Single Source Shortest Path: The Bellman-Ford Algorithm

Slides based on Kevin Wayne / Pearson-Addison Wesley

Single Source Shortest Path Problem

Shortest path network.

- Directed graph $G = (V, E, w)$.
- Weight w_e = weight of edge e.

Shortest path problem: given a starting vertex s, find shortest directed path from s to all other vertices

Option #1: if all edge weights are non-negative, use Dijkstra

- Greedy algorithm works in $O(n^2)$ or $O(m \log n)$

Option #2: if some edge weights are negative, use Bellman-Ford

- Runs in $O(mn)$ time
- Also used to determine if there exists a negative-weight cycle

Applications Involving Shortest Paths with Negative Edge Weights

Applications:

- Arbitrage scenarios in finance
  - Situations in which it is possible to make guaranteed profit
    - E.g. currency exchange in which you start with one currency and end up after a sequence of conversions with more money than you started with
    - A negative cycle implies an arbitrage scenario

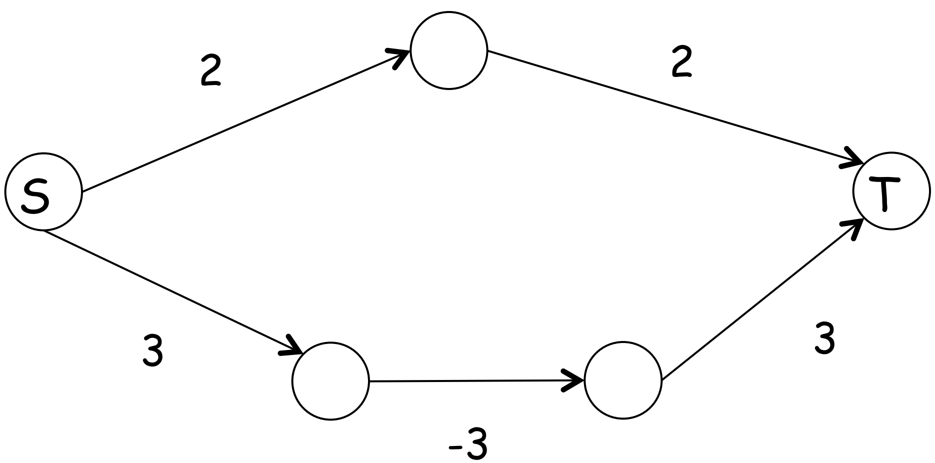
- Distance-vector routing protocols in networking

Bellman-Ford Algorithm

Converting to Positive Weights.

- Couldn't we just add a sufficiently large constant to each edge weight so that they all become positive?
- And then back out this extra weight from each path when we're done?

- No – the problem is that the actual shortest paths might change to the wrong answer



Bellman-Ford Algorithm

Key Observation:

- If there are no negative cycles, then the shortest path from s to t is simple (no repeat nodes) and has at most n-1 edges

- Why?
  - Any path with more than n-1 edges must visit at least one node twice, but the corresponding cycle can't help the shortest path (because all cycles have non-negative weight), so it could be removed and the total distance could only get better.

Bellman-Ford Algorithm

Basic Idea:

- Iterate over the number of vertices
- Keep track of current shortest path (distance and parent) for each vertex
- For each iteration, "relax" all edges
  - Consider whether this edge can be used to improve the current shortest path of the vertex at its endpoint

- After k iterations, the first k steps of any shortest path are correct and will never change from that point (we'll prove shortly)

- Because every shortest path is simple, we know that after at most n-1 iterations, we'll have every shortest path determined

Bellman-Ford Algorithm

Bellman-Ford algorithm.

- Complexity: O(mn)
  - Iterate over the vertices
  - Consider every edge during each iteration

```
Bellman-Ford ( G=(V,E,w), s )

For every vertex v
    d[v] = ∞
d[s]=0
For i=1 to |V|-1 do
    For every edge (u,v) in E do
        If d[v]>d[u]+w(u,v) then
            d[v]=d[u]+w(u,v), parent[v] = u
For every edge (u,v) in E do
    If d[v]>d[u]+w(u,v) then
        Return NEGATIVE CYCLE
Return d[], parent[]
```

Bellman-Ford – Correctness

Correctness Argument assuming no negative weight cycles:

- Proof by induction over the iterations of the algorithm
- Claim: Consider any vertex t and a shortest path P from s to t. Let P be defined as $v_0 v_1 v_2 \ldots v_k$, where $v_0 = s$ and $v_k = t$. After $n \leq k$ iterations of Bellman-Ford, all vertices along the path $v_0 v_1 \ldots v_n$ have had their shortest path computed.

- Base case: n = 0. Shortest path for s is 0, and can't be improved by any cycles.
- Inductive case: Suppose true for j iterations. Then the shortest path from s to v_j has been calculated, and in iteration j+1, all edges are relaxed again, in particular edge $e = (v_j, v_{j+1})$, such that the shortest path is correctly computed for v_{j+1}.

- Note that by definition of the Bellman-Ford algorithm, the shortest path to a vertex can never increase from one iteration to the next, and it can never get lower than the true shortest path.

Bellman-Ford: Negative Cycles

Negative Cycles

- Assuming there are no negative cycles, every shortest path is simple and contains at most n-1 edges
- Therefore the Bellman-Ford algorithm correctly identifies all shortest paths from source vertex s in at most n-1 iterations.

- As soon as you have an iteration in which nothing changes (no vertices receive improved shortest paths) the algorithm is finished – nothing can change from that point.

- So to check for cycles, after completing n-1 iterations of Bellman-Ford, simply scan all edges one more time to see if there is a vertex that could still be improved.
  - If so, that implies a path longer than n-1 edges to achieve the shortest path, which implies a negative cycle.