The many cases of finding shortest paths

- We’ve already seen how to calculate the shortest path in an unweighted graph (BFS traversal)
- We’ll now study how to compute the shortest path in different circumstances for weighted graphs
  - Single-source shortest path on a weighted DAG
  - Single-source shortest path on a weighted graph with nonnegative weights (Dijkstra’s algorithm)
  - Single-source shortest path on a weighted graph including negative weights (Bellman-Ford algorithm)

Shortest paths

Shortest path problem. Given a digraph \(G = (V, E)\), with arbitrary edge weights or costs \(c_{uv}\), find cheapest path from node \(s\) to node \(t\).

Cost of path = 9 - 3 + 1 + 11 = 18
Shortest paths: failed attempts

Dijkstra. Can fail if negative edge weights.

Reweighting. Adding a constant to every edge weight can fail.

Negative cycles

Def. A negative cycle is a directed cycle such that the sum of its edge weights is negative.

Lemma 1. If some path from \( v \) to \( t \) contains a negative cycle, then there does not exist a cheapest path from \( v \) to \( t \).

Pf. If there exists such a cycle \( W \), then can build a \( v \rightarrow t \) path of arbitrarily negative weight by detouring around cycle as many times as desired.

Lemma 2. If \( G \) has no negative cycles, then there exists a cheapest path from \( v \) to \( t \) that is simple (and has \( \leq n - 1 \) edges).

Pf.
- Consider a cheapest \( v \rightarrow t \) path \( P \) that uses the fewest number of edges.
- If \( P \) contains a cycle \( W \), can remove portion of \( P \) corresponding to \( W \) without increasing the cost.
Shortest path and negative cycle problems

**Shortest path problem.** Given a digraph $G = (V, E)$ with edge weights $c_{vw}$ and no negative cycles, find cheapest $v \rightarrow t$ path for each node $v$.

**Negative cycle problem.** Given a digraph $G = (V, E)$ with edge weights $c_{vw}$, find a negative cycle (if one exists).

![shortest-paths tree](image1.png)  ![negative cycle](image2.png)

Bellman-Ford algorithm intuition

- With negative edges, we can’t select the next vertex to visit cleverly.
- Instead, we just relax all $m$ edges $n$ times in a row.
- If, after $n$ rounds, the upper bounds don’t shrink, then we’ve found the shortest path from the source.
- But if the upper bounds still shrink after $n$ rounds, then there is a negative cycle in the graph, which is a problem (why?)
- Running time: $\Theta(m \cdot n)$

Bellman-Ford algorithm code

```python
def bellman_ford(G, s):
    D, P = {s: 0}, {}
    # Zero-dist to s; no parents
    for rnd in G:
        changed = False
        # No changes in round so far
        for u in G:
            # For every from-node...
            for v in G[u]:
                # ... and its to-nodes...
                if relax(G, u, v, D, P):
                    # Shortcut to v from u?
                    changed = True
                    # Yes! So something changed
                if not changed:
                    break
            # No change in round: Done
        else:
            # Not done before round n?
            raise ValueError('negative cycle')
            # Negative cycle detected
    return D, P
    # Otherwise: D and P correct
```

Shortest paths: dynamic programming

**Def.** $OPT(i, v) =$ cost of shortest $v \rightarrow t$ path that uses $\leq i$ edges.

- **Case 1:** Cheapest $v \rightarrow t$ path uses $\leq i − 1$ edges.
  - $OPT(i, v) = OPT(i − 1, v)$

- **Case 2:** Cheapest $v \rightarrow t$ path uses exactly $i$ edges.
  - if $(v, w)$ is first edge, then $OPT$ uses $(v, w)$, and then selects best $w \rightarrow t$ path using $\leq i − 1$ edges.

$$OPT(i, v) = \begin{cases} \infty & \text{if } i = 0 \\ \min \{ OPT(i − 1, v), \min_{(v, w) \in E} \{ OPT(i − 1, w) + c_{vw} \} \} & \text{otherwise} \end{cases}$$

**Observation.** If no negative cycles, $OPT(n − 1, v) =$ cost of cheapest $v \rightarrow t$ path.

**Pf.** By Lemma 2, cheapest $v \rightarrow t$ path is simple. •

Bellman-Ford algorithm code

```python
def bellman_ford(G, s):
    D, P = {s: 0}, {}
    # Zero-dist to s; no parents
    for rnd in G:
        changed = False
        # No changes in round so far
        for u in G:
            # For every from-node...
            for v in G[u]:
                # ... and its to-nodes...
                if relax(G, u, v, D, P):
                    # Shortcut to v from u?
                    changed = True
                    # Yes! So something changed
                if not changed:
                    break
            # No change in round: Done
        else:
            # Not done before round n?
            raise ValueError('negative cycle')
            # Negative cycle detected
    return D, P
    # Otherwise: D and P correct
```
Bellman-Ford algorithm example

Edge Eval. Order
(t,x)
(t,y)
(t,z)
(x,t)
(y,x)
(y,z)
(z,x)
(z,s)
(s,t)
(s,y)

<table>
<thead>
<tr>
<th>Node</th>
<th>d[Node]: upper bd. dist. from s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>init. 1 2 3 4 5</td>
</tr>
<tr>
<td>s</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>t</td>
<td>∞ 6 6 2 2 2</td>
</tr>
<tr>
<td>x</td>
<td>∞ ∞ 114 4 4 4</td>
</tr>
<tr>
<td>y</td>
<td>∞ 7 7 7 7 7</td>
</tr>
<tr>
<td>z</td>
<td>∞ ∞ 2 2 -2 -2</td>
</tr>
</tbody>
</table>