COMP319 Algorithms 1
Lecture 16
Shortest Paths part 2

Instructor: Gil-Jin Jang

School of Electronics Engineering, Kyungpook National University

Textbook Chapters 25 and 26

Slide credits: 홍석원, 명지대학교; 김한준, 서울시립대학교;
J. Lillis, UIC; Roger Crawfis, CSE 680;
George Bebis, Analysis of Algorithms, CS 477/677
David Luebke, CS332, Virginia University

Table of Contents

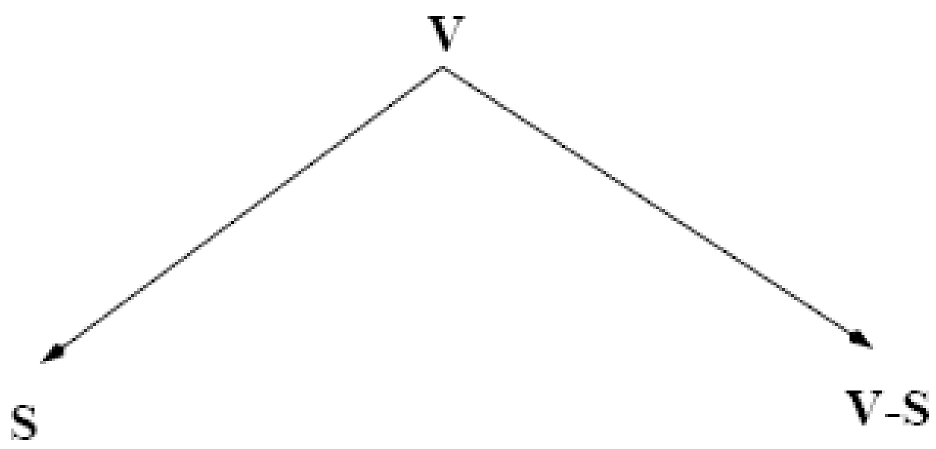
Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Single-Source Shortest Path without negative edge weight

DIJKSTRA'S ALGORITHM

Dijkstra's Algorithm

- Single-source shortest path problem:
  - No negative-weight edges: $w(u, v) > 0, \ \forall (u, v) \in E$
- Each edge is relaxed **only once!**
- Maintains two sets of vertices:



$d[v] = \delta(s, v)$ $\quad$ $d[v] > \delta(s, v)$

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Dijkstra's Algorithm (cont.)

- Vertices in $V–S$ reside in a min-priority queue
  - Keys in Q are estimates of shortest-path weights $d[u]$
- Repeatedly select a vertex $u \in V–S$, with the minimum shortest-path estimate $d[u]$
- Relax all edges leaving u
- Steps
  - Extract a vertex u from Q (i.e., with the highest priority)
  - Insert u to S
  - Relax all edges leaving u
  - Update Q

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Dijkstra Algorithm

All the edge weights are non-negative

```
Dijkstra(G, r)
▷ G=(V, E): given graph
▷ r: source vertex, given
{
    S ← {} ;                ▷ S : set of chosen vertices
    for each u∈V
        d_u ← ∞ ;
    d_r ← 0 ;
    while (S≠V){            ▷ repeat n times
        u ← extractMin(V-S, d) ;
        S ← S ∪ {u};
        for each v∈L(u) ▷ L(u) : set of vertices connected from u
            if (v∈V-S and d_v<d_u+w_u,v) then d_v ← d_u+w_u,v ;
    }                       relaxation
}

extractMin(Q, d)
{
    Returns a vertex u ϵ Q whose value of d is the smallest;
}
```

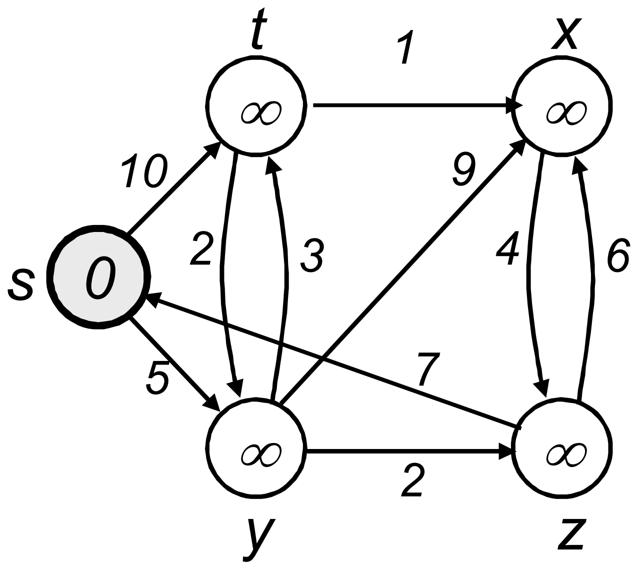
✓*Time complexity:* $O(|E|\log|V|)$

Using heap

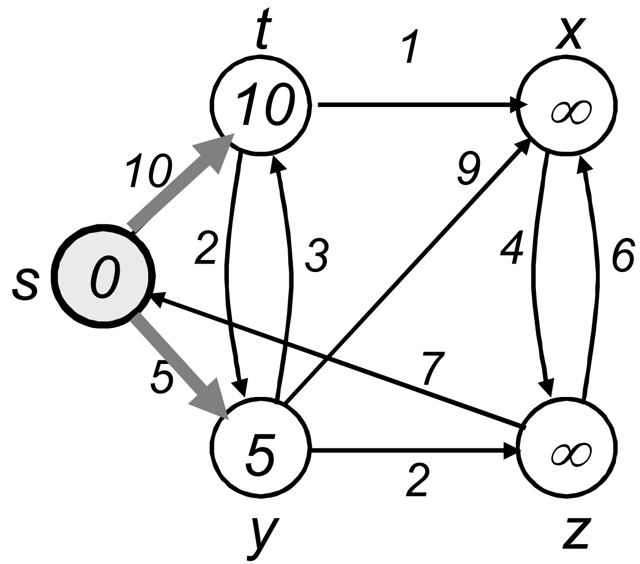
Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Dijkstra (G, w, s)

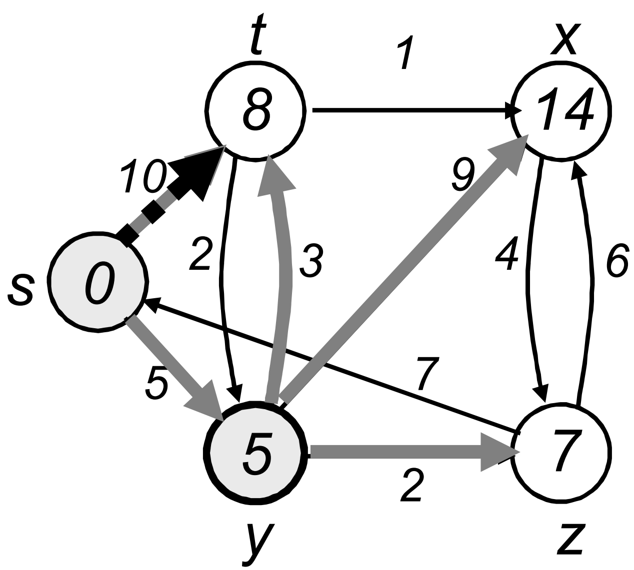
S=<> Q=<s,t,x,z,y>



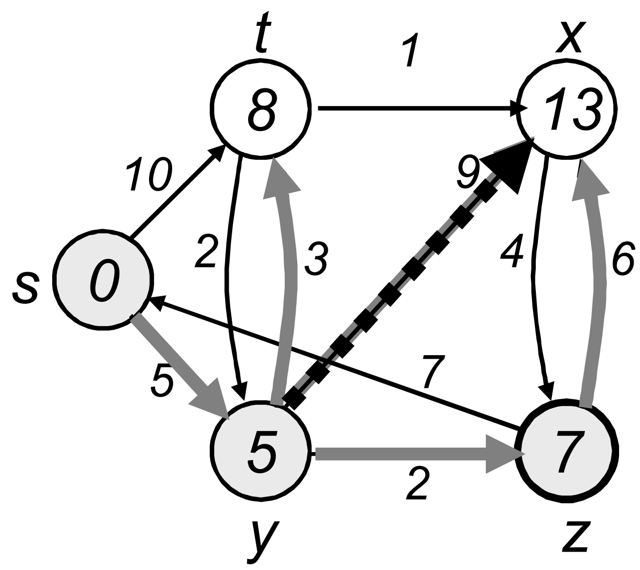
S=<s> Q=<y,t,x,z>



Example (cont.)



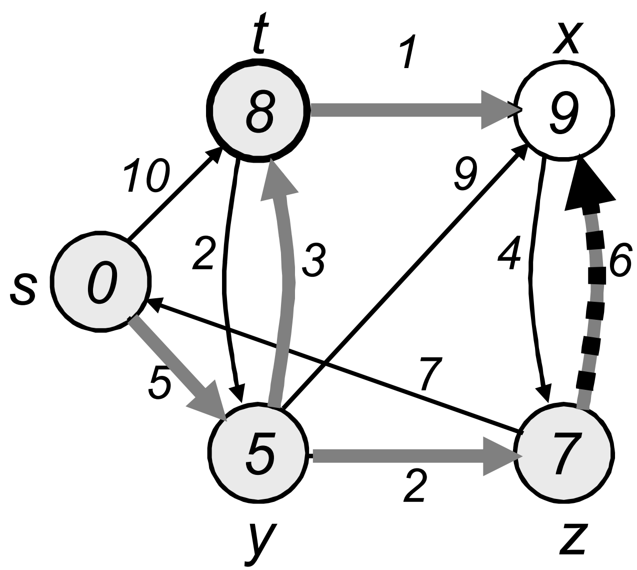
S=<s,y> Q=<z,t,x>



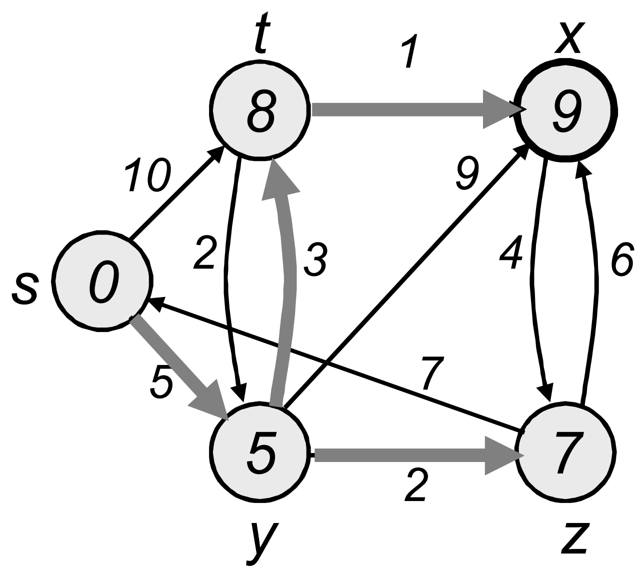
S=<s,y,z> Q=<t,x>

Example (cont.)

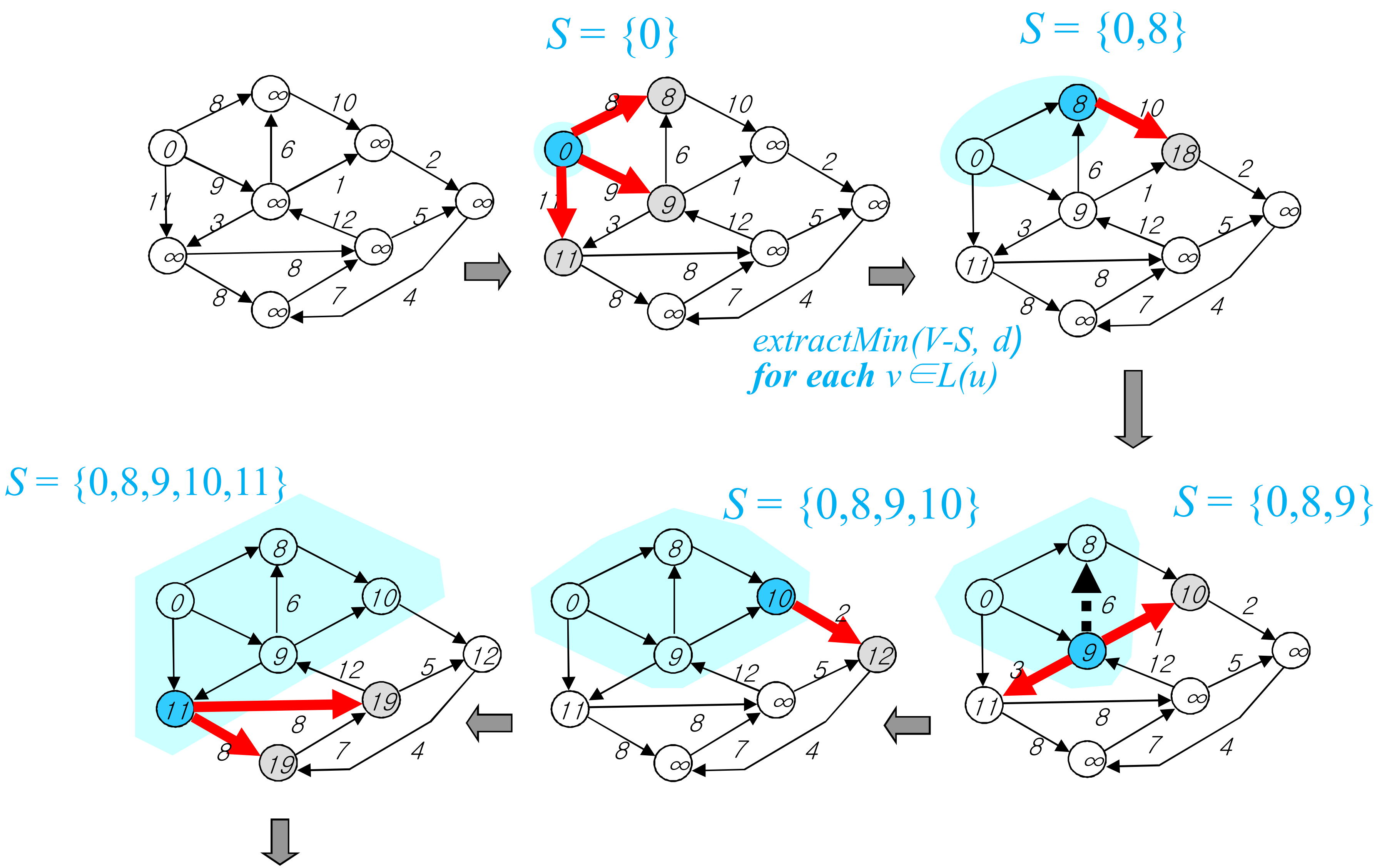
S=<s,y,z,t> Q=<x>



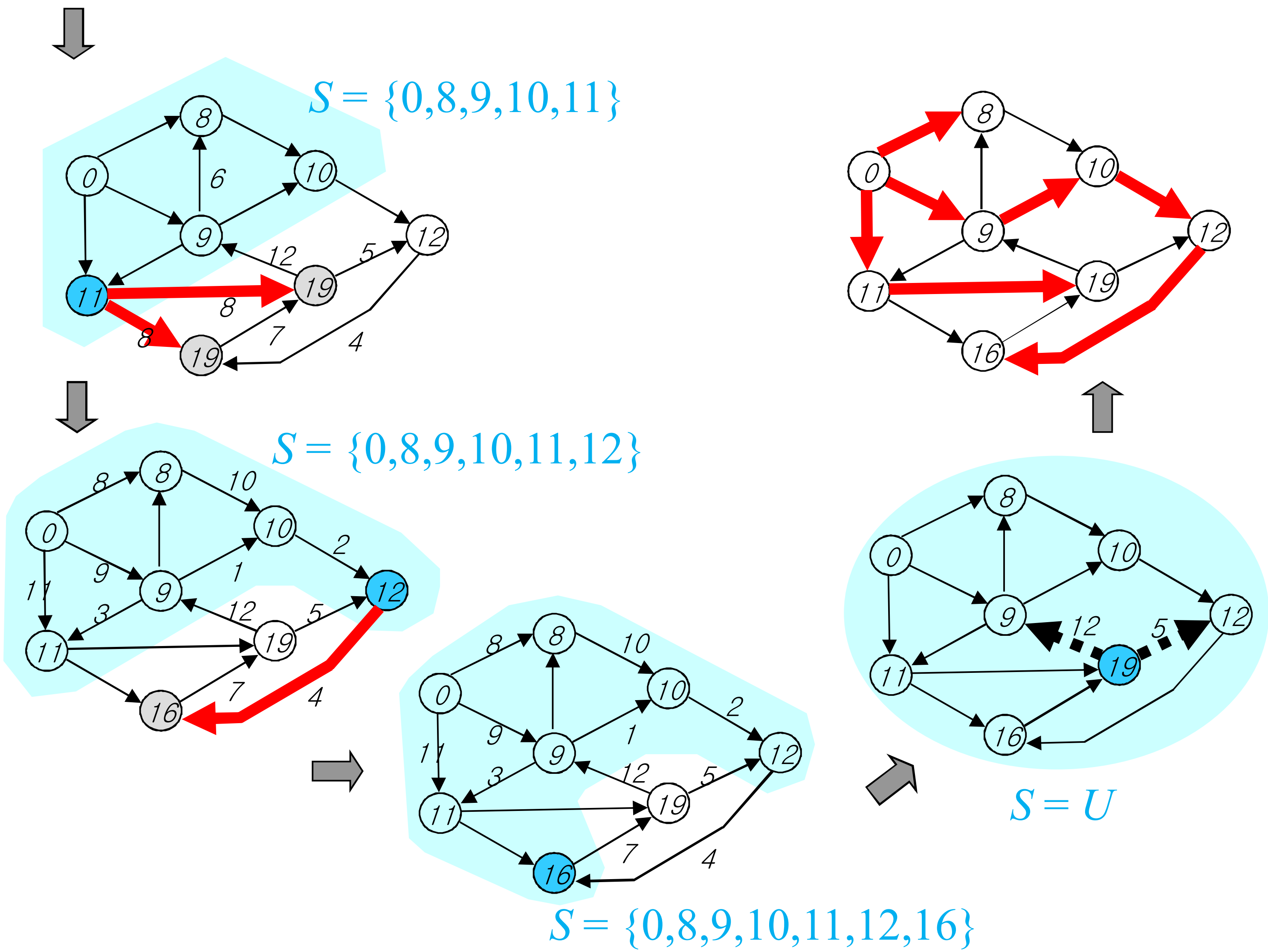
S=<s,y,z,t,x> Q=<>



Dijkstra Algorithm Illustration



Dijkstra Algorithm Illustration



Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Dijkstra (G, w, s)

```
INITIALIZE-SINGLE-SOURCE(V, s)          ← Θ(V)
S ← ∅
Q ← V[G]          ← O(V) build min-heap
while Q ≠ ∅          ← Executed O(V) times
    do u ← EXTRACT-MIN(Q)          ← O(lgV)
        S ← S ∪ {u}
        for each vertex v ∈ Adj[u]          ← O(E) times (total)
            do RELAX(u, v, w)
            Update Q (DECREASE_KEY)          ← O(lgV)
```

O(VlgV)

O(ElgV)

Running time: $O(V\lg V + E\lg V) = O(E\lg V)$

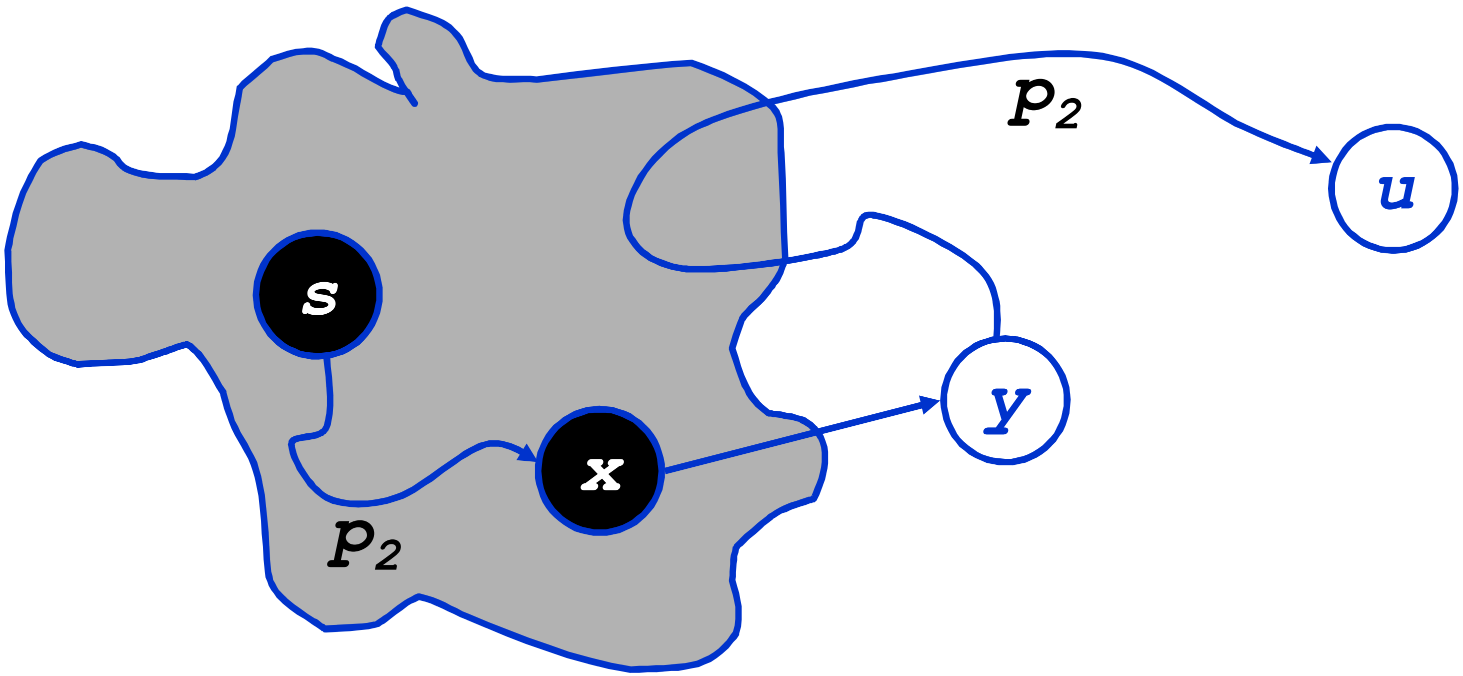
Binary Heap vs Fibonacci Heap

Running time depends on the implementation of the heap

	EXTRACT-MIN	DECREASE-KEY	Total
binary heap	O(lgV)	O(lgV)	O(ElgV)
Fibonacci heap	O(lgV)	O(1)	O(VlgV + E)

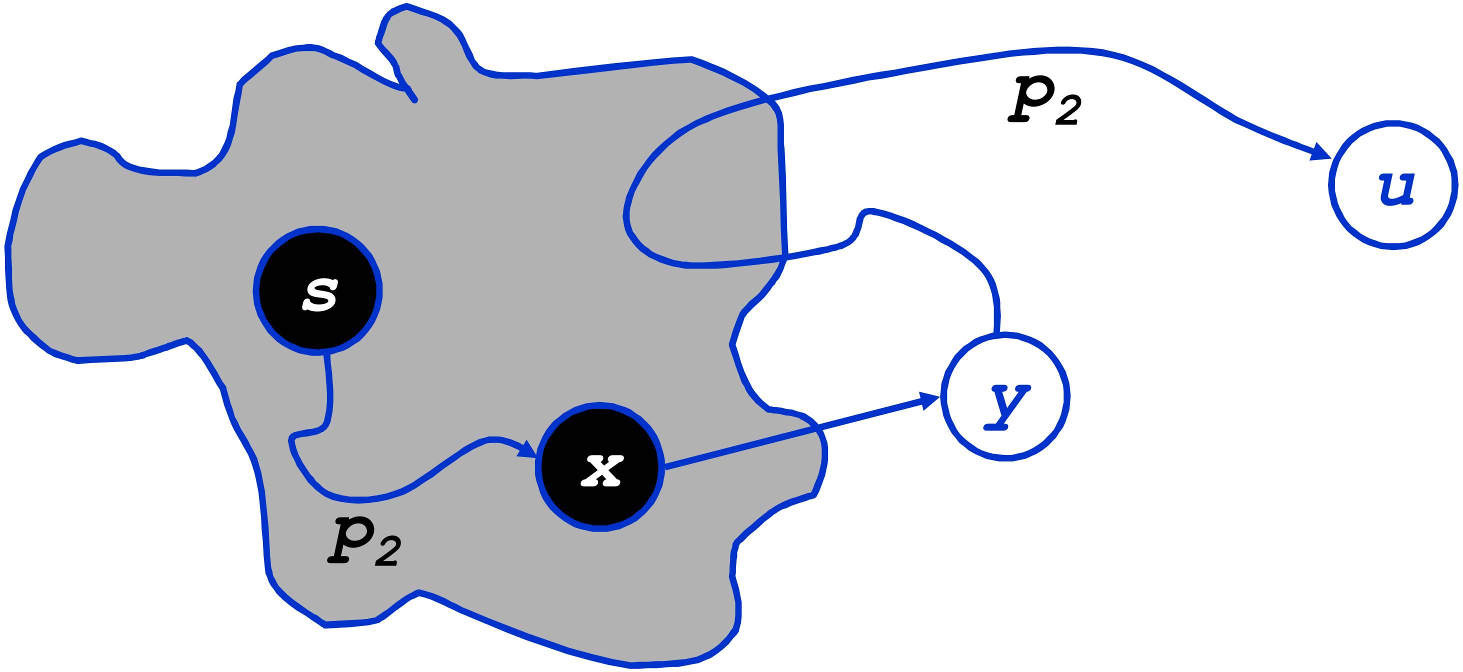
Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Correctness Of Dijkstra's Algorithm



- Note that d[v] ≥ δ(s,v) ∀v
- Let u be first vertex picked s.t. ∃ shorter path than d[u] ⇒d[u] > δ(s,u)
- Let y be first vertex ∈V-S on actual shortest path from s→u ⇒ d[y] = δ(s,y)
  - Because d[x] is set correctly for y's predecessor x ∈ S on the shortest path, and
  - When we put x into S, we relaxed (x,y), giving d[y] the correct value

Correctness Of Dijkstra's Algorithm



- Note that $d[v] \geq \delta(s,v)\ \forall v$
- Let u be first vertex picked s.t. $\exists$ shorter path than d[u] $\Rightarrow d[u] > \delta(s,u)$
- Let y be first vertex $\in$V-S on actual shortest path from $s \rightarrow u$ $\Rightarrow d[y] = \delta(s,y)$
- $d[u] > \delta(s,u)$
  $= \delta(s,y) + \delta(y,u)$ (*Why?*)
  $= d[y] + \delta(y,u)$
  $\geq d[y]$ But if d[u] > d[y], wouldn't have chosen u. Contradiction.

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

문병로, 쉽게 배우는 알고리즘

Cevdet Aykanat and Mustafa Ozdal, CS473, Bilkent Univ

Andreas Klappenecker

THE FLOYD-WARSHALL ALGORITHM

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

All-Pairs Shortest Path Problem

- Suppose we are given a directed graph *G* and a weight function **w**:
  - $G=(V,E)$; ω: E->R
- We assume that *G* does not contain cycles of weight 0 or less.
- The All-Pairs Shortest Path Problem asks to find the length of the shortest path between **any pair** of vertices in *G*.
- Applications
  - Network communications

All Pairs Shortest Paths (APSP)

given : directed graph $G = (V, E)$,

weight function $\omega: E \to R$, $|V| = n$

goal : create an $n \times n$ matrix $D = (d_{ij})$ of shortest path distances

i.e., $d_{ij} = \delta(v_i, v_j)$

trivial solution : run a SSSP algorithm n times, one for each vertex as the source.

All Pairs Shortest Paths (APSP)

- all edge weights are nonnegative : use Dijkstra's algorithm
  - PQ = linear array : $O(V^3 + VE) = O(V^3)$
  - PQ = binary heap : $O(V^2 \lg V + EV\lg V) = O(V^3 \lg V)$ for dense graphs
    - **better only for sparse graphs**
  - PQ = fibonacci heap : $O(V^2 \lg V + EV) = O(V^3)$ for dense graphs
    - **better only for sparse graphs**

- negative edge weights : use Bellman-Ford algorithm
  - $O(V^2 E) = O(V^4)$ on dense graphs

Floyd-Warshall Algorithm

- A dynamic programming solution that solved the APSP problem with time complexity O(n^3) for a graph with n vertices.
  - Dynamic programming

Adjacency Matrix Representation of Graphs

► n x n matrix W = (ω_{ij}) of edge weights :

$$\omega_{ij} = \begin{cases} \omega(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ \infty & \text{if } (v_i, v_j) \notin E \end{cases}$$

► assume $\omega_{ii} = 0$ for all $v_i \in V$, because

- no neg-weight cycle

  $\Rightarrow$ shortest path to itself has no edge,

  i.e., $\delta(v_i, v_i) = 0$

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Intermediate Vertices

- Without loss of generality, we will assume that $V=\{1,2,\ldots,n\}$, i.e., that the vertices of the graph are numbered from 1 to n.
- Given a path $p=(v_1, v_2,\ldots, v_m)$ in the graph, we will call the vertices v_k with index k in $\{2,\ldots,m\text{-}1\}$ the intermediate vertices of path p.
- If k is an intermediate vertex of path p, then we break p down into:
  - $i \rightarrow k \rightarrow j$ (p1 and p2)
  - p1 is a shortest path from i to k with all intermediate vertices in the set {1,2,…,k-1}
  - p2 is a shortest path from k to j with all intermediate vertices in the set {1,2,…,k-1}

All intermediate vertices in {1, 2, …, *k*-1}

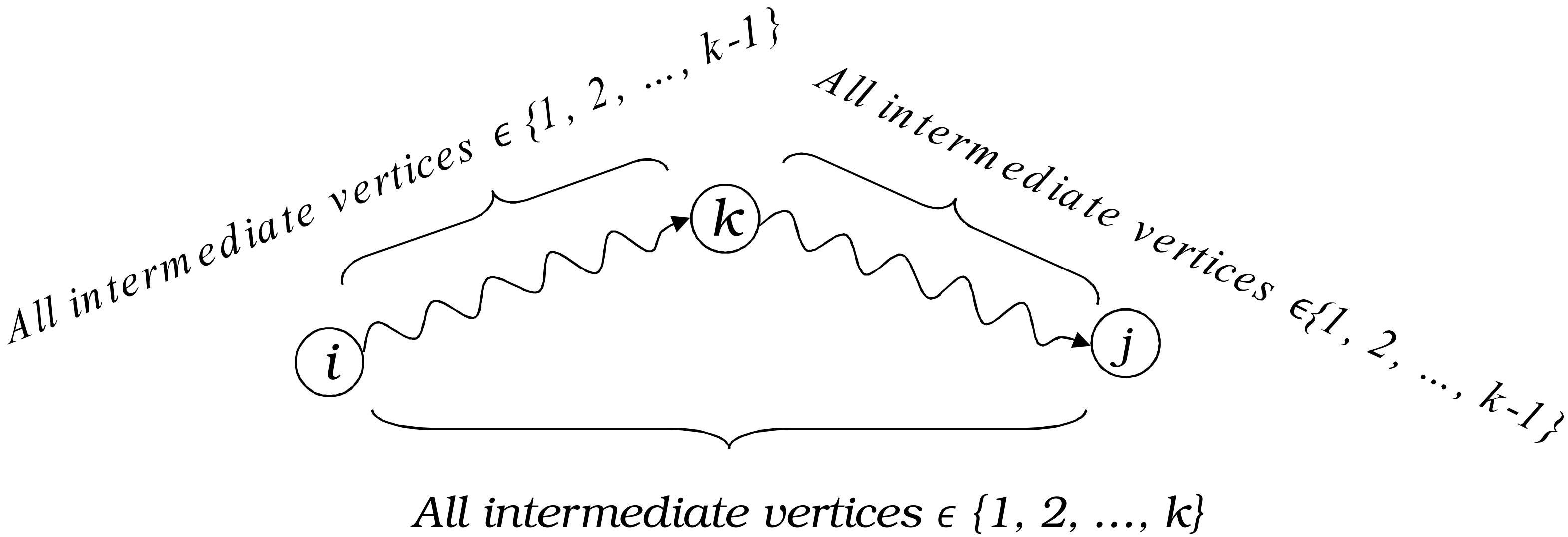


Figure 2. Path *p* is a shortest path from vertex *i* to vertex *j*, and *k* is the highest-numbered intermediate vertex of *p*. Path *p*1, the portion of path *p* from vertex *i* to vertex *k*, has all intermediate vertices in the set {1, 2, …, *k*-1}. The same holds for path *p*2 from vertex *k* to vertex *j*.

Key Idea

- Let $d_{ij}^{(k)}$ denote the length of the shortest path from i to j such that all intermediate vertices are contained in the set {1,…,*k*}.
- Consider a shortest path *p* from *i* to j such that the intermediate vertices are from the set {1,…,*k*}.
  - If the vertex *k* is not an intermediate vertex on *p*, then $d_{ij}^{(k)} = d_{ij}^{(k-1)}$.
  - If the vertex *k* is an intermediate vertex on *p*, then $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.
  - Interestingly, in either case, the subpaths contain merely nodes from {1,…,*k*-1}.

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Recursive Formulation

If we do not use intermediate nodes, i.e., when k=0, then

$$d_{ij}^{(0)} = w_{ij}$$

If k>0, then

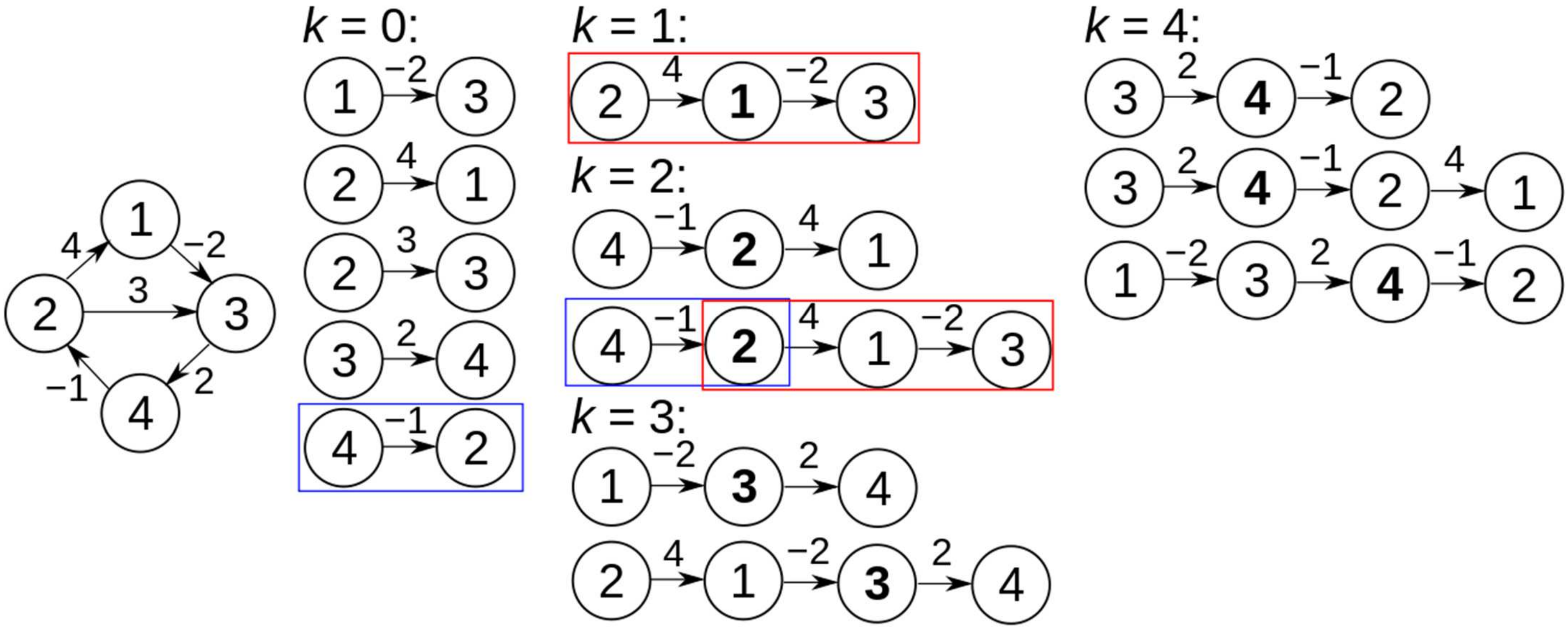
$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

The Floyd-Warshall Algorithm

```
Floyd-Warshall(W)
n = # of rows of W;
D^(0) = W;      /* initialization, equivalent to the next nested loop */
/* for (i,j) = (1, 1) to (n, n) do; d_ij^(k) = w_ij; od;   */
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            d_ij^(k) = min{d_ij^(k-1) , d_ik^(k-1) + d_kj^(k-1)};
        od;
    od;
od;
 return D^(n);
```

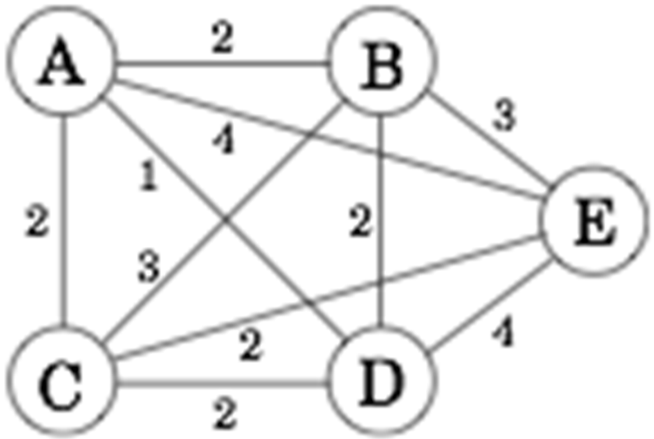
✓ $d^{(k)}_{ij}$: *shortest path from i to j using intermediate edges in {1, 2, …, k}*

Example: Floyd-Warshall Algorithm



Quiz

Run Floyd-Warshall algorithm for the following graph:



Alternate Implementation of the Floyd-Warshall

EXTEND (D , **W**)

▶ D = (d_{ij}) is an n x n matrix

for $i \leftarrow 1$ **to** n **do**

  for $j \leftarrow 1$ **to** n **do**

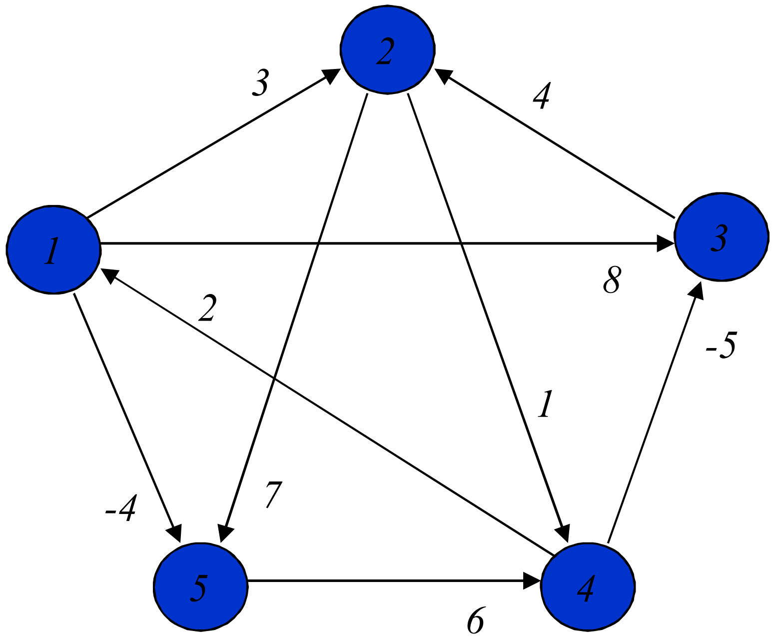
    $d_{ij} \leftarrow \infty$

    for $k \leftarrow 1$ **to** n **do**

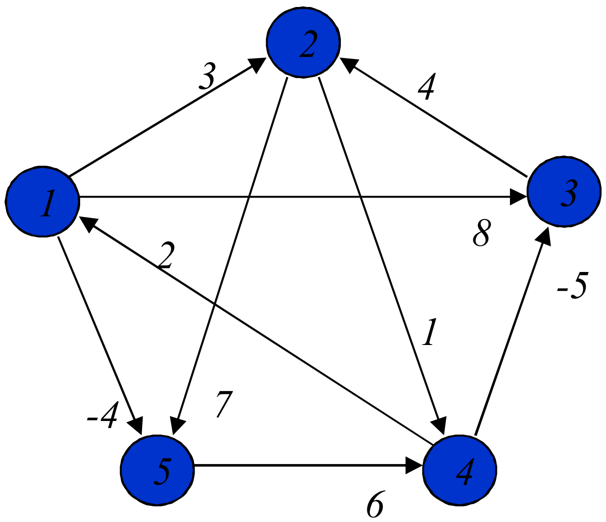
      $d_{ij} \leftarrow \mathbf{min}\{d_{ij}, d_{ik} + \omega_{kj}\}$

return D

Floyd-Warshall: EXTEND



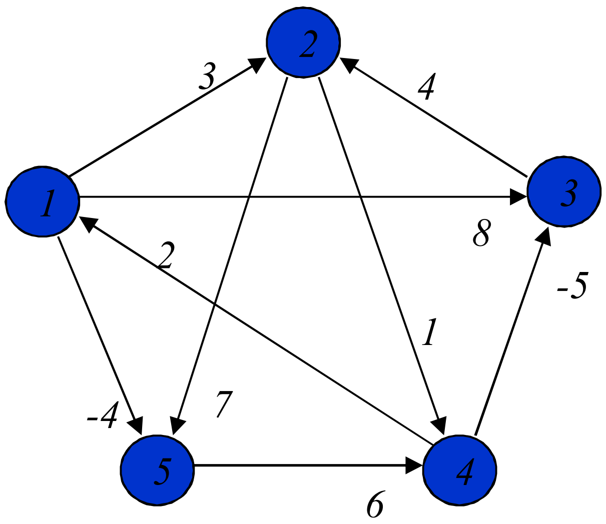
Floyd-Warshall: EXTEND



	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$$D^1 = D^0 W$$

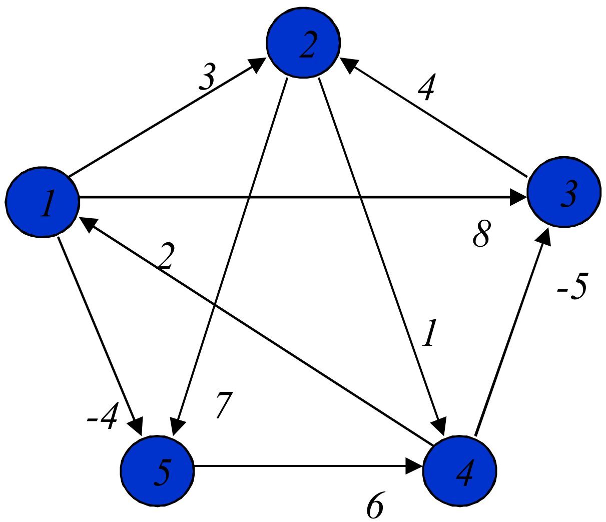
Floyd-Warshall: EXTEND



	1	2	3	4	5
1	0	3	8	2	-4
2	3	0	-4	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	8	∞	1	6	0

$D^2 = D^1 W$

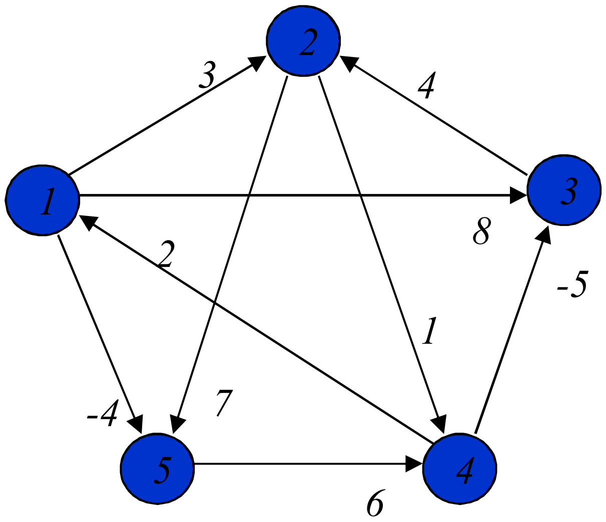
Floyd-Warshall: EXTEND



	1	2	3	4	5
1	0	3	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	11
4	2	-1	-5	0	-2
5	8	5	1	6	0

$D^3 = D^2 W$

Floyd-Warshall: EXTEND



	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$D^4 = D^3 W$

Time and Space Requirements

The running time is obviously $O(n^3)$.

However, in this version, the space requirements are high. One can reduce the space from $O(n^3)$ to $O(n^2)$ by using a single array d.

Conclusion

- Negative weights and negative cycles.
  - If no negative cycles, can find shortest paths via Bellman-Ford.
  - If negative cycles, can find one via Bellman-Ford.
- Dijkstra's algorithm.
  - Nearly linear-time when weights are nonnegative.
- All-pair shortest path.
  - can be solved via Floyd-Warshall
  - Floyd-Warshall can also compute the transitive closure of directed graph.

Next topic: NP-Completeness

END OF LECTURE 17