------------------------------ 24 ------------------------------
Single Source Shortest Paths

Single-source shortest paths problem

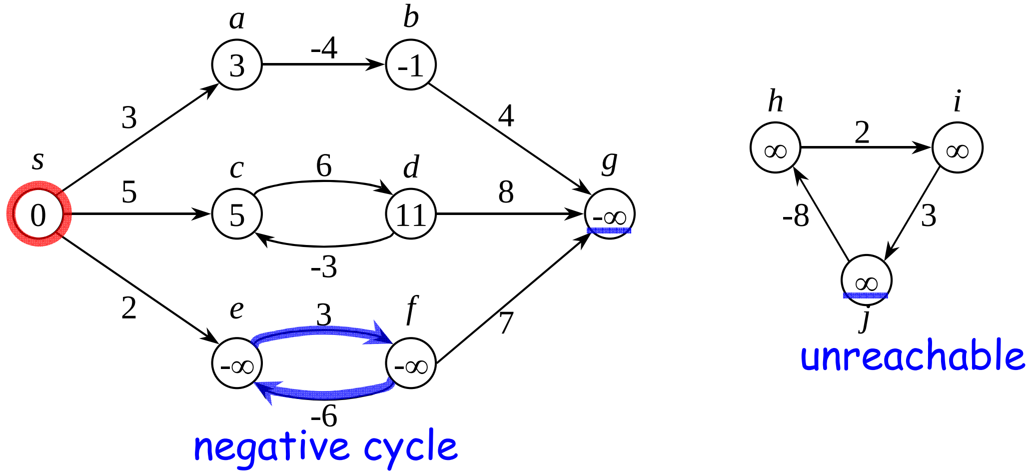
Input: A weighted directed graph $G=(V, E)$ and a source vertex s.

Output: for every $v \in V$, find a shortest path from s to v.

Negative-weight edges:

$\delta(u, v)$: the shortest-path weight from u to v.

If G contains no negative-weight cycles reachable from s, then $\delta(s, v)$ is well-defined for all $v \in V$. Otherwise, if there is a negative-weight cycle on some path from s to v, we define $\delta(s, v)=-\infty$.



Variants (Applications):

Single-destination shortest paths problem 24-2x

(Transform to single-source by reversing edges.)

Single-pair shortest paths problem

(No faster algorithms exist.)

All-pairs shortest paths problem

(Solve single-source problem V times. Faster algorithms exist: Chapter 25.)

Representing shortest paths:

for every $v \in V$, maintain a predecessor $\pi(v)$.

Predecessor subgraph: $G_\pi = (V_\pi, E_\pi)$, where 24-2y

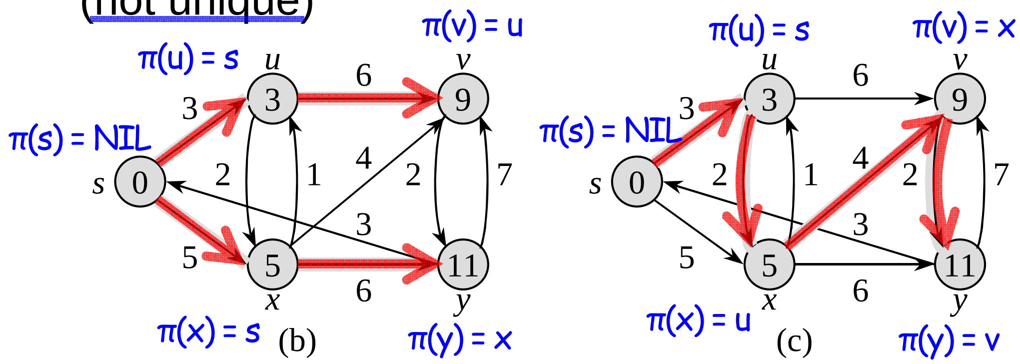
$V_\pi=\{v \mid v \in V \text{ and } \pi(v) \neq \text{NIL}\} \cup s$ ≠ V

(similar to BFS)

$E_\pi=\{(\pi(v), v) \mid v \in V_\pi - \{s\}\}$

G_π is a *shortest-paths tree* rooted at s.

(not unique)



Relaxation:

Repeatedly decrease $d[v]$ until $d[v]=\delta(s, v)$. 目前已知最好的答案

Two procedures:

INITIALIZE-SINGLE-SOURCE(G, s)

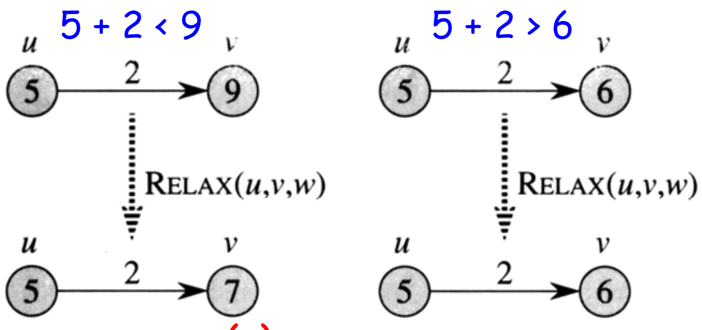
```
for each vertex v ∈ V[G]
    do d[v] ← ∞
       π[v] ← NIL
d[s] ← 0
```

RELAX(u, v, w)

① 存在 edge (u, v)
② 試著用 u 的答案去修正 v 的答案

```
if d[v] > d[u] + w(u, v)
    then d[v] ← d[u] + w(u, v)
         π[v] ← u
```

目前 v 最好答案是由 u 走過來



(a) decrease (b) no change

24.1 The Bellman-Ford algorithm

no negative cycles, can detect

* Weights can be negative. If there is a negative cycle reachable from s, it returns FALSE.

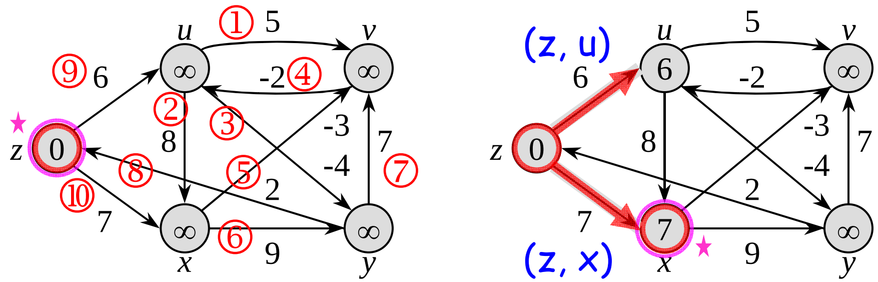
BELLMAN-FORD(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)                V
for i ← 1 to |V[G]| − 1
    do for each edge (u, v) ∈ E[G]            (V-1) × E
        do RELAX(u, v, w)  → O(1)
for each edge (u, v) ∈ E[G]
    do if d[v] > d[u] + w(u, v)               E
        then return FALSE                     → no solution
return TRUE
       computed
```

check negative cycles

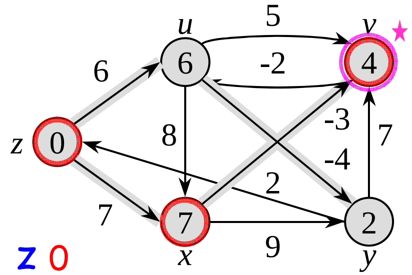
* $O(VE)$ time

Example: (Each phase relaxes edges in lexicographic order: (u,v), (u,x), (u,y), (v,u), (x,v), (x,y), (y,v), (y,z), (z,u), (z,x). Shaded edges indicate π.)

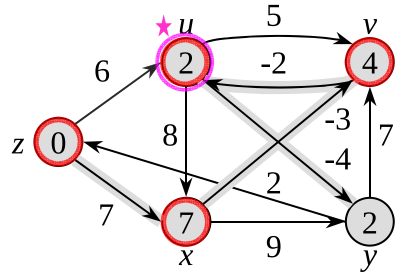


(a) (b)

i = 1



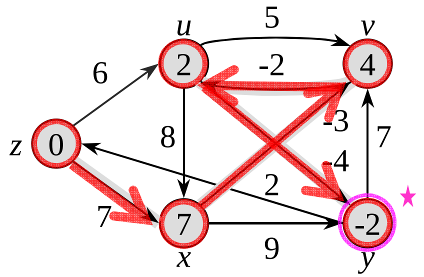
(c) i = 2



(d) i = 3

z 0
7
x 7
-3
v 4
-2
u 2
-4
y -2

Shortest-Path tree



(e) i = 4 (V - 1 times)

Lemma 24.2: If G contains no negative-weight cycles reachable from s. Then, Bellman-Ford algorithm computes $d[v]=\delta(s, v)$ for all vertices v that are reachable from s. 24-5a

Proof: By induction, we can easily show that if the shortest path from s to v contains i edges, $d[v]$ will be correctly computed after phase i. Since a simple path contains at most V-1 edges, The lemma holds.

24-app

Theorem 24.4: Bellman-Ford algorithm is correct. (See textbook for its proof.)

* speedup ??? 24-5x

24.2 Single-source shortest path in DAGs

24-6a

(allow negative edges)

DAG-SHORTEST-PATHS(G, w, s)

topologically sort the vertices of G E + V

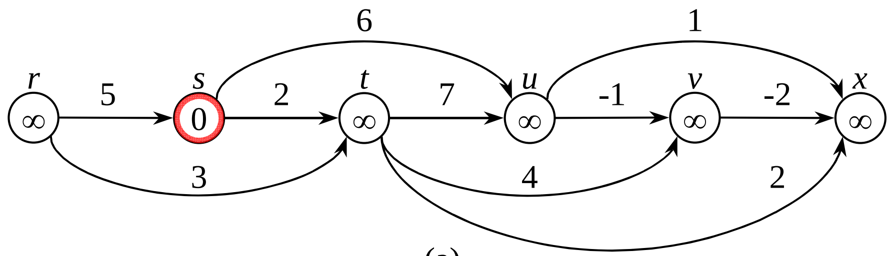
INITIALIZE-SINGLE-SOURCE(G, s) V

for each vertex u, taken in topologically sorted order

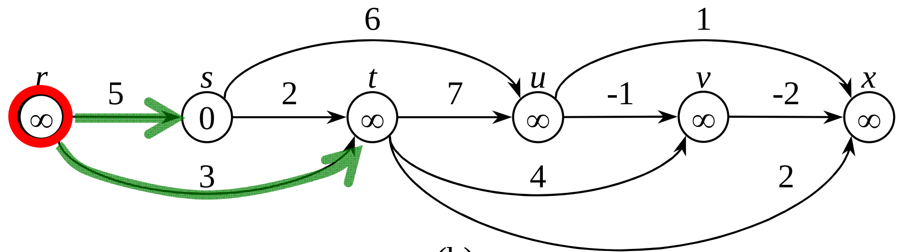
O(E)   **do for** each vertex $v \in Adj[u]$

do RELAX(u, v, w)

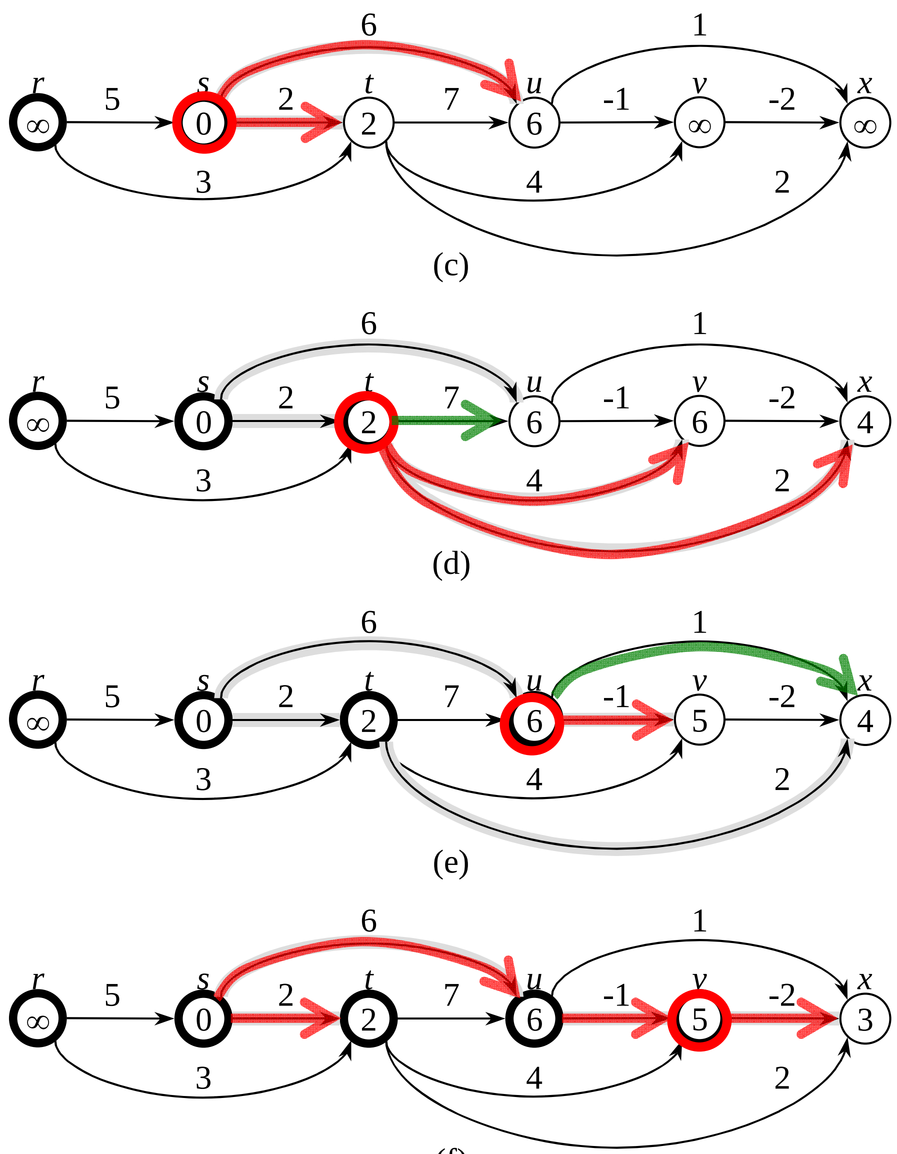
⇨ Time: O(E + V)



(a)



(b)



(c) (d) (e) (f)

* Correctness: By Induction

24-7a

simple

***The critical path problem*:** Finding a longest path through a DAG. (NP-hard for general graph)

(1) Since a DAG contains no cycles, this problem can be solved by negating the weights and then running DAG-SHORTEST-PATHS.

24-8x

(2) DP: $d(v) = \max_{(u,v) \in E} \{d(u) + w(u, v)\}$

24-10r

24.3 *Dijkstra's algorithm*: why?

(All weights should be nonnegative.)

24-8a

24-8b

It maintains a set *S* of vertices *v* whose $\delta(s, v)$ is computed. And it repeatedly selects a vertex $u \in V-S$ with minimum $d[u]$, inserts it into *S*, and relaxes all edges leaving *u*.

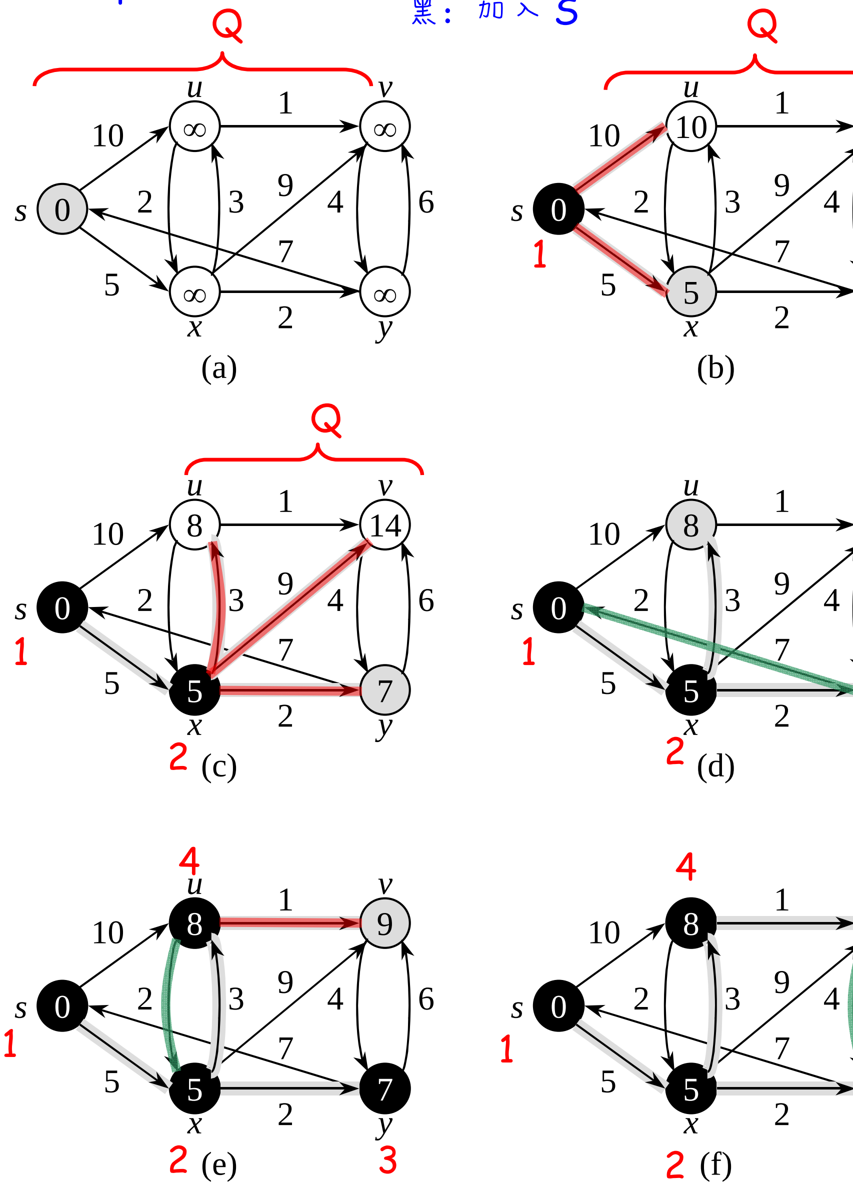
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S ← Ø                          /* computed vertices */
Q ← V[G]                       /* using d[v] as keys */
while Q ≠ Ø
    do u ← EXTRACT-MIN(Q)      V times
       S ← S ∪ {u}
       for each vertex v ∈ Adj[u]
           do RELAX(u, v, w)   2E times (E for directed)
```

build priority queue Q

☆ decrease-key

S: computed vertices

灰: 選中
黑: 加入 S



Time Complexity:

* Similar to Prim's MST algorithm. 24-10x

(a) Implement priority queue Q as an array 24-10z

unsorted

Steps 1~3: $O(V)$ (Build Q)

Step 5: $O(V^2)$ (V times Extract-Min) O(V)

Step 6: $O(V)$

Steps 7~8: $O(E)$ (2E times Decrease-key) O(1)

Total: $O(V^2+E) = O(V^2)$ (for dense G)

($E \approx V^2$) ~simple

(b) Implement Q as a binary heap

Step 5: $O(V\lg V)$ (V times Extract-Min) lg V

Steps 7~8: $O(E\lg V)$ (2E times Decrease-key) lg V

Total: $O(E\lg V)$ (for sparse G)

($E \ll V^2$)

(c) Implement Q as a Fibonacci heap

Step 5: $O(V\lg V)$ (V times Extract-Min) lg V

Steps 7~8: $O(E)$ (2E times Decrease-key) O(1)

Total: $O(E+V\lg V)$ (for sparse G)

($E \ll V^2$)

Homework: Ex. 24.1-4, 24.2-1, 24.3-2, Pro. 24-2, 24-3, 24-5 (mean-weight cycle).