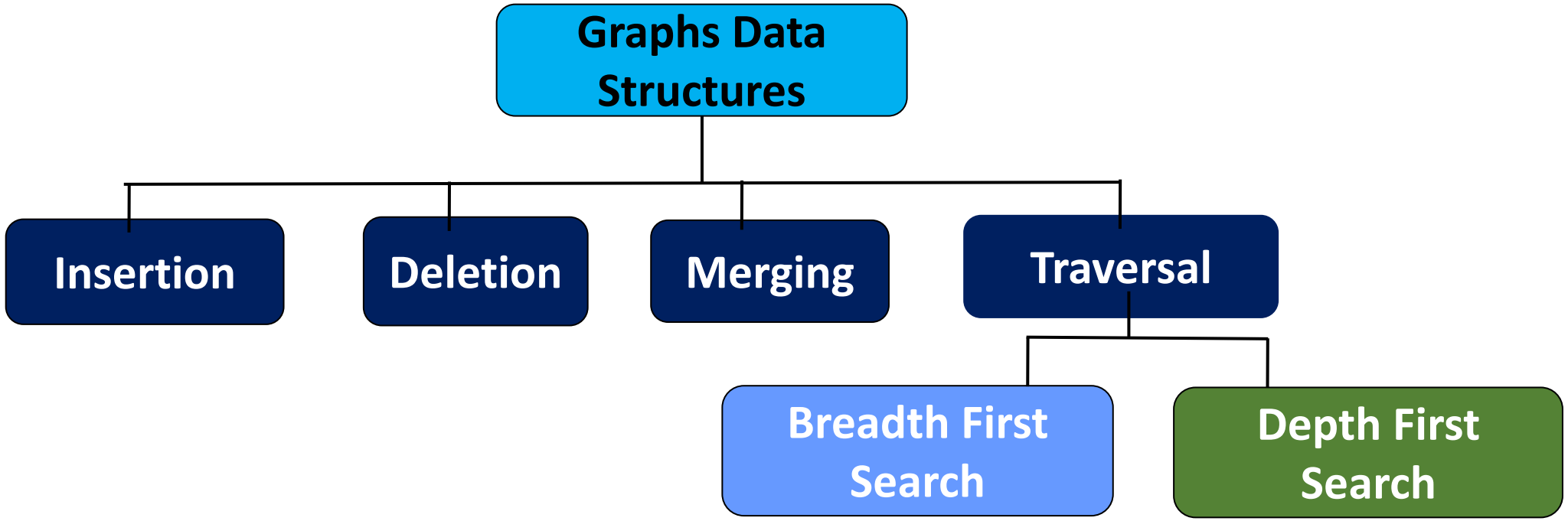
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

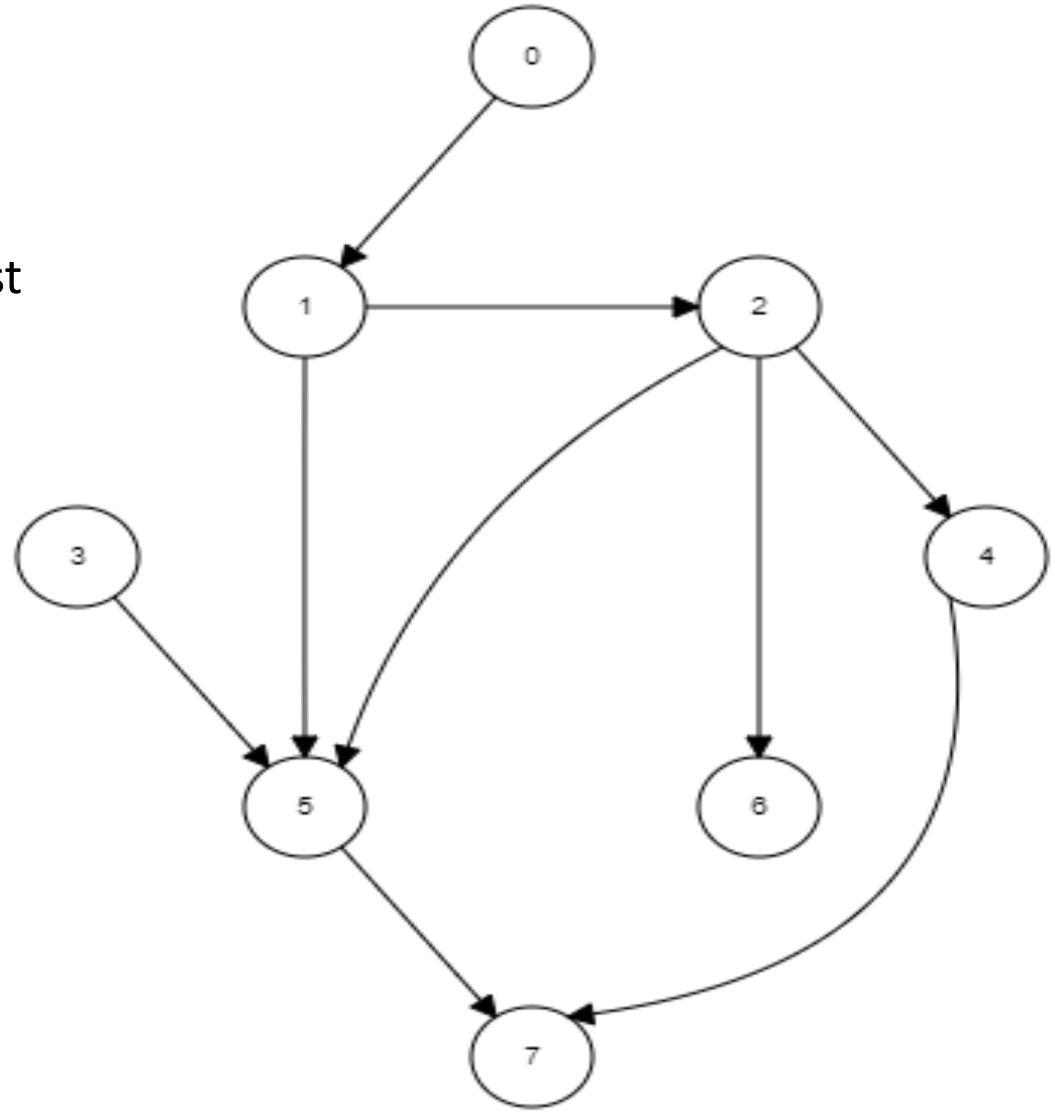
Recap: Graph Traversal



Recap: Graph Traversal Applications → DFS

TOPOLOGICAL – SORT (G)

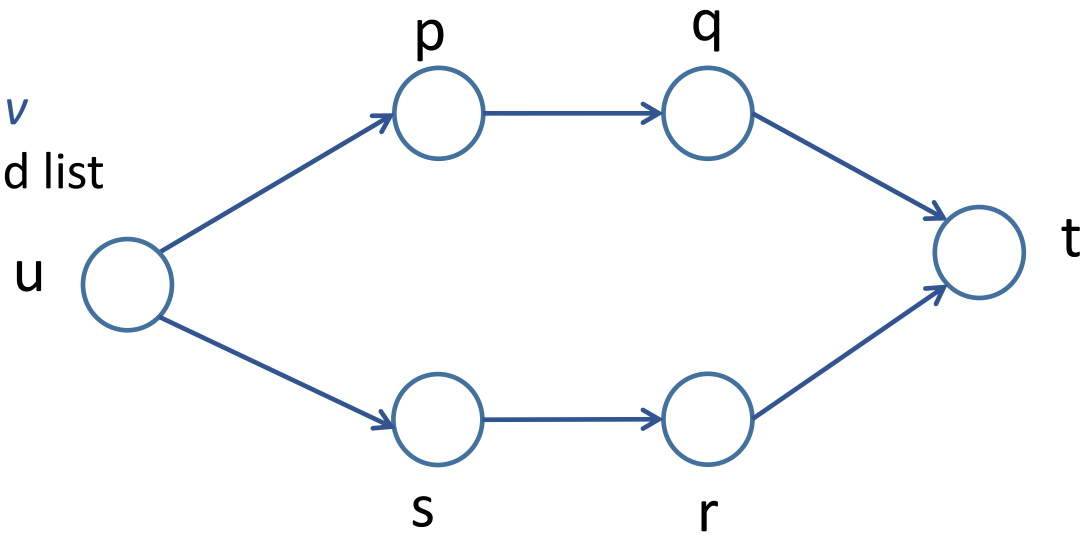
1 call DFS (G) to compute finished times *v. f* for each vertex *v*
2 as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices

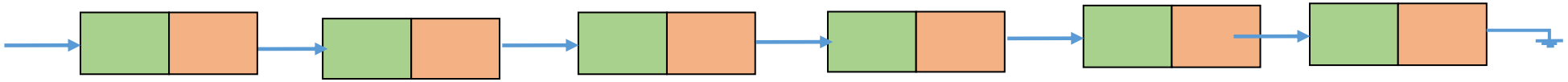


Graph Traversal Applications: DFS

TOPOLOGICAL – SORT (G)

1 call DFS (G) to compute finished times *v. f* 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 – Sorting : Time Complexity analysis

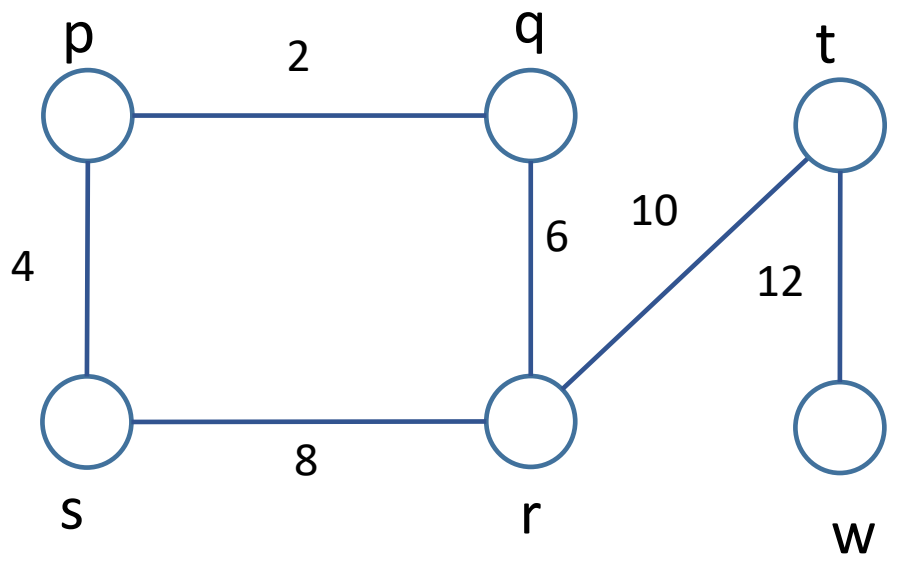
TOPOLOGICAL – SORT (G)

1 call DFS (G) to compute finished times *v. f* 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

→ *O (n+m)*
→ *Insertion into linked list O (1)*
→ *For n vertices* → *O (n)*
→ *To return* → *O (1)*

Total time complexity = *O (n+m) + O (n) + O (1)*
= *O (n+m)*
= *O (V+E)*

Spanning Tree (1)



- Weighted graph
- Undirected graph
- What if we get an unweighted graph?
  - ✓ *Implicit definition*: Every edge in unweighted graph contains a weight "1"
    - Example: BFS of unweighted graph → every edge weight is "1"

- Spanning Tree: A ***subgraph*** T of a undirected graph G = (V, E) is a ***spanning tree*** of G if it is a tree and contains every vertex of G

Vertices → T. V = G. V

Edges → T. E ⊆ G. E

where T → Tree (Spanning) → Acyclic connected graph

G → Graph

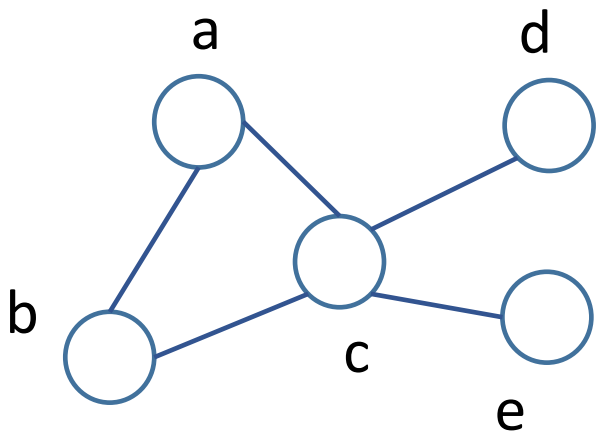
Spanning Tree (2)

- Spanning Tree: A ***subgraph*** T of a undirected graph G = (V, E) is a ***spanning tree*** of G if it is a tree and contains every vertex of G

Vertices → T. V = G. V

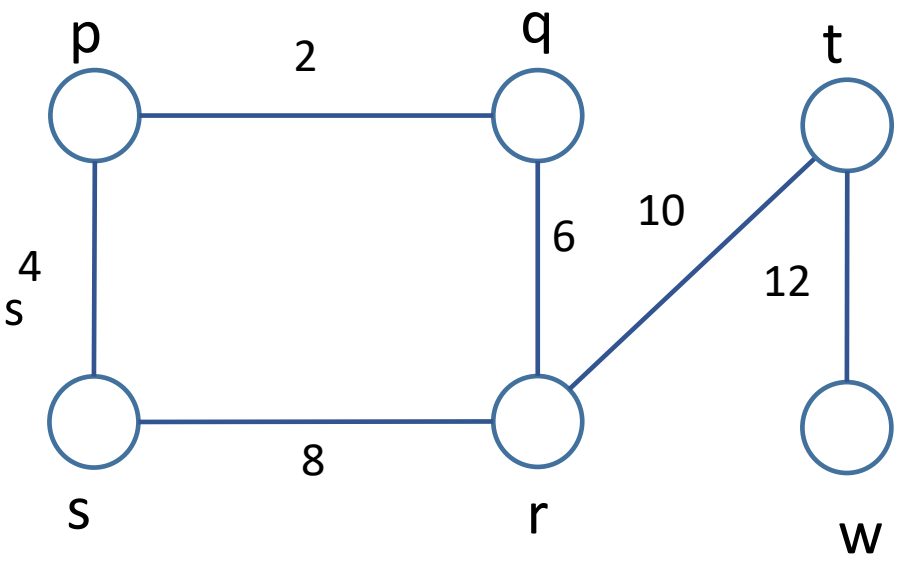
Edges → T. E ⊆ G. E

where T → Tree (Spanning) → Acyclic connected graph, G → Graph

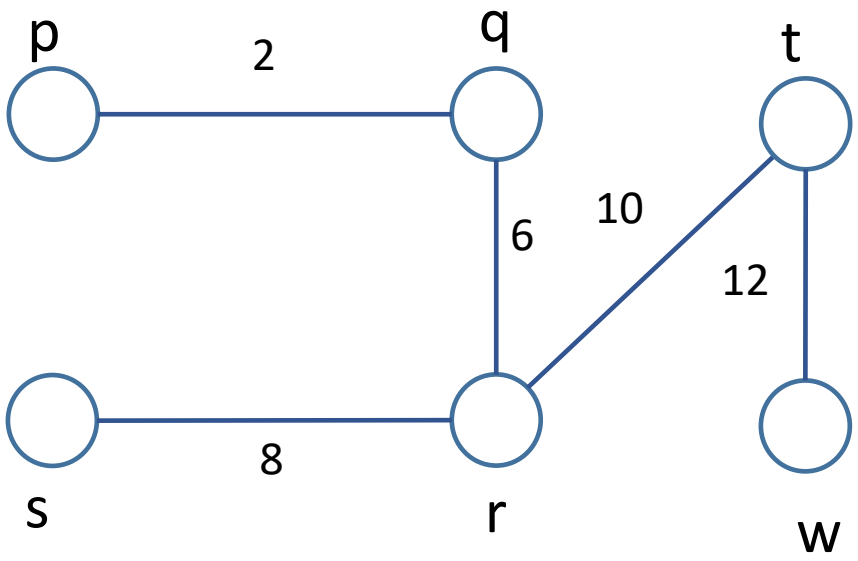


G → Graph

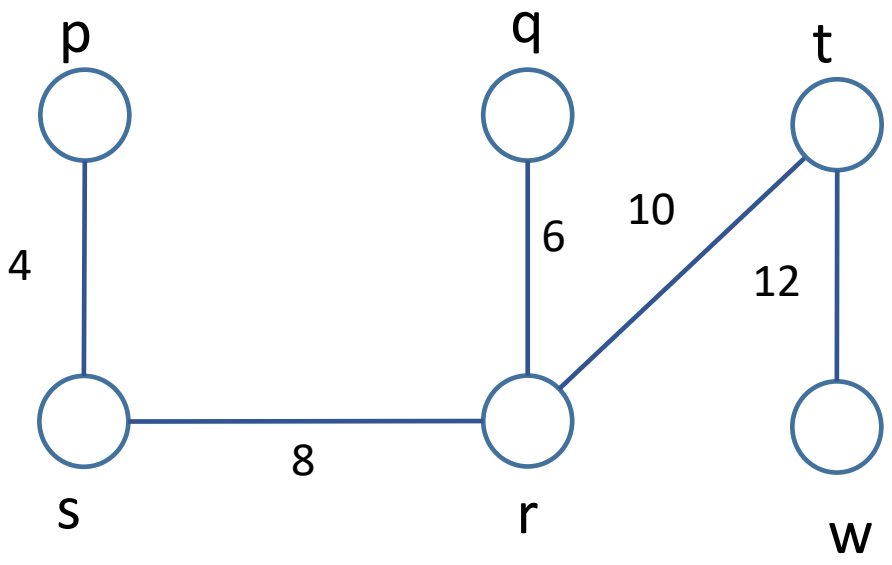
Spanning Tree (3)



- Weighted graph: A weighted graph is a graph, in which each edge has a weight
- Weight of graph: The sum of the weights of all the edges

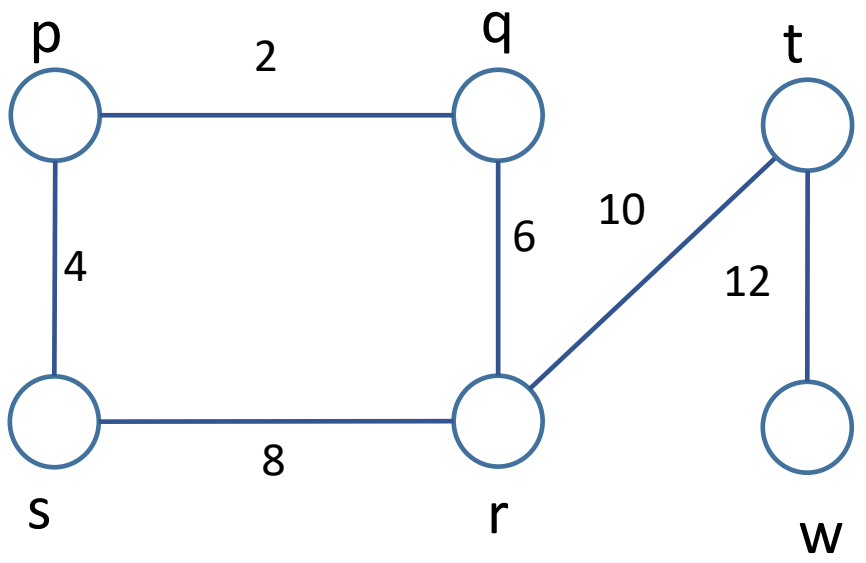


Spanning Tree → T1; W→ 38

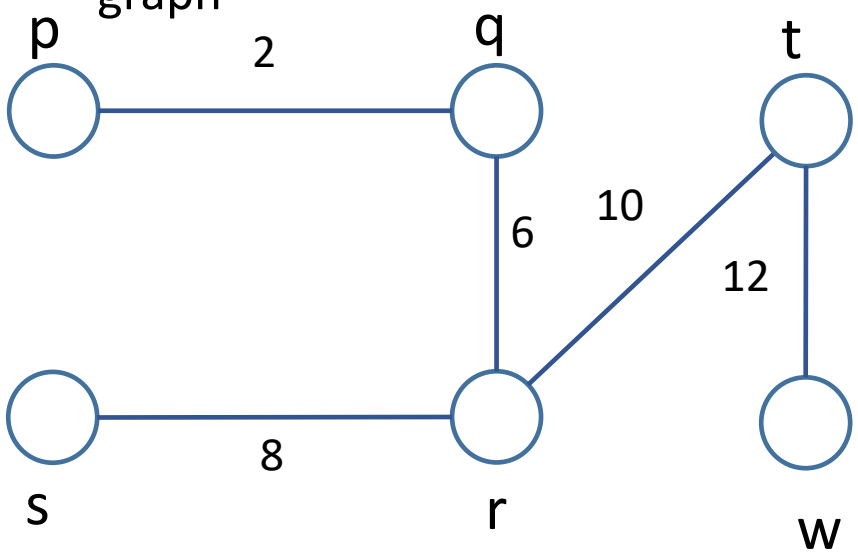


Spanning Tree → T1; W→ 40

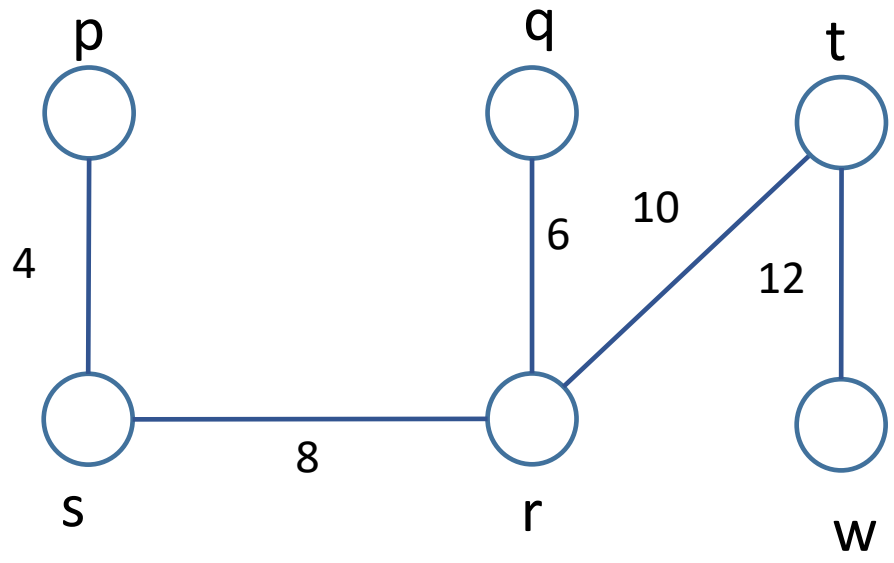
Minimum Spanning Tree (4)



- Minimum Spanning Tree (MST):
  - ✓ A MST in an undirected connected weighted graph is a spanning tree of ***minimum weight*** among all spanning trees.
  - ✓ Length of the spanning tree = sum of weights of all edges in T → L_1
  - ✓ G → multiple spanning trees T_1 T_2 T_3 … L_1 L_2 L_3
  - ✓ A spanning tree with *smallest possible length*;
  - ✓ *Note:* multiple minimum spanning trees are possible for a given graph



Spanning Tree → T1; L_1 → 38

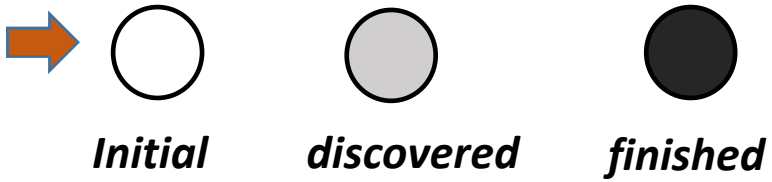


Spanning Tree → T2; L_2 → 40

Minimum Spanning Tree

Minimum Spanning Tree (MST)

- Different algorithms → usage of data structure
  - ✓ Prim's algorithm
    - Algorithm → Priority Queue (Min. Heap)
    - Implementation
    - Time complexity analysis
  - ✓ Kruskal's algorithms
    - Algorithm → Sets
    - Implementation
    - Time complexity analysis



MST → Prim's algorithm (1)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.              u = EXTRACT-MIN(Q)
8.              for each v ϵ G.Adj[u]
9.                      if v ϵ Q and w(u,v) < v.key
10                              v. π = u
11                              v.key = w(u,v)
        }
```

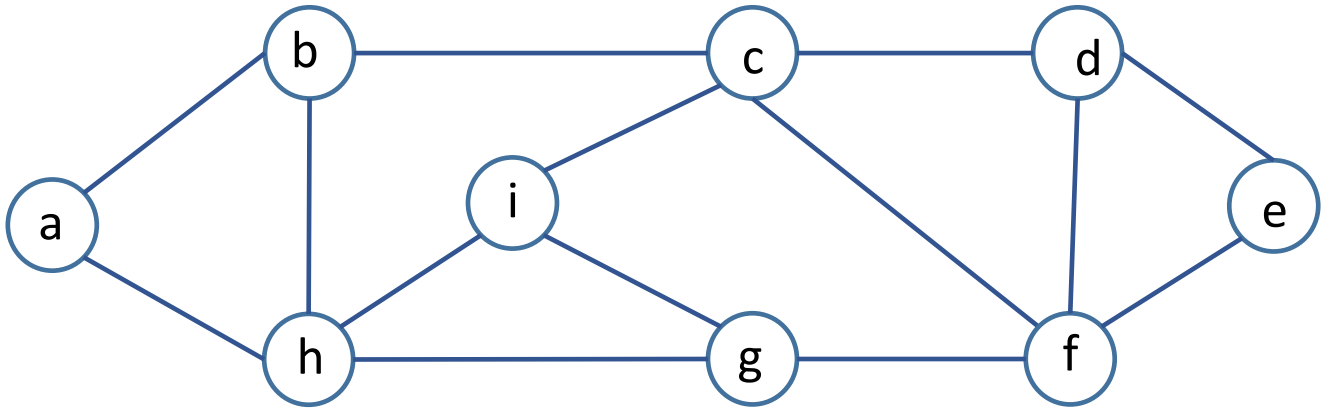
G: A connected graph with any vertex r to be the root of the tree

Q: *min_priority* Q based on key attribute

V: Vertex

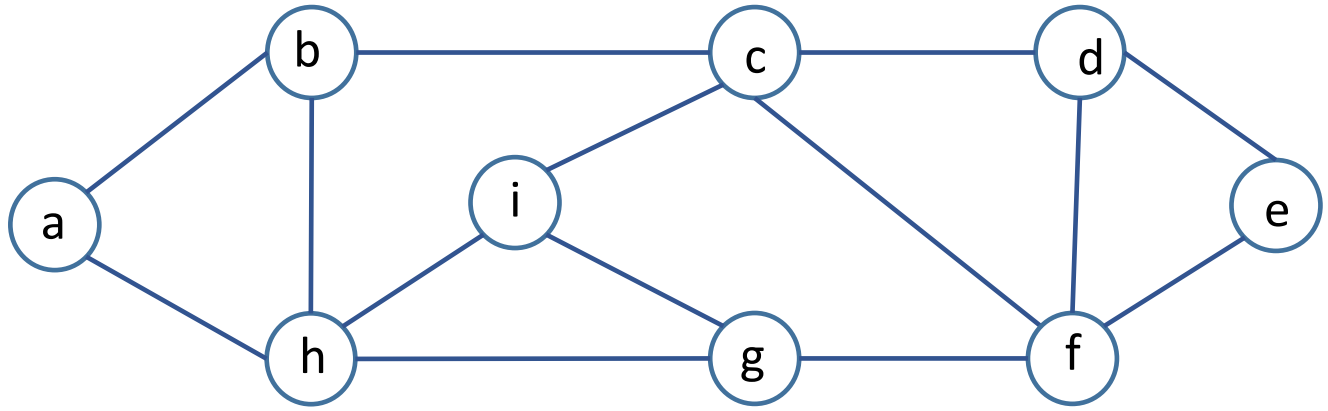
v. key : The minimum weight of any edge connecting v to a vertex in the tree

v. π: The predecessor of *v* in the tree

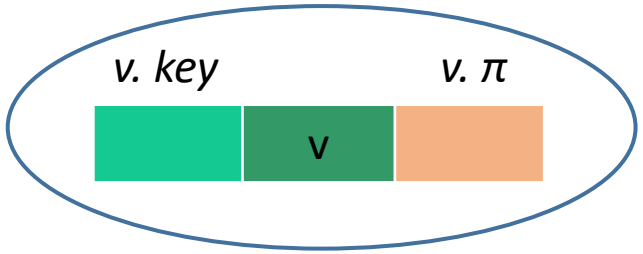


MST → Prim's algorithm (2)

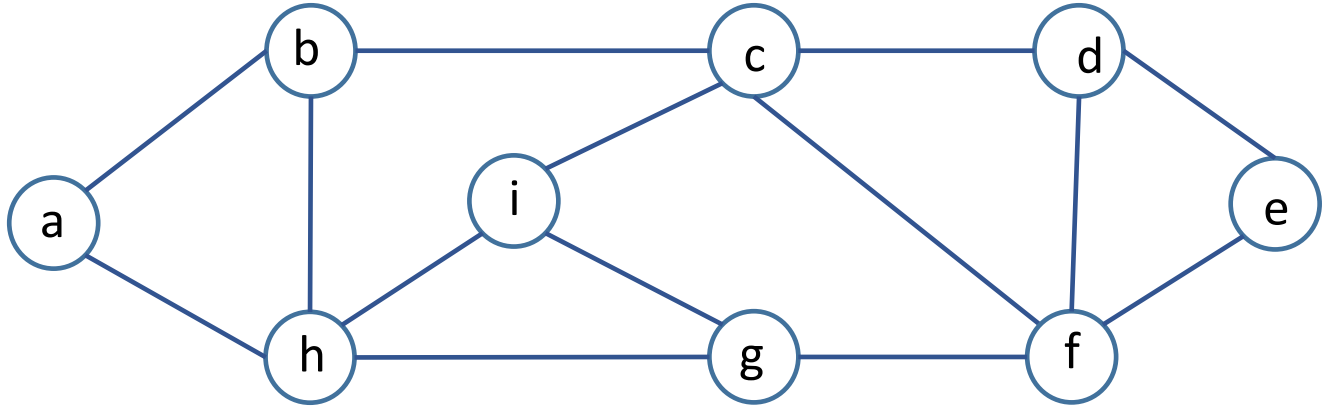
```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.              u = EXTRACT-MIN(Q)
8.              for each v ϵ G.Adj[u]
9.                      if v ϵ Q and w(u,v) < v.key
10                              v. π = u
11                              v.key = w(u,v)
        }
```

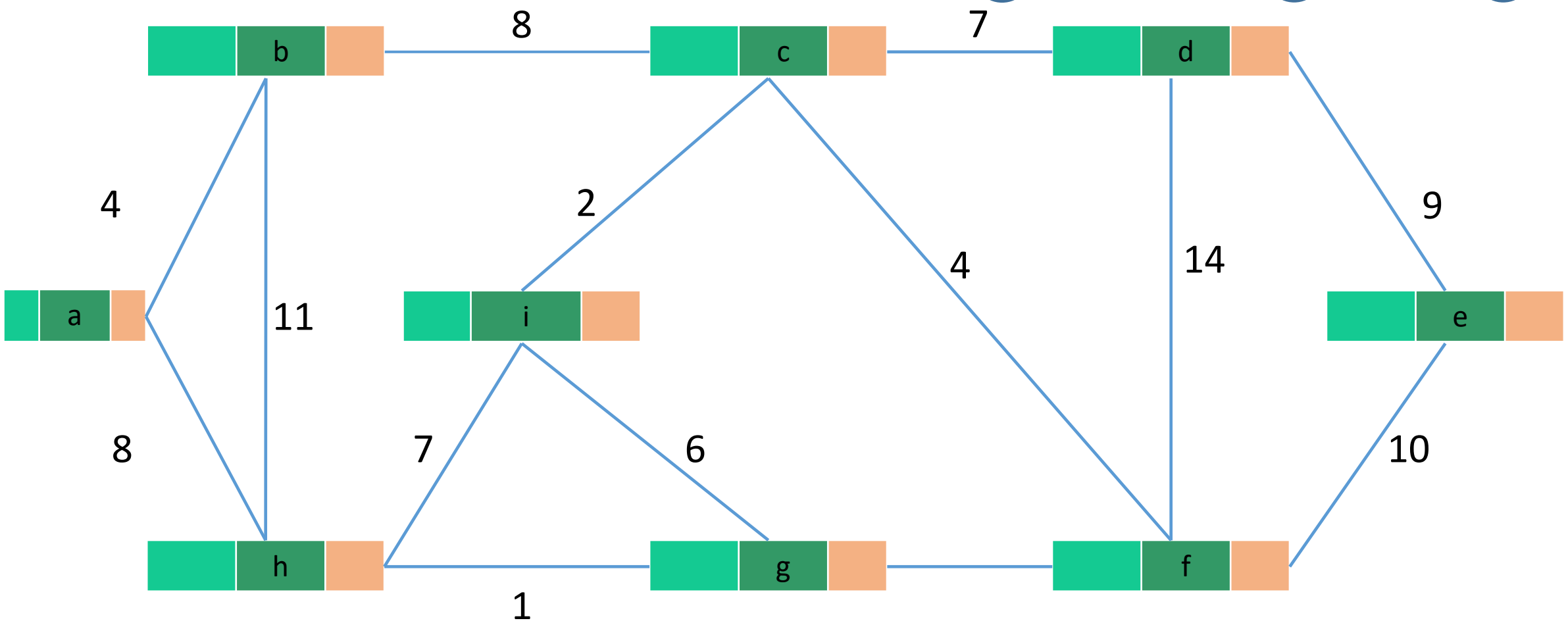


Representation:



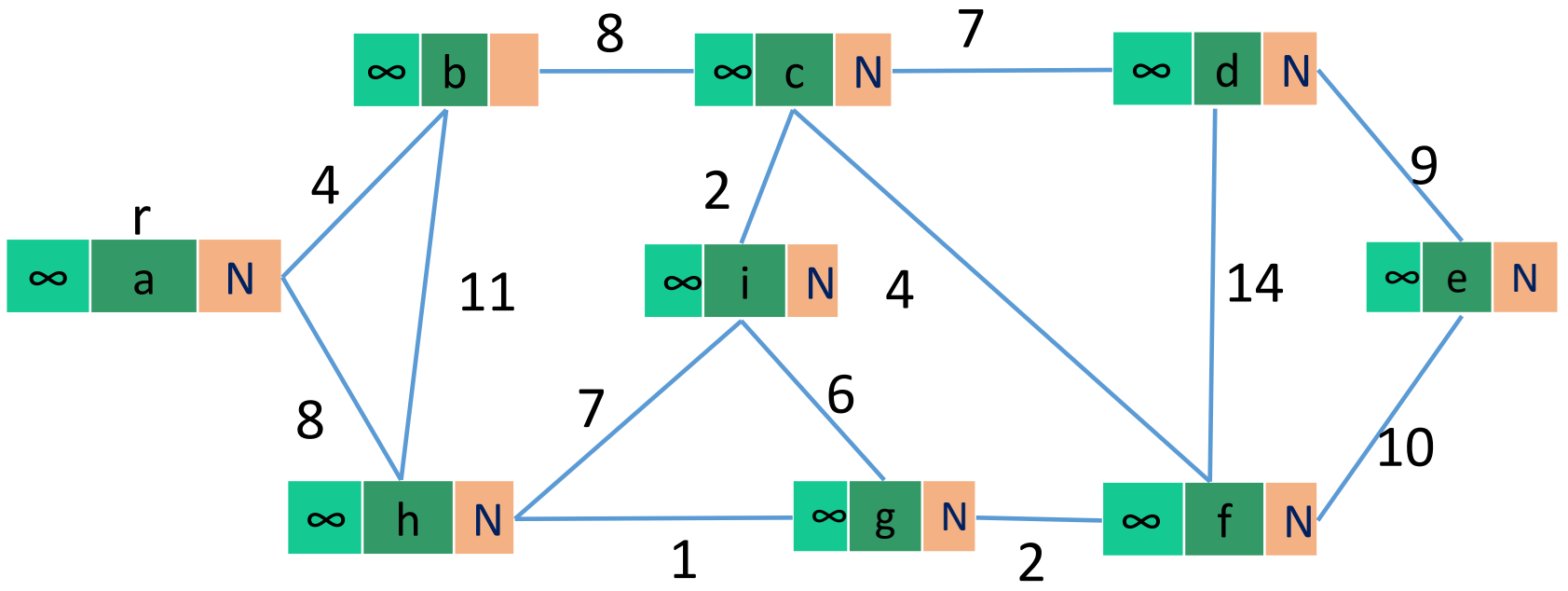
MST → Prim's algorithm (3)





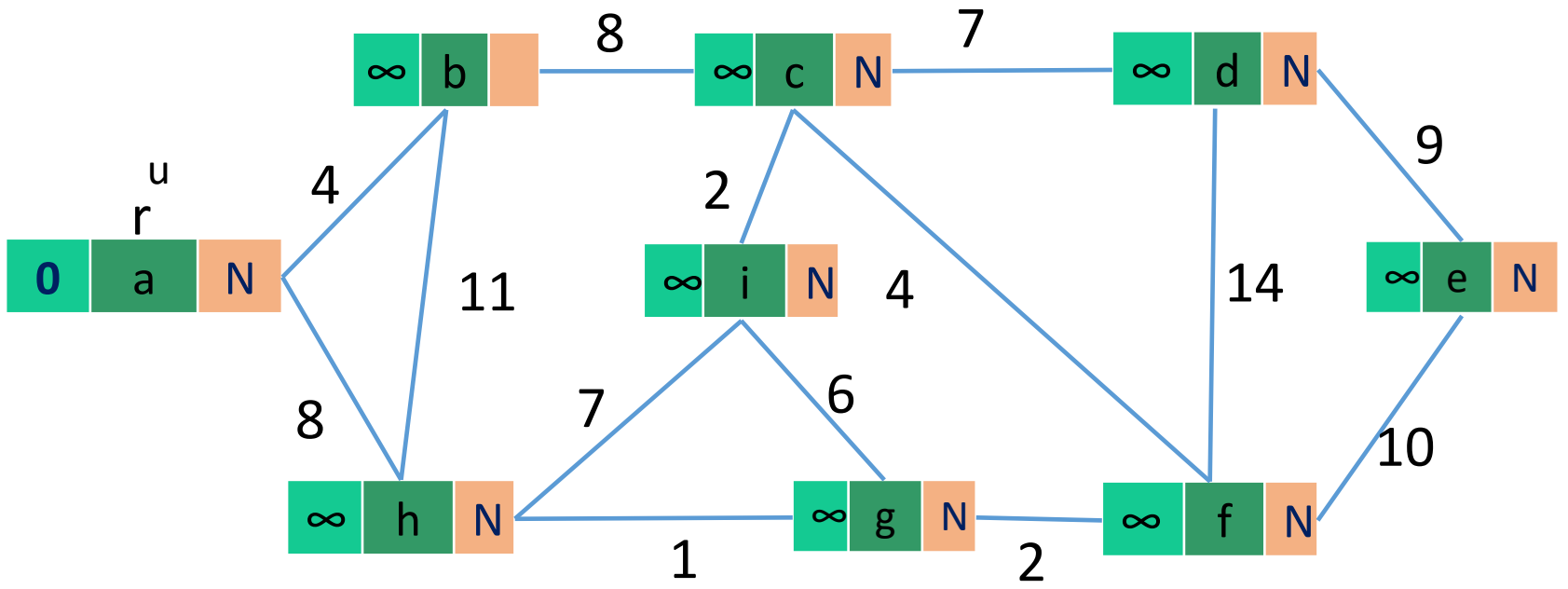
MST → Prim's algorithm (4)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G.Adj[u]
9.          if v ϵ Q and w(u,v) < v.key
10                v. π = u
11                v.key = w(u,v)
}
```



MST → Prim's algorithm (5)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u. key = ∞
3.              u. π = NIL
4.      r. key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11                v.key = w(u,v)
}
```

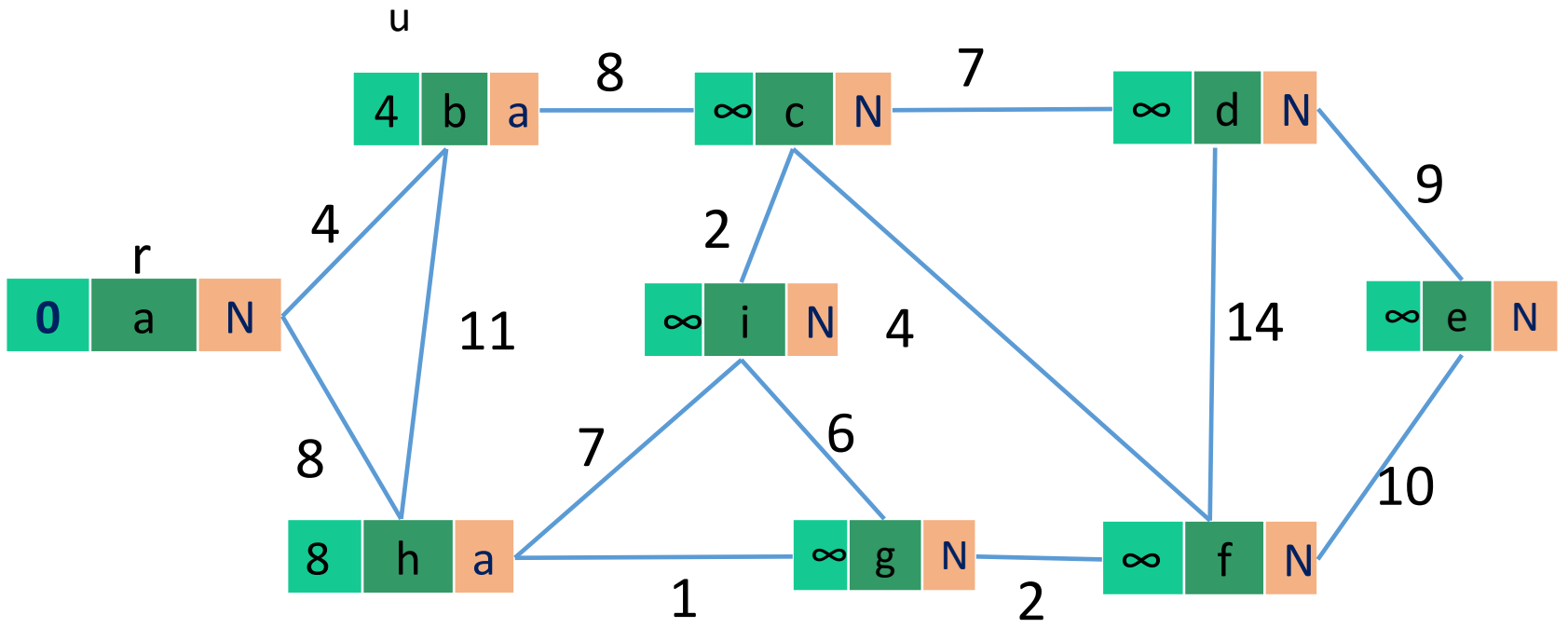


Step 5: Q={a, b, c, d, e, f, g, h, i}
Step 6*: Q is not Empty*
Step 7: *u =* ***a***
Step 8: *v = G. Adj [a] = {b, h}*
Step 9: *w (a, b) < b. key*
4 < ∞
Step 10: *b. predecessor = a*
Step 11: *b. key = w (a, b) = 4*

Step 9: *w (a, h) < h. key*
8 < ∞
Step 10: *h. predecessor = a*
Step 11: *h. key = w (a, h) = 8*

MST → Prim's algorithm (6)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11                v.key = w(u,v)
}
```

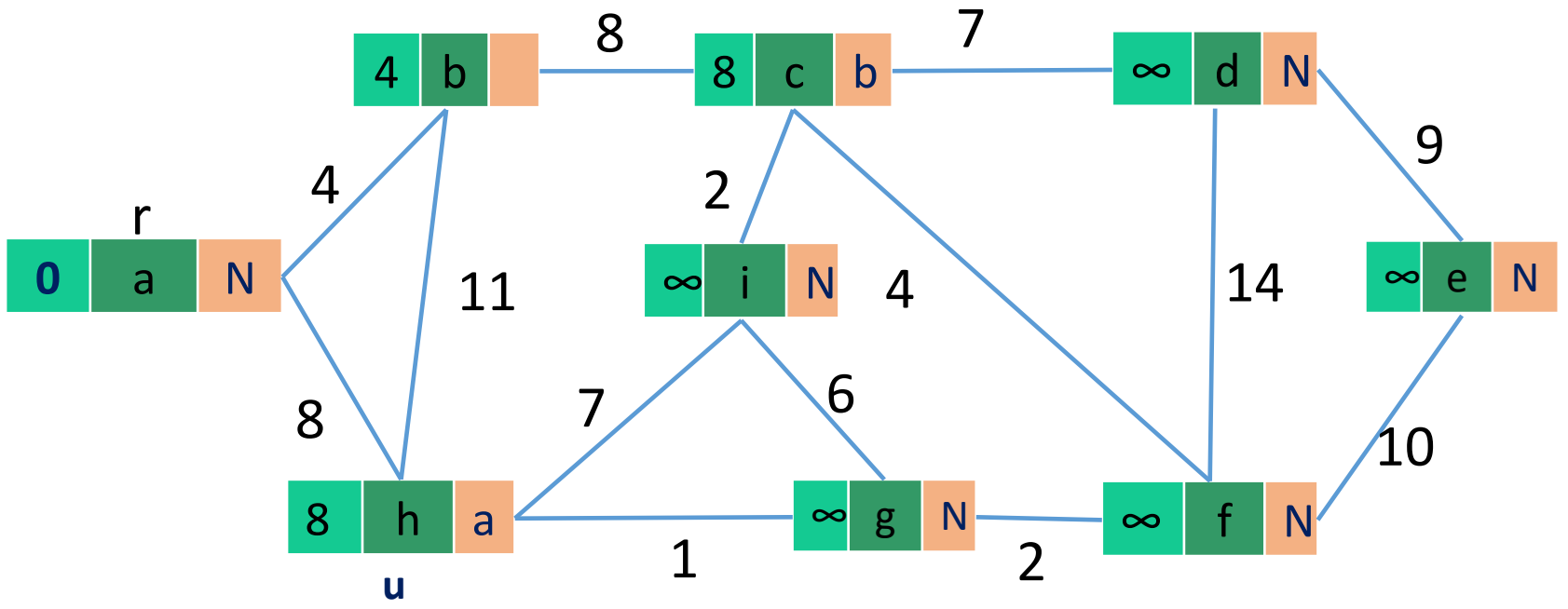


Step 5: Q={b, c, d, e, f, g, h, i}
Step 6*: Q is not Empty*
Step 7: *u =* ***b***
Step 8: *v = G. Adj [b] = {h, c}*
Step 9: *w (b, h) < h. key*
11 < 8
Step 10:
Step 11:

Step 9: *w (b, c) < c. key*
8 < ∞
Step 10: *c. predecessor = b*
Step 11: *c. key = w (b, c) = 8*

MST → Prim's algorithm (7)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11                v.key = w(u,v)
}
```

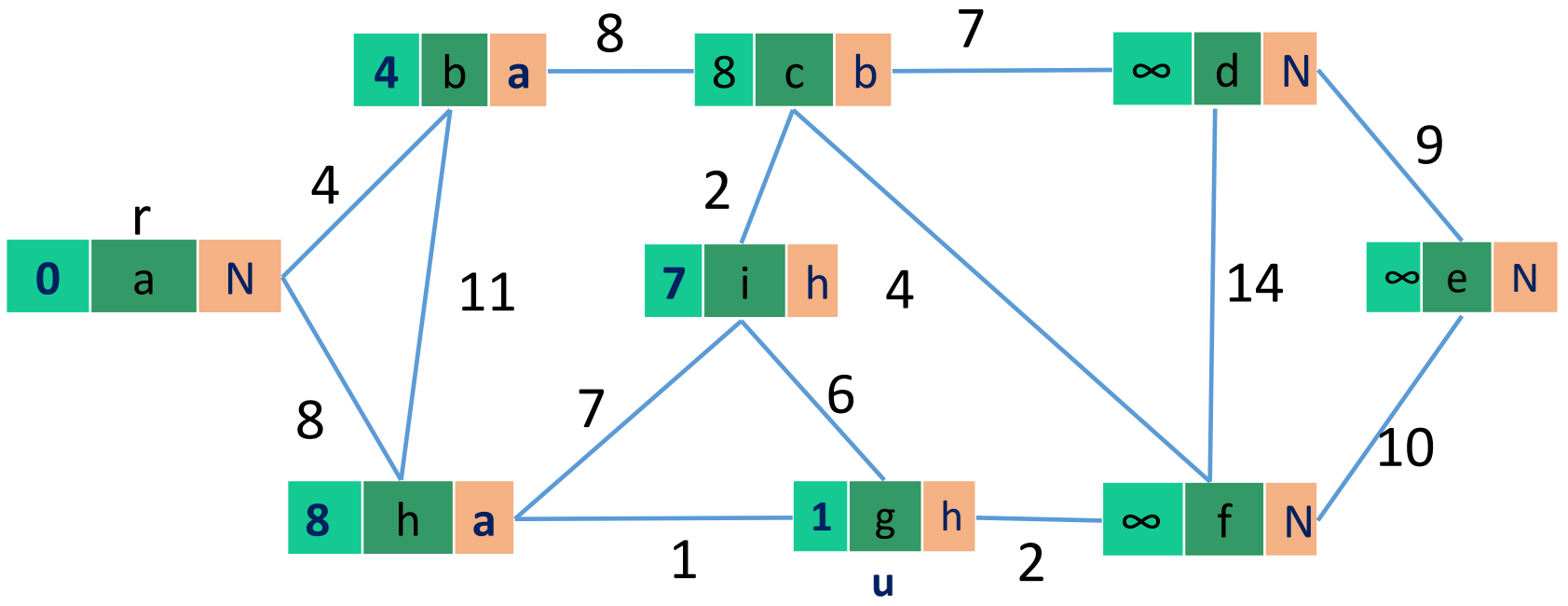


Step 5: Q={c, d, e, f, g, h, i}
Step 6*: Q is not Empty*
Step 7: *u =* ***h***
Step 8: *v = G. Adj [h] = {a, b, i, g}*
Step 9: *w (h, i) < i. key*
7 < ∞
Step 10: *i. predecessor = h*
Step 11: ***i****. key = w (h, i) = 7*

Step 9: *w (h, g) < g. key*
1 < ∞
Step 10: *g. predecessor = h*
Step 11: *g. key = w (h, g) = 1*

MST → Prim's algorithm (8)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11                v.key = w(u,v)
}
```

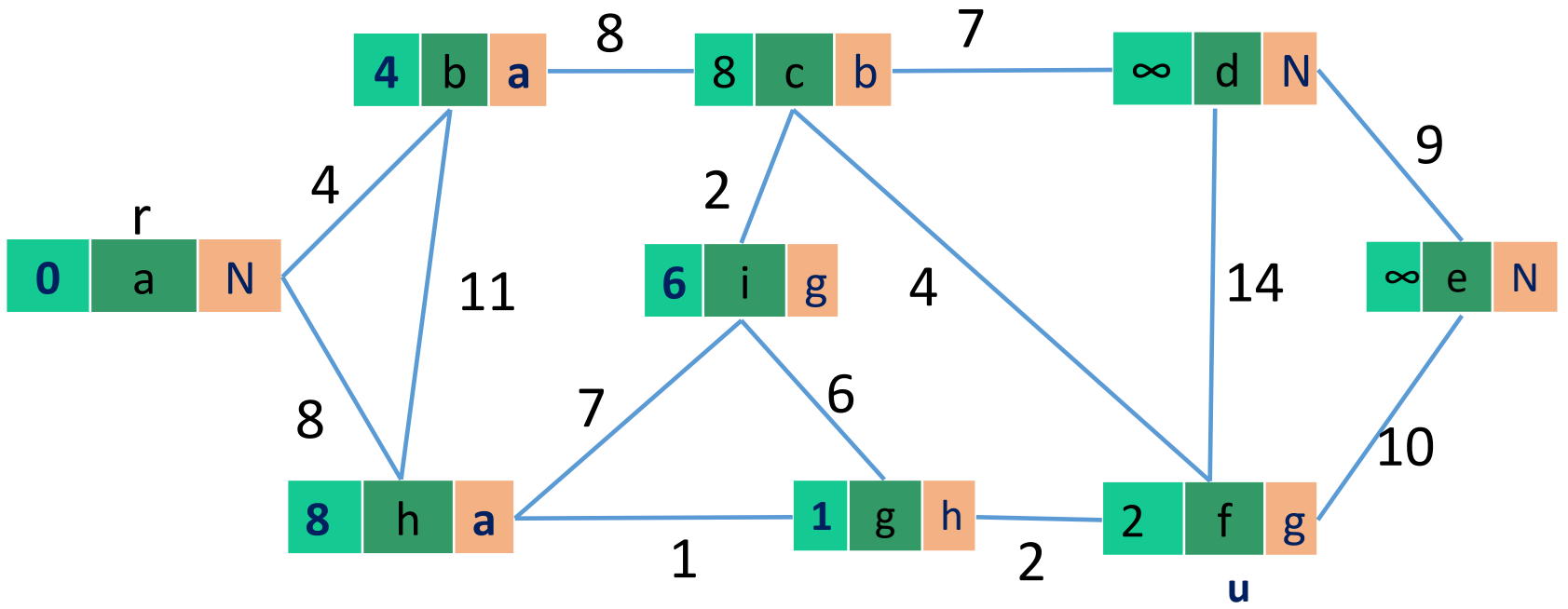


Step 5: Q={c, d, e, f, g, i}
Step 6*: Q is not Empty*
Step 7: *u = **g***
Step 8: *v = G. Adj [g] = {h, i, f}*
Step 9: *w (g, f) < f. key*
2 < ∞
Step 10: *f. predecessor = g*
Step 11: *f. key = w (g, f) = 2*

Step 9: *w (g, i) < i. key*
6 < 7
Step 10: *i. predecessor = g*
Step 11: *i. key = w (h, g) = 6*

MST → Prim's algorithm (9)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                  v. π = u
11                  v.key = w(u,v)
}
```



Step 5: Q={c, d, e, f, i}

Step 6: *Q is not Empty*

Step 7: *u = f*

Step 8: *v = G. Adj [f] = {g, c, d, e}*

Step 9: *w (f, c) < c. key*
4 < 8

Step 10: *c. predecessor = f*

Step 11: *c. key = w (f, c) = 4*

Step 9: *w (f, d) < d. key*
14 < ∞

Step 10: *d. predecessor = f*

Step 11: *d. key = w (f, d) = 14*

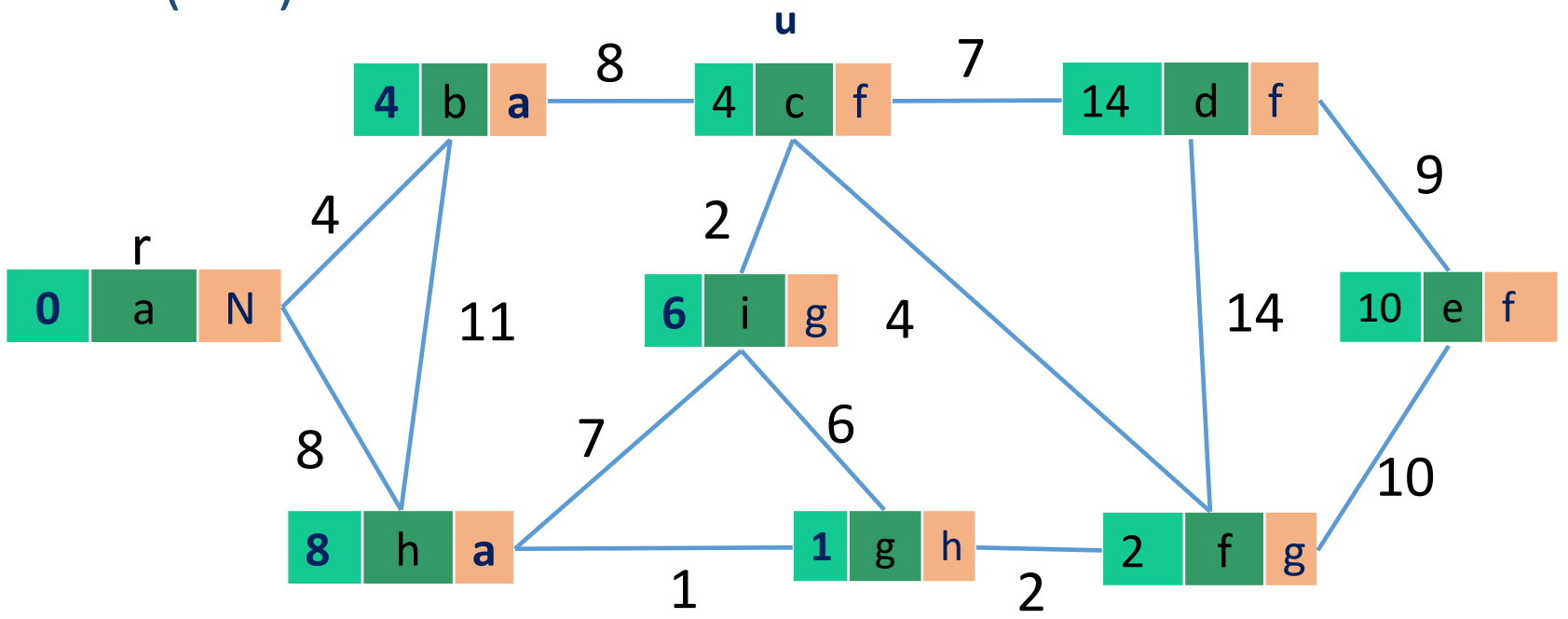
w (f, e) < e. key
10 < ∞

e. predecessor = f

e. key = w (f, d)
= 10

MST → Prim's algorithm (10)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11                v.key = w(u,v)
}
```



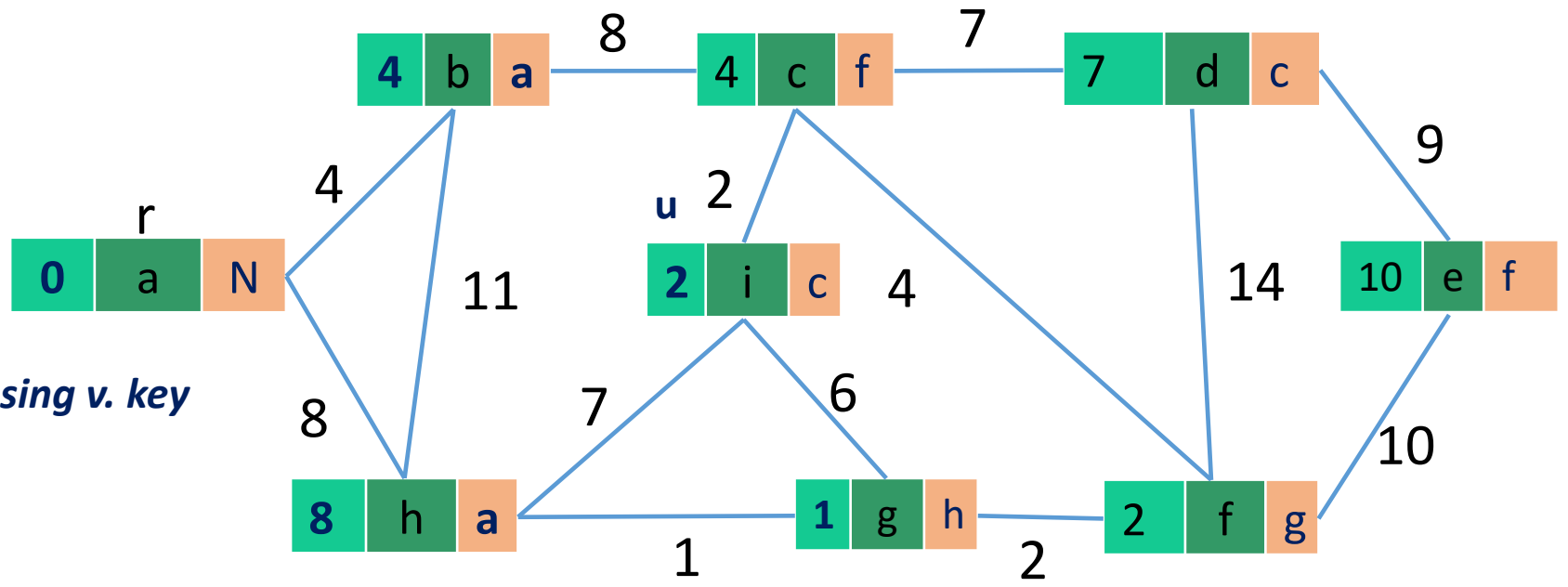
Step 5: Q={c, d, e, i}
Step 6: *Q is not Empty*
Step 7: *u = **c***
Step 8: *v = G. Adj [c] = {b, i, d, f}*
Step 9: *w (c, i) < i. key*
2 < 6
Step 10: *i. predecessor = c*
Step 11: *i. key = w (c, i) = 2*

Step 9: *w (c, d) < d. key*
7 < 14
Step 10: *d. predecessor = c*
Step 11: *d. key = w (c, d) = 7*

MST → Prim's algorithm (11)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11              v.key = w(u,v)
}
```

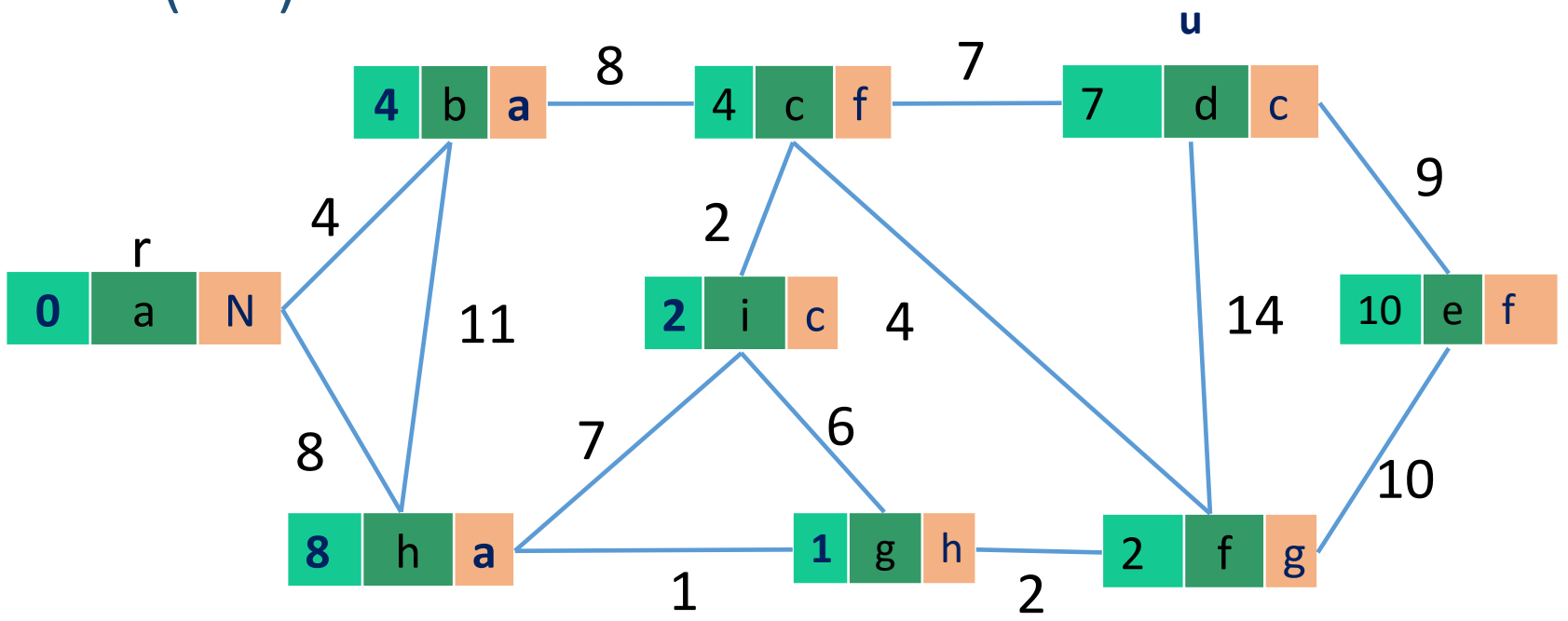
Min-heap using v. key



Step 5: Q={d, e, i}
Step 6*: Q is not Empty*
Step 7: *u = **i***
Step 8: *v = G. Adj [i] = {h, c, g}*
Step 9:
Step 10:
Step 11:

MST → Prim's algorithm (12)

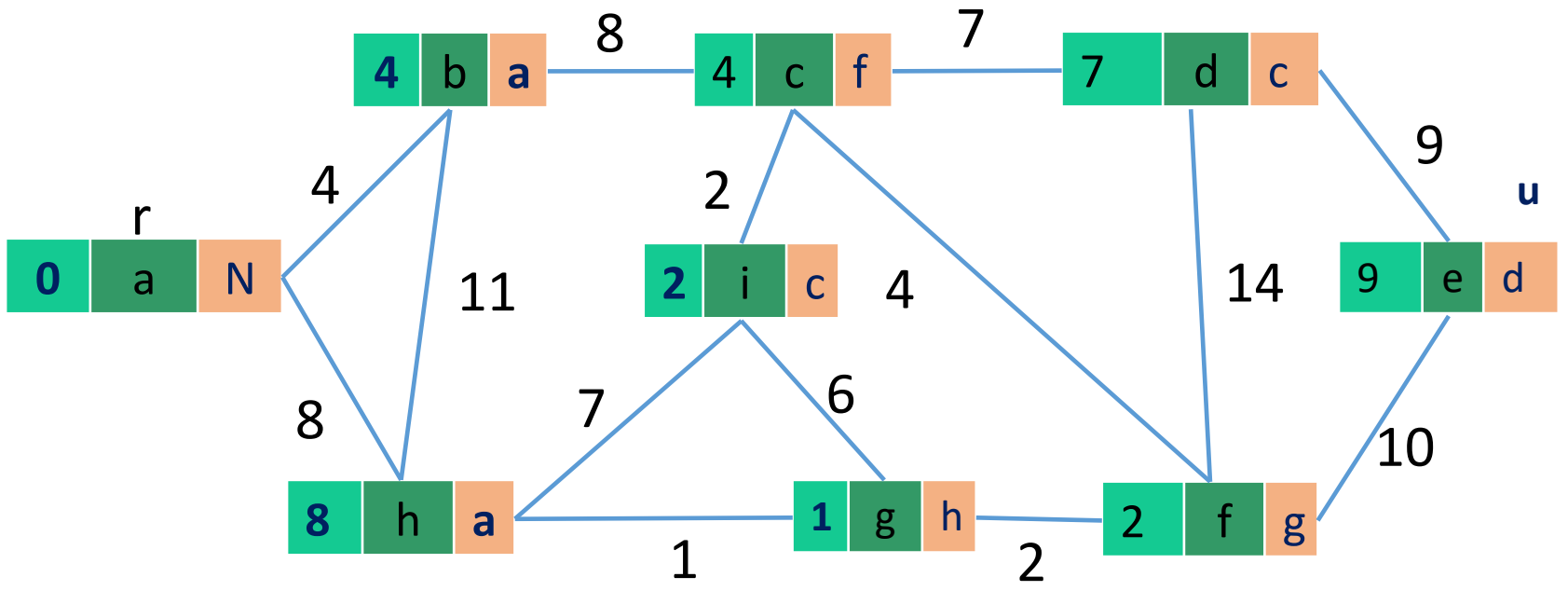
```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                  v. π = u
11                  v.key = w(u,v)
}
```



Step 5: Q={d, e}
Step 6*: Q is not Empty*
Step 7: *u =* ***d***
Step 8: *v = G. Adj [d] = {c, f, e}*
Step 9: *w (d, e) < e. key*
9 < 10
Step 10: ***e****. predecessor = d*
Step 11: *e. key = w (d, e) = 9*

MST → Prim's algorithm (13)

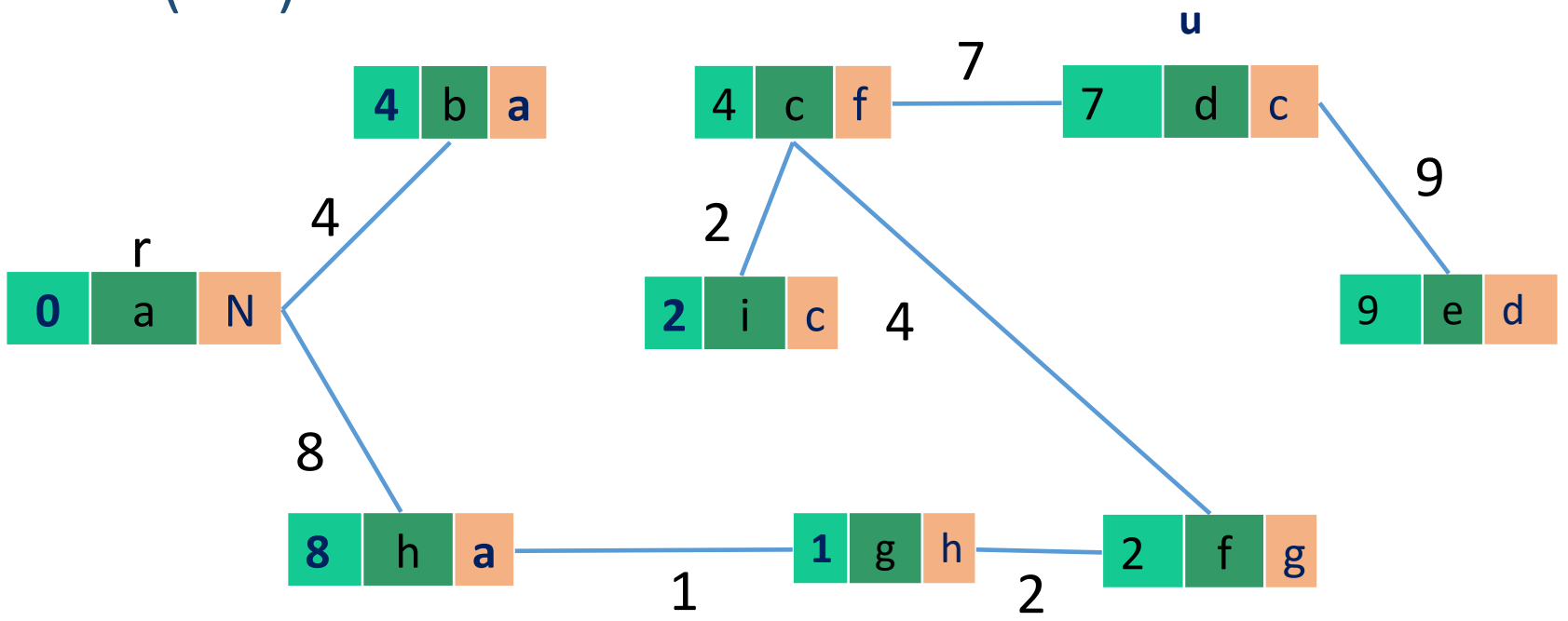
```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11               v.key = w(u,v)
}
```



Step 5: Q={e}
Step 6*: Q is not Empty*
Step 7: *u =* ***e***
Step 8: *v = G. Adj [d] = {f, d}*
Step 9:
Step 10:
Step 11:

MST → Prim's algorithm (14)

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11                v.key = w(u,v)
}
```



MST → Prim's algorithm→ Time complexity analysis

```
MST-PRIM (G, w, r) {
1.      for each u ϵ G.V
2.              u.key = ∞
3.              u. π = NIL
4.      r.key = 0
5.      Q = G.V
6.      while Q ≠ ∅ ;
7.        u = EXTRACT-MIN(Q)
8.        for each v ϵ G. Adj[u]
9.          if v ϵ Q and w(u, v) < v. key
10                v. π = u
11                v.key = w(u,v)
}
```

O(V) if Q is implemented as a min-heap

O(log V)

|V| times

O(log V)

O(V logV)

|E| times

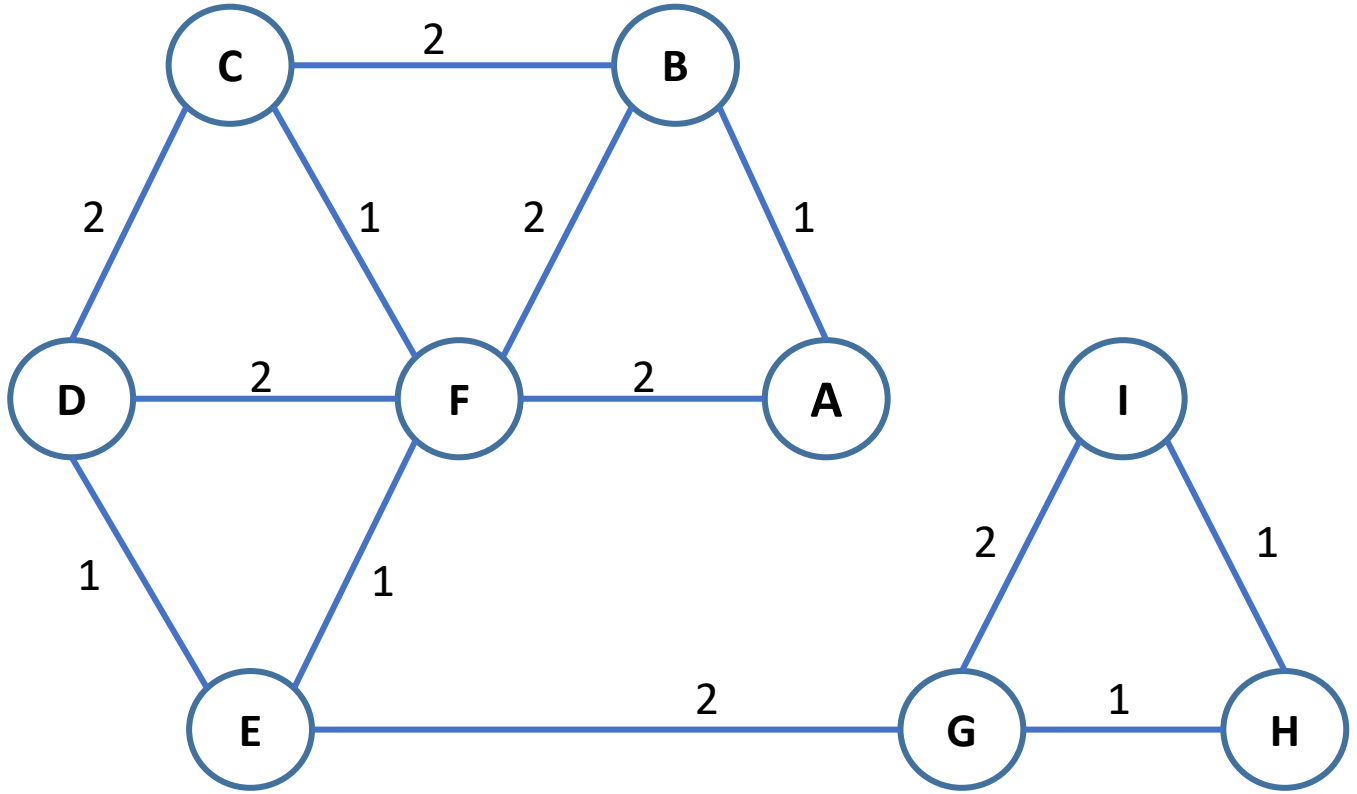
O(log V)

O(E logV)

Total time complexity: O(V logV + E logV) = O((V+E) logV) = O(E logV)

Exercise: Prim's algorithm

1. What will the path obtained after applying Prim's algorithm with a starting *vertex C ?*



2. The number of distinct minimum spanning trees for the weighted graph below is
   a. 4
   b. 5
   c. 6
   d. 7

thank you!

email:
k.kondepu@iitdh.ac.in