Graphs II - Shortest paths

Single Source Shortest Paths

All Sources Shortest Paths

some drawings and notes from prof. Tom Cormen

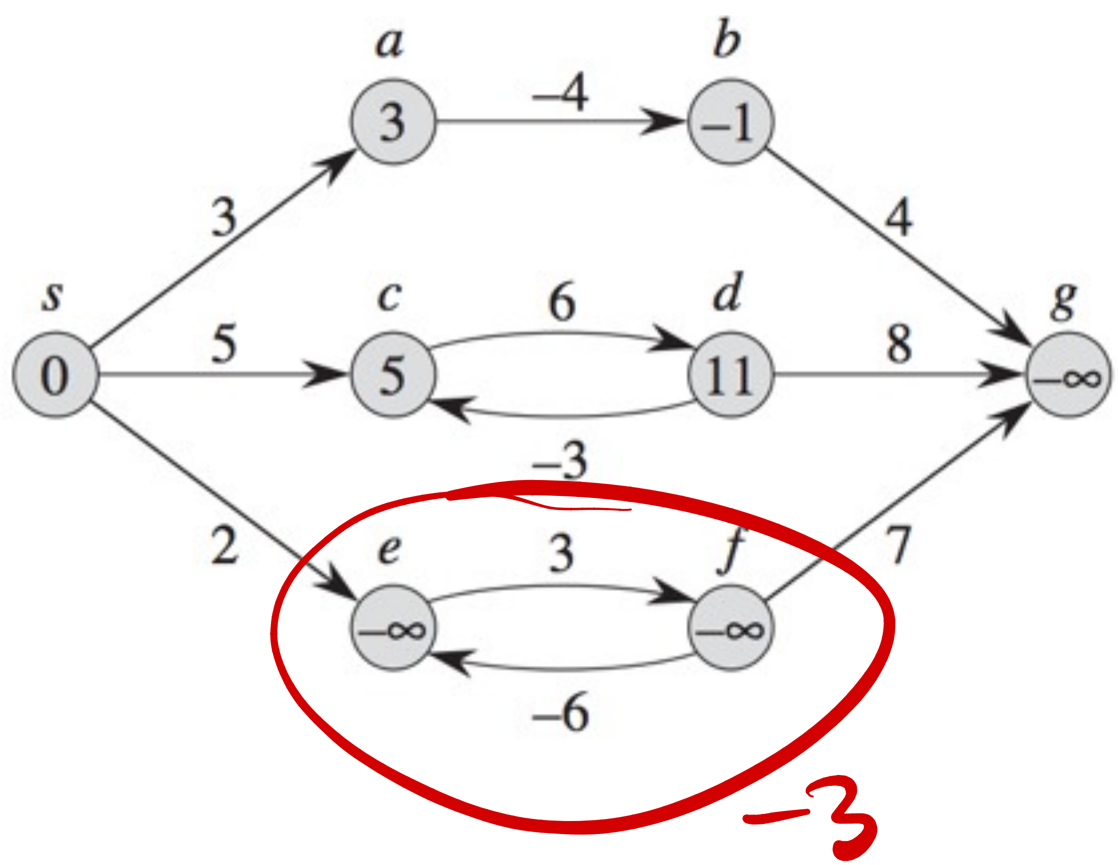
Single Source SP

- Context: directed graph G=(V,E,w), weighted edges
- The shortest path (SP) between vertices u and v is the path that has minimum total weight
  - total weight is obtained by summing up path's edges weights

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise .} \end{cases}$$

- Note: SP cannot contain cycles
  - positive cycles: a shortest path obtained by taking out the cycle
  - negative cycles: a shortest path obtained by iterating through the cycle few more times, minimum weight is $-\infty$.

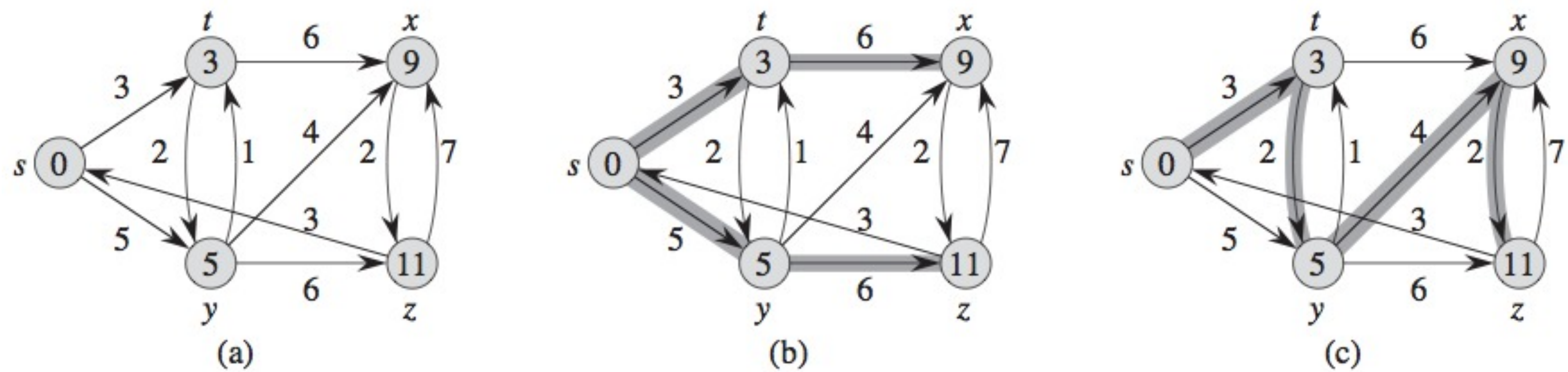
Negative edges and cycles



- negative weights possible
- negative cycles make some shortest paths $-\infty$

- Exercise: explain the following :
- SP(s,a)=3
- SP(s,b)= -1
- SP(s,g)=3
- SP(s,e)=$-\infty$

Single Source SP



- Task: Given a source vertex s∈V, find the shortest path from s to all other vertices
  - will write inside each vertex v the shortest path estimate ESP(s,v) weight from the source
    - these estimates change as the algorithm progresses
  - highlight edges that give the SP-s
  - highlighted edges form a tree with source as root
    - tree not unique as (b) and (c) are both valid

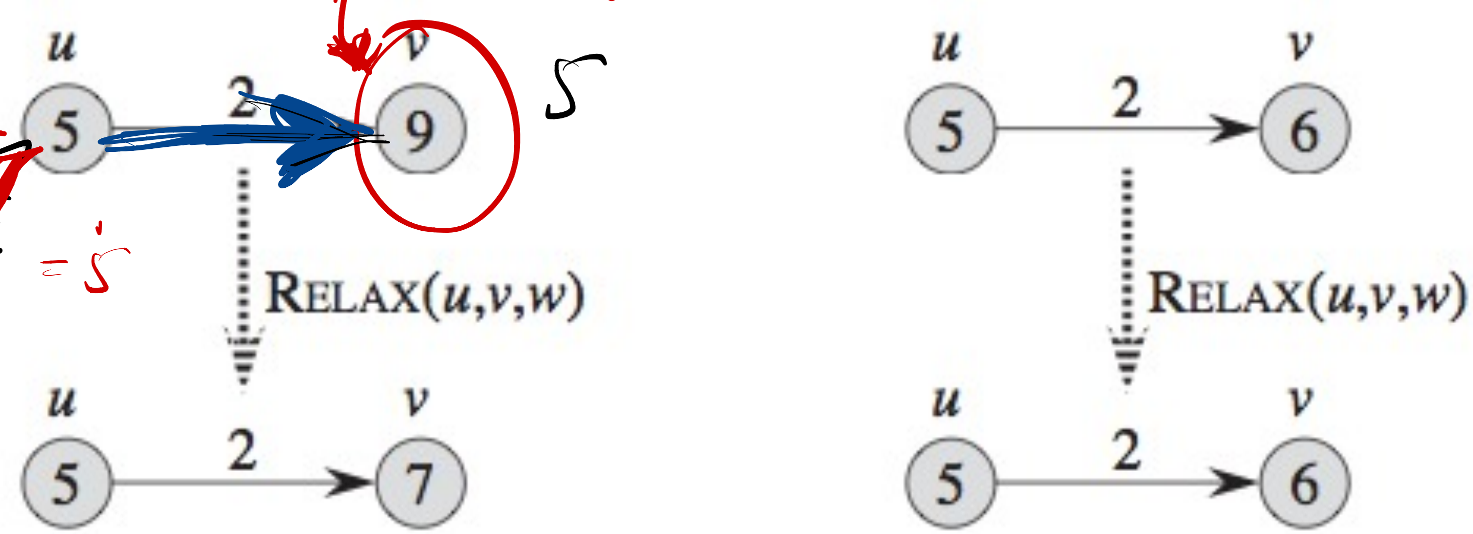
Relaxation

estimated shortest path (so far)

- if current (estimate) ESP(s,u) is 5 and edge (u,v) has weight w(u,v)=2, we can reach v with a path of 5+2=7
  - if current estimate ESP(s,v) is more than 7, we "relax edge (u,v)" by replacing the estimate ESP(s,v) =7.
  - if not (ESP(s,v)≤7), we do nothing

S=source

ESP=9



(a) (b)

neither path s⇝u, path s⇝v are necessarily final.

Th A shortest path $s \to u_1 \to u_2 \cdots u_k \to v$

is discovered $(ESP(v) = \text{final (min value)})$

if the edges of the path
$s \to u_1$
$u_1 \to u_2$
$\cdots$
$u_k \to v$
are

relaxed in this order

version

$SP(u_t) + w = SP(u_{t+1})$

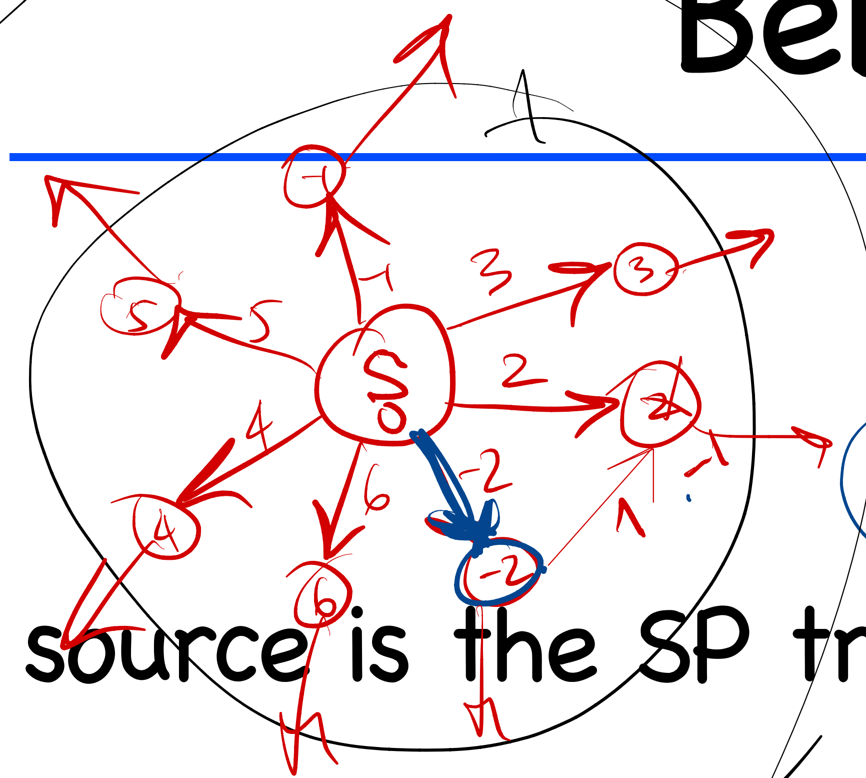
$s \to u_1 \to u_2 \to \cdots \to u_t \xrightarrow{w} u_{t+1}$

if $ESP(s, u_t)$ is already the SP (discovered)

when relaxing the next edge in the true SP ~~to~~ u_{t+1}

$\Rightarrow ESP(u_{t+1})$ is final

Bellman Ford



1) find $SP(s,u)$ $\forall u$ of exact 1 edge
relax all edges $(s \to u) \Rightarrow ESP$

$\exists u$ $SP(s \to u)$ is one edge $\Longrightarrow$ I found it

2) find $SP(s,u)$ with 1 or 2 edges
– relax ALL edges

- source is the SP tree root
- BF algorithm progresses in "waves", similar to BFS
- takes a maximum of |V|-1 waves to find SP
  - since there cannot be cycles

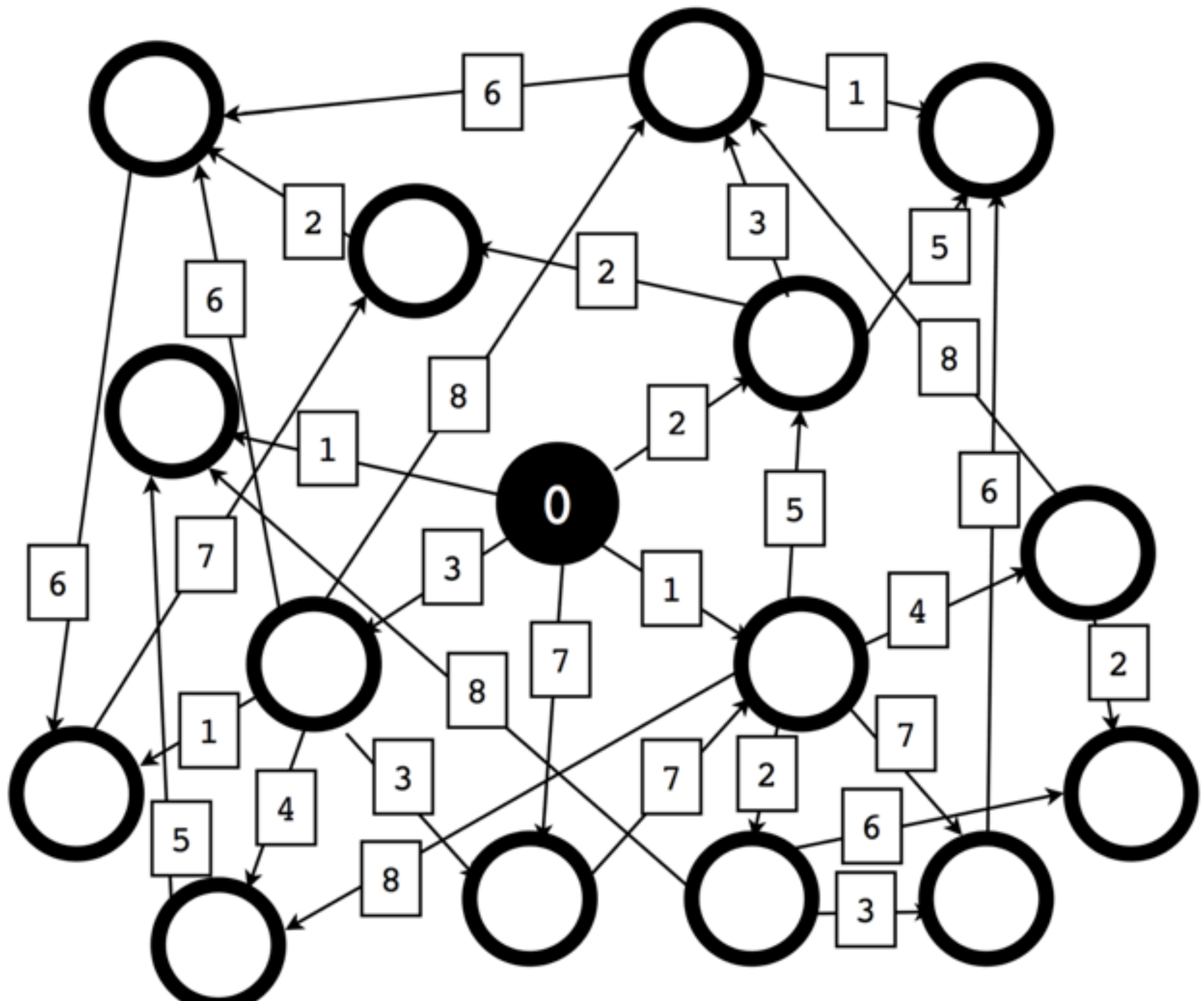
Bellman-Ford SSSP algorithm

- idea : relax all edges once (in any order) and we've got CORRECT all SP-s of one edge
  - relax again all edges (any order) and we obtained all SP-s of two edges
  - relax .... again, and get all SP-s of three edges
  - no SP can have more than |V|-1 edges, so repeat the relax-all-edges step |V|-1 times, to get all SP-s

```
BELLMAN-FORD
    init all SP : SP(s,v)= ∞ for all v, SP(s,s)=0
  for k=1:|V|-1
    relax all edges
  check for negative cycles
```

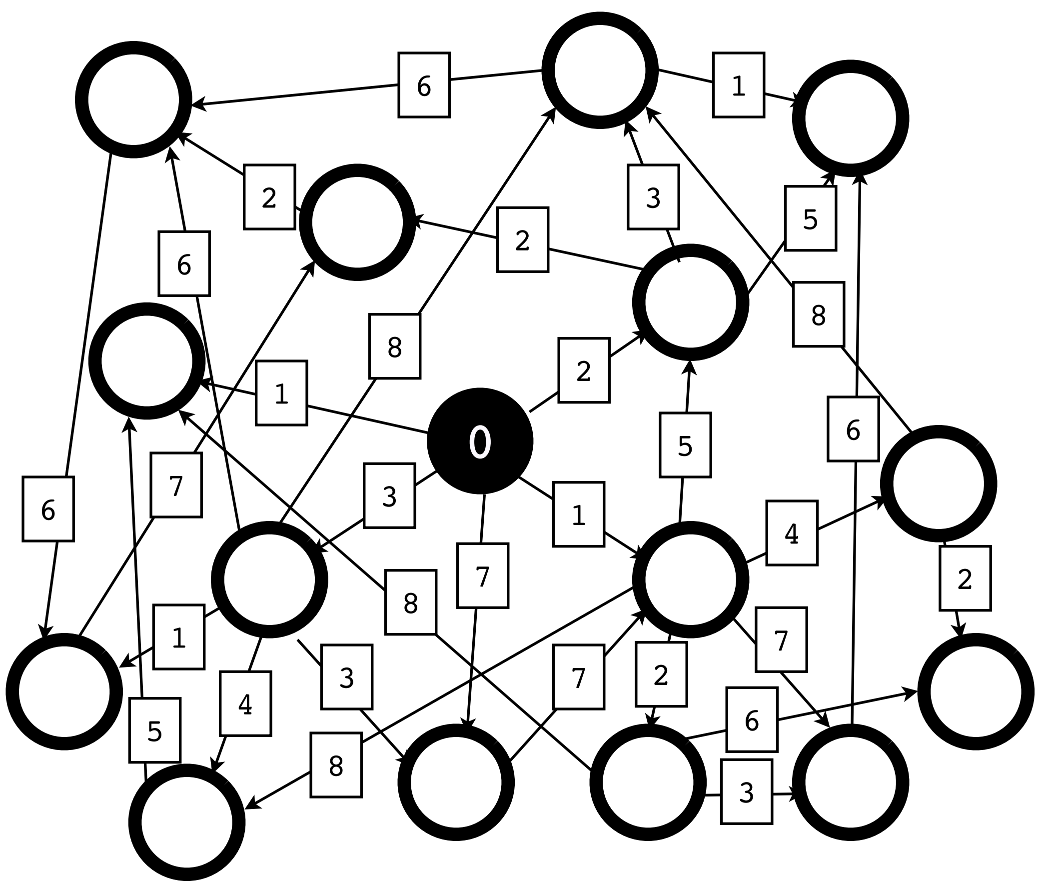
SSSP exercise

- Discover SP by hand (start from source)



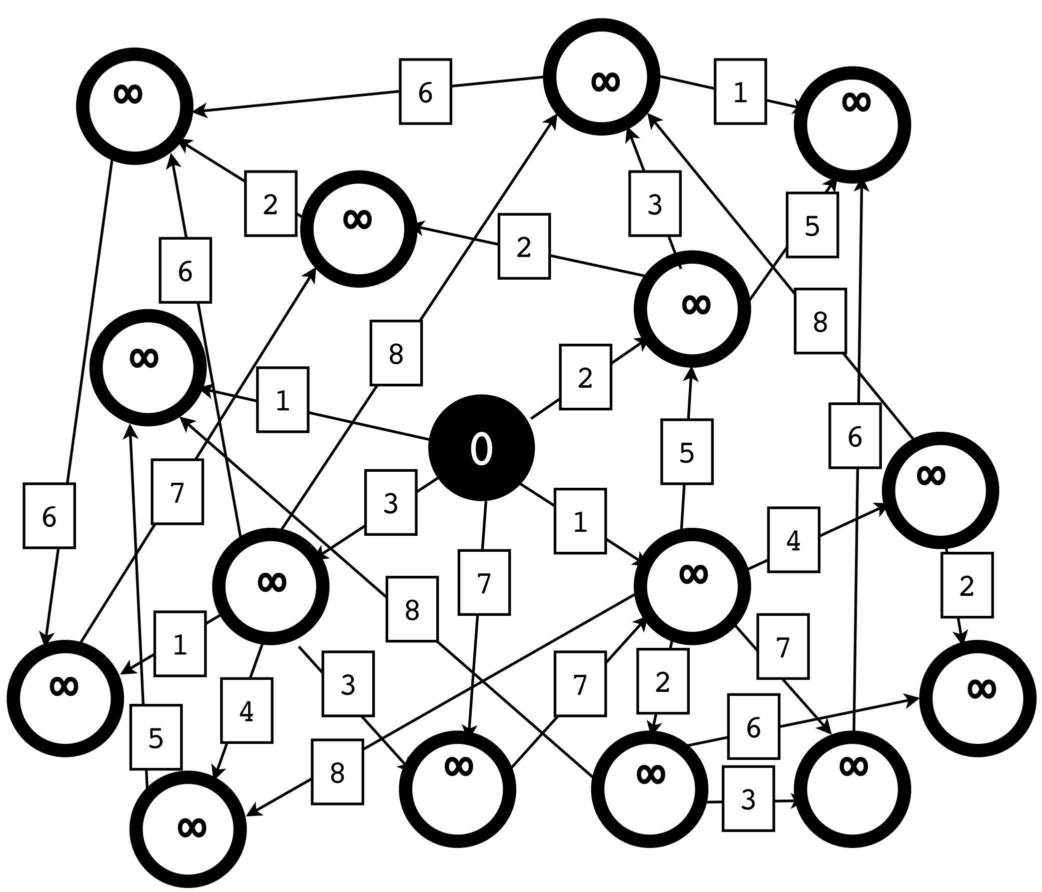
Bellman Ford

- discover SP(s,v) means having the current estimate equal with the actual (unknown) SP
  - discover SP : ESP(s,v) = SP(s,v)
  - ESP written "inside" each node, it may further decrease
  - once SP discovered, the ESP never decreases



Bellman Ford

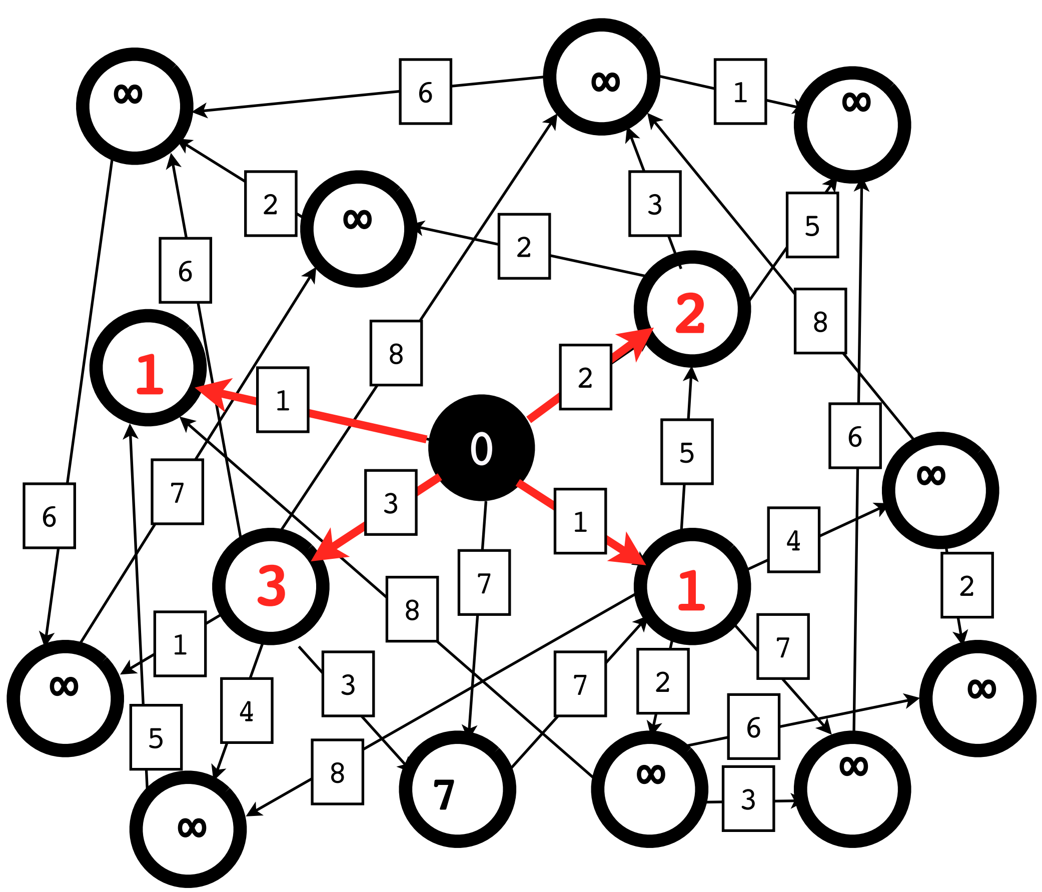
- discover SP(s,v) means having the current estimate equal with the actual (unknown) SP
  - discover SP : ESP(s,v) = SP(s,v)
  - ESP written "inside" each node, it may further decrease
  - once SP discovered, the ESP never decreases



- init all ESP = ∞

Bellman Ford

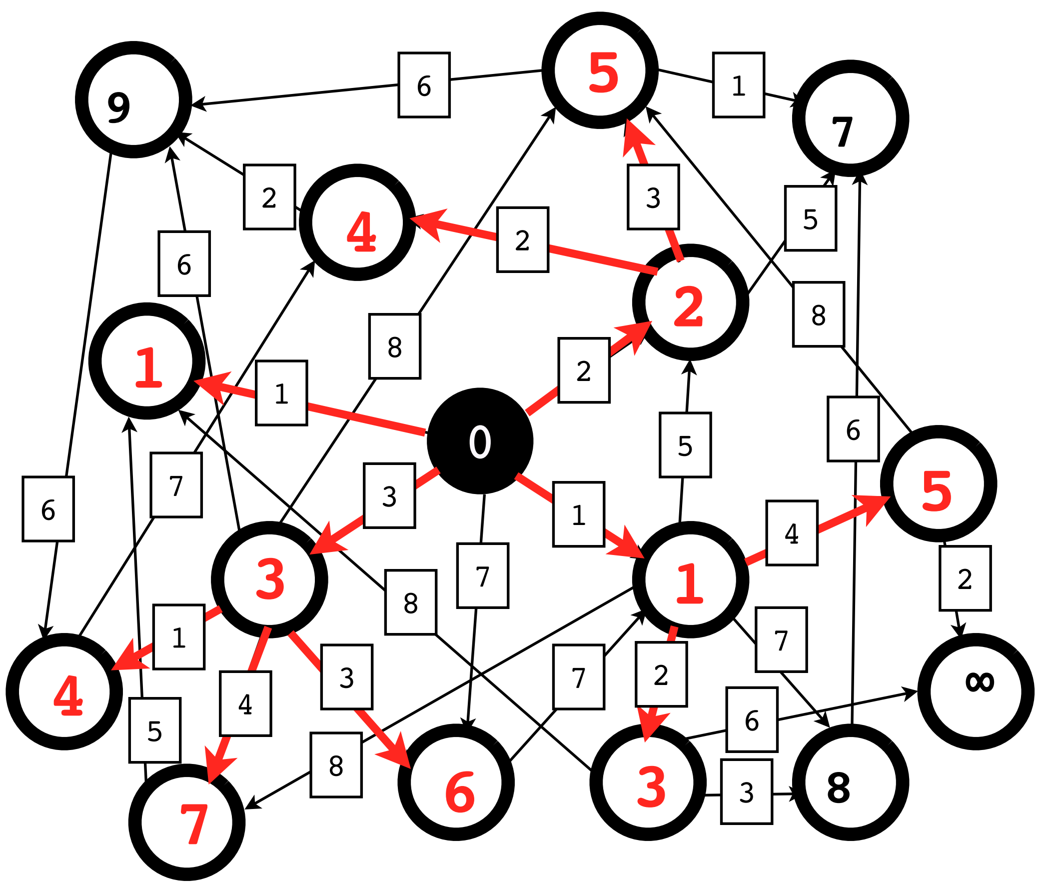
- discover SP(s,v) means having the current estimate equal with the actual (unknown) SP
  - discover SP : ESP(s,v) = SP(s,v)
  - ESP written "inside" each node, it may further decrease
  - once SP discovered, the ESP never decreases



- init all ESP = ∞
- relax all edges (first time): discover all SP-s of one edge

Bellman Ford

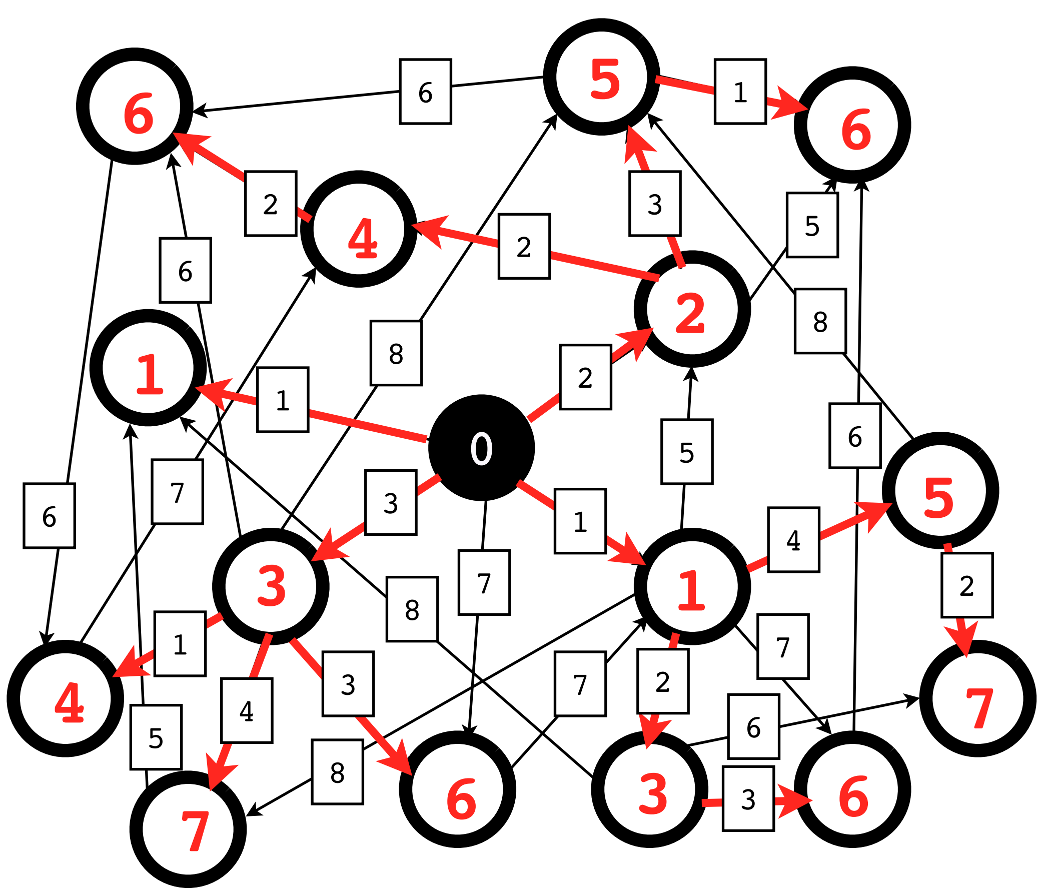
- discover SP(s,v) means having the current estimate equal with the actual (unknown) SP
  - discover SP : ESP(s,v) = SP(s,v)
  - ESP written "inside" each node, it may further decrease
  - once SP discovered, the ESP never decreases



- init all ESP = ∞
- relax all edges (first time): discover all SP-s of one edge
- relax all edges (second time): discover all SP-s of two edges

Bellman Ford

- discover SP(s,v) means having the current estimate equal with the actual (unknown) SP
  - discover SP : ESP(s,v) = SP(s,v)
  - ESP written "inside" each node, it may further decrease
  - once SP discovered, the ESP never decreases



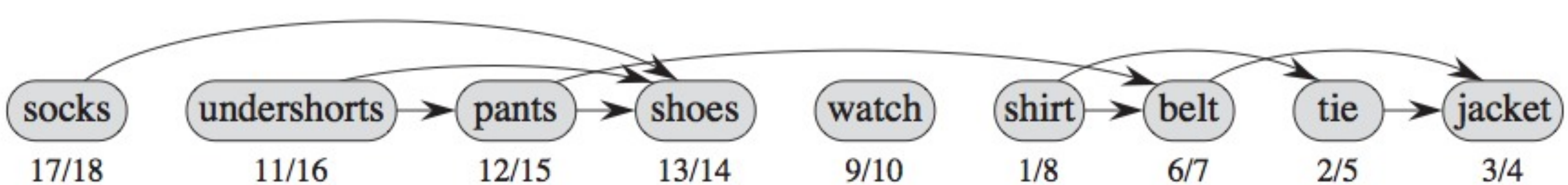
- init all ESP = ∞
- relax all edges (first time): discover all SP-s of one edge
- relax all edges (second time): discover all SP-s of two edges
- . . . repeat
  - how many times?

Bellman Ford

- Essential mechanism (BF proof):
  - SP(s,v) = [a1, a2, a3, a4]
  - Relaxing a1, then a2, then a3, then a4 - you can do them over any amount of time, but it has to be in the right order
    - SP(s,v) discovered
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
  - overall quite a few more relaxations than necessary, in order to enforce correctness in all possible cases
- Running time: |V|-1 iterations for the outer loop
- inner loop: relax all edges O(E)
- Total V*O(E) = O(VE)

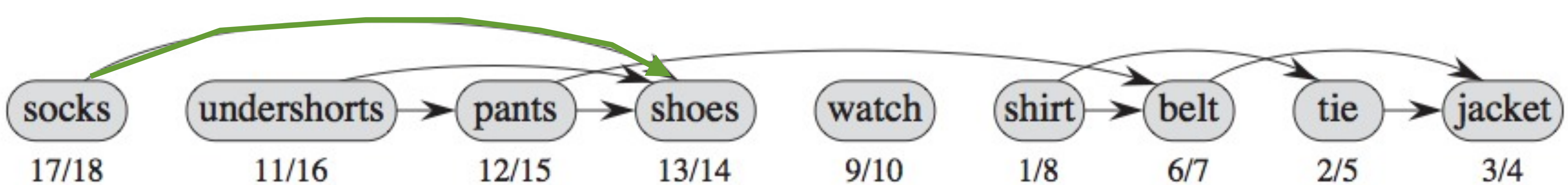
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



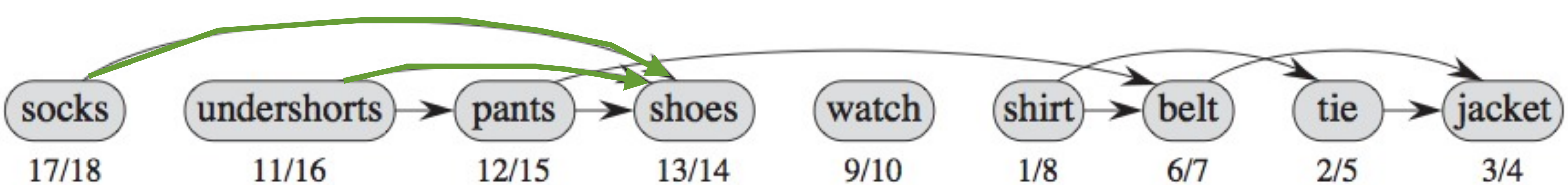
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



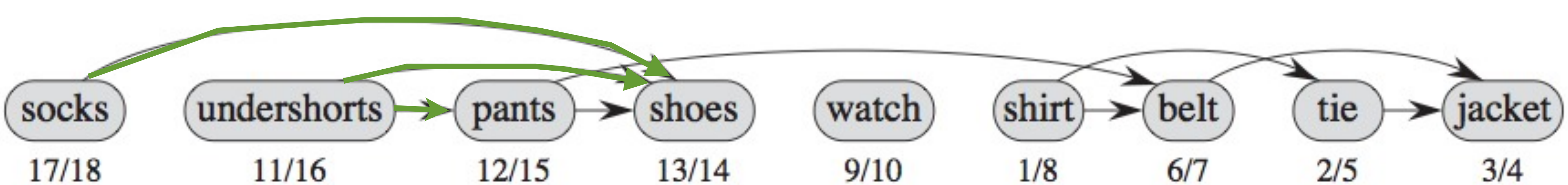
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



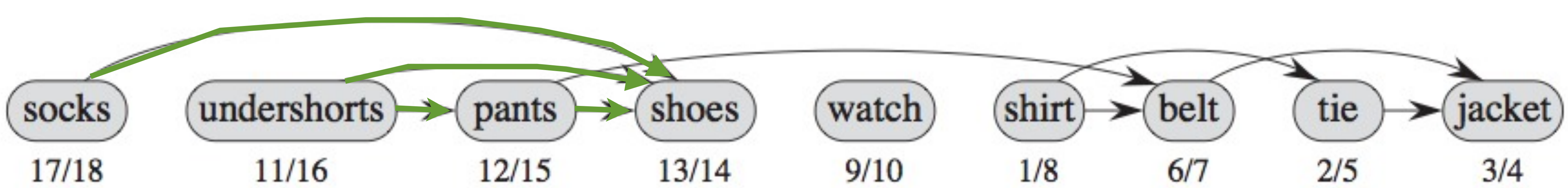
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,…) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



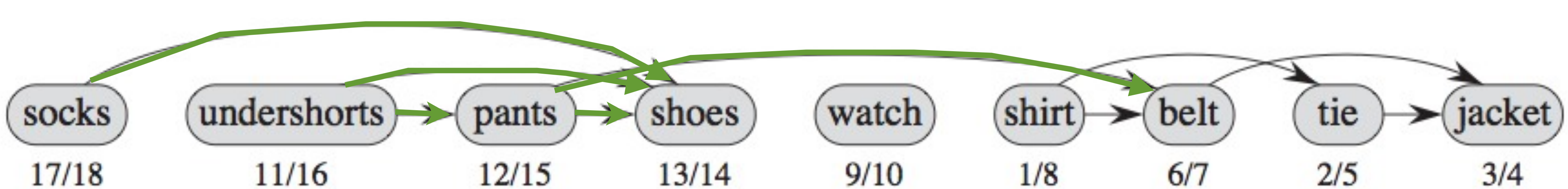
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



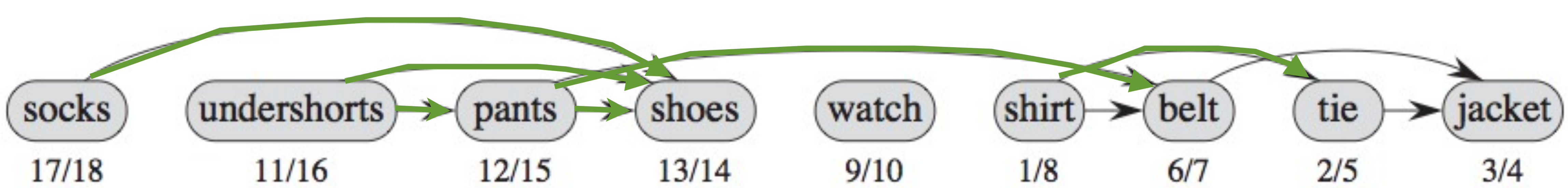
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



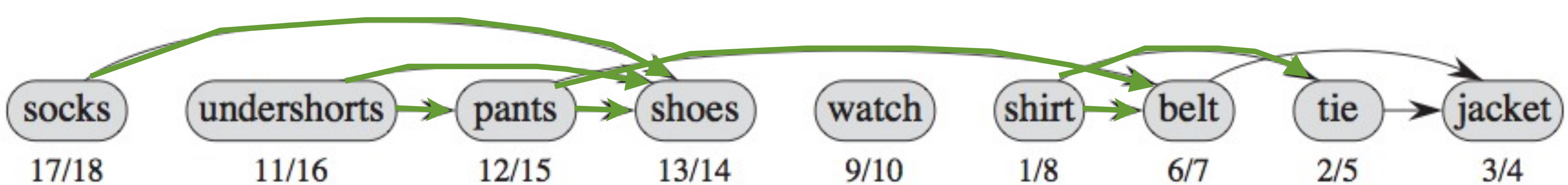
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



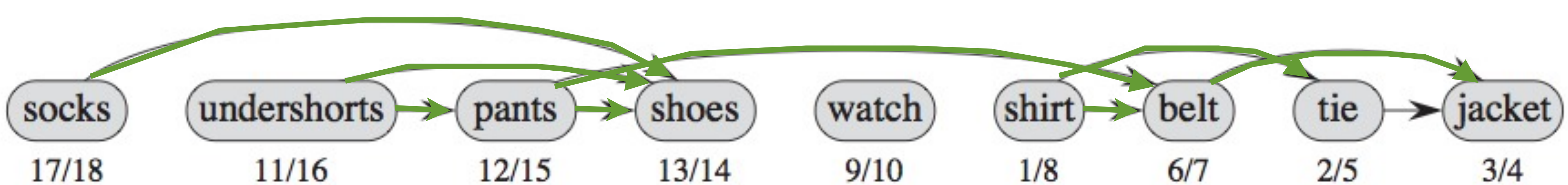
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



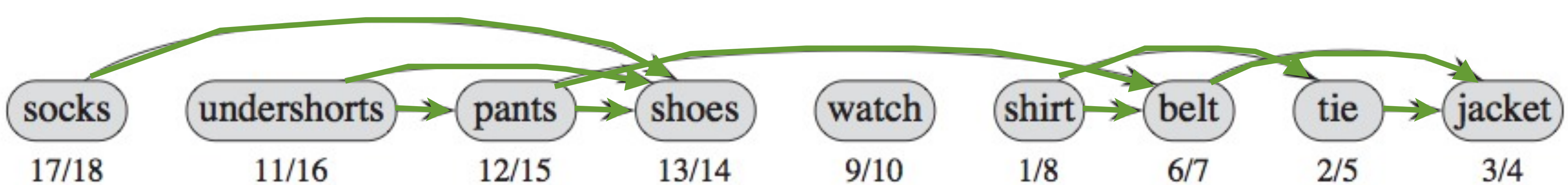
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)



SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.
- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.
  - topological sort is given by finishing DFS times (on picture)
- Running time O(E) (if E>V)
  - formally O(E+V) VS Bellman Ford O(VE)

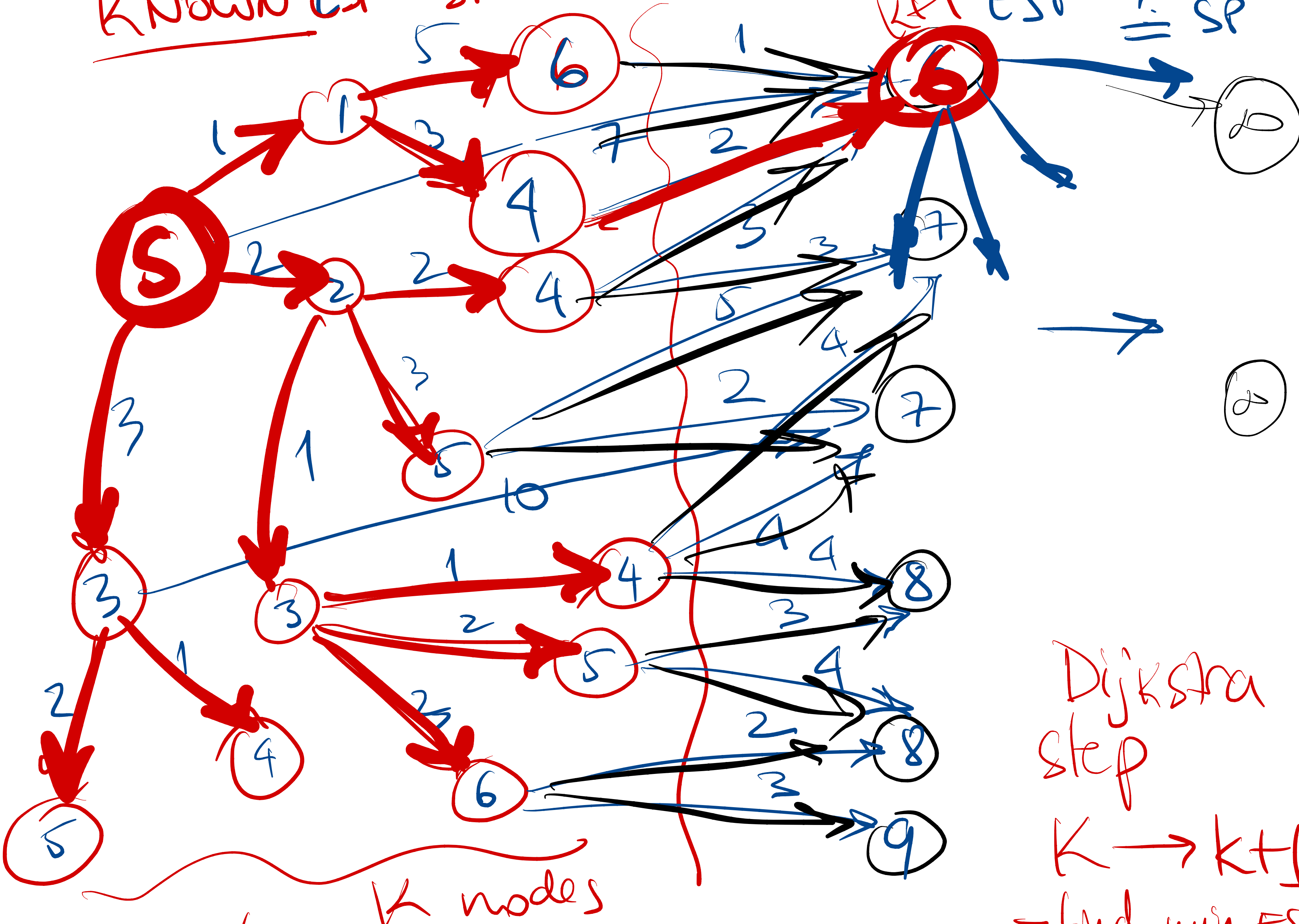


Dijkstra SSSP algorithm

- No negative weight edges allowed
- instead of relaxing all edges (like Bellman Ford), keep track of a current "closest" vertex to the SP tree
  - "closest" = minimum ESP(s,v) of nodes not already part of SP tree
  - add the current-closest to the partial SP tree, v
  - relax the outing edges of v (all edges v->x)
- repeat
- similar to Prim's algorithm (conceptually)

partially a tree
that $\boxed{SP(s,u)}$
$u \in$ tree

Known ESP = SP



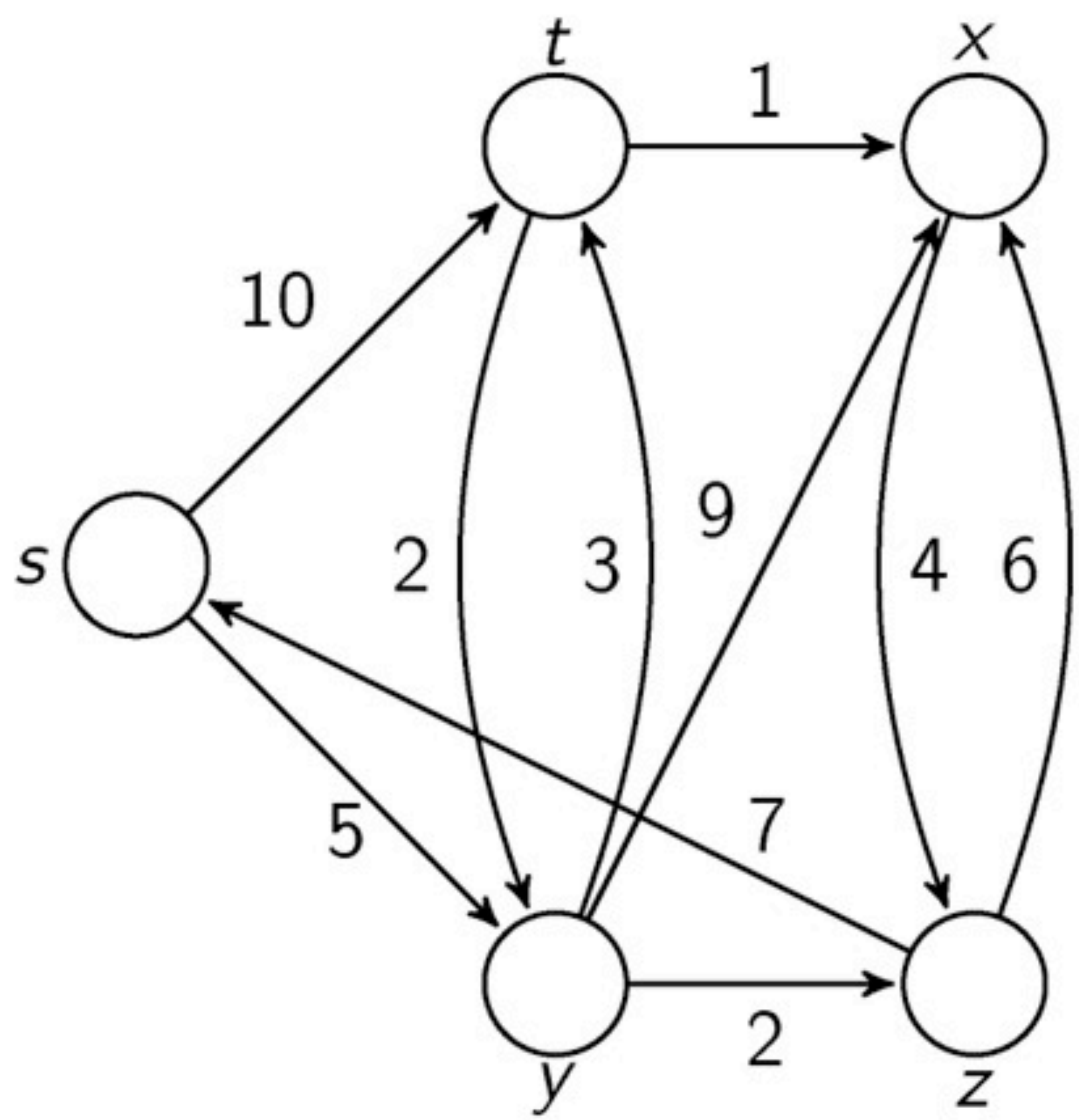
Dijkstra step

K → K+1

- find min ESP
- add to Dij-tree with proper edge
- relax edges from there

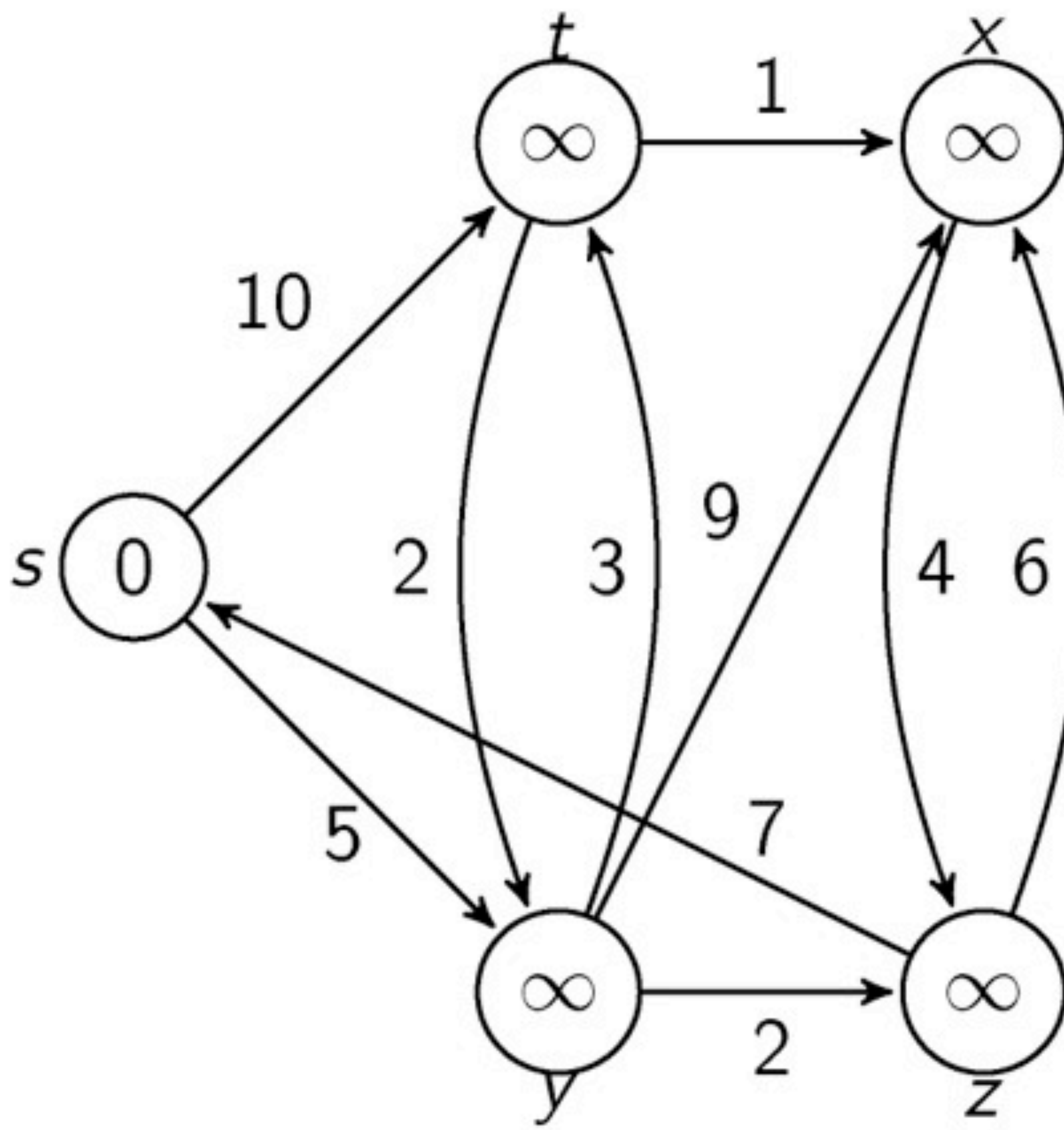
final / true SP

done: all relaxations from red nodes



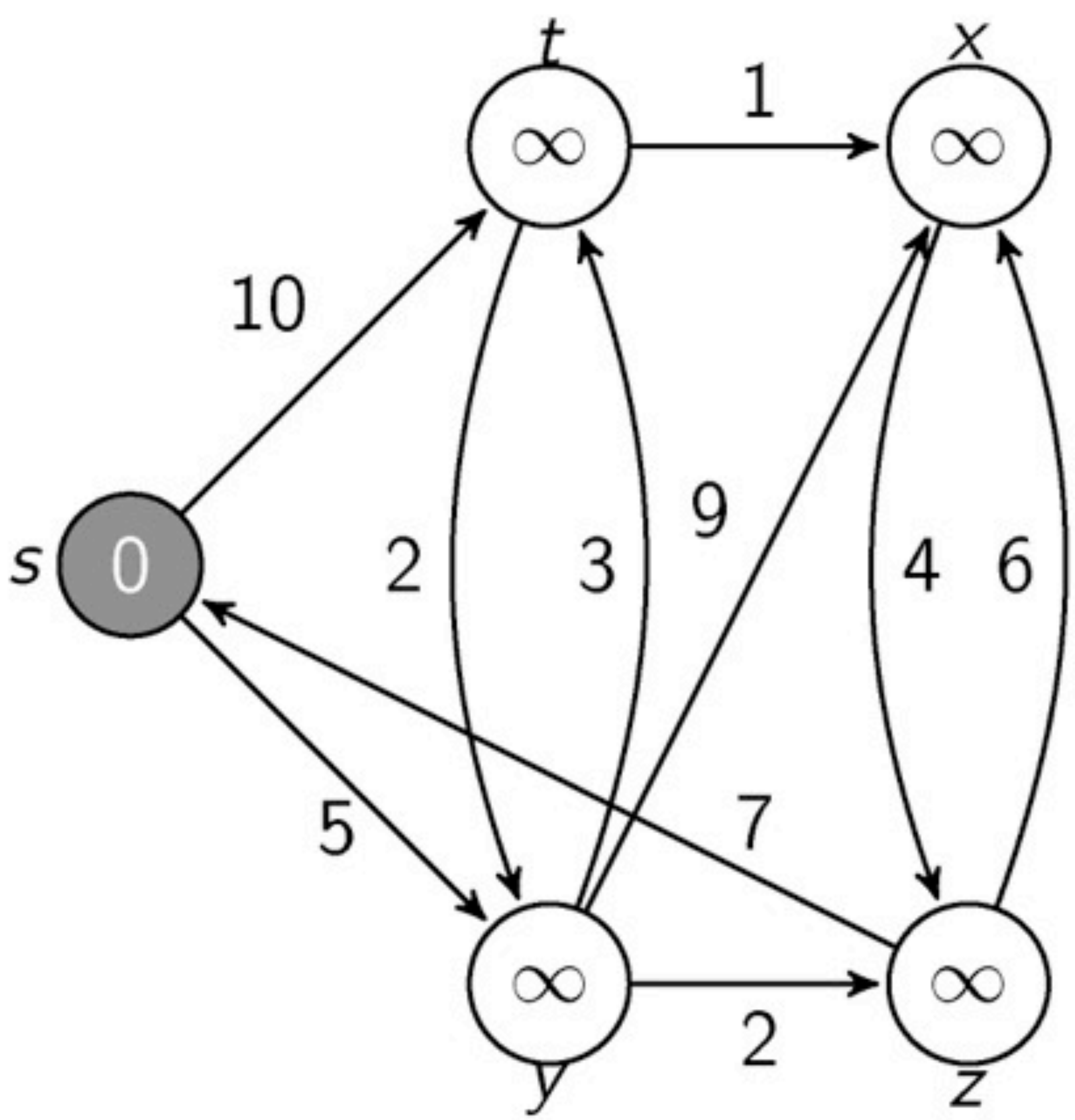
graph G
source = s

We want to find the shortest path from s to every node



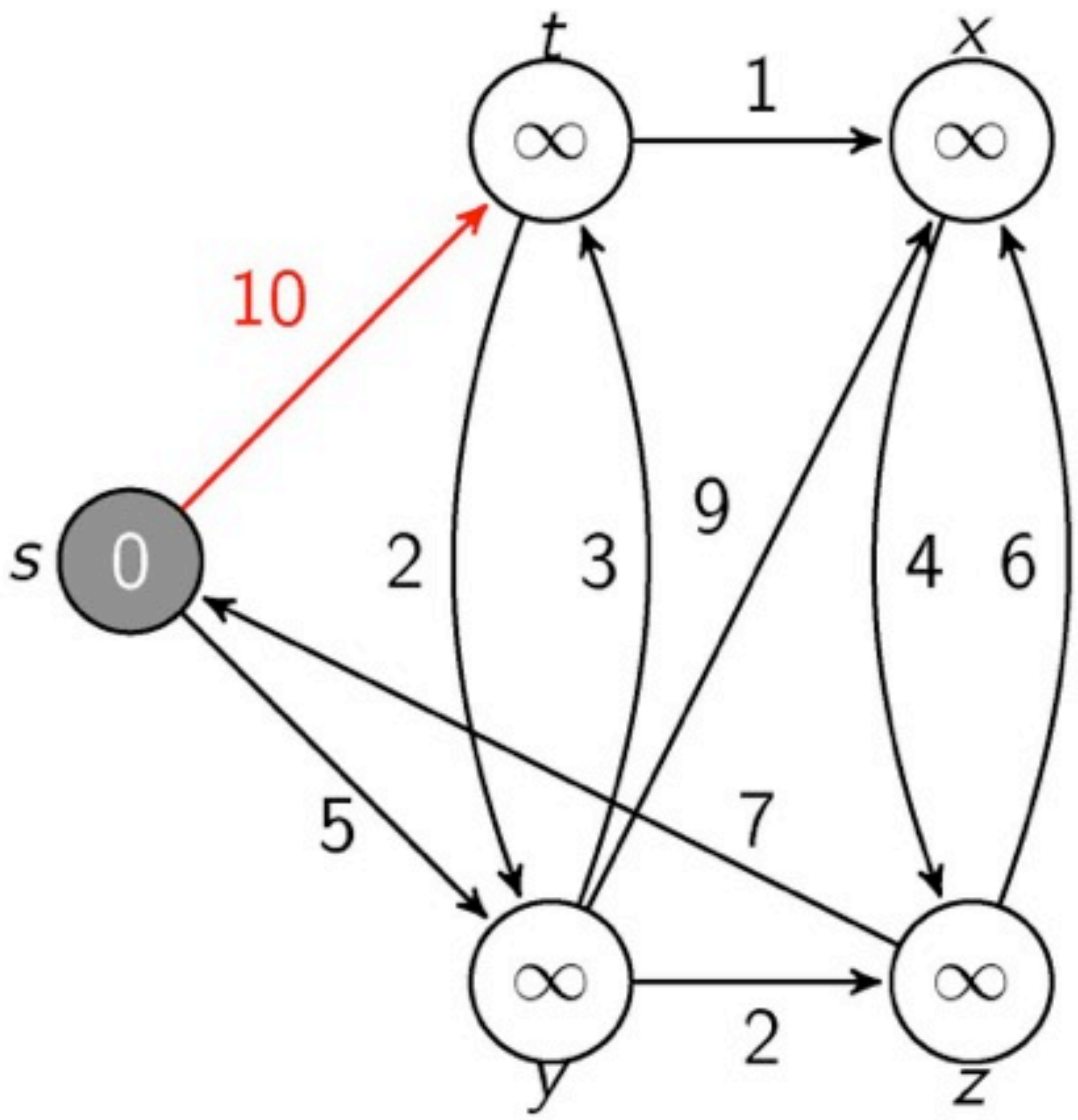
INITIALIZE -SINGLE
-SOURCE(G,s)
$S = \Phi$
$Q = G.V$

After initialization, we have $v.\pi = NIL$ for all $v \in V, s.d = 0$, and $v.d = \infty$ for $v \in V - \{s\}$



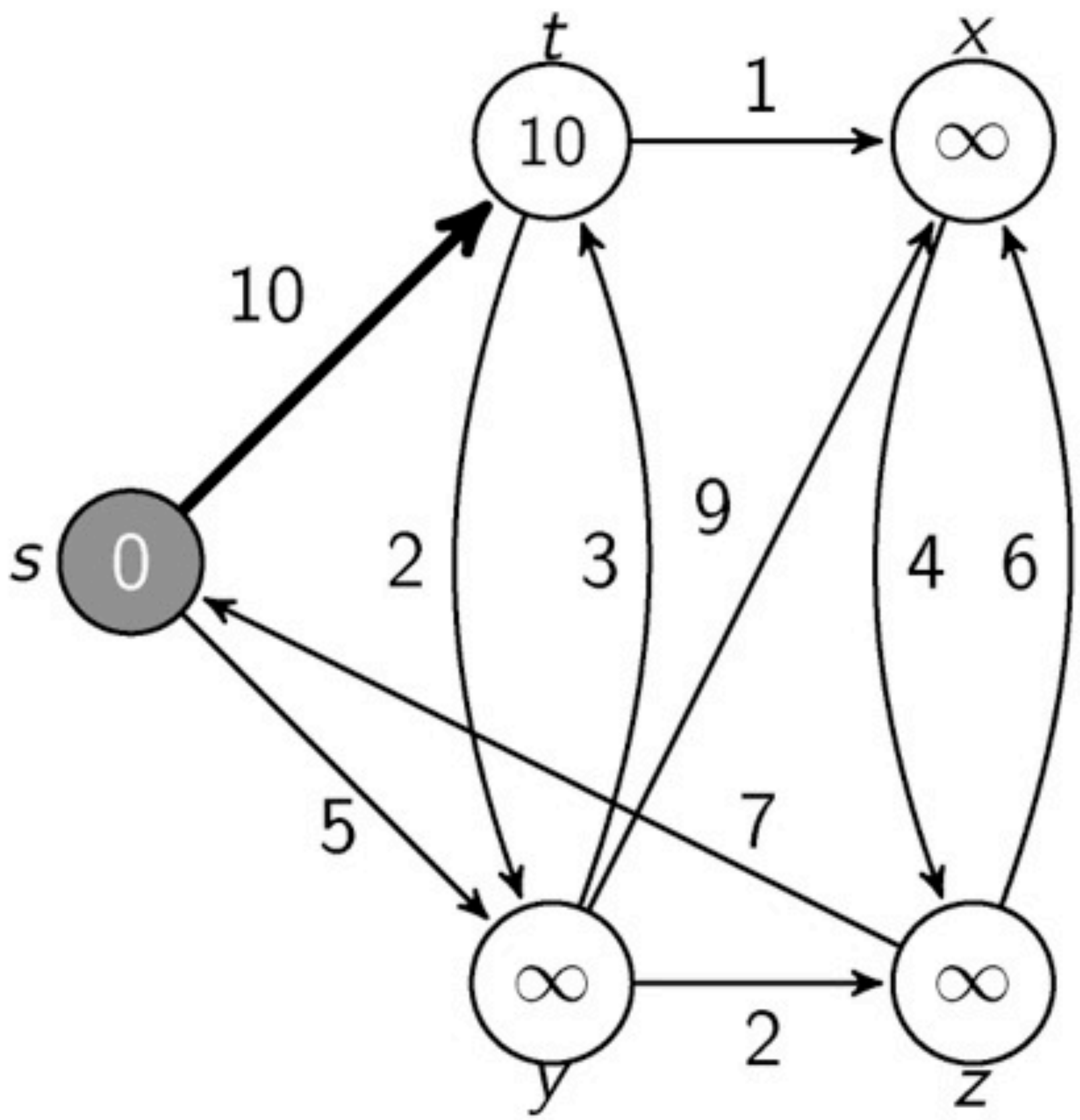
s=EXTRACT-MIN(Q)
$S = \{s\}$
$Q = \{t, x, y, z\}$

We are at node s



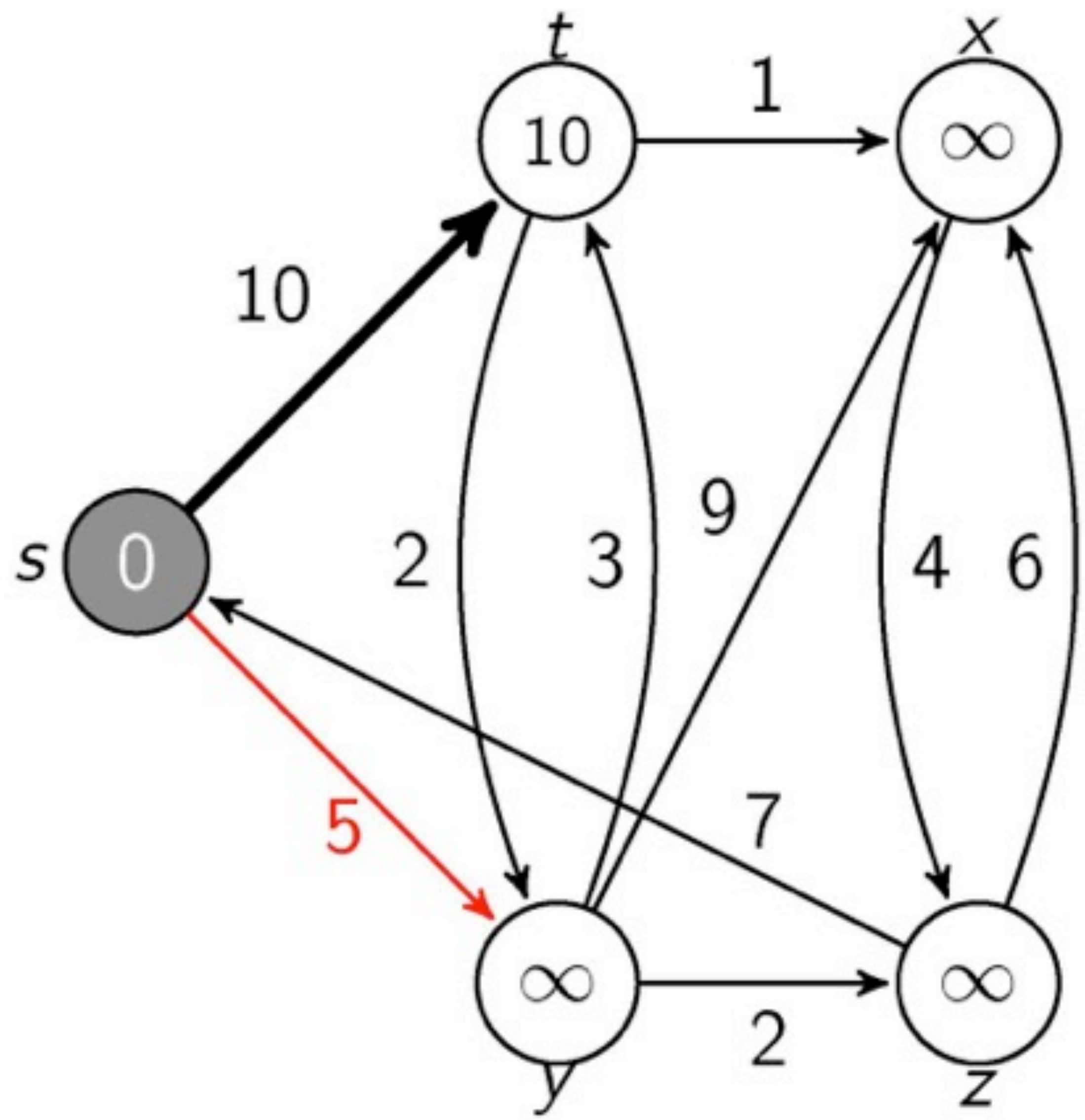
$\text{Relax}(s, t, w)$
$S = \{s\}$
$Q = \{t, x, y, z\}$

Test whether we can improve the shortest path to t found so far by going through s



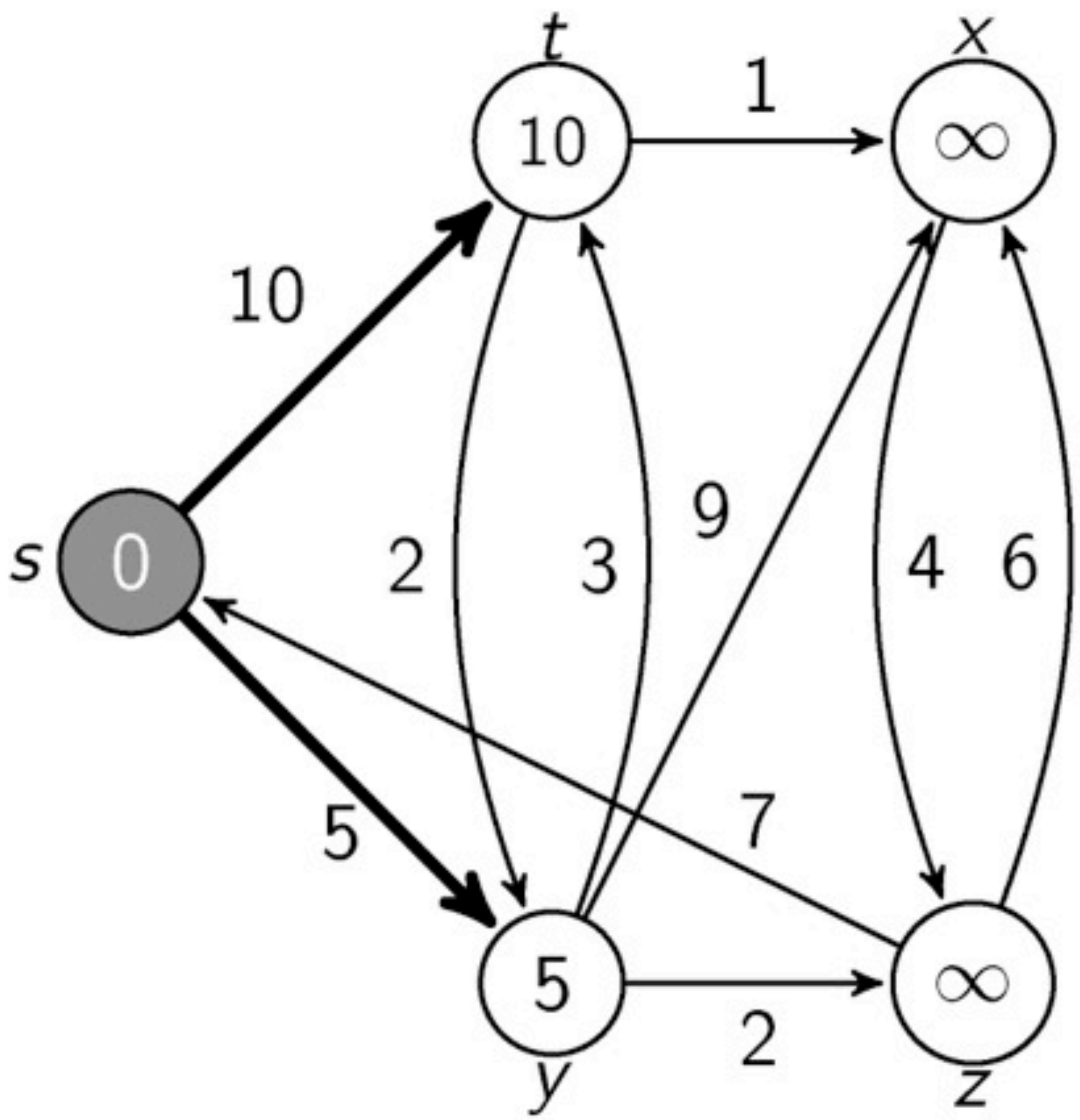
$\textsc{Relax}(s, t, w)$
$S = \{s\}$
$Q = \{t, x, y, z\}$

Update $t.d = 10$ and $t.\pi = s$



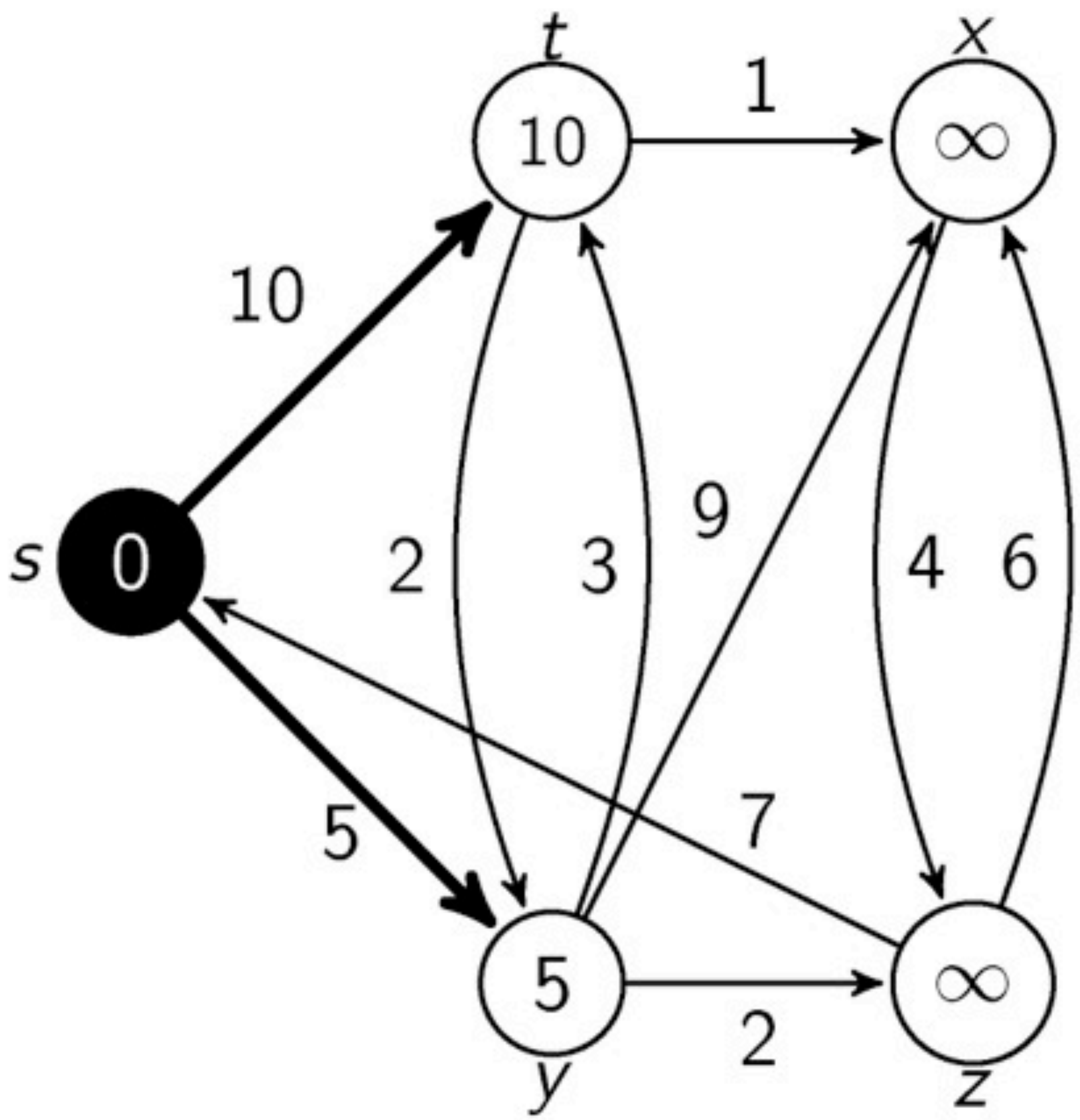
$\text{RELAX}(s, y, w)$
$S = \{s\}$
$Q = \{t, x, y, z\}$

Test whether we can improve the shortest path to y found so far by going through s



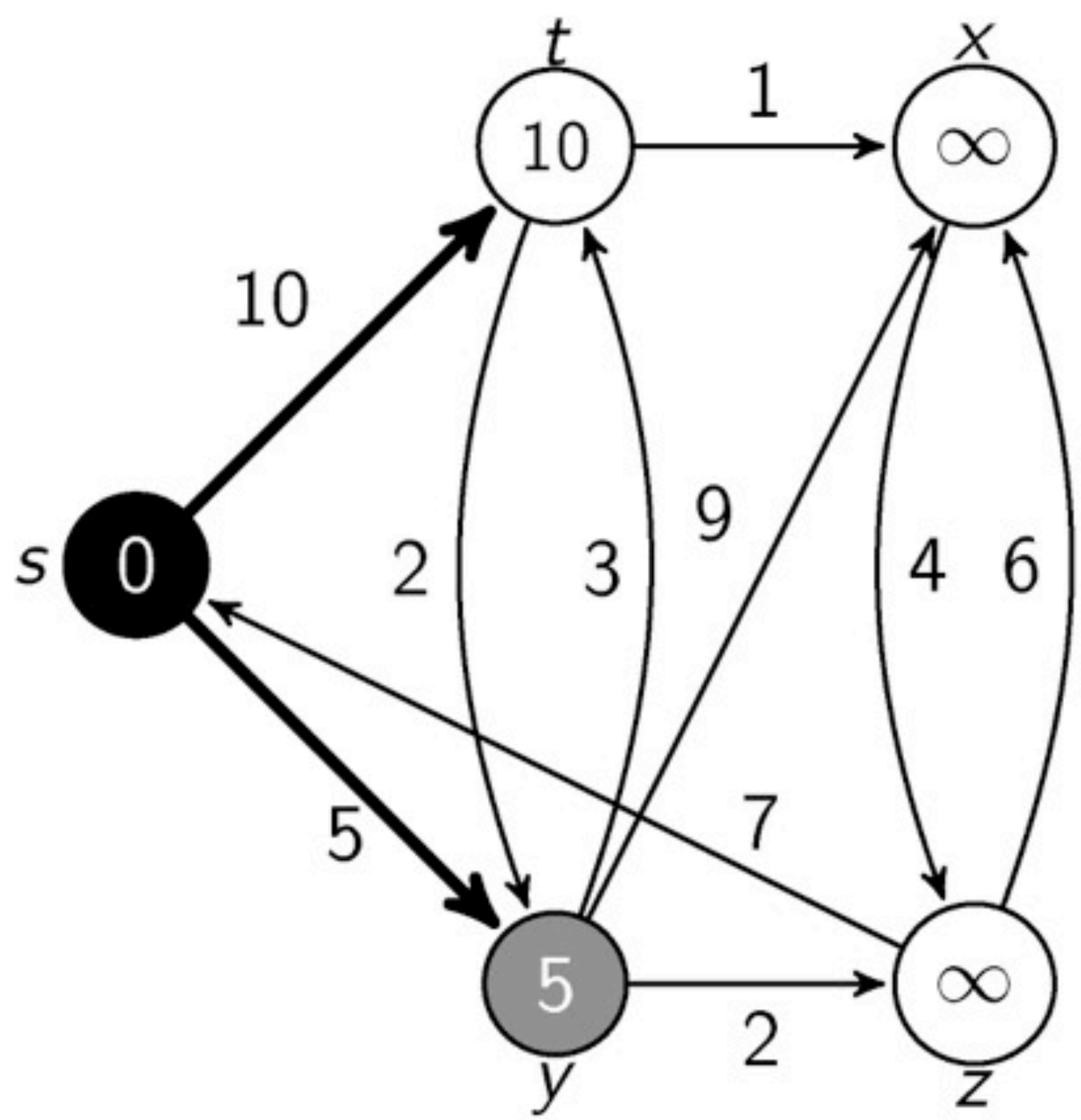
$\text{RELAX}(s, y, w)$
$S = \{s\}$
$Q = \{t, x, y, z\}$

Update $y.d = 5$ and $y.\pi = s$



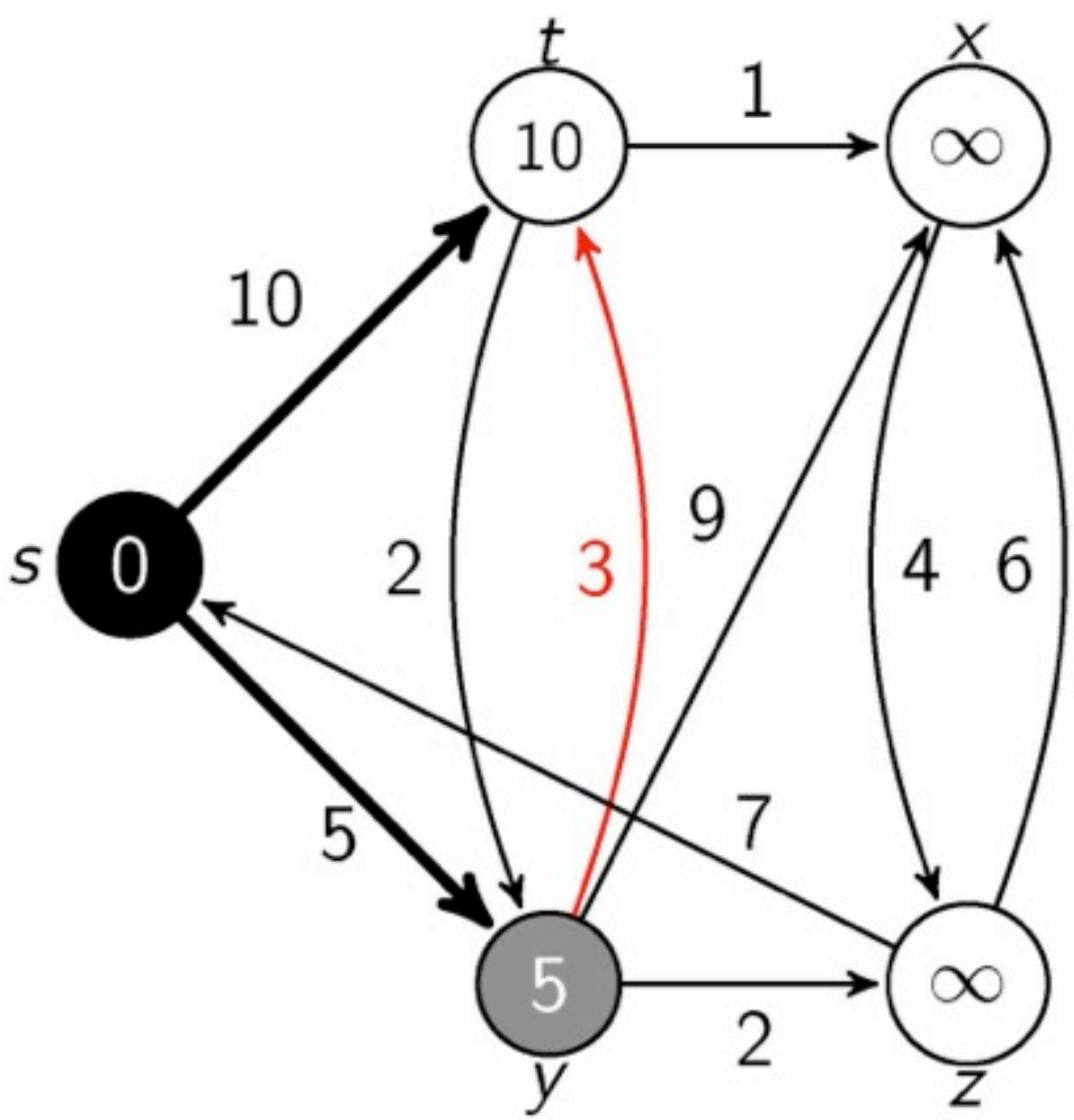
$S = \{s\}$
$Q = \{t, x, y, z\}$

All edges leaving s have been tested



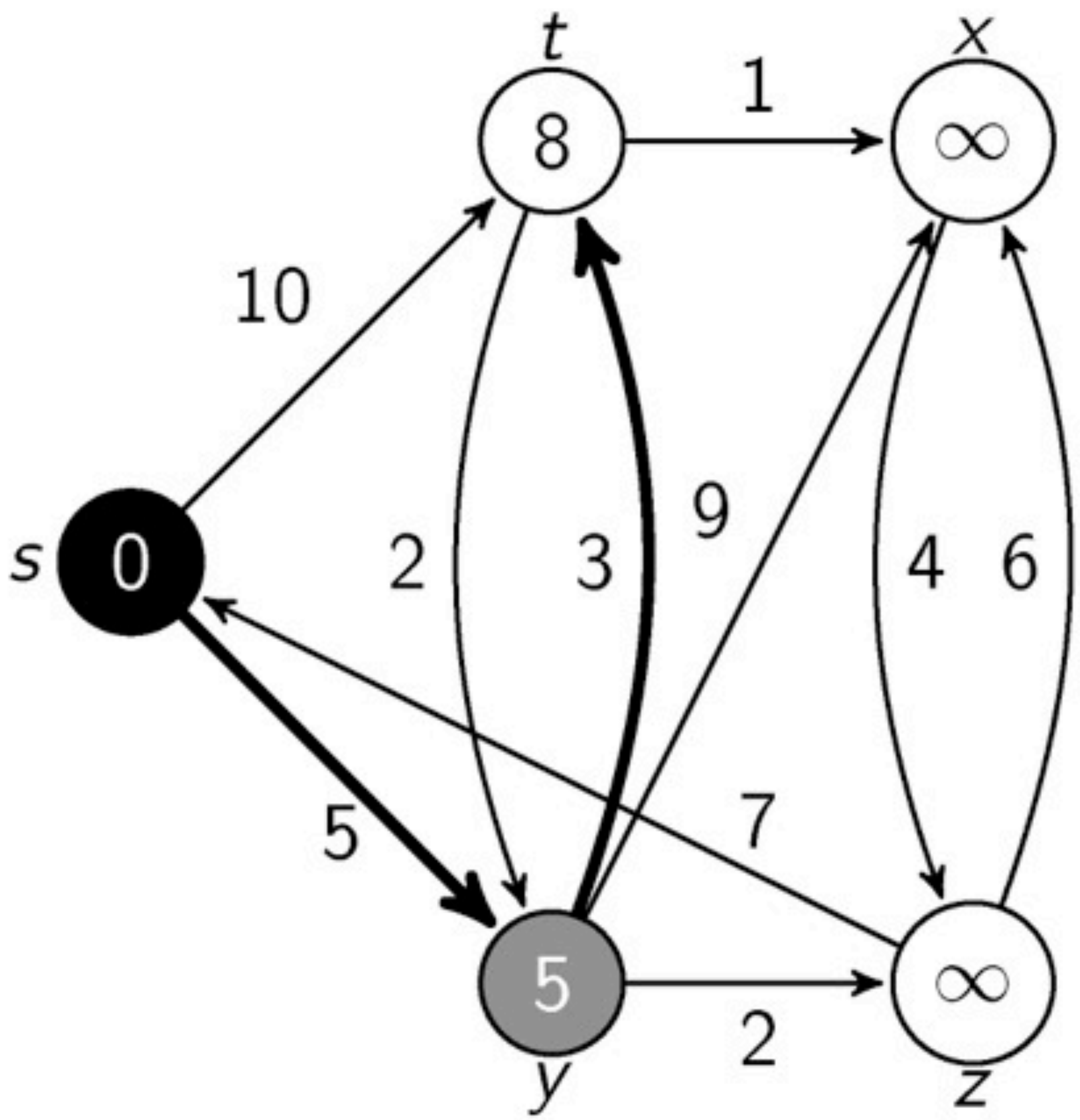
y=EXTRACT-MIN(Q)
$S = \{s, y\}$
$Q = \{t, x, z\}$

We are at node y



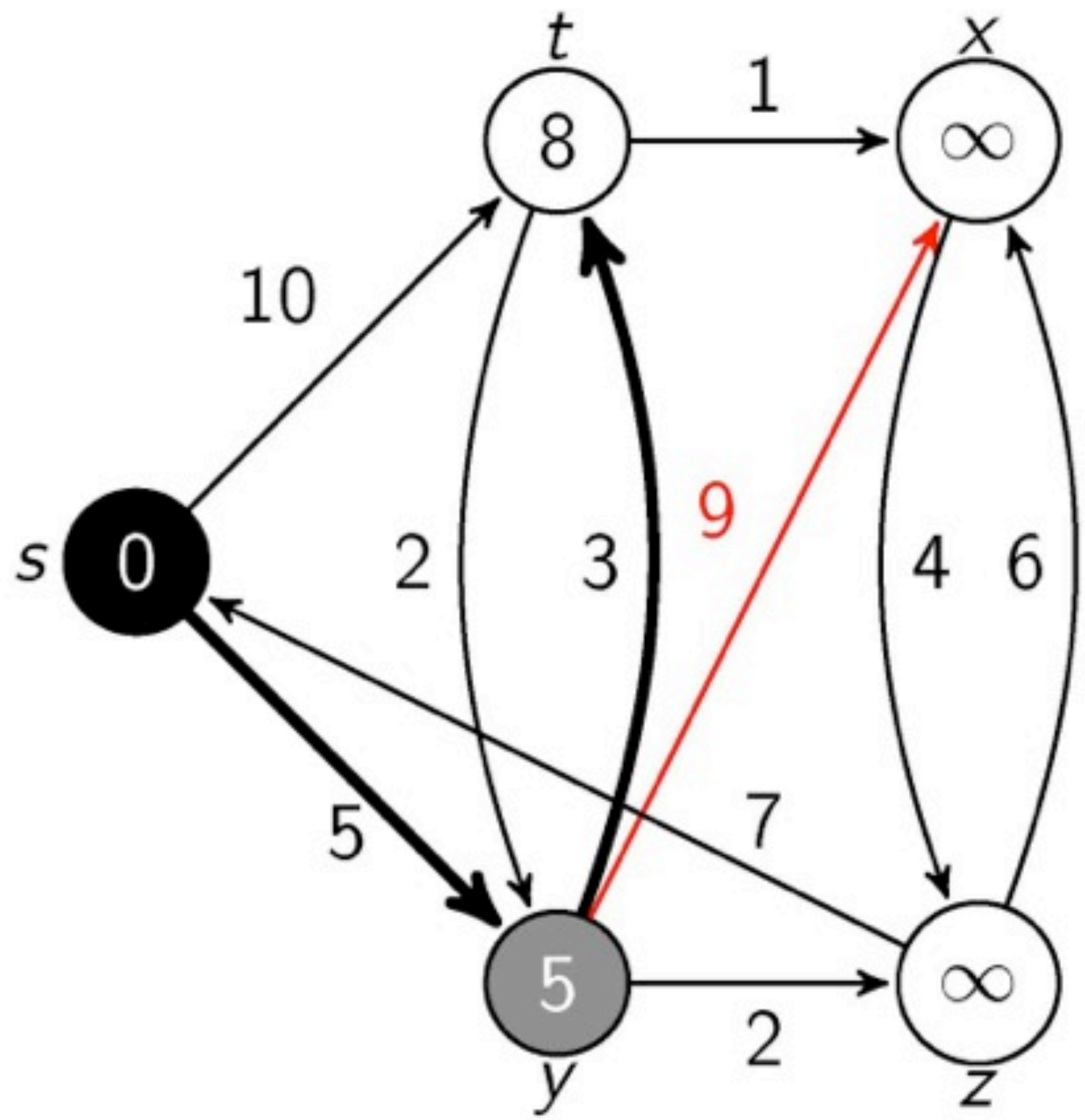
$\text{RELAX}(y, t, w)$
$S = \{s, y\}$
$Q = \{t, x, z\}$

Test whether we can improve the shortest path to t found so far by going through y



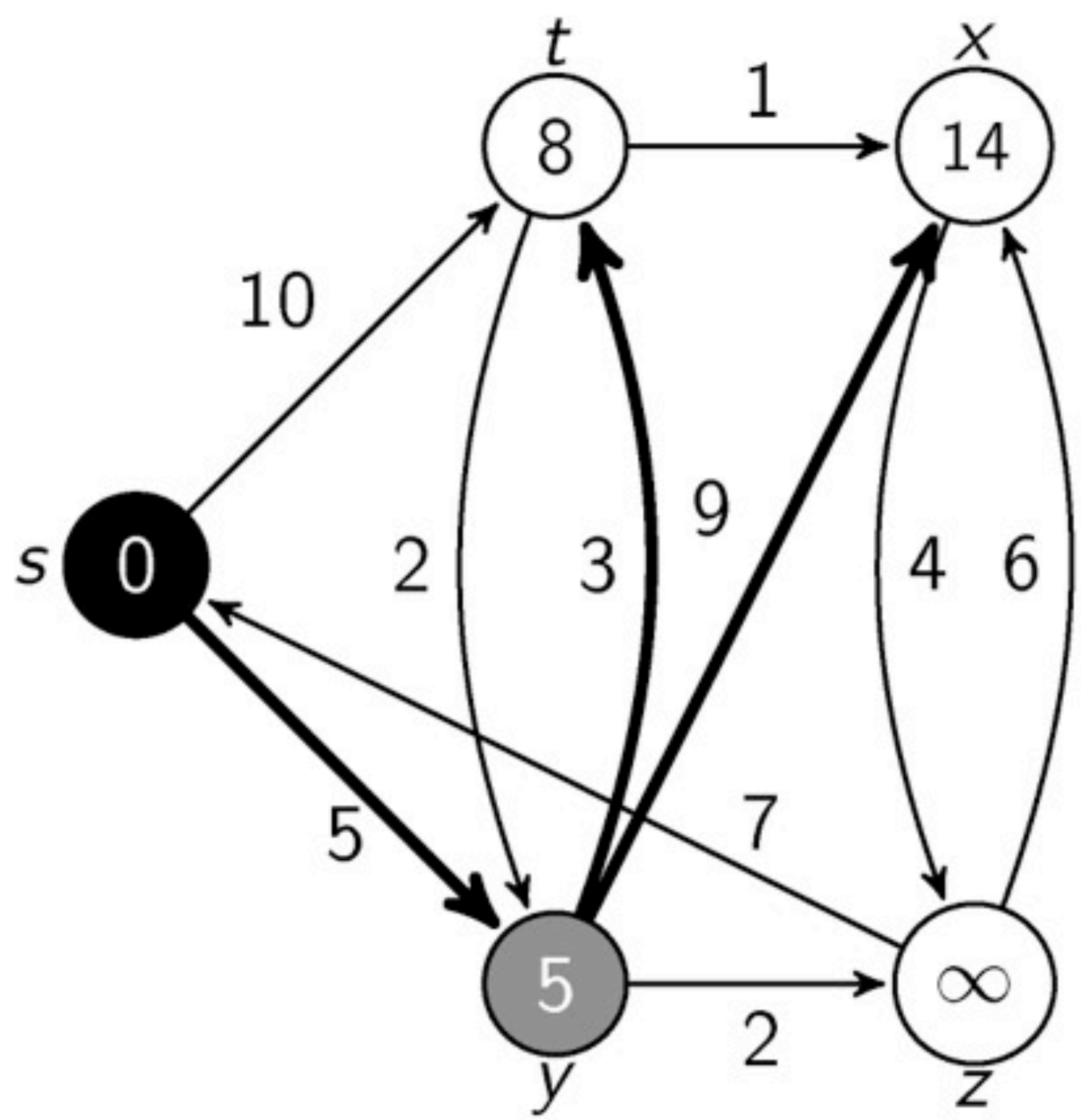
$\textsc{Relax}(y, t, w)$
$S = \{s, y\}$
$Q = \{t, x, z\}$

Update $t.d = 8$ and $t.\pi = y$



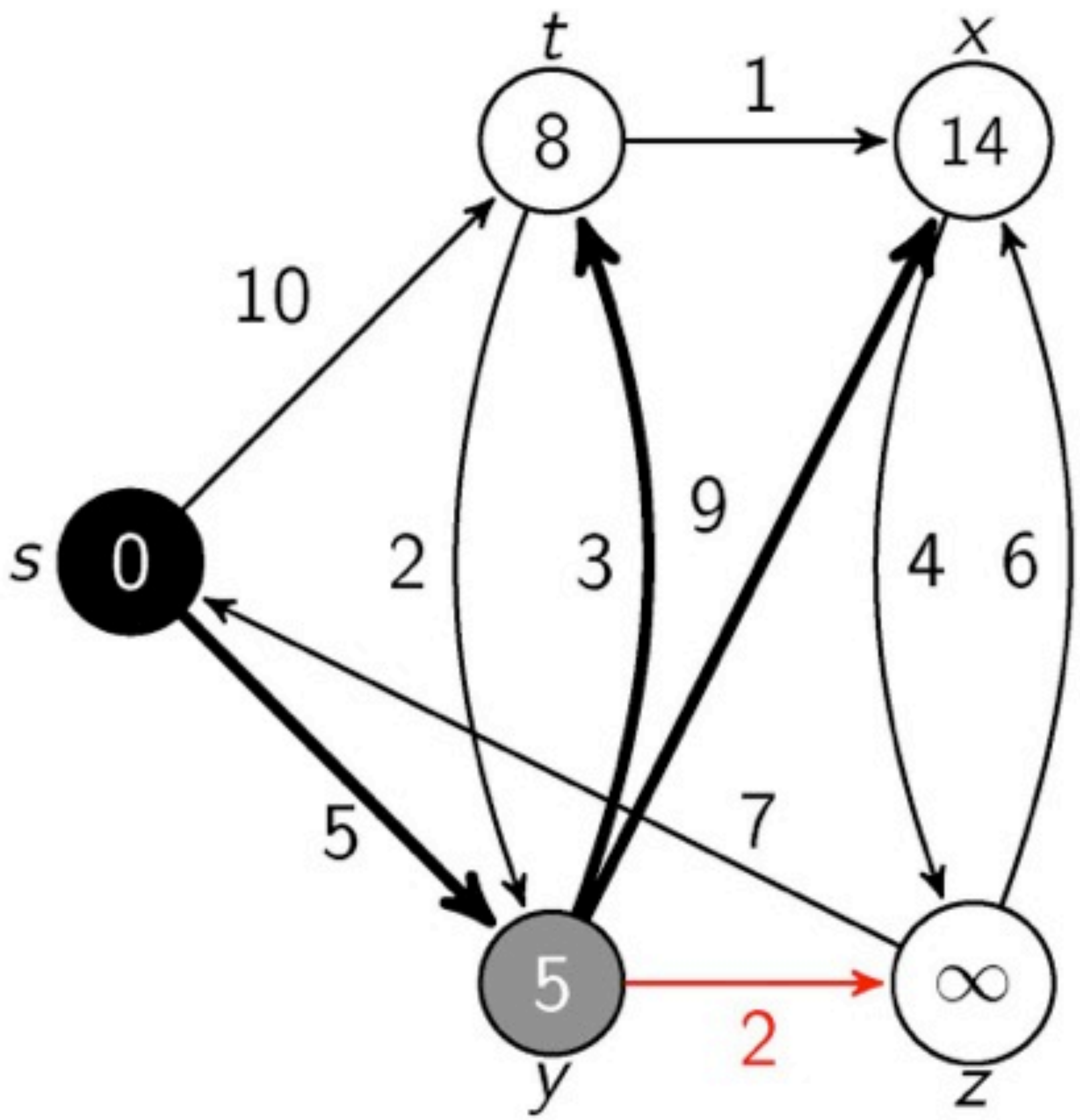
$\text{RELAX}(y, x, w)$
$S = \{s, y\}$
$Q = \{t, x, z\}$

Test whether we can improve the shortest path to x found so far by going through y



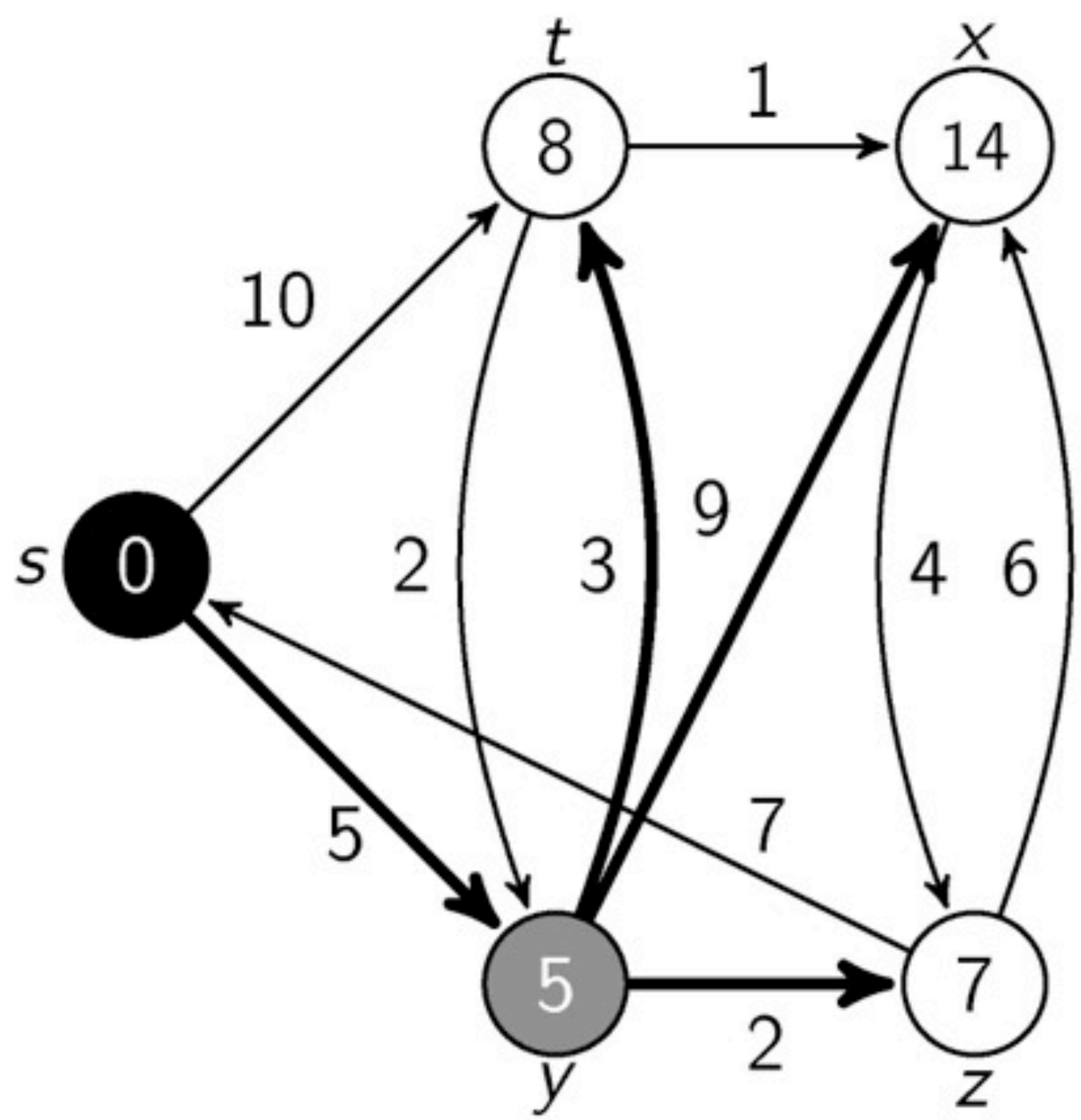
$\textsc{Relax}(y, x, w)$
$S = \{s, y\}$
$Q = \{t, x, z\}$

Update $x.d = 14$ and $x.\pi = y$



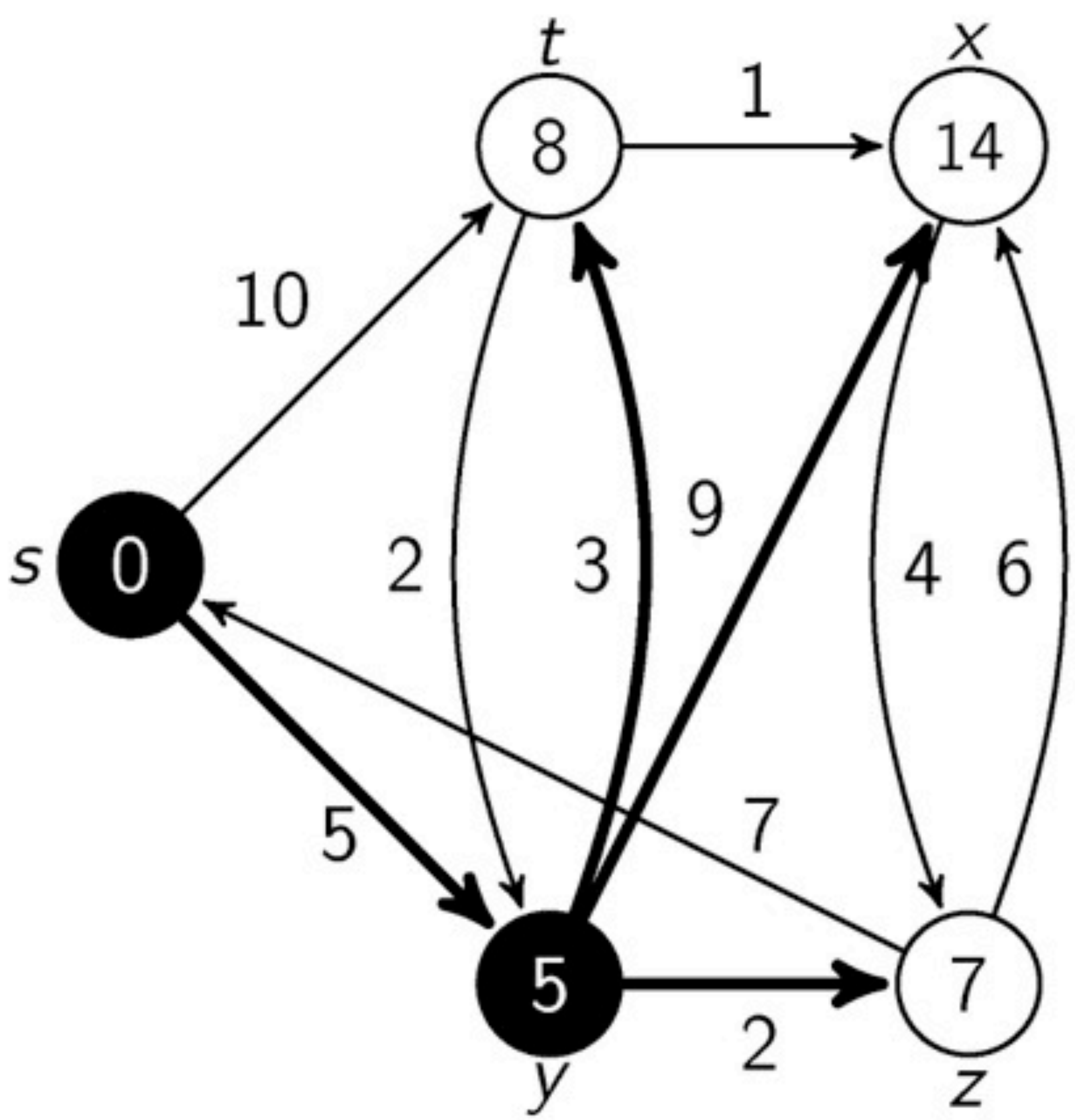
$\text{RELAX}(y, z, w)$
$S = \{s, y\}$
$Q = \{t, x, z\}$

Test whether we can improve the shortest path to z found so far by going through y



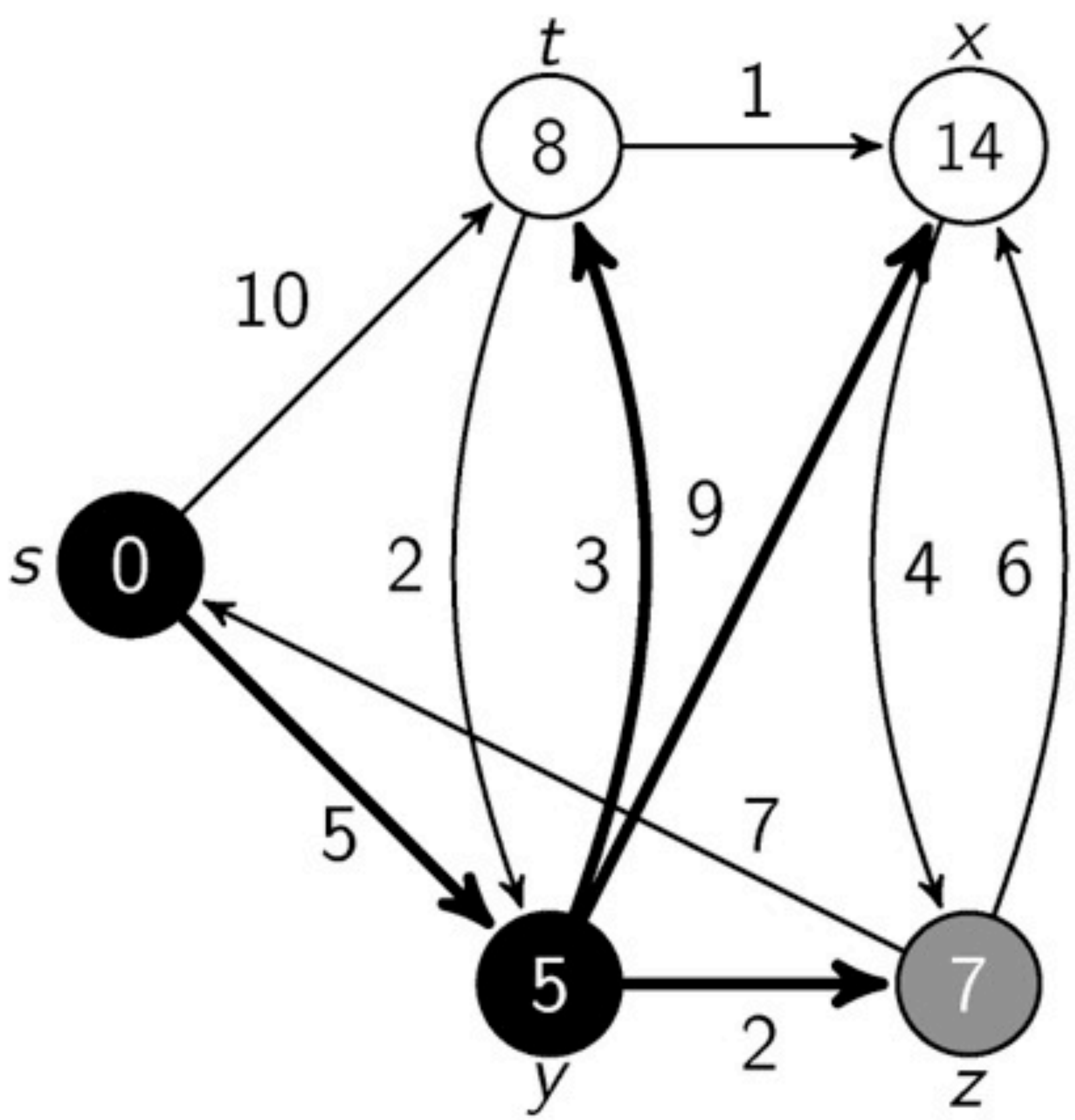
$\text{RELAX}(y, z, w)$
$S = \{s, y\}$
$Q = \{t, x, z\}$

Update $z.d = 7$ and $z.\pi = y$



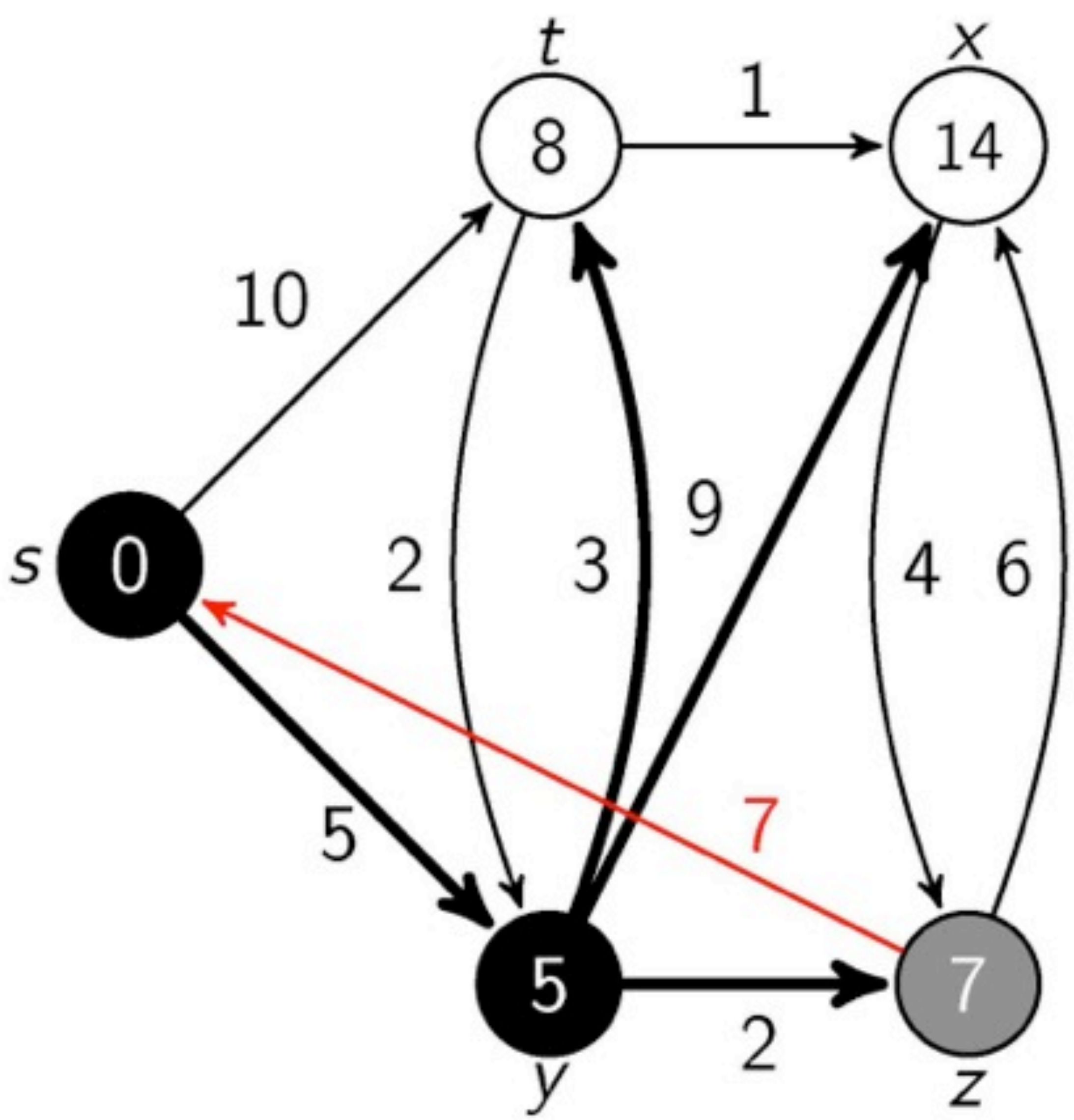
$S = \{s, y\}$
$Q = \{t, x, z\}$

All edges leaving y have been tested



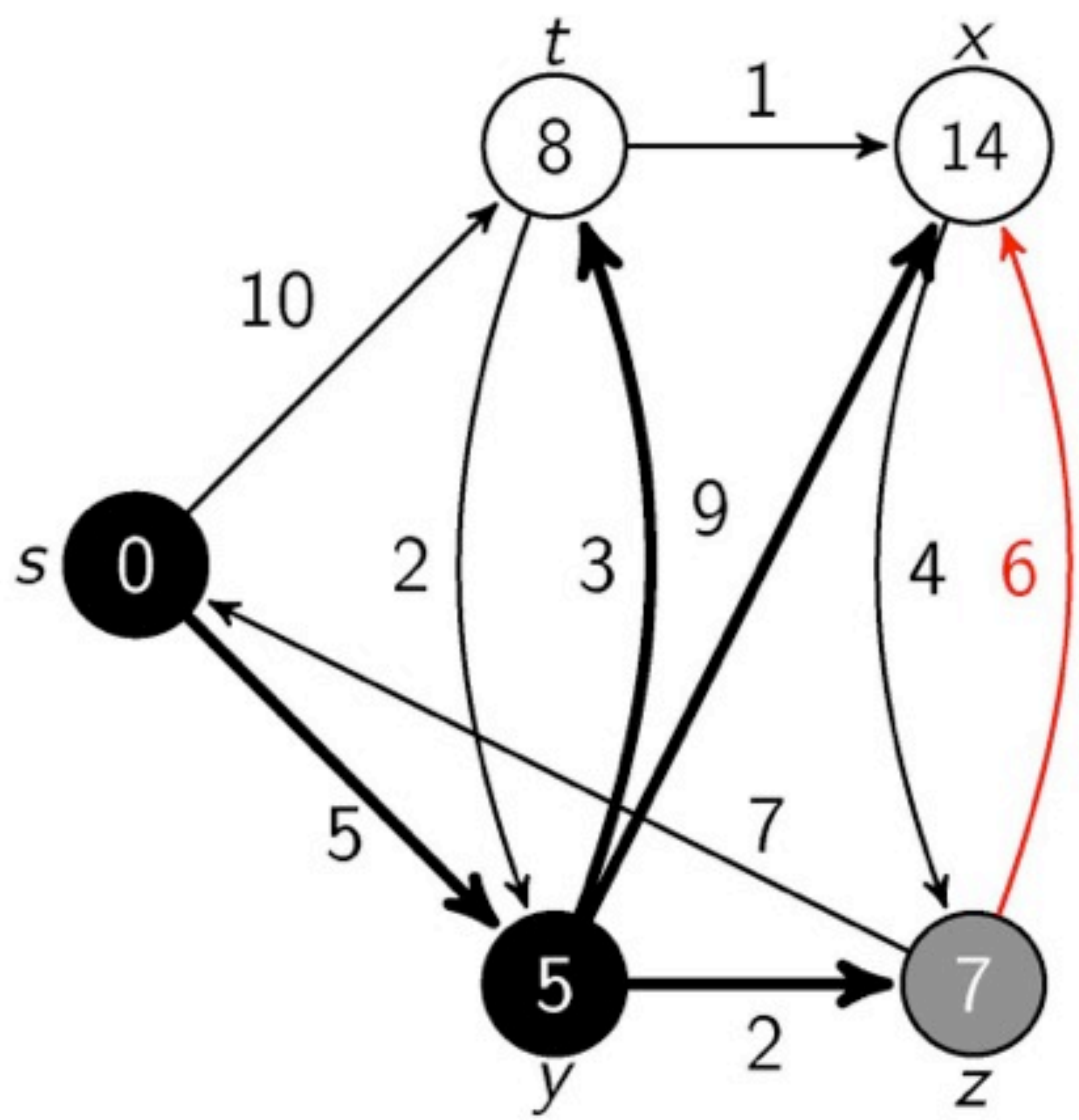
z=EXTRACT-MIN(Q)
$S = \{s, y, z\}$
$Q = \{t, x\}$

We are at node z



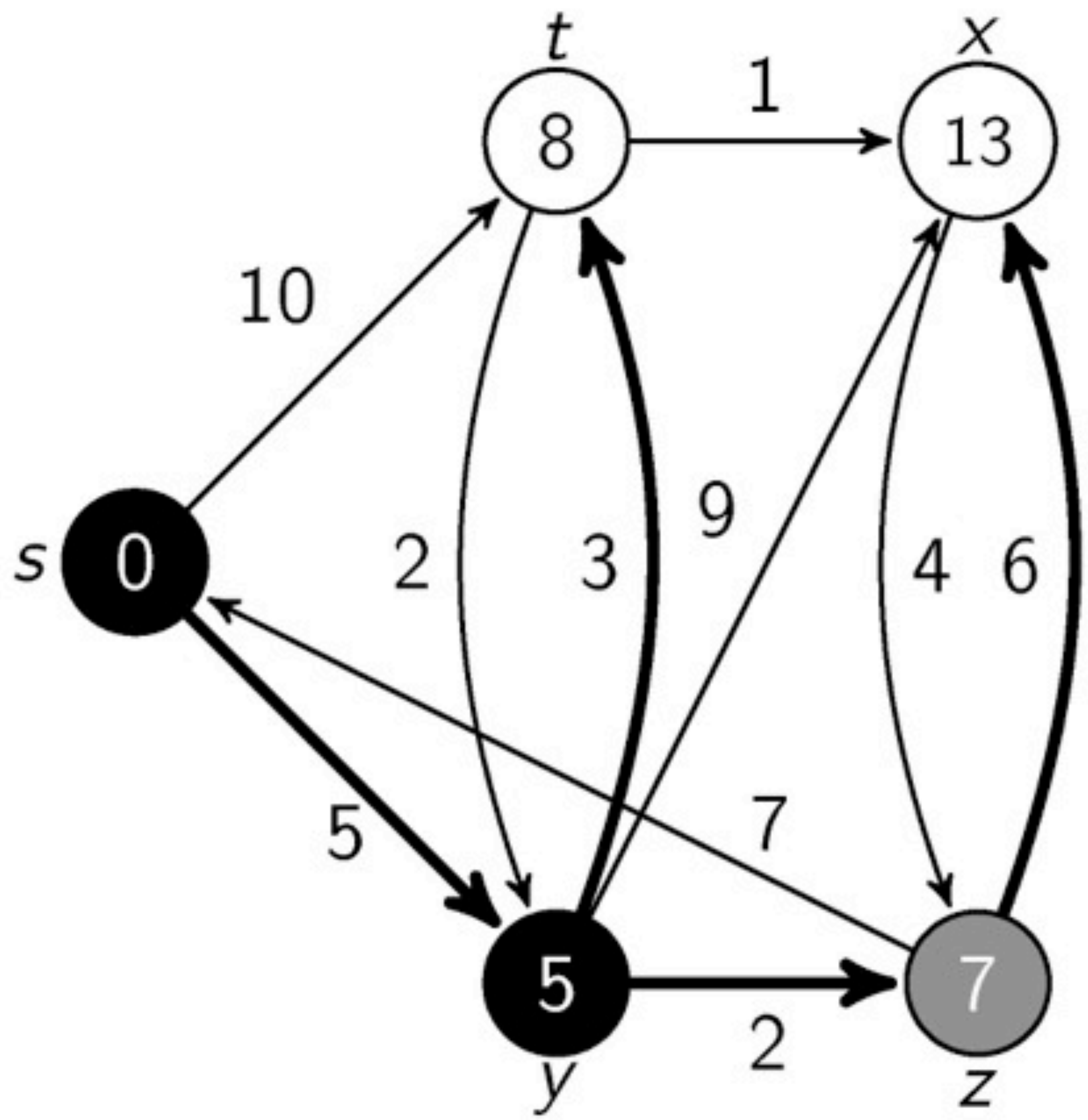
$\text{RELAX}(z, s, w)$
$S = \{s, y, z\}$
$Q = \{t, x\}$

Test whether we can improve the shortest path to s found so far by going through z



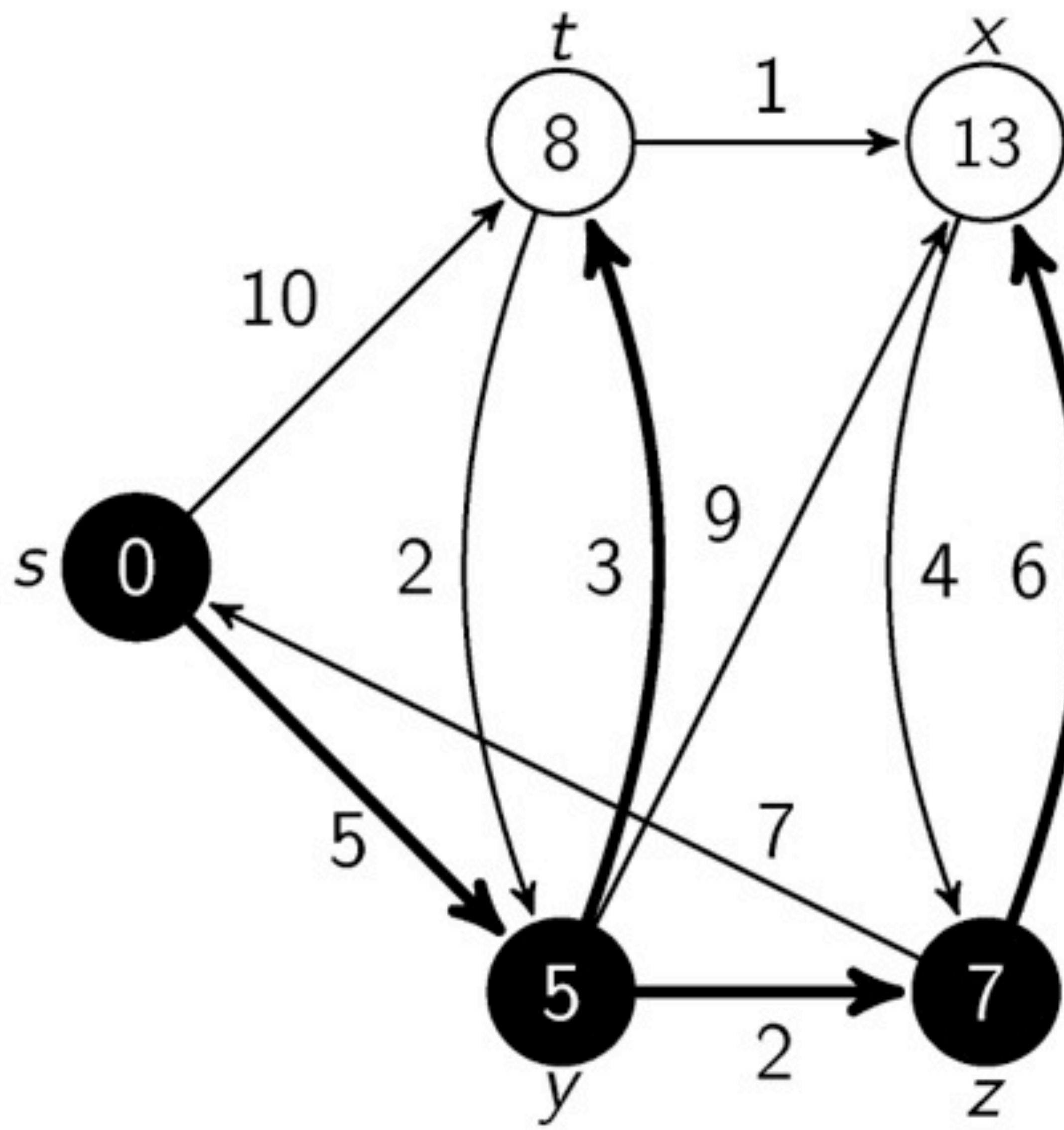
RELAX(z, x, w)
$S = \{s, y, z\}$
$Q = \{t, x\}$

Test whether we can improve the shortest path to x found so far by going through z



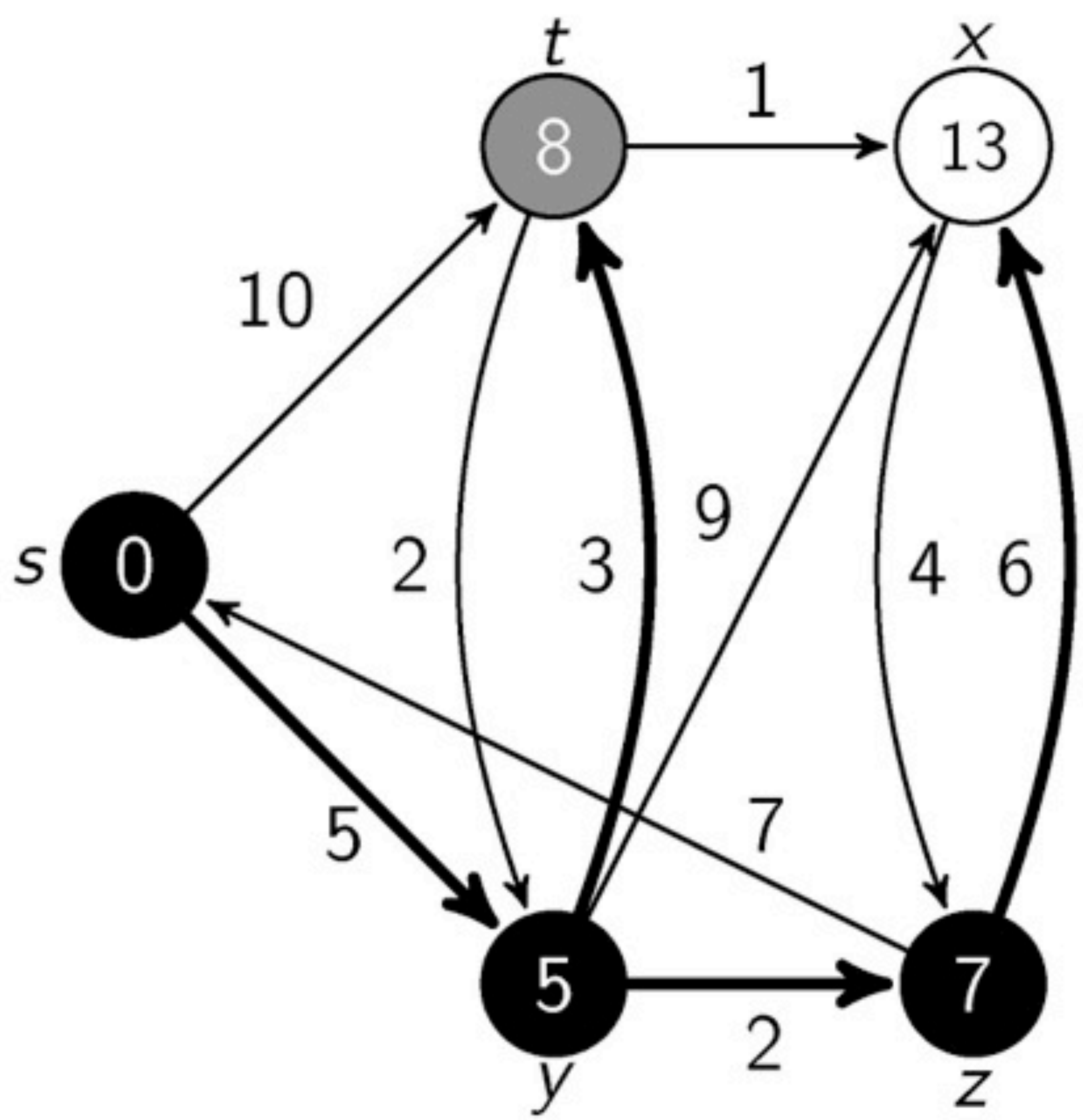
$\text{Relax}(z, x, w)$
$S = \{s, y, z\}$
$Q = \{t, x\}$

Update $x.d = 13$ and $x.\pi = z$



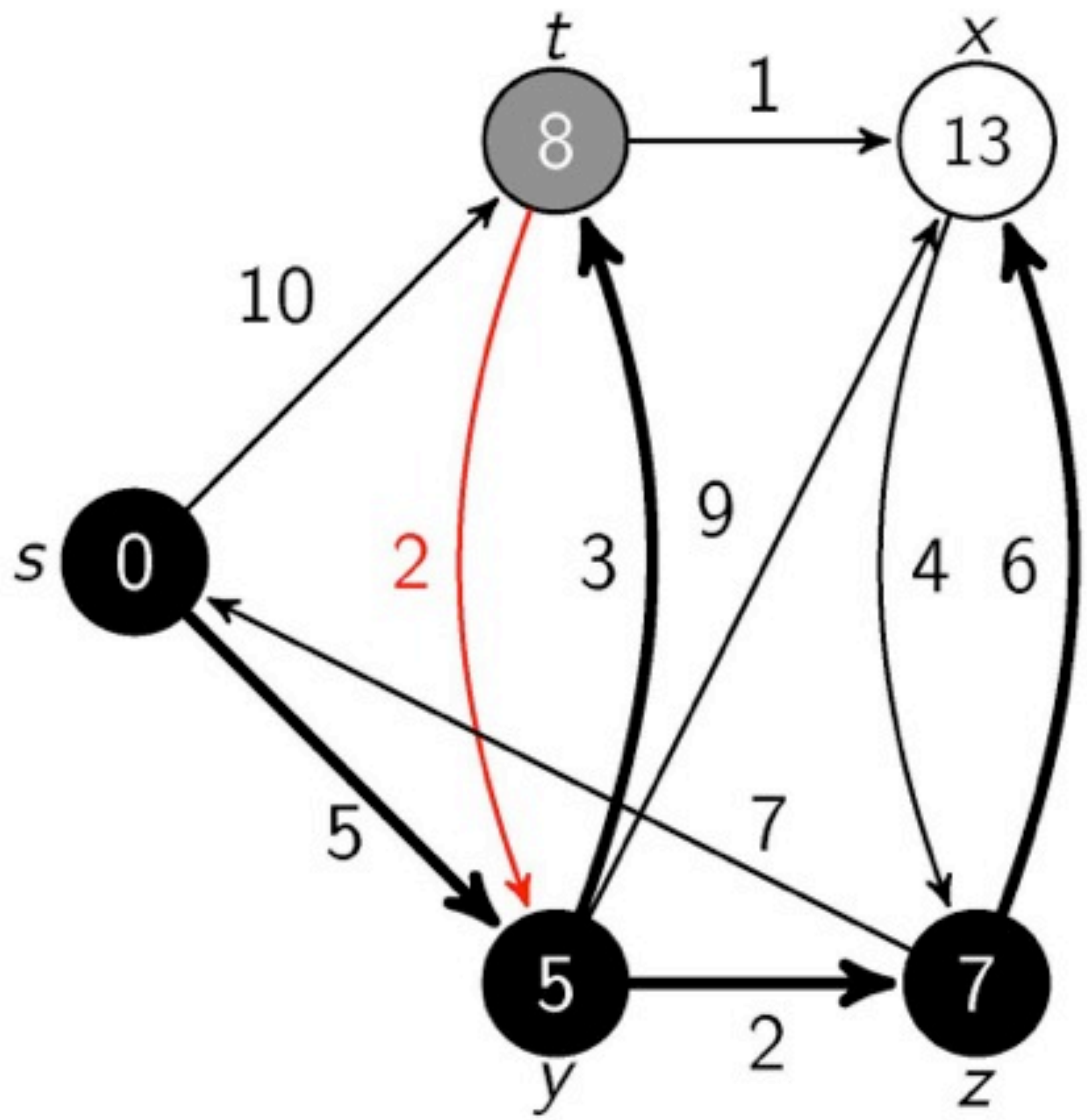
$S = \{s, y, z\}$
$Q = \{t, x\}$

All edges leaving z have been tested



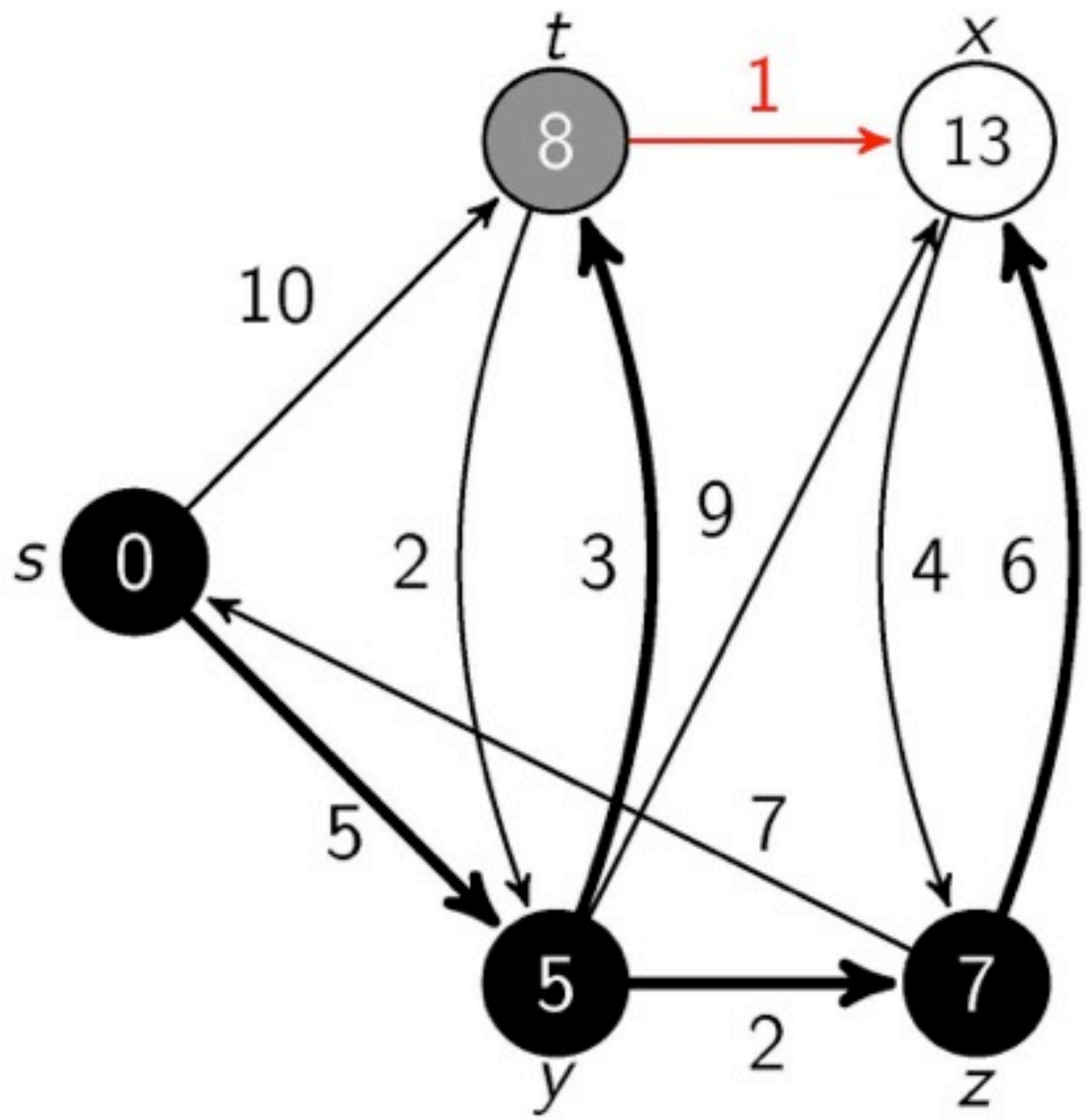
t=EXTRACT-MIN(Q)
$S = \{s, y, z, t\}$
$Q = \{x\}$

We are at node t



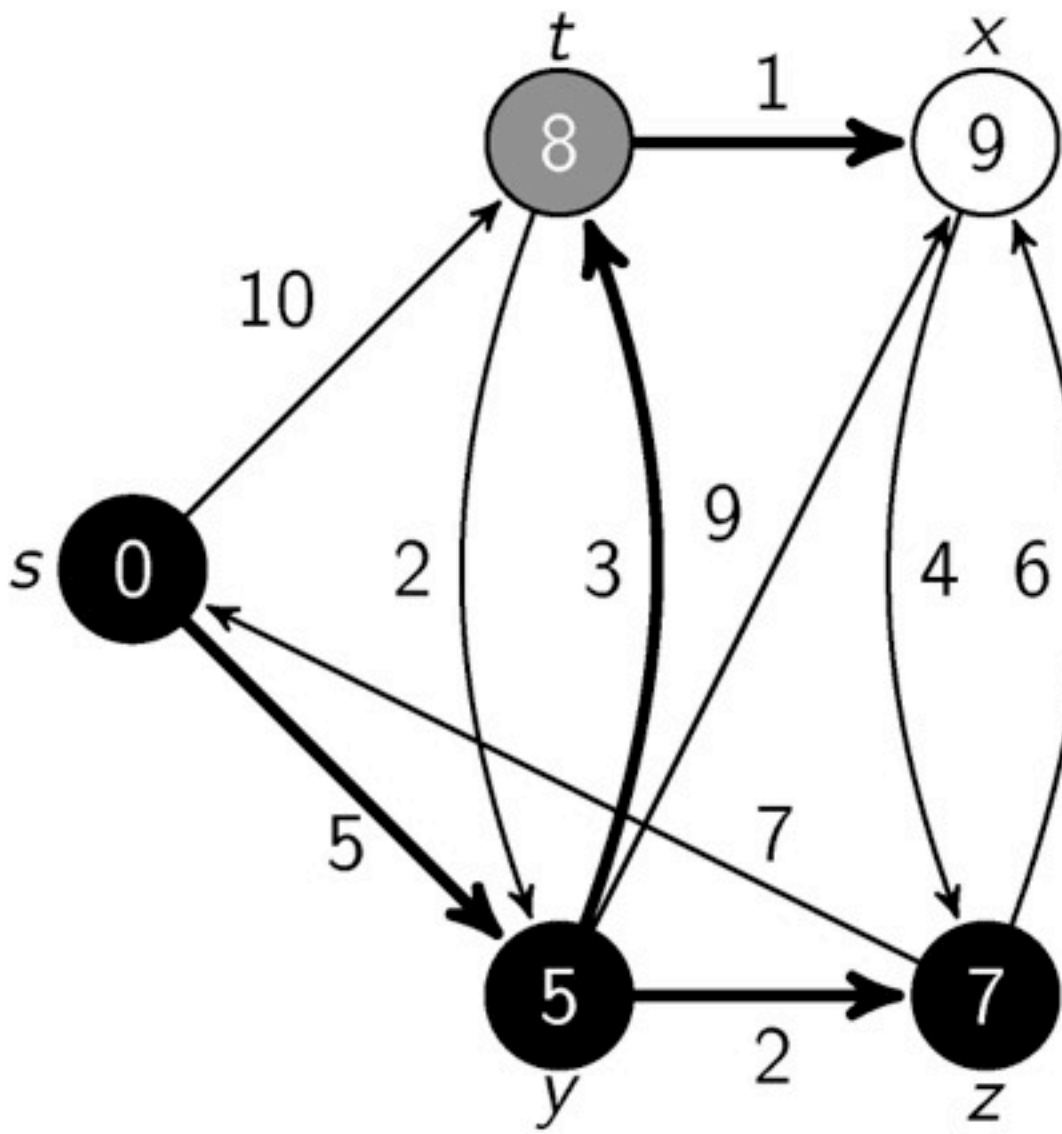
$$\text{RELAX}(t, y, w)$$
$$S = \{s, y, z, t\}$$
$$Q = \{x\}$$

Test whether we can improve the shortest path to y found so far by going through t



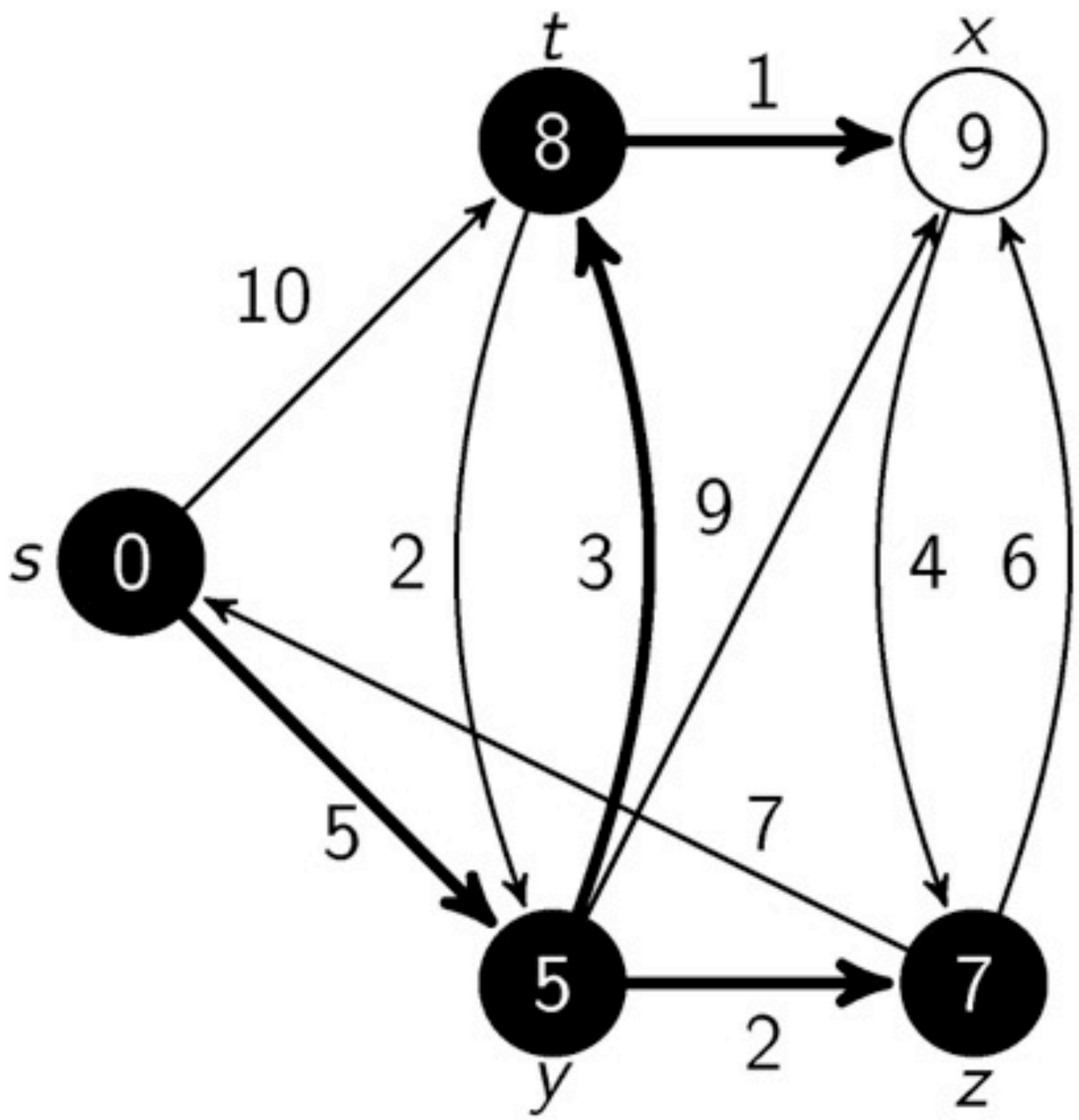
$\textsc{Relax}(t, x, w)$
$S = \{s, y, z, t\}$
$Q = \{x\}$

Test whether we can improve the shortest path to x found so far by going through t



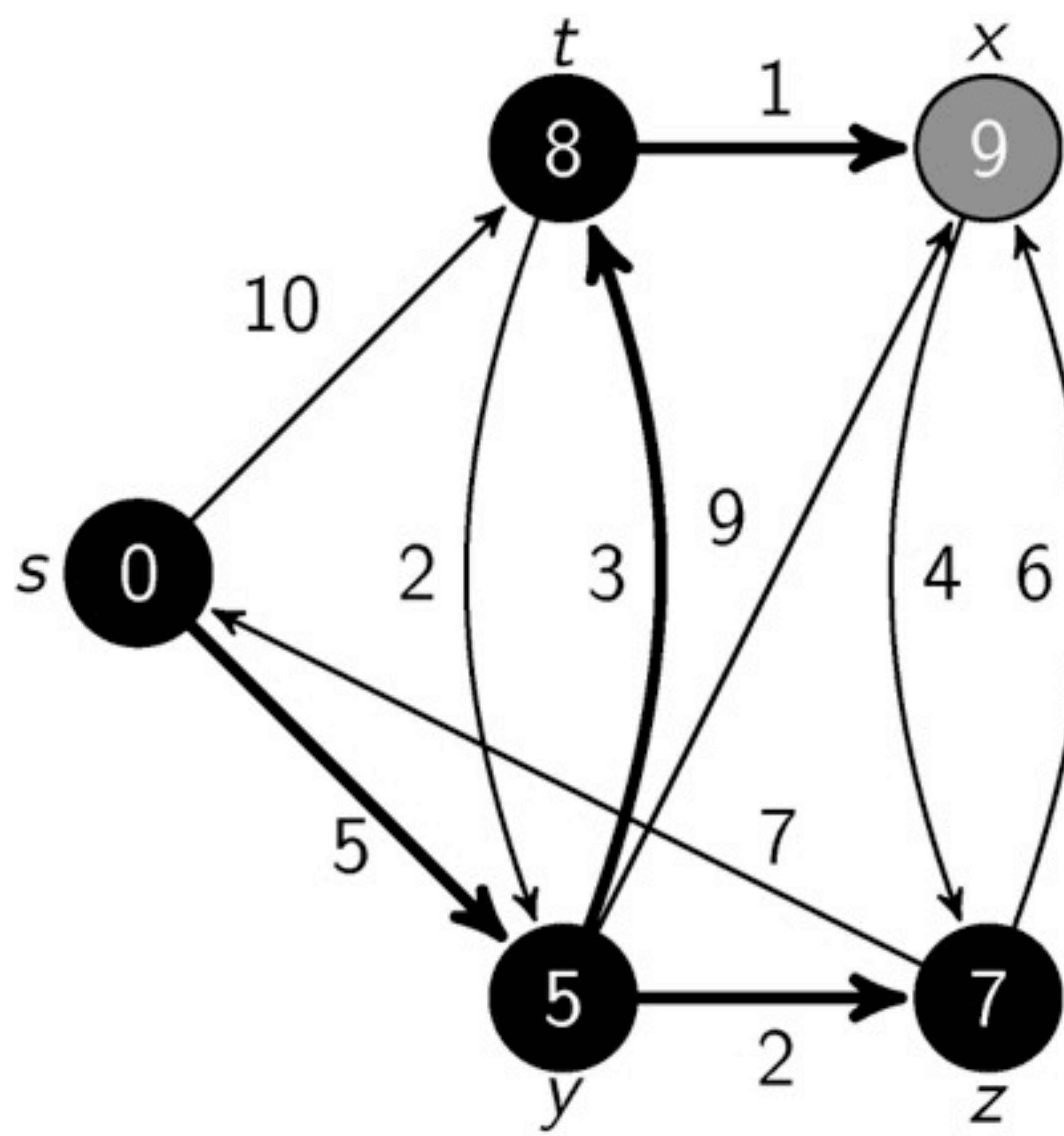
$\text{RELAX}(t, x, w)$
$S = \{s, y, z, t\}$
$Q = \{x\}$

Update $x.d = 9$ and $x.\pi = t$



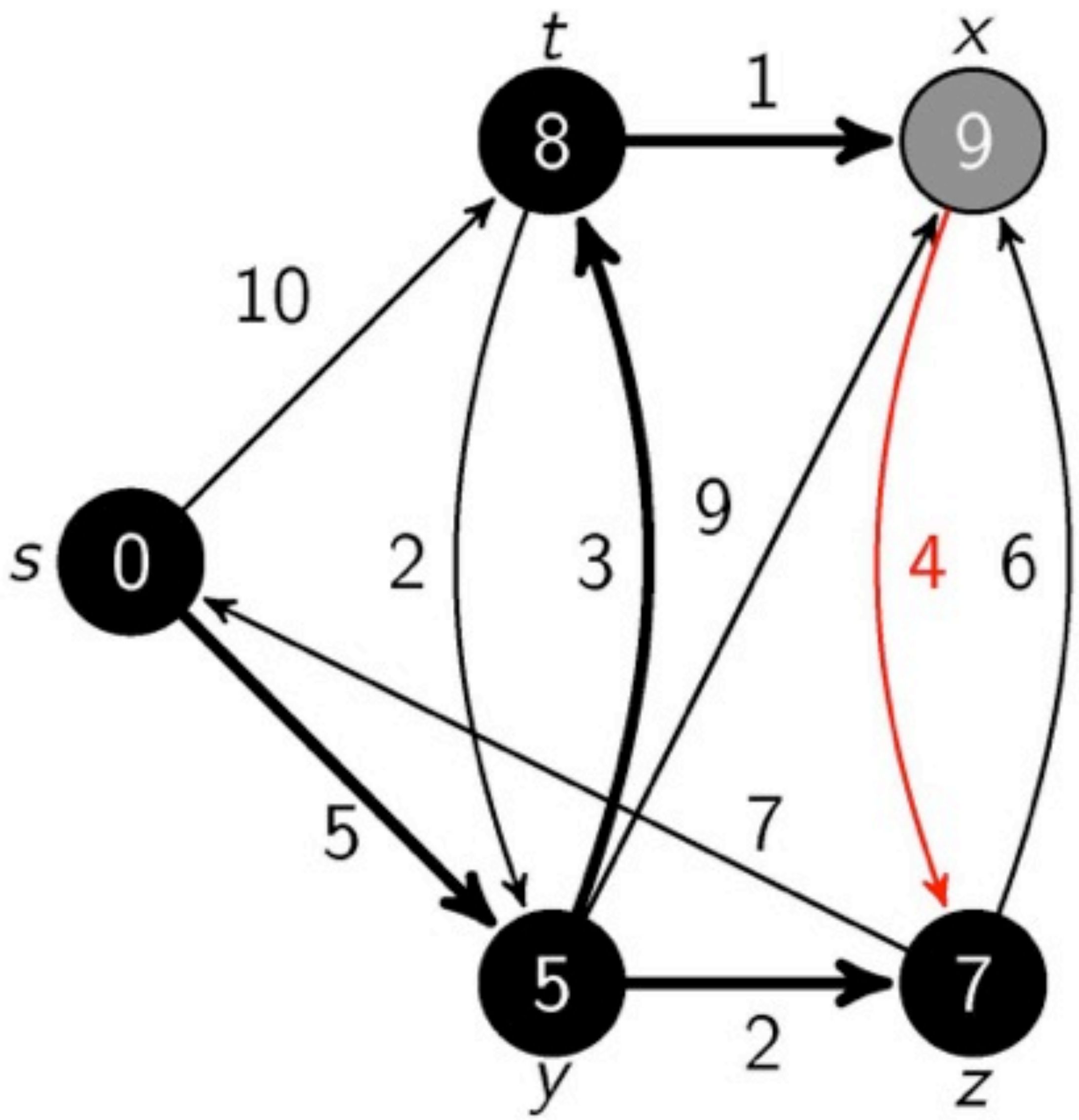
$S = \{s, y, z, t\}$
$Q = \{x\}$

All edges leaving t have been tested



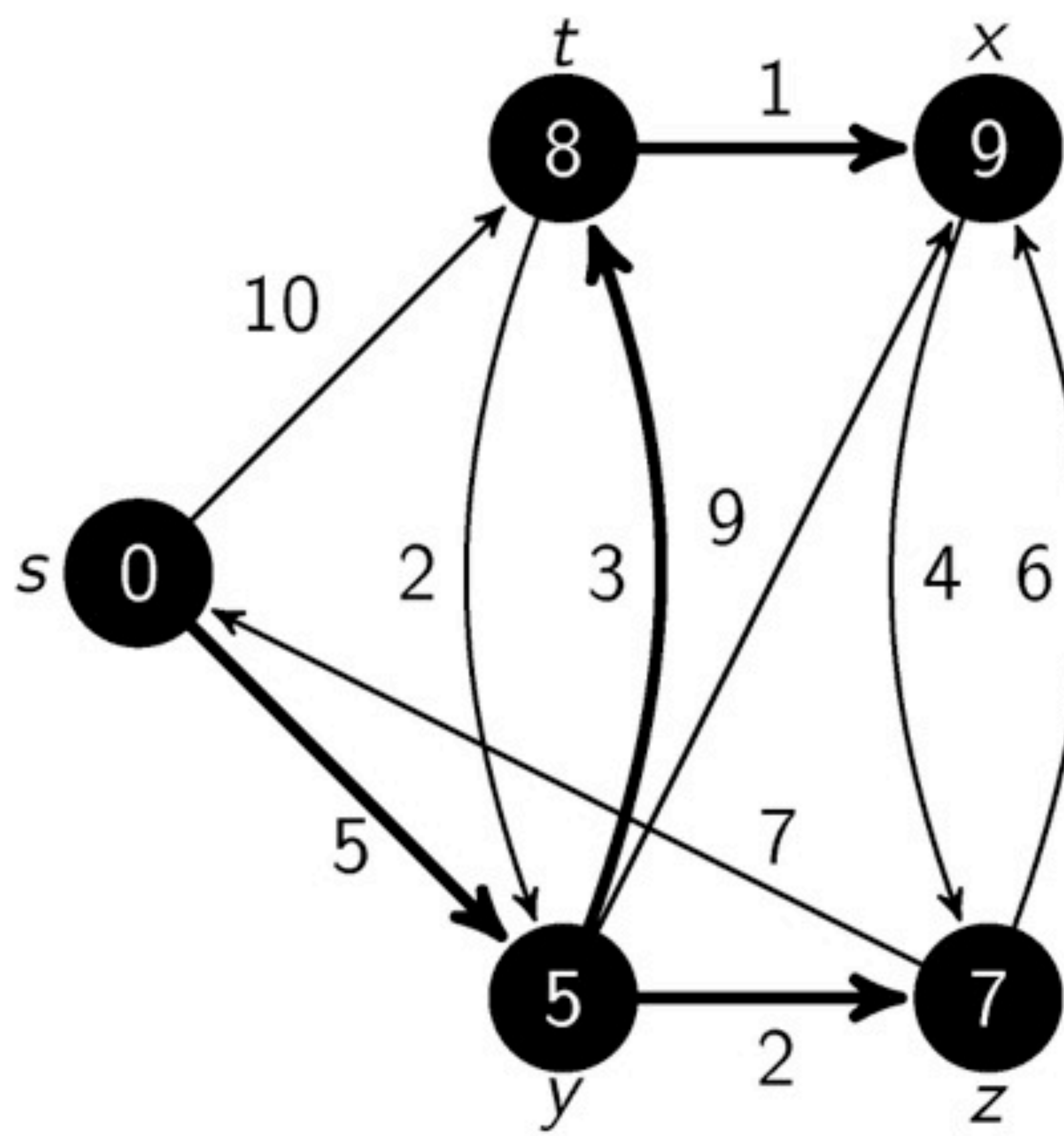
x=EXTRACT-MIN(Q)
$S = G.V$
$Q = \Phi$

We are at node x



$\text{RELAX}(x, z, w)$
$S = G.V$
$Q = \Phi$

Test whether we can improve the shortest path to z found so far by going through x



$S = G.V$
$Q = \Phi$
Done!

All edges leaving x have been tested.
Every vertex's shortest path from s has been determined. We are done.

Dijkstra's Algorithm

DIJKSTRA(G, w, s)

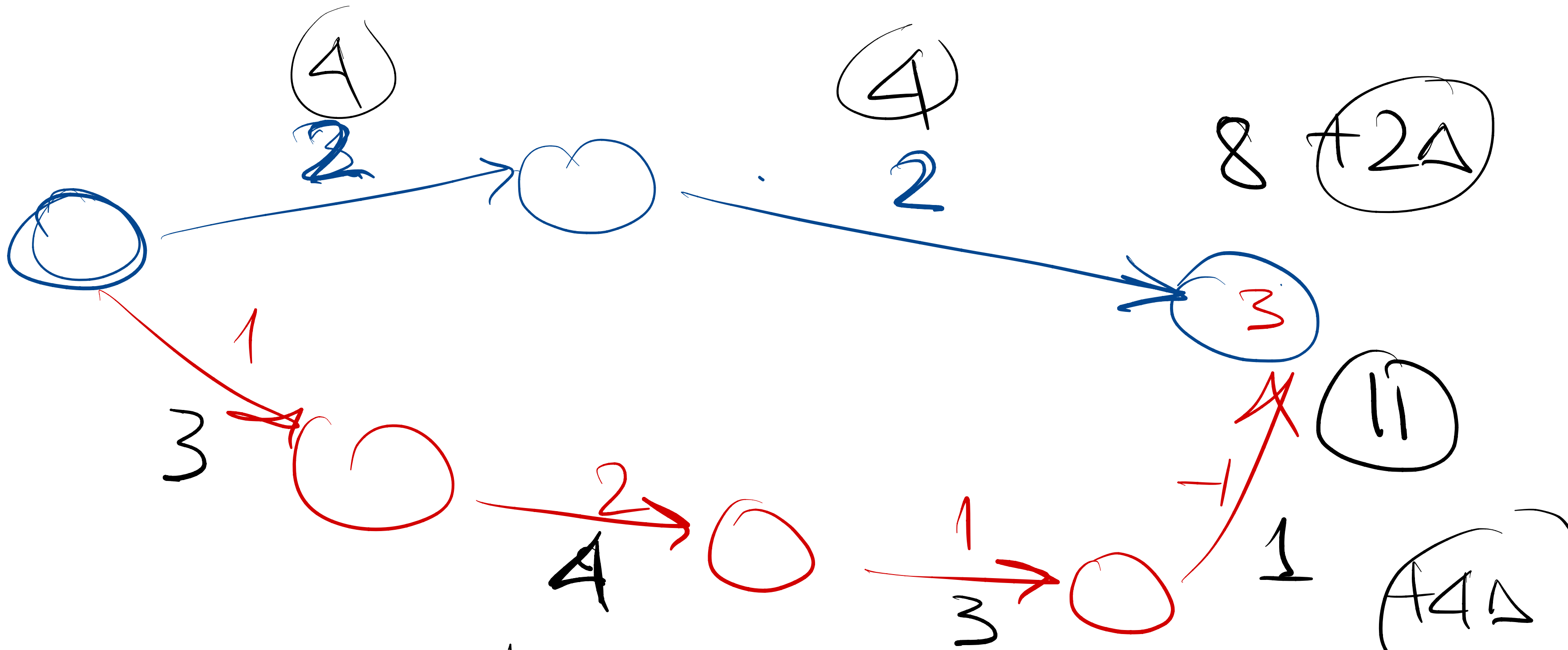
```
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

quurateed ESP = SP

dijkstra current SP-tree

all edges from u

- correctness proof in the book
  - idea: proof that for each SP, there is a relaxation sequence of its edges in path-order
- Running Time depends on implementation of queue operations
  - |V| * extract-min
  - |E| * decrease key (at relaxation)
- Total
  - $O(V^*T_{extract\text{-}min}+E^*T_{decrease\text{-}key})$
  - with Fibonacci heaps: extract-min is O(logV) and decrease-key is O(1) ; total O(E+VlogV)



$\Delta = +2$ need to add to break neg cycles

add to all edges

Graphs II - Shortest paths

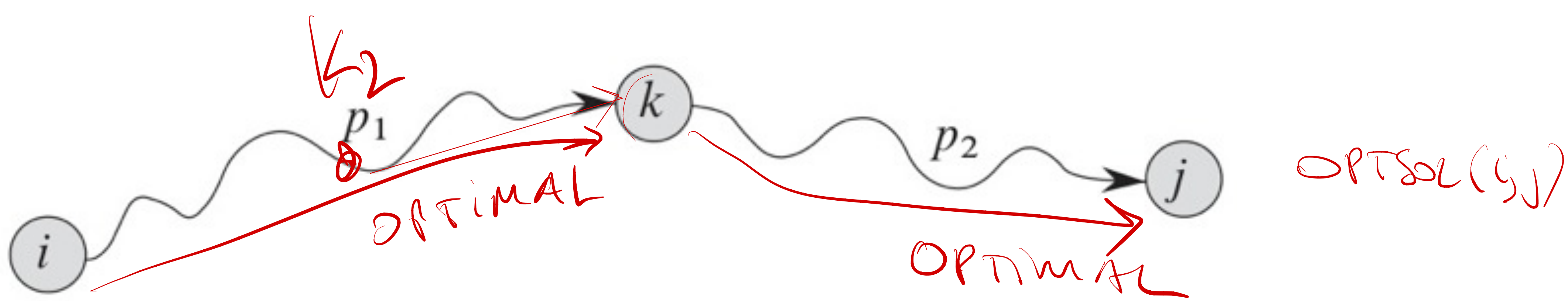
Lesson 2: All Sources Shortest Paths

ASSP

- Task: find all shortest paths, between any two vertices (no fixed source)
- Slow: run Bellman Ford separately from each vertex as source.
  - running time $|V|$ * BF-time = $V*O(VE) = O(V^2E)$
  - that is $O(V^4)$ if graph dense $E \approx V^2$

- Instead, we will use dynamic programming
- C_{ij} = min SP weight (objective) between vertices i,j
- optimal solution structure:
  - if path P(i->j) from i to j in optimal and passes vertex k, then the subpaths P(i->k) and P(k->j) must be also optimal
  - optimal = shortest

① charact. OPTSOL



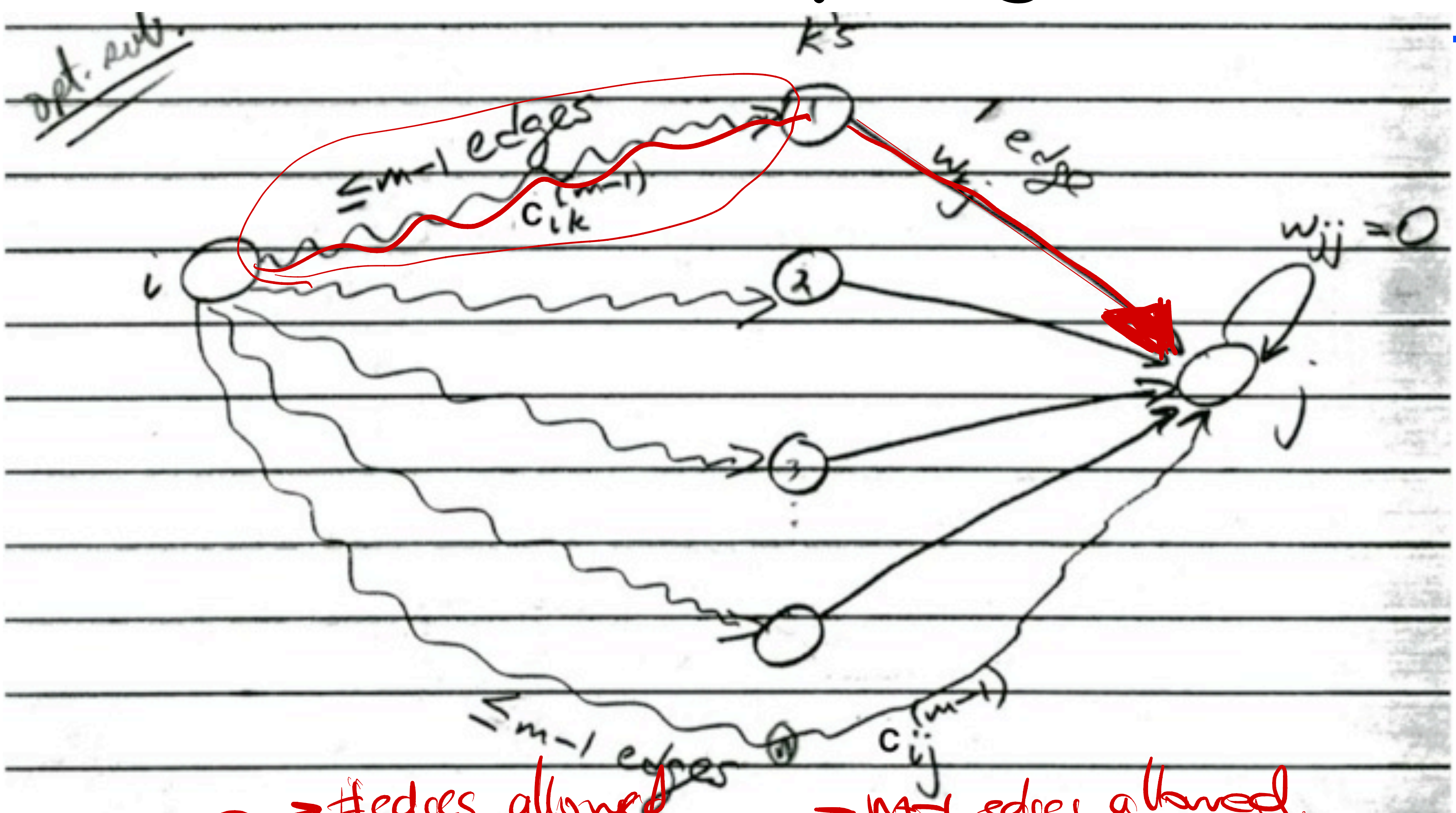
ASSP dynamic programming

- two options for dynamic programming

OPTSOL($\leq k$ edges)
$\Downarrow$
OPTSOL($\leq k+1$ edges)

- A. go by the number of edges used in a path
  - $C_{ij}^{(m)}$= minimum path weight between i and j using at most m edges
  - $C_{ij}^{(1)}$= weight of edge i->j, if exists (one edge)
  - $C_{ij}^{(2)}$= min weight of any path i->k->j (max 2 edges)
  - $C_{ij}^{(0)}$= we 0 if i≠j, ∞ otherwise (no edge)
- B. by the intermediary nodes in a certain fixed order
  - fix order of all vertices 1,2,3,...,|V|
  - $C_{ij}^{(m)}$= minimum path weight between i and j using only intermediary vertices {1,2,...m}
  - similar to discrete knapsack idea, see module 6

ASSP dynamic programming by edges



#edges allowed   m-1 edges allowed

- $C_{ij}^{(m)} = \min_k \{ C_{ij}^{(m-1)}, \; C_{ik}^{(m-1)} + w_{kj} \}$ *//bottom up computation*
- the Cij using m edges is either
  - the same as Cij using m-1 edges, OR
  - C_{ik} using m-1 edges to intermediary k, plus an edge from k to j w_{kj}
  - all nodes k are eligible as possible "last" intermediary

ASSP dynamic programming by edges

- Compute the $C^{(m)}$ matrix from $C^{(m-1)}$ matrix using edges matrix W
- `Extend-SP (C^(m-1),W)`

```
for i=1:n
  for j=1:n
    a=∞;
    for k=1:n
      a=min{a, C_ik^(m-1) + W_kj};
    C_ij^(m)=a
```

every cell $c_{ij}^{(m)}$

$\Theta(n^3)$

search for prev vertex $k \to j$

$c_{ij}^{(m-1)}$ included when $k=j$

new possibility.

- `ASSP-slow(W)`

```
C^(1) = W
for m=2:n-1
  C^(m)=Extend-SP(C(m-1),W)
return C^(n-1)
```

$\Theta(n^3)$

$T^{100} = T^{64} * T^{32} \times T^{4}$

ASSP dynamic programming by edges

- Extend-SP looks like matrix multiplication!
  - Extend-SP running time $O(n^3)$
- ASSP-slow is $n^*O(n^3) = O(n^4)$, same as running Bellman Ford separately from each vertex

wrapper Matrix-Mult

$A = T$

for $m = 2 : n$

$A = A * T$

$A = T^n$

Extend-SP $(C^{(m-1)}, W)$

```
for i=1:n
  for j=1:n
    a=∞;
    for k=1:n
      a=min{a, Cik^(m-1) + wkj};
    Cij^(m)=a
```

cell

$\Theta(n)$ cmp

operator

D=multiply(C,W)

```
for i=1:n
  for j=1:n
    a=0;
    for k=1:n
      a=a+ Cik * wkj;
    Dij=a
```

cell

$\Theta(n)$ cmp

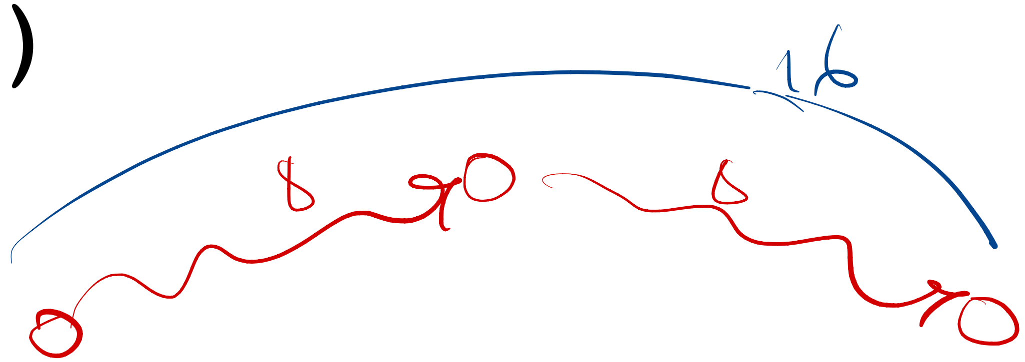
operator

ASSP dynamic programming by edges

- Think of Extending-SP as of matrix multiplication
  - $C^{(1)} = C^{(0)} * W = W$; the "*" means "a=min{a, $c_{ik}^{(m-1)} + w_{kj}$}" inner operation
  - $C^{(2)} = C^{(1)} * W = W2$
  - $C^{(3)} = C^{(2)} * W = W3$
  - . . . . . .
- Only need $C^{(n-1)}$, not the intermediary ones
  - $C^{(1)} = W$
  - $C^{(2)} = W^2 = (W^1)^2$
  - $C^{(4)} = W^4 = (W^2)^2$
  - $C^{(8)} = W^8 = (W^4)^2$, etc

ASSP dynamic programming by edges

- `ASSP-fast(W)`
  - $C^{(1)} = W;$
  - `while m<n-1`
    - $C^{(m)}=\text{Extend-SP}(C^{(m-1)}, C^{(m-1)}, W);$
    - $\boxed{m=2*m;}$
  - `return` $C^{(m)}$



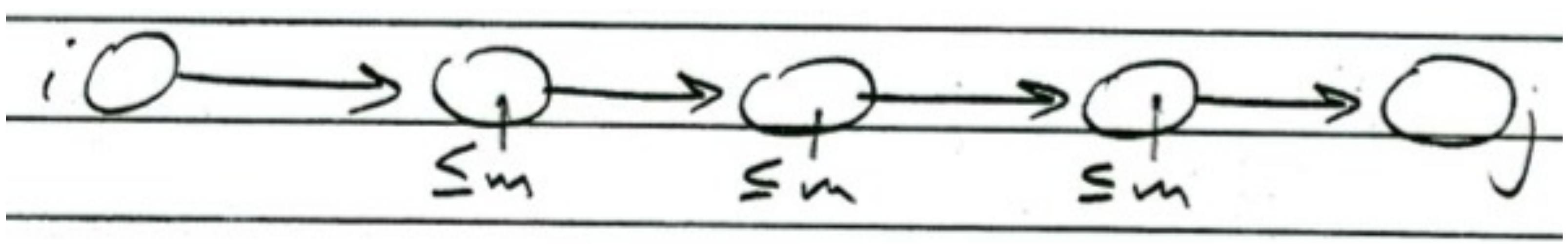
$T^{100} = T^{64} \cdot T^{32} \cdot T^{4}$ not needed

wasted use $\cancel{T^{128}}$ C^{128}

- After $\lceil \lg(n) \rceil$ iterations we have computed $C^{(m)}$ with $m \geq n-1$. Its ok to "overshoot" as C doesnt change after finding the SP.
- Running time $\Theta(V^3 \log V)$ $V^3 \cdot V$

ASSP dynamic programming by intermediary vertices

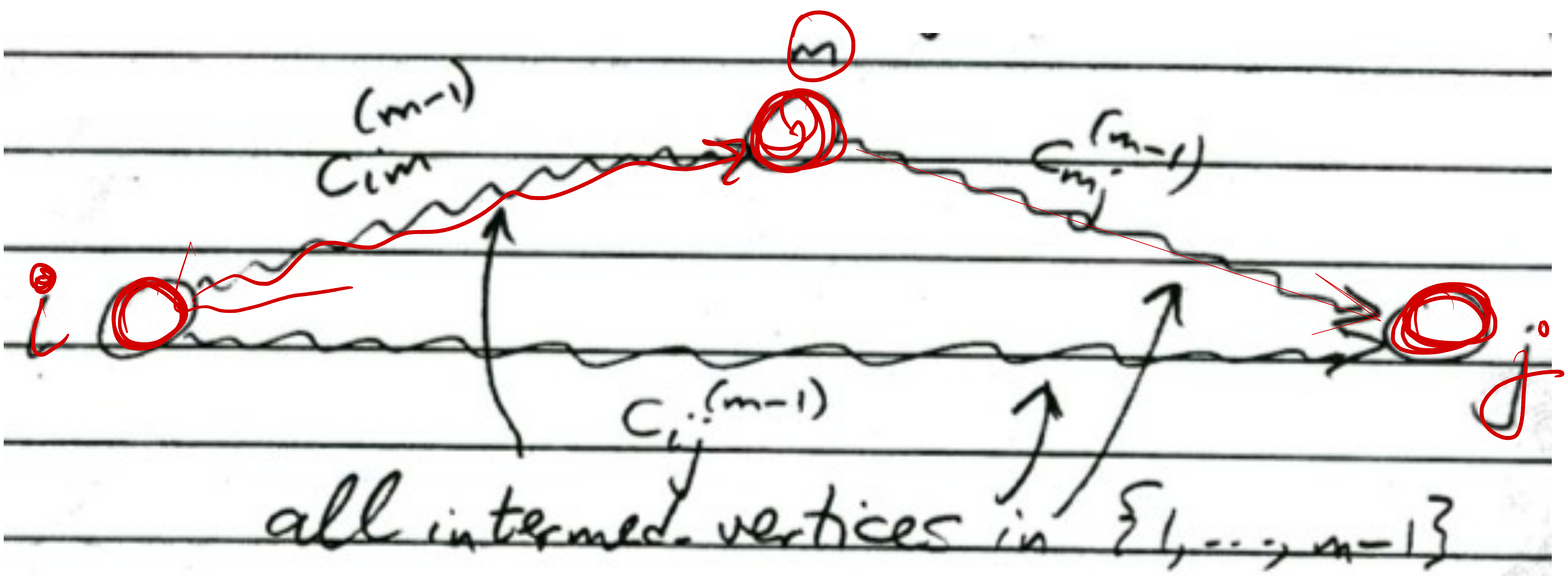
- "Floyd-Warshall" algorithm
- Fix a vertex order : 1, 2, 3, … ,n   Knapsack trick.
  - S_m= set first k of vertices = {v_1, v_2, … , v_m)
    the first m,
- $C_{ij}^{(m)}$ = the weight of SP(i,j) going only through intermediary vertices in set S_m



- m=0 : no intermediary allowed; $C_{ij}^{(0)}=w_{ij}$
- m=1 : only $k=v_1$ intermediary allowed
  - $C_{ij}^{(1)}= \min \{w_{ij} , w_{ik}+w_{kj} \}$

ASSP dynamic programming by vertices

- dynamic recursion
- $c_{ij}^{(m)} = \min\{ \overset{\text{prev stage}}{c_{ij}^{(m-1)}}, \underbrace{c_{im}^{(m-1)} + c_{mj}^{(m-1)}}_{\text{use } m^{th} \text{ vertex}} \}$ $\quad m-1 \longrightarrow m$
  - $c_{ij}^{(m)}$ = minimum between $c_{ij}^{(m-1)}$ and the SP including vertex v_m and only other intermediaries <m.



ASSP dynamic programming by vertices

- bottom up computation

```
▸ Floyd-Warshall-ASSP(W)
    ▸ for m=1:n
        ▸ for i=1:n
            ▸ for j=1:n
                ▸ c_ij^(m) = min{ c_ij^(m-1), c_im^(m-1) + c_mj^(m-1) }
    ▸ return c^(n)
```

$$c_{ij}^{(m)} = \min\{\, c_{ij}^{(m-1)},\ \underbrace{c_{im}^{(m-1)} + c_{mj}^{(m-1)}}_{\text{passing thru } m \text{ vertex}} \,\}$$

stage m

cell

prev stage

prev stage $i \to m$

prev stage $m \to j$

passing thru m vertex

- Running time $\Theta(V^3)$
  - for dense graphs $E \approx V^2$, Floyd-Warshall-ASSP same cost as Bellman-Ford-SSSP

Transitive Closure

$$T_{ij} = \begin{cases} 1 & \text{if path } i \leadsto j \\ 0 & \text{if not} \end{cases}$$

← adj matrix

$$T^m_{ij} = \begin{cases} 1 & \text{if } \exists \text{ path } i \leadsto j \text{ at most } m \text{ edges} \\ 0 & \text{if not} \end{cases}$$

T^1_{ii} = adj matrix