Graphs
Topological Sort
Single Source Shortest Path

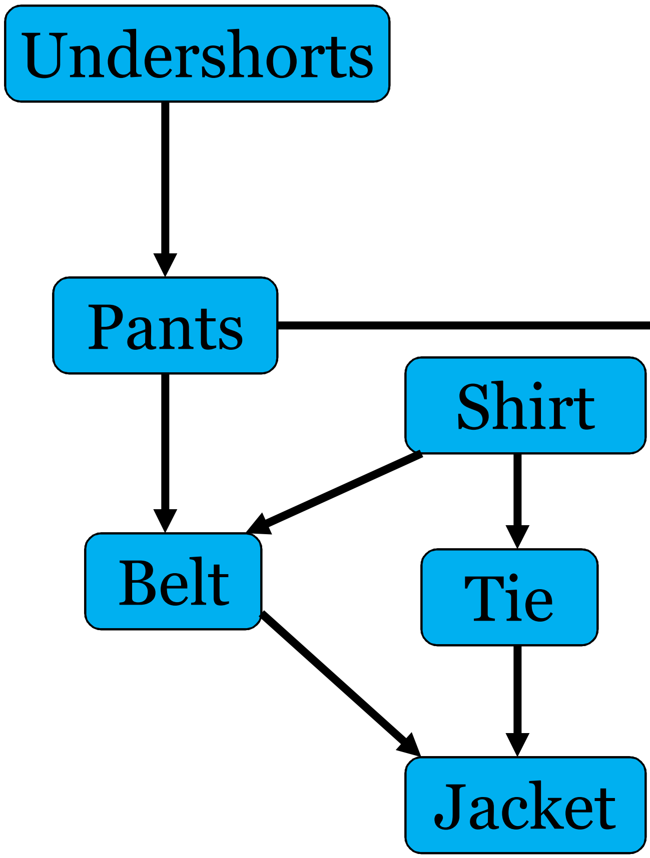
Manoj Kumar
DTU, Delhi

SAMSUNG

DTU
Delhi Technological
UNIVERSITY

Directed Acyclic Graph (DAG)

- A Directed Graph without a cycle.



a DAG implies an ordering on events

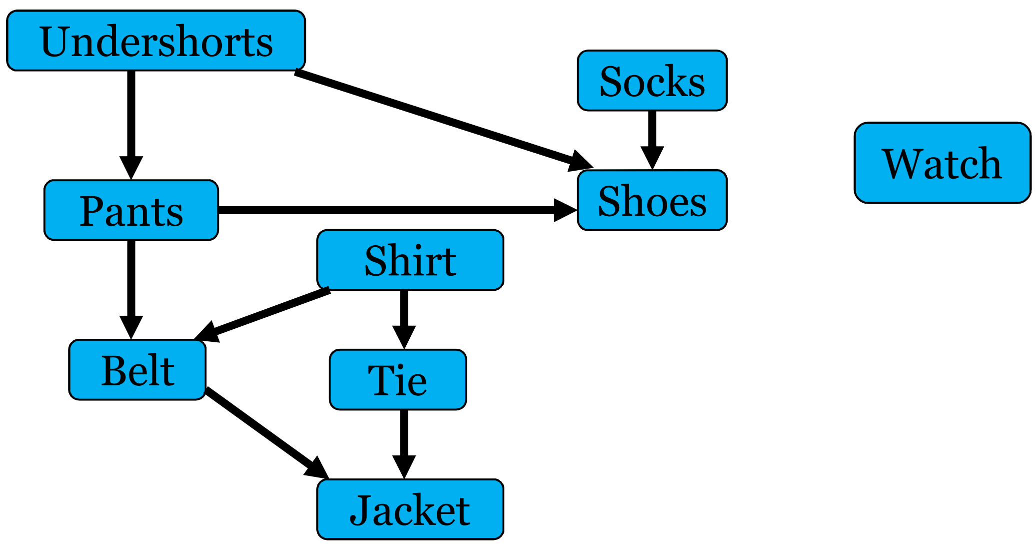
In a complex DAG, it can be hard to find a schedule that obeys all the constraints.

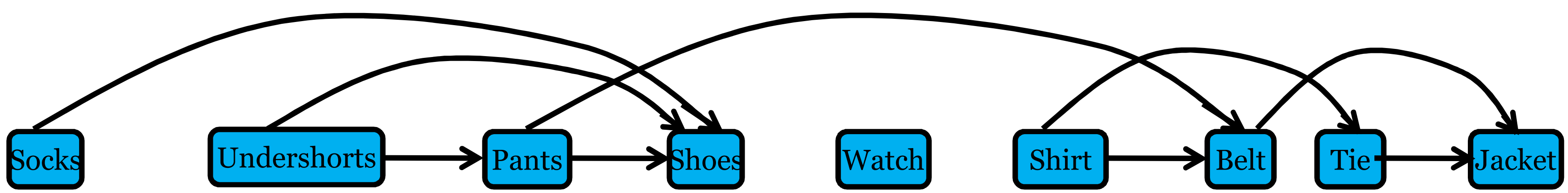
Topological Sort

- For a directed acyclic graph $G = (V,E)$, a topological sort is a linear ordering of all vertices of G such that if G contains an edge (u,v), then u appears before v in the ordering.

- A topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.

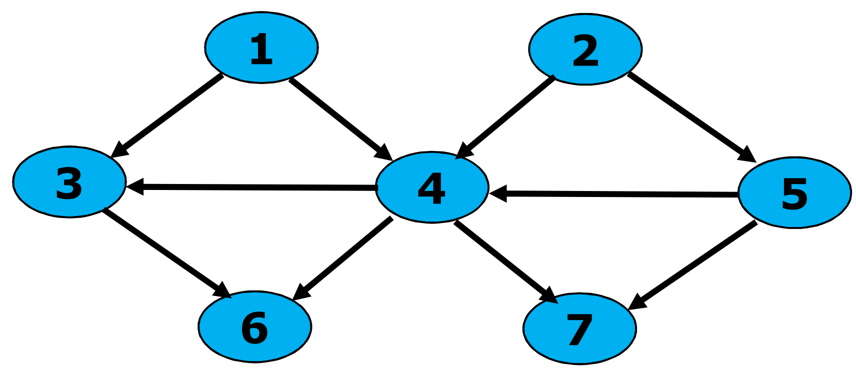
Topological Sort:Example





Topological sort

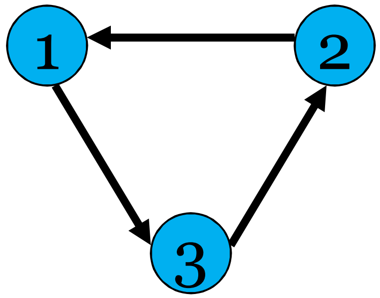
- There are often many possible topological sorts of a given DAG (*Directed Acyclic Graph*)
- Topological orders for this DAG :
  - 1,2,5,4,3,6,7
  - 2,1,5,4,7,3,6
  - 2,5,1,4,7,3,6
  - *Etc.*



- Each topological order is a *feasible schedule*.

Topological Sorts for Cyclic Graphs?

Impossible!



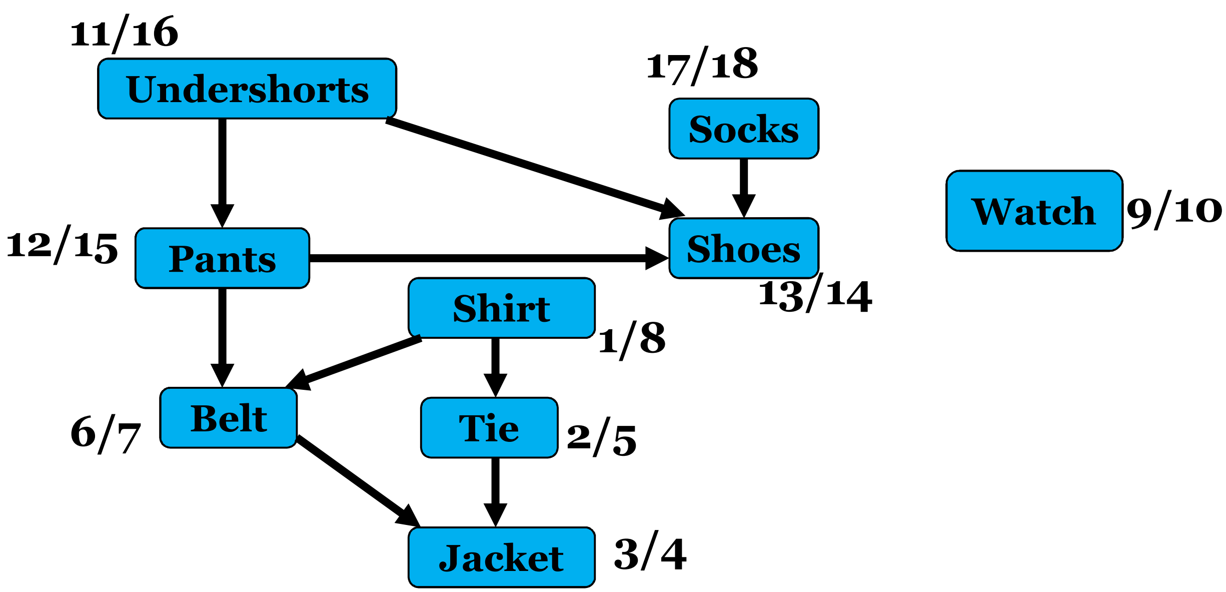
- ➢ If v and w are two vertices on a cycle, there exist paths from v to w *and* from w to v.
- ➢ Any ordering will contradict one of these paths

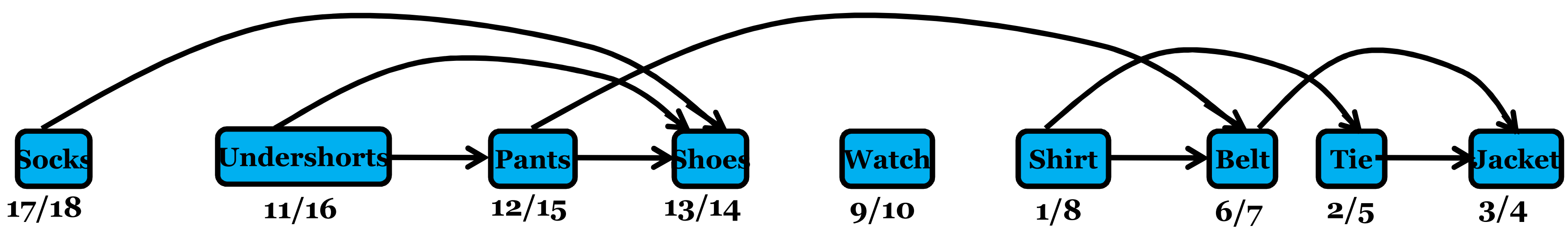
Topological Sort: Algorithm

TOPOLOGICAL-SORT(G)

1. Call DFS(G) to compute finishing time *f[v]* for each vertex *v*.
2. As each vertex is finished, insert it onto the front of a linked list.
3. Return the linked list of vertices.

Topological Sort





All edges of G are going from left to right only

Strongly Connected Components

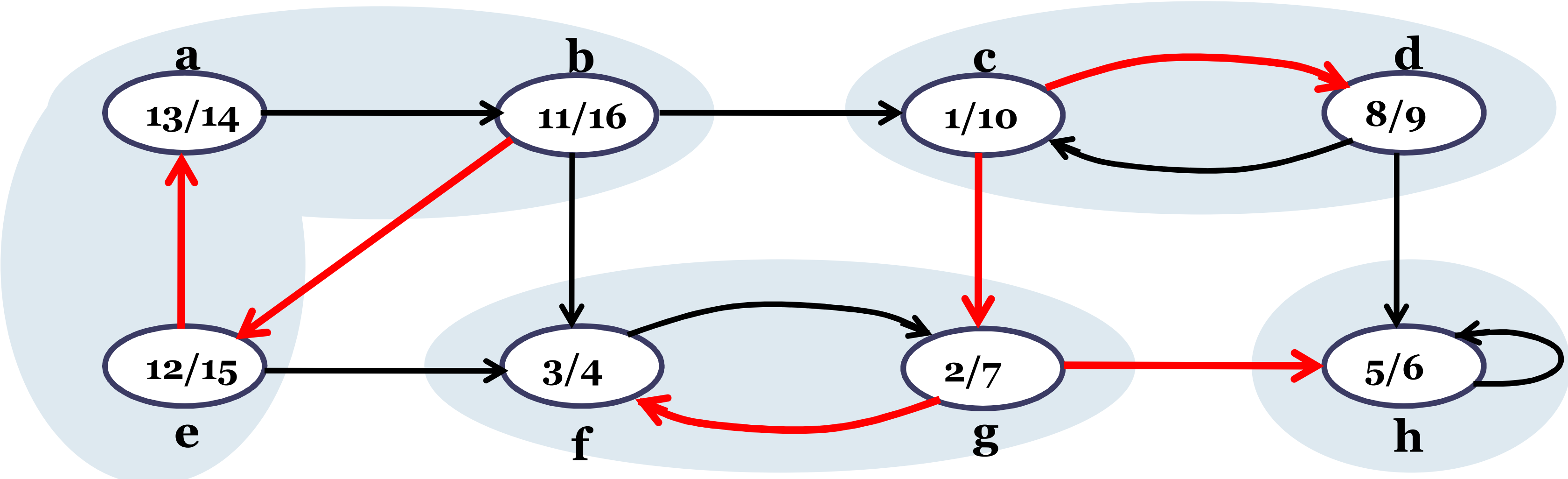
- Strongly connected component of a directed graph $G=(V,E)$ is a maximal set of vertices $U \subseteq V$ such that for every pair of vertices u and v in U, we have both $u \rightsquigarrow v$ and $v \rightsquigarrow u$, that is u and v are reachable from each other.

Strongly Connected Components: Algorithm

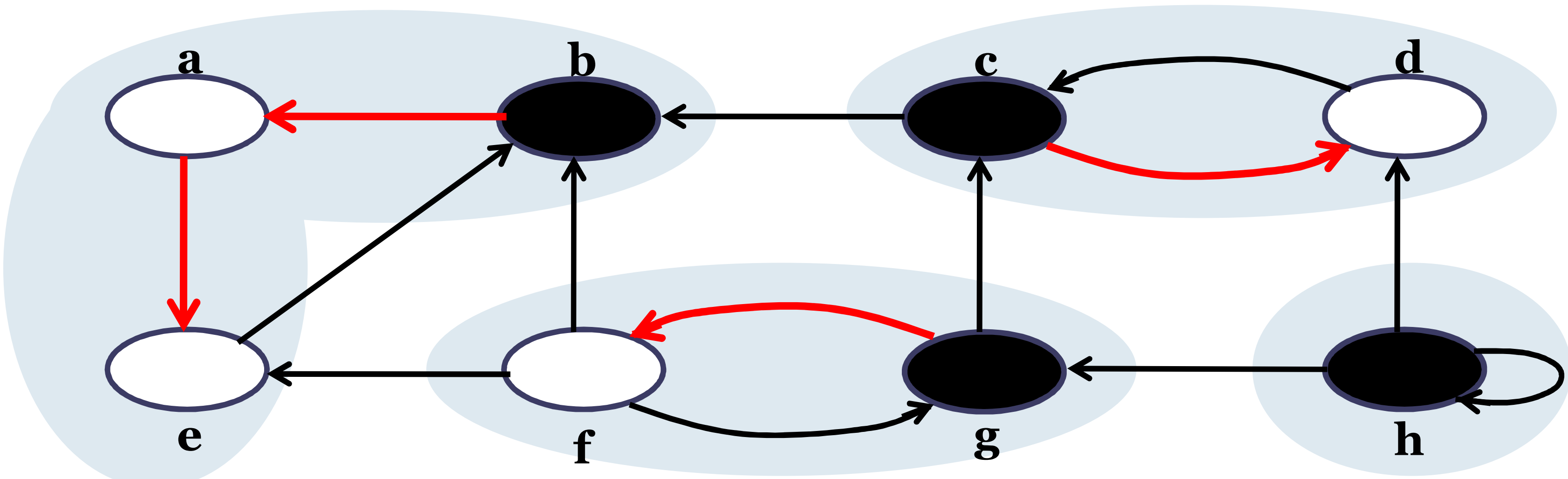
STRONGLY-CONNECTED-COMPONENTS(*G*)

1. Call DFS(*G*) to compute finishing time *f[u]* for each vertex *u*.
2. Compute G^T, transpose of *G* by reversing all edges.
3. Call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing *f[u]* as computed in line 1.
4. Output the vertices of each tree in the depth-first forest of step 3 as a separate strongly connected component.

Strongly Connected Components: Example

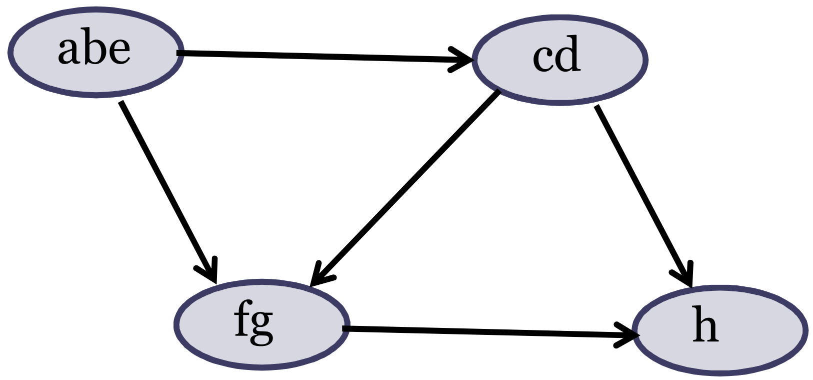


DFS on graph G with 4 SCCs, tree edges are in RED



DFS on Graph G^T (transpose of G)

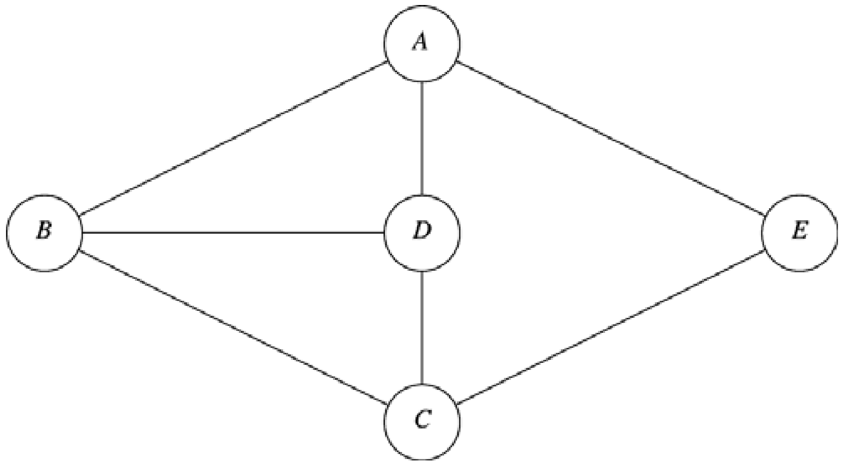
SCC tree



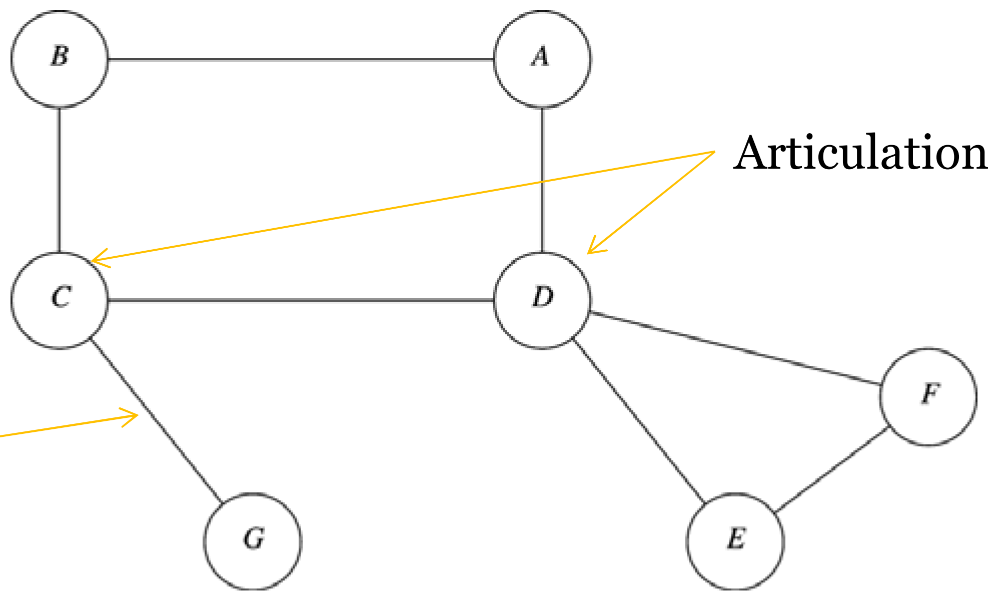
Articulation Points, bridges, and biconnected graph

- Let G be a connected, undirected graph.
- An articulation point of G is a vertex whose removal disconnects G.
- A bridge of G is an edge whose removal disconnects G.
- A graph is biconnected if it contains no articulation point.
- A biconnected component of G is a maximal biconnected subgraph.

Articulation Points, bridges, and biconnected graph



Biconnected Graph



Articulation points

bridge

Single Source Shortest Path

- Given a weighted directed graph $G=(V,E)$, with weight function $w:E \rightarrow R$ mapping edges to real valued weights.
- Weight of path $p=<v_0,v_1,v_2,\ldots v_k>$ is the sum of the weights of its constituent edges.
  $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$
- We define the shortest path weight from u to v by

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

Representing shortest path

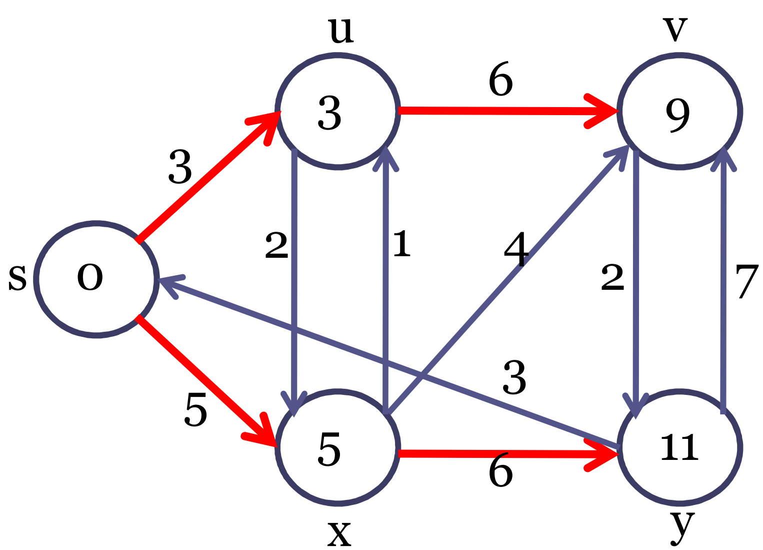
Shortest path tree

- Rooted at source vertex s,
- A directed graph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that

1. V' is the set of vertices reachable from s in G.
2. G' forms a rooted tree with root s, and
3. For all $v \in V'$, the unique simple path from s to v in G is a shortest path from s to v in G.

For each vertex v, we store $\Pi[v]$ pointing to its predecessor vertex. For source vertex s, $\Pi[s]$ =NIL

Representing shortest path



Π[s] = NIL
Π[u] = s
Π[v] = u
Π[x] = s
Π[y] = x

δ(s,s) = 0
δ(s,u) = 3
δ(s,v) = 9
δ(s,x) = 5
δ(s,y) = 11

Relaxation

- Algorithms keep track of d[v], π[v]. **Initialized** as follows:

```
INITIALIZE-SINGLE-SOURCE(G, s)
1.    for each v ∈ V[G] do
2.          d[v] ← ∞;            //distance from source
3.          π[v] ← NIL           //parent node
4.    d[s] ← 0                   // source to source distance =0
```

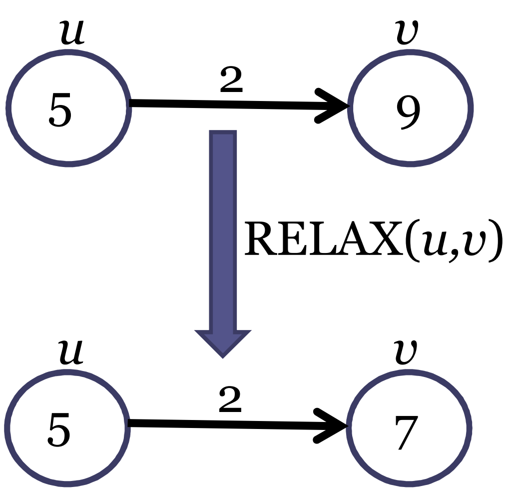
- These values are changed when an edge (u, v) is **relaxed**:

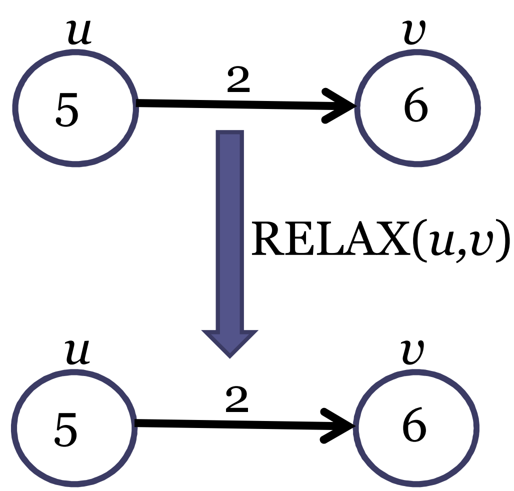
```
RELAX(u, v, w)
1.    if d[v] > d[u] + w(u, v) then     //new path from s to v through u
                                        is smaller
2.          d[v] ← d[u] + w(u, v);      //set new path
3.          π[v] ← u
```

SAMSUNG

DTU Delhi Technological University

Relaxation





d[v] > d[u]+w(u,v)

d[v] <= d[u]+w(u,v)

Dijkstra's Algorithm

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

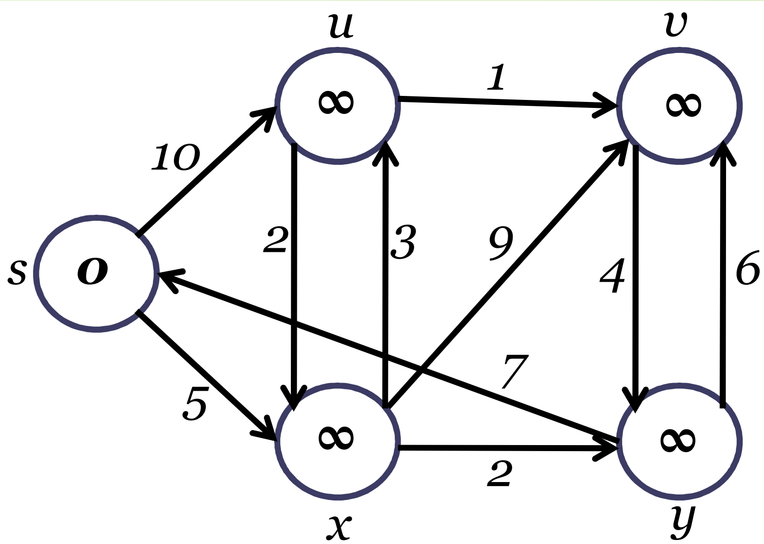
```
INITIALIZE-SINGLE-SOURCE(G, s)
    for each v ∈ V[G] do
        d[v] ← ∞;
        π[v] ← NIL
    d[s] ← 0
```

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

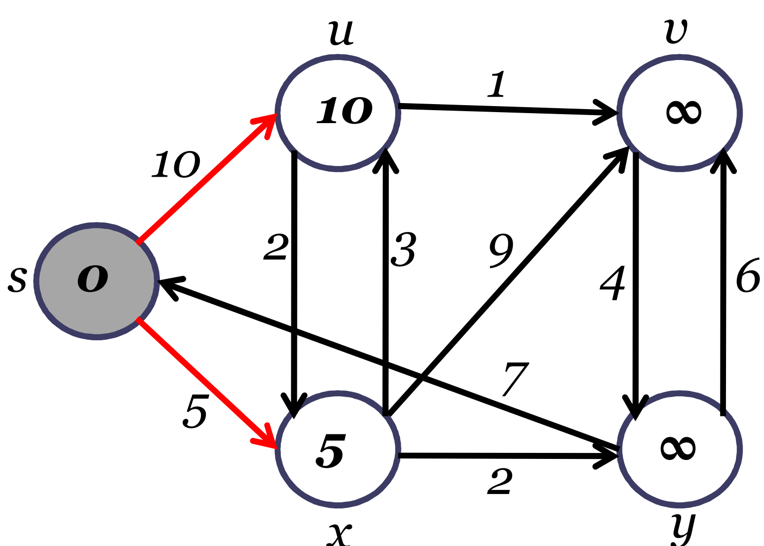
Dijkstra's Algorithm

```
DIJKSTRA(G, w, s)
    for each v ∈ V[G] do                          ⎫
        d[v] ← ∞;                                 ⎬ INITIALIZE-
        π[v] ← NIL                                ⎪ SINGLE-SOURCE
    d[s] ← 0                                      ⎭
    S←∅
    Q←V
    while Q ≠∅                  // Q is priority queue using minHeap
    do  u ← EXTRACT-MIN(Q)
          S←S ∪{u}
          for each vertex v ∈Adj[u]
        do  if d[v] > d[u] + w(u, v)                 ⎫
              then  d[v] ←d[u] + w(u, v);            ⎬      RELAX
                     π[v] ←u                         ⎭
```

Dijkstra: Example

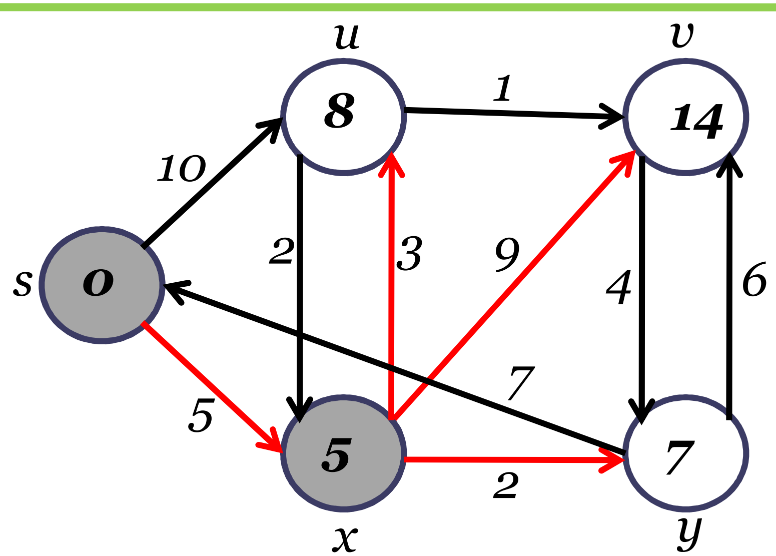


$\Pi[s]$ = NIL
$\Pi[u]$ = NIL
$\Pi[v]$ = NIL
$\Pi[x]$ = NIL
$\Pi[y]$ = NIL

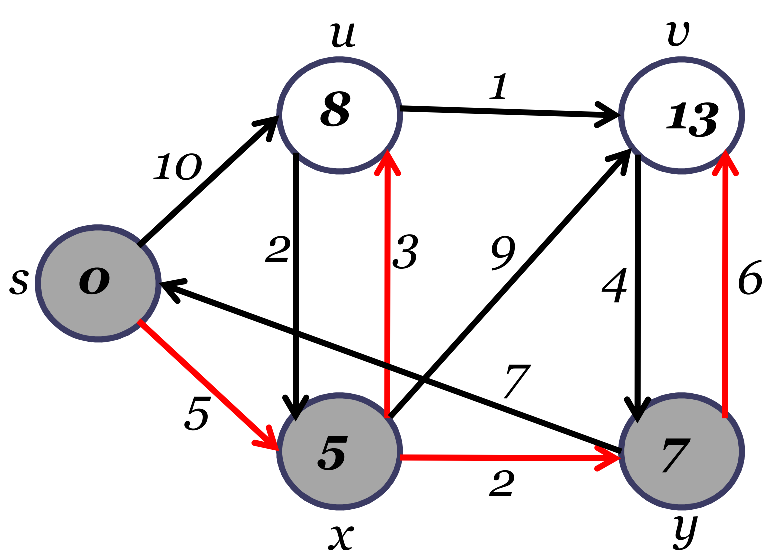


$\Pi[s]$ = NIL
$\Pi[u]$ = s
$\Pi[v]$ = NIL
$\Pi[x]$ = s
$\Pi[y]$ = NIL

Dijkstra: Example…

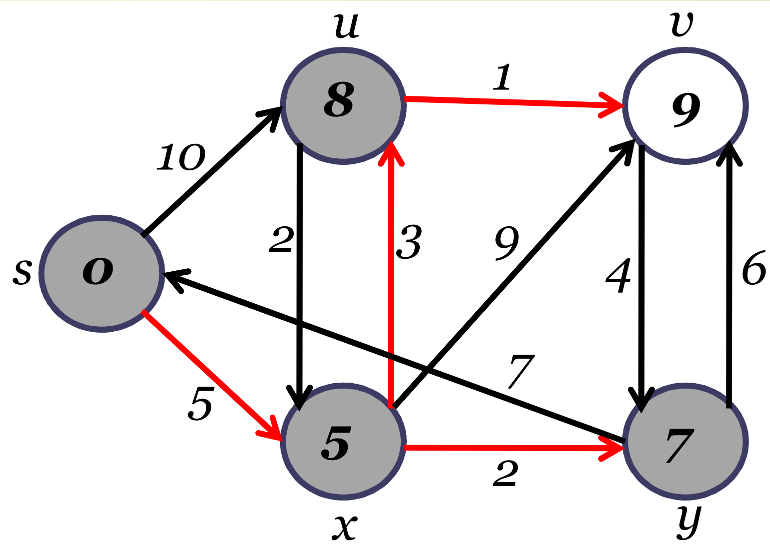


Π[s] = NIL
Π[u] = x
Π[v] = x
Π[x] = s
Π[y] = x

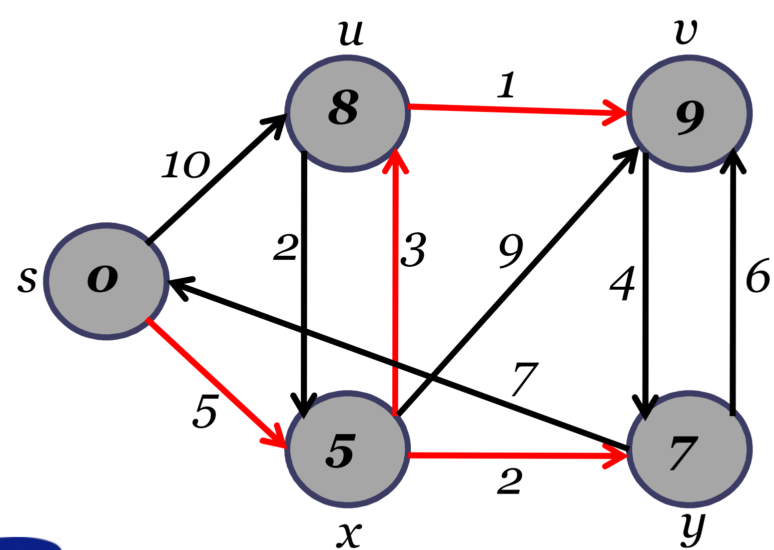


Π[s] = NIL
Π[u] = x
Π[v] = y
Π[x] = s
Π[y] = x

Dijkstra: Example…

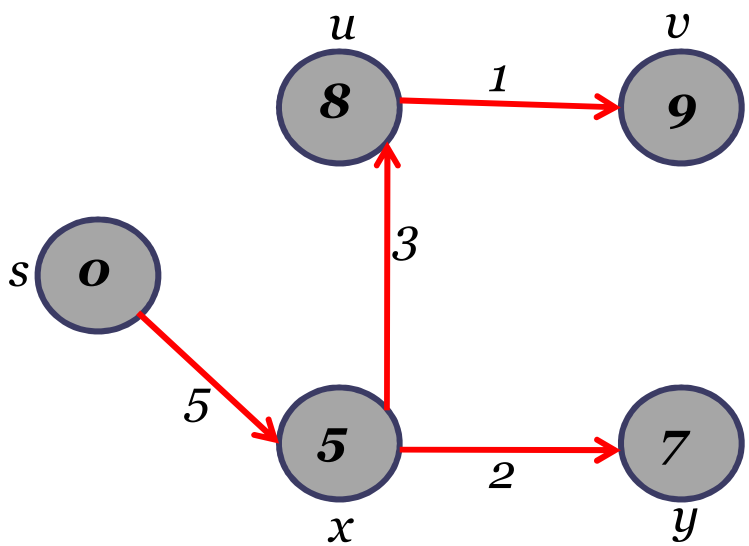


Π[s] = NIL
Π[u] = x
Π[v] = u
Π[x] = s
Π[y] = x



Π[s] = NIL	d(s,s) = 0
Π[u] = x	d(s,u) = 8
Π[v] = u	d(s,v) = 9
Π[x] = s	d(s,x) = 5
Π[y] = x	d(s,y) = 7

Shortest-path tree



$\Pi[s] = NIL$ $\quad$ $d(s,s) = 0$

$\Pi[u] = x$ $\quad$ $d(s,u) = 8$

$\Pi[v] = u$ $\quad$ $d(s,v) = 9$

$\Pi[x] = s$ $\quad$ $d(s,x) = 5$

$\Pi[y] = x$ $\quad$ $d(s,y) = 7$