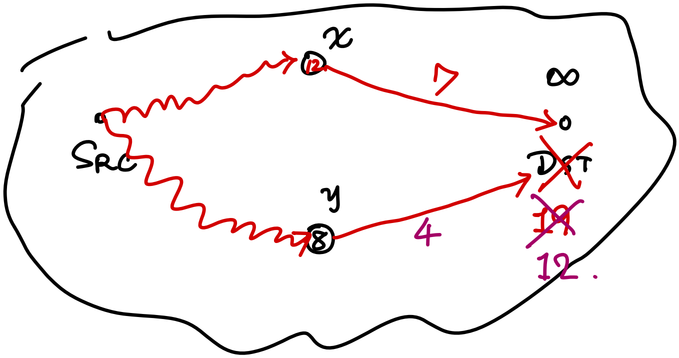
Shortest Path



finding shortest path contains shortest paths

- by relaxing an edge: improving shortest path of a vertex.

Restate:

- Input: Directed graph $G = \langle V, E \rangle$.

  weight fn. $w : E \to \mathbb{R}$.

  weight of path: $p = \langle v_0, v_1, \cdots, v_n \rangle$

  $= \sum_{i=1}^{k} w(v_{i-1}, v_i)$

  Shortest path u to v:

$$\delta(u, v) = \begin{cases} \min\{ w(p) ; u \overset{p}{\leadsto} v \} & , \exists u \overset{p}{\leadsto} v. \\ \infty & , \text{o.w.} \end{cases}$$

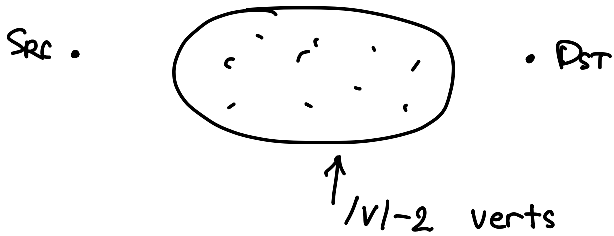
Variances

▷ Single src ▷ Single dest ▷ single pair

▷ All pairs.

Negative weight edges: OK, but no negative cycles reachable.

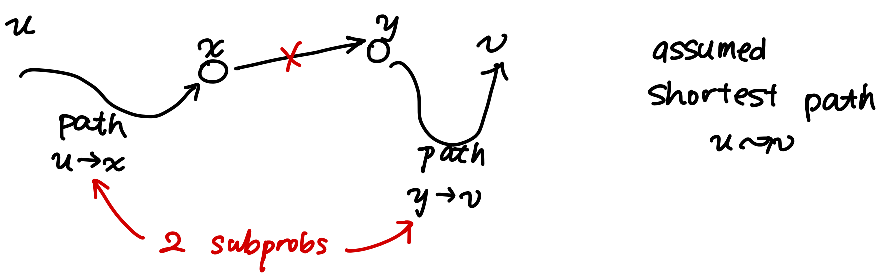
Naive approach . Try all possibilities



Worst case, $(|V|-2)(|V|-3)\cdots 1 \simeq n! \simeq$ exponential

Optimal Substructure for SP problem.

- Assume we have an optimal answer to the problem,
- Remove sth. from that ans,
- should yield subproblems
- show the original optimal ans, contains optimal ans to subproblems.



by contrad. If $\exists\, w'(u,x) < w(u,x)$.
then original answer is not optimal

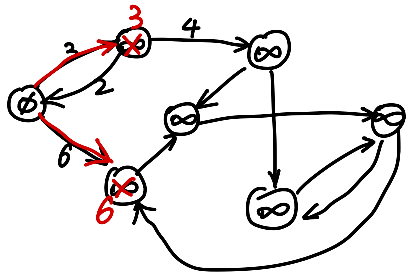
Shortest path contain shortest subpaths.

Output of single source shortest path algo

- $d[v] :=$ current $\delta(s,v)$
- $\pi[v] :=$ pred of shortest path.

▷ INIT

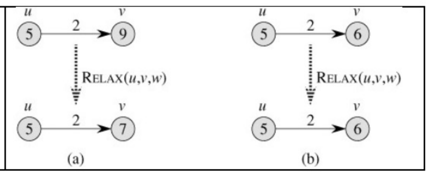
```
INIT-SINGLE-SOURCE(V, s)
for each v ∈ V
    d[v]←∞
    π[v] ← NIL
d[s] ← 0
```



▷ INCREMENTAL IMPROVEMENT : Relaxation

O(1)

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



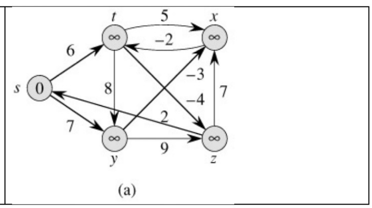
Algorithm differ in order the time it relaxes.

#1. Bellman-Ford

Simply relax all of them ! (in any fixed order)

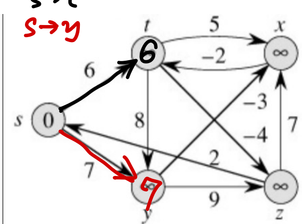
if d(src)<0 stop

```
BELLMAN-FORD(V, E, w, s)
  INIT-SINGLE-SOURCE(V, s)
  for i ← 1 to |V|-1
    for each edge (u, v) ∈ E
      RELAX(u,v,w)
  for each edge (u, v) ∈ E  // all edges, in any order, same order each time
    if d[v] > d[u] + w(u, v)
      then return FALSE
  return TRUE
```



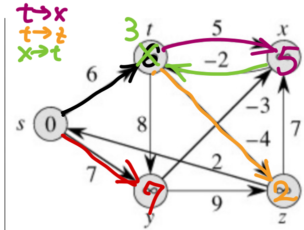
3. Execute Bellman-Ford on the above graph from source s for this edge order (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y). Update the d[v] and π[v] values for each iteration.

s→t
s→y



RELAX #1

t→x
t→z
x→t

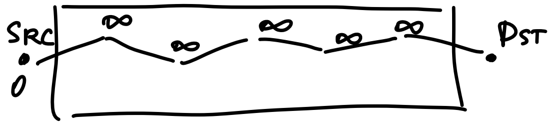


Relax No. 2.

. . .

Why should we do so many times?

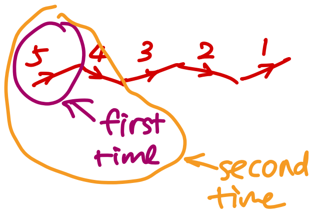
$|V|-1$ is the longest path for the verticies.



$|V|-1$ edges

- Now relax all edges, but what order to do?

Yes! only once.



do many times.

If that fixed order of edges is the rev of edges on the largest path, then I need $|V|-1$ sets of relax all edges.

(a prf why it works by pigeonhole principle).

That's why we have to do it for the last time.

- Time complexity : $O(V^3)$.
- Proof of correctness.

If we found a better order to relax edges, we might not need to do it $|V|-1$

If no changes, STOP. (successfully found!).

DAG Shortest Path Algorithm

Topo sort finds dependency order, how about use this order?

```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices of G
  INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
      do for each vertex v ∈ Adj[u]
          do RELAX(u, v, w)
```

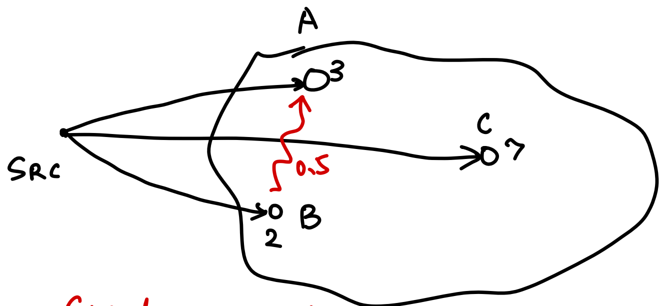
Restriction

- no cycle

Runtime: $\underbrace{O(V+E)}_{\text{toposort}} + \underbrace{O(V^2)}_{\text{relax edge}}$

Correct: because vertices are ordered by dependency. we have final d value for a vertex before we relax edges leaving that vertex.

4. If we restrict the graph to having no negative edges, given a source s, what is the shortest path from s to one of its adjacent vertexes?



The shortest path from SRC → A is 2.5 instead of 3. Is it a counterexample?

Greedy appoarch if no negative edges with the smallest path is an actual shortest path

↪ Dijkestra: greedy algorithm.