed). *E.g. We use a binary balanced search tree. Each node contains, a key and pointers to its children. We augment the tree so each node also contains a field...*
   (b) High level description of the algorithms
   (c) Proof of correctness (why your algorithm provides what is required).
   (d) A claim about the running time, and a proof showing this claim.
9. When discussing trees, do not confuse the *children* of a node with the *descendants* of the node.

1. Run the simulations of Bellman-Ford, and of Warshall-Floyd, and the algorithm presented in class for topological sort a DAG. Compare you solutions to the examples in the textbook. You do not need to submit anything in this question.

2. Let $G(V, E)$ be a given graph, let s be a vertex in V, and and let k be a number (not known to you) but with the property that for every pair of vertices $u, v \in V$, the shortest path $u \to v$ contains at most k vertices.

   Suggest a modification of Bellman-Ford algorithm, such that the output is the same as the original algorithm,(the distance $\delta(s, v)$ for every $v \in V$) but the running time is improved to $O(km)$. Prove the correctness of your modification.

3. In the previous homework you were asked to provide an example of a graph $G(V, E)$ with negative weights on its edges, for which Dijkstra outputs wrong results.

   Run Bellman-Ford on the very same example.

4. Let n be a parameter given to you. Create a connected graph $G(V, E)$ with n vertices, and $n - 1$ edges, for which, for at least one vertex $v \in V$, the following two conditions are simultaneously true. During the execution of Bellman-Ford algorithm.

   (a) $d[v] = \delta(s, v)$ when the algorithm terminates. And

   (b) $d[v] > \delta(s, v)$ after $\Omega(n^2)$ operation.

5. In an execution of a shortest path algorithm on a graph $G(V, E)$ we usually store with every vertex $v_i \in V$ a cell in array $A[1..n]$, where each cell $A[i]$ has several fields. For example, in C++ terminology, $A[i]$ is a struct, and $A[i].d$ contains the value $d[v_i]$ (for $i = 1..n$).

   Assume that beside storing a few other variables, you could store other data structures. You could access and read the list of edges (e.g. via adjacency list) but you could not modify it.

   (a) Which fields would you store if you aim to run Dijkstra Algorithm and aim for an $\Theta(n^2)$ running time?

   (b) Which fields would you store if you aim to run Dijkstra Algorithm and aim for an $O((n + m) \log n)$ running time?

   (c) Which fields would you store if you aim to run Belman-Ford Algorithm and aim for an $O(nm)$ running time?

6. Let $G(V, E)$ be a DAG, where the weight of each edge is **positive**. Assume that vertices are given to us in a topological order (see slides for a definition). Let s and t be vertices in V. Suggest an $O(n + m)$ algorithm for computing the shortest path from s to t.

7. Let $G(V, E)$ be a DAG, where the weight of each edge is positive. Assume that vertices are given to us in a topological order (see slides for a definition). Let s and t be vertices of V. Suggest an $O(n + m)$ algorithm for computing the **longest** path from s to t. Prove the correctness of your algorithm.

8. Let $S = \{s_1 \dots s_n\}$ be a set of vertical segments in the plane. See Figure 1. Each segment s_i is given by its x-coordinate x_i and two values y_i, Y_i where (x_i, y_i) is the lower point of the segment s_i and (x_i, Y_i) is the upper point of this segment. Also given to each segment a unique color. Each segment is either **blue**, **orange** or **black**.

   You are also given a list E of pairs of endpoints of the segments (p_i, q_j), such that if $(p_i, q_i) \in E$ if and only if p_i sees q_j. That is, the segment connecting them does not cross any segment of S.

   Suggest an algorithm that in $O(n+m)$ time finds the shortest path that starts at an endpoint of a blue segment, ends at an endpoint of orange segment, and does not cross any other segment (from any color).

   Here $m = |E|$.

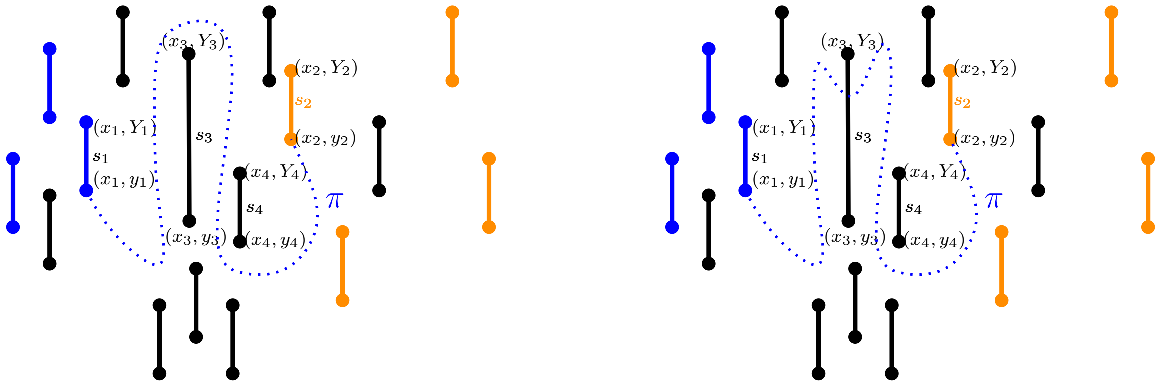

Figure 1: Left: An example of the input to the problem, and a possible path π from a point on a blue segment to a point on an orange segment. This path does not cross any input segment. Right - the same, but now the path crosses one of the segments

9. Suggest a practical approach for answering shortest path queries between street addresses in the USA. Each such query specifies a source address and a destination address. Your goal is to pre-process the road map into a reasonable size data-structure that would enable answering such queries efficiently.

   To clarify possible doubt: Assume the number of houses is about $125M$, so a data structure of size proportional to all roughly $(125M)^2$ is not considered reasonable.

10. Explain how from the output of Warshall-Floyd algorithm you could deduce if a graph contains a negative cycle.

11. You are given the output of Warshall-Floyd algorithm for a graph $G(V, E)$, and you are also given 2 vertices $v_i, v_j \in V$. Explain how you could find the shortest path $p_i \to p_j$ in optimal time. Prove the optimality of your proposal.