Dynamic Programming

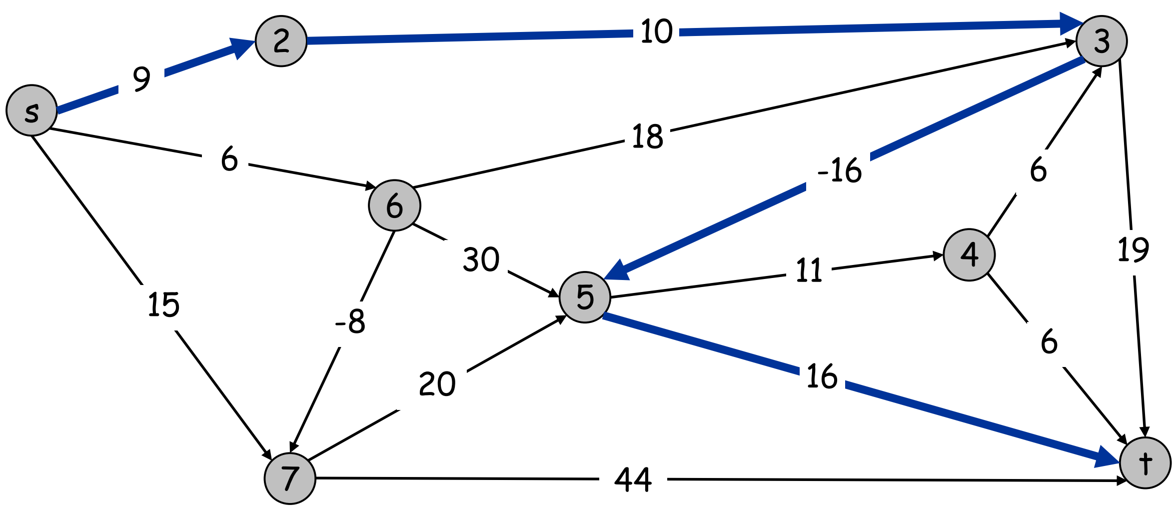
Bellman-Ford Algorithm for Shortest Paths

Shortest Paths

Shortest path problem. Given a directed graph $G = (V, E)$, with edge weights c_{vw}, find shortest path from node s to node t.

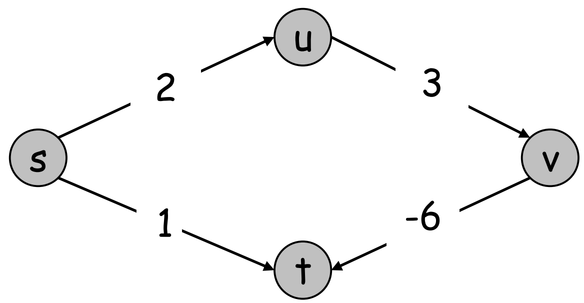
allow negative weights

Ex. Nodes represent agents in a financial setting and c_{vw} is cost of transaction in which we buy from agent v and sell immediately to w.

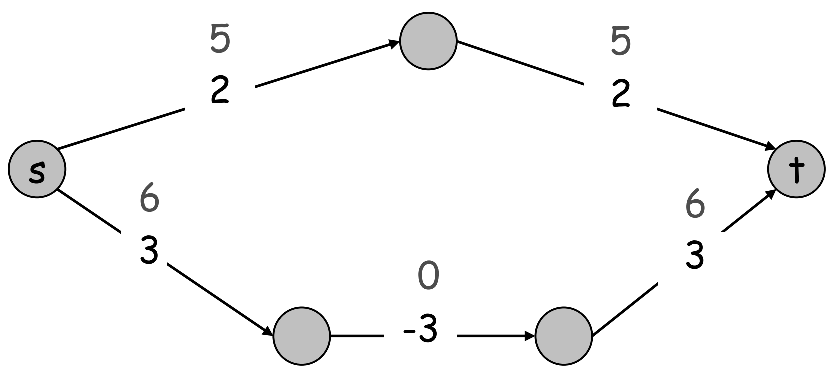


Shortest Paths: Failed Attempts

Dijkstra. Can fail if negative edge costs.

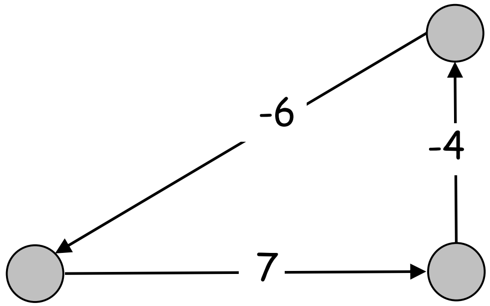


Re-weighting. Adding a constant to every edge weight can fail.

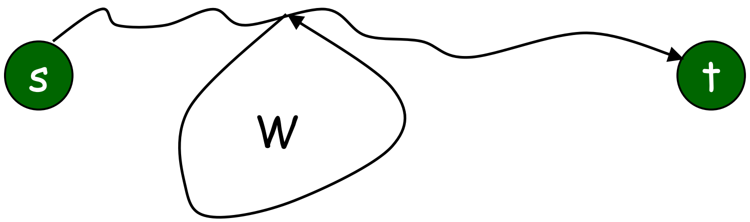


Shortest Paths: Negative Cost Cycles

Negative cost cycle.



Observation. If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path; otherwise, there exists one that is simple.



$c(W) < 0$

Shortest Paths: Dynamic Programming

Def. OPT(i, v) = length of shortest v-t path P using at most i edges.

- Case 1: P uses at most i-1 edges.
  - OPT(i, v) = OPT(i-1, v)

- Case 2: P uses exactly i edges.
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best w-t path using at most i-1 edges

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

- OPT(0,t) = 0

Remark. By previous observation, if no negative cycles, then OPT(n-1, v) = length of shortest v-t path.

Shortest Paths: Implementation

```
Shortest-Path(G, t) {
   foreach node v ∈ V
      M[0, v] ← ∞
   M[0, t] ← 0

   for i = 1 to n-1
      foreach node v ∈ V
         M[i, v] ← M[i-1, v]
      foreach edge (v, w) ∈ E
         M[i, v] ← min { M[i, v], M[i-1, w] + c_vw }
}
```

Analysis. $\Theta(mn \log(n))$ time, $\Theta(n^2 \log(n))$ space.

Finding the shortest paths. Maintain a "successor" for each table entry. We will see it shortly.

Shortest Paths: Practical Improvements

Practical improvements.

- Maintain only one array M[v] = shortest v-t path that we have found so far.
- No need to check edges of the form (v, w) unless M[w] changed in previous iteration.

Theorem. Throughout the algorithm, M[v] is length of some v-t path, and after i rounds of updates, the value M[v] is no larger than the length of shortest v-t path using $\leq$ i edges.

Overall impact.

- Memory: $O((m + n)\log n)$.
- Running time: $O(mn \log n)$ worst case, but substantially faster in practice.

Bellman-Ford: Efficient Implementation

```
Push-Based-Shortest-Path(G, s, t) {
   foreach node v ∈ V {
      M[v] ← ∞
      successor[v] ← φ
   }

   M[t] = 0
   for i = 1 to n-1 {
      foreach node w ∈ V {
      if (M[w] has been updated in previous iteration) {
         foreach node v such that (v, w) ∈ E {
            if (M[v] > M[w] + c_vw) {
               M[v] ← M[w] + c_vw
               successor[v] ← w
            }
         }
      }
      If no M[w] value changed in iteration i, stop.
   }
}
```

Summary

Greedy Algorithms:

- Interval Scheduling (a.k.a. Activity-Selection) [CLRS, Ch16.1]
- Shortest Paths in a Graph, Dijkstra's Algorithm [CLRS, Ch24.3]
- Minimum Spanning Tree [CLRS, Ch 23.1-2]
- Huffman code [CLRS, Ch16.3]

Dynamic Programming:

- Shortest path graphs, Bellman-Ford algorithm [CLRS, Ch24.1]

Next Lecture. Dynamic Programming:

- Weighted interval scheduling
- Knapsack
- Longest common subsequence [CLRS] Ch15.4
- RNA secondary structure