The single-source shortest path algorithm

Jayadev Misra

June 17, 2016

1 Introduction

Given is a directed graph in which each edge (x, y) has a positive length $l(x, y)$. (All the algorithms in this note are correct even when some edge lengths are zero though the proofs are slightly more elaborate.) The length of a path is the sum of the edge-lengths along that path. It is required to find the shortest path from a given node, *source*, to another specified node. We describe an algorithm, due to Dijkstra, that solves the shortest path problem in $O(n^2)$ steps where n is the number of nodes. The algorithm finds the shortest path from *source* to all nodes. Henceforth, *path* refers to a path from *source*.

We show two different developments of the same algorithm, the first based on discrete event simulation and the second a more conventional treatment.

2 A Discrete Event Simulation Algorithm

Construct a fantasy algorithm to list all paths ordered by their lengths.

Nodes are autonomus and they operate in real time. They communicate using light quanta, called *photons*. A photon carries a *hop count* with it. Let node x receive (along an incoming edge) a photon with hop count n at time t; then it records the triple (n, t, x) in a common *ledger* and sends photons with hop counts $n + 1$ along each of its outgoing edges. A photon consumes $l(x, y)$ time units to travel from node x to y. Nodes take no time in recording in the ledger nor sending and receiving photons. The procedure starts by having the source receive a photon with hop count 0 at time 0. Observe that a typical graph with cycles may have an infinite number of paths to some nodes, so the procedure may never terminate and the ledger may become infinitely long.

We prove below that (n, t, x) is a ledger entry iff there is a path with n edges of length t to x. For the shortest path problem a node may ignore all but the first photon it receives. This is because if x receives photons at t and t', $t \leq t'$, t represents as short, or even a shorter, path, and any successor z of x receives photons at $t + l(x, z)$ and $t' + l(x, z)$, and $t + l(x, z) \leq t' + l(x, z)$. Also, the hop count, which has been introduced to simplify the proof, may be ignored because the number of edges in the shortest path is usually irrelevant.

Correctness The proof of the claim, that (n, t, x) is a ledger entry iff there is a path with n edges of length t to x, is by induction on n. For $n = 0$ the only path with 0 edges is the one from *source* to itself, which is given by the ledger entry $(0, 0, source)$. For $n > 0$,

$$
\begin{array}{ll}
 & \text{there is a path with } n \text{ edges of length } t \text{ to } x \\
\equiv & \{\text{graph theory}\} \\
 & \text{for some predecessor } y \text{ of } x\text{:} \\
 & \text{there is a path with } n-1 \text{ edges of length } t - l(y,x) \text{ to } y \\
\equiv & \{\text{induction}\} \\
 & \text{for some predecessor } y \text{ of } x\text{:} \\
 & \text{there is a ledger entry } (n-1, t-l(y,x), y) \\
\equiv & \{x \text{ receives } n \text{ at } t,\ n > 0, \text{ iff} \\
 & x \text{ has a predecessor } y \text{ that receives } n-1 \text{ at } t - l(y,x)\} \\
 & \text{there is a ledger entry } (n, t, x)
\end{array}
$$

Implementation outline The algorithm as described is concurrent, and runs in real time in which processing steps by the nodes consume no time. Such algorithms can be implemented using the classic discrete event simulation scheme. We describe the simulation algorithm as it pertains to this problem.

Call the receipt of a photon an *event* and simulate the events in order of their occurrence. Occurrence of an event may cause other events to happen in the future, in this case receipt of a photon may cause sending of photons that cause receipts of photons, i.e., events, in the future. Construct a set *event_queue* that includes all events (t, x), for x receiving a photon at time t, that are known to happen in the future. Initially, the only known event is $(0, source)$. A step of simulation removes the next event that is known to happen, in this case the entry (t, x) in the *event_queue* that has the smallest time component t (break ties arbitrarily), and processes it as follows. If event (t, x) corresponds to the receipt of the first photon by x then (1) insert (t, x) in a set *ledger*, and (2) since sending a photon to z causes the event $(t + l(x, z), z)$ to happen in the future add all such events to *event_queue*. If (t, x) corresponds to the receipt of a non-first photon by x, ignore it.

To determine if an event corresponds to the first photon, call a node *lit* if it has already received a photon, *unlit* otherwise. The implementation outline:

event_queue := $\{(0, source)\}$;
every node is unlit;
while there is an unlit node **do**
    remove (t, x) with the smallest t from *event_queue*;
    if x is unlit **then**
        record (t, x) in the ledger and that x is now lit;
        for every edge (x, z): insert $(t + l(x, z), z)$ in *event_queue*
    endif
enddo

Exercises

1. Augment the program to compute the shortest path in addition to its length.

2. Show that as long as there is an unlit node *event_queue* is not empty. You will have to use the fact that every node is reachable from *source*.

3. Derive an upper bound on the number of steps assuming that the simplest procedure is used for removing entries from *event_queue*.

4. Improve the last (insert) step by noting that if z is lit then the event will be ignored in the future. Does this modification improve the number of steps substantially?

3 Conventional Description of the Algorithm

We show a conventional development of the shortest path algorithm that is identical to the implementation described in the previous section. Shortest paths to all nodes are computed in order of their lengths. This is precisely the order of the ledger entries in the previous section though it is not necessary to read the that section to follow the rest of this note.

Let d_x be the length of the shortest path to node x. Given the shortest path values sorted in ascending order define x to have *rank* k if the position of d_x is k in this order (break ties arbitrarily); *source* has rank 0. Henceforth, a *non-final* node on a path is any node except the final node.

Observation 1 The ranks of the nodes in any shortest path is an increasing sequence. Consequently, the final node has the highest rank in any shortest path.

Proof: Let x be a non-final node on a shortest path to some node. The segment up to x is a shortest path to x because otherwise the segment may be replaced by a shortest path to x, thus decreasing the entire path length. Let y be the node following x on the path; such a node exists because x is non-final. The segment lengths up to x and y are d_x and d_y, respectively, from the above arguments. And $d_y = d_x + l(x, y)$ because y follows x in the path. From $l(x, y) > 0$, $d_y > d_x$; so y has rank higher than x.

The Algorithm The given observation suggests a way to determine the shortest paths in order of ranks. *source* is of rank 0 and has a shortest path of length 0. Given all nodes of rank k or lower —call these nodes *lit* and the remaining nodes *unlit*— compute the node of rank $k + 1$ as follows. For any x in *unlit* let d'_x be the length of the shortest path to node x using only nodes from *lit* as non-final nodes, and $d'_x = \infty$ if there is no such path. From the observation, node z with the minimum value of d' in *unlit* has rank $k + 1$ and $d_z = d'_z$.

Initially $lit = \{\}$, $d'_{source} = 0$ because it has no non-final node in its shortest path and $d'_x = \infty$, for all other x because they need to have at least *source* as a non-final node. After finding z, the node of the next rank, add it to lit. Then recompute d'_x for all x in $unlit$ as follows. The paths to x that use non-final nodes from lit are of two kinds: (1) the ones that do not include z have the shortest path length d'_x, and (2) the ones that include z have z as the penultimate node, from the observation; so the shortest such path length is $d'_z + l(z, x)$. Update d'_x to $\min(d'_x, d'_z + l(z, x))$.

Abstract Program

$unlit :=$ all nodes; $d'_{source} := 0$; $d'_x := \infty$, for all x, $x \neq source$;
while there is a node in $unlit$ **do**
  let z in $unlit$ be such that $d'_z = \min\{d'_x \mid x \in unlit\}$;
  $d_z := d'_z$; $unlit := unlit - \{z\}$;
  $d'_x := \min(d'_x, d'_z + l(z, x))$, for all x, $x \in unlit$
enddo