COMP319 Algorithms 1
Lecture 14
Shortest Paths

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; David Luebke, CS332, Virginia University

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Table of Contents

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Shortest Paths

- 조건
  - 간선 가중치가 있는 유향 그래프
  - 무향 그래프는 각 간선에 대해 양쪽으로 유향 간선이 있는 유향 그래프로 생각할 수 있다
    - 즉, 무향 간선 (u, v)는 유향 간선 (u, v)와 (v, u) 두 개가 존재함을 의미한다고 가정하면 된다

- 두 정점 사이의 최단경로
  - 가능한 경로들 중 간선 가중치 합이 최소인 경로
  - 간선 가중치의 합이 음인 싸이클이 있으면 문제가 정의되지 않는다
    - 무한반복된다

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Shortest Paths

- 단일 시작점 최단경로
  - 주어진 단일 시작점으로부터 각 정점에 이르는 최단경로를 구한다
    - 싸이클이 없는 그래프의 최단경로
  - 벨만-포드(Bellman-Ford) 알고리즘
    - 음의 가중치를 허용하는 최단경로
  - 다익스트라(Dijkstra's) 알고리즘
    - 음의 가중치를 허용하지 않는 최단경로
- 모든 쌍 최단경로
  - 모든 정점 쌍 사이의 최단경로를 모두 구한다
  - 플로이드-워샬(Floyd-Warshall) 알고리즘

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Single-source shorted path

BELLMAN-FORD ALGORITHM

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

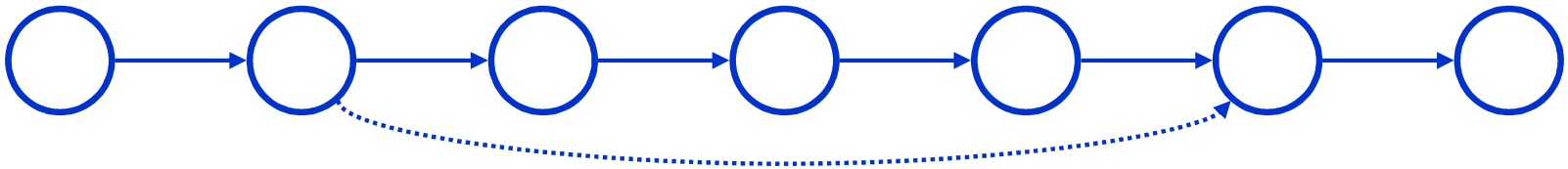
Single-Source Shortest Path

- Problem: given a weighted directed graph G, find the minimum-weight path from a given source vertex *s* to another vertex *v*
  - "Shortest-path" = minimum weight
  - Weight of path is sum of edges
  - E.g., a road map: what is the shortest path from Chapel Hill to Charlottesville?

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Shortest Path Properties

- Again, we have *optimal substructure*: the shortest path consists of shortest subpaths:

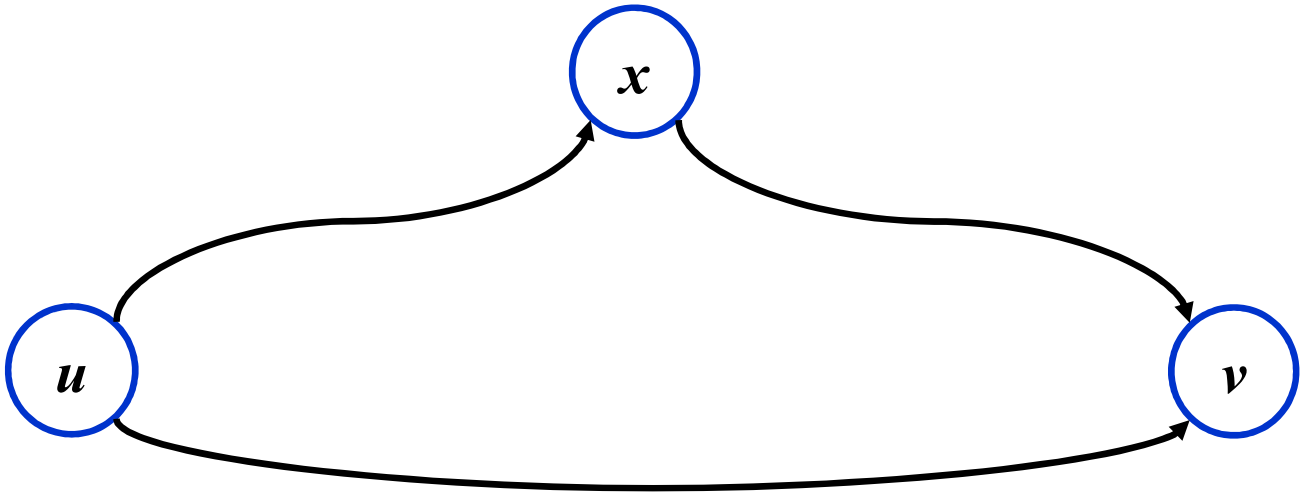


  - Proof: suppose some subpath is not a shortest path
    - There must then exist a shorter subpath
    - Could substitute the shorter subpath for a shorter path
    - But then overall path is not shortest path. Contradiction

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Shortest Path Properties

- Define $\delta(u,v)$ to be the weight of the shortest path from u to v
- Shortest paths satisfy the *triangular inequality*:
  - $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$
- "Proof":

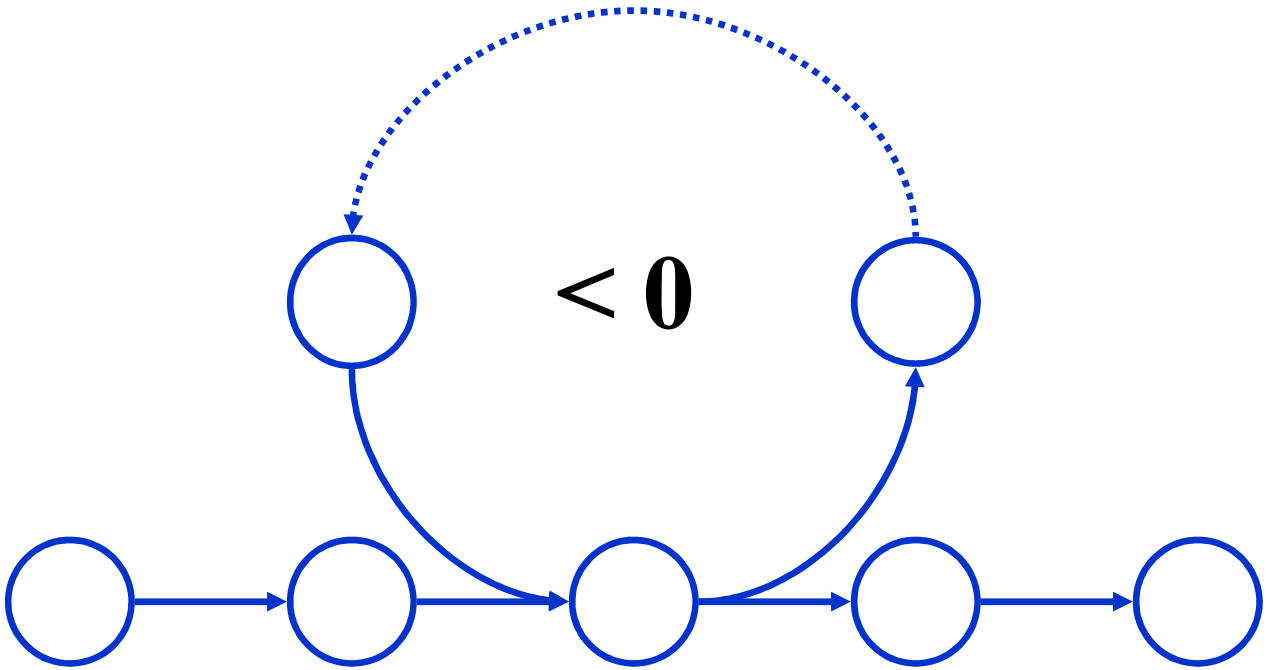


This path is no longer than any other path

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Shortest Path Properties

- In graphs with negative weight cycles, some shortest paths will not exist *(Why?)*:

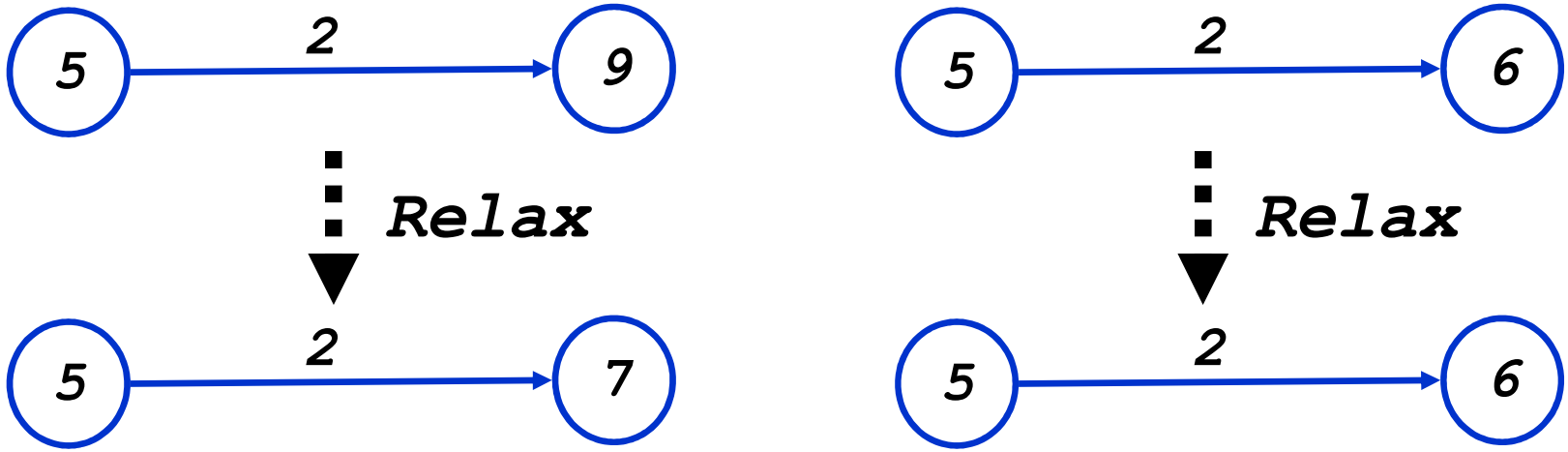


Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Relaxation

- A key technique in shortest path is *relaxation*
  - Idea: for all v, maintain upper bound d[v] on $\delta(s,v)$

```
Relax(u,v,w) {
    if (d[v] > d[u]+w) then d[v]=d[u]+w;
}
```



Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Bellman-Ford by Dynamic Programming

- d_t^k : 중간에 최대 k 개의 간선을 거쳐 정점 r로부터 정점 t에 이르는 최단거리
- 목표: d_t^{n-1}

✓ *재귀적 관계*

$$\begin{cases} d_v^k = \min_{\text{for 모든 간선 } (u,\, v)} \{d_u^{k-1} + w_{u,\,v}\}, & k > 0 \\ d_r^0 = 0 \\ d_t^0 = \infty, \quad t \neq r \end{cases}$$

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Bellman-Ford Algorithm

```
BellmanFord()
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
            Relax(u,v, w(u,v));
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
            return "no solution";
```

h-1 repetition

Initialize d[], which will converge to shortest-path value δ

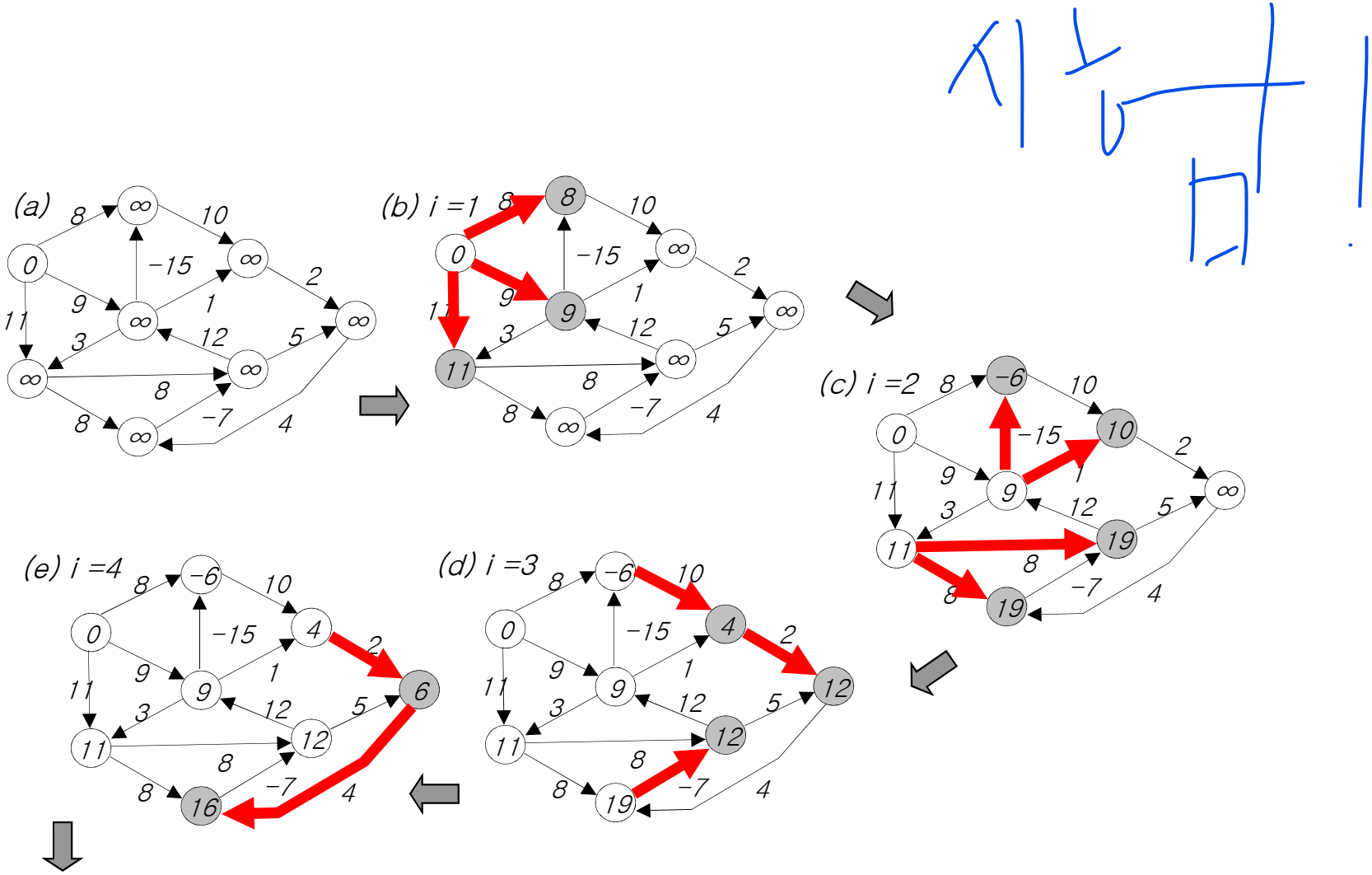
Relaxation: Make |V|-1 passes, relaxing each edge

Test for solution
Under what condition do we get a solution?

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

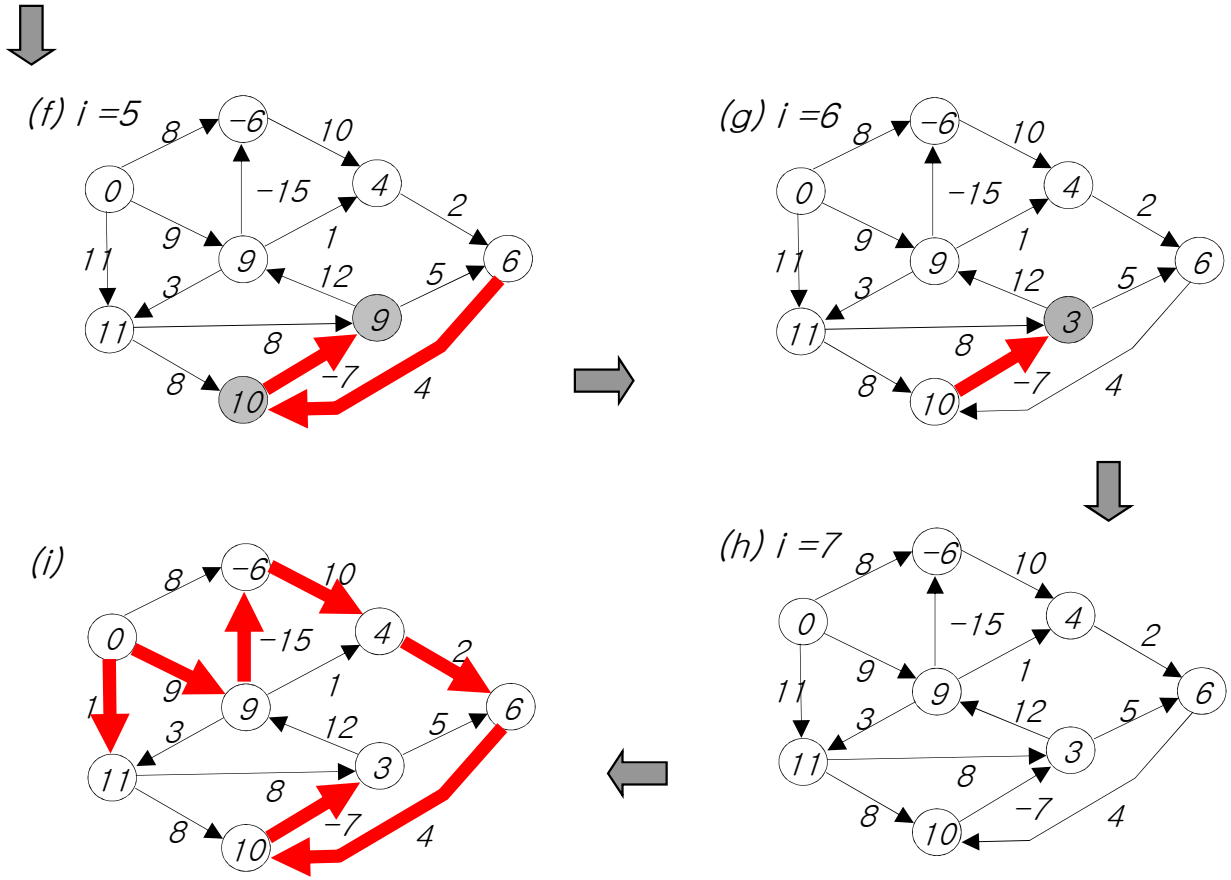
Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Bellman-Ford Example



Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

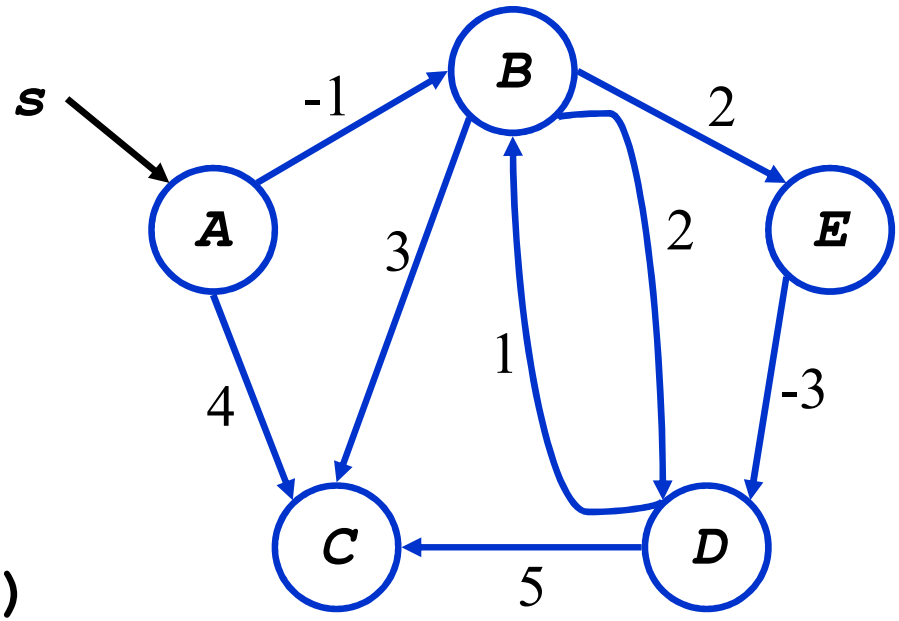
Bellman-Ford Example



Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Bellman-Ford Algorithm

```
BellmanFord()
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
            Relax(u,v, w(u,v));
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
            return "no solution";
```



Ex: work on board

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Bellman-Ford Algorithm

```
BellmanFord()
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
            Relax(u,v, w(u,v));
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
            return "no solution";
```

What will be the running time?

A: O(VE)

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Correctness of Bellman-Ford

- If $G=(V,E)$ contains no negative-weight cycles, then after then after the Bellman-Ford algorithm executes, $d[v] = \delta(s,v)$ for all $v \in V$.
- Proof. Let $v \in V$ be any vertex, and consider a shortest path $\boldsymbol{p}$ from s to v with the minimum number of edges.

$$s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v$$

  - Since $\boldsymbol{p}$ is a shortest path, we have

$$\delta(s,v_i) = \delta(s,v_{i-1}) + w(v_{i-1}, v_i) .$$

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Correctness of Bellman-Ford

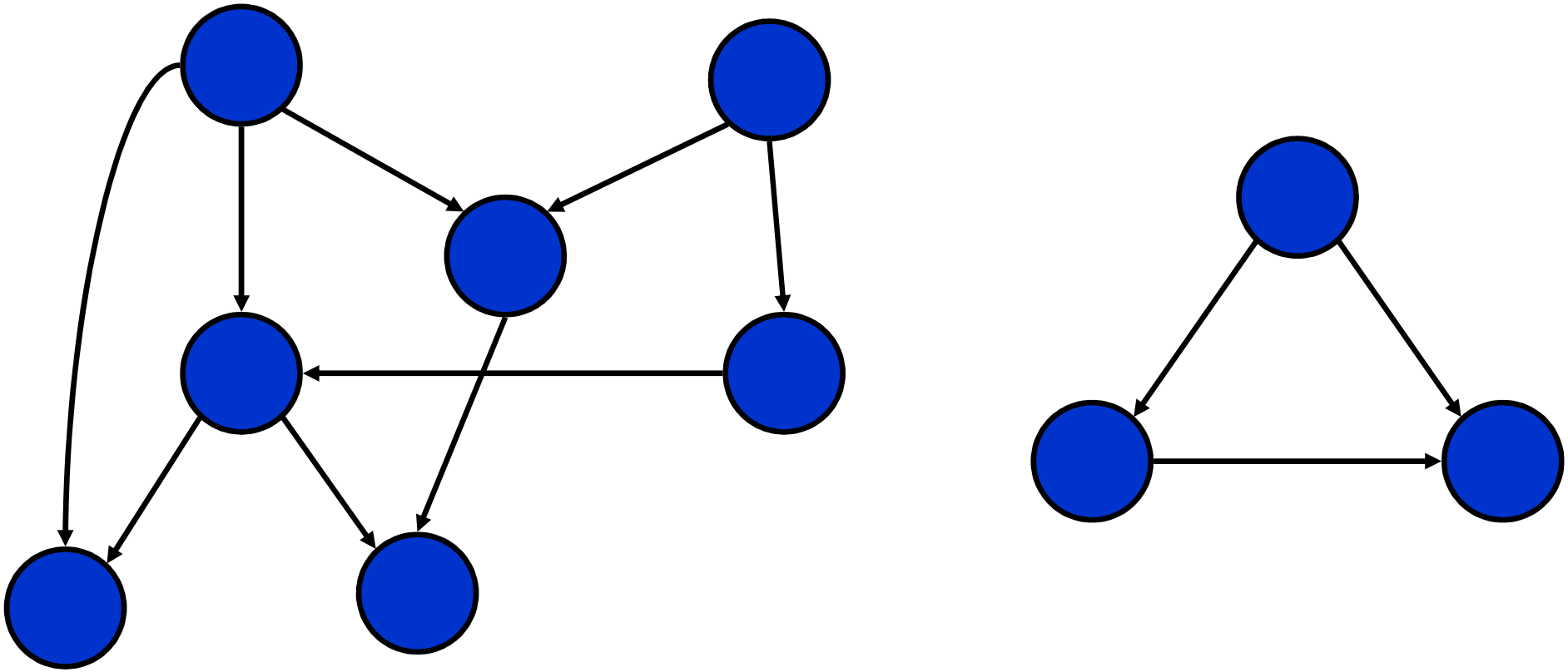
$$s \to v_1 \to v_2 \to v_3 \to v_4 \to v$$

- Initially, $d(v_0) = 0 = \delta(s,v_0)$, and $d[s]$ is unchanged by subsequent relaxations (because $d[v] \geq \delta(s,v)$).
  - After 1 pass through E, we have $d(v_1) = \delta(s,v_1)$.
  - After 2 passes through E, we have $d(v_2) = \delta(s,v_2)$.
  - …
  - After k passes through E, we have $d(v_k) = \delta(s,v_k)$.
- Since G contains no negative-weight cycles, $\boldsymbol{p}$ is a longest simple path that has $\leq |V| - 1$ edges.

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Directed Acyclic Graphs

- A *directed acyclic graph* or *DAG* is a directed graph with no directed cycles:



Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

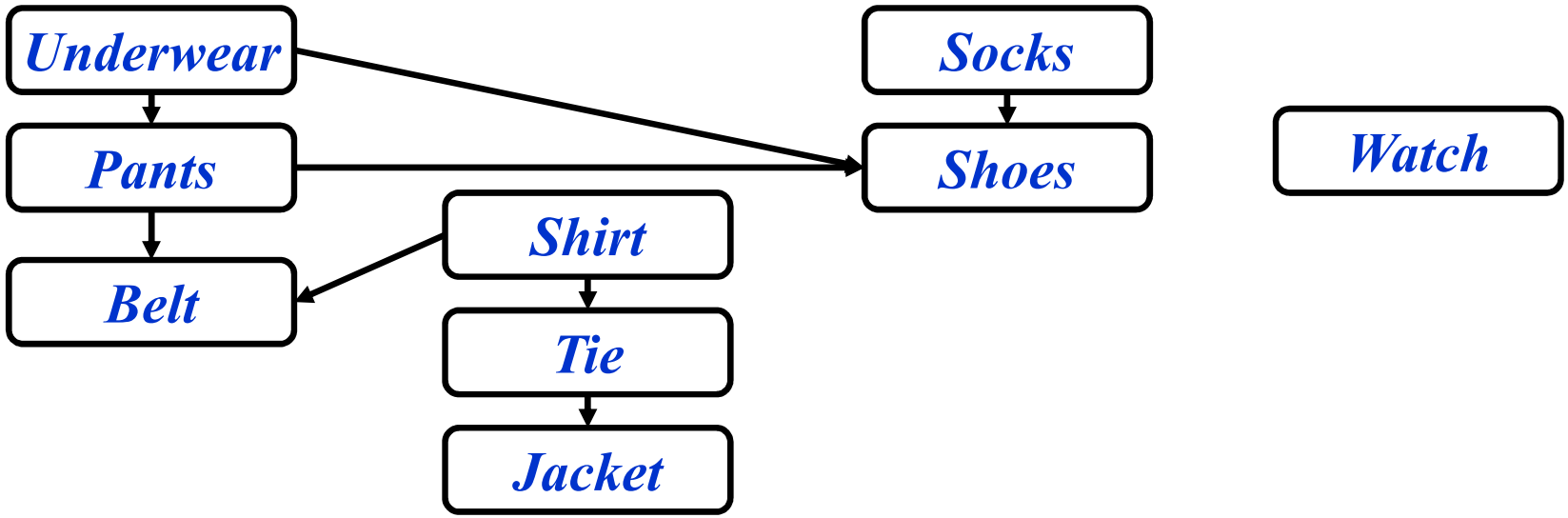
DAG Shortest Paths

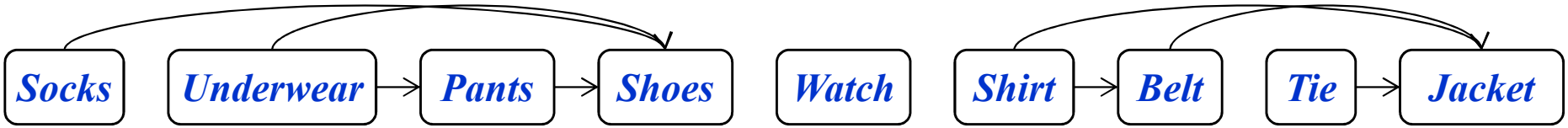
- Problem: finding shortest paths in DAG
  - Bellman-Ford takes O(VE) time.
  - *How can we do better?*
  - Idea: use topological sort
    - If were lucky and processes vertices on each shortest path from left to right, would be done in one pass
    - Every path in a directed acyclic graph is a subsequence of topologically sorted vertex order, so processing vertexes in that order, we will do each path in forward order (will never relax edges out of vertex before doing all edges into vertex).
    - Thus: just one pass. *What will be the running time?*

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Review: Topological Sort

- *Topological sort* of a DAG (directed acyclic graph):
  - Linear ordering of all vertices in graph G such that vertex *u* comes before vertex *v* if edge $(u, v) \in G$
- Real-world example: getting dressed





Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

DAG Shortest Paths

```
DAG-SHORTEST-PATHS(V,E,w,s)
     topologically sort the vertices
     INIT-SINGLE-SOURCE(V,s)
     for each vertex u,
          take in topologically sorted order
          do for each vertex v <- Adj[u]
               do RELAX(u,v,w)
```

Time complexity: O(V+E)

Source: https://zoomkoding.github.io/algorithm/2019/05/09/shortest-paths-1.html

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Review: Bellman-Ford

- Running time: O(VE)
  - Not so good for large dense graphs
  - But a very practical algorithm in many ways
- Note that order in which edges are processed affects how quickly it converges (show example)
  - Using topological sort: O(V+E)

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Single-Source Shortest Path without negative edge weight

DIJKSTRA'S ALGORITHM

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

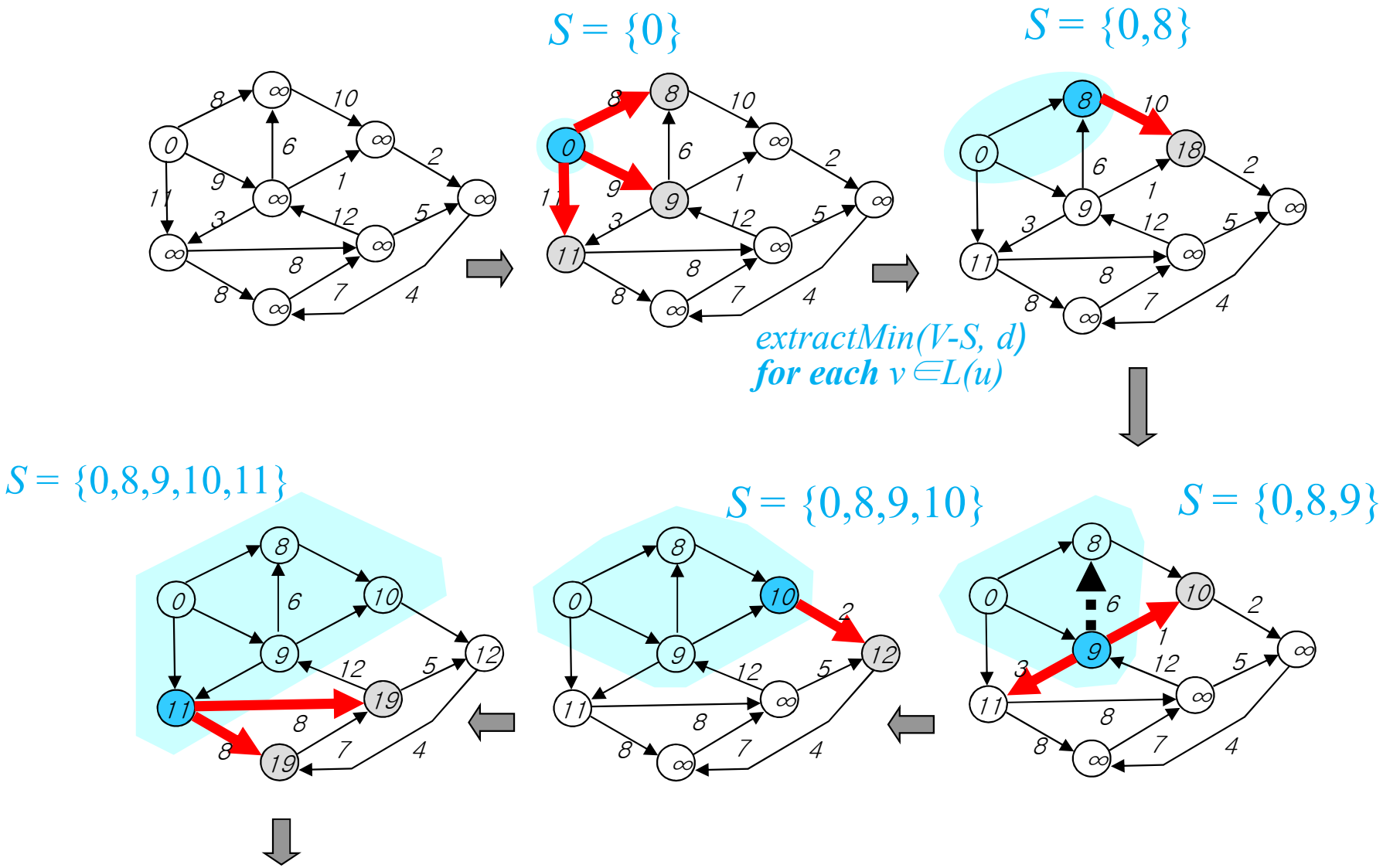
Dijkstra Algorithm

```
Dijkstra(G, r)
▷ G=(V, E): 주어진 그래프
▷ r: 시작으로 삼을 정점
{                                                    모든 간선의 가중치는 음이 아니어야 함
     S ← Φ ;                    ▷ S : 정점 집합
     for each u∈V
          d_u ← ∞ ;
     d_r ← 0 ;
     while (S≠V){               ▷ n회 순환된다
          u ← extractMin(V-S, d) ;
          S ← S ∪ {u};
          for each v∈L(u) ▷ L(u) : u로부터 연결된 정점들의 집합
               if (v∈V-S and d_v<d_u+w_u,v) then d_v ← d_u+w_u,v ;
     }                          이완(relaxation)
}

extractMin(Q, d)                                     ✓수행시간: O(|E|log|V|)
{                                                                  ↖
     집합 Q에서 d값이 가장 작은 정점 u를 리턴한다 ;                    힙 이용
}
```

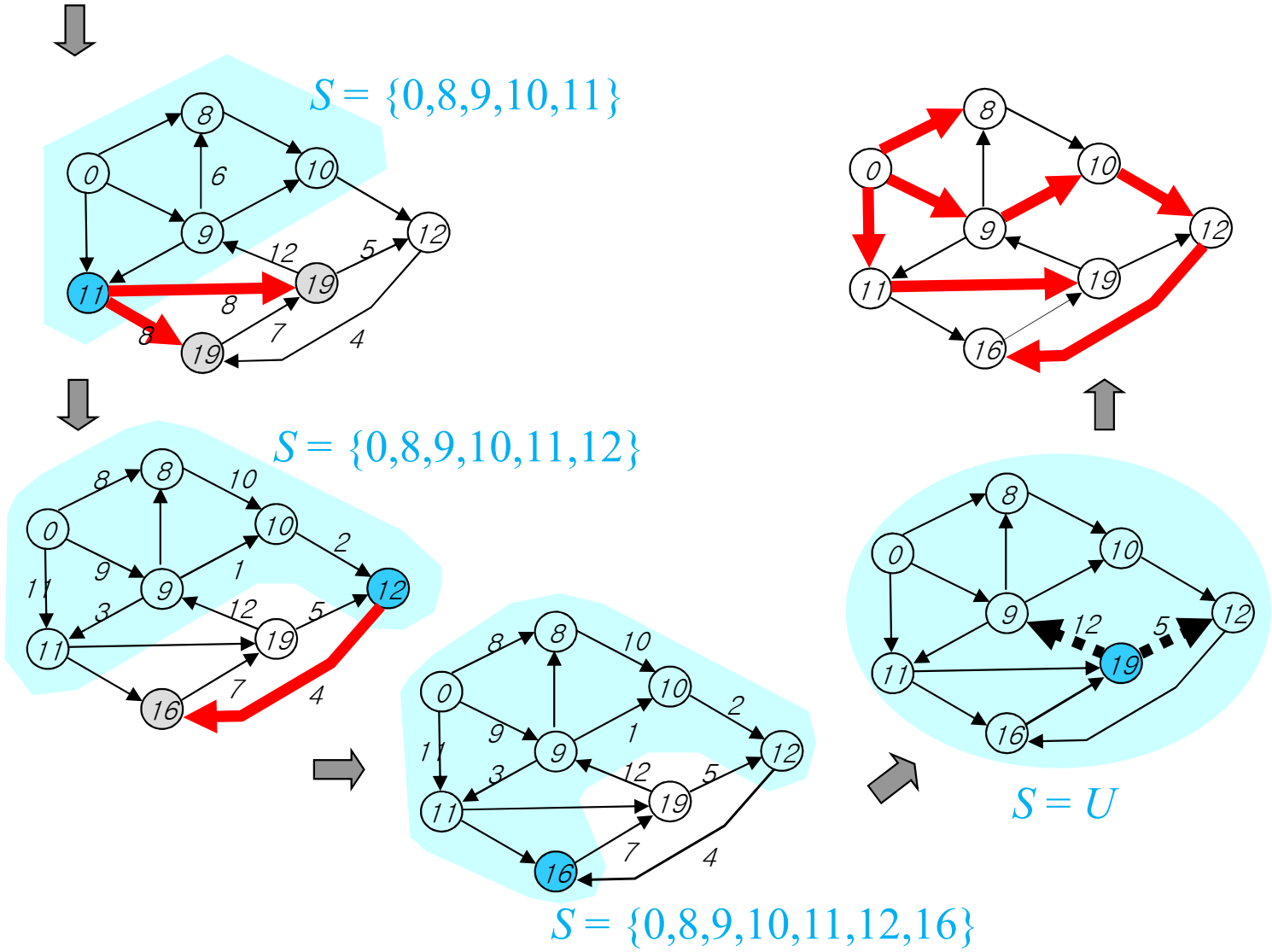
Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Dijkstra Algorithm의 작동 예



Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

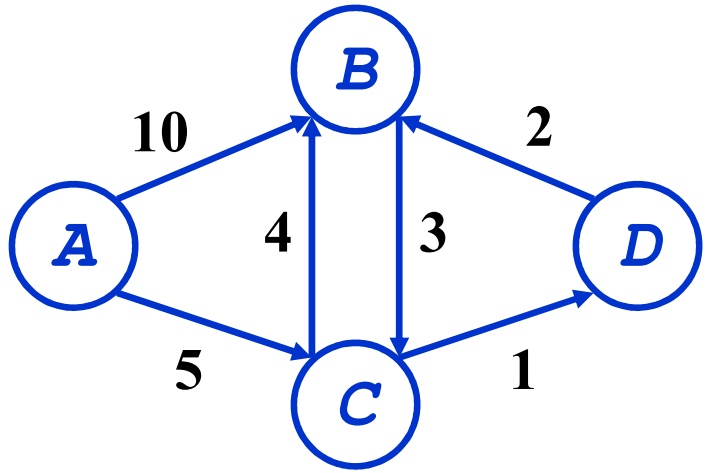
Dijkstra Algorithm의 작동 예



Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Dijkstra's Algorithm

```
Dijkstra(G)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0; S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ u->Adj[]
            if (d[v] > d[u]+w(u,v))
                d[v] = d[u]+w(u,v);
```



Ex: run the algorithm

Relaxation Step

Note: this is really a call to Q->DecreaseKey()

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Dijkstra's Algorithm

```
Dijkstra(G)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0; S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ u->Adj[]
            if (d[v] > d[u]+w(u,v))
                d[v] = d[u]+w(u,v);
```

How many times is ExtractMin() called?

How many times is DecraseKey() called?

What will be the total running time?

A: O(E lg V) using binary heap for Q

Can achieve O(V lg V + E) with Fibonacci heaps

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

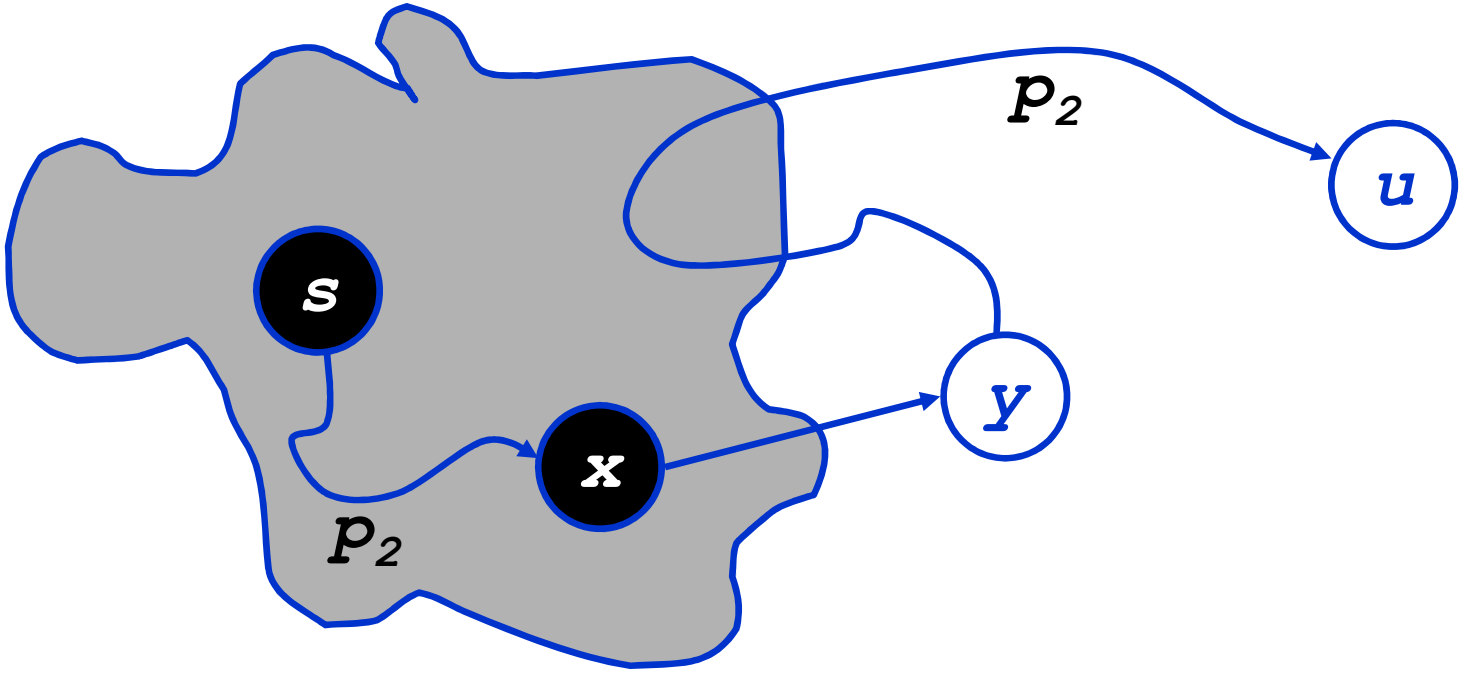
Dijkstra's Algorithm

```
Dijkstra(G)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0; S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ u->Adj[]
            if (d[v] > d[u]+w(u,v))
                d[v] = d[u]+w(u,v);
```

Correctness: we must show that when u is removed from Q, it has already converged

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

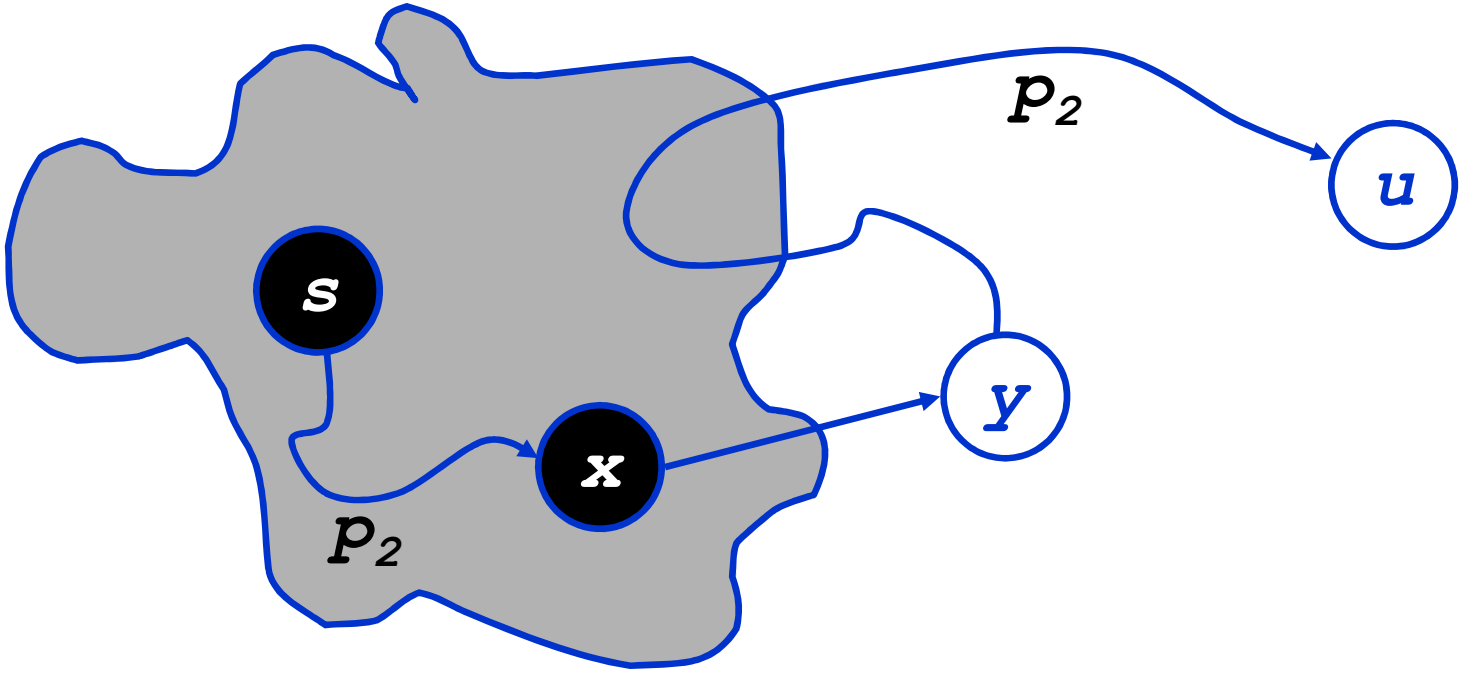
Correctness Of Dijkstra's Algorithm



- 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 $\Rightarrow d[y] = \delta(s,y)$
  - Because d[x] is set correctly for y's predecessor x $\in$ S on the shortest path, and
  - When we put x into S, we relaxed (x,y), giving d[y] the correct value

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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 credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

문병로, 쉽게 배우는 알고리즘

Cevdet Aykanat and Mustafa Ozdal, CS473, Bilkent Univ

Andreas Klappenecker

THE FLOYD-WARSHALL ALGORITHM

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

All Pairs Shortest Paths (APSP)

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

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

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.

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Adjacency Matrix Representation of Graphs

- $n \times n$ matrix $W = (\omega_{ij})$ 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 credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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}

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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

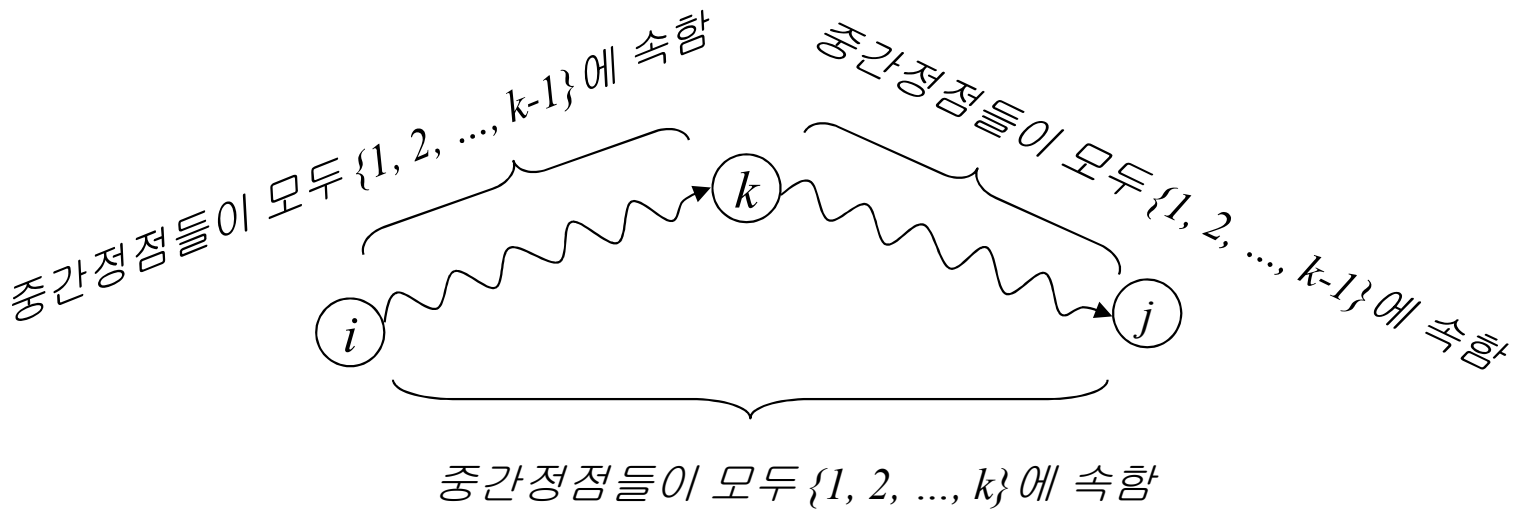


Figure 2. Path *p* is a shortest path from vertex *i* to vertex *j*, and *k* is the highest-numbered intermediate vertex of *p*. Path *p*1, the portion of path *p* from vertex *i* to vertex *k*, has all intermediate vertices in the set {1, 2, …, *k*-1}. The same holds for path *p*2 from vertex *k* to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

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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 credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Recursive Formulation

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

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

If k>0, then

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

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

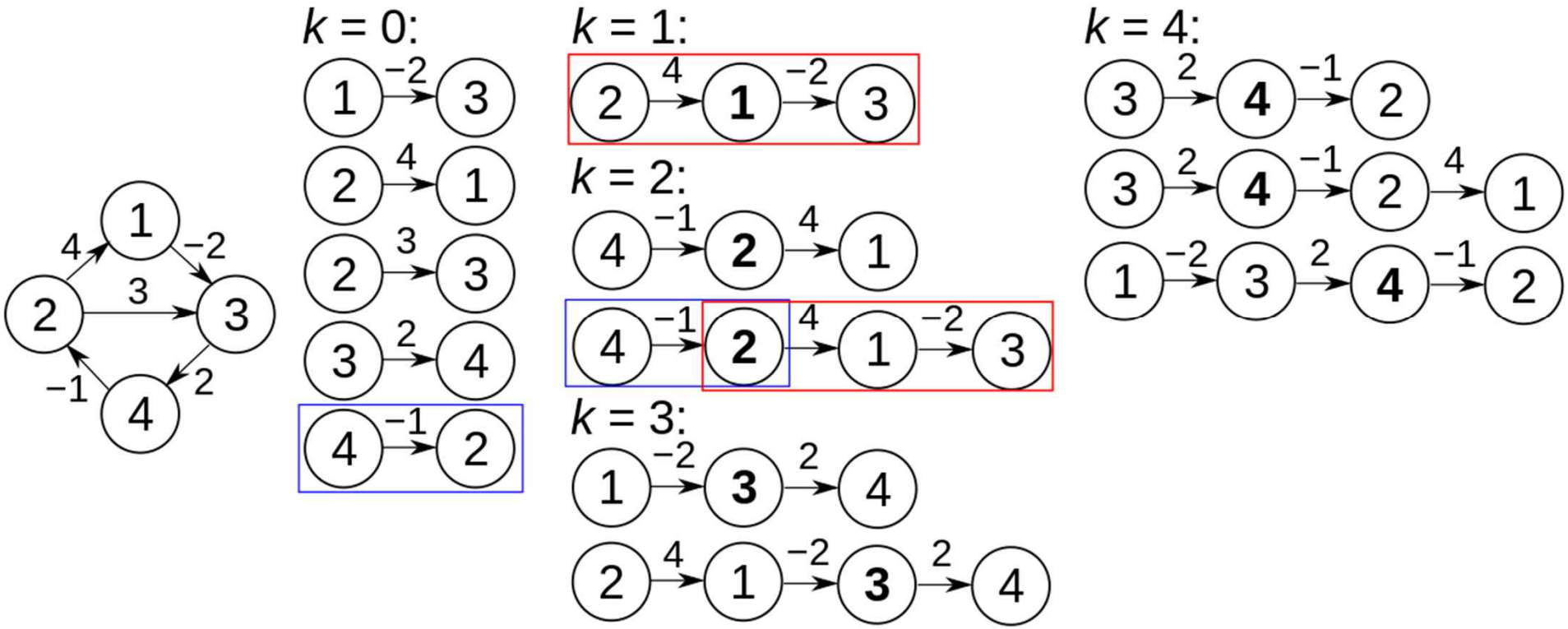
The Floyd-Warshall Algorithm

```
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; dij(k) = wij; od;   */
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            dij(k) = min{dij(k-1) , dik(k-1) + dkj(k-1)};
        od;
    od;
od;
 return D(n);
```

✓ $d^{(k)}_{ij}$: 중간 정점으로 정점 집합 $\{1, 2, \ldots, k\}$ 만을 사용하여 정점 i에서 정점 j에 이르는 최단경로

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Example: Floyd-Warshall Algorithm



Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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**

$d_{ij} \leftarrow \infty$

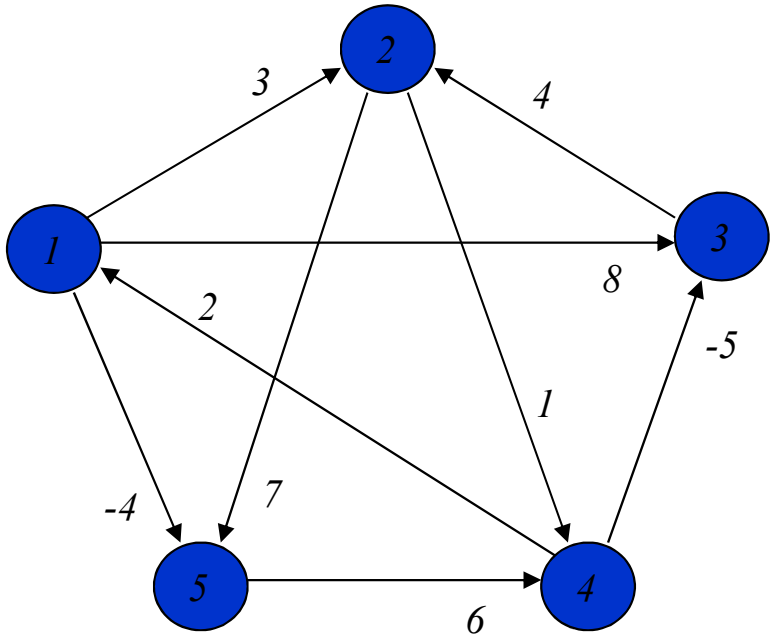
for $k \leftarrow 1$ **to** n **do**

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

return D

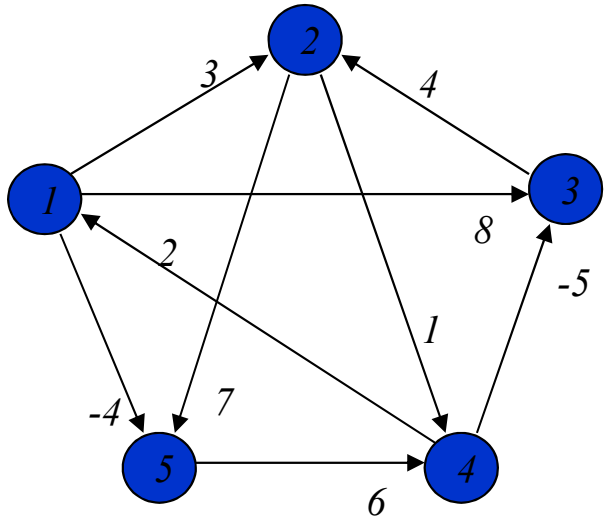
Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Floyd-Warshall: EXTEND



Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Floyd-Warshall: EXTEND

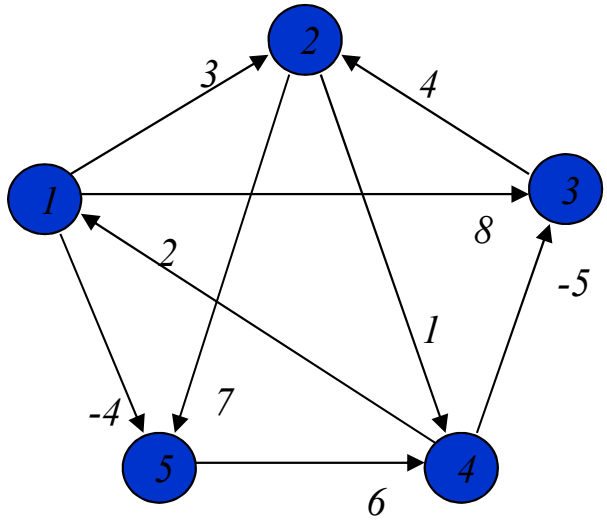


	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$D^1 = D^0 W$

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Floyd-Warshall: EXTEND

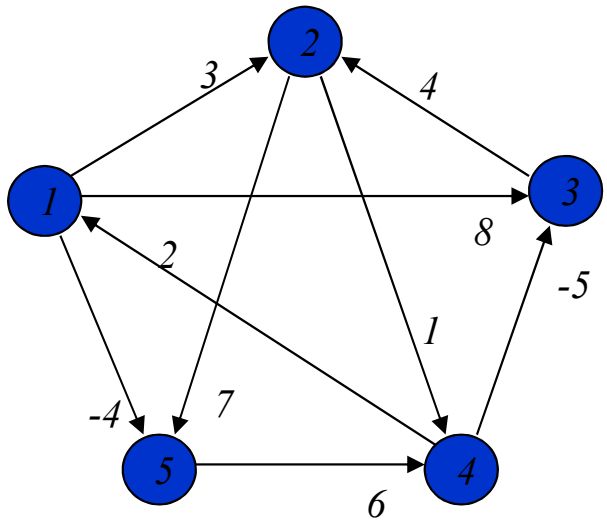


	1	2	3	4	5
1	0	3	8	2	-4
2	3	0	-4	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	8	∞	1	6	0

$D^2 = D^1 W$

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Floyd-Warshall: EXTEND

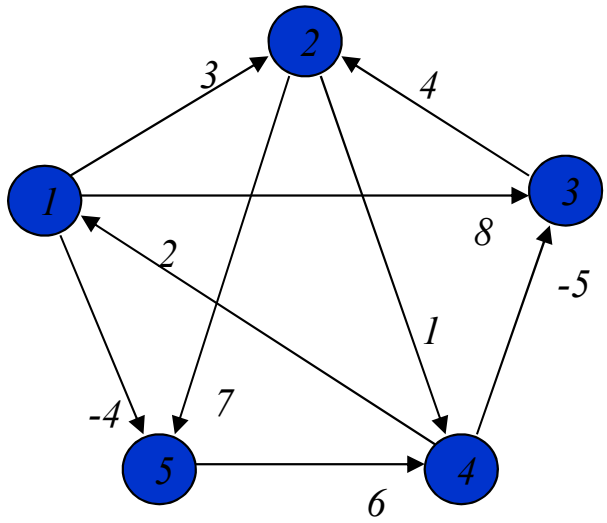


	1	2	3	4	5
1	0	3	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	11
4	2	-1	-5	0	-2
5	8	5	1	6	0

$D^3 = D^2 W$

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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

$D^4 = D^3 W$

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

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.

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC

Next topic: Minimum Spanning Trees

END OF LECTURE 14

Slide credit: 홍석원, 명지대학교; 김한준, 서울시립대학교; J. Lillis, UIC