Fundamental Algorithms, Lecture Notes. Reductions among Shortest Path Problems

We will consider a variety of shortest path problems all of which can be solved by applying Dijkstra's algorithm.

Example 1. Let $G = (V, E)$ be a directed graph, let $d \in V$ be a destination vertex, and suppose each edge has a non-negative length. We want to compute the length of the shortest path from v to d for each $v \in V$.

To solve this problem we run Dijkstra's algorithm on the graph $G^R = (W, F)$, the reversal of G, in which each edge of G is reversed; i.e. if $(u, v) \in E$ then $(v, u) \in F$; these edges all have the same length as their unreversed versions. d will be the start vertex.

Clearly, a path P from v to d in G has the same length as the reversal of this path in G^R, and therefore the length of the shortest path from d to v in G^R is the length of the shortest path from v to d in G.

Clearly, G^R can be constructed in $O(n + m)$ time. G^R and G have the same size and thus Dijkstra's algorithm takes time $O(n \log n + m)$ when run on graph G^R. Thus, in total, this algorithm takes $O(n \log n + m)$ time (when using the most efficient implementation).

Example 2. Let $G = (V, E)$ be a directed graph, and suppose each edge has a non-negative length. Let $U \subset V$ be a subset of vertices called firehouses. The problem is to find the length of the shortest path in G to each vertex $v \in V$ from the firehouse nearest to v.

To solve this problem, we construct a new graph $H = (W, F)$ on which we will run Dijkstra's algorithm. $W = V \cup \{s\}$, where $s \notin V$ is a new vertex. F contains the edges in E, and keeps their lengths unchanged. In addition, F includes the following length 0 edges: (s, f), for each firehouse $f \in U$.

We then compute the lengths of the shortest paths from s to every vertex in W in the graph H.

We claim that the length of the shortest path to v in H equals the length of the shortest path from the nearest firehouse to v in G. To see this, note that $v_1, v_2 \ldots, v_k = v$ is a path in G from a firehouse to vertex v exactly if $s, v_1, v_2 \ldots, v_k = v$ is a path from s to v in H. Since the edge (s, v_1) has length 0, these two paths are of equal length, and therefore a shortest path from s to v in H has the same length as a shortest path from the nearest firehouse to v in G.

H can be constructed in $O(n + m)$ time; it has $n + 1$ vertices and at most $m + n$ edges. Consequently Disjkstra's algorithm runs in time $O(n \log n + m)$ on H (when using the most efficient implementation).

Example 3. The bottleneck problem. Again, let $G = (V, E)$ be a directed graph, let $b \in V$ be a bottleneck vertex, and suppose every edge has a non-negative length. Suppose you want to be able to answer queries of the following form in $O(1)$ time: What is the shortest path from u to v which goes via b. Show how to preprocess the graph in time proportional to the runtime of Dijkstra's algorithm so as to be able to answer such queries.

We run Dijkstra's algorithm twice, once on graph G with b as the start vertex, and once on graph G^R, again with b as the start vertex. Let $\mathrm{Dist}[1 : n]$ and $\mathrm{DistR}[1 : n]$ be the arrays storing the results. Then the length of a shortest path from u to v of the required type is $\mathrm{DistR}[u] + \mathrm{Dist}[v]$. This is because a shortest path of this sort from u to v consists of a shortest path from u to b, which has the same length as a shortest path from b to u in G^R, plus a shortest path from b to v.

The runtime of this algorithm is $O(n+m)$ to construct G^R, plus twice the runtime of Dijkstra's algorithm on a graph of n vertices and m edges. Thus the preprocessing runs in time proportional to the runtime of Dijkstra's algorithm. Clearly, once the entries in the arrays Dist and DistR have been computed, the queries can be answered in $O(1)$ time.

Example 4. Multi-mode transport. Again, let $G = (V, E)$ be a directed graph, $s \in V$ be the start vertex, and suppose every edge has a non-negative length. In addition, suppose each edge is of one of two types: Rail or Truck (R or T for short). We want to compute the length of a shortest path from s to every vertex $v \in V$, where the path is restricted to beginning on rail edges and ending on truck edges (it's possible there are no rail and/or no truck edges on such path).

We build the following graph H. It comprises two copies of the vertex set V, denoted V^r and V^t. The rail edges are added between the vertices in V^r, i.e. if (u, v) is a rail edge, then we add an edge (u^r, v^r) to H; similarly, the truck edges are added between the vertices in V^t. In addition, for each vertex $v \in V$ we add a length 0 edge (v^r, v^t) to H.

We run Dijkstra's algorithm on H with vertex s^r as the start vertex. The length of a shortest path to vertex v^t is the length of the desired shortest path to v in G. To see this note that each path from s^r to v^t comprises an initial portion among the vertices in V^r, $s^r = v_1^r, v_2^r, \ldots v_j^r$ from s^r to some vertex v_j^r, followed by a zero length edge (v_j^r, v_j^t), followed by a path from v_j^t to v_k^t among the vertices in V^t, $v_j^t, \ldots, v_k^t = v^t$, say. But this is true exactly if $s = v_1, v_2, \ldots, v_k = v$ is a path from d to v in G, which switches from rail to truck at vertex v_j.

H has $2n$ vertices and $2m + n$ edges. H can be computed in $O(n + m)$ time. Thus the runtime of the above algorithm, up to constant factors, is the same as for Dijkstra's algorithm on n vertex, m edge graphs, which is $O(n \log n + m)$ when using the most efficient implementation of Dijkstra's algorithm.

Example 5. Colored edges. Again, let $G = (V, E)$ be a directed graph, $s \in V$ be the start vertex, and suppose every edge has a non-negative length. In addition, suppose each edge has a color, either Red or Blue (R or B for short). We want to compute the length of shortest color alternating paths from s to every vertex v. By a color alternating path we mean one on which every pair of adjacent edges have different colors, i.e. the form is RBRB $\ldots$ or BRBR $\ldots$.

We build the following graph H. It comprises two copies of the vertex set V, denoted V^r and V^b, plus one more vertex z, which will be the start vertex in H. Being at a vertex in V^b will indicate that the next edge on the corresponding path in G is blue, and being at a vertex in V^r will indicate that the next edge is red. Accordingly, we add edges to H as follows. For each blue edge $(u, v) \in E$ of length ℓ we add edge (u^b, v^r) of length ℓ to H, and for each red edge $(w, x) \in E$ of length ℓ we add edge (w^r, x^b) of length ℓ to H. In addition, we add length 0 edges (z, s^b) and (z, s^r) to H. Clearly, any path in H alternates between vertices in V^b and V^r.

For each alternating path $s = v_1, v_2, \ldots, v_k = v$ in G that begins with a blue edge, the following path in H has the same length: $z, v_1^b, v_2^r, v_3^b, \ldots$ (the path ends at v_k^b if it is of odd length, and at v_k^r if it is of even length). A similar statement applies to paths that begin with a red edge. It follows that the length of a shortest alternating path from s to v in G has length equal to the minimum of the shortest paths from z to v^r and from z to v^b in H.

Computing H takes $O(n+m)$ time, running Dijkstra's algorithm on H takes time $O(n \log n+m)$ (when using the most efficient implementation), and reporting the answers to the original problem takes an additional $O(n)$ time. This is a total of $O(n \log n + m)$ time.

We now summarize the approach a bit more abstractly. We suppose we are given a problem instance x whose solution is $f(x)$ for some function f that we want to compute. The proposed algorithm will have the following form.
Step 1. Construct an instance $y = g(x)$ of the single source shortest problem (or sometimes two of more instances $y_1 = g_1(x), y_2 = g_2(x), \ldots$).

Step 2. Solve the instance(s) obtaining solution $z = h(y)$ (or $z_1 = h(y_1), z_2 = h(y_2), \ldots$).
Step 3. Compute the answer to the original problem: $f(x) = k(z)$ (or $f(x) = k(z_1, z_2, \ldots)$.

The algorithmic task is to find the algorithms that compute the functions g and k; these functions are not given; you have to discover them. h is simply the function computed by Dijkstra's algorithm.

There are still a couple of tasks remaining.
Task 1. Analyze the running of the above algorithm.
Task 2. Argue that $h(z)$ is the correct answer to the problem.

This solution method is called a reduction (from the new problem to the single shortest path problem). Reductions are not limited to the single source shortest path problem, of course. Reductions provide a systematic way of using an existing algorithm to solve similar problems. Not infrequently, the similarity is clear only after finding the reduction. Essentially, a reduction encodes an instance of the new problem as a instance of the target problem, which is single source shortest path here. Often, our constructions are using different subsets of the vertices to encode different properties of the paths in our new problem; we then add edges so as to enforce these properties.