Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

Lecture No 35

Dijkstra's Algorithm

Problem Statement

- Given a graph $G = (V, E)$ with a source vertex s, weight function w, edges are non-negative, i.e.,

  $w(u, v) \geq 0, \forall (u, v) \in E$

  The graph is directed, i.e., if $(u, v) \in E$ then (v, u) may or may not be in E.
- The objective is to find shortest path from s to every $u \in V$.

Approach

- A “cloud S” of vertices, beginning with s, will be constructed, finally covering all vertices of graph
- For each vertex v, a label d(v) is stored, representing distance of v from s in the subgraph consisting of the cloud and its adjacent vertices
- At each step
  - We add to the cloud the vertex u outside the cloud with the smallest distance label, d(u)
  - We update labels of the vertices adjacent to u

Mathematical Statement of Problem

Input Given graph $G(V, E)$ with source s, weights w

Assumption

- Edges non-negative, $w(u, v) \geq 0, \forall (u, v) \in E$
- Directed, if $(u, v) \in E$ then (v, u) may be in E

Objective: Find shortest paths from s to every $u \in V$

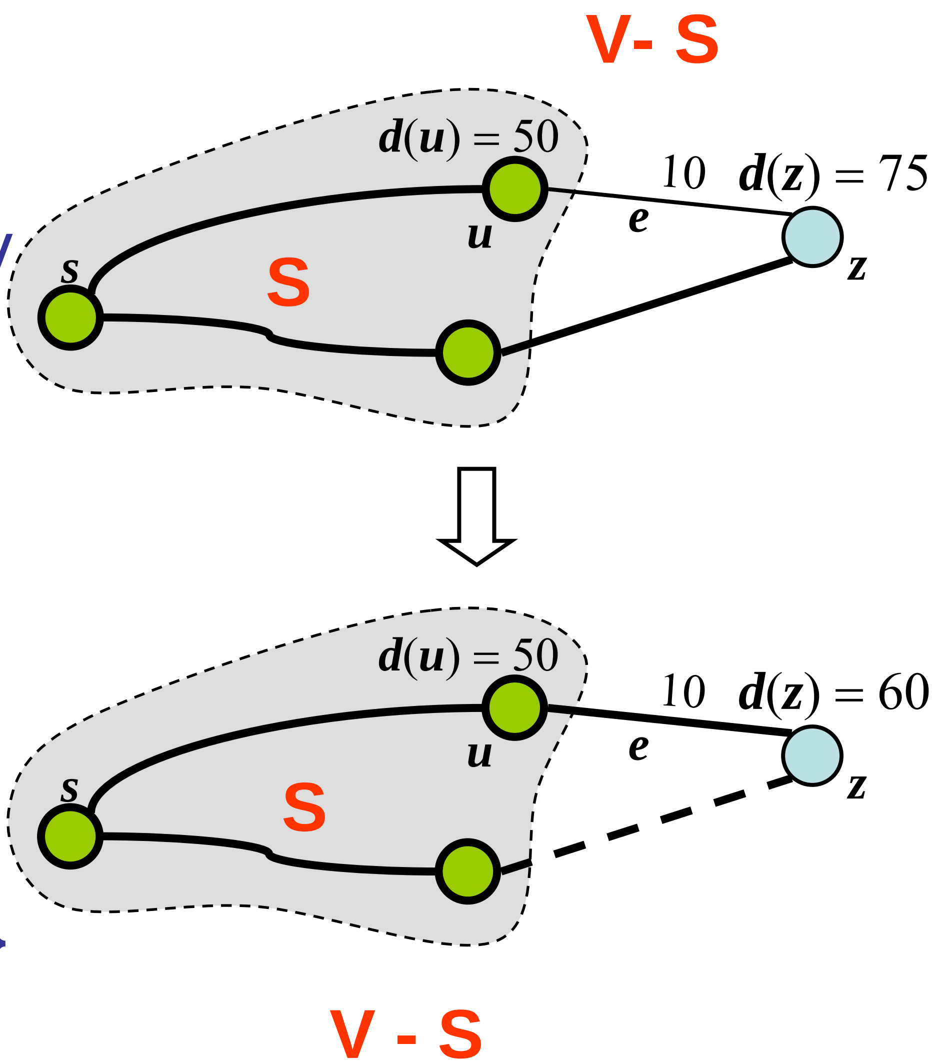
Approach

- Maintain a set S of vertices whose final shortest-path weights from s have been determined
- Repeatedly select, $u \in V - S$ with minimum shortest path estimate, add u to S, relax all edges leaving u.
- Greedy, always choose light vertex in V-S , add to S

Edge Relaxation

- Consider edge e = (u, z) such that
  - u is vertex most recently added to the cloud S
  - z is not in the cloud

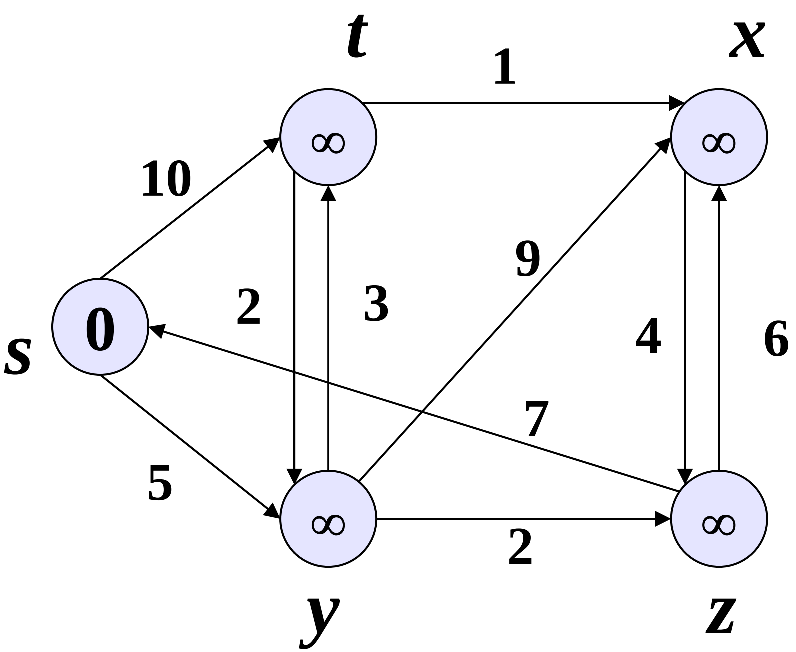
- Relaxation of edge e updates distance d(z) as
  d(z) =
  min {d(z), d(u) + weight(e)}



Dijkstra's Algorithm

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

Example: Dijkstra's Algorithm



For each vertex $v \in V(G)$

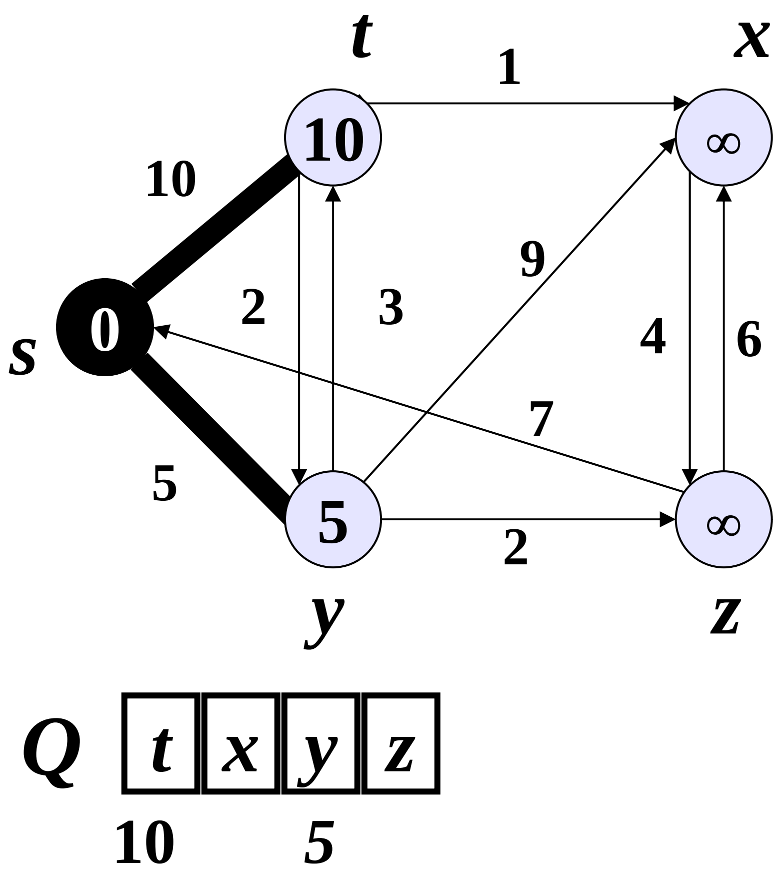
$$d[v] \leftarrow \infty$$
$$\pi[v] \leftarrow \text{NIL}$$

Considering s as root node

$$d[s] \leftarrow 0$$
$$S \leftarrow \emptyset$$

Q	*s*	*t*	*x*	*y*	*z*
	0				

Example: Dijkstra's Algorithm



s is extracted form queue

$S \leftarrow S \cup \{s\}$

$Adj[s] = t, y$

$d[t] > d[s] + w(s, t)$

$(\infty > 0 + 10)$

$d[t] \leftarrow d[s] + w(s, t)$

$0 + 10 = 10$

$\pi[t] \leftarrow s$

$d[y] > d[s] + w(s, y)$

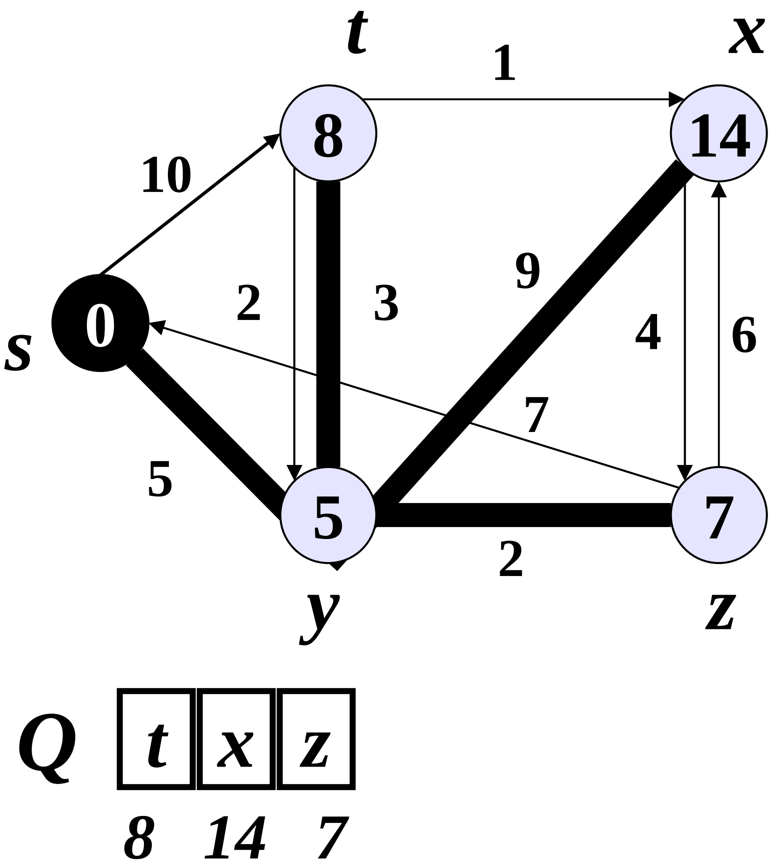
$(\infty > 0 + 5)$

$d[y] \leftarrow d[s] + w(s, y)$

$0 + 5 = 5$

$\pi[y] \leftarrow s$

Example: Dijkstra's Algorithm



y is extracted form queue

$S \leftarrow S \cup \{y\}$

$Adj[y] = t, x, z$

$d[t] > d[y] + w(y, t)$

$(10 > 5 + 3)$

$d[t] \leftarrow d[y] + w(y, t)$

$5 + 3 = 8$

$\pi[t] \leftarrow y$

$d[x] > d[y] + w(y, x)$

$(\infty > 5 + 9)$

$d[x] \leftarrow d[y] + w(y, x)$

$5 + 9 = 14$

$\pi[x] \leftarrow y$

$d[z] > d[y] + w(y, z)$

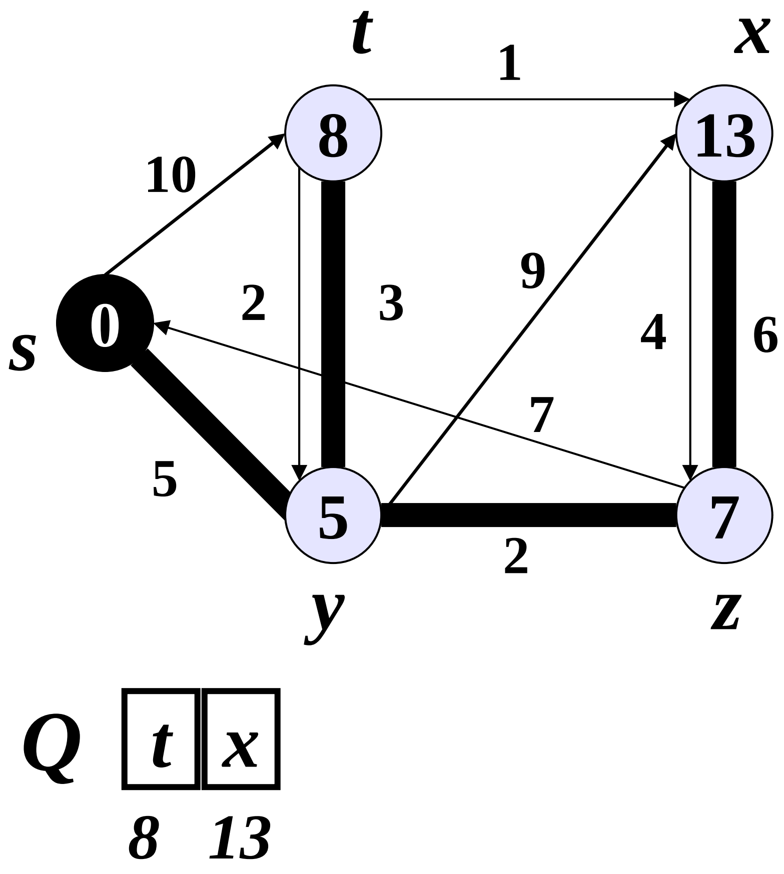
$(\infty > 5 + 2)$

$d[z] \leftarrow d[y] + w(y, z)$

$5 + 2 = 7$

$\pi[z] \leftarrow y$

Example: Dijkstra's Algorithm

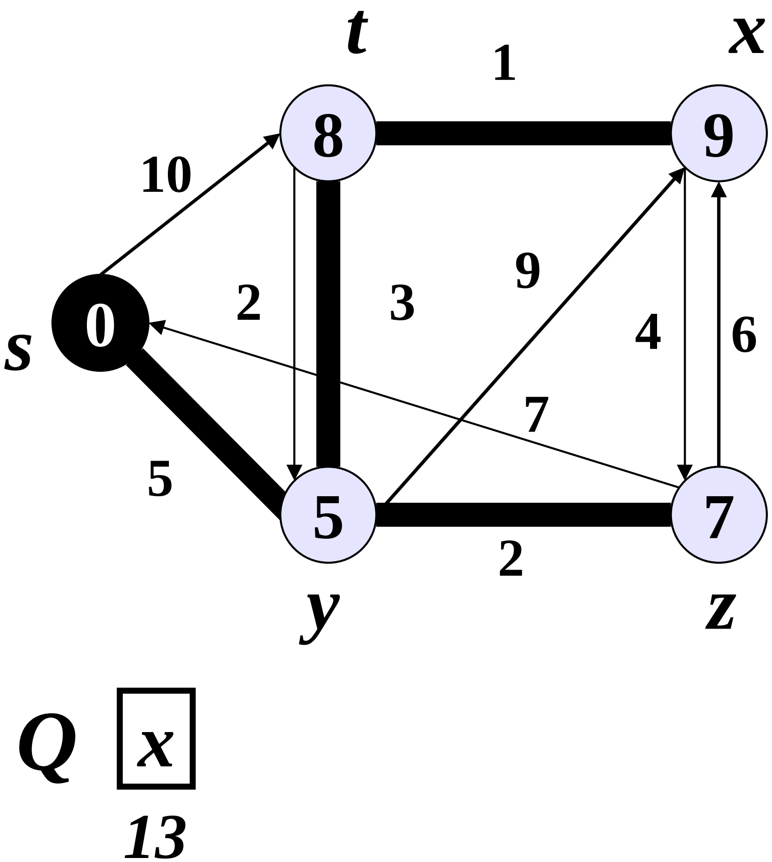


z is extracted form queue
$S \leftarrow S \cup \{z\}$
$Adj[z] = s, x$

$d[s] > d[z] + w(s, z)$
But $(0 < 7 + 7)$

$d[x] > d[z] + w(z, x)$
$(14 > 7 + 6)$
$d[x] \leftarrow d[z] + w(z, x)$
$7 + 6 = 13$
$\pi[x] \leftarrow z$

Example: Dijkstra's Algorithm



t is extracted form queue

$S \leftarrow S \cup \{t\}$

$Adj[t] = x, y$

$d[x] > d[t] + w(t, x)$

$(13 > 8 + 1)$

$d[x] \leftarrow d[t] + w(t, x)$

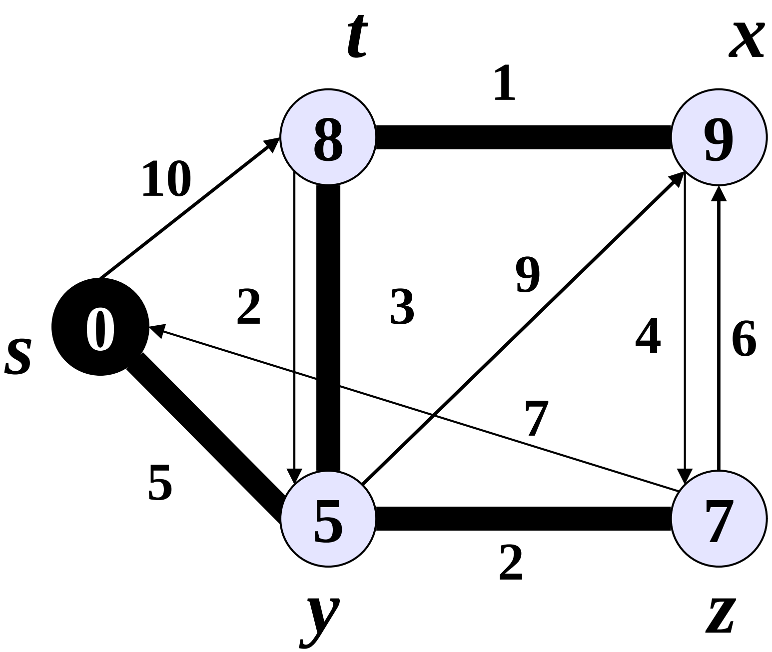
$8 + 1 = 9$

$\pi[x] \leftarrow t$

$d[y] > d[t] + w(t, y)$

But $(5 < 8 + 3)$

Example: Dijkstra's Algorithm



Q $\boxed{\varnothing}$

x is extracted form queue

$S \leftarrow S \cup \{x\}$

$Adj[x] = z$

$d[z] > d[x] + w(x, z)$

But $(7 < 9 + 4)$

Analysis: Dijkstra's Algorithm

Cost depends on implementation of min-priority queue

Case 1:

Vertices being numbered 1 to $|V|$

- INSERT, DECREASE-KEY operations takes $O(1)$
- EXTRACT-MIN operation takes $O(V)$ time
- ***Sub cost is $O(V^2)$***
- **Total number of edges in all adjacency list is |E|**
- **Total Running time = $O(V^2 + E) = O(V^2)$**

Analysis: Dijkstra's Algorithm

Case 2:

Graph is sufficiently spare, e.g., $E = O(V^2/\lg V)$

Implement min-priority queue with binary min heap

Vertices being numbered 1 to $|V|$

- Each EXTRACT-MIN operation takes $O(\lg V)$
- There $|V|$ operations, time to build min heap $O(V)$
- ***Sub cost is $O(V \lg V)$***
- *Each DECREASE-KEY operation takes time $O(\lg V)$, and there are $|E|$ such operation.*
- ***Sub cost is $O(E \lg V)$***

Hence Total Running time = $O(V + E) \lg V = E \lg V$

Analysis: Dijkstra's Algorithm

Case 3:

Implement min-priority queue with Fibonacci heap

Vertices being numbered 1 to |*V*|

- Each EXTRACT-MIN operation takes *O(lgV)*
- There |V| operations, time to build min heap O(V)
- ***Sub cost is O(V lgV)***
- *Each DECREASE-KEY operation takes time O(1), and there are |E| such operation.*
- ***Sub cost is O(E)***

Hence Total Running time = *O (V.lgV + E) = V* lgV

Case 1: Computation Time

1. INITIALIZE-SINGLE-SOURCE(V, s) $\leftarrow$ $\Theta(V)$
2. S $\leftarrow \varnothing$
3. Q $\leftarrow$ V[G] $\leftarrow$ O(V) build min-heap
4. **while** Q $\neq \varnothing$ $\leftarrow$ O(V)
5. **do** u $\leftarrow$ EXTRACT-MIN(Q) $\leftarrow$ O(V)
6. S $\leftarrow$ S $\cup$ {u}
7. **for** each vertex v $\in$ Adj[u] $\leftarrow$ O(E)
8. **do** RELAX(u, v, w)

Running time: $O(V^2 + E) = O((V^2)$

Note:

Running time depends on Impl. Of min-priority (Q)

Case 2 : Binary min Heap

1. INITIALIZE-SINGLE-SOURCE(V, s) ← $\Theta(V)$
2. S ← $\emptyset$
3. Q ← V[G] ← O(V) build min-heap
4. **while** Q $\neq \emptyset$ ← Executed O(V) times
5. **do** u ← EXTRACT-MIN(Q) ← O(lgV)
6. S ← S $\cup$ {u}
7. **for** each vertex v $\in$ Adj[u]
8. **do** RELAX(u, v, w) ← O(E) times O(lgV)
9. Running time: O(VlgV + ElgV) = O(ElgV)

Case 3 : Fibonacci Heap

1. INITIALIZE-SINGLE-SOURCE(V, s) $\leftarrow$ $\Theta(V)$
2. S $\leftarrow \varnothing$
3. Q $\leftarrow$ V[G] $\leftarrow$ O(V) build min-heap
4. **while** Q $\neq \varnothing$ $\leftarrow$ Executed O(V) times
5. **do** u $\leftarrow$ EXTRACT-MIN(Q) $\leftarrow$ O(lgV)
6. S $\leftarrow$ S $\cup$ {u}
7. **for** each vertex v $\in$ Adj[u]
8. **do** RELAX(u, v, w) $\leftarrow$ O(E) times O(1)
9. Running time: O(VlgV + E) = O(VlgV)

Theorem : Correctness of Dijkstra's Algorithm

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

Proof

- We use the following loop invariant:
  - At start of each iteration of the **while** loop of lines 4-8, $d[v] = \delta(s, v)$ for each vertex $v \in S$.

Contd..

- It suffices to show for each vertex $u \in V$, we have $d[u] = \delta(s, u)$ at the time when u is added to set S.
- Once we show that $d[u] = \delta(s, u)$, we rely on the upper-bound property to show that the equality holds at all times thereafter.

Initialization:

- Initially, $S = \emptyset$, and so the invariant is trivially true

Maintenance:

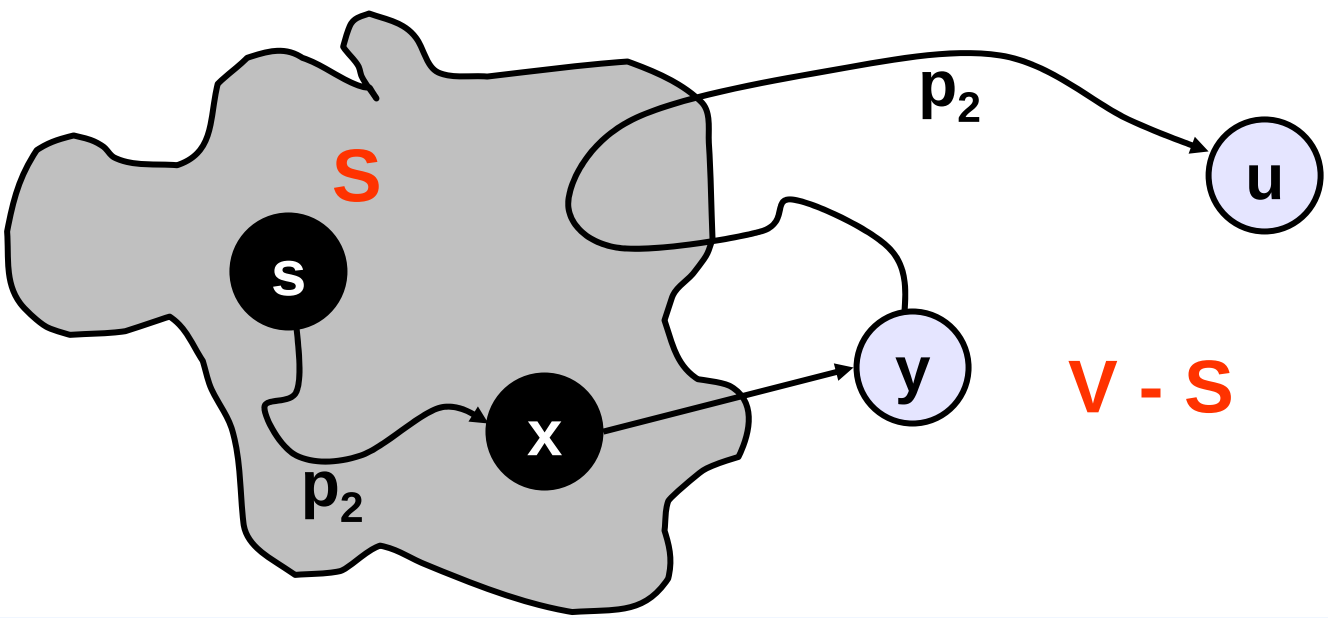
- We wish to show that in each iteration, $d[u] = \delta(s, u)$, for the vertex added to set S.

Contd..

- On contrary suppose that $d[u] \neq \delta(s, u)$ when u is added to set S. Also suppose that *u is* the first vertex for which the equality does not hold.
- We focus on situation at beginning of **while** loop in which u is added to S and derive a contradiction.
- First of all, $u \neq s$ because s is the first vertex added to set S and $d[s] = \delta(s, s) = 0$ at that time.
- Secondly S ≠ Ø just before u is added to S, this is because s is at least in S.
- There must be some path from s to u, otherwise d[u] = δ(s, u) = ∞ by no-path property, which would violate our assumption that d[u] ≠ δ(s, u).

Contd..

- Because there is at least one path, there must be a shortest path *p* from *s* to *u*.
- Prior to adding *u* to *S*, path *p* connects a vertex in *S*, namely *s*, to a vertex in *V* - *S*, namely *u*.



Contd..

- Let us consider the first vertex *y* along *p* such that $y \in V - S$, and let $x \in S$ be *y*'s predecessor.
- Thus, path *p* can be decomposed: $s \overset{p_1}{\leadsto} x \to y \overset{p_2}{\leadsto} u$ (either of paths *p*1 or *p*2 may have no edges.)
- We claim that $d[y] = \delta(s, y)$ when *u* is added to *S*.

Proof of Claim: observe that $x \in S$.

- Because *u* is chosen as the first vertex for which $d[u] \neq \delta(s, u)$ when it is added to *S*, we had $d[x] = \delta(s, x)$ when *x* was added to *S*.
- Edge (x, y) was relaxed at that time, and hence $d[y] = \delta(s, y)$ (convergence property).

Contd..

- Because y occurs before u on a shortest path from s to u and all edge weights are nonnegative (on path p2), we have $\delta(s, y) \leq \delta(s, u)$,
- Now $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u] \Rightarrow d[y] \leq d[u]$ (1)
- But because both vertices u and y were in V - S when u was chosen, we have $d[u] \leq d[y]$. (2)
- From (1) and (2), $d[u] = d[y]$
- Now, $d[y] = \delta(s, y) \leq \delta(s, u) = d[u] = d[y]$

  $\Rightarrow \delta(s, y) = \delta(s, u)$.
- Finally, $d[u] = \delta(s, u)$, it contradicts choice of u
- Hence, $d[u] = \delta(s, u)$ when u is added to S, and this equality is maintained at all times after that

Contd..

Termination:

- At termination, $Q = \varnothing$ which, along with our earlier invariant that $Q = V - S$, implies that $S = V$.
- Thus, $d[u] = \delta(s, u)$ for all vertices $u \in V$.

Lemma 1

Statement

- Let G = (V, E) be a weighted, directed graph with weight function $w : E \rightarrow R$, let $s \in V$ be a source vertex. Assume that G contains no negative-weight cycles reachable from s. Then, after the graph is initialized by INITIALIZE-SINGLE-SOURCE(G, s), the predecessor sub-graph G_π forms a rooted tree with root s, and any sequence of relaxation steps on edges of G maintains this property as an invariant.

Proof

- Initially, the only vertex in G_π is the source vertex, and the lemma is trivially true.

Contd..

- Let Gπ be a predecessor subgraph that arises after a sequence of relaxation steps.

a. First we prove that Gπ is a rooted tree.

1. G_π is acyclic

- On contrary suppose that some relaxation step creates a cycle in the graph Gπ .
- Let $c = \langle v_0, v_1, \ldots, v_k \rangle$ be cycle, where $v_k = v_0$.
- Then, $\pi[v_i] = v_{i-1}$ for $i = 1, 2, \ldots, k$
- Now, without loss of generality, we can assume that it was the relaxation of edge (v_{k-1}, v_k) that created the cycle in G_π.

Contd..

Claim: all vertices on cycle c reachable from s.

- Because each vertex has non-NIL predecessor, and it was assigned a finite shortest-path estimate when it was assigned non-NIL π value
- By upper-bound property, each vertex on c has a finite shortest-path weight, and reachable from s.

Shortest-path on c just prior RELAX(v_{k-1}, v_k, w)

- Just before call, $\pi[v_i] = v_{i-1}$ for i = 1, 2,..., k - 1.
- Thus, for i = 1, 2,..., k - 1, last update to $d[v_i]$ was $d[v_i] \leftarrow d[v_{i-1}] + w(v_{i-1}, v_i)$.

Contd..

- It is obvious that, $d[v_k] > d[v_{k-1}] + w(v_{k-1}, v_k)$.
- Summing it with k - 1 inequalities,

$$\sum_{i=1}^{k} d[v_i] > \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

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

$$\text{But, } \sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

$$\text{Hence, } 0 > \sum_{1=1}^{k} w(v_{i-1}, v_i)$$

Contd..

- Thus, sum of weights around cycle c is negative, which provides the contradiction.
- We have proved that Gπ is a directed, acyclic.

2. To show that it forms a rooted tree with root s

- Sufficient to prove that $\forall$ v $\in$ Vπ, there is a unique path from s to v in Gπ.
- On contrary, suppose there are two simple paths from s to some vertex v, and (x ≠ y)

  p_1: $s \leadsto u \leadsto x \rightarrow z \leadsto v$

  p_2: $s \leadsto u \leadsto y \rightarrow z \leadsto v$

- π[z] = x and π[z] = y, $\Rightarrow$ x = y, a contradiction.

Contd..

- Hence there exists unique path in $G\pi$ from s to v.

  Thus $G\pi$ forms a rooted tree with root s.

b. Now by predecessor subgraph property

- $d[v] = \delta(s, v)$ for all vertices $v \in V$. Proved

Lemma 2

- If we run Dijkstra's algorithm on weighted, directed graph $G = (V, E)$ with nonnegative weight function w and source s, then at termination, predecessor subgraph $G\pi$ is a shortest paths tree rooted at s.

Proof:

- Immediate from the above lemma.