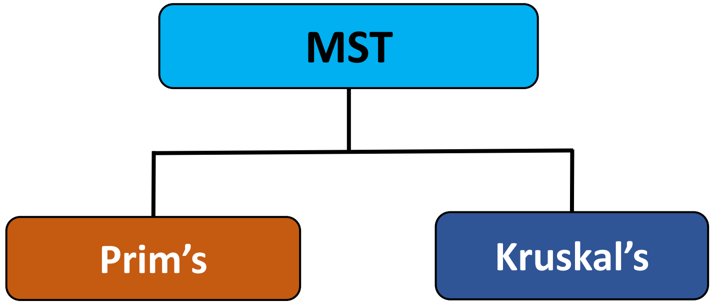
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Recap: Minimum Spanning Tree (MST)

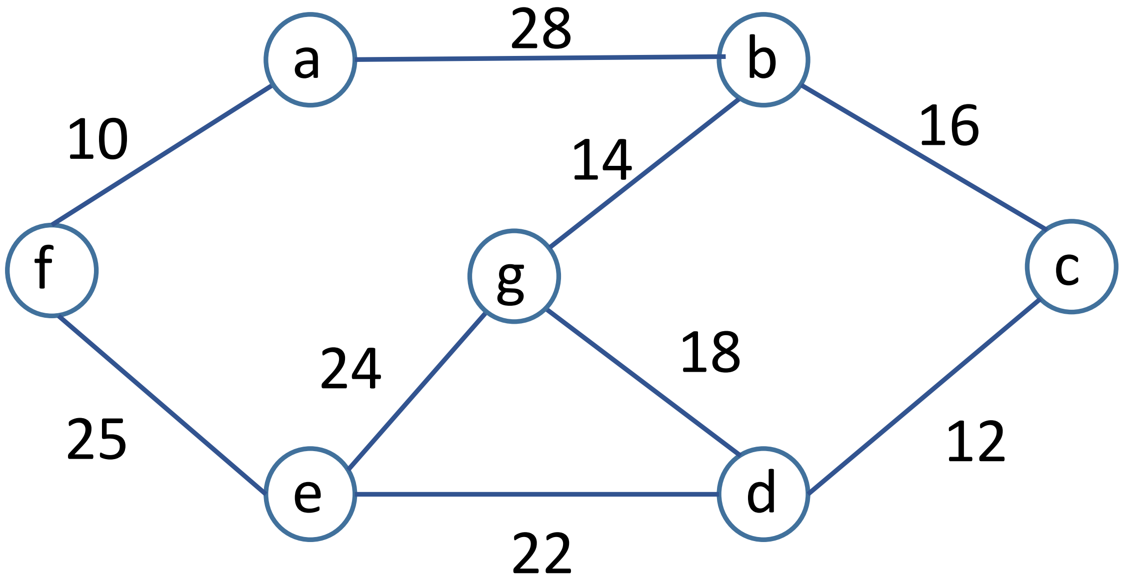
```
MST-PRIM (G, w, r) {
        for each u ϵ G.V
                u.key = ∞
                u. π = NIL
        r.key = 0
        Q = G.V
        while Q ≠ ∅ ;
            u = EXTRACT-MIN(Q)
            for each v ϵ G.Adj[u]
                if v ϵ Q and w(u,v) < v.key
                    v. π = u
                    v.key = w(u,v)
}
```



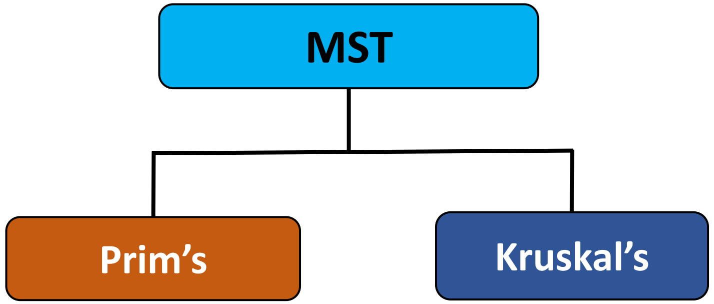
Exercise: MST Prims's algorithm

Construct the minimum spanning tree (MST) for the given graph using Prims's algorithm?

What is the minimum length of the edges?



Recap: Minimum Spanning Tree (MST)



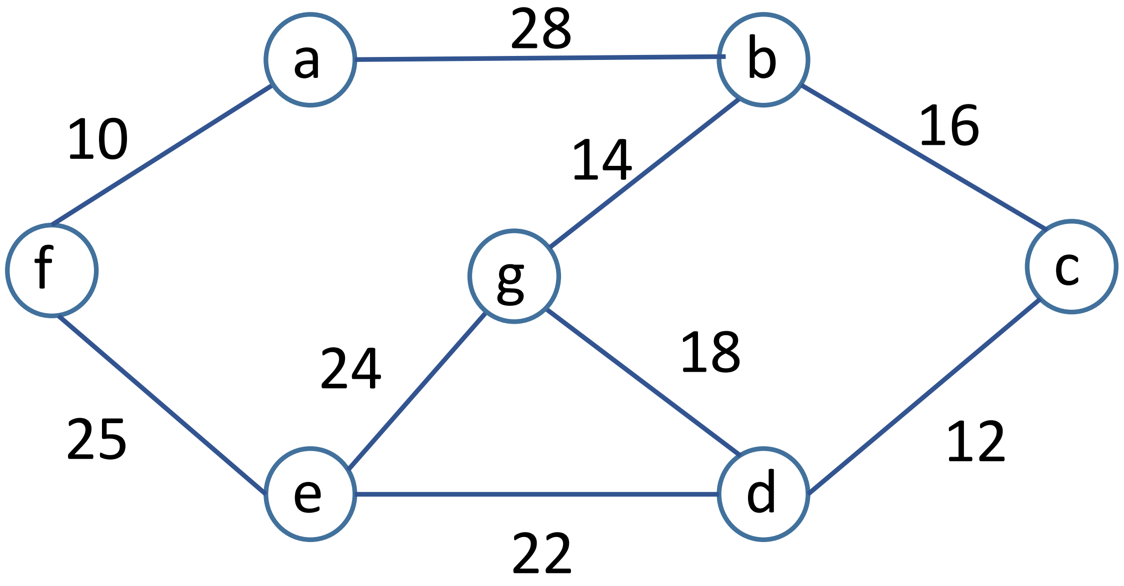

```
MST- KRUSKAL(G, w) {
  A = ∅
  for each vertex v ∈ G. V
        MAKE-SET (v)
  sort the edges of G. E into non-decreasing order by weight w
  for each edge (u, v) ∈ G. E, taken in non-decreasing order by weight
        if FIND-SET (u) ≠ FIND-SET (v)
                A = A ∪ {(u, v)}
                UNION (u, v)
  return A
}
```

Total time complexity: $O(n) + O(m \log m) + O(m) = O(m \log m)$
Worst case → $m = n^2$
Total time complexity = $O(m \log m) = O(m \log n^2) = O(2m \log n) = O(m \log n)$

Exercise: MST Kruskal's algorithm

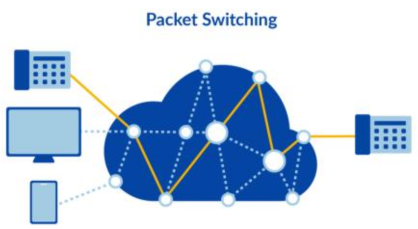
Construct the minimum spanning tree (MST) for the given graph using Kruskal's algorithm?

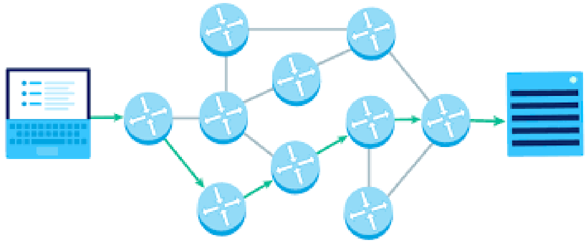
What is the minimum length of the edges?



Graphs: Shortest Path (SP)

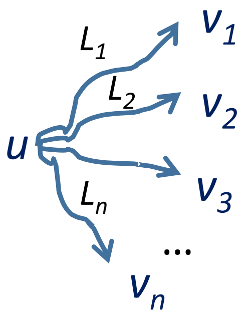
- Digital Mapping Services in Google Maps
- Social Networking Applications
- Telephone Network
- IP routing to find Open shortest Path First
- Designate file server
- Robotic Path





Shortest Path

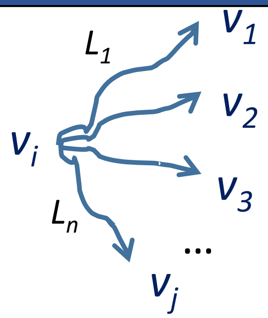
Single-pair shortest path



Shortest path

From a single source → to every other vertex

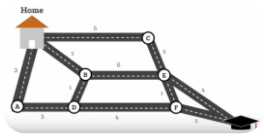
All-pair shortest path



Shortest path

From every vertex → to every other vertex

(i) Edge weights are *positive*



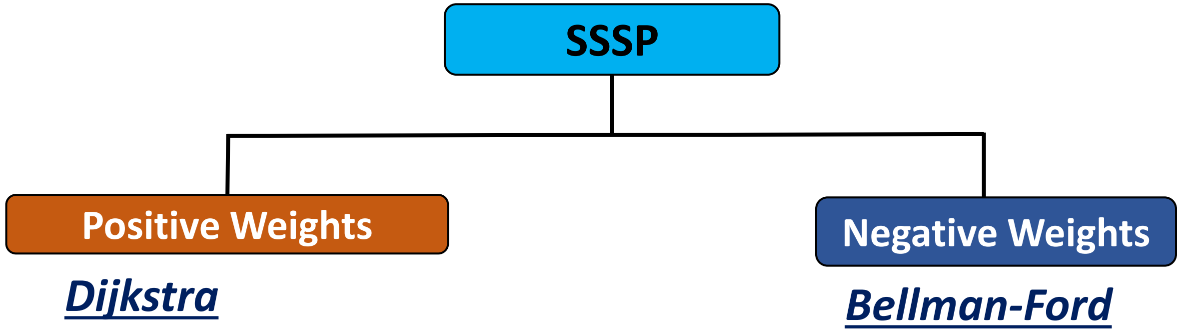
(ii) Edge weights are *negative*

TransferWise Splitwise

Single Source Shortest Path (SSSP)

Dijkstra algorithm:

- An Iterative algorithm that finds the *shortest path* from the *source vertex* to all other vertices in the graph
- It does not work on the graphs with *negative weights*
- It is a *greedy algorithm*



SSSP: Dijkstra (1)

```
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 ∊ Q. Adj[u]
     RELEAX (u, v, w)
}
```

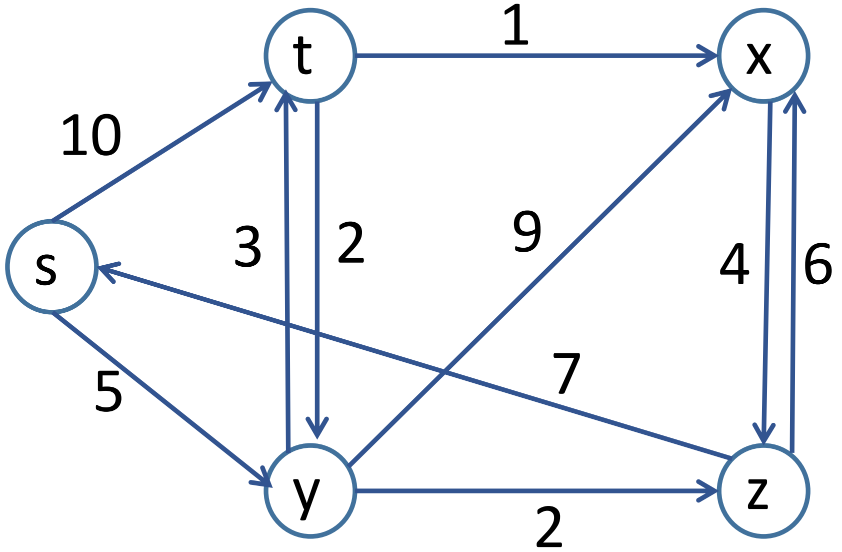
1

```
INITIALIZE-SINGLE-SOURCE (G, s) {
for each v ∊ G.V
   v. d = ∞
   v. π = NIL
s.d = 0
```

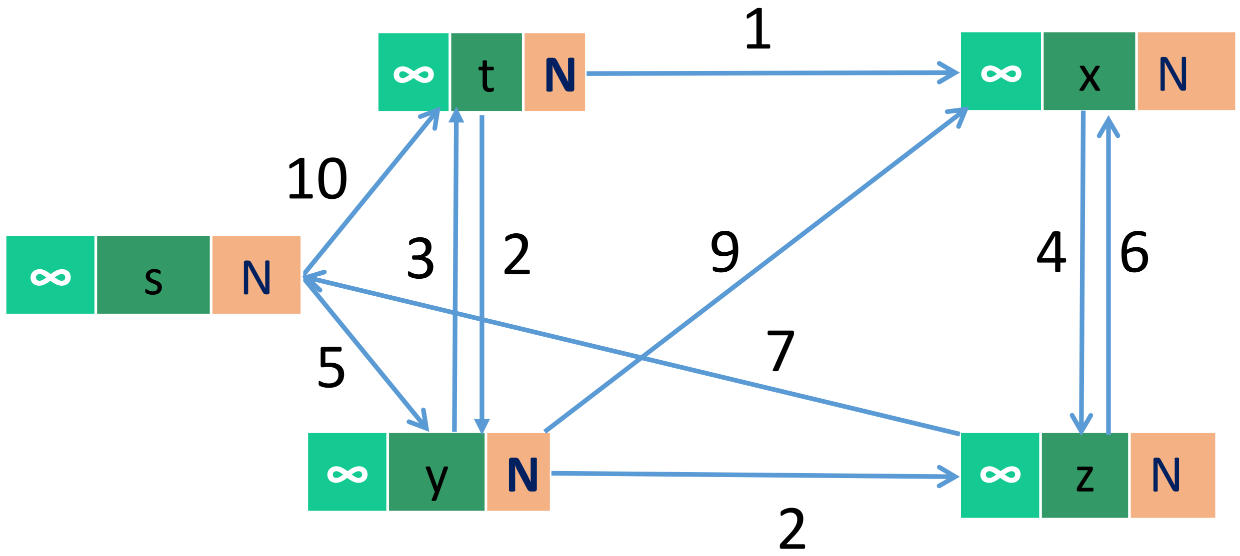
2

```
RELAX (u, v, w) {
if v.d > u.d + w(u, v)
    v. d =  u.d + w(u, v)
    v. π = u
```

3



Step 1: INITIALIZE-SINGLE-SOURCE
Step 2: Initialize Set S to empty set
Step 3: Q = {s, t, y, x, z}



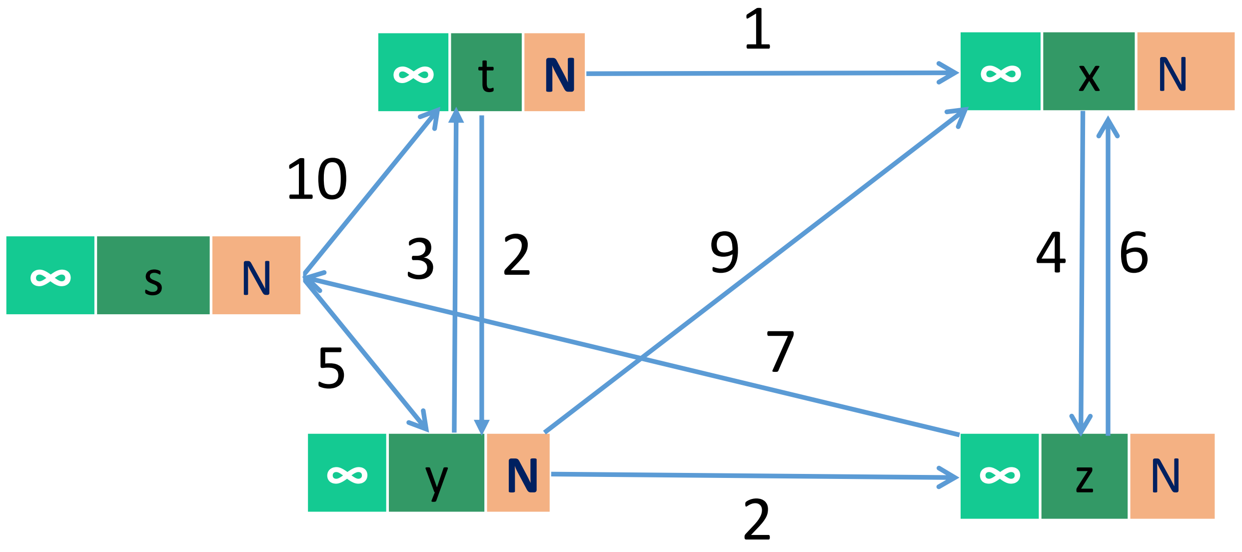
SSSP: Dijkstra (1)

```
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 ϵ Q. Adj[u]
    RELEAX (u, v, w)
}
```

```
INITIALIZE-SINGLE-SOURCE (G, s) {
for each v ϵ G.V
  v. d = ∞
  v. π = NIL
s.d = 0
```

```
RELAX (u, v, w) {
if v.d > u.d + w(u, v)
    v. d =  u.d + w(u, v)
    v. π = u
```

Step 1: INITIALIZE-SINGLE-SOURCE
Step 2: Initialize Set S to empty set
Step 3: Q = {s, t, y, x, z}



SSSP: Dijkstra (2)

```
DIJKSTRA (G, w, s) {
1  INITIALIZE-SINGLE-SOURCE (G, s)
2  S = ∅
3  Q = G. V
4  while Q ≠ ∅ ;
5     u = EXTRACT-MIN(Q)
6     S = S ∪ {u}
7     for each vertex v ϵ Q. Adj[u]
8       RELEAX (u, v, w)
}
```

```
INITIALIZE-SINGLE-SOURCE (G, s) {
for each v ϵ G.V
   v. d = ∞
   v. π = NIL
s.d = 0
```

```
RELAX (u, v, w) {
if v.d > u.d + w(u, v)
   v. d =  u.d + w(u, v)
   v. π = u
```

Step 4: *Q= {s, t, y, x, z}*

Step 5: *u = s*

Step 6: *S = ∅ ∪ {s} = {s}*

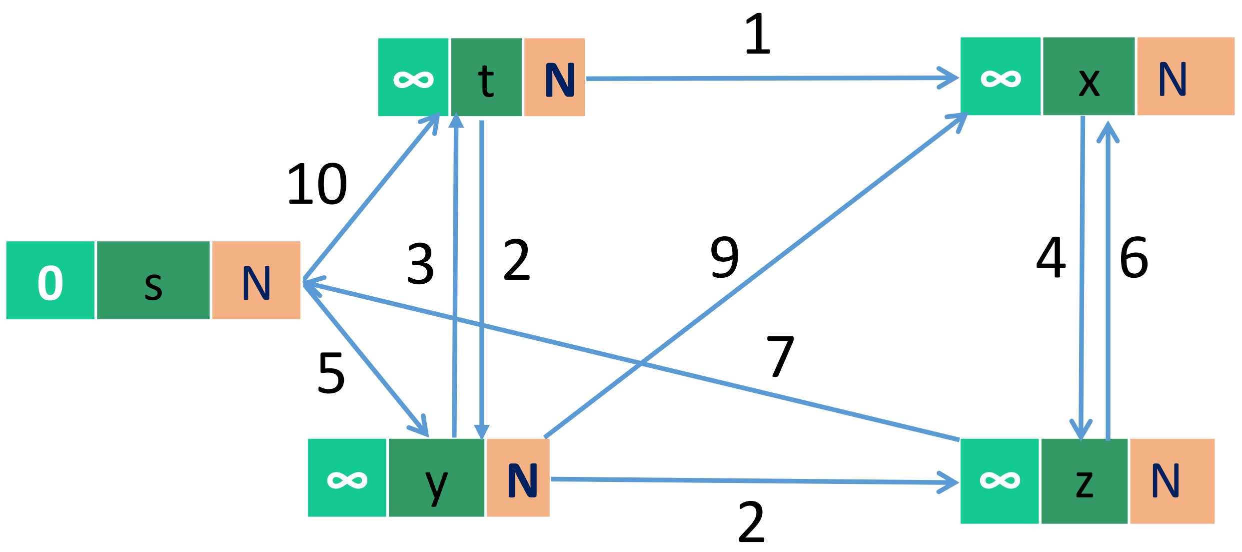
Step 7: *v = Q. Adj[s]*

= {t, y}

Step 8: *if t. d > s.d + w(s, t) = ∞ > 0 + 10*

t. d = 10

t. π = s



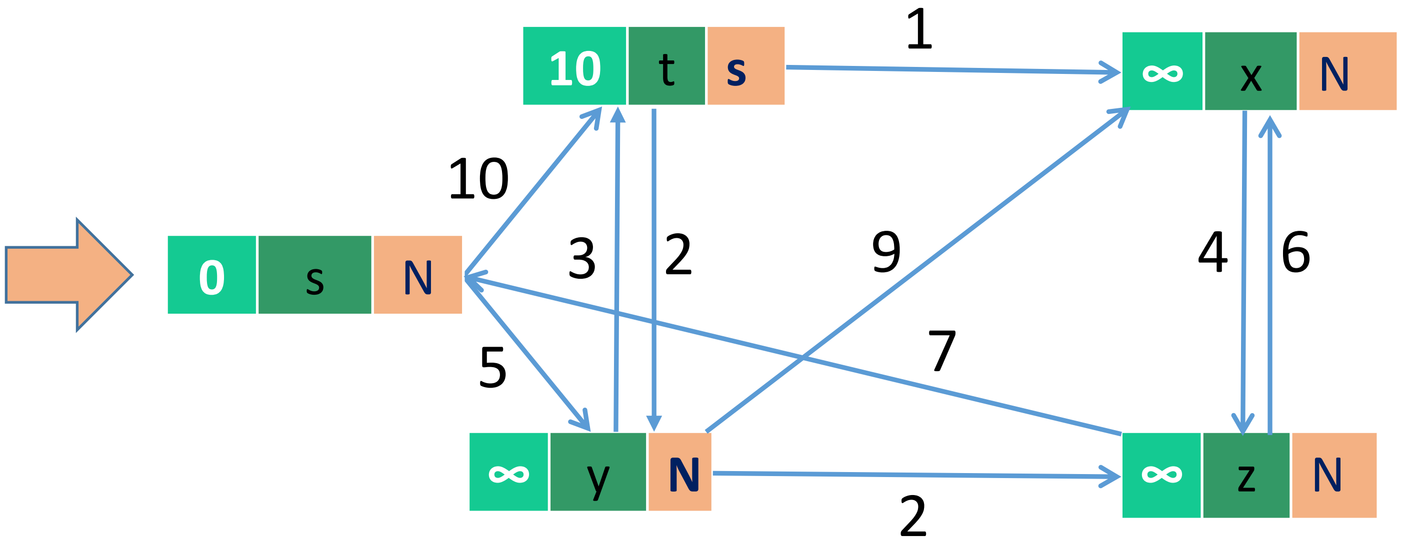
SSSP: Dijkstra (3)

DIJKSTRA (G, w, s) {
1 INITIALIZE-SINGLE-SOURCE (G, s)
2 S = ∅
3 Q = G. V
4 while Q ≠ ∅ ;
5 u = EXTRACT-MIN(Q)
6 S = S ∪ {u}
7 for each vertex v ϵ Q. Adj[u]
8 RELEAX (*u, v, w*)
}

INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each *v* ϵ G.*V*
2 *v. d* = ∞
3 *v. π* = NIL
4 *s.d* = 0

RELAX (*u, v, w*) {
1 if v.d > u.d + w(u, v)
2 v. *d* = u.d + w(u, v)
3 *v. π* = u

Step 4: *Q= {s, t, y, x, z}*
Step 5: *u = s*
Step 6: *S = ∅ ∪ {s} = {s}*
Step 7: *v = Q. Adj[s]*
= {t, y}
Step 8: *if y. d > s. d + w(s, y) = ∞ > 0 + 5*
y. d = 5
y. π = s



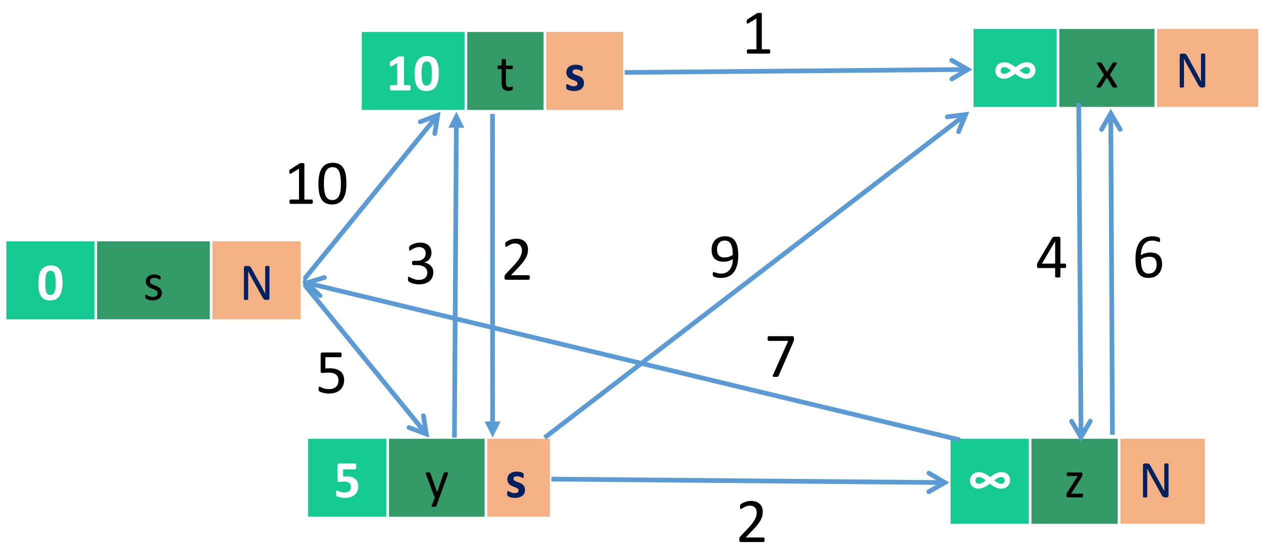
SSSP: Dijkstra (4)

DIJKSTRA (G, w, s) {
1 INITIALIZE-SINGLE-SOURCE (G, s)
2 S = ∅
3 Q = G. V
4 while Q ≠ ∅ ;
5 u = EXTRACT-MIN(Q)
6 S = S ∪ {u}
7 for each vertex v ϵ Q. Adj[u]
8 RELEAX (*u, v, w*)
}

INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each *v* ϵ G.*V*
2 *v. d* = ∞
3 *v. π* = NIL
4 *s. d* = 0

RELAX (*u, v, w*) {
1 if v.d > u.d + w(u, v)
2 v. *d* = u.d + w(u, v)
3 *v. π* = u

Step 4: *Q= {t, y, x, z}*
Step 5: *u = y*
Step 6: *S = {s} ∪ {y} = {s, y}*
Step 7: *v = Q. Adj[y]*
= {t, x, z}
Step 8: *if t. d > y. d + w (y, t) = 10 > 5+ 3*
t. d = 8
t. π = y



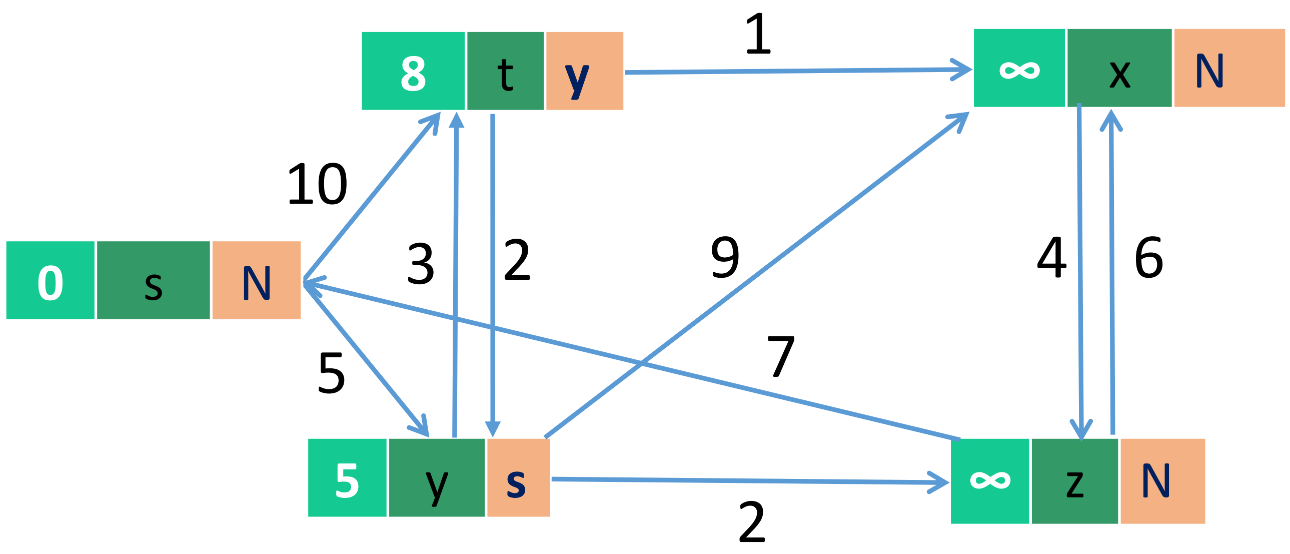
SSSP: Dijkstra (5)

```
DIJKSTRA (G, w, s) {
1  INITIALIZE-SINGLE-SOURCE (G, s)
2  S = ∅
3  Q = G. V
4  while Q ≠ ∅ ;
5    u = EXTRACT-MIN(Q)
6    S = S ∪ {u}
7    for each vertex v ϵ Q. Adj[u]
8      RELEAX (u, v, w)
}
```

```
INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each v ϵ G.V
2    v. d = ∞
3    v. π = NIL
4 s. d = 0
```

```
RELAX (u, v, w) {
1    if v.d > u.d + w(u, v)
2       v. d =  u.d + w(u, v)
3       v. π = u
```

Step 4: *Q= {t, x, z}*
Step 5: *u = y*
Step 6: *S = {s} ∪ {y} = {s, y}*
Step 7: *v = Q. Adj[y]*
= {t, x, z}
Step 8: *if x. d > y. d + w (y, x) = ∞ > 5+ 9*
x. d = 14
x. π = y



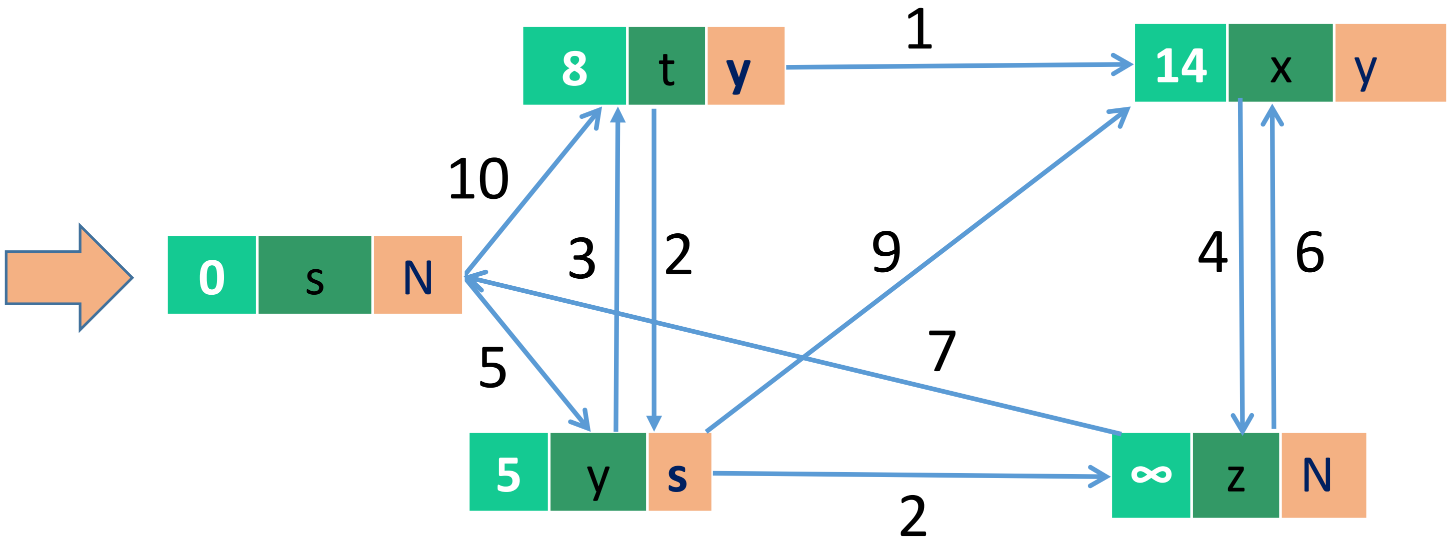
SSSP: Dijkstra (6)

DIJKSTRA (G, w, s) {
1 INITIALIZE-SINGLE-SOURCE (G, s)
2 S = ∅
3 Q = G. V
4 while Q ≠ ∅ ;
5 u = EXTRACT-MIN(Q)
6 S = S ∪ {u}
7 for each vertex v ∊ Q. Adj[u]
8 RELEAX (*u, v, w*)
}

INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each *v* ∊ G.*V*
2 *v. d* = ∞
3 *v. π* = NIL
4 *s. d* = 0

RELAX (*u, v, w*) {
1 if v.d > u.d + w(u, v)
2 v. *d* = u.d + w(u, v)
3 *v. π* = u

Step 4: *Q= {t, x, z}*
Step 5: *u = y*
Step 6: *S = {s} ∪ {y} = {s, y}*
Step 7: *v = Q. Adj[y]*
= {t, x, z}
Step 8: *if z. d > y. d + w (y, z) = ∞ > 5+ 2*
z. d = 7
z. π = y



SSSP: Dijkstra (7)

```
DIJKSTRA (G, w, s) {
1  INITIALIZE-SINGLE-SOURCE (G, s)
2  S = ∅
3  Q = G. V
4  while Q ≠ ∅ ;
5     u = EXTRACT-MIN(Q)
6     S = S ∪ {u}
7     for each vertex v ϵ Q. Adj[u]
8        RELEAX (u, v, w)
}
```

```
INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each v ϵ G.V
2     v. d = ∞
3     v. π = NIL
4 s. d = 0
```

```
RELAX (u, v, w) {
1    if v.d > u.d + w(u, v)
2        v. d =  u.d + w(u, v)
3        v. π = u
```

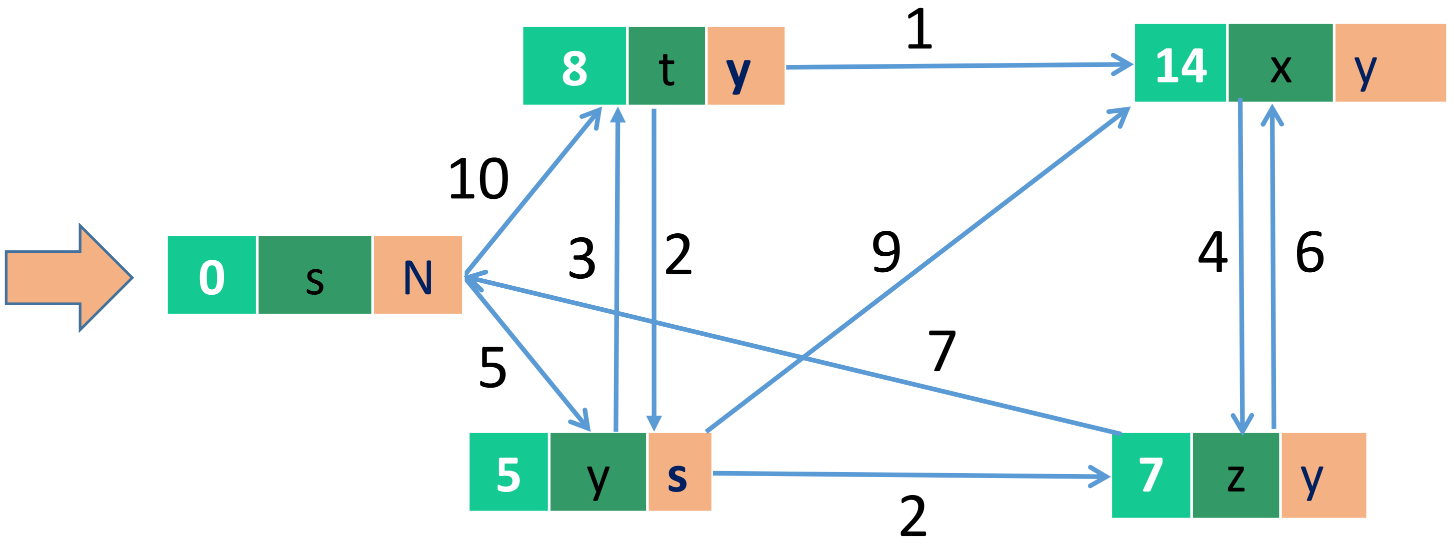
Step 4: *Q= {t, x, z}*

Step 5: *u = z*

Step 6: *S = {s,y} ∪ {z} = {s, y, z}*

Step 7: *v = Q. Adj[z]*
= {x, s}

Step 8: *if x. d > z. d + w (z, x) = 14 > 7+ 6*
x. d = 13
x. π = z



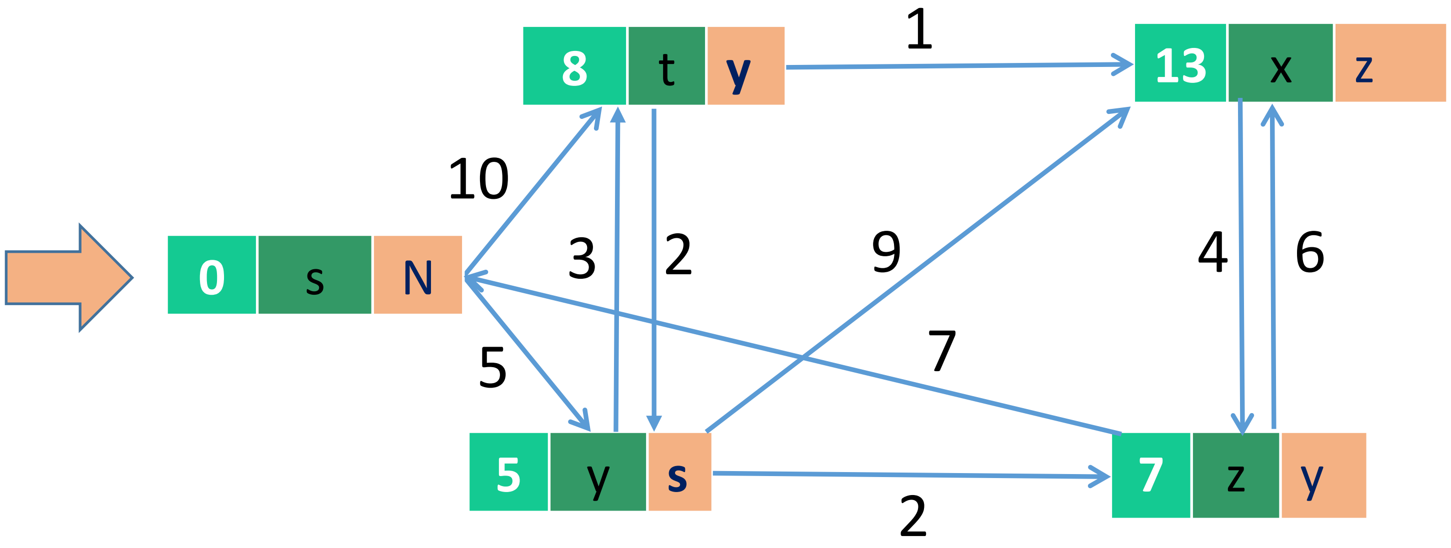
SSSP: Dijkstra (8)

```
DIJKSTRA (G, w, s) {
1  INITIALIZE-SINGLE-SOURCE (G, s)
2  S = ∅
3  Q = G. V
4  while Q ≠ ∅ ;
5     u = EXTRACT-MIN(Q)
6     S = S ∪ {u}
7     for each vertex v ϵ Q. Adj[u]
8       RELEAX (u, v, w)
}
```

```
INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each v ϵ G.V
2    v. d = ∞
3    v. π = NIL
4 s. d = 0
```

```
RELAX (u, v, w) {
1    if v.d > u.d + w(u, v)
2       v. d =  u.d + w(u, v)
3       v. π = u
```

Step 4: *Q= {t, x}*
Step 5: *u = z*
Step 6: *S = {s,y} ∪ {z} = {s, y, z}*
Step 7: *v = Q. Adj[z]*
= {x, s}
Step 8: *if s. d > z. d + w (z, s) = 0 > 7+ 7*



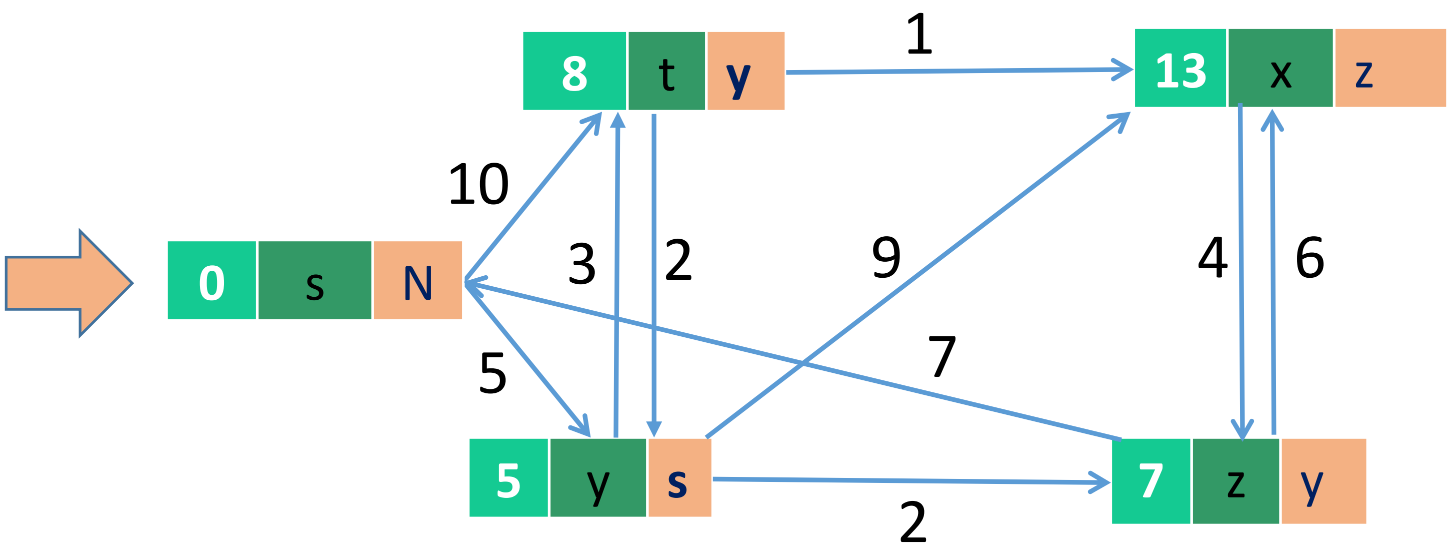
SSSP: Dijkstra (9)

```
DIJKSTRA (G, w, s) {
1  INITIALIZE-SINGLE-SOURCE (G, s)
2  S = ∅
3  Q = G. V
4  while Q ≠ ∅ ;
5     u = EXTRACT-MIN(Q)
6     S = S ∪ {u}
7     for each vertex v ϵ Q. Adj[u]
8        RELEAX (u, v, w)
}
```

```
INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each v ϵ G.V
2    v. d = ∞
3    v. π = NIL
4 s. d = 0
```

```
RELAX (u, v, w) {
1    if v.d > u.d + w(u, v)
2       v. d =  u.d + w(u, v)
3       v. π = u
```

Step 4: *Q= {t, x}*
Step 5: *u = t*
Step 6: *S = {s,y, z} ∪ {t} = {s, y, z, t}*
Step 7: *v = Q. Adj[t]*
= {y, x}
Step 8: *if y. d > t. d + w (t, y) = 5 > 8+ 2*



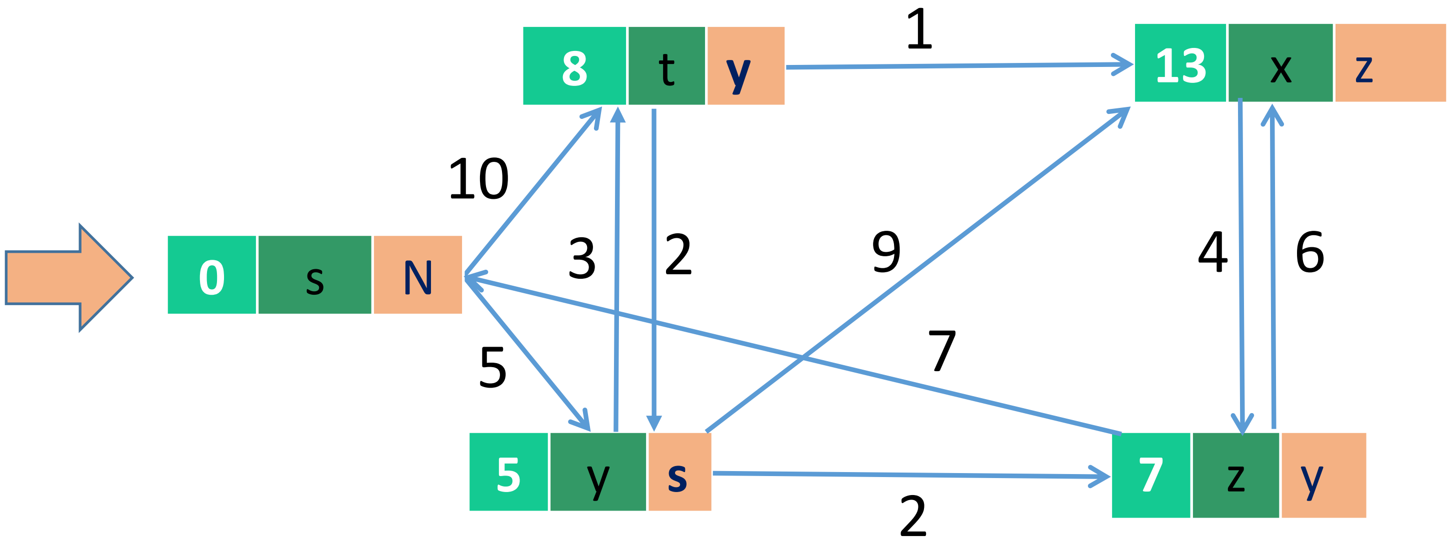
SSSP: Dijkstra (10)

DIJKSTRA (G, w, s) {
1 INITIALIZE-SINGLE-SOURCE (G, s)
2 S = ∅
3 Q = G. V
4 while Q ≠ ∅ ;
5 u = EXTRACT-MIN(Q)
6 S = S ∪ {u}
7 for each vertex v ϵ Q. Adj[u]
8 RELEAX (*u, v, w*)
}

INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each *v* ϵ G.*V*
2 *v. d* = ∞
3 *v. π* = NIL
4 *s. d* = 0

RELAX (*u, v, w*) {
1 if v.d > u.d + w(u, v)
2 v. *d* = u.d + w(u, v)
3 *v. π* = u

Step 4: *Q= {x}*
Step 5: *u = t*
Step 6: *S = {s,y, z} ∪ {t} = {s, y, z, t}*
Step 7: *v = Q. Adj[t]*
= {y, x}
Step 8: *if x. d > t. d + w (t, x) = 13 > 8+ 1*
x. d = 9
x. π = t



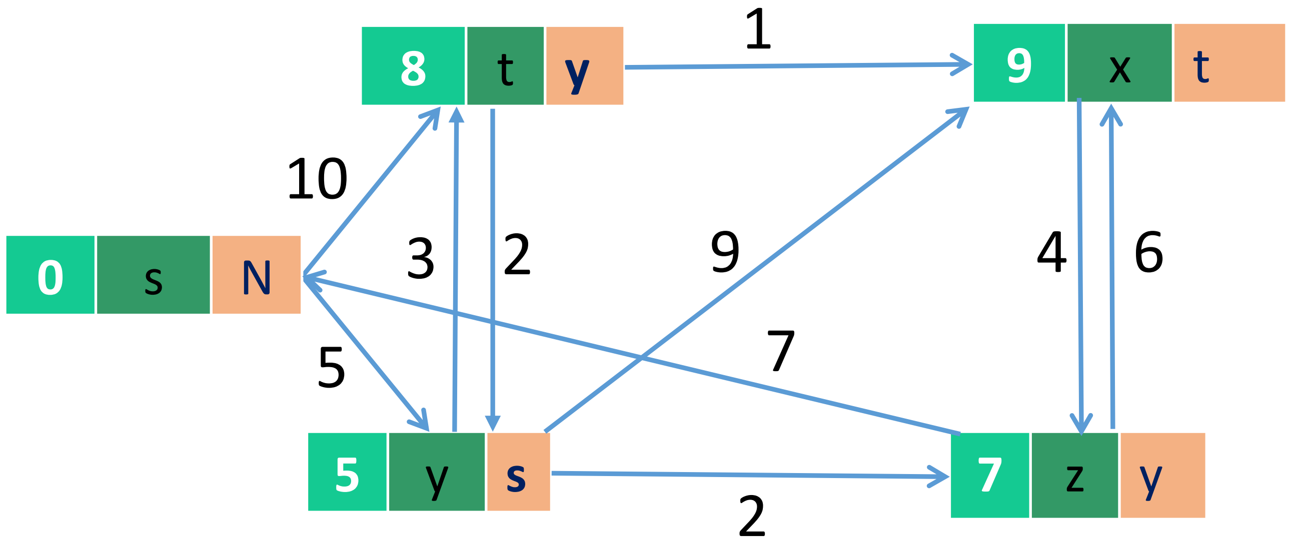
SSSP: Dijkstra (11)

DIJKSTRA (G, w, s) {
1 INITIALIZE-SINGLE-SOURCE (G, s)
2 S = ∅
3 Q = G. V
4 while Q ≠ ∅ ;
5 u = EXTRACT-MIN(Q)
6 S = S ∪ {u}
7 for each vertex v ϵ Q. Adj[u]
8 RELEAX (*u, v, w*)
}

INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each *v* ϵ G.*V*
2 *v. d* = ∞
3 *v. π* = NIL
4 *s. d* = 0

RELAX (*u, v, w*) {
1 if v.d > u.d + w(u, v)
2 v. *d* = u.d + w(u, v)
3 *v. π* = u

Step 4: *Q= {x}*
Step 5: *u = x*
Step 6: *S = {s,y, z, t} ∪ {x} = {s, y, z, t, x}*
Step 7: *v = Q. Adj[x]*
= {z}
Step 8: *if z. d > x. d + w (x, z) = 7 > 9+ 4*



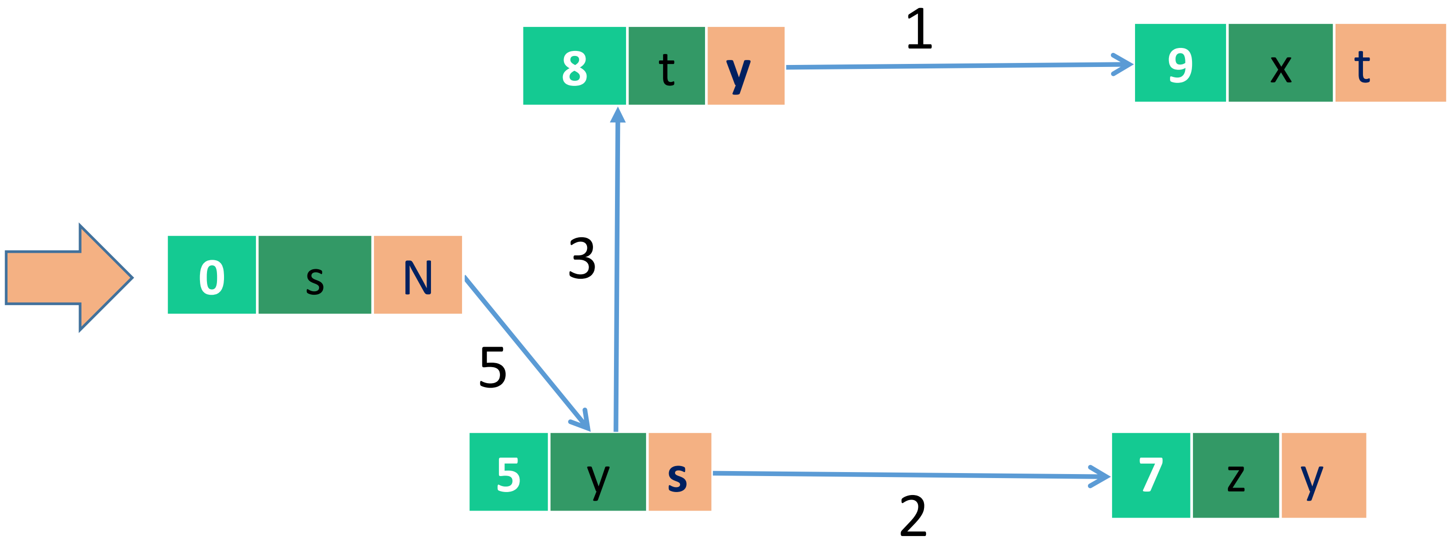
SSSP: Dijkstra (12)

DIJKSTRA (G, w, s) {
1 INITIALIZE-SINGLE-SOURCE (G, s)
2 S = ∅
3 Q = G. V
4 while Q ≠ ∅ ;
5 u = EXTRACT-MIN(Q)
6 S = S ∪ {u}
7 for each vertex v ϵ Q. Adj[u]
8 RELEAX (*u, v, w*)
}

INITIALIZE-SINGLE-SOURCE (G, s) {
1 for each *v* ϵ G.*V*
2 *v. d* = ∞
3 *v. π* = NIL
4 *s. d* = 0

RELAX (*u, v, w*) {
1 if v.d > u.d + w(u, v)
2 v. *d* = u.d + w(u, v)
3 *v. π* = u

Step 4: *Q= { }*
Step 6: *S = {s, y, z, t, x}*



SSSP: Dijkstra → time complexity analysis

DIJKSTRA (G, w, s) {
INITIALIZE-SINGLE-SOURCE (G, s) ← **O(n) → n = # of vertices → O (V)**
S = ∅ ← **O(1)**
Q = G. V ← **O (n) → O (V)** Heap construction
while Q ≠ ∅ ;
  u = EXTRACT-MIN(Q) } ***O(V logV)***
  S = S ∪ {u} ← **O(1) → O (V)**

  for each vertex v ϵ Q. Adj[u] ← **|E| times**
    RELEAX (*u, v, w*) ← ***O(log V)***
}

|E| times and ***O(log V)*** } ***O(E logV)***

INITIALIZE-SINGLE-SOURCE (G, s) {
for each *v* ϵ G.*V*
  v. d = ∞
  v. π = NIL
s. d = 0

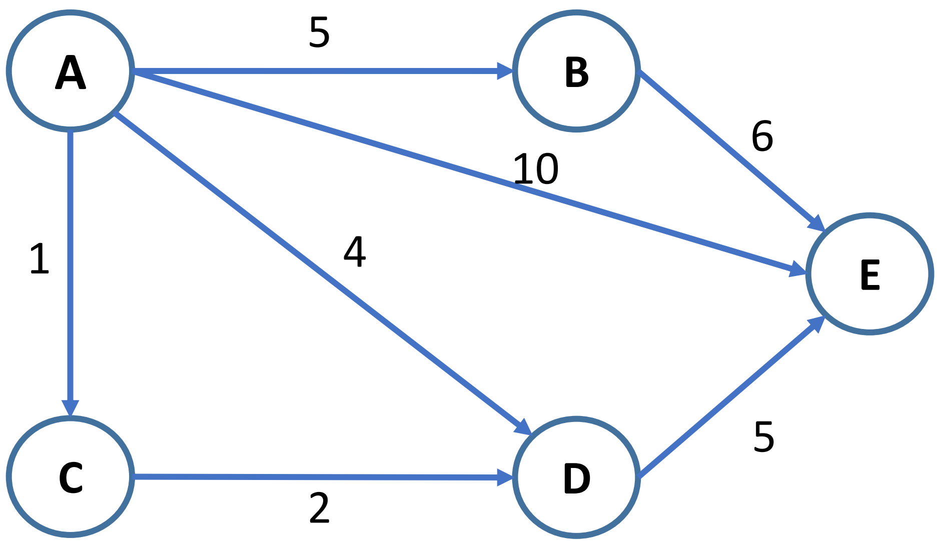
RELAX (*u, v, w*) {
if v.d > u.d + w(u, v)
  v. *d* = u.d + w(u, v)
  v. π = u

Total time complexity: <u>O(V) + O (V log V) + O(V) + O(E logV)</u> → O(ElogV)

Exercise: Dijkstra

Consider the following graph

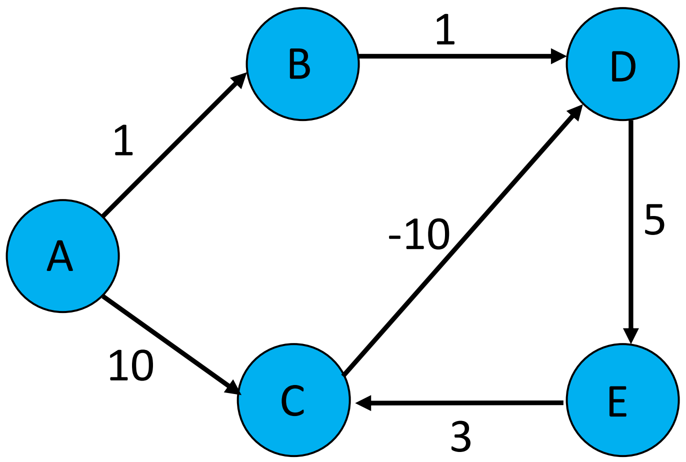
1. Draw the final possible shortest path reported by Dijkstra's algorithm?



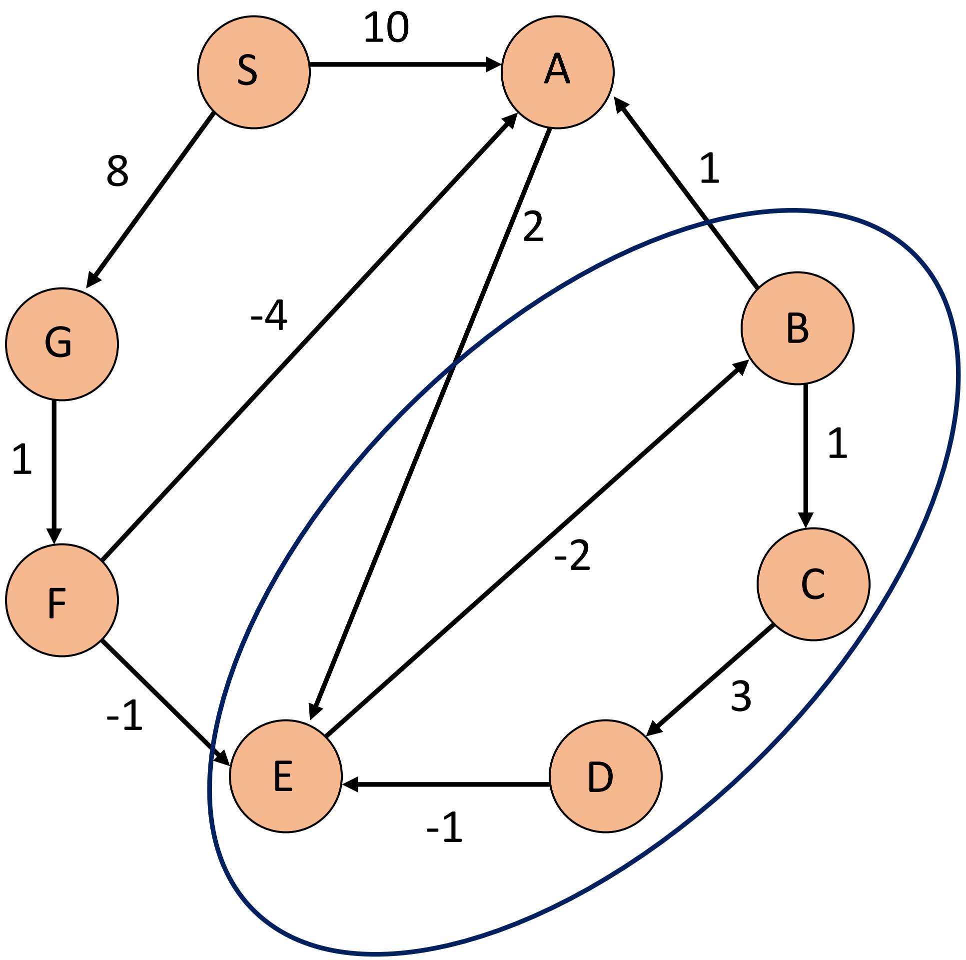
Exercise: Dijkstra (1)

Negative cycles

What is the shortest path from A to E?

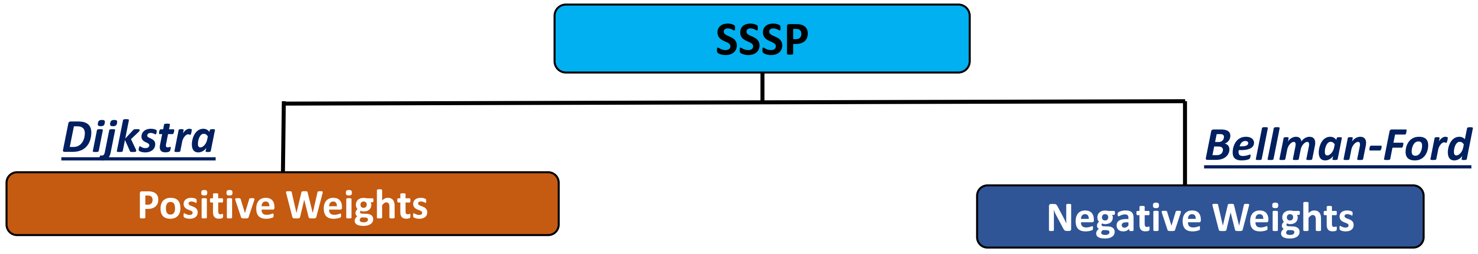


What is the shortest path from S to D?



- ❖ Dijkstra doesn’t work for **<u>Graphs with negative weight cycle</u>**, Bellman-Ford works for such graphs.
- ❖ Bellman-Ford is also simpler than Dijkstra and suites well for distributed systems.
- ❖ It does not use Priority Queue

SSSP → Dijkstra

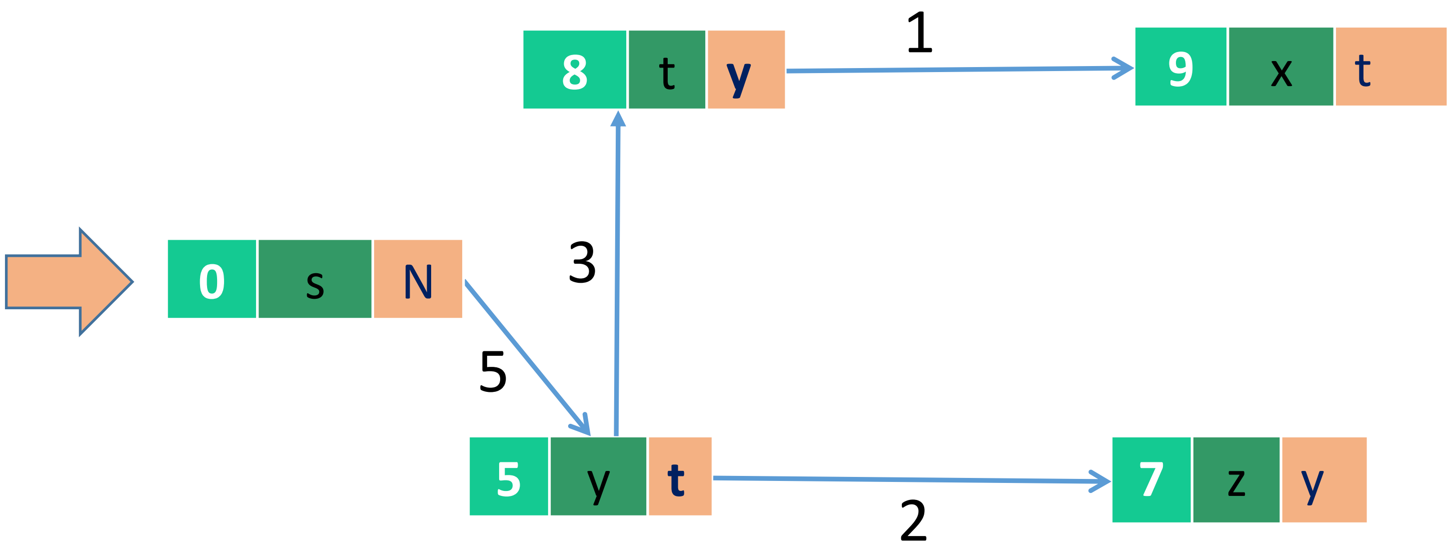


DIJKSTRA (G, w, s) {
INITIALIZE-SINGLE-SOURCE (G, s)
S = ∅
Q = G. V
while Q ≠ ∅ ;
  u = EXTRACT-MIN(Q)
  S = S ∪ {u}
  for each vertex v ϵ Q. Adj[u]
    RELEAX (*u, v, w*)
}

INITIALIZE-SINGLE-SOURCE (G, s) {
for each *v* ϵ G.*V*
  v. d = ∞
  v. π = NIL
s. d = 0

RELAX (*u, v, w*) {
if v.d > u.d + w(u, v)
  v. *d* = u.d + w(u, v)
  v. π = u

Step 6: *S = {s, y, z, t, x}*



Total time complexity: **<u>O(V) + O (V log V) + O(V) + O(E logV)</u> → O(ElogV)**

thank you!

email:
k.kondepu@iitdh.ac.in