Lecture Notes #18

Dijkstra's Algorithm for Single-Source Shortest Paths

Assumption: all edges have non-negative weight.

procedure *Dijkstra*(G, w, s)
{
    Initialize-Single-Source(G, s)
    $S := \emptyset$;
    $Q := V$;
    while $Q \neq \emptyset$ **do**
    {
        $u :=$ *Extract-Min*(Q);
        $S := S \cup \{u\}$;
        for each node $v \in Adj[u]$ **do** *Relax*(u, v, w);
    }
}

Correctness of the Algorithm

Let $\delta(s, v)$ be defined as in Lecture Notes # 16.

Let $P = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v_m$ be a path. The nodes in $\{v_1, v_2, \cdots, v_{j-1}\}$ are called the predecessors of v_j in P.

Define the shortest path P from s to v such that all predecessors of v are in S as *the shortest path from s to v with respect to S.*

Lemma 1 *Right after the j-th iteration of the while-loop of Dijkstra, $d[v']$, $v' \in Q$, is the weight of the shortest path (SP) from s to v' with respect to S. Furthermore, for u in Q such that $d[u] = \min\{d[z] | z \in Q\}$, $d[u] = \delta(s, u)$.*

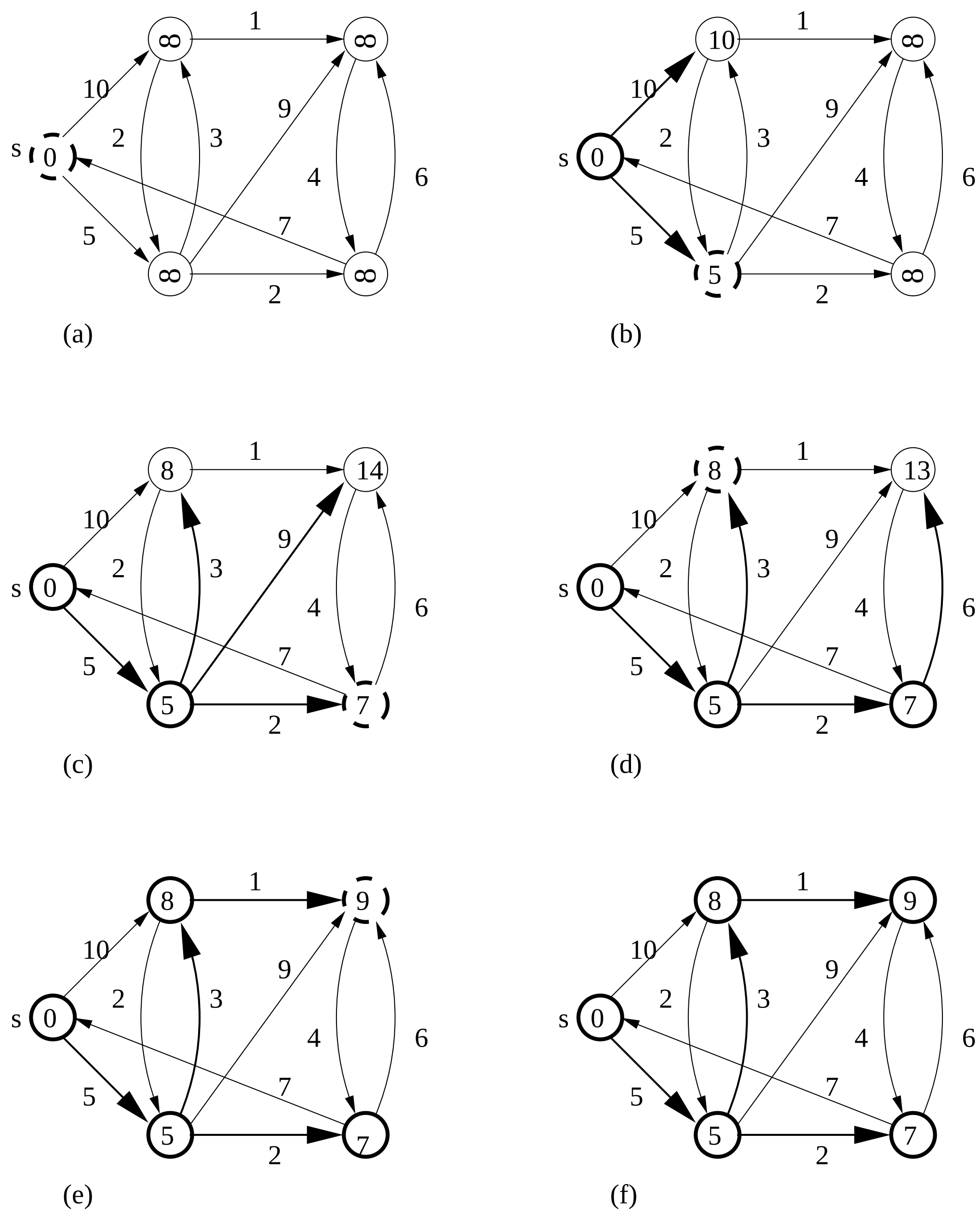

Figure 1: Execution of algorithm *Dijkstra*. The shortest-path estimates are shown within the nodes. Darkened circles are the nodes in S. Darkened edges indicate predecessor values. (a) The situation right before the first iteration of the while-loop. Dashed circle is the node in Q with smallest d value and chosen as node u. (b)-(f) The situation after each iteration, with dashed circle chosen as u in the next iteration. d and π of (f) are the final values.

Proof. We prove this lemma by induction.

Base: After the first iteration of the while-loop, $S = \{s\}$ and $Q = V - \{s\}$. The weights of SPs for nodes in Q with respect to S are correctly computed. Furthermore, among all nodes in Q reachable from s by a single edge, $d[u] = \delta(s, u)$ for node u with the smallest d value. This is because that otherwise the SP from s to u must have at least two edges, and this contradicts $d[u] = \min\{d[z] | z \in Q\}$ due to the fact that all edge weights are non-negative.

Hypothesis: Suppose the lemma holds after j iterations, $j = m < n - 1$.

Induction: Consider $j = m + 1$. Let the set S formed in the m-th iteration be the old S. In the $(m + 1)$-th iteration, node u in Q such that $d[u] = \min\{d[z] | z \in Q\}$ is included into S (and deleted from Q at the same time), and the old $d[v]$ values of all nodes $v \in Adj[u]$ are compared with $d[u] + w(u, v)$ and updated as $d[v] := d[u] + w(u, v)$ if the old $d[v]$ is greater than $d[u] + w(u, v)$. The S with u added is called the new S. We want to prove that after the $(m + 1)$-th iteration of the while-loop, $d[v']$ is the weight of the shortest path (SP) from s to v' for any $v' \in Q$ with respect to (new) S. There are 3 cases.

Case (i): There is no edge (u, v'). Then, by the hypothesis, this claim is true.

Case (ii): There is an edge (u, v') and the shortest path from s to v' with all predecessors in new S includes u. Then, $d[v']$ with respect to the new S is correctly computed based on the hypothesis.

Case (iii): There is an edge (u, v') but the shortest path from s to v' with all predecessors in new S does not include u, and $d[v']$ with respect to the new S is also correctly computed based on the hypothesis.

Thus, right after the $(m + 1)$-th iteration of the while-loop, $d[v']$, $v' \in Q$, is the weight of the shortest path (SP) from s to v' with respect to S. This completes the inductive proof of the first part of the lemma.

Now, we want to prove that after j-th iteration of the while-loop, $d[u] = \min\{d[z] | z \in Q\} = \delta(s, u)$; i.e. the shortest path from s to u, whose $d[u]$ is $\min\{d[z] | z \in Q\}$, with respect to S is the shortest path from s to u in G. Suppose there is a hypothetical shorter path P to u that first leaves S to go to a node x (of course it is in Q) then (perhaps) wanders into and out of S before ultimately arriving at u (see Figure 2). Then, the path from s to x is shorter than the hypothetical path from s to u (note: the fact of non-negative edge lengths ensures this). In this case, $d[u] > d[x]$, contradicting the assumption that $d[u] = \min\{d[z] | z \in Q\}$. Therefore, $d[u] = \delta(s, u)$. □

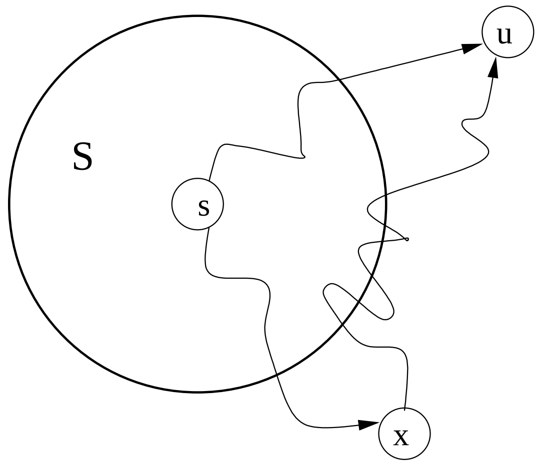

Figure 2: Hypothetical shorter path to u.

Theorem 1 *Algorithm Dijkstra, run on a directed graph $G = (V, E)$ with non-negative weight function w and source s, terminates with $d[v] = \delta(s, v)$ for all nodes $v \in V$.*

Proof. By Lemma 1, after the j-th iteration, weights of shortest path for $j + 1$ nodes have been computed. These weights will not be changed in later iterations. After $n - 1$ iterations, $d[v] = \delta(s, v)$ for all nodes $v \in V$. The while-loop runs for n iterations. The last iteration is actually redundant. □

Time Complexity Analysis

(1) Q is implemented as a min-heap.

- *Initialize-Single-Source* takes $O(|V|)$ time.
- Initializing min-heap Q takes $O(|V|)$ time.
- Since the number of iterations of the while-loop is $|V|$, each calls *Extract-Min* once (which takes $O(\lg |V|)$ time), the total time for all *Extract-Min* calls is $O(|V| \cdot \lg |V|)$.
- The total number of calls to *Relax* is $O(|E|)$. Since Q is a min-heap, each call to *Relax* may implicitly executes *Heap-Decrease-Key*, which takes $O(\log |V|)$ time. All calls to *Relax* take a total of $O(|E| \cdot \lg |V|)$ time.

The total time for *Dijkstra* is $O(|E| \cdot \lg |V|)$, if Q is implimentaed by a min-heap.

(2) Q is implemented by a linear array.

Extract-Min takes $O(|V|)$ time. Total time for all calls to *Extract-Min* is $O(|V|^2)$. Since decreasing a key value takes $O(1)$ time if Q is an array, total time for all calls to *Relax* is $O(|E|)$. Thus, the total running time of *Dijkstra* is $O(|V|^2)$ if Q is implemented by an array.

Note: For dense graphs as as those with $|E| = O(|V|^2)$, (2) is more efficient than (1).