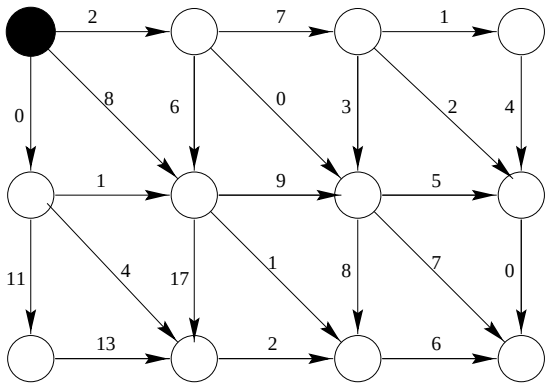
CPS 130 Homework 21 - Solutions

1. Consider a directed weighted graph with non-negative weights and V vertices arranged on a rectangular grid. Each vertex has an edge to its southern, eastern and southeastern neighbors (if existing). The northwest-most vertex is called the root. The figure below shows an example graph with V=12 vertices and the root drawn in black:



Assume that the graph is represented such that each vertex can access **all** its neighbors in constant time.

(a) How long would it take Dijkstra's algorithm to find the length of the shortest path from the root to all other vertices?

Solution: Dijkstra's algorithm has running time $O(E \log V)$. In the graph described above, each vertex has at most three outgoing edges, so that the number of edges in the graph is at most $3V$, that is, $E = O(V)$. In this case, Dijkstra's algorithm will run in $O(V \log V)$.

(b) Describe an algorithm that finds the length of the shortest paths from the root to all other vertices in $O(V)$ time.

Solution: This question is easily answered if you realize the graph is a directed acyclic graph (or dag), since the SSSP problem can be solved on dags in $\Theta(V + E)$ time using e.g. the DAG-SHORTEST-PATHS algorithm. Because in this case we have $E = O(V)$, we can compute SSSP using DAG-SHORTEST-PATHS in $O(V)$.

You can also design your own $O(V)$ algorithm, in which case you must analyze its running time and prove correctness. Here is one example algorithm. The paths from the root to all vertices with in-degree 1 (first row, first column) are unique, so we can find the shortest paths traveling along the path from the root and summing the weights of the edges encountered along the way. Otherwise a vertex u has in-degree 3, that is, there are three predecessors $p_{u_1}, p_{u_2}, p_{u_3}$ of u. If we already know the shortest paths $\delta(r, p_{u_1})$, $\delta(r, p_{u_2})$, $\delta(r, p_{u_3})$ from the root r to the predecessors of u then the shortest path $\delta(r, u)$ from r to u is given by

$$\min\left\{\delta(r, p_{u_1}) + w(p_{u_1}, u),\ \delta(r, p_{u_2}) + w(p_{u_2}, u),\ \delta(r, p_{u_3}) + w(p_{u_3}, u)\right\},$$

where $w(p_{u_{(\cdot)}}, u)$ denotes the weight of the edge from a predecessor of u to u. Thus if all $\delta(p_{u_{(\cdot)}}, u)$ are already known, we can find the shortest path from the root at any vertex u in $O(1)$ time, and we perform this computation once for each vertex. To ensure we perform computations in the correct order (i.e. we know $\delta(r, p_{u_{(\cdot)}})$ before attempting to compute $\delta(r, u)$) we must first perform a topological sort of the vertices.

A topological sort takes time $O(V + E) = O(V)$ and our algorithm performs $O(1)$ work at each vertex, so the total running time is $O(V)$.

We now need to prove our algorithm correctly solves SSSP. For any vertex with in-degree 1 the path from the root is unique and therefore must be the shortest path. For all other vertices, we prove correctness by induction on a vertex v in the topological ordering of the vertices. The first vertex is the root, and the path from the root to itself is zero and therefore the shortest path. Assume that at a vertex v the shortest paths from the root to all vertices before v in the topological order are known. In particular, the shortest paths to all predecessors of v (of which there are three) are known. The only possible paths from the root to v must pass through a predecessor $p_{v_{(\cdot)}}$ of v, and from each predecessor there is only one possible (shortest) path to v (i.e. $\delta(p_{v_{(\cdot)}}, v) = w(p_{v_{(\cdot)}}, v)$, so that the possible shortest paths to v are in the set

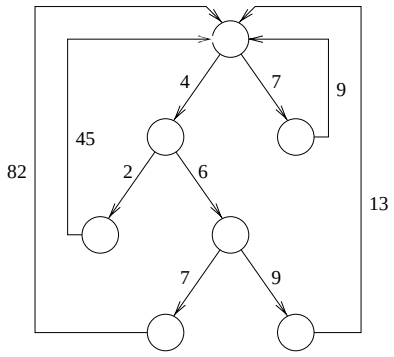
$$\{\delta(r, p_{u_1}) + w(p_{u_1}, u),\ \delta(r, p_{u_2}) + w(p_{u_2}, u),\ \delta(r, p_{u_3}) + w(p_{u_3}, u)\}\,.$$

The minimum value in this set must be the shortest path from r to v.

(c) Describe an efficient algorithm for solving the all-pair-shortest-paths problem on the graph (it is enough to find the length of each shortest path).

Solution: We can solve APSP by computing SSSP for each vertex in the graph. In (b) we gave an $O(V)$ algorithm to solve SSSP, thus we can solve APSP in $O(V^2)$. This is optimal, since there are $O(V^2)$ pairs in the graph.

2. Consider a directed weighted graph with non-negative weights which is formed by adding an edge from every leaf in a binary tree to the root of the tree. Let the graph/tree have n vertices. An example of such a graph with $n = 7$ could be the following:



We want to design an algorithm for finding the shortest path between two vertices in such a graph.

(a) How long time would it take Dijkstra's algorithm to solve the problem?

Solution: Dijkstra's algorithm has running time $O(E \log V)$. In this graph, each vertex has at most two outgoing edges, so that the number of edges in the graph is at most $2V$. Thus we have $E = O(V)$ and Dijkstra's algorithm will run in $O(V \log V)$.

(b) Describe and analyze a more efficient algorithm for the problem.

Solution: We first make some observations: There is exactly one vertex in the graph with in-degree > 1, call it r. We can identify r in $O(V)$ time (CLRS 22.1-1). The shortest path $\delta(s, t)$ between two vertices s and t will then either (i) not pass through r or (ii) pass through r. We can check which of these cases apply in $O(V)$ time using a modified graph search (BFS, DFS) at s which 'ignores' all outgoing edges of vertices with out-degree 1.

Consider case (i). If a path from s to t does not pass through r then it is unique, and moreover we can find it in $O(V)$ time – $\delta(s, t)$ is the sum of edge weights on the path from t back up to s.

Now consider case (ii). We know the path for s to t passes through r, and that the path from r to t is unique. So we only need to find $\delta(s, r)$. This can be done in $O(V)$ using SSSP for directed acyclic graphs – we just 'ignore' the outgoing edges of r and apply the DAG-SHORTEST-PATHS algorithm at vertex s. Although there is only one path from r to t, we still have to find it. But we already know how to do this in $O(V)$ from case (i). Then $\delta(s, t) = \delta(s, r) + \delta(r, t)$.

Every operation we perform is $O(V)$ and we do at most two searches, at most one DAG-SHORTEST-PATHS, and one scan of the graph to identify r. Thus the total running time of our algorithm is $O(V)$.

Our algorithm is clearly correct for case (i) as a unique path between two vertices must be the shortest one. For case (ii), $\delta(s, r) + \delta(r, t)$ must give the shortest path from s to t, because DAG-SHORTEST-PATHS will correctly return the shortest path from s to r and the path from r to t is unique.