You're expected to work on the discussion problems before coming to the lab. Discussion session is not meant to be a lecture. TA will guide the discussion and correct your solutions if needed. We will not release 'official' solutions. If you're better prepared for discussion, you will learn more. TAs will record names of the students who actively engage in discussion and report them to the instructor; they are also allowed to give some extra points to those students at their discretion. The instructor will factor in participation in final grade.

1. (Basic) Explain the "optimality of subpaths" lemma.

   Sol. Any subpath of a shortest path must be a shortest path between the two ending points of the subpath.

2. (Intermediate) Suppose every edge has weight 1. To compute a shortest path from s to another vertex u, what algorithm would you like to use?

   Sol. We can run BFS starting from s. The running time is $O(V + E)$, which is better than using Dijkstra; recall that Dijkstra's running time is $O(E \log V)$ using a min priority queue based on binary heap.

3. (Basic) Consider the graph in Fig 24.4. We would like to run the Bellman-Ford algorithm with s as the starting vertex. Suppose that in each iteration we relax edges in a lexicographic order – (u, v) is considered before (u', v') if u is ahead of u' in alphabetical order, or $u = u'$ and v is ahead of v' in alphabetical order. Show how $v.d$ and $v.\pi$ are updated during the execution of the algorithm.

   Sol. Omitted.

4. (Basic) Consider the graph in Fig 24.4. Run the Dijkstra algorithm with s as the starting point. Does it correctly compute a shortest path from s to each vertex?

   Sol. Omitted.

5. (Basic) Consider the graph in Fig 24.6. Run the Dijkstra algorithm with z as the starting point.

   Sol. Omitted.

6. (Basic) What is the limitation of the Dijkstra algorithm compared to the Bellman-Ford?

   Sol. Dijkstra can't handle negative weight edges.

7. (Advanced) We are given a directed graph with positive weight on edges. We're also given two disjoint subsets S and T of vertices. We would like to find a shortest path from any vertex in S to any vertex in T. What is your algorithm? The faster, the better.

   Sol. Create a super source s. Connect s to every vertex in S with weight 0. Run Dijkstra with s as the starting point. At the end, we choose the vertex $t \in T$ with the minimum $t.d$, and return a shortest path from s to t with the first edge removed.

8. (Advanced) By running the Dijkstra algorithm we obtain a shortest paths tree T with the starting point s as the root. Suppose we deleted an edge (u, v) of the tree from the input graph. To compute a shortest paths tree for this new graph, do we need to run the Dijkstra algorithm from the beginning? Can we partly recycle T?

   Sol. If weremove (u, v) from T, then the tree T will be split into two subtrees; if not, there's nothing to do, and we can recycle the whole T. We can recycle the subtree T_1 including s. We can ignore "inside" edges between any two points of T_1. For all out-going edges from T_1, update $v.d$ for all v not in T_1. So assuming that S is the set of vertices in T_1, (and after relaxing every edge leaving S), we run Dijkstra.

9. (Advanced) We're given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $0 \leq r(u, v) \leq 1$, which is the probability the communication channel from u to v will not fail. Give an algorithm to find the most reliable path between two given vertices. Assume that the events that distinct edges fail are independent.

   Sol. Assume that every edge e has $r(e) > 0$; if not, we can remove such edges. We want to find a path P from u to v such that $\prod_{e \in P} r(e)$ is maximized. In other words, such that $\sum_{e \in P} \log r(e)$ is maximized. This is equivalent to finding a path P from u to v such that $\sum_{e \in P} - \log r(e)$ is minimized. So, pretending that each edge e has weight $-\log r(e) \geq 0$, we run Dijkstra's algorithm to find a shorted path from u to v.