Lecture 17: Shortest Paths III: Bellman-Ford

Lecture Overview

- Review: Notation
- Generic S.P. Algorithm
- Bellman-Ford Algorithm
  - Analysis
  - Correctness

Recall:

$$\begin{aligned} \text{path } p &= \; < v_0, v_1, \; \ldots, \; v_k > \\ & \quad (v_1, v_{i+1}) \; \in E \quad 0 \leq i < k \\ w(p) &= \sum_{i-0}^{k-1} w(v_i, v_{i+1}) \end{aligned}$$

Shortest path weight from u to v is $\delta(u, v)$. $\delta(u, v)$ is ∞ if v is unreachable from u, undefined if there is a negative cycle on some path from u to v.

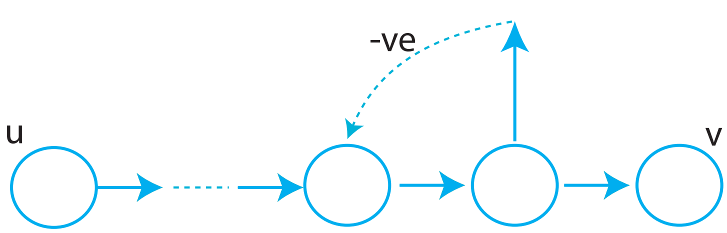

Figure 1: Negative Cycle.

Generic S.P. Algorithm

Initialize: for $v \in V$: $d[v] \leftarrow \infty$, $\Pi[v] \leftarrow$ NIL

$d[S] \leftarrow 0$

Main: repeat

select edge (u, v) [somehow]

"Relax" edge (u, v):
- if $d[v] > d[u] + w(u, v)$:
  - $d[v] \leftarrow d[u] + w(u, v)$
  - $\pi[v] \leftarrow u$

until you can't relax any more edges or you're tired or . . .

Complexity:

Termination: Algorithm will continually relax edges when there are negative cycles present.

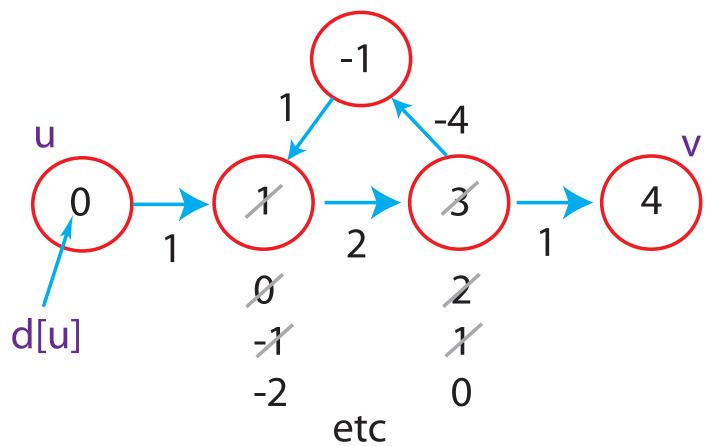

Figure 2: Algorithm may not terminate due to negative cycles.

Complexity could be exponential time with poor choice of edges.

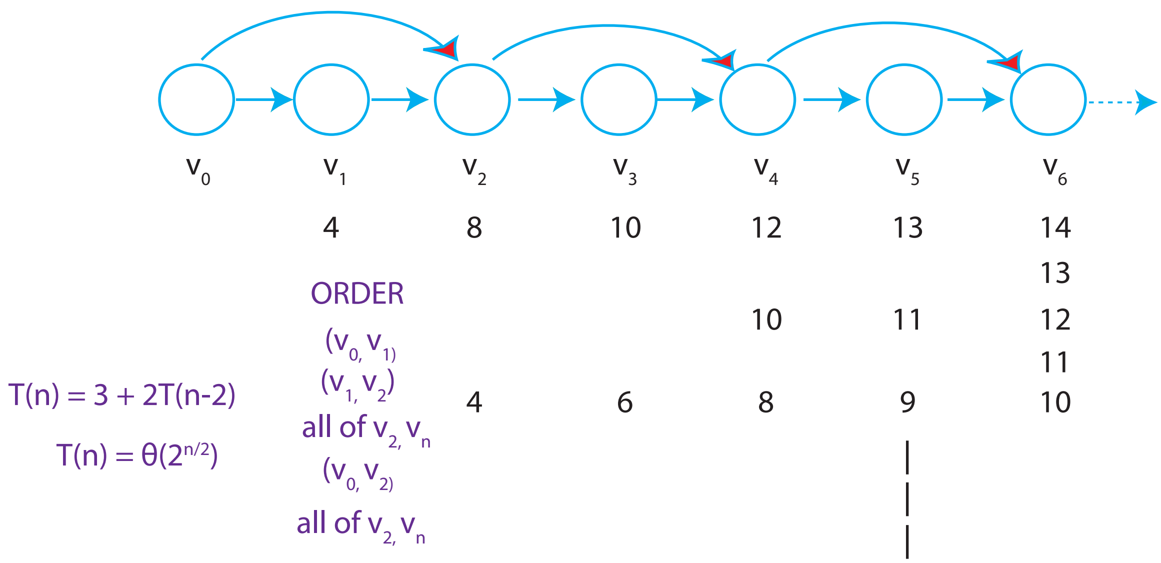

Figure 3: Algorithm could take exponential time. The outgoing edges from v_0 and v_1 have weight 4, the outgoing edges from v_2 and v_3 have weight 2, the outgoing edges from v_4 and v_5 have weight 1.

5-Minute 6.006

Figure 4 is what I want you to remember from 6.006 five years after you graduate!

Bellman-Ford(G,W,s)

```
Initialize ()
for i = 1 to |V| − 1
    for each edge (u, v) ∈ E:
        Relax(u, v)
for each edge (u, v) ∈ E
    do if d[v] > d[u] + w(u, v)
        then report a negative-weight cycle exists
```

At the end, $d[v] = \delta(s, v)$, if no negative-weight cycles.

Theorem:
If $G = (V, E)$ contains no negative weight cycles, then after Bellman-Ford executes $d[v] = \delta(s, v)$ for all $v \in V$.

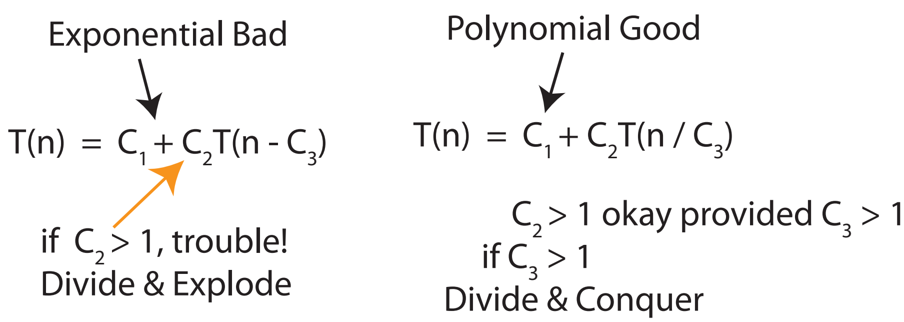

Figure 4: Exponential vs. Polynomial.

Proof:
Let $v \in V$ be any vertex. Consider path $p = \langle v_0, v_1, \ldots, v_k \rangle$ from $v_0 = s$ to $v_k = v$ that is a shortest path with minimum number of edges. No negative weight cycles $\implies$ p is simple $\implies$ $k \leq |V| - 1$.

Consider Figure 6. Initially $d[v_0] = 0 = \delta(s, v_0)$ and is unchanged since no negative cycles.
After 1 pass through E, we have $d[v_1] = \delta(s, v_1)$, because we will relax the edge (v_0, v_1) in the pass, and we can't find a shorter path than this shortest path. (Note that we are invoking optimal substructure and the safeness lemma from Lecture 16 here.)
After 2 passes through E, we have $d[v_2] = \delta(s, v_2)$, because in the second pass we will relax the edge (v_1, v_2).
After i passes through E, we have $d[v_i] = \delta(s, v_i)$.
After $k \leq |V| - 1$ passes through E, we have $d[v_k] = d[v] = \delta(s, v)$. □

Corollary

If a value $d[v]$ fails to converge after $|V| - 1$ passes, there exists a negative-weight cycle reachable from s.
Proof:
After $|V| - 1$ passes, if we find an edge that can be relaxed, it means that the current shortest path from s to some vertex is not simple and vertices are repeated. Since this cyclic path has less weight than any simple path the cycle has to be a negative-weight cycle. □

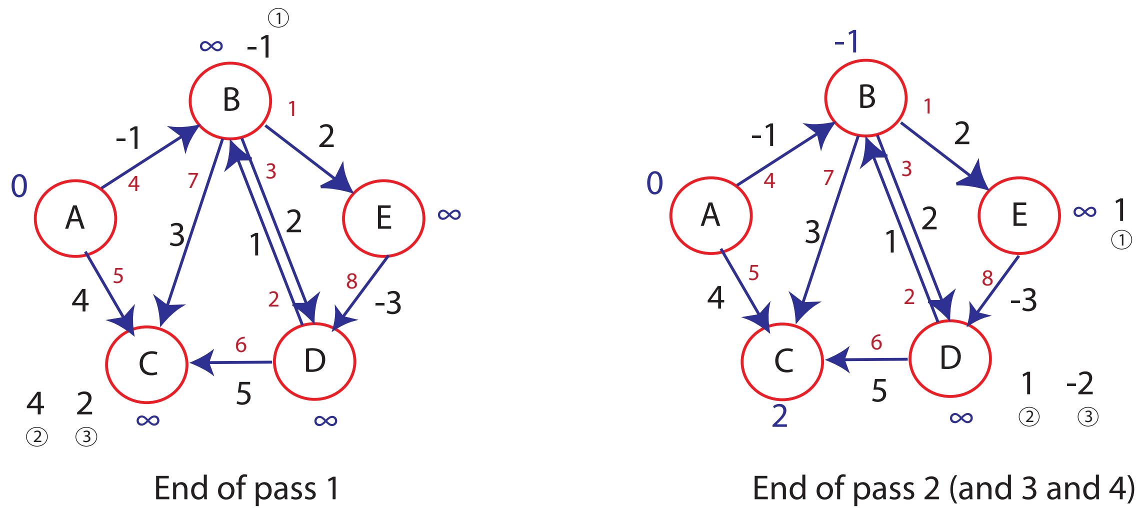

Figure 5: The numbers in circles indicate the order in which the δ values are computed.

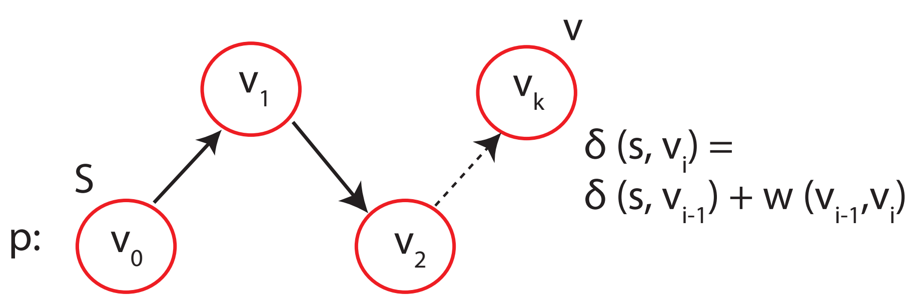

Figure 6: Illustration for proof.

Longest Simple Path and Shortest Simple Path

Finding the longest simple path in a graph with non-negative edge weights is an NP-hard problem, for which no known polynomial-time algorithm exists. Suppose one simply negates each of the edge weights and runs Bellman-Ford to compute shortest paths. Bellman-Ford will not necessarily compute the longest paths in the original graph, since there might be a negative-weight cycle reachable from the source, and the algorithm will abort.

Similarly, if we have a graph with negative cycles, and we wish to find the longest *simple* path from the source s to a vertex v, we cannot use Bellman-Ford. The shortest simple path problem is also NP-hard.

6.006 Introduction to Algorithms
Fall 2011