CS 381 – FALL 2019

Week 10.1, Monday, Oct 21

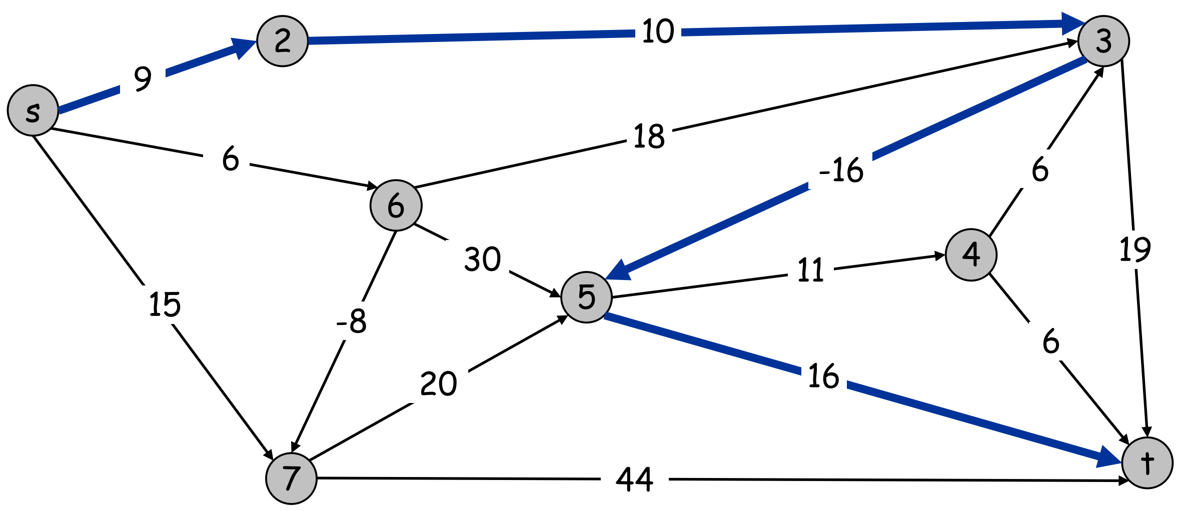
Homework 5 Due October 26 @ 11:59PM on Gradescope

6.8 Shortest Paths (with Negative Weights)

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

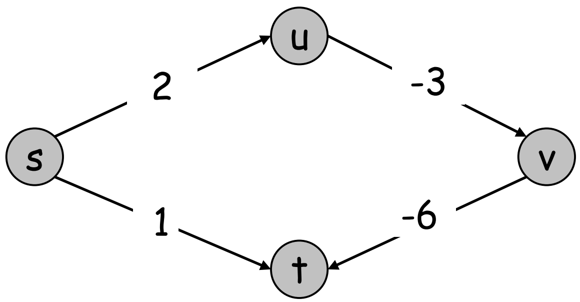
Ex. Nodes represent agents in a financial setting and c_{vw} is cost of transaction in which we buy from agent v and sell immediately to w.



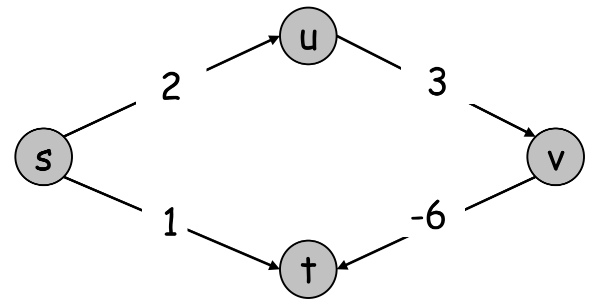
Clicker Question

For which of the following directed graphs will Dijkstra find the shortest path from s to t?

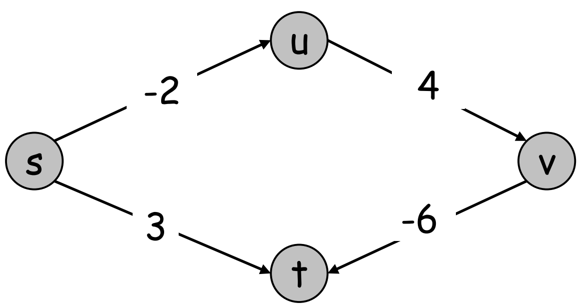
A



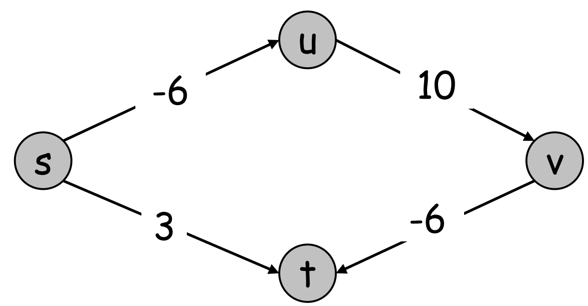
B



C



D

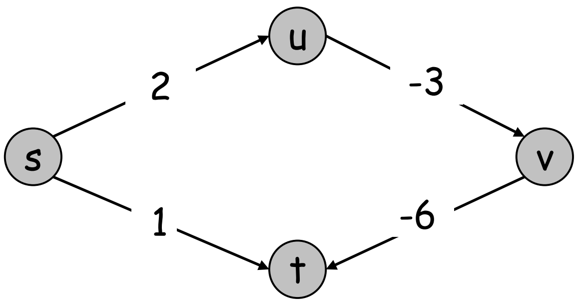


E: None of the above

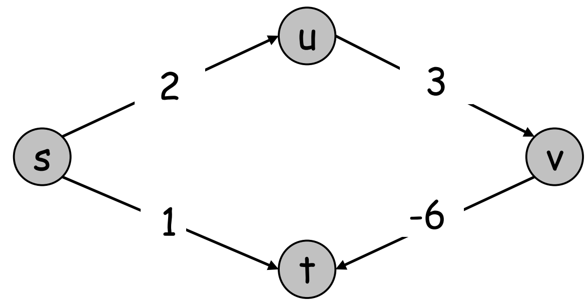
Clicker Question

For which of the following directed graphs will Dijkstra find the shortest path from s to t?

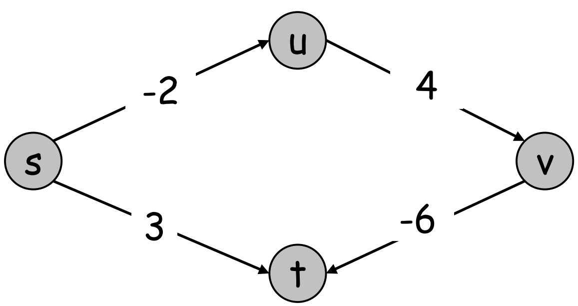
A



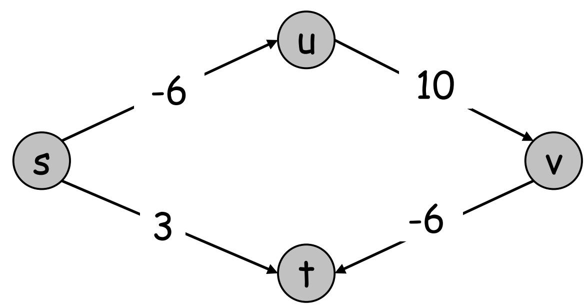
B



C

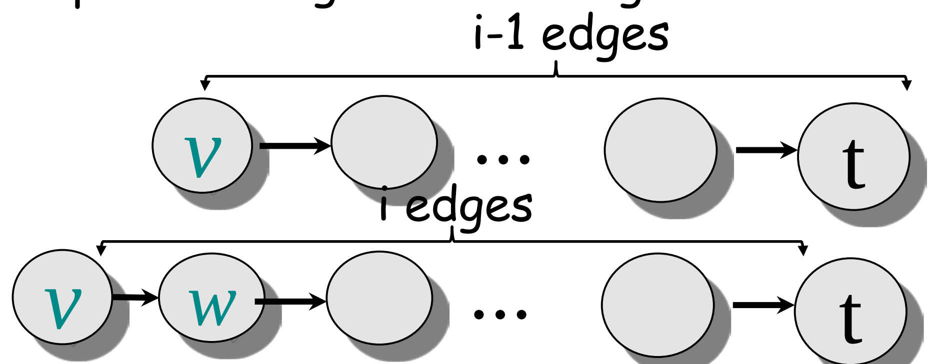


D



E: None of the above

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

$$\mathrm{OPT}(\mathrm{i},\mathrm{v}) = \begin{cases} 0 & i = 0, v = t \\ \infty & i = 0, v \neq t \\ \min\left\{\mathrm{OPT}(\mathrm{i}-1,\mathrm{v}),\ \min_{(v,w)\in E}\{\mathrm{OPT}(\mathrm{i}-1,\mathrm{w}) + \mathrm{c}_{\mathrm{vw}}\}\right\} & \text{otherwise} \end{cases}$$

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

Fact: If there is a negative cycle then OPT(n,v) < OPT(n-1,v) for some node v

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 }
}
```

M[i-1,v] no longer used

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

Finding the shortest paths. Maintain a "successor" for each table entry i.e. if (v,w) is the first edge on the shortest i-edge path P from v to t then Successor[v] = w

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.
   }
}
```

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.

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 ≤ i edges.

Overall impact.

- Memory: O(n) additional memory beyond O(m+n) for input (adjacency list).
- Running time: O(mn) worst case, but substantially faster in practice.

Detect Negative Cycle.

- Run Outer Loop for n iterations (**for i = 1 to ~~n-1~~ n)**
- **If any M[v]changes in iteration n → negative cycle**

All Pairs Shortest Path

Input:

Graph G=(V,E), directed and weighted, with weights w(e)

Output:

Shortest path matrix D, where d(u,v) represents the cost of the shortest paths from u to v

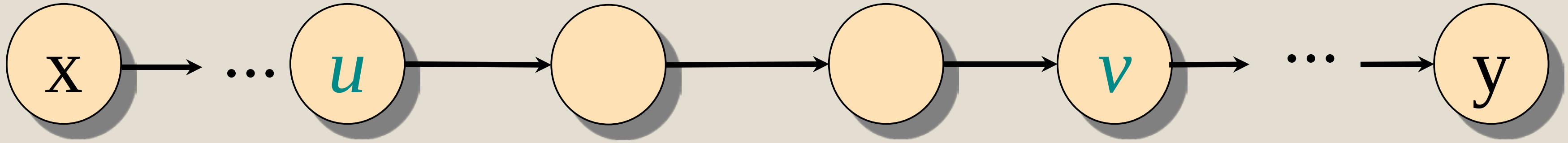
- The vertices on a shortest path are typically generated on a need basis

One solution: solve n single-source problems

- No negative weights: $O(nm + n^2 \log n)$ time using Dijkstra

How do we get started on a dynamic programming formulation?

- We compute n^2 entries of matrix D
- We do not know how many edges the shortest path from u to v contains
- We do not know in what order vertices are visited
- The principle of optimality holds for subpaths in a shortest path



How do we build up solutions in a systematic way?

A First DP Solution

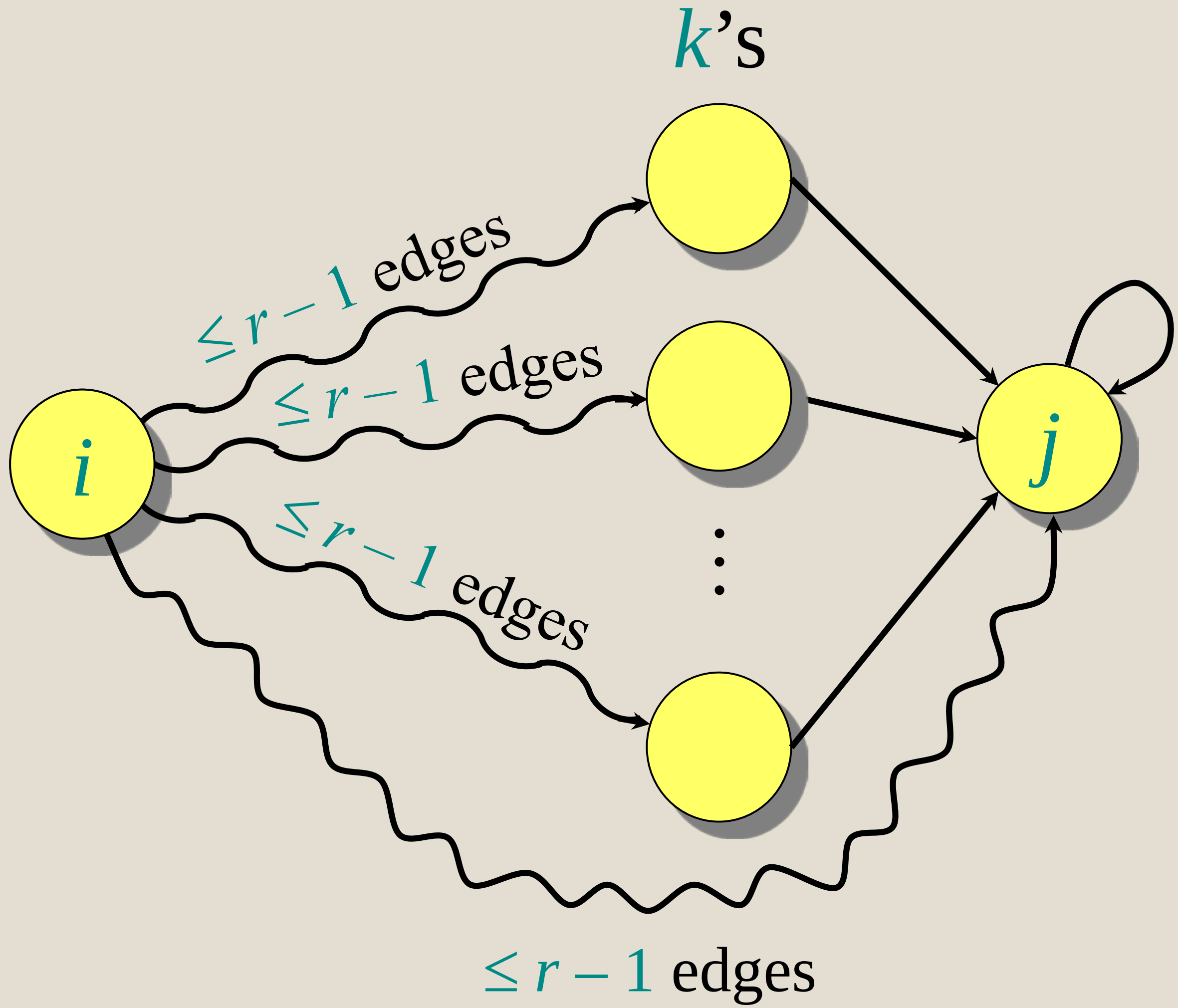
Input is adjacency matrix A; no negative cycles

$d(i,j)^r$ = cost of shortest path from i to j

using at most r edges

We know

- $d(i,i)^0 = 0$ and $d(i,j)^0 = \infty$ for $i \neq j$
- determine $d(i,j)^r$ from earlier computed values
- $d(i,j)^{n-1}$ represent the shortest paths

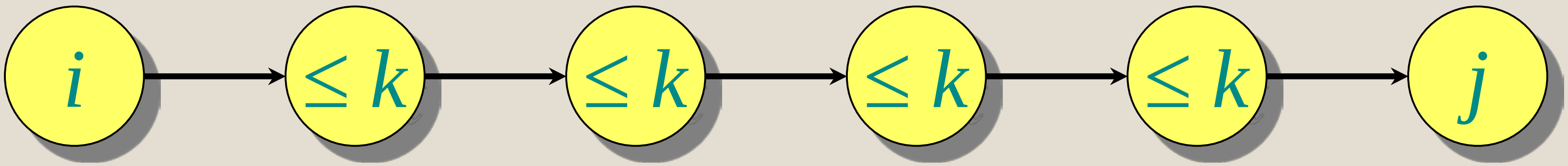


O(n^2m) time to fill in DP table

$$d(i,j)^r = \min_{\substack{k:\,(k,j)\in E \\ \text{(or } k=j)}} \{ d(i,k)^{r-1} + w(k,j) \}$$

Floyd-Warshall algorithm

Define $c_{ij}^{(k)}$ = weight of a shortest path from i to j with intermediate vertices belonging to the set $\{1, 2, \ldots, k\}$.



Thus, $\delta(i, j) = c_{ij}^{(n)}$.

Floyd-Warshall all-pair shortest paths

Input is an adjacency matrix A

$c(i,j)^k$ = cost of the shortest path from i to j with intermediate vertices belonging to set {1, 2, 3, …, k}

- $c(i,j)^0 = A(i,j)$
- $c(i,j)^n$ is the final answer

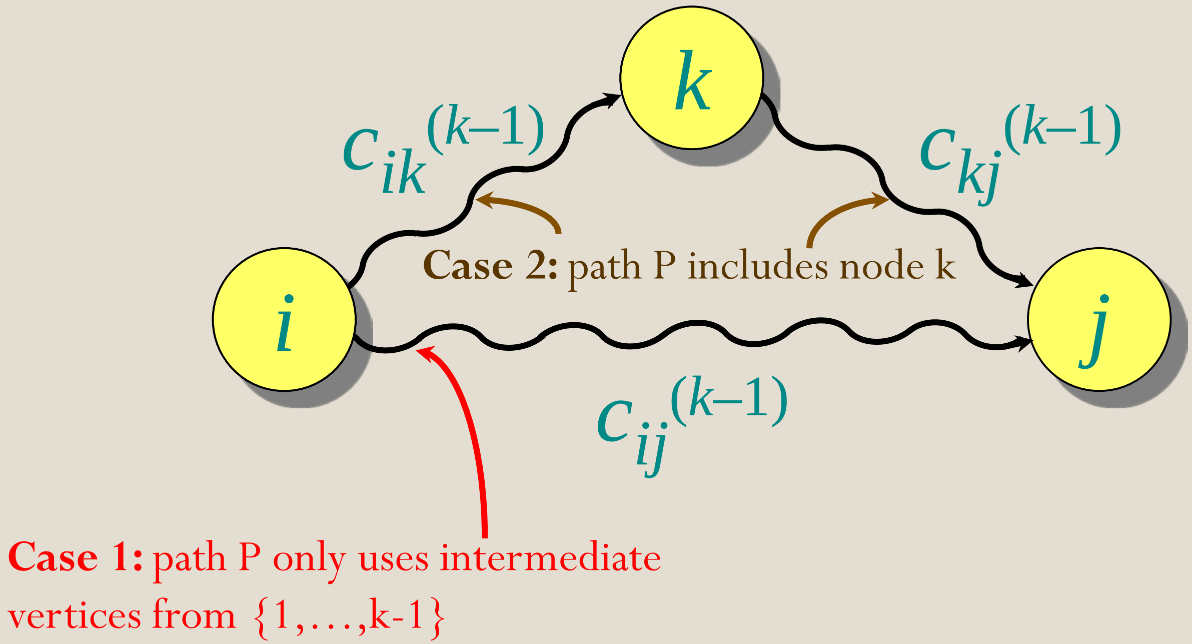
Recursive formulation:

$$\mathbf{c(i,j)^k = \min \{ c(i,j)^{k-1}, c(i,k)^{k-1} + c(k,j)^{k-1} \}}$$

$O(n^3)$ time algorithm

Floyd-Warshall recurrence

$$c_{ij}^{(k)} = \min_k \{c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$



P := shortest path from i to j such that intermediate vertices are in set $\{1, 2, \ldots, k\}$

Pseudocode for Floyd-Warshall

for $k \leftarrow 1$ **to** n
    do for $i \leftarrow 1$ **to** n
        do for $j \leftarrow 1$ **to** n
            do if $c_{ij} > c_{ik} + c_{kj}$
                then $c_{ij} \leftarrow c_{ik} + c_{kj}$ } ***relaxation***

- Can drop the superscripts (extra relaxations can't hurt)
- $\Theta(n^3)$ time.
- Simple to code.