Slido: #ADA2022

CSIE 2136 Algorithm Design and Analysis, Fall 2022

National Taiwan University 國立臺灣大學

Graph Algorithms - III

Hsu-Chun Hsiao

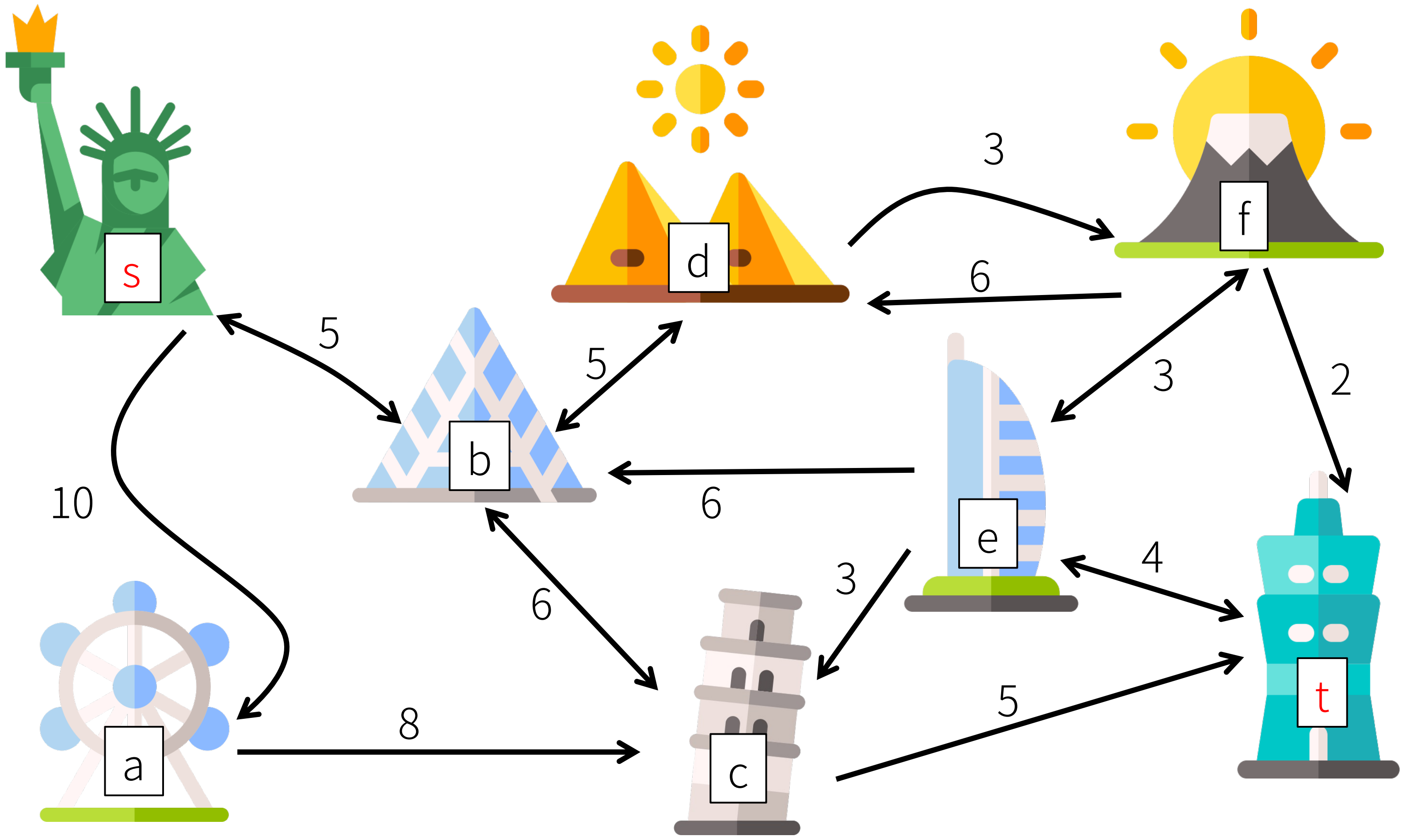
Today's Agenda

- Shortest paths: terminology and properties
  - Edge relaxation
  - Shortest-paths properties
- Single-source shortest paths [Ch. 24]
  - Bellman-Ford algorithm
  - Dijkstra algorithm
  - Single-source shortest paths in DAG
- Appendix: All-pairs shortest paths [Ch. 25]
  - Floyd-Warshall algorithm
  - Johnson's algorithm

Shortest Paths: Terminology and Properties

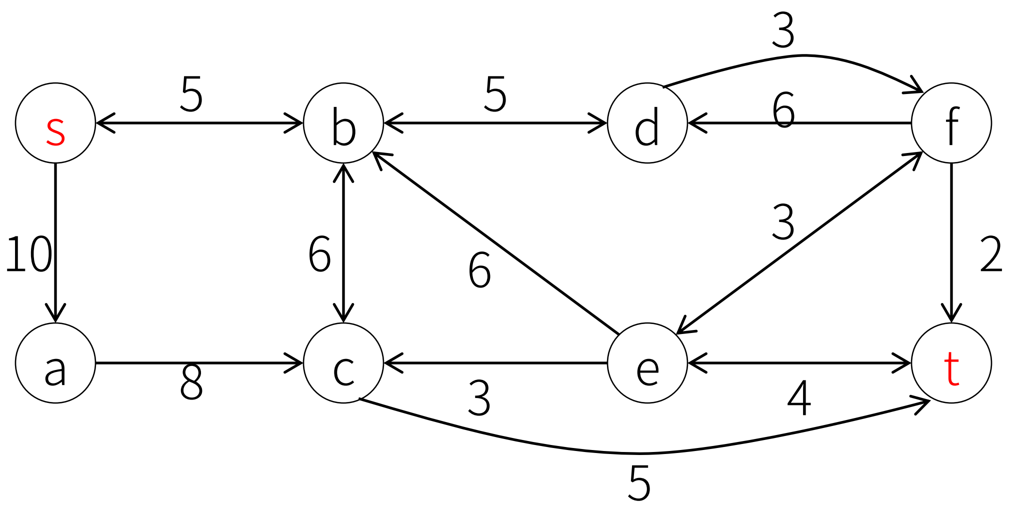
Textbook Chapter 24

Example



Definitions

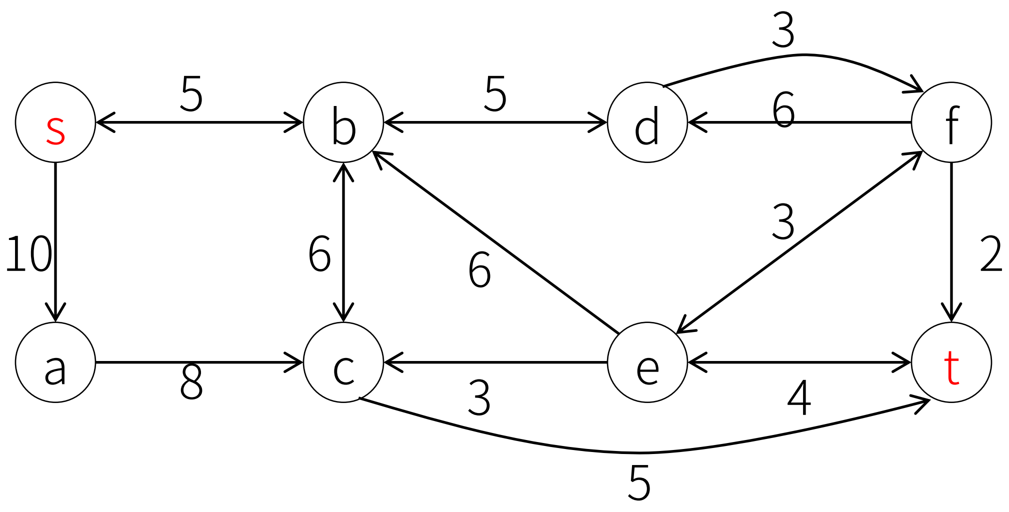
- Given a weighted, directed graph $G = (V, E)$
- Given a weight function w mapping an edge to a weight
  - Note that weights are arbitrary numbers, not necessarily distances
  - Weight function needs not satisfy triangle inequality (e.g., airfares)
- Weight of path $p = w(p)$ = sum of weights of edges on p
  - Sometimes we also call it cost



The weight of path s->a->c->t is 23

Definitions

- Shortest-path weight $\delta(s, t)$ = minimum weight of path from s to t
- A shortest path from s to t = any path with weight $\delta(s, t)$

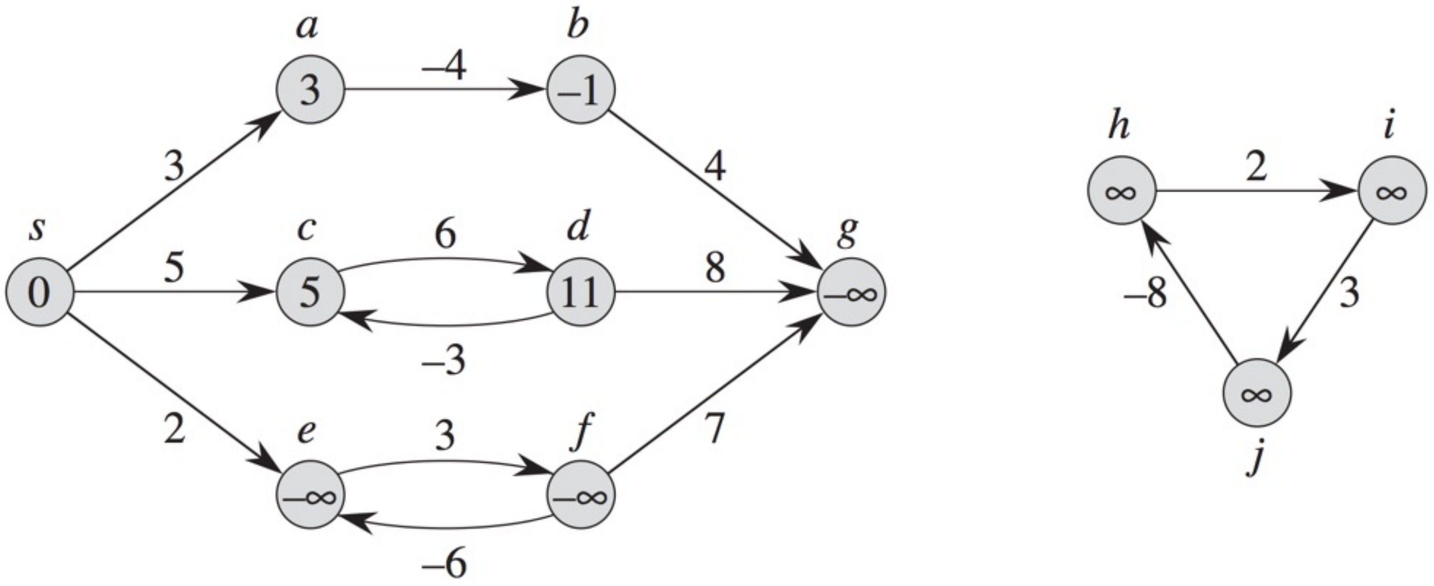


$\delta(s, t)$ = ?

Shortest path from s to t = ?

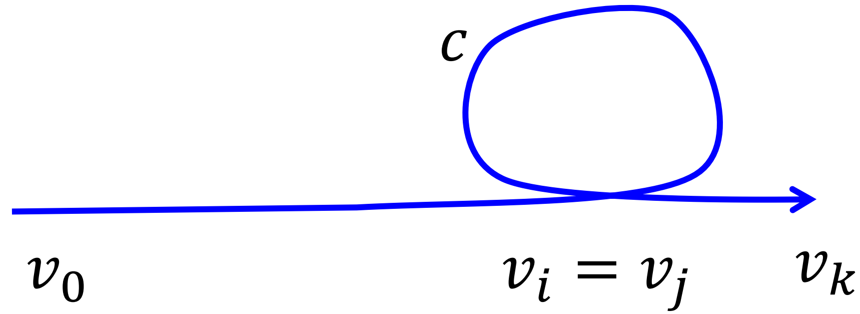
Q: Can a shortest path contain a negative-weight edge?

Q: Can a shortest path contain a negative-weight cycle?



Q: Can a shortest path contain a positive-weight cycle?

Q: Can a shortest path contain a zero-weight cycle?



$p = < v_0, v_1, \ldots, v_k >$

$c = < v_i, v_{i+1}, \ldots, v_j >$

Let $p' = < v_0, v_1, \ldots v_i, v_{j+1}, v_{j+2}, \ldots, v_k >$

$w(p') \leq w(p)$ if $w(c) \geq 0$

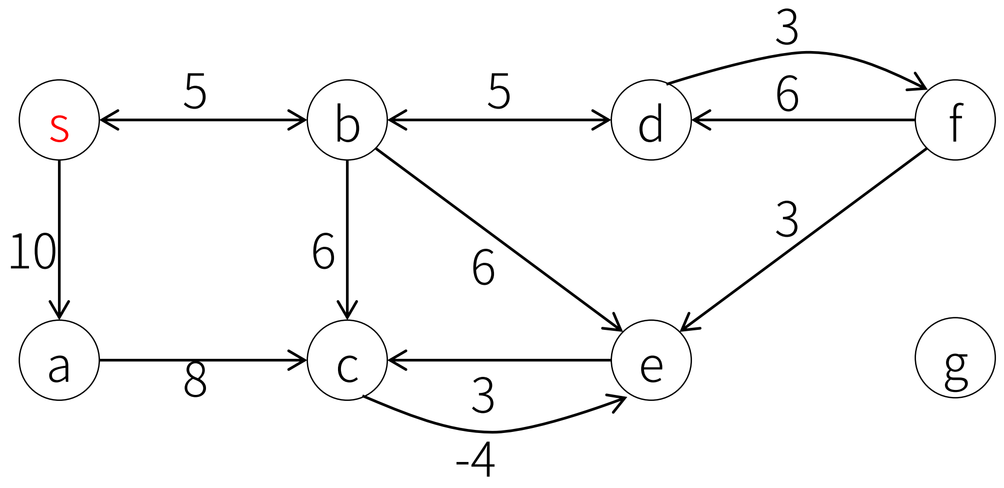
Definitions

We safely assume shortest paths have no cycles

- Define $\delta(u, v) = \infty$ if v is unreachable from u
- Define $\delta(u, v) = -\infty$ if there exists a negative cycle on a path from u to v

True/False: A shortest path has at most $|V| - 1$ edges.

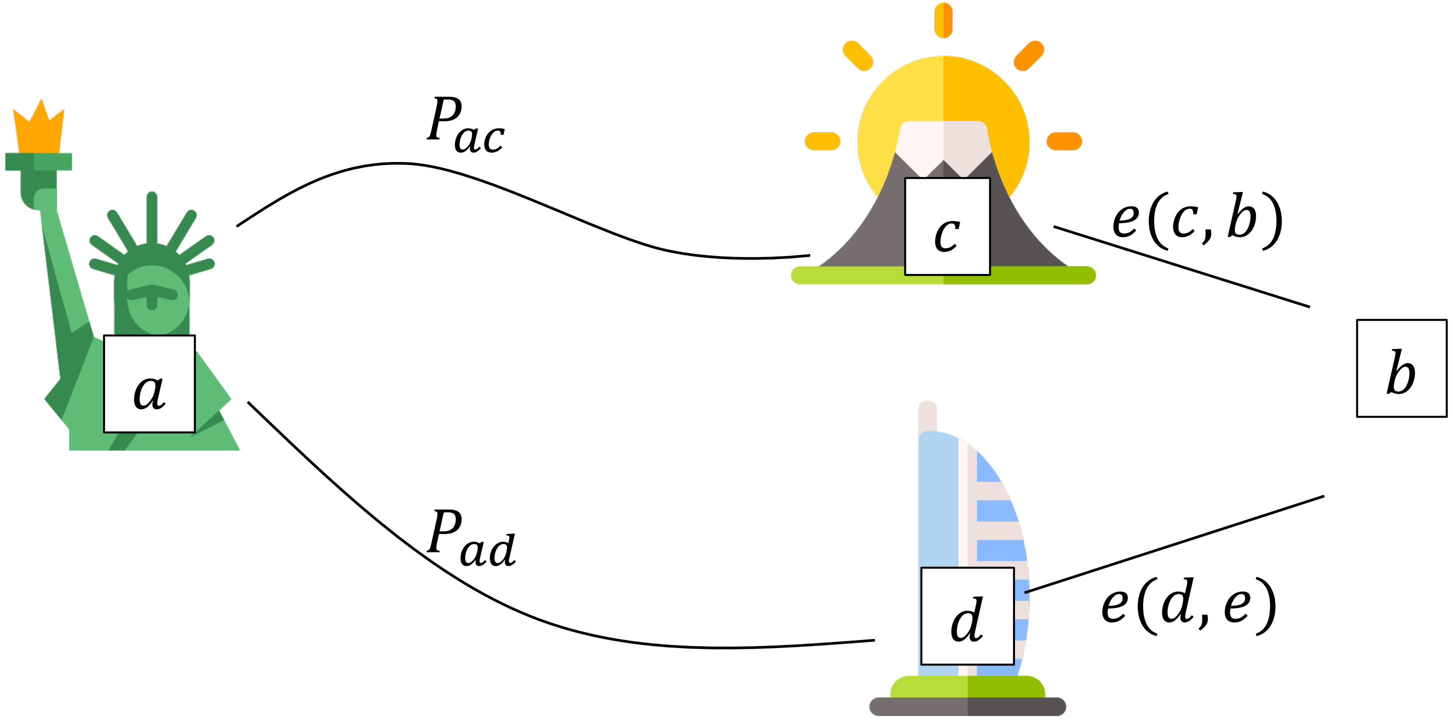
Practice



Destination v	Shortest path from s to v	Shortest path weight
a	s a	10
b		
c	NIL	-∞
d		
e		
f	s b d f	13
g	NIL	∞

Shortest paths and optimal substructure

Shortest-path problem (最短路徑問題) has optimal substructure

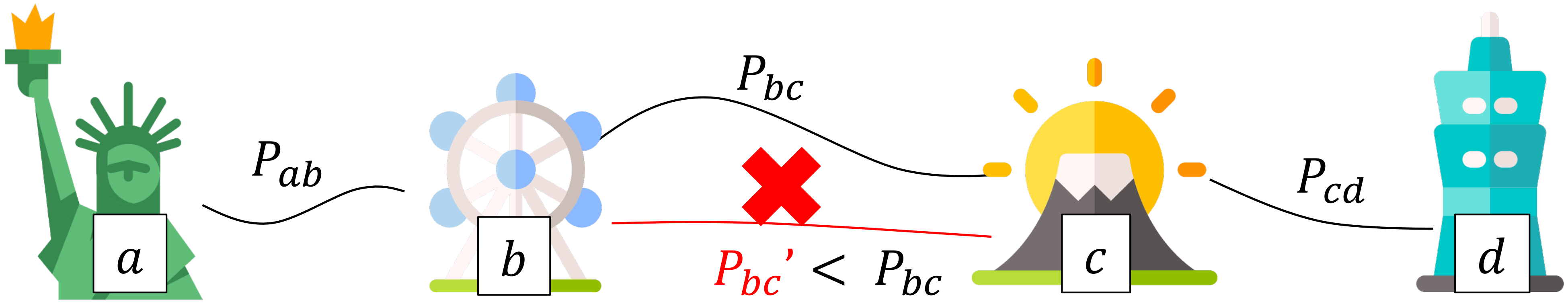


$$\delta(a,b) = \min(\delta(a,c) + w(c,b), \delta(a,d) + w(d,b))$$

Subpaths of shortest paths are shortest paths (Lemma 24.1)

Given a weighted, directed graph $G = (V, E)$ with weight function $w: E \to \mathbb{R}$, let $p =< v_0, v_1, \dots, v_k >$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} =< v_i, v_{i+1}, \dots, v_j >$ be the subpath of p from vertex i to vertex j. Then, p_{ij} is a shortest path from i to j.

Proof by contradiction



Path $P_{ab} + P_{ac} + P_{cd}$ is a shortest path between a and d
⇒Then P_{bc} must be a shortest path between b and c

Single-source Shortest Paths

Textbook Chapter 24

Single-source shortest-path algorithms

- Given a graph $G = (V, E)$ and a source vertex s in V, find the minimum cost paths from s to every vertex in V
- Bellman-Ford algorithm
  - Dynamic programming
  - General case, edge weights may be negative
- Dijkstra algorithm
  - Greedy
  - Requiring that all edge weights are nonnegative
- Single-source shortest paths in DAG
  - Requiring a DAG
- All on a weighted, directed graph

A very important technique: Relaxation

A common workflow for single-source shortest-path algorithms:

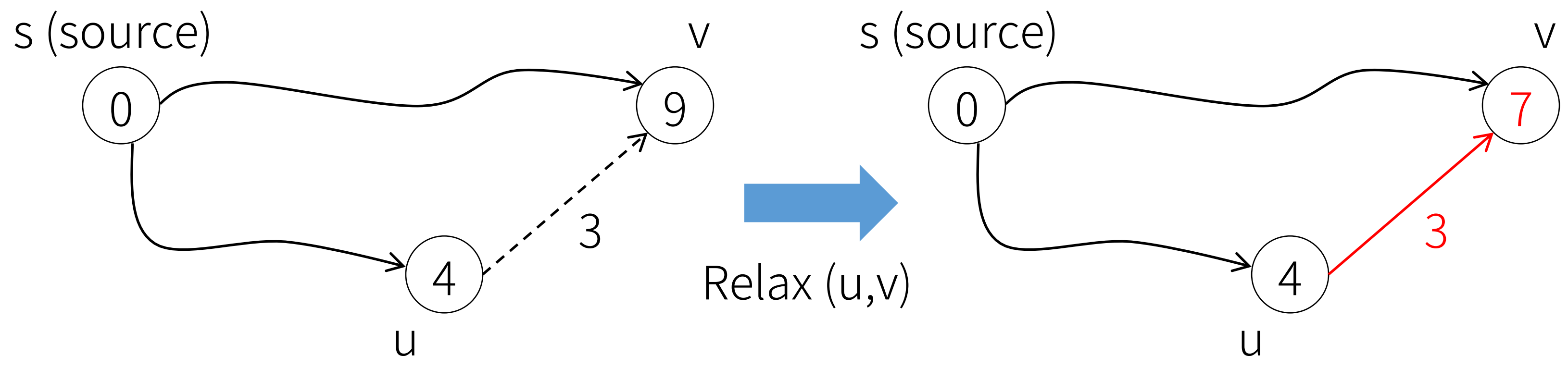
```
INITIALIZE-SINGLE-SOURCE(G,s)
    for v in G.V
        v.d = ∞ //estimate
        v.π = NIL //predecessor
    s.d = 0
```

→ Take a **sequence of** relaxation **steps** to update `v.d` and `v.π`

→ Output `v.d` and reconstruct shortest-paths from `v.π`

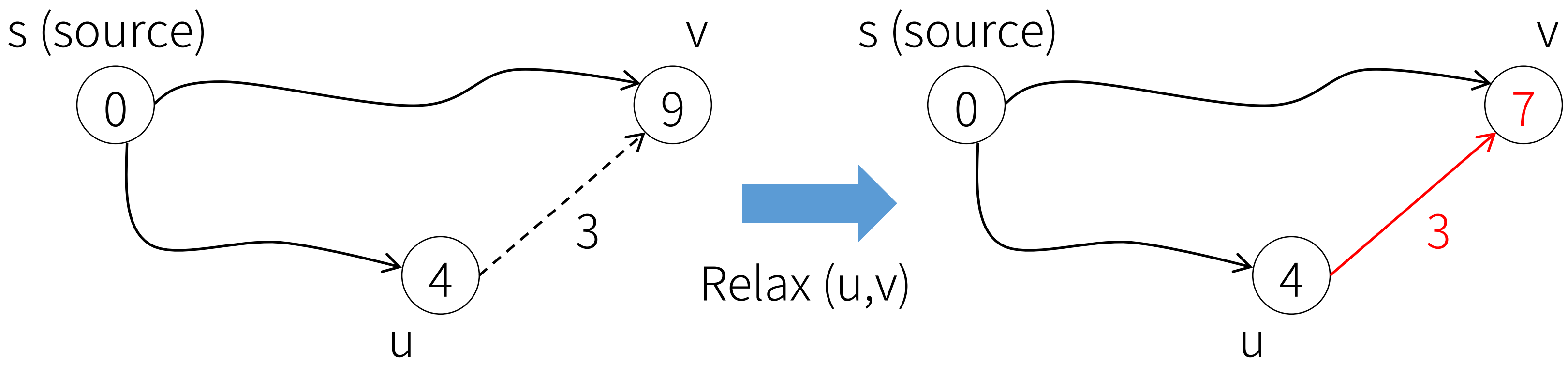
A very important technique: Relaxation

- The process of relaxing an edge (u, v)
  = testing whether the shortest path weight of v found so far can be reduced by traveling over u
- 試試看經過 u 會不會比較好（更短的 $s \rightsquigarrow v$ 路徑）



A very important technique: Relaxation

- The process of relaxing an edge (u, v)
  = testing whether the shortest path weight of v found so far can be reduced by traveling over u




```
RELAX(u, v)
    if v.d > u.d + w(u, v)
        v.d = u.d + w(u, v)
        v.π = u
```

$v.d$ = shortest-path estimate
- An upper bound on $\delta(s, v)$ (Lemma 24.11)
- $v.d$ never increases during relaxation

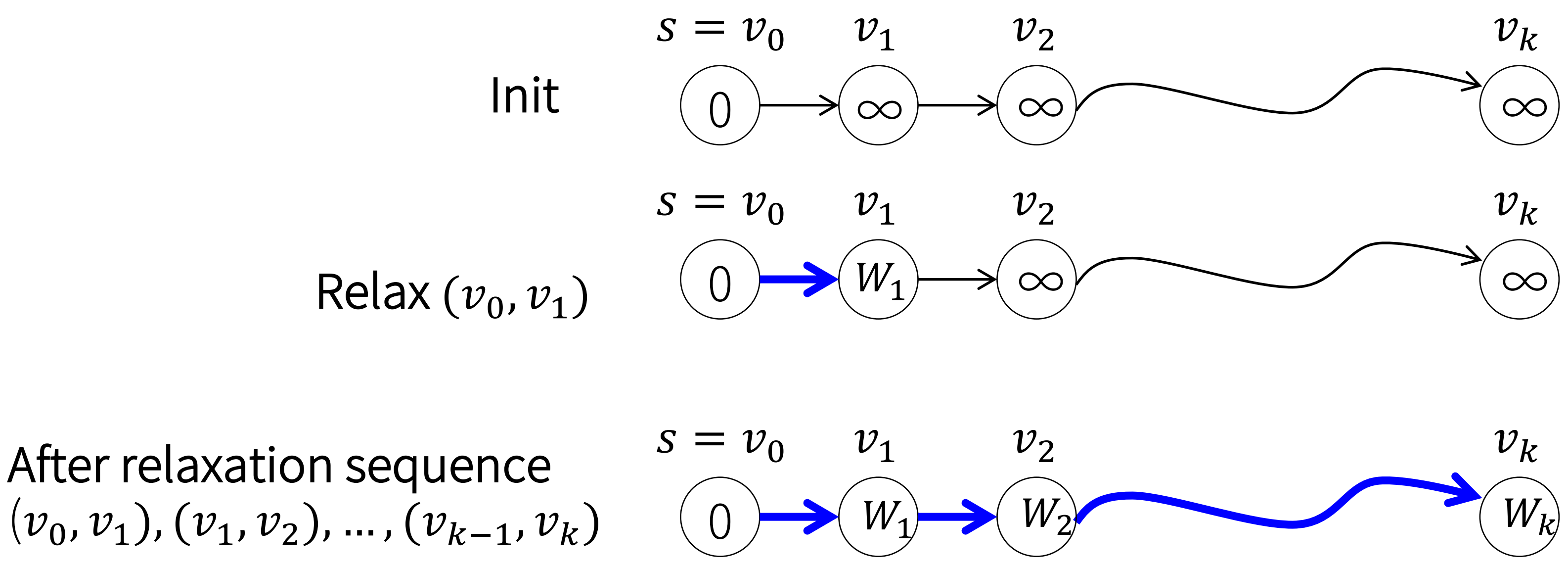
$v.\pi$ = predecessor attribute

Path-relaxation property (Lemma 24.15)

- Let $p = < v_0, v_1, \dots, v_k >$ be a shortest path from $s = v_0$ to v_k
- $v_k.d \ = \ \delta(s, v_k)$ after any relaxation sequence that contains a subsequence $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$

Proof by induction on relaxing the ith edge (v_{i-1}, v_i) on p

Let $W_i = \sum_1^i w(v_{i-1}, v_i)$. W_i is the shortest path weight $\delta(s, v_i)$ because of optimal substructure

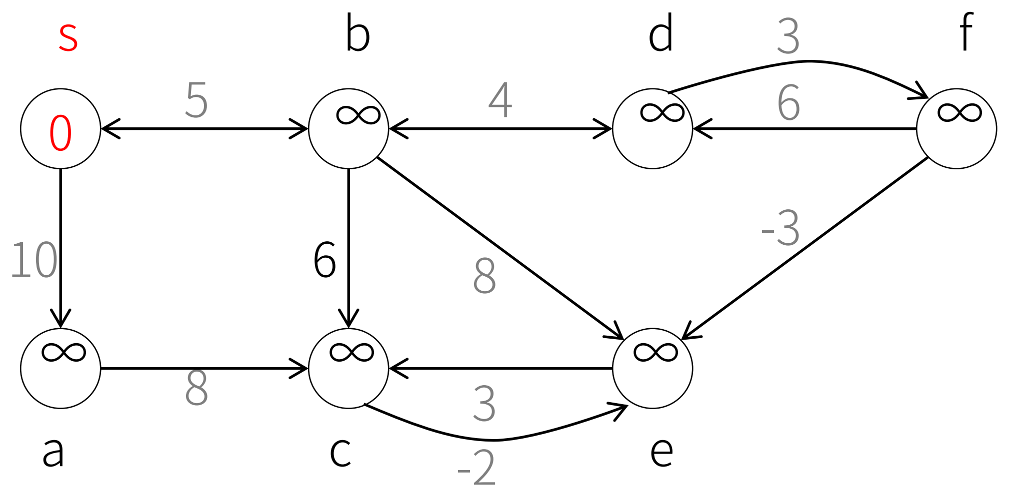


Note: 此性質對於任何包含這個最短路徑邊的 relaxation sequence 都成立, e.g.,

$(v_0, v_1), (a, b), (d, c), (v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k), \dots$

Initial state

Suppose we know $\delta(s, e) = 9$, and a shortest path from s to $e = < s, b, d, f, e >$



Q: After relaxing $(s, b), (b, d), (d, f), (f, e)$ in order, what's the value of $e.d$?

9, according to the path-relaxation property

Q: Will the value of $e.d$ remain the same after relaxing the edges in a different order, such as (s, b), (d, f), (b, d), (f, e)?

Not necessarily

Q: How about relaxing (s, b), $(b, e), (s, a)$, $(b, d), (d, f)$, (e, c), (f, e)?

9, according to the path-relaxation property

Bellman-Ford algorithm

Textbook Chapter 24.1

The DP view

- Bellman-Ford algorithm is based on dynamic programming
  - What are the subproblems?
  - Does it have optimal substructure?
  - How to recursively define the value of an optimal solution?

- Idea: using the shortest paths of at most $k - 1$ edges to construct the shortest paths of at most k edges

The DP view

- Let $\ell_{sv}^{(k)}$ be the shortest path value from s to v using at most k edges
  - Subproblems: given s, $\ell_{sv}^{(k)}$ for all v, k
  - Optimal substructure: by Lemma 24.1
- Base case: $\ell_{ss}^{(0)} = 0$; $\ell_{sv}^{(0)} = \infty$ when $s \neq v$
- Recurrence relation can be formulated as

$$\ell_{sv}^{(k)} = \min_{u \in \mathrm{V}} \left\{ \ell_{su}^{(k-1)} + w_{uv} \right\}$$

- Optimal values: $\ell_{sv}^{(|V|-1)}$ for all $v \in V$

$$w_{ij} = \begin{cases} 0, & i = j \\ w(i,j), & i \neq j \text{ and } (i,j) \in E \\ \infty, & i \neq j \text{ and } (i,j) \notin E \end{cases}$$

Bellman-Ford algorithm: implementation

- 共執行 $|V|-1$ 回合，每一回合中，relax 所有的邊，順序不重要
- 保證在第 k 回合結束後，節點 v 的最短路徑估計值 $\leq$ 所有邊數至多為 k 的 $s \rightsquigarrow v$ 路徑的最短距離（i.e., $\ell_{sv}^{(k)}$）
  - $|V|-1$ 回合結束後，節點 v 的最短路徑估計值 $\leq$ 所有邊數至多為 $|V|-1$ 的 $s \rightsquigarrow v$ 路徑的最短距離
  - 若最短路徑存在，由於最短路徑的邊數不會大於 $|V|-1$，因此 Bellman-Ford 結束後的確能正確算出最短路徑值

Bellman-Ford algorithm

```
BELLMAN-FORD(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    for i = 1 to |G.V|-1
        for (u,v) in G.E
            RELAX(u,v,w)
    for (u,v) in G.E
        if v.d > u.d + w(u,v)
            return FALSE
    return TRUE
```

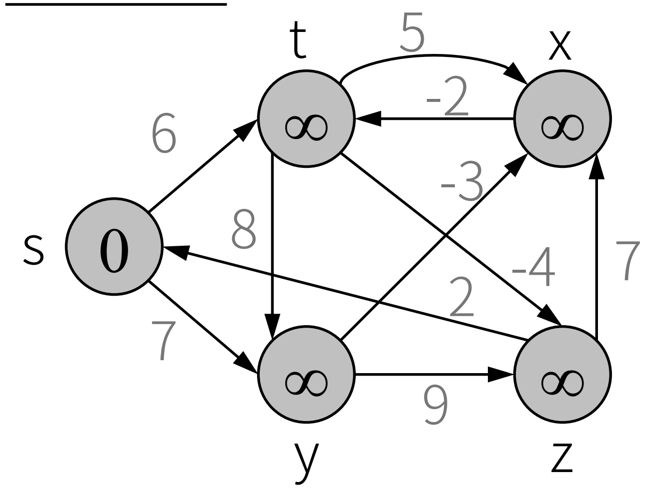
```
INITIALIZE-SINGLE-SOURCE(G,s)
    for v in G.V
        v.d = ∞
        v.π = NIL
    s.d = 0
```

```
RELAX(u, v, w)
    if v.d > u.d + w(u, v)
        //DECREASE-KEY
        v.d = u.d + w(u, v)
        v.π = u
```

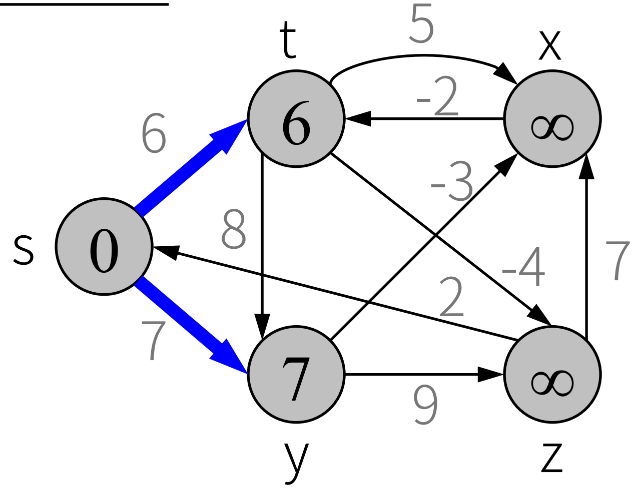
- Relax each edge e; repeat $V-1$ times
- Detect a negative cycle if exists
- Find shortest simple path if no negative cycle exists

Relaxation sequence in each iteration: $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$

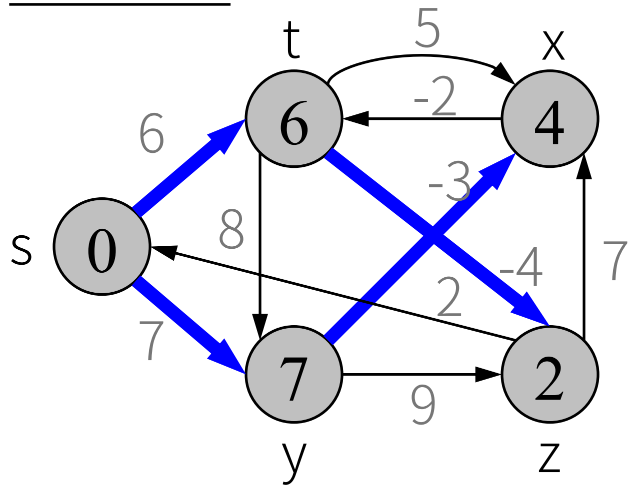
Iteration 0



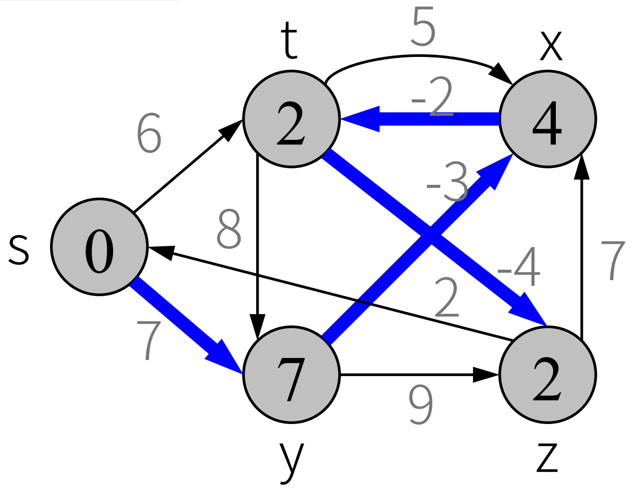
Iteration 1



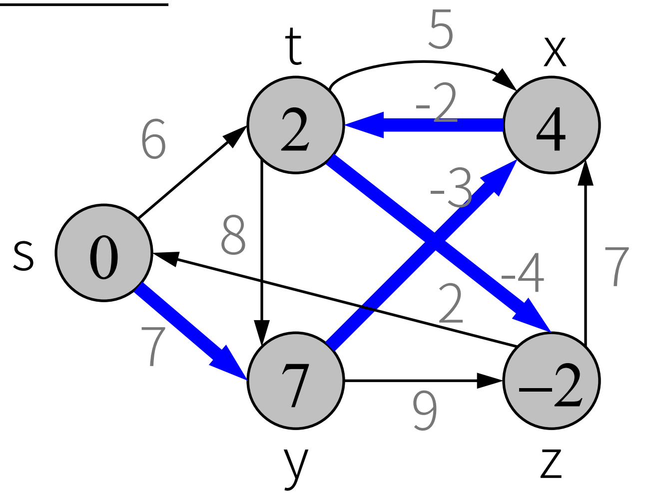
Iteration 2



Iteration 3



Iteration 4



Running time analysis

```
BELLMAN-FORD(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    for i = 1 to |G.V|-1
        for (u,v) in G.E
            RELAX(u,v,w)
    for (u,v) in G.E
        if v.d > u.d + w(u,v)
            return FALSE
    return TRUE
```

$\Theta(V)$

$\Theta((V-1)E)$

$\Theta(E)$

- Running time = $\Theta(VE)$, assuming we can enumerate all edges in $\Theta(E)$
- SPFA [1] can run in $\Theta(E)$ on average, but the worst case is still $\Theta(VE)$

[1] https://en.wikipedia.org/wiki/Shortest_Path_Faster_Algorithm

Correctness of Bellman-Ford (Theorem 24.4)

We want to prove the following two statements:

1. Correctly compute $\delta(s, v)$ when no negative-weight cycle
   - After the $|V| - 1$ iterations of relaxation of all edges, it must hold that $v.d = \delta(s, v)$ for all vertices $v \in V$ that are reachable from s
   - For each vertex$v \in V$, there is a path from s to v if and only if the algorithm terminates with $v.d < \infty$.
2. Correctly detect the existence of negative cycles
   - Return FALSE If G does contain a negative-weight cycle reachable from s

Correctness of Bellman-Ford (Theorem 24.4)

1. Correctly compute $\delta(s, v)$ when no negative-weight cycle
   - After the $|V| - 1$ iterations of relaxation of all edges, it must hold that $v.d = \delta(s, v)$ for all vertices $v \in V$ that are reachable from s

Proof

Although the shortest path p from s to v is unknown, we know it has at most $V - 1$ edges if the path exists

- The relaxation sequence must contain all edges in p in order:

$$\underbrace{\underbrace{e_1, e_2, \dots, e_m}_{\text{Must contain 1st edge in } p}; \underbrace{e_1, e_2, \dots, e_m}_{\text{Must contain 2nd edge in } p}; \cdots\cdots; e_1, e_2, \dots, e_m}_{\text{Repeated } V-1 \text{ times, must contain all edges in } p \text{ in order}} \qquad (m = |E|)$$

- According to path-relaxation property (Lemma 24.15), $v.d = \delta(s, v)$ for all vertices $v \in V$ that are reachable from s

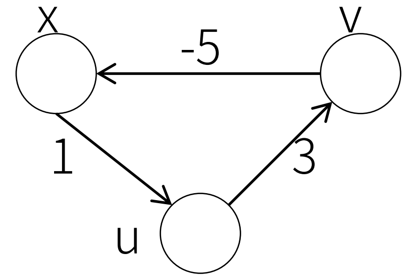
Correctness of Bellman-Ford (Theorem 24.4)

2. Correctly detect the existence of negative cycles
   - Return FALSE If G does contain a negative-weight cycle reachable from s

Proof by contradiction

- Suppose Bellman-Ford returns TRUE while G does contain a negative-weight cycle C reachable from s
- $\Rightarrow v.d \le u.d + w(u,v), \forall (u,v) \in E$
- Consider the edges on C,
- $\Rightarrow \sum_{v \in C} v.d \le \sum_{v \in C} u.d + \sum_{(u,v) \in C} w(u,v)$
- $\Rightarrow 0 \le \sum_{(u,v) \in C} w(u,v)$
- => contradiction

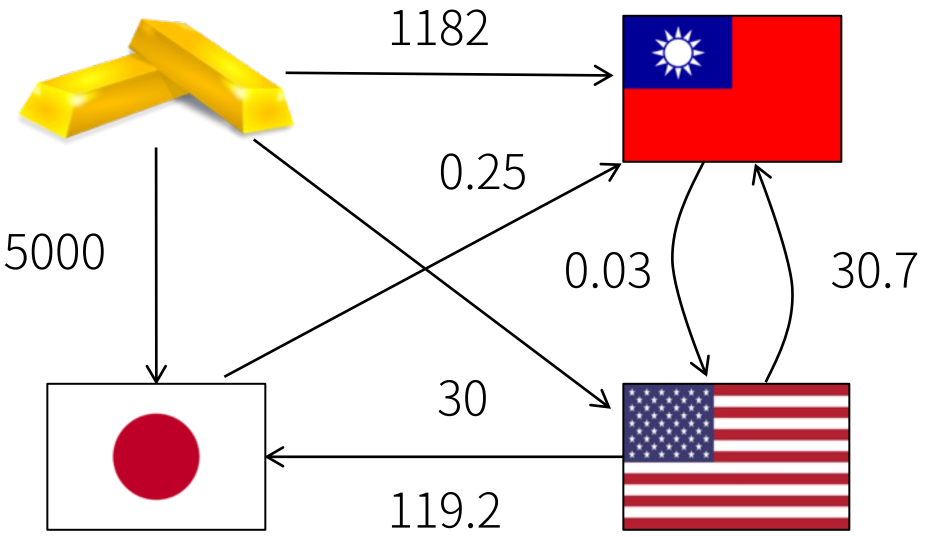
```
//negative cycle detection
    for (u,v) in G.E
        if v.d > u.d + w(u,v)
            return FALSE
```



Q: 匯率換算問題（假設零手續費）

a. 1 單位黃金最多可以換到多少 TWD？

b. 是否有套利空間（利用匯差賺錢）？

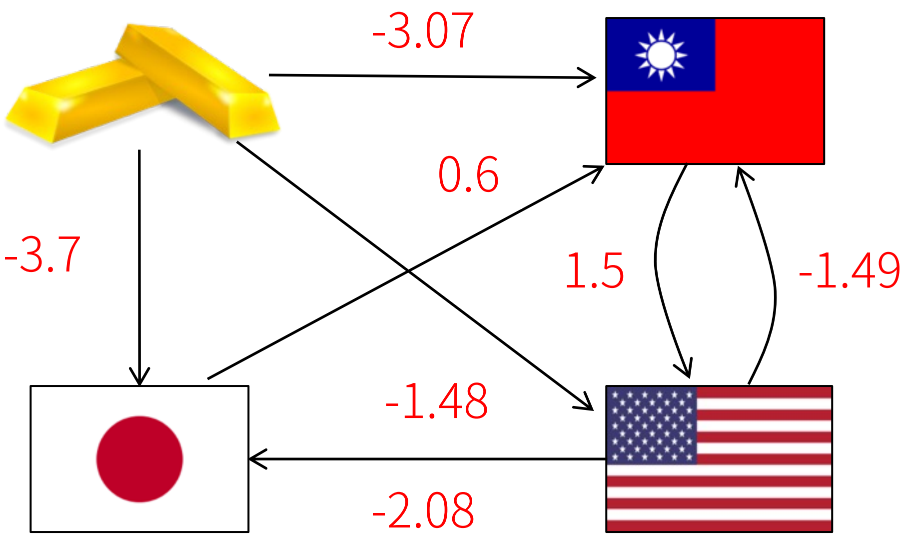


找weight相乘後最大路徑？

是否能轉成最短路徑問題？

Q: 匯率換算問題（假設零手續費）
a. 1公克黃金最多可以換到多少 TWD？
b. 是否有套利空間（利用匯差賺錢）？

After reweighting using $w'(e) = -\log w(e)$, we can find the shortest path (最佳的兌換率) and detect the existence of negative cycles (利用匯差賺錢).



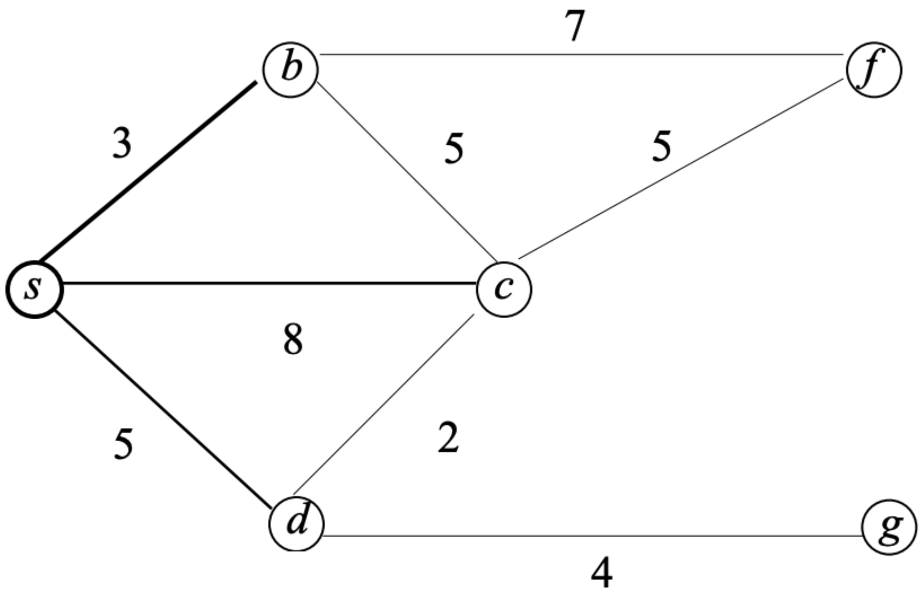
Reweighting:
$w'(e) = -\log w(e)$

Dijkstra’s algorithm

Textbook Chapter 24.3

Dijkstra's algorithm: intuition

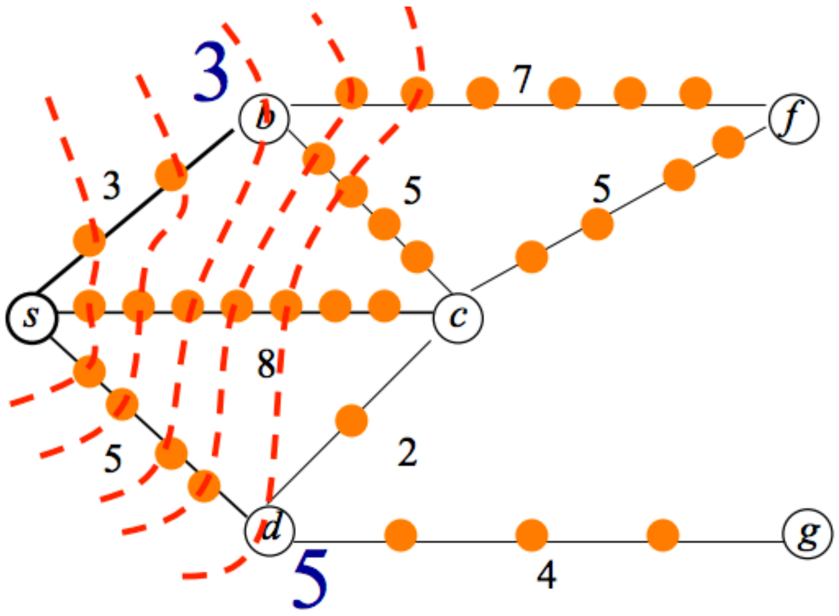
- Recall that BFS finds shortest paths on an unweighted graph by expanding the search frontier like ripples.
- Can we do the same on weighted graph?

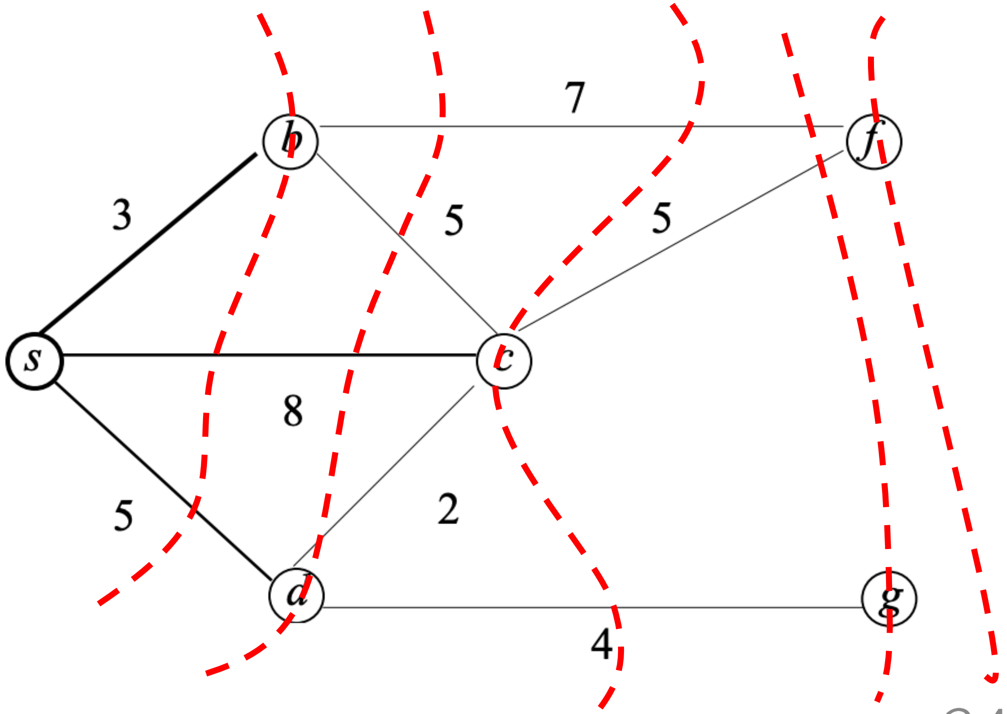


Source: http://courses.csail.mit.edu/6.006/spring11/lectures/lec16.pdf

Dijkstra's algorithm: intuition

- Dijkstra's algorithm speeds up the process by “skipping” layers that do not intersect with any vertex!



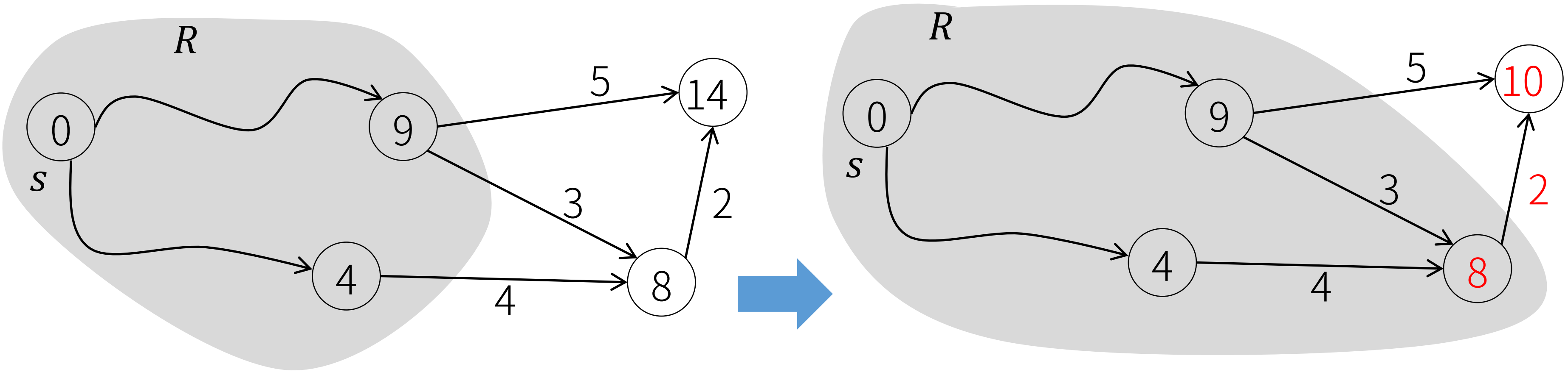


Source: http://courses.csail.mit.edu/6.006/spring11/lectures/lec16.pdf

Dijkstra's algorithm

Dijkstra greedily adds vertices by increasing distance

- Maintains a set of explored vertices R whose final shortest-path weights have already been determined
  1. Initially, $R = \{s\}, s.d = 0$
  2. At each step, select unexplored vertex u in $V - R$ with minimum $u.d$
  3. Add u to R, and relaxes all edges leaving u. Go back to Step 2.



Implementation of Dijkstra's algorithm

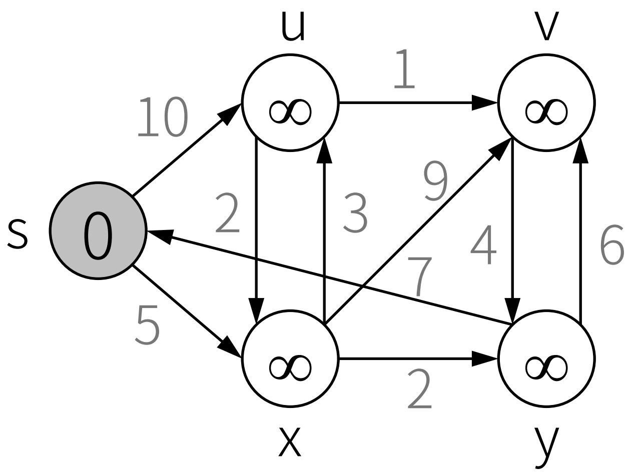
```
DIJKSTRA(G,w,s)
  INITIALIZE-SINGLE-SOURCE(G,s)
  R = empty
  Q = G.v //BUILD-PRIORITY-QUEUE
  while Q ≠ empty
      u = EXTRACT-MIN(Q)
      R = R U {u}
      for v in G.adj[u]
          RELAX(u,v,w)
```

```
INITIALIZE-SINGLE-SOURCE(G,s)
    for v in G.V
        v.d = ∞
        v.π = NIL
    s.d = 0
```

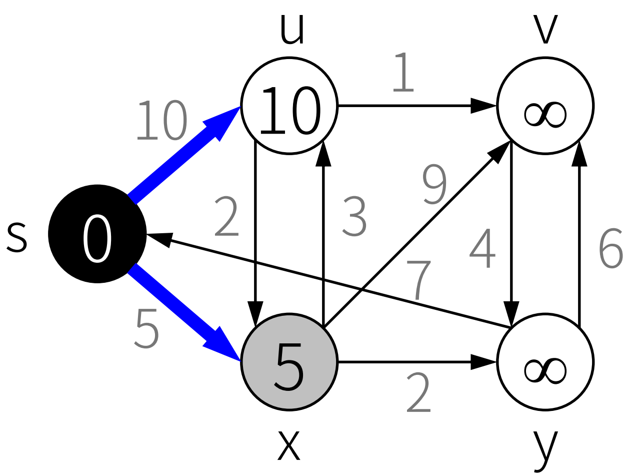
```
RELAX(u, v, w)
    if v.d > u.d + w(u, v)
        //DECREASE-KEY
        v.d = u.d + w(u, v)
        v.π = u
```

- Q is a min-priority queue of vertices, keyed by d values
- Observations (will prove these later)
  - For u in Q (that is, $V - R$), $u.d$ is the shortest-path estimate (i.e., minimum length over all observed $s \rightsquigarrow u$ paths so far).
  - For u in R, $u.d = \delta(s, v)$

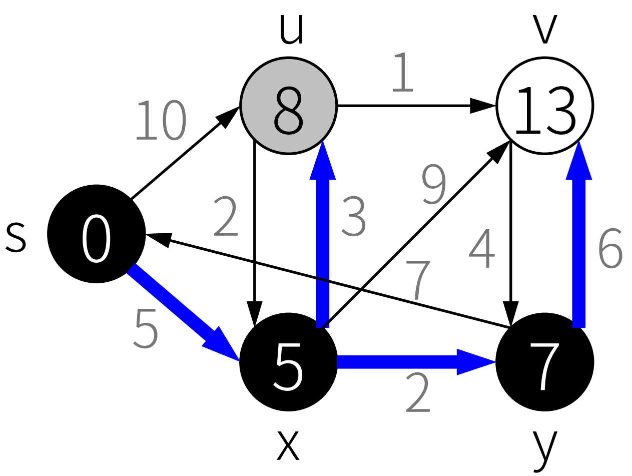
Step 0



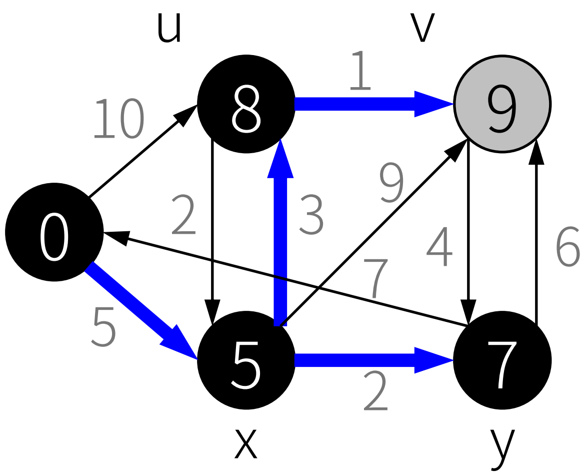
Step 1



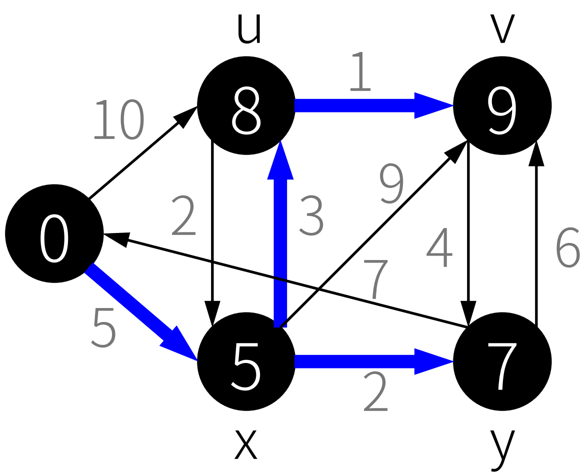
Step 3



Step 4



Step 5



Black: in R
White: in Q
Grey: selected
Blue line: shortest path tree

Running time analysis

- Q is a min-priority queue of vertices, keyed by d values
  - # of INSERT = $\Theta(V)$
  - # of EXTRACT-MIN = $\Theta(V)$
  - # of DECREASE-KEY = $\mathrm{O}(E)$
- The running time depends on queue implementation
  - Implementing the min-priority queue using an array indexed by v: $O(V^2 + E) = O(V^2)$
    - INSERT: $O(1)$
    - EXTRACT-MIN: $O(V)$
    - DECREASE-KEY: $O(1)$
  - Can be improved to $O(E + V \lg V)$ using Fibonacci heaps

Correctness of Dijkstra's algorithm (Theorem 24.6)

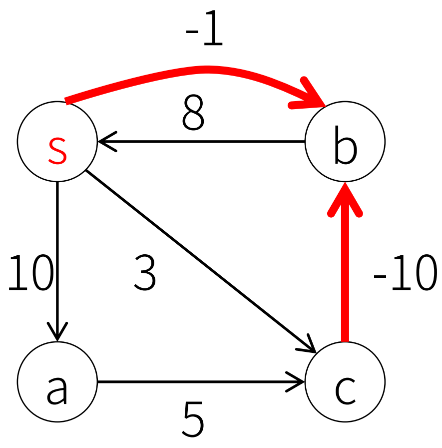
Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with non-negative weight function w and source s, terminates with $u.d = \delta(s, u)$ for all vertices $u \in V$.

Idea

- R: the set of explored vertices whose final shortest-path weights have already been determined
  - Initially, $R = \{s\}, s.d = 0$
  - Invariant: for all u in R, $u.d = \delta(s, u)$, the length of the shortest path from s to u
  - Note that for u in $V - R$, $u.d$ = length of some path from s to u
- We want to prove that the loop invariant holds throughout the execution of the algorithm.

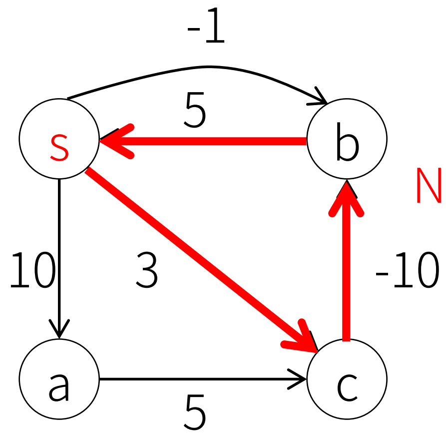
Dijkstra's algorithm may work incorrectly with negative-weight edges

- C.f. Bellman-Ford: a dynamic programming algorithm either detects negative cycles or returns the shortest-path tree



Negative-wight edges

$\delta(s, b) = -7$
In Dijkstra, $b.d = -1$



Negative-weight cycle

$\delta(s, b) = -\infty$
In Dijkstra, $b.d = -1$

Q: What is the similarity between BFS, DFS, Prim and Dijkstra?

- They are each a special case of priority-first search

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

```
MST-PRIM(G, w, r)
for u in G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ empty
    u = EXTRACT-MIN(Q)
    for v in G.adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

```
DIJKSTRA(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
R = empty
Q = G.v
while Q ≠ empty
    u = EXTRACT-MIN(Q)
    R = R ∪ {u}
    for v in G.adj[u]
    RELAX(u,v,w)
```

Priority-first search

- Maintain a set of explored vertices S
- Grow S by exploring highest-priority edges with exactly one endpoint leaving S

Q: What's the priority in each variant (BFS, DFS, Prim and Dijkstra)?

BFS: edges from the earliest discovered/explored vertex

DFS: edges from the latest discovered/explored vertex

Prim: edges of minimum weight

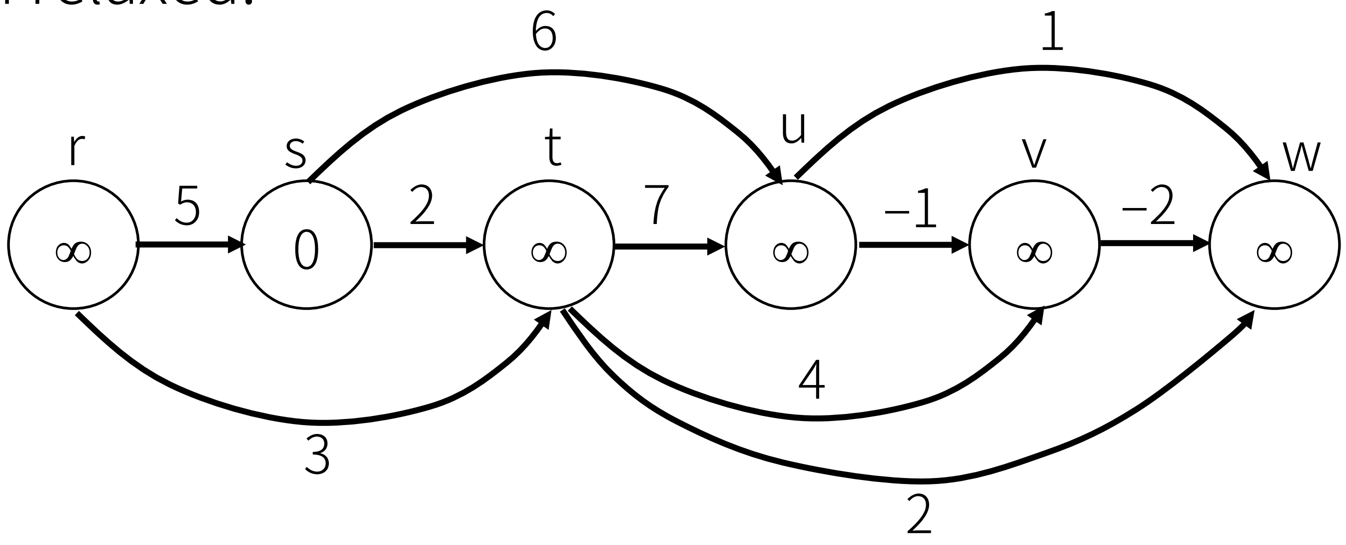
Dijkstra: edges to vertex closest to s

Single-source shortest paths in directed acyclic graphs

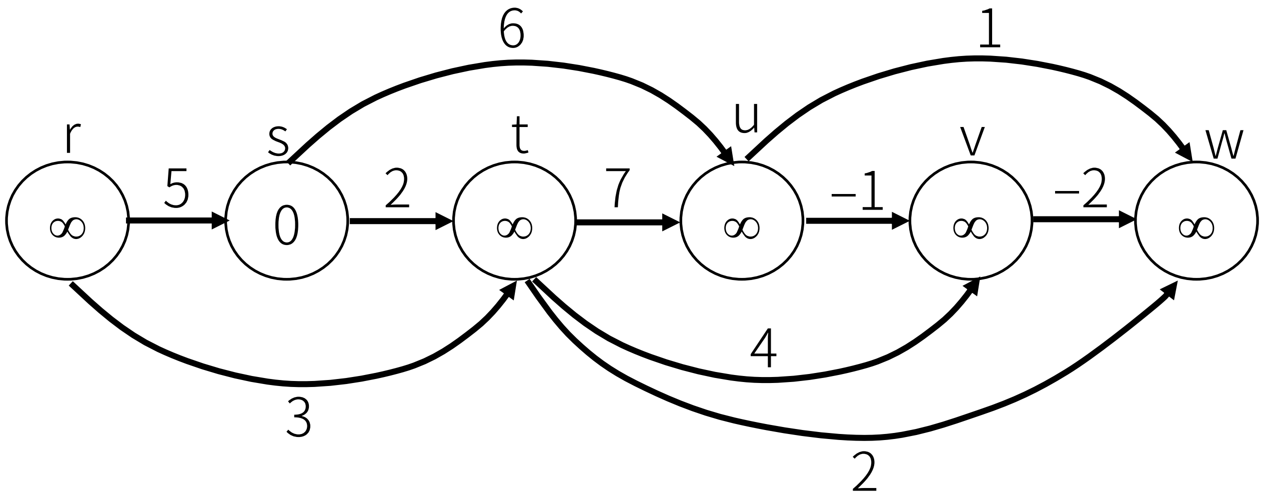
Textbook Chapter 24.2

Single-source shortest paths in DAG

- Claim: relaxing the edges in topologically sorted order correctly computes the shortest paths in DAG
- Intuition: putting vertices in a topologically sorted order, edges only go from left to right; so when relaxing an edge (u, v), all edges to u must have been relaxed.

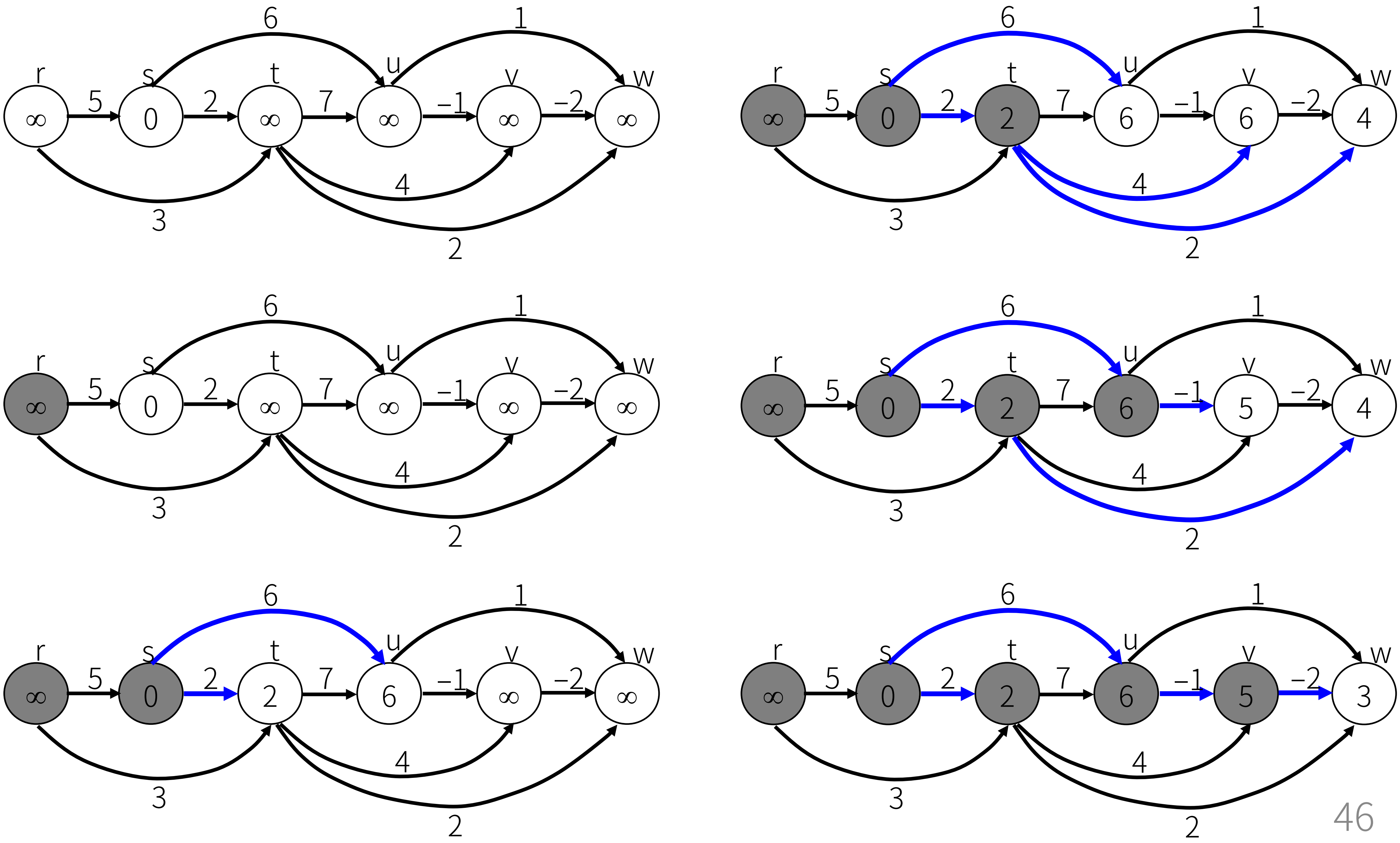


```
DAG-SHORTEST-PATHS(G,w,s)
1   topologically sort the vertices of G
2   INITIALIZE-SINGLE-SOURCE(G,s)
3   for each vertex u, taken in topologically sorted order
4       for each vertex v in G.adj[u]
5           RELAX(u,v,w)
```




```
INITIALIZE-SINGLE-SOURCE(G,s)
    for v in G.V
        v.d = ∞
        v.π = NIL
    s.d = 0
```

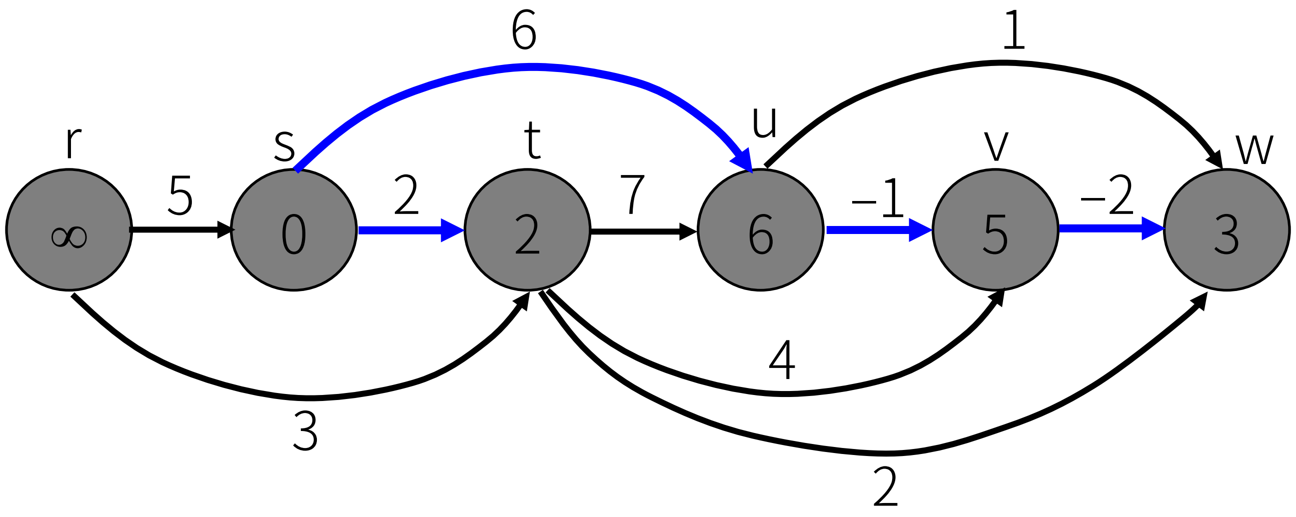
```
RELAX(u, v, w)
    if v.d > u.d + w(u, v)
        //DECREASE-KEY
        v.d = u.d + w(u, v)
        v.π = u
```


r s t u v w
5 2 7 −1 −2
6 1 3 4 2
∞ 0 ∞ ∞ ∞ ∞
∞ 0 ∞ ∞ ∞ ∞
∞ 0 2 6 ∞ ∞
∞ 0 2 6 6 4
∞ 0 2 6 5 4
∞ 0 2 6 5 3

Running time analysis

```
DAG-SHORTEST-PATHS(G,w,s)
1   topologically sort the vertices of G  // Θ(V+E)
2   INITIALIZE-SINGLE-SOURCE(G,s)  // Θ(V)
3   for each vertex u, taken in topologically sorted order  ⎤
4       for each vertex v in G.adj[u]                        ⎬ Θ(V+E)
5           RELAX(u,v,w)                                     ⎦
```

=> total running time is $\Theta(V + E)$, same as topological sort



Theorem 24.5

If $G = (V, E)$ is a DAG, then at the termination of DAG-SHORTEST-PATHS, $v.d = \delta(s, v)$, for all $v \in V$

Proof by induction on the position in topological sort order

- Inductive hypothesis: if all the vertices before v in a topological sort order have been updated, then $v.d = \delta(s, v)$
- Base case:
  - For all v before s, $v.d = \infty = \delta(s, v)$
  - For s, $s.d = 0 = \delta(s, s)$

Theorem 24.5

If $G = (V, E)$ is a DAG, then at the termination of DAG-SHORTEST-PATHS, $v.d = \delta(s, v)$, for all $v \in V$

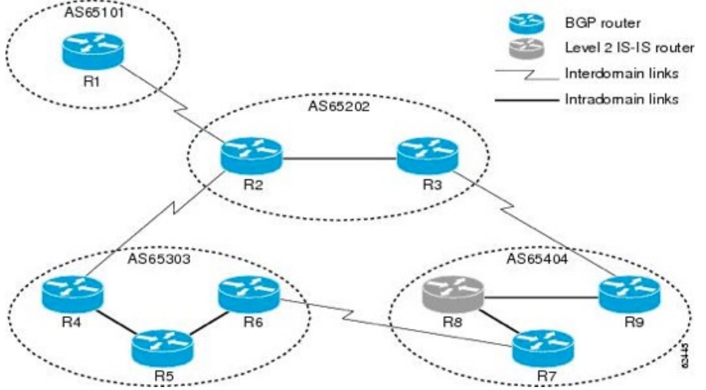
Proof by induction on the position in topological sort order (Cont.)

- Inductive hypothesis: if all the vertices before v in a topological sort order have been updated, then $v.d = \delta(s, v)$
- Inductive step:
  - Consider a vertex v (to the right of s)
  - By construction, $v.d = \min_{(u,v) \in E} (u.d + w(u, v))$
  - By inductive hypothesis, $u.d = \delta(s, u)$
  - Since some (u, v) must be on the shortest path, by optimal substructure, $v.d = \delta(s, v)$

Single-source shortest-path algorithms

SSSP algorithm	Applicable graph types	Running time
Dijkstra	Nonnegative weights	$O(V^2)$ (array-based)
Topological sort based	DAG	$O(V+E)$
Bellman-Ford	generic	$O(VE)$

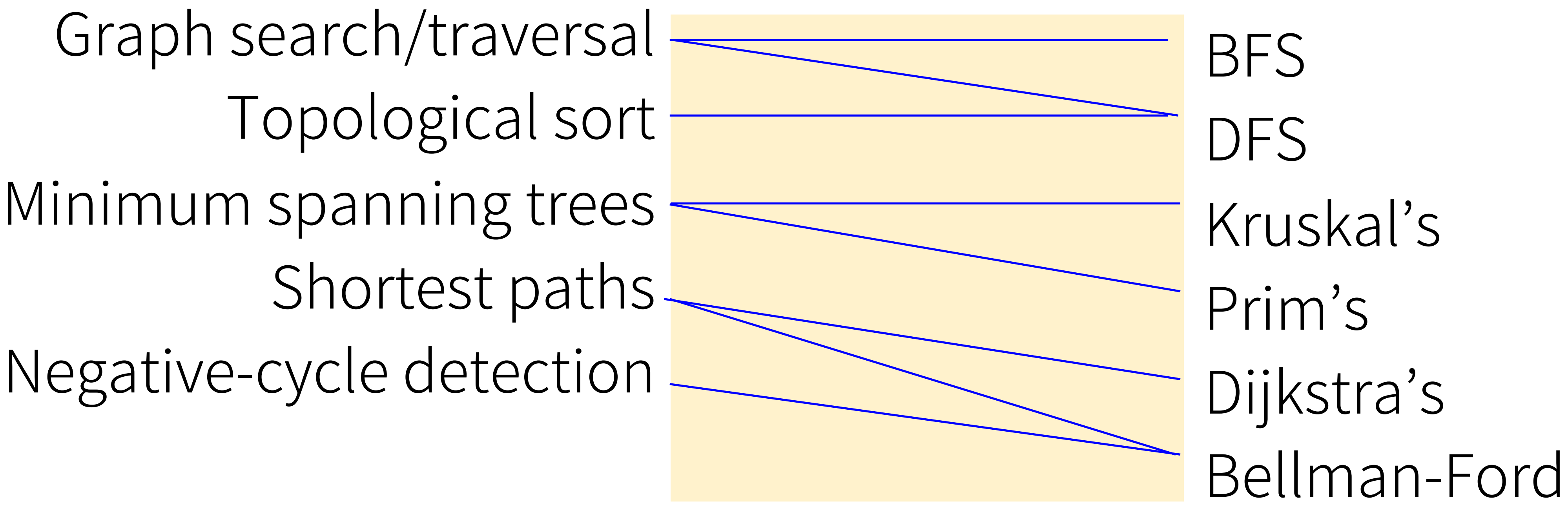
Application: Internet routing



Source: cisco.com

- Vertices = routers, ASes
- Edges = network links between routers
- Edge weight = delay, cost, hop count, etc.
- Link-state (commonly using Dijkstra's algorithm)
  - Nodes flood link state to whole network
  - E.g., Open Shortest Path First (OSPF)
- Distance-vector (commonly using Bellman-Ford's algorithm)
  - Nodes send vectors of destination and distance to neighbors
  - E.g., Routing Information Protocol (RIP)
- Path-vector (not necessarily shortest paths)
  - Nodes advertise the full paths to each destination
  - E.g., Border Gateway Routing Protocol (BGP)

Summary of graph algorithms



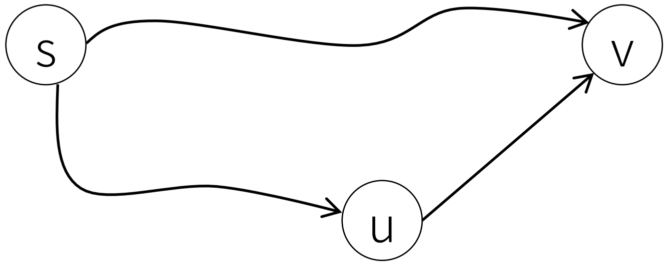
Appendix: Proofs of Shortest-path Properties

Triangle inequality (Lemma 24.10)

For any edge $(u, v) \in E$, $\delta(s, v) \le \delta(s, u) + w(u, v)$

Proof

- By definition, $\delta(s, v)$ is the minimum weight of all paths from s to t
- Consider a shortest path from $s \leadsto u$ and the edge (u, v). Together, it forms one of the paths from s to v, whose weight is $\delta(s, u) + w(u, v)$
- => $\delta(s, v) \le \delta(s, u) + w(u, v)$



Upper-bound property (Lemma 24.11)

Let the graph be initialized by `INITIALIZE-SINGLE-SOURCE`(G, s). We always have $v.d \geq \delta(s, v)$ for all vertices $v \in V$ over any sequence of relaxation steps, and once $v.d$ achieves the value $\delta(s, v)$, it never changes.

Proof

We can prove this by induction over the number of relaxation steps

Base case:

At the beginning, $v.d = \infty \geq \delta(s, v)$ for all $v \in V - \{s\}$. Also, $s.d = 0 \geq \delta(s, s)$.

Inductive case:

Consider relaxing edge (u, v), which may change the value of $v.d$ but not others. If it changes, $v.d = u.d + w(u, v) \geq \delta(s, u) + w(u, v) \geq \delta(s, v)$

Because $v.d$ can never increase and always $\geq \delta(s, v)$, it will never change once reaching $\delta(s, v)$.

No-path property (Corollary 24.12)

If there is no path from s to v, then we always have $v.d = \delta(s, v) = \infty$

Proof

- By the upper-bound property, we always have $v.d \geq \delta(s, v)$.
- => $v.d = \delta(s, v) = \infty$

Convergence property (Lemma 24.14)

If $s \rightsquigarrow u \rightarrow v$ is a shortest path in G for some $u, v \in V$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all times afterward.

Proof

- By definition, immediately after relaxing (u, v), $v.d$ will not exceed $u.d + w(u, v)$. Thus, immediately after relaxing (u, v),
- => $v.d \leq u.d + w(u, v) = \delta(s, u) + w(u, v) = \delta(s, v)$ [why?]
- Also, by the upper-bound property, $v.d \geq \delta(s, v)$
- => $v.d = \delta(s, v)$ immediately after relaxing (u, v)
- => $v.d = \delta(s, v)$ at all times afterward, according to the upper-bound property

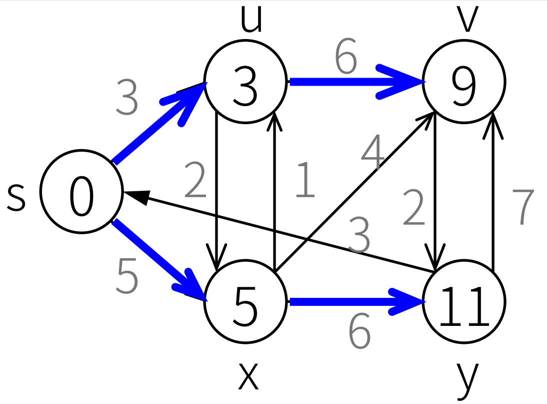
Predecessor-subgraph property (Lemma 24.17)

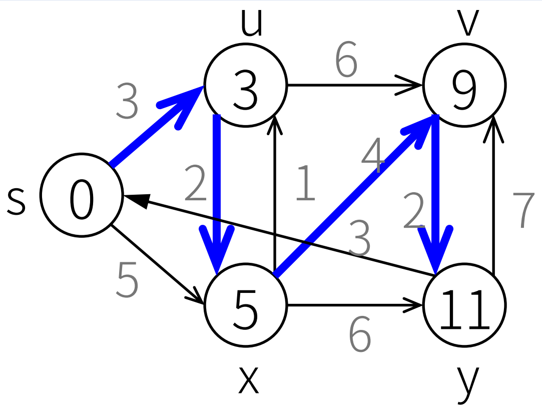
Suppose G contains no negative-weight cycles reachable from s. Once $v.d = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

Shortest-paths tree

A shortest-paths tree $G' = (V', E')$ of s is a subgraph of G s.t.:

- V' is the set of vertices reachable from s in G
- G' forms a rooted tree with root s
- For all v in V', the unique simple path from s to v in G' is a shortest path from s to v in G





Appendix: Correctness of Dijkstra’s algorithm

Correctness of Dijkstra's algorithm (Theorem 24.6)

Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with non-negative weight function w and source s, terminates with $u.d = \delta(s, u)$ for all vertices $u \in V$.

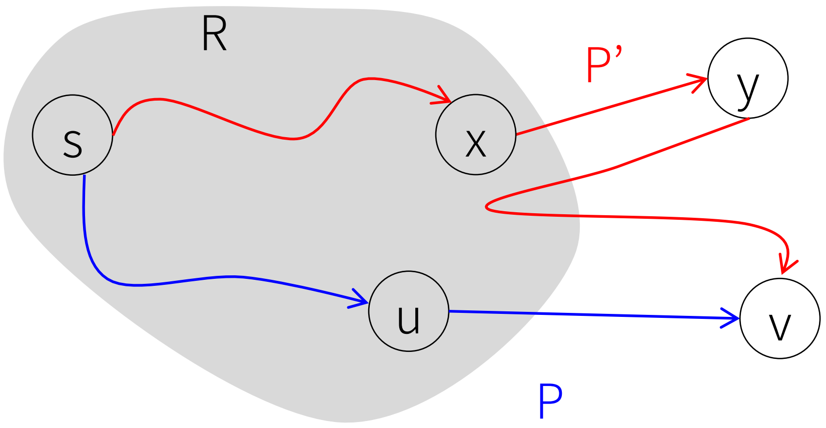
Idea

- R: the set of explored vertices whose final shortest-path weights have already been determined
  - Initially, $R = \{s\}, s.d = 0$
  - Invariant: for all u in R, $u.d$ = length of the shortest path from s to u
  - Note that for u in $V - R$, $u.d$ = length of some path from s to u
- We want to prove that the loop invariant holds throughout the execution of the algorithm.

Loop invariant: for u in R, $u.d = \delta(s, u)$

Proof by induction on the size of R

- Base case: $|R| = 1$, correct
- Inductive step: Let v be the next vertex to be added to R, $u = v.\pi$, P = shortest path from s to $u + (u, v)$
- $\Rightarrow v.d = w(P) = \delta(s, u) + w(u, v)$
- Consider any other $s \leadsto v$ path P'
- We want to prove that $w(P') \geq w(P)$



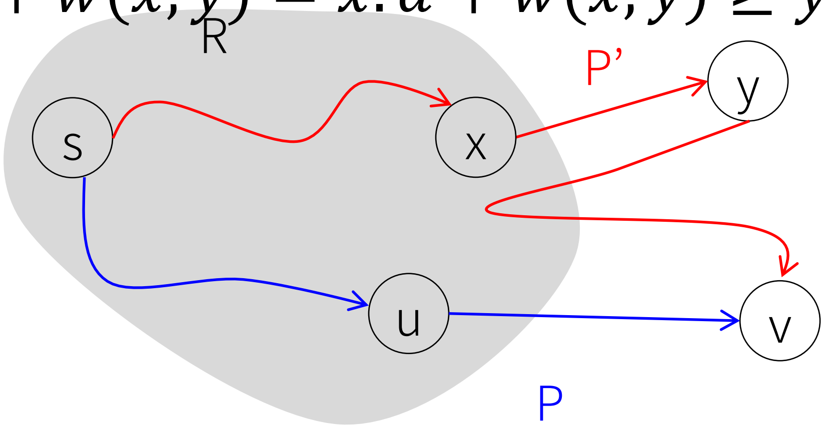
Loop invariant: for u in R, $u.d = \delta(s,u)$

Proof by induction on the size of R (cont'd)

- Prove that $w(P') \geq w(P)$
- Let y be the first vertex on path P' outside R

1. Because of no negative edges, $w(P') \geq \delta(s,x) + w(x,y)$
2. By induction hypothesis, $x.d = \delta(s,x)$
3. By construction, $y.d \geq v.d$
4. By construction, $y.d \leq x.d + w(x,y)$

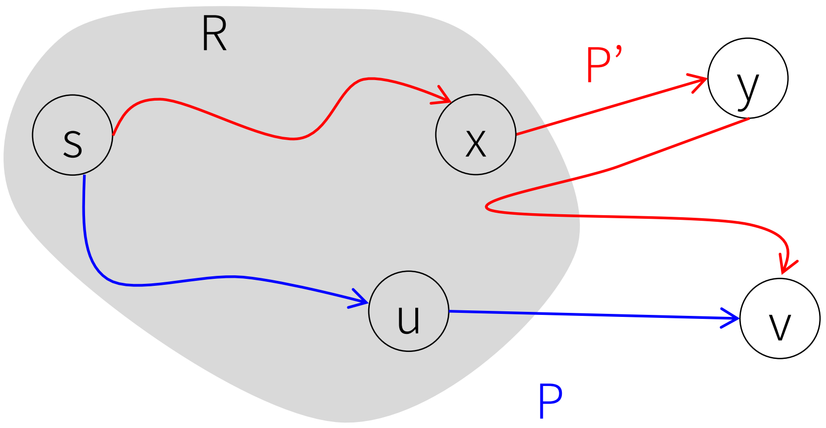
- $\Rightarrow$ $w(P') \geq \delta(s,x) + w(x,y) = x.d + w(x,y) \geq y.d \geq v.d = w(P)$



Loop invariant: for u in R, $u.d = \delta(s, u)$

Proof by induction on the size of R (cont'd)

- Hence, the greedy choice v (and the corresponding path P) is at least as good as any other path from s to v
- => The invariant still holds after adding one more vertex v to R
- At termination, every vertex is in R
- Thus, $u.d = \delta(s, v)$ for all u in V



Appendix:
All-pairs Shortest Paths

Variants of shortest-path problems

- Single-source shortest-path problem: Given a graph $G = (V, E)$ and a source vertex s in V, find the minimum cost paths from s to every vertex in V
- Single-destination shortest-path problem: Given a graph $G = (V, E)$ and a destination vertex t in V, find the minimum cost paths to t from every vertex in V
- Single-pair shortest-path problem: Find a shortest path from s to t for given s and t
- All-pair shortest path problem: Find a shortest path from s to t for every pair of s and t

All-pairs shortest paths Algorithms

- Repeated squaring of matrices
- Floyd-Warshall algorithm
- Johnson’s algorithm

Recap: DP view of Bellman-Ford algorithm

- Let $\ell_{sv}^{(k)}$ be the shortest path value from s to v using at most k edges
  - Subproblems: given s, $\ell_{sv}^{(k)}$ for all v, k
  - Optimal substructure: by Lemma 24.1
- Base case: $\ell_{ss}^{(0)} = 0$; $\ell_{sv}^{(0)} = \infty$ when $s \neq v$
- Recurrence relation can be formulated as

$$\ell_{sv}^{(k)} = \min_{u \in V} \left\{ \ell_{su}^{(k-1)} + w_{uv} \right\}$$

- Optimal values: $\ell_{sv}^{(|V|-1)}$ for all $v \in V$

$$w_{ij} = \begin{cases} 0, & i = j \\ w(i,j), & i \neq j \text{ and } (i,j) \in E \\ \infty, & i \neq j \text{ and } (i,j) \notin E \end{cases}$$

Generalization to all-pairs shortest paths

- Let $\ell_{ij}^{(k)}$ be the shortest path value from i to j using at most k edges
  - Subproblems: $\ell_{ij}^{(k)}$ for all i, j, k
  - Optimal substructure: by Lemma 24.1
- Base cases: $\ell_{ii}^{(0)} = 0; \ell_{ij}^{(0)} = \infty$ when $i \neq j$
- Recurrence relation can be formulated as

$$\ell_{ij}^{(k)} = \min_{x \in V} \left\{ \ell_{ix}^{(k-1)} + w_{xj} \right\}$$

- Optimal values: $\ell_{ij}^{(|V|-1)}$ for all $i, j \in V$

```
//Extend shortest paths by one hop
EXTEND-SHORTEST-PATHS(L, W)
    n = W.rows
    let L' = (ℓ_ij') be a new nxn matrix
    for i = 1 to n
        for j = 1 to n
                ℓ'_ij = min_{x∈V} {ℓ_ix + w_xj}
    return L'
```

```
for x = 1 to n
      ℓ'_ij = min{l'_ij, ℓ_ix + w_xj}
```

- $L^{(k)} = (\ell_{ij}^{(k)})$, the matrix of $\ell_{ij}^{(k)}$s
- $W = (w_{ij})$, the matrix of w_{ij}s
- $L^{(1)} = W$
- Running time of Extend-Shortest-Paths: $\Theta(V^3)$

Similarity to matrix multiplication

- Think of `EXTEND-SHORTEST-PATHS(L,W)` as "multiplying" the two matrics, $L \cdot W$
  - $+$ is replaced by min, $\cdot$ is replaced by $+$
  - 0 (the identity for $+$) is replaced by ∞ (the identity for min)
- Then we have
  - $L^{(1)} = W$
  - $L^{(k)} = L^{(k-1)} \cdot W = W^k$
- Shortest path wights are: $L^{(n-1)} = W^{n-1}$
- The overall running time: $\Theta(V^4)$

Can we do better than $\Theta(V^4)$?

Observation: $L^{(k)} = L^{(n-1)}$ for all $k \geq n-1$

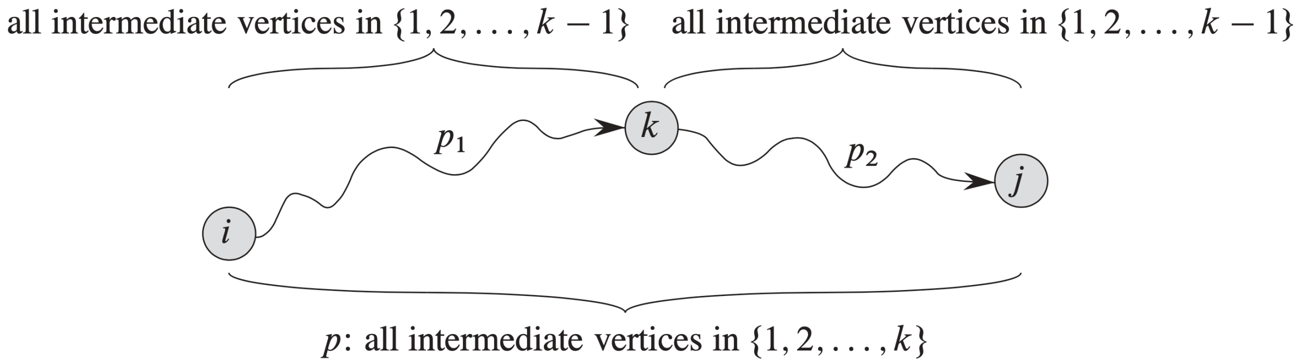
Q: Based on this observation, can we reduce it to $\Theta(V^3 \lg V)$?

Repeated squaring: keep squaring W for r times until $2^r > n-1$

Floyd-Warshall algorithm

Floyd-Warshall algorithm: intution

- Consider a shortest path p_{ij} from i to j whose imtermediate vertices are all in $\{1,2,\dots,k\}$
- Depdending on whether k is an intermediate vertex of p_{ij}, there are two possible cases:
  - k is not an intermediate vertex of p_{ij} : all intermediate vertices are in $\{1,2,\dots,k-1\}$
  - k is an intermediate vertex of p_{ij}: p_{ij} can be decomposed into two sub-paths, $p_{ij} = i \rightsquigarrow k \rightsquigarrow j$, and the first (second) sub-path is a shortest path from i to k (k to j) with all intermediate vertices in $\{1,2,\dots,k-1\}$.



Floyd-Warshall algorithm: intution

- Based on the observation, we can define a recurrence relation among shortest paths
- Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to j whose imtermediate vertices are all in $\{1,2,\ldots,k\}$

$$d_{ij}^{(k)} = \begin{cases} w_{ij}, & k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right), & k \geq 1 \end{cases}$$

$$w_{ij} = \begin{cases} 0, & i = j \\ w(i,j), & i \neq j \text{ and } (i,j) \in E \\ \infty, & i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- Claim: $d_{ij}^{(n)} = \delta(i,j)\ \forall i,j \in V$

Floyd-Warshall algorithm

```
FLOYD-WARSHALL(W)// W is the matrix of wijs
    n = W.rows
    D(0)= W
    for k = 1 to n
        let D(k) = (d_ij^(k)) be a new nxn matrix
        for i = 1 to n
            for j = 1 to n
                d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
    return D(n)
```

Q: What's the running time?

$\Theta(n^3)$

Q: How to construct the shortest paths?

Exercise 25.2-3, Exercise 25.2-7

Q: Can the following variant correctly compute all-pairs shortest path values?

```
FLOYD-WARSHALL-1(W)// W is the matrix of w_ijs
    n = W.rows
    D^(0) = W
    for k = 1 to n
        let D^(k) = (d_ij^(k)) be a new nxn matrix
    for i = 1 to n
        for j = 1 to n
            for k = 1 to n
                d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
    return D^(n)
```

No

Johnson's algorithm for sparse graphs

Key idea: Reweighing

- Observation: If all edge weights are nonnegtive, simply run Dijkstra's algorithm from each vertex
  - $O(V^2 \lg V + VE)$ using Fibonacci-heap min-priority queue

- Can we somehow reweigh each edge such that all edge weights become nonnegative, while preserving the shortest paths?

Key idea: Reweighing

- Reweighing (using weight function $\hat{w}$ instead of w) should satisfy two important properties:
  1. Shortest-path preservation: $\forall\, u, v \in V$, a path p is a shortest path from u to v using weight function $w \Leftrightarrow \forall\, u, v \in V$, a path p is a shortest path from u to v using weight function $\hat{w}$
  2. Nonnegative weights: $\forall\, u, v \in V$, $\hat{w}(u, v)$ is nonnegative

Preserving shortest paths by reweighting

- Let $h: V \to \mathbb{R}$ be any function mapping vertices to real numbers
- Define a new weight function as

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

Q: Show that this reweighting preserve shortest paths

Q: Show that G has a negative-weight cycle using $w \Leftrightarrow G$ has a negative-weight cycle using $\hat{w}$

Producing nonnegative weights by reweighting

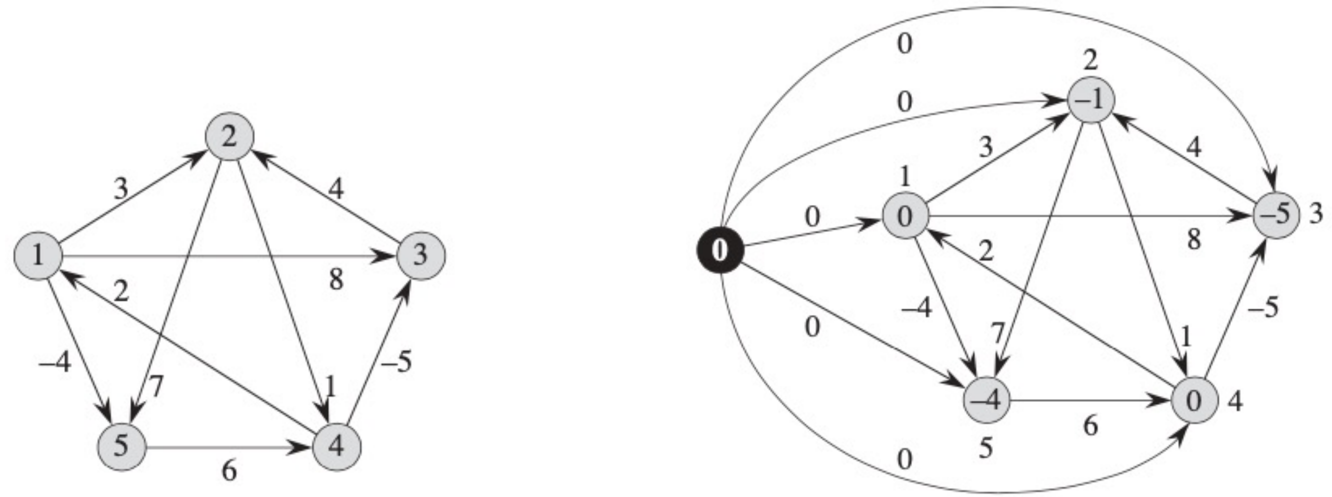
- Goal: Pick a function $h: V \to \mathbb{R}$ such that for all $u, v \in V$
$$\hat{w}(u,v) = w(u,v) + h(u) - h(v) \geq 0$$
- Johnson's algorithm takes advantage of the triangle inequality for shortest paths (Lemma 24.10)

Triangle inequality (Lemma 24.10)

Given a source vertex s, for any edge $(u,v) \in E$, $\delta(s,v) \leq \delta(s,u) + w(u,v)$

Producing nonnegative weights by reweighting

- Pick a function $h: V \to \mathbb{R}$ such that for all $u, v \in V$
$$\hat{w}(u,v) = w(u,v) + h(u) - h(v) \geq 0$$
  - Add an additional source vertex s
  - Add an edge from s to every vertex v in the original graph, $w(s,v) = 0$
  - Let $h(v) = \delta(s,v)$, which can be computed using Bellman-Ford algorithm



Johnson's Algorithm

```
JOHNSON(G, w)
compute G′, where G′.V = G.V ∪ {s},
      G′.E = G.E ∪ {(s, v) : v ∈ G.V}, and
      w(s, v) = 0 for all v ∈ G.V
if BELLMAN-FORD(G′, w, s) == FALSE
      print "the input graph contains a negative-weight cycle"
else for each vertex v ∈ G′.V
            set h(v) to the value of δ(s, v)
                  computed by the Bellman-Ford algorithm
      for each edge (u, v) ∈ G′.E
            ŵ(u, v) = w(u, v) + h(u) − h(v)
      let D = (d_uv) be a new n × n matrix
      for each vertex u ∈ G.V
            run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
            for each vertex v ∈ G.V
                  d_uv = δ̂(u, v) + h(v) − h(u)
      return D
```

1. Transform the graph and run Bellman-Ford algorithm from the added source vertex

2. Reweigh edges

3. Run Dijkstra from each vertex and reconstruct the original distance

Time complexity

- Johnson's algorithm: $O(V^2 \lg V + VE)$
- C.f. Floyd-Warshall algorithm: $\Theta(V^3)$

Q: When will Johnson's algorithm run faster than Floyd-Warshall algorithm?
On sparse graphs, i.e., $|E| \sim |V|$