Lecture 13: Bellman-Ford

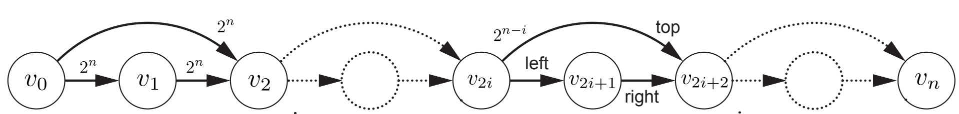
Extended Review: Shortest Path Problem

- Recall: $\delta(u, v)$ is the minimum weight of a path from u to v, or ∞ if there is no such path, or $-\infty$ if there's a negative weight cycle on the way.

- Single Source Shortest Path Problem: for a fixed node s, compute $\delta(s, v)$ for all $v \in V$, and a *tree* of parent pointers whose paths realize all the finite weights.
  - We proved such a tree exists last class
  - In particular, all $O(|V|)$ paths can be described in $O(|V|)$ space

- Relaxation Algorithm Framework:
  - Keep value $d[v]$ for each node v; the value $d[v]$ will only decrease during the algorithm, and will always satisfy $d[v] \geq \delta(s, v)$ (Safety lemma)
  - Relax edge (u, v): reassign $d[v]$ to $d[u] + w(u, v)$ if the latter is smaller
  - Continue relaxing edges *in some order*
  - If no edge is relaxable, then $d[v] = \delta(s, v)$ for all v (Termination lemma)

- All about finding the right order to relax edges. Not all orders are good. With poor choice, this example makes exponentially many updates.



- SSSP in a DAG:
  - Relax edges in top-sort order (wave frantically at the DAG example)
  - Time $O(|V| + |E|)$ total: DFS for top-sort, then relax each edge once
  - Can't be negative weight cycles, so $\delta(s, v)$ always finite
  - Correctness: every shortest path is guaranteed to be relaxed in order (details similar to Bellman-Ford analysis below).

General SSSP: cycles allowed

- Most general SSSP problem: negative weights allowed, and cycles allowed
- New difficulty: if there's a negative weight cycle **reachable from s**, some nodes will have $\delta(s, v) = -\infty$. What is desired behavior?
  - Version 1: Detect whether there's a negative weight cycle. If so, reject the input as ill-formed. This is the usual meaning of "Bellman-Ford Algorithm"
  - Version 2: Locate all nodes with $\delta(s, v) = -\infty$, i.e., all nodes reachable from a negative weight cycle.
  - Version 3: Find an actual negative weight cycle.
- The main algorithm is simple:
  - In each **round**, relax every edge once, in any order.
  - Bellman-Ford algorithm: do $|V| - 1$ rounds.

```
Bellman-Ford:
  initialize parent and d arrays
  for round in range(|V|-1):
    for edge (u,v) in G:
      relax(u,v)
  handle negative weight cycles somehow
  return parent, d
```

- Lemma: if $\delta(s, v)$ is finite, then there's a min-weight path that is **simple**: visits each node at most once. In particular, there is a min-weight path with length $\leq |V| - 1$.
  - Proof of lemma: cut out cycles, since they can't be negative weight.
- Recall the Subpath property: subpaths of shortest paths are shortest paths
- Bellman-Ford correctness:
  - Suppose $\delta(s, v)$ is finite, and $(s = v_0) \to v_1 \to \cdots \to v_{k-1} \to (v_k = v)$ is a min-weight path to v with at most $|V| - 1$ edges.
  - By Subpath property, each $\delta(s, v_i)$ is finite
  - Claim: after round i and all later rounds, $d[v_i] = \delta(s, v_i)$. Proof by induction.
  - Base case: $\delta(s, s) = 0$ when we start, because no min-weight cycles en route from s to v.
  - Before round i, $d[v_{i-1}] = \delta(s, v_{i-1})$ by assumption. During round i, edge (v_{i-1}, v_i) is relaxed, so $d[v_i] \leq \delta[v_{i-1}] + w(v_{i-1}, v_i)$, which is $\delta(v_i)$ by the Subpath property. This value never goes up, and by Safety it cannot drop below $\delta(v_i)$, so it stays there forever.

- Basically the same correctness proof as DAG-SSSP.
- Runtime: $O(|V| \cdot |E|)$.
  - $O(|V|)$ rounds, $O(|E|)$ relaxations per round
  - Minor detail: a naive loop "`for u in Adj:  for v in Adj[u]:`" takes $O(|V| + |E|)$ time to enumerate the edges, not $O(|E|)$, which would lead to $O(|V||E| + |V|^2)$ runtime overall. Often this difference doesn't matter (e.g., connected implies $|V| = O(|E|)$), but to be safe, just make a list of the edges at the beginning.

V1: Detecting Negative Weight Cycles

- Standard Bellman-Ford detects the presence of a negative weight cycle reachable from s, but doesn't identify all $\delta = -\infty$ verts, and doesn't actually find a negative weight cycle
- The algorithm: After $|V| - 1$ rounds, check if any edge is still relaxable. If so, there must be a negative weight cycle.

```
Bellman-Ford:
  initialize parent and d arrays
  for round in range(|V|-1):
    for edge (u,v) in G:
    relax(u,v)
  # Now check for neg-weight cycles
  for edge (u,v) in G:
    if d[v] > d[u] + w(u,v):  # This edge can be relaxed
      raise ValueError("There's a neg-weight cycle reachable from s!!!")
  return parent, d
```

- Correctness follows from the Termination condition: If no edge can be relaxed, then $d[v] = \delta(s, v)$ for all v. (Proved last time.)
- Runtime: $O(|E|)$ for the final check, so still $O(|V||E|)$ total

V2: All nodes with $\delta(s, v) = -\infty$

- The Algorithm: After $|V|-1$ rounds, collect the set $N = \{v \mid \text{some edge } (u, v) \text{ is relaxable}\}$. All nodes reachable from nodes in N are the ones with $\delta = -\infty$.
- How to implement `multi_source_search(N)`? Lots of ways, all $O(|V| + |E|)$
  - Run a BFS where L_0 is initialized to N instead of a single node.
  - Run a version of full-DFS where only nodes in N are used in the outer loop.
  - Add a new node x and a new edge from x to each node in N. Run DFS or BFS from x.

```
Bellman-Ford V2:
  initialize parent and d arrays
  for round in range(|V|-1):
    for edge (u,v) in G:
    relax(u,v)
  # Now find verts with delta = -infinity
  N = {}
  for edge (u,v) in G:
    if d[v] > d[u] + w(u,v):  # This edge can be relaxed
      N.insert(v)
  for v in multi_source_search(N):  # Nodes reachable from anything in N
    d[v] = -math.infty
    parent[v] = None   # These aren't part of the shortest path tree
  return parent, d
```

- Runtime: $O(|V| + |E|)$ additional, still $O(|V||E|)$ total.
- Correctness: if $\delta(s, v)$ is finite, we've shown that $d[v]$ converges to $\delta(s, v)$ in at most $|V| - 1$ rounds. So all nodes added to N must have $\delta = -\infty$, as do all nodes reachable from N.
- How do we know we found them all? Claim: Every negative weight cycle **reachable from** s has a relaxable edge.
  - Proof: Say $v_0 \to \cdots \to v_{k-1} \to (v_k = v_0)$ is any cycle, and suppose no edge is relaxable. Then $d[v_i] + w(v_i, v_{i+1}) \geq d[v_{i+1}]$. Adding all these, we can cancel $\sum d[v_i]$ from both sides, leaving $\sum w(v_i, v_{i+1}) \geq 0$. So the cycle has nonnegative weight.
- Every reachable negative weight cycle contributes at least one node to N, so `multi_source_search(N)` grabs everything in this cycle or reachable from it.

V3: Find a Negative Weight Cycle

Wait for Pset 8.

What about parent pointers?

- When restricted to only the finite vertices, claim the parent pointer graph is a tree, with all paths leading to s. Proof idea: any cycle would have to have negative weight, and s is the only finite node with no parent. Full proof in CLRS, Lemma 24.16.
- Now if $\delta(s, v_k)$ is finite and $v_k \to v_{k-1} \to \cdots \to (v_0 = s)$ is a path following parent pointers (so its *reverse* is a path in G), we have $\delta(s, v_i) \leq \delta(s, v_{i-1}) + w(v_{i-1}, v_i)$ because the edge is not relaxable. Adding these and cancelling yields $\sum w(v_{i-1}, v_i) \leq \delta(v_k)$, so they're equal.
- Not Bellman-Ford specific. Applies to any relaxation framework algorithm whose finite vertices have converged.

SSSP Algorithm	Lecture	Setting	Running Time
BFS	10	$w(e) = 1$ for all $e \in E$	$O(V + E)$
DAG shortest paths	12	G acyclic	$O(V + E)$
Dijkstra	14	$w(e) \geq 0$ for all $e \in E$	$O(V \log V + E)$
Bellman-Ford	13	(general)	$O(VE)$

```
def relaxation_sssp(Adj, w, s, pick_edge):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) ≤ 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u

    while True:
        u, v = pick_edge(Adj, w, s, d)
        if u is None: break
        relax(u, v)
```

```
def dag_sssp(Adj, w, s):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) ≤ 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u

    for u in topological_sort(Adj):
        for v in Adj[u]:
            relax(u, v)
```

```
def bellman_ford_sssp(Adj, w, s):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) ≤ 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u

    for i in range(len(Adj)-1): # O(V) rounds
        for u in Adj:
            for v in Adj[u]:    # O(V+E) / round
                relax(u, v)
    <handle negative weight cycles somehow>
```

```python
def bellman_ford_sssp(Adj, w, s):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) ≤ 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u

    edges = list_of_edges(Adj)   # O(V+E)
    for i in range(len(Adj)-1):  # O(V) rounds
        for (u,v) in edges:      # O(E) / round
            relax(u, v)
    <handle negative weight cycles somehow>
```

```
def bellman_ford_sssp(Adj, w, s):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) ≤ 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u

    edges = list_of_edges(Adj)  # O(V+E)
    for i in range(len(Adj)-1): # O(V) rounds
        for (u,v) in edges:     # O(E) / round
            relax(u, v)
    for (u,v) in edges:         # O(E)
        if d[v] > d[u] + w(u,v):
            raise ValueError(“Neg Cycle!”)
```

```python
def bellman_ford_sssp_2(Adj, w, s):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) ≤ 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u
    edges = list_of_edges(Adj)  # O(V+E)
    for i in range(len(Adj)-1): # O(V) rounds
        for (u,v) in edges:     # O(E) / round
            relax(u, v)
    N = set()
    for (u,v) in edges:
        if d[v] > d[u] + w(u,v):
            N.add(v)
    for v in multi_source_search(N):
        d[v] = -math.inf
        parent[v] = None
```