Lecture 10
All-Pairs Shortest Paths Continued

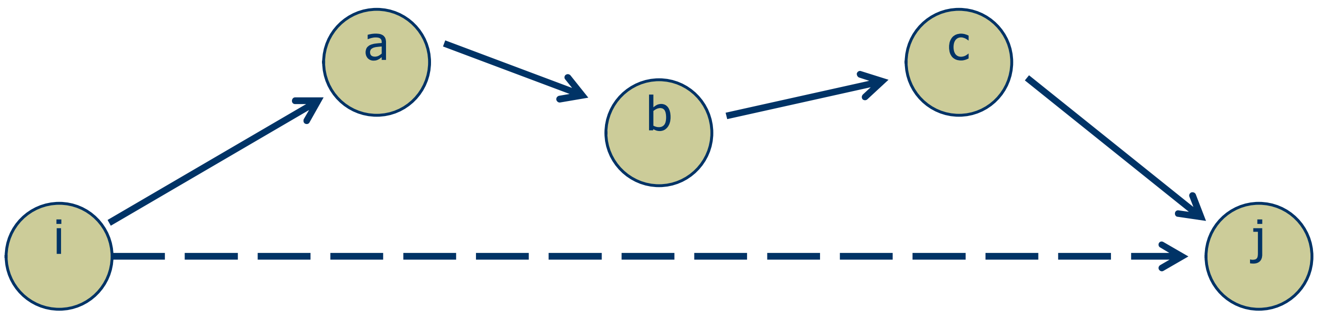
- Transitive closure
- Johnson's algorithm

Transitive closure (the problem)

- Find out *whether* there is a path from i to j and compute

$$G^* = (V, E^*),$$

where $E^* = \{(i,j)$: there is a path from i to j in $G\}$



Transitive closure

- One way:

  set $w_{ij} = 1$ and

  run the Floyd-Warshall algorithm

- *running time* $O(n^3)$

Transitive closure

- Another way:
  substitute “+” and “min” by AND and OR
  in Floyd’s algorithm

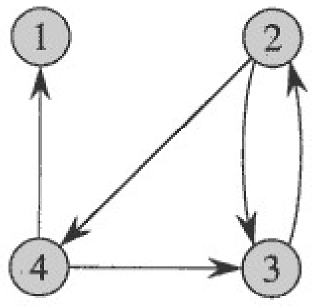
- *running time* $O(n^3)$

Transitive closure (pseudo-code)

TRANSITIVE-CLOSURE(G)

$n \leftarrow |V[G]|$
for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
        do if $i = j$ or $(i, j) \in E[G]$
            then $t_{ij}^{(0)} \leftarrow 1$
            else $t_{ij}^{(0)} \leftarrow 0$
for $k \leftarrow 1$ **to** n
    do for $i \leftarrow 1$ **to** n
        do for $j \leftarrow 1$ **to** n
            do $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
return $T^{(n)}$

Transitive closure



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Figure 25.5 A directed graph and the matrices $T^{(k)}$ computed by the transitive-closure algorithm.

Johnson's algorithm (idea)

- If { all edge weights are non-negative }

  Invoke Dijkstra's algorithm *V* times,
- else if { there are negative weight edges but no negative weight cycles }

  Reweight graph so that

  1. **all edge weights are non-negative, and that**
  2. **shortest path does not change after reweighting**

  Invoke Dijkstra's algorithm for each vertex as source.

* Time Complexity: $O(V^2\log V+VE)$ //using Fibonacci heap

* If *G* is **sparse**, better than Floyd-Warshall

Reweighting Technique (I)

- Lemma 25.1 (reweighting does not change shortest paths):

Given a weighted, directed graph $G=(V,E)$ with weight function w. Let $h: V \rightarrow R$ be any function. For each edge $(u,v) \in E$, define

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

Let $p=<v_0,\ldots,v_k>$ be any path from v_0 to v_k. Then ***p* is a shortest path using weight function *w* if and only if it is a shortest path using weight function $\hat{w}$. That is $w(p)=\delta(v_0,v_k)$ if and only of $\hat{w}(p)=\hat{\delta}(v_0,v_k)$**. Also, G has a negative-weight cycle with w if and only if G has a negative-weight cycle with $\hat{w}$.

Proof

Proof: 1. For any path p from v_0 to v_k, we have

$$\hat{w}(p)=w(p)+\boldsymbol{h(v_0)-h(v_k)},$$

so, p^* is shortest with w if and only if it is shortest with $\hat{w}$.

2. Consider any cycle $C=<v_0,\ldots,v_k>$, we have

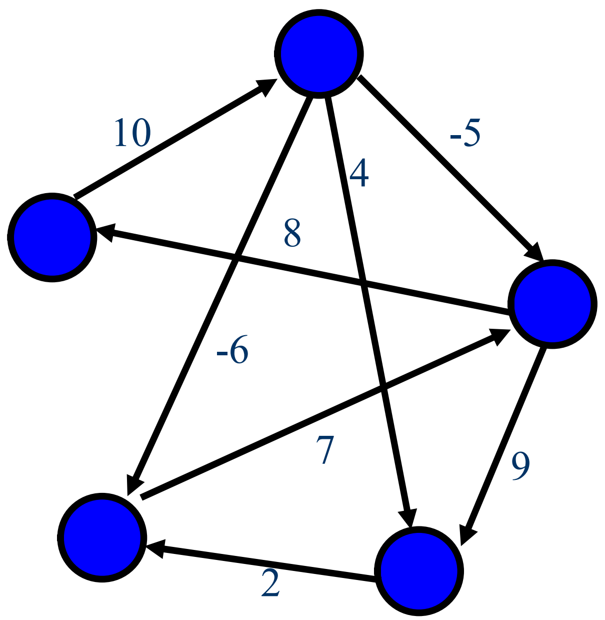
$$\hat{w}(C) = w(C)+\boldsymbol{h(v_0)-h(v_0)} = \boldsymbol{w(C)},$$

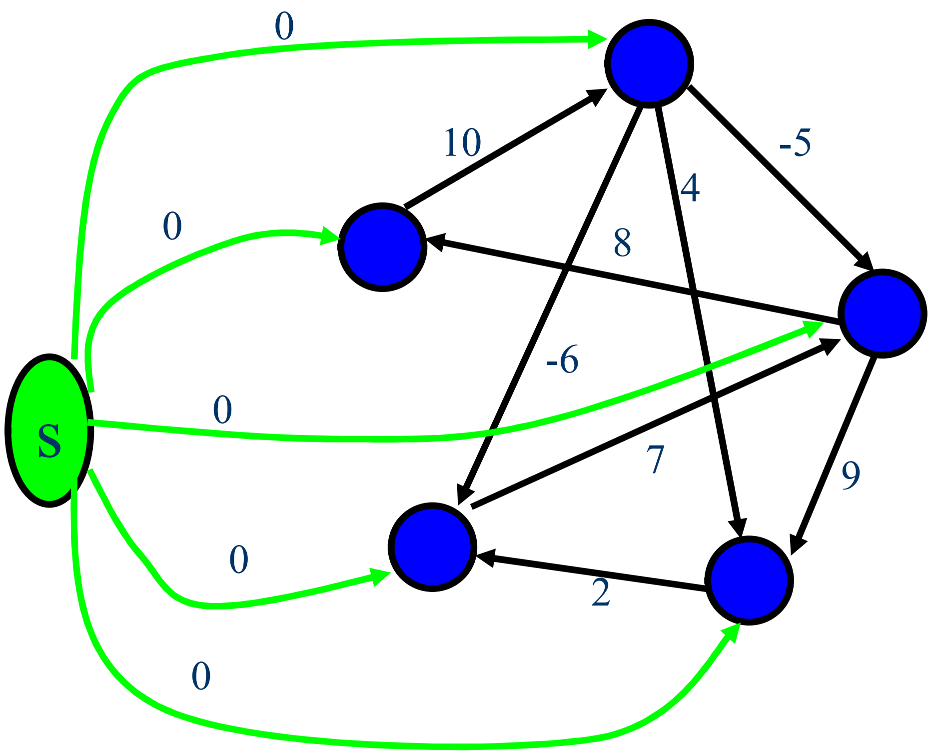
G has a negative cycle with w if and only if it has a negative cycle with $\hat{w}$.

Reweighting Technique (II)

- Producing nonnegative weights by reweighting

  Given a weighted, directed graph $G = (V, E)$ with w, we make a new graph G' $= (V',E')$, where $V' = V \cup \{s\}$, s is not in V, and $E' = E \cup \{(s,v) : v \text{ is in } V\}$. $w(s,v)$ is set to 0, for all v in V.





Reweighting Technique (III)

- Producing nonnegative weights by reweighting

Suppose that G and G' have no negative weight cycle. Define $h(v) = \delta(s,v)$ for all v in V'. By the triangle inequality (Lemma 24.10), we have $h(v) <= h(u) + w(u,v)$ for all edges (u,v) in E'. Thus, we have

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v) >= 0$$

Critical Steps

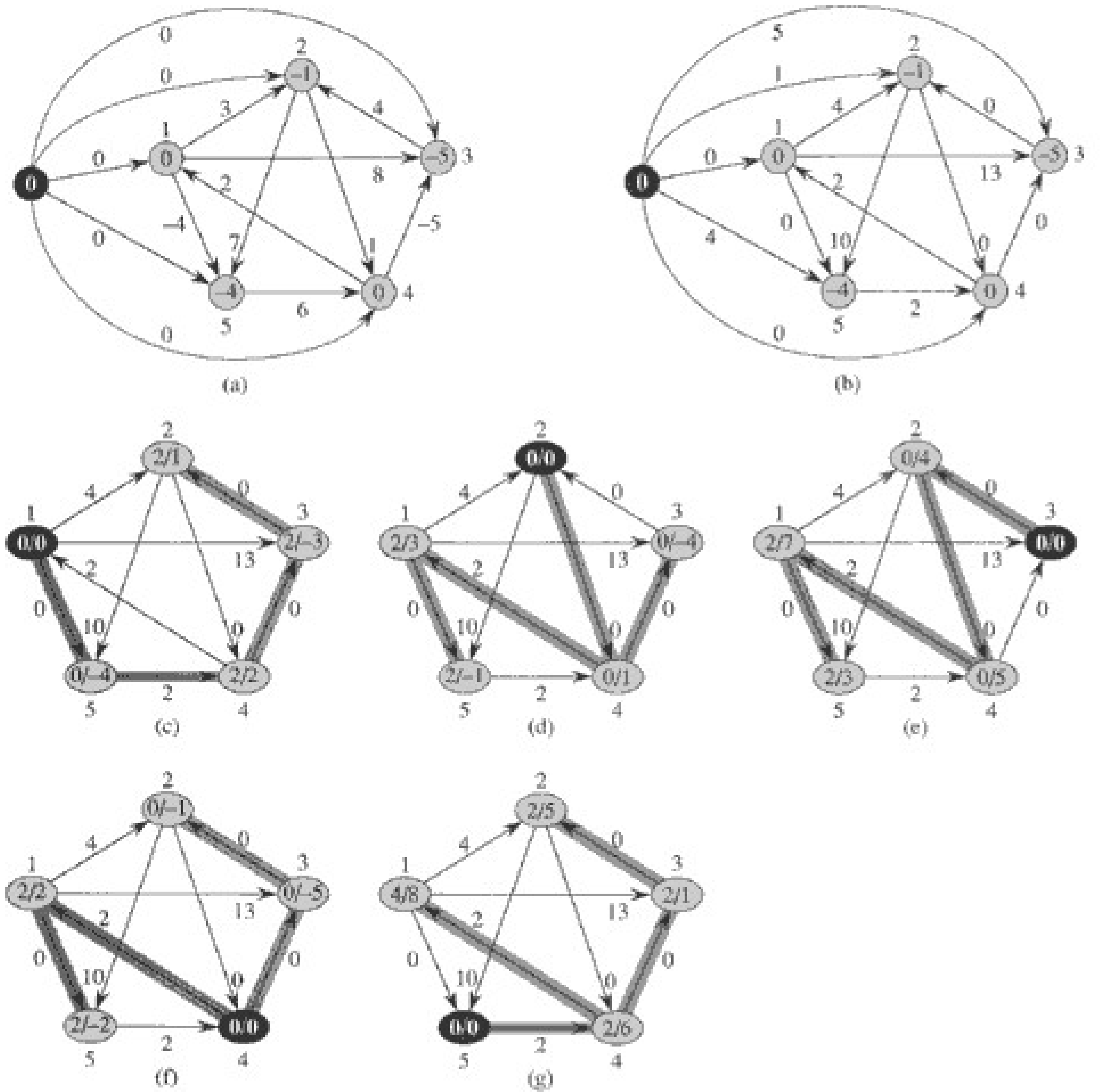
- How to compute $\delta(s,v)$?
  - Bellman-Ford algorithm.
  - Do you remember?
- How to compute all $\hat{\delta}(u,v)$?
  - Run Dijkstra algorithm |V| times.
  - $\delta(u,v)=\hat{\delta}(u,v)+h(v)-h(u)$

Johnson's algorithm

JOHNSON(G)

compute G', where $V[G'] = V[G] \cup \{s\}$,
  $E[G'] = E[G] \cup \{(s, v) : v \in V[G]\}$, and
  $w(s, v) = 0$ for all $v \in V[G]$
if BELLMAN-FORD$(G', w, s) =$ FALSE
  then print "the input graph contains a negative-weight cycle"
  else **for** each vertex $v \in V[G']$
      do set $h(v)$ to the value of $\delta(s, v)$
        computed by the Bellman-Ford algorithm
    for each edge $(u, v) \in E[G']$
      do $\widehat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$
    for each vertex $u \in V[G]$
      do run DIJKSTRA$(G, \widehat{w}, u)$ to compute $\widehat{\delta}(u, v)$ for all $v \in V[G]$
        for each vertex $v \in V[G]$
          do $d_{uv} \leftarrow \widehat{\delta}(u, v) + h(v) - h(u)$
    return D

Johnson's (example)



Johnson's algorithm

- Time complexity:

1. Compute *G*': O(V)
2. Bellman-Ford: O(VE)
3. Re-weighting: O(E)
4. Dijkastra's: O(VlgV+E)

5. **Total: $O(V^2 \log V + VE)$**

6. If *G* is sparse, e.g., $E=O(V)$, then better than Floyd-Warshall.

All-pairs shortest paths problem algorithm comparison

algorithm	running time
Dijkstra‘s	$O(n^2 \log n + nm)$
Bellman-Ford	$O(n^2 m)$
matrix multiplication	$O(n^4)$
improved matrix mult.	$O(n^3 \log n)$
Floyd-Warshall Transitive closure	$O(n^3)$
Johnson‘s	$O(n^2 \log n + nm)$

Homework

- 25.3-4