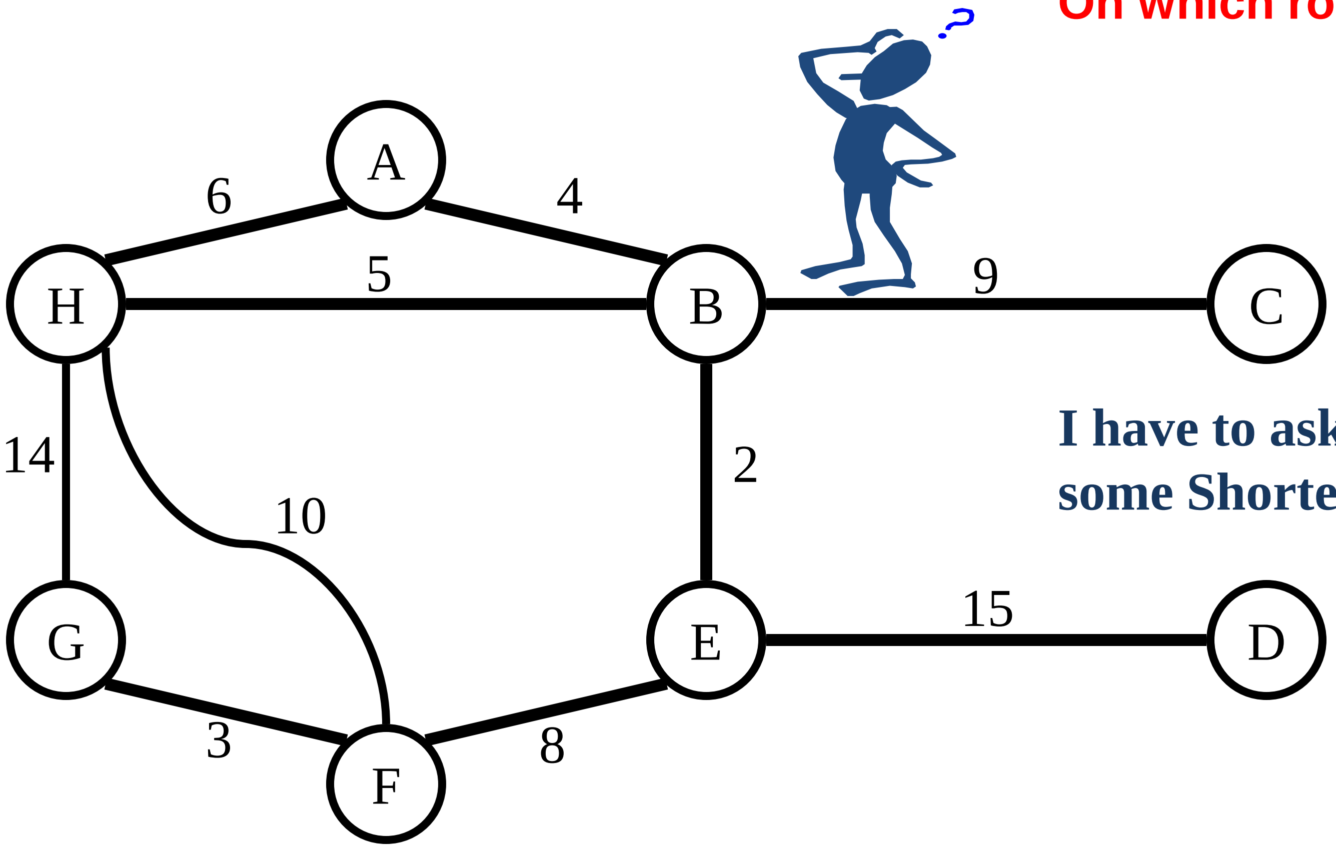
Shortest Path Algorithms

Dr Dayal Kumar Behera

Shortest Path



On which route I will go to G ?

I have to ask KIIT students for some Shortest Path Algorithms

Shortest Path

- Shortest path = a path of the minimum weight

- Applications of Shortest Path Algo.
  - static/dynamic network routing
  - robot motion planning
  - route generation in traffic
  - road map applications

Types of Shortest Path Problems

For a Graph G(V, E)

- **Single-Source Shortest Path:**

  Find a shortest path from a given source (vertex $s \in V$) to all of the vertices v $\in V$.

  (One source, Many Destinations)

- **Single-Destination Shortest Path:**

  Find shortest path to a given destination vertex

  (Many sources, One Destination)

- **Sigle-Pair Shortest Path:**

  Find shortest path from u to v.

  (One Source, One Destination)

- **All-Pair Shortest Path:**

  Find Shortest path from u to v, for all u, v $\in V$.

  (Many Sources, Many Destinations)

Shortest Path Weight

We define the ***shortest-path weight*** δ(u , v) from u to v by

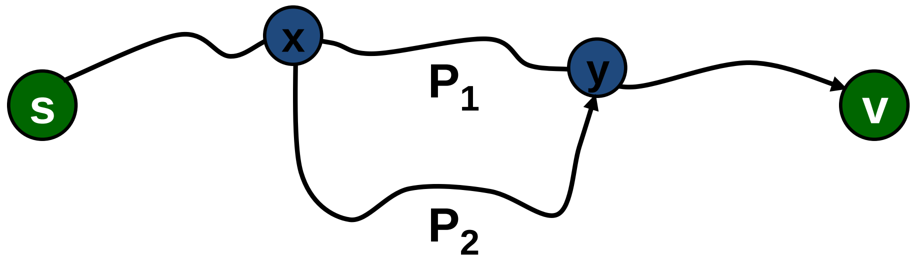
$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v \text{ ,} \\ \infty & \text{otherwise .} \end{cases}$$

The ***weight*** w(p) of path ‘p’ is the sum of the weights of its constituent edges over the path.

Shortest Path Properties

1. Optimal substructure property

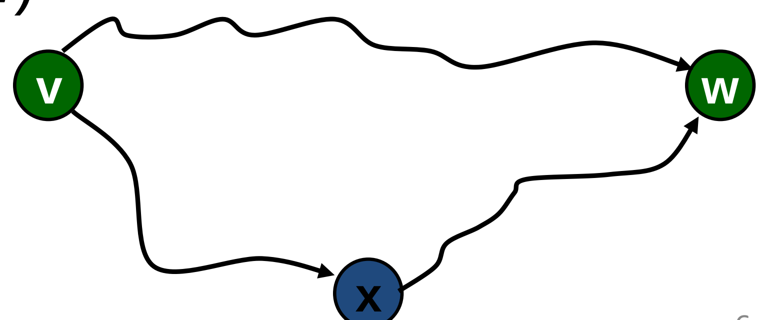
All sub-paths of shortest paths are also shortest paths.



2. Triangle inequality.

Let $\delta(v, w)$ or d(v, w) be the length of the shortest path from v to w.

Then, $d(v, w) \leq d(v, x) + d(x, w)$



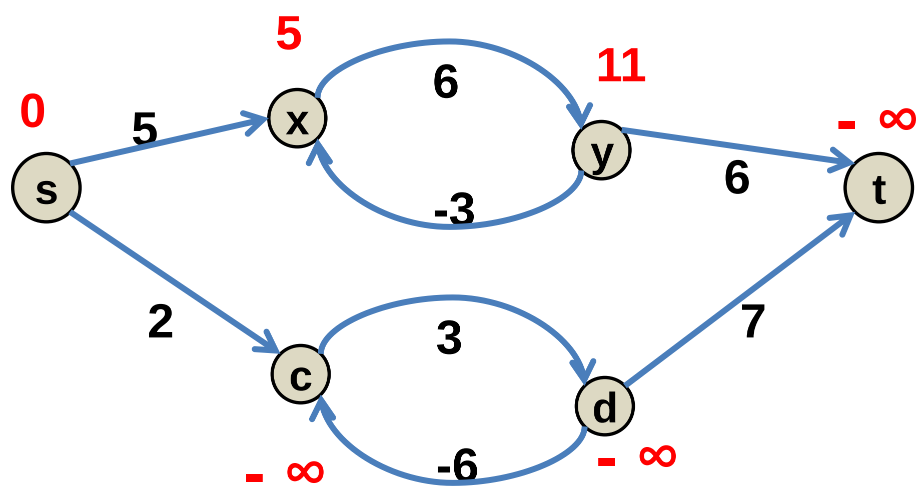
Basics

Negative Weight and Cycles

If there is a negative weight cycle on some path from s to t
Shortest Path weight can be defined as $\delta(s\ ,\ t) = -\ \infty$

Cycle <x, y, x> has weight
6+(-3) = 3 > 0
(positive weight cycle)

Cycle <c, d, c> has weight
3+(-6) = -3 < 0
(Negative weight cycle)



Shortest Path must not contain negative Cycles

Negative edges are OK, as long as there are no *negative weight cycles*

Different Shortest Path Algorithms

1. Single-Source Shortest Path Algorithms

   1. **Dijkstra's Algorithm:** doesn't allow negative edge weights — Greedy

   2. **Bellman-ford's Algorithm:** Allows negative edge weights and returns false if there is any negative weight cycles. — Dynamic Programming

2. All-Pair Shortest Path Algorithm

   - **Floyd-Warshall's algorithm:** allows negative edge weights, But assumption is that there is no negative weight cycles

Dynamic Programming

Initialization

All single source shortest path algorithms start with INITIALIZE-SINGLE-SOURCE routine

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

d[v] : shortest Path estimate

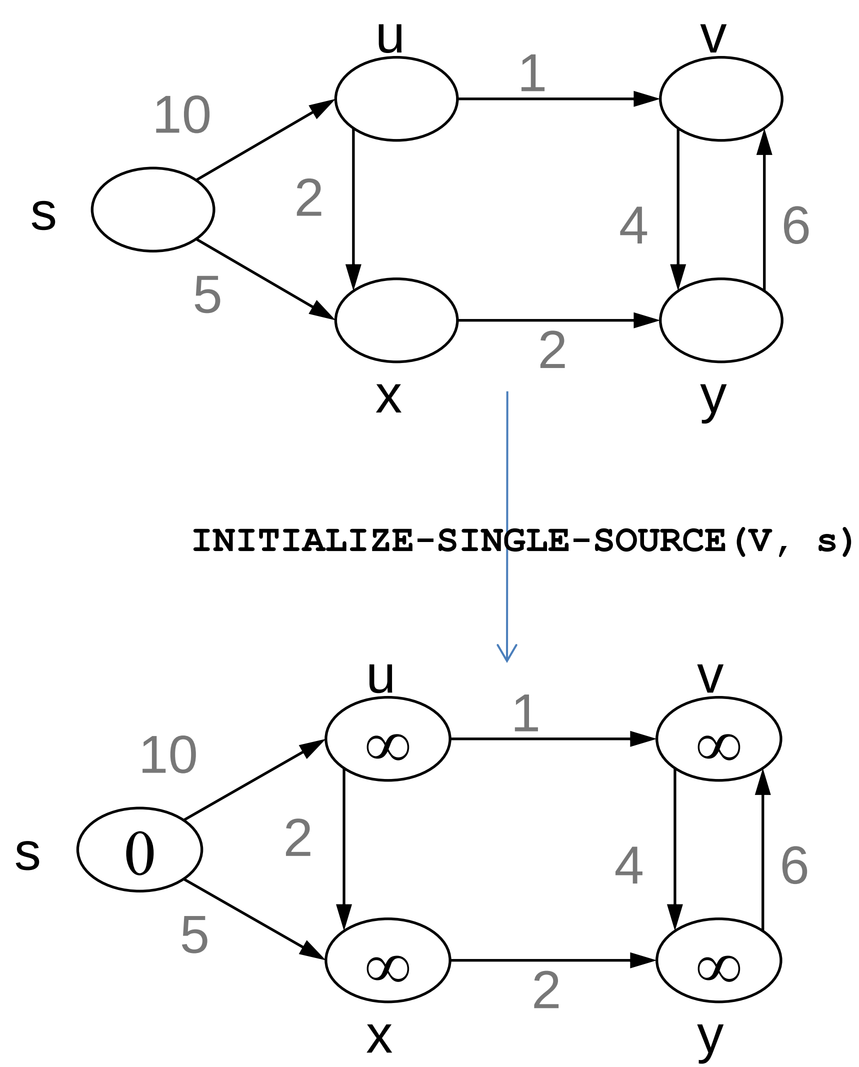
d[v] : $\delta(s, v)$

π[v] : predecessor of v on a shortest path from source vertex

If no predecessor then π[v]= NIL

Initialization(Example)

```
INITIALIZE-SINGLE-SOURCE(V, s)
for each v ∈ V[G] do
        d[v] ← ∞
        π [v] ← NIl
d[s] ← 0
```



Edge Relaxation

Relaxing an edge (u,v) means testing whether we can improve the shortest path to ‘v’ found so far by going through ‘u’.

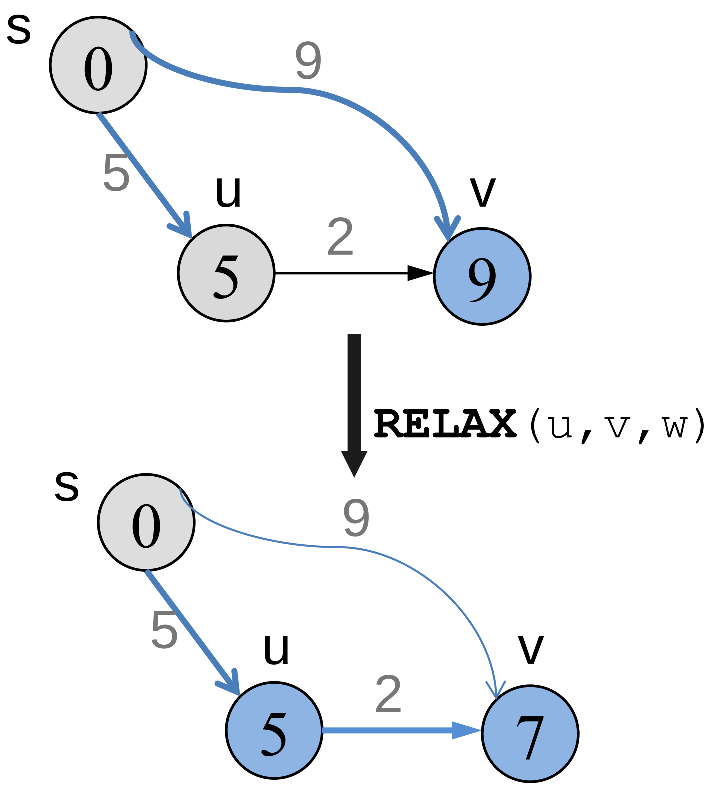
Edge Relaxation on edge (u,v) with weight w is as follows

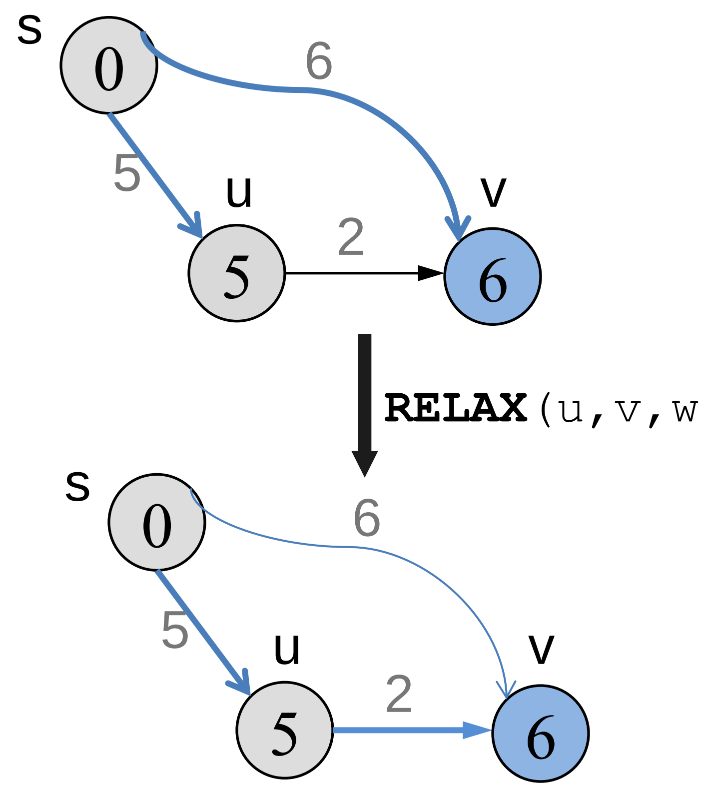
```
RELAX(u,v,w)
1. if d[v] > d[u]+w(u,v)then
2.     d[v] ← d[u]+w(u,v)
3.     π[v] ← u
```

Edge Relaxation(Example)

```
RELAX(u,v,w)
1. if 9 > 5 + 2 then
2.       d[v] ← 7
3.       π[v] ← u
```

```
RELAX(u,v,w)
1. if 6 > 5 + 2 then
```





Dijkstra’s Algorithm

- Solves single-source shortest path problems on a weighted graph.

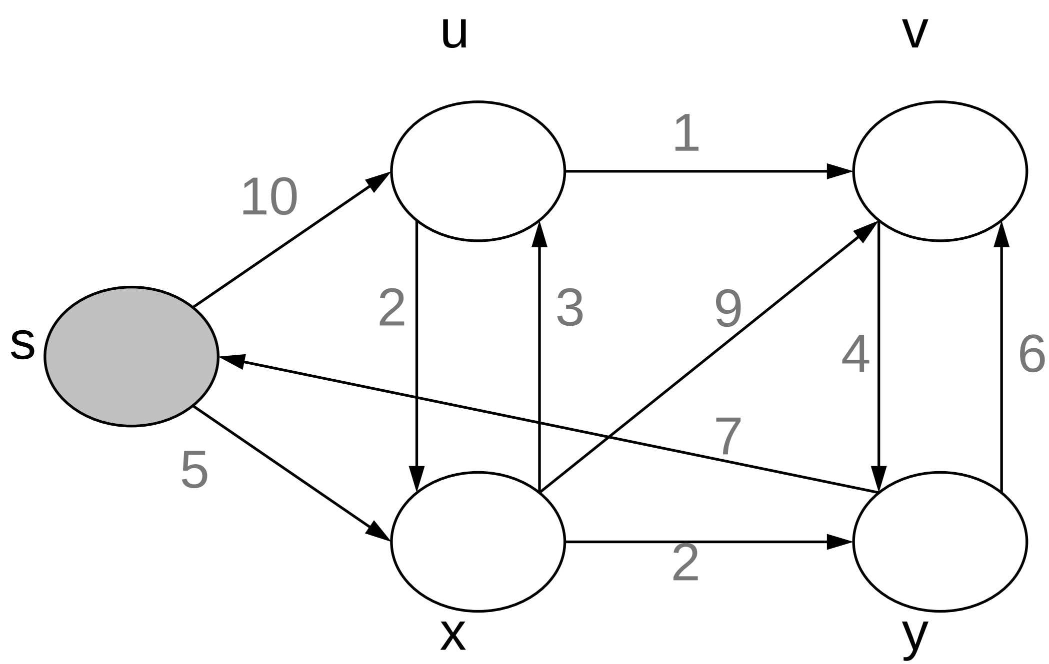
- Assumption: No Negative Edge Weights

- Basic Idea
  - maintain a set *S* of solved vertices
  - at each step select "closest" vertex *u*, add it to *S*, and relax all edges from *u*

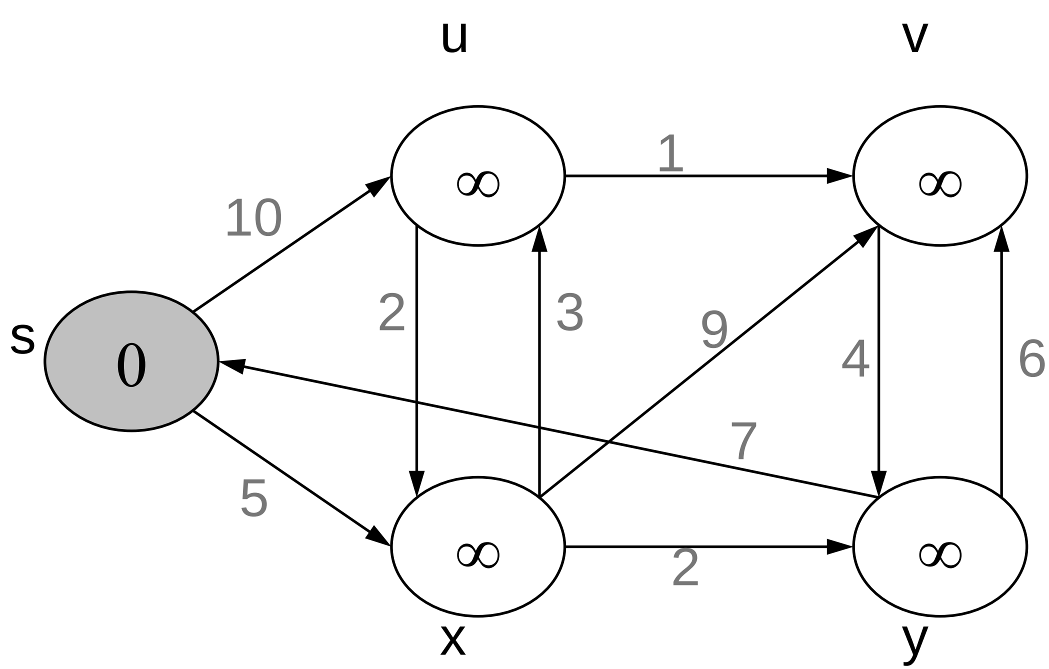
- Algorithm

```
DIJKSTRA(G, w, s)
1. INITIALIZE-SINGLE-SOURCE(G,s)
2. S ← { }
3. Q ← V[G]
4. While Q ≠ Φ do
5.           u ← EXTRACT-MIN(Q)
6.           S ← S ᴜ {u}
7.           for each vertex  v ε Adj [u] and v ε Q do
8.                     RELAX(u,v,w)
```

Dijkstra’s Algorithm(Example)



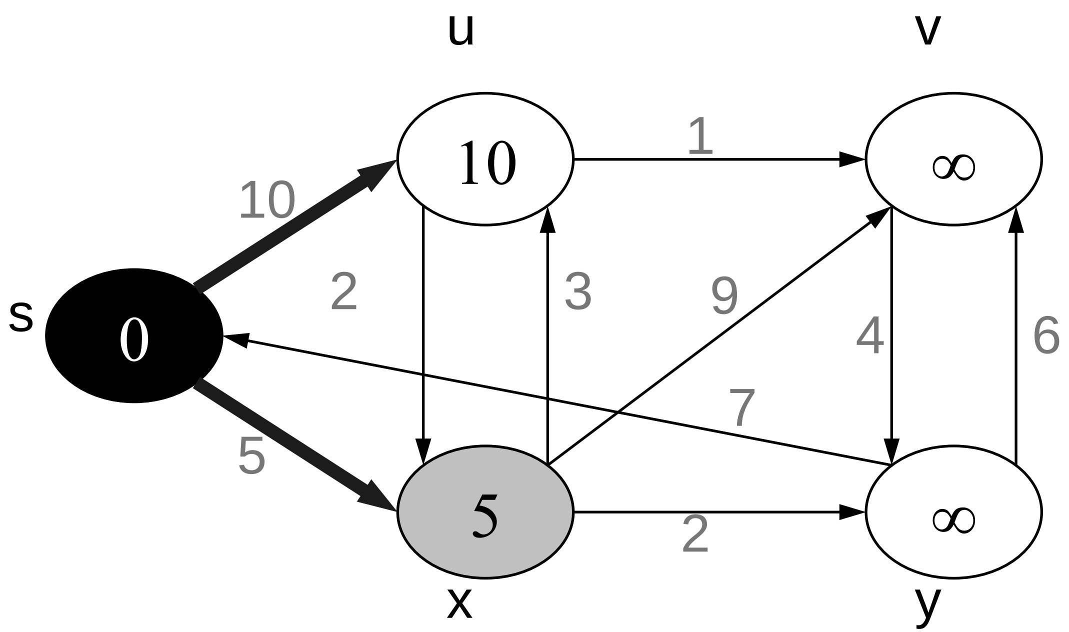
Dijkstra’s Algorithm(Example)



S : { }

Q : {s, u, v, x, y}

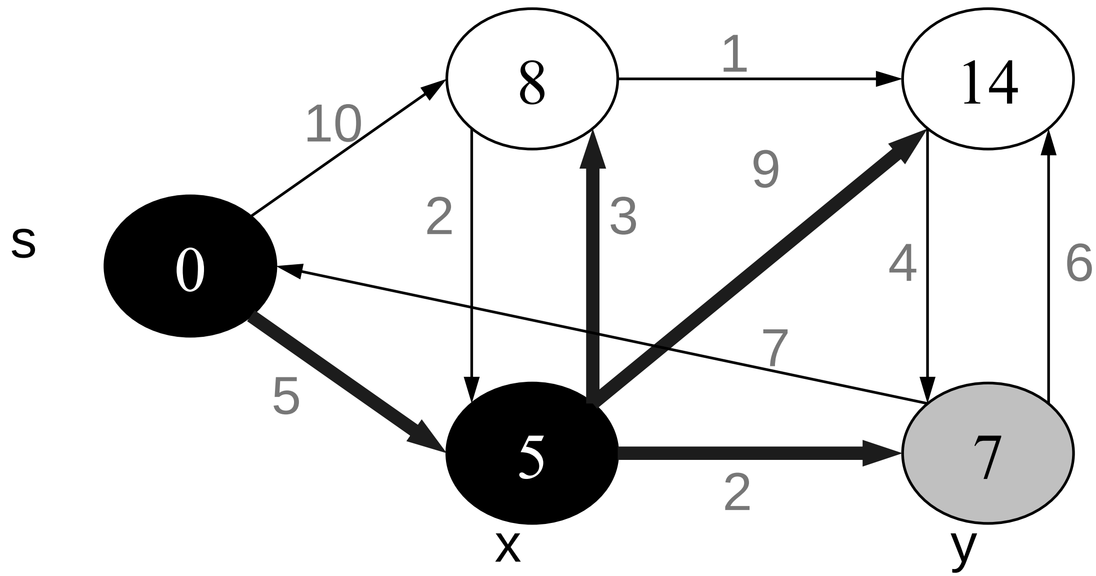
Dijkstra’s Algorithm(Example)



S : { s }

Q : {x, u, v, y}

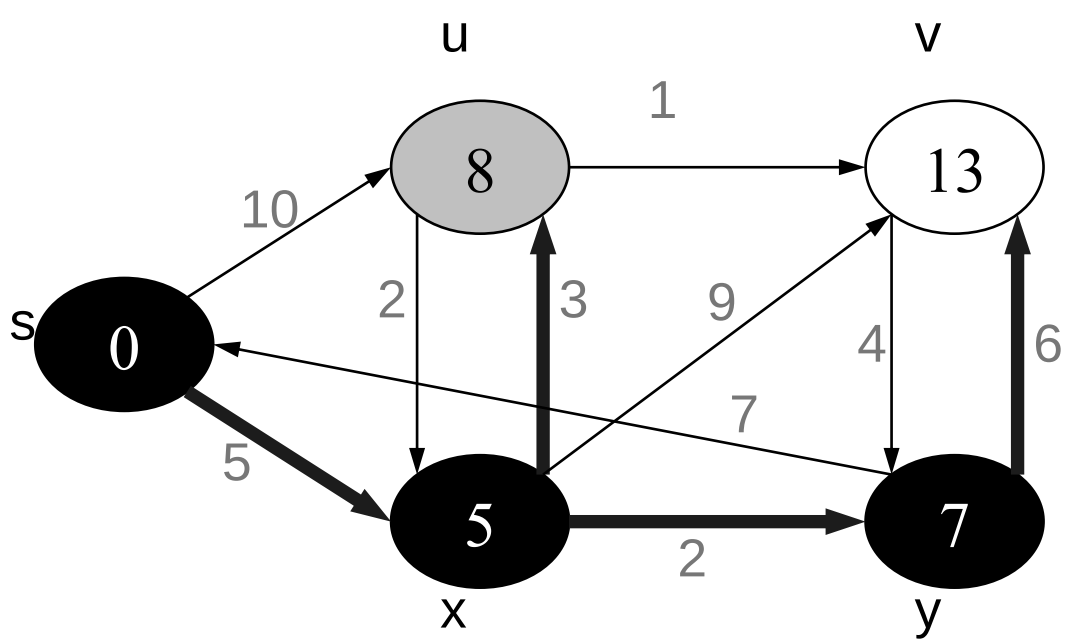
Dijkstra’s Algorithm(Example)



S : { s, x }

Q : { u, v, y }

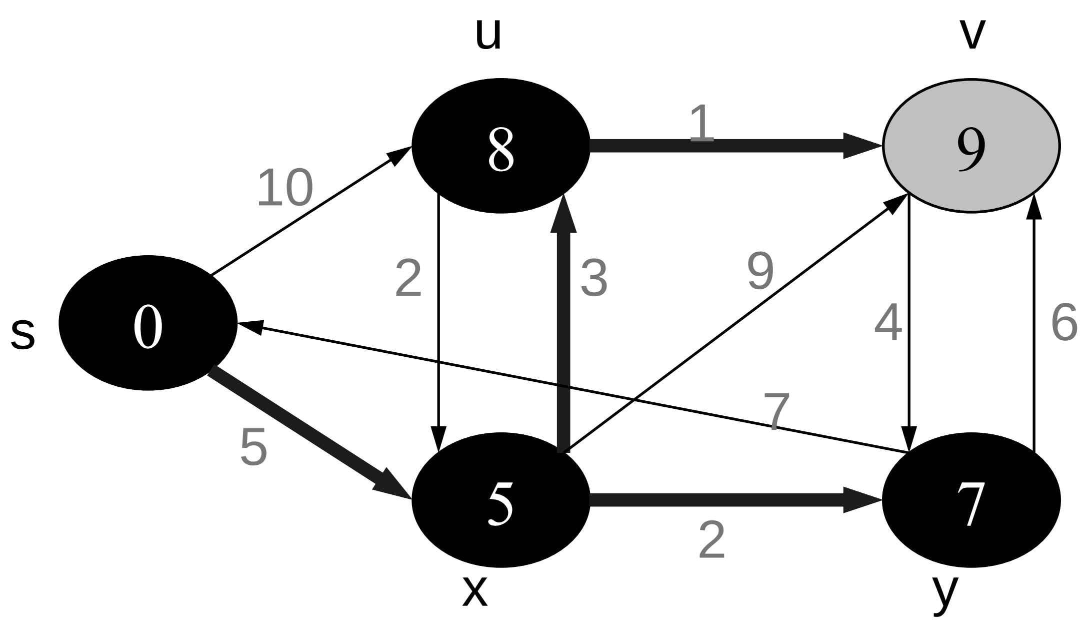
Dijkstra’s Algorithm(Example)



S : { s, x, y }

Q : { u, v }

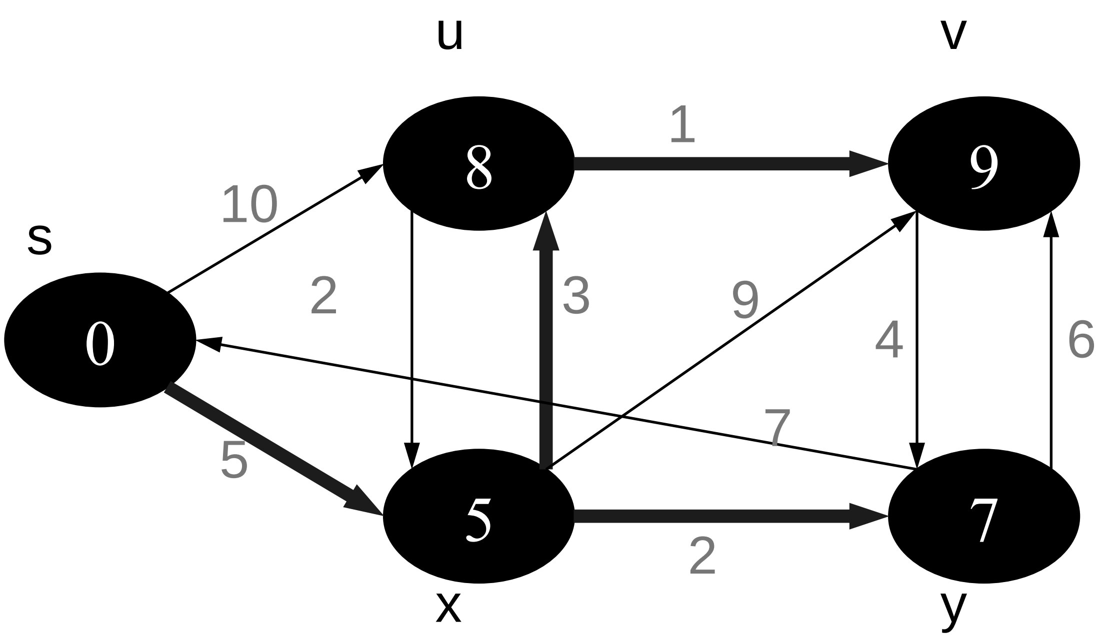
Dijkstra’s Algorithm(Example)



S : { s, x, y, u }

Q : { v }

Dijkstra’s Algorithm(Example)



S : { s, x, y, u , v}

Q : { }

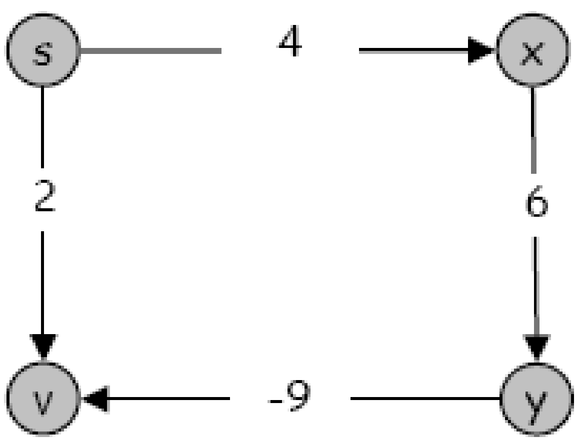
Now Q is empty.
So stop here

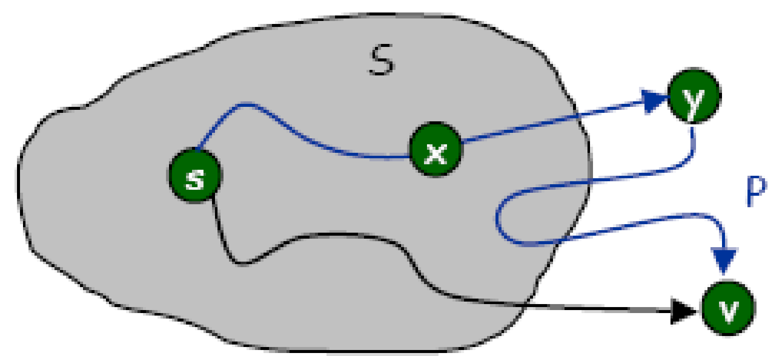
Dijkstra’s Algorithm(Weakness)

Dijkstra's Algorithm With Negative Costs

Dijkstra's algorithm fails if there are negative weights.

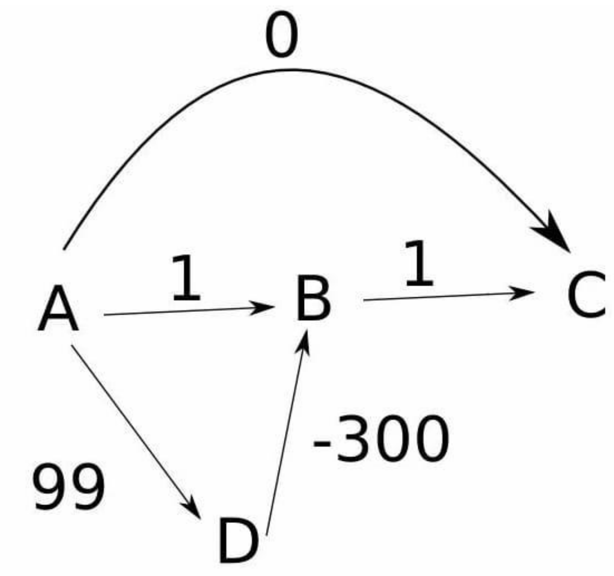
- Ex: Selects vertex v immediately after s.
  But shortest path from s to v is s-x-y-v.





Dijkstra proof of correctness breaks down since it assumes cost of P is nonnegative.

Dijkstra's Algorithm(Weakness)



Dijkstra's Algorithm(Analysis)

```
DIJKSTRA(G, w, s)
1. INITIALIZE-SINGLE-SOURCE(G,s)
2. S ← { }
3. Q ← V[G]
4. While Q ≠ Φ do
5.              u ← EXTRACT-MIN(Q)
6.              S ← S u {u}
7.              for each vertex  v ε Adj [u]  and v ε Q do
8.                    RELAX(u,v,w) > May cause
                                > DECREASE-KEY(Q, v, d[v])
```

O(V)

O(VlogV)

O(logV) } O(VlogV)

Total in loop O(ElogV)

We can observe that the statements in inner loop are executed O(V+E) times (similar to BFS).
The inner loop has decreaseKey() operation which takes O(logV) time.

So overall time complexity is O((E+V)*logV) = O(E logV)
Note that the above code uses Binary Heap for Priority Queue implementation.