Instructions.

1. Solution may **not** be submitted by students in pairs.

2. You may submit a pdf of the homework, either printed or hand-written and scanned, as long as it is **easily** readable.

3. If your solution is illegible and not clearly written, it might not be graded.

4. Unless otherwise stated, you should prove the correctness of your answer. A correct answer without justification may be worth less.

5. If you have discussed any problems 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.

6. Unless otherwise specified, all questions have the same weight.

7. You may refer to data structures or their properties studied in class without having to repeat details, and may reference theorems we have studied without proof. If your answer requires only modifications to one of the algorithms, it is enough to mention the required modifications, and the effect (if any) on the running time and on other operations that the algorithm performs.

8. In general, a complete solution should contain the following parts:

   (a) 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) A clear description of the main ideas of the algorithm. You may include pseudocode in your solution, but this may not be necessary if your description is clear.

   (c) Proof of correctness (e.g. show that your algorithm always terminates with the desired output).

   (d) A claim about the running time, and a proof showing this claim.
Questions.

In all the questions considering graphs, we denote \( n = |V| \) and \( m = |E| \).

Unless otherwise specified, assume that \( G(V,E) \) is given. With every edge \((u,v) \in E\) we are given a weight \( w(u,v) \). Also assume that a vertex \( s \in V \) as specified as the source, and \( t \in V \) is the destination. The distance \( \delta(u,v) \) is as defined in the slides.

1. Assume that in a graph \( G(V,E) \), the weight of each edge \((u,v)\) is an integer between 1 and 5. Compute \( \delta(s,t) \) in time \( O(m + n) \).

2. In the context of the previous question, how would you also find the vertices which are on the shortest path \( s \rightarrow t \)?

3. As in Question 1 assume again that the weight of every edge is an integer between 1 and 5. Furthermore assume \( \delta(s,t) = 100 \). You run Dijkstra algorithm on this graph. Consider the heap when \( t \) is deleted from the heap. At this moment, \( d[t] = 100 \). What are the possible values of \( d[v] \) for other vertices in the heap at this instance?

For example you could argue (incorrectly) that the heap only contains keys \{103.5, 97, 40, 3.1415926, \( \infty \} \). In this case you have to explain why other values are not included.

4. Given an example where \( G(V,E) \) is directed, has edges with negative weights, and the result of running Dijkstra algorithm on this graph yields wrong results. That is \( d[t] \neq \delta(s,t) \).

5. Let \( G(V,E) \) be a directed graph where \( V = \{v_1 \ldots v_n\} \), there are exactly \( n - 1 \) edges \( \{e_1 \ldots e_{n-1}\} \), and \( e_i \) connects \( v_i \) to \( v_{i+1} \) (for every \( i \)). See Figure 1. Assume \( v_1 \) is the source. You run the Bellman-Ford algorithm on this graph. As you recall, we did not specify in which order the algorithm accesses the edges of the graph. After the last iteration of the outer loop of the algorithm, \( d[t] = \delta(s,t) \).

![Figure 1: An example of the graph of Question 5](image)

Show an example of this order of the edges for which \( d[v_n] \neq \delta(v_1,v_n) \) during every iteration excluding the last iteration of the algorithm.

Repeat this question, but now show an order of the edges that is different, and \( d[v_n] = \delta(v_1,v_n) \) after the first iteration of the (outer loop of the) algorithm.

6. The Bellman-Ford algorithm claims to indicate whether the graph \( G(V,E) \) contains a negative cycle. Is it possible that the algorithm will fail to find such a cycle because this cycle is not on a shortest path from \( s \) to some vertex?

7. Modify the Bellman-Ford Algorithm so it stops once the \( d[t] = \delta(s,t) \). Note that this is easier said than done, since \( d[t] \) is not known to you. Note also that the algorithm should indicate if a negative cycle exists.

Demonstrate the efficiency of the modified algorithm by proving that if \( G(V,E) \) is a graph with the property that between every pair of vertices \( u, v \in V \) the shortest path contains at most 5 edges, then the running time is only \( O(m + n) \).

Note that this assumption does not mean that neither \( m \) nor \( n \) is particularly small.

2
8. Show an example of a graph $G(V, E)$ with \textbf{positive} weights on its edges. Design this graph such that during the execution of the Dijkstra algorithm on your graph, $d[t]$ is modified at least 17 times.

9. Let $G(V, E)$ be a directed graph, with positive weights given for its edges, and let $A, B \subseteq V$ be subsets of $V$, given to you. That is, for every vertex $v \in V$ we are given a label specifying if it belongs to $A$, belongs to $B$, or neither.

Suggest an algorithm that computes

$$\min_{a \in A} \min_{b \in B} \delta(a, b) = \{\delta(a_i, b_j) \mid a_i \in A, b_j \in B\}$$

You could visualize this question as follows: When we ask “what is the distance from Tucson to Phoenix”, we will usually means the smallest distance between any two locations, one in Tucson and one in Phoenix.

In other words, if running time could have been ignored, we could have run Dijkstra algorithm multiple times, once from each $a_i \in A$, and compute its distance to each $b_j \in B$. Finally we would have reported the smallest such distance. However this would require time $O(|A| \cdot m \log n)$.

The running time of your answer should be $O((m + n) \log n)$. 


10. Let \( S = \{s_1 \ldots s_n\} \) be a set of vertical segments in the plane. See Figure 2. 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.

So the input for your algorithm is a file with \( n \) rows, each containing 3 numbers. You could assume that no two segments have the same \( x \)-coordinates. In addition, your input contains a list \( L \) of pairs of endpoints \((p_i, q_j)\) such that \( p_i \) sees \( q_j \). Here we say that two points \( p, q \) see each other iff the line segment connecting them does not intersect any segment of \( S \).

Suggest an \( O(n^2 \log n) \) time algorithm for finding how you could place a rope \( \pi \) connecting \((x_1, y_1)\) to \((x_n, y_n)\). This rope could not cross any segment, and should be as short as possible. See Figure 3.

![Figure 2: An example of an input to the problem. The set \( S = \{s_1, s_2, s_3, s_4\} \) of four segments is given. In this example, the point \((x_1, Y_1)\) sees \((x_3, y_3)\). The point \((x_1, Y_1)\) does not see \((x_2, y_2)\).](image1)

![Figure 3: Left: A possible position of the rope \( \pi \) from the point \((x_1, y_1)\) to the point \((x_2, y_2)\) is shown in dotted purple. This path does not cross any segment of \( S \). Note however that this is not the shortest such position of the rope. Right: The path \( \pi' \) (in red) is not allowed, since it crosses \( s_3 \).](image2)

**Comment:** This problem is a basic robot Path-Planning problem. Think about the segments
of $S$ as obstacles in a room, and think about $\pi$ as the trajectory that your Roomba robot needs to follow, in order to move from one point to another. Multiple-axis robots might face similar path-planning problems, but the paths might be in a $d$-dimensional space, rather than a 2D plane.