6.8 Shortest Paths

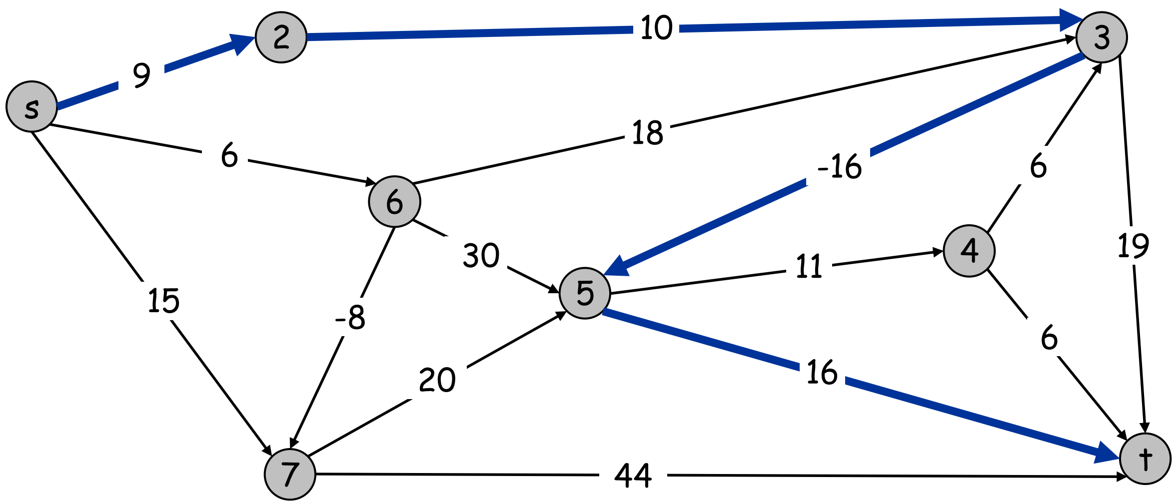
- Shortest path problem
- DP approach
  - analysis
  - recursive function
  - iterative algorithm
  - improved algorithm (Bellman-Ford)

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. Find negative cycles!

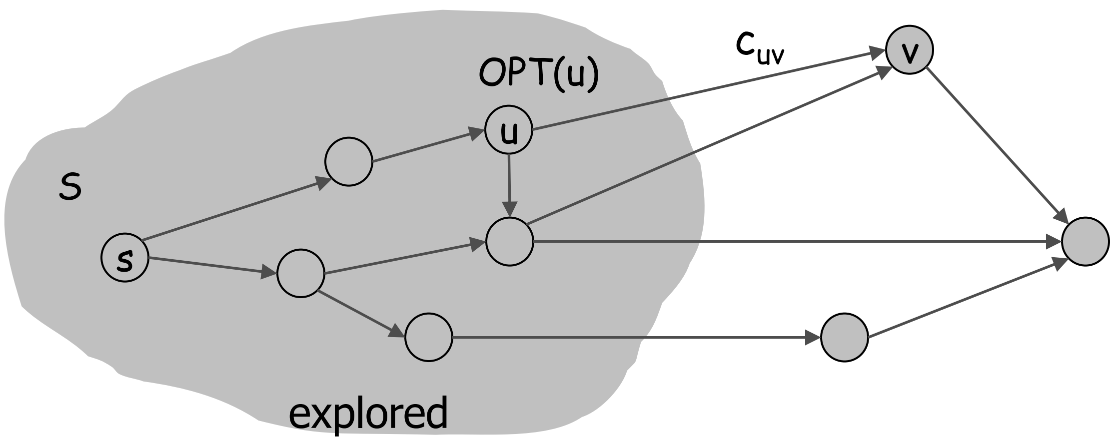


We have already seen an algorithm for shortest paths…

Dijkstra's Algorithm

For each unexplored node, explicitly maintain $\pi(v) = \min_{(u,v)\,:\,u \in S} OPT(u) + c_{uv}$

- Next node to explore = node with minimum $\pi(v)$.
- Add v to S, $OPT(v) = \pi(v)$.
- For each incident edge e = (v, w), update $\pi(w) = \min \{\pi(w),\ \pi(v) + c_{vw}\}$.

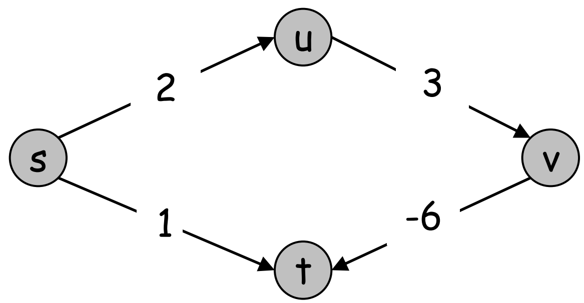


Shortest Paths: Failed Attempts

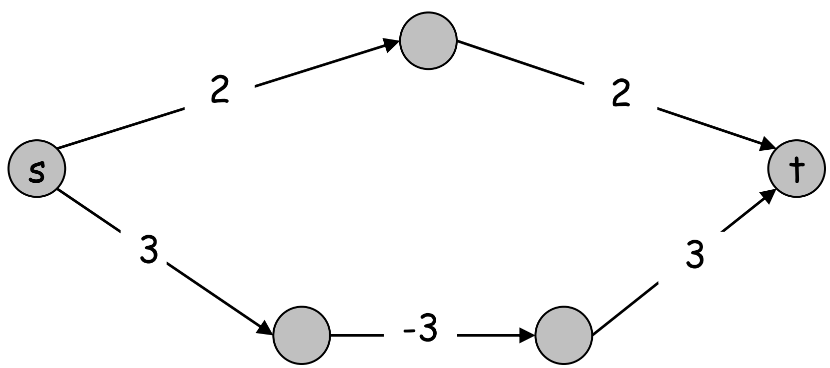
Q. What is the distance from s to t according to Dijkstra when run on this graph?

Dijkstra:
Next node to explore
= node with minimum $\pi(v)$.

$$\pi(v) = \min_{(u,v)\,:\, u \in S} OPT(u) + c_{uv}$$



Q. What can we do to fix Dijkstra's shortest path algorithm?



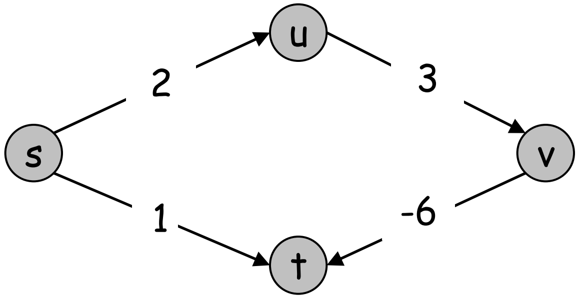
Shortest Paths: Failed Attempts

Dijkstra:
Next node to explore
= node with minimum $\pi(v)$.

$$\pi(v) = \min_{(u,v)\,:\,u \in S} OPT(u) + c_{uv}$$

Q. What is the distance from s to t according to Dijkstra when run on this graph?

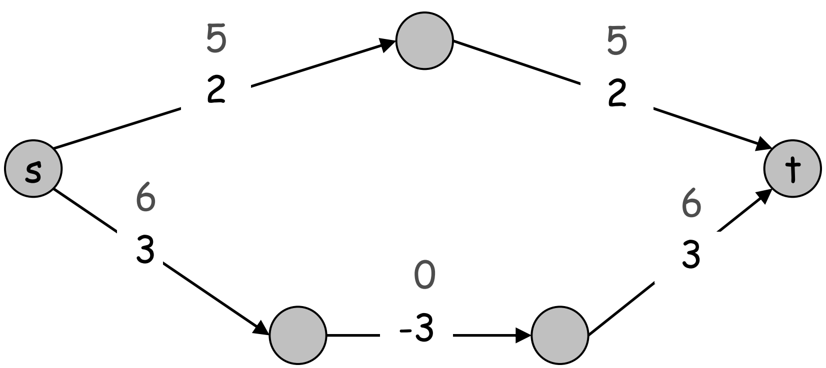
A. +1 instead of -1



Q. What can we do to fix Dijkstra's shortest path algorithm?

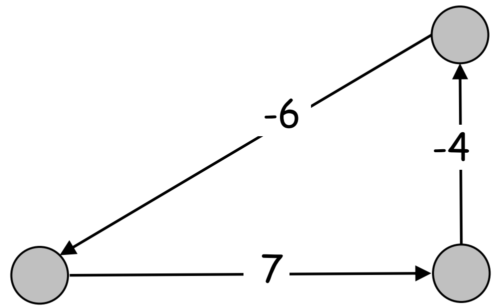
Re-weighting. What happens if we add a constant to every edge weight?

A. Dijkstra returns 4 (10) instead of 3 (12).



Shortest Paths: Negative Cost Cycles

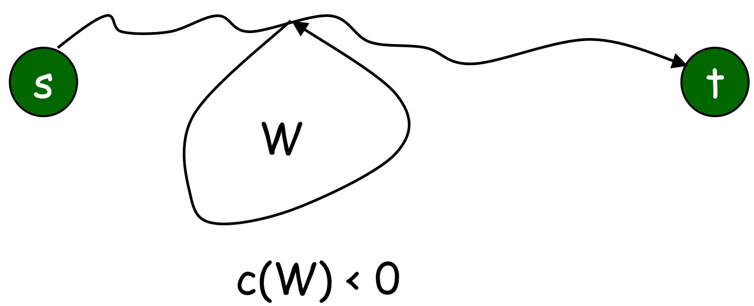
Negative cost cycle.



Q. What is the desired solution if some path from s to t contains a negative cost cycle?

A. No shortest s-t path exists. (Path needs to have finite steps.)

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.



Dynamic Programming Recipe

Recipe.

- Characterize structure of problem.
  - make one decision (eg for one object)
  - determine how it depends on subproblem(s)
- Recursively define value of optimal solution
- Compute value of optimal solution.
- Construct optimal solution from computed information.

Dynamic Programming Summary

Dynamic programming techniques.

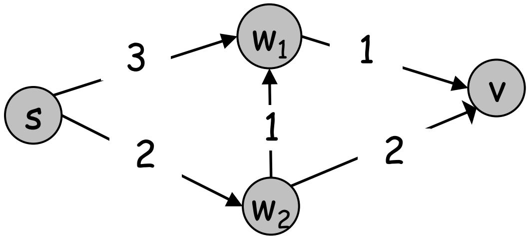
- Binary choice: weighted interval scheduling.
- Multi-way choice: segmented least squares.
- More subproblems per choice (intervals): RNA secondary structure.
- Two inputs: sequence alignment
- Adding a new variable: knapsack, shortest-path (Bellman-Ford).

Top-down vs. bottom-up (or from left to right): different people have different intuitions.

Shortest Paths: Dynamic Programming: False Start

DP recipe. Characterize structure of finding shortest path from s to v:

Q. Which decision to make in one step?



Q. How does it depend on subproblems?

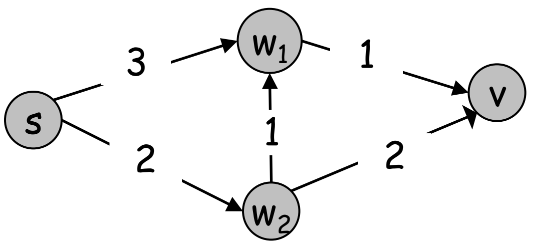
Q. What is the recursive definition of the optimal solution?

TU Delft

Shortest Paths: Dynamic Programming: False Start

DP recipe. Characterize structure of finding shortest path from s to v:

Q. Which decision to make in one step?
A. "Via which edge (w,v) is the shortest path from s?"

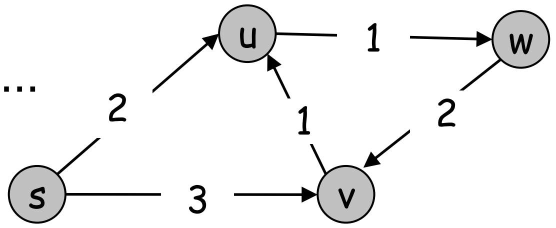


Q. How does it depend on subproblems?
A. If we use (w,v): "what is (length of) shortest path to w?"

Q. What is the recursive definition of the optimal solution?

$$OPT(v) = \begin{cases} 0 & \text{if } v = s \\ \displaystyle\min_{(w,v)\in E} \{ OPT(w) + c_{wv} \} & \text{otherwise} \end{cases}$$

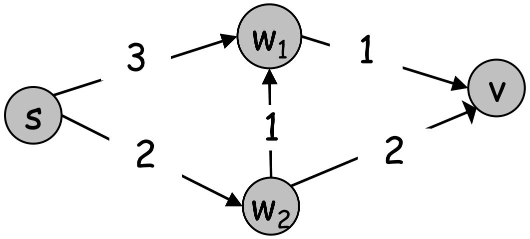
However, OPT(w) is not really a (smaller) sub-problem…
How can we make sure that this is really smaller?



Shortest Paths: Dynamic Programming

DP recipe. Characterize structure of finding shortest path from s to v:

Q. Which decision to make in one step?



Q. How does it depend on subproblems?

Q. What is the recursive definition of the optimal solution?

TU Delft

Shortest Paths: Dynamic Programming

DP recipe. Characterize structure of finding shortest path from s to v:

Q. Which decision to make in one step?
A. "Via which edge (w,v) is the shortest path with at most i edges?"

Q. How does it depend on subproblems?
A. If we use (w,v): what is shortest path to w using at most i-1 edges?

Q. What is the recursive definition of the optimal solution?

Shortest Paths: Dynamic Programming

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

- OPT(i,v) uses an edge (w,v) and
  - then selects best s-w path using at most i-1 edges (recursively)

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

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

Remark. The approach in this slide differs from Kleinberg (p293-294):

- we reason backwards from t and have s as the base case
- we define OPT(i,s)=0 for all and don't need OPT(i-1,v)

TU Delft

Shortest Paths: Implementation

```
Shortest-Path(G, s) {
   foreach node v ∈ V
      M[0, v] ← ∞
   for i = 0 to n-1
      M[i, s] ← 0

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

Q. What is the time complexity?

Q. What is the space complexity?

Shortest Paths: Implementation

```
Shortest-Path(G, s) {
   foreach node v ∈ V
      M[0, v] ← ∞
   for i = 0 to n-1
      M[i, s] ← 0

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

Q. What is the time complexity?

A. Θ(mn) time

Q. What is the space complexity?

A. $\Theta(n^2)$ space.

Shortest Paths: Practical Improvements

```
Shortest-Path(G, s) {
   foreach node v ∈ V
      M[0, v] ← ∞
   for i = 0 to n-1
      M[i, s] ← 0

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

Q. How to reduce use of memory?

Shortest Paths: Practical Improvements

```
Shortest-Path(G, s) {
   foreach node v ∈ V
      M[0, v] ← ∞
   for i = 0 to n-1
      M[i, s] ← 0

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

Q. How to reduce use of memory?

A. Maintain only one array M[v] = length of shortest s-v path that we have found so far (using i or i-1 edges).

Q. But how then to recover found solution (path)?

A. Maintain predecessor array **`predecessor[v]`** = best step found so far

Observation. No need to check edges of the form (w, v) unless M[w] changed in previous iteration. (“push-based”)

Bellman-Ford: Efficient Implementation

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

   M[s] = 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 (w, v) ∈ E {
               if (M[v] > M[w] + c_wv) {
                  M[v] ← M[w] + c_wv
                  predecessor[v] ← w
               }
            }
         }
      }
   }
}
```

Q. When can we stop?

Bellman-Ford: Efficient Implementation

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

   M[s] = 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 (w, v) ∈ E {
               if (M[v] > M[w] + c_wv) {
                  M[v] ← M[w] + c_wv
                  predecessor[v] ← w
               }
            }
         }
      }
      If no M[w] value changed in iteration i, stop.
   }
   return M[t]
}
```

Shortest Paths: Practical Improvements

Theorem. Throughout the algorithm,

- M[v] is length of some s-v path, and
- after i rounds of updates, the value M[v] is no larger than the length of shortest s-v path using $\leq$ i edges.
- Almost the same algorithm for calculating shortest v-t path (see book).

Q. Spot the differences! (at home)

Overall impact. [Bellman-Ford algorithm, 1958]

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