CS 430 Lecture 27 Activities

Shortest Path Problem

How to find the shortest route between two points on a map.

Input

- Directed graph $G = (V, E)$
- Weight function $w : E \to \mathbf{R}$

Weight of path

$$\begin{aligned} p &= \langle v_0, v_1, \ldots, v_k \rangle \\ &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\ &= \text{sum of edge weights on path } p. \end{aligned}$$

Shortest-path weight u to v

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \stackrel{p}{\rightsquigarrow} v & \text{if there exists a path } u \rightsquigarrow v, \} \\ \infty & \text{otherwise.} \end{cases}$$

Shortest path u to v is any path p such that $w(p) = \delta(u, v)$.
Variants

- Single-source: Find shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$.
- Single-destination: Find shortest paths to a given destination vertex.
- Single-pair: Find shortest path from u to v. No way known that's better in worst case than solving single-source.
- All-pairs: Find shortest path from u to v for all $u, v \in V$. We'll see algorithms for all-pairs in the next chapter.

Negative-weight edges – OK, as long as no negative-weight cycles are reachable from the source.

- If we have a negative-weight cycle, just keep going around it, and get $w(s, v) = -\infty$ for all v on the cycle.
- But OK if the negative-weight cycle is not reachable form the source.
- Some algorithms work only if there are no negative-weight edges in the graph.

1. What would the brute force approach be to solve the shortest path problem, and what is its run time?
2. Prove optimal substructure for the shortest path problem.

Output of single-source shortest-path algorithm For each vertex $v \in V$:

- $d[v] = \delta(s, v)$, Initially, $d[v] = \infty$; reduces as algorithms progress. But always maintain $d[v] \leftarrow \delta(s, v)$. Call $d[v]$ a shortest-path estimate.
- $\pi[v]$ = predecessor of v on a shortest path from s. If no predecessor, $\pi[v] = NIL$, π induces a tree–shortest-path tree.

Initialization – All the shortest-paths algorithms start with Init-Single-Source.

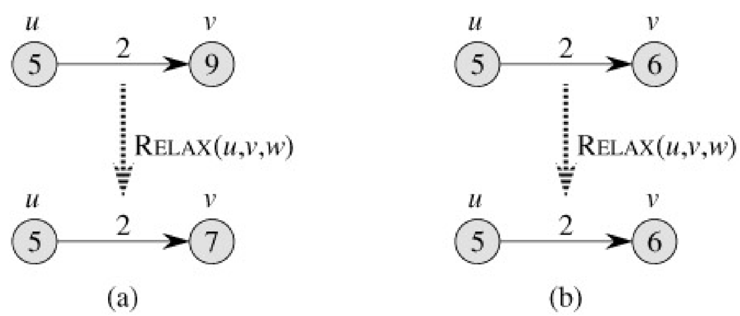
Algorithm 27.1 Single Source Initialization

function Init-Single-Source(V, s)
    for all $v \in V$ **do**
        $d[v] \leftarrow \infty$
        $\pi[v] \leftarrow$ NIL
    end for
    $d[s] \leftarrow 0$
end function

Relaxing an edge (u, v) - Can we improve the shortest-path estimate (best seen so far) from the source s to v be going through u and taking edge (u, v)?

Algorithm 27.2 Relaxing an Edge

function Relax(u, v, w)
    if $d[v] > d[u] + w(u, v)$ **then**
        $d[v] \leftarrow d[u] + w(u, v)$
        $\pi[v] \leftarrow u$
    end if
end function



The algorithms differ in the order and how many times they relax each edge.

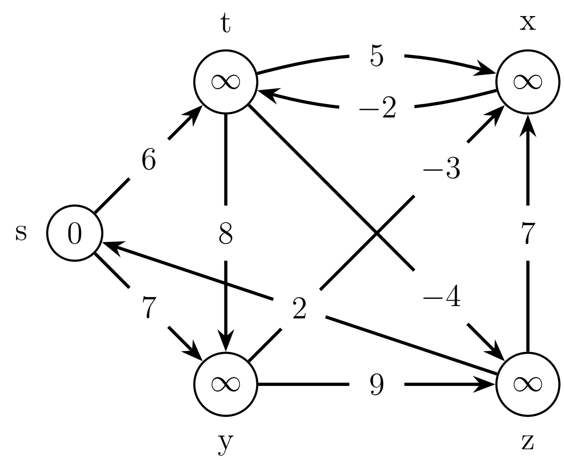
Shortest Path Algorithm - Bellman-Ford

The most straightforward of the "relax an edge" algorithms. Relaxes the edges in a fixed order (any fixed order) $|v| - 1$ times. Not a greedy algorithm.

- Allows negative-weight edges.
- Computers $d[v]$ and $\pi[v]$ for all $v \in V$.
- Returns TRUE if no negative-weight cycles are reachable from s, FALSE otherwise.

Algorithm 27.3 Bellman-Form Shortest Path Algorithm

```
function BELLMAN-FORD(V, E, w, s)
    INIT-SINGLE-SOURCE(V, s)
    for i ← 1 to |V| − 1 do
        for all edge (u, v) ∈ E do
            RELAX(u, v, w)
        end for
    end for
    for all edge (u, v) ∈ E do
        ▷ All edges, in any order, same order each time
        if d[v] > d[u] + w(u, v) then
            return FALSE
        end if
        return TRUE
    end for
end function
```



3. Execute Bellman-Ford on the above graph from source s for this edge order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. Update the $d[v]$ and $\pi[v]$ values for each iteration.

4. What is the runtime of Bellman-Ford?

5. Prove Bellman-Ford is correct.
   Values you get on each pass how quickly it converges depends on order of relaxation. But guaranteed to converged after $|V| - 1$ passes, assume no negative-weight cycles.