Prim's Algorithm

- Special case of the generic minimum-spanning-tree method.
- A greedy algorithm since at each step it adds to the tree an edge that contributes the minimum amount possible to the tree's weight.
- The edges in the set A always form a single tree.
- Each step adds to the tree A a light edge that connects A to an isolated vertex – one on which no edge of A is incident.
- By Corollary 23.2, this rule adds only edges that are safe for A

Implementation of Prim's algorithm

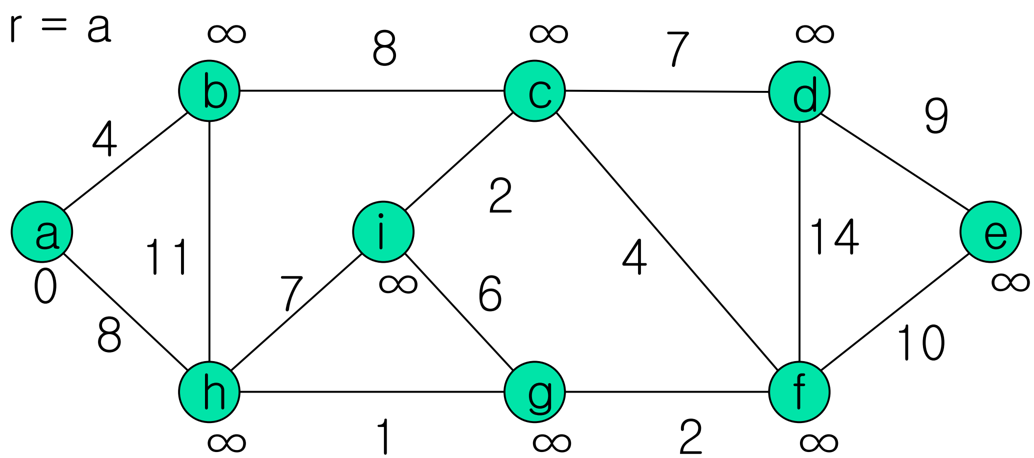
- Input is a connected Graph G and the root r of the minimum spanning tree.
- During execution of the algorithm, all vertices that are not in the tree reside in a min-priority queue Q based on a key attribute.
- For each v, v.key is the minimum weight of any edge connecting v to a vertex in the tree.
- By convention, v.key = ∞ if there is no such edge.
- The attribute v.π names the parent of v in the tree.
- The algorithm maintains the set A from Generic-MST as A=$\{(v, v.\pi)$:$v \in V-\{r\}-Q\}$.
- It terminates when the min-priority queue Q is empty.

Prim’s Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π=NIL
r.key = 0
Q = G.V
while Q ≠ Ø
    u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

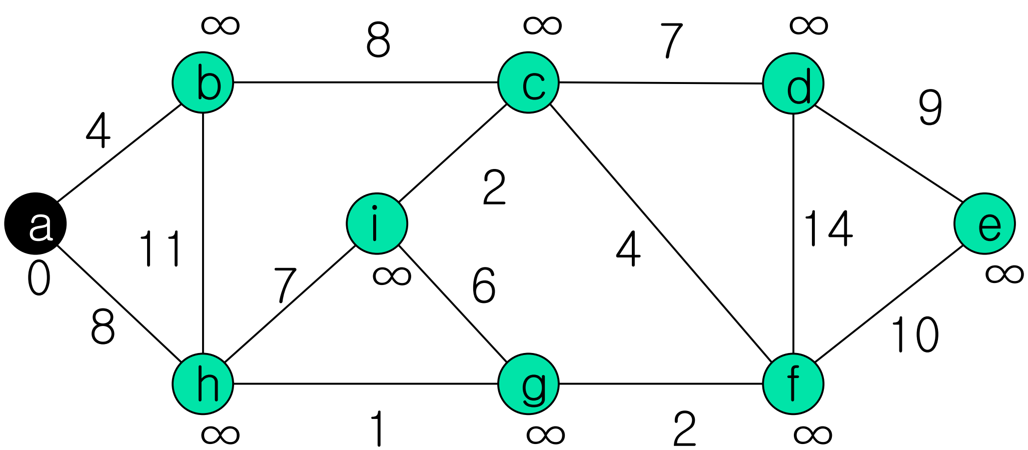
Prim's Algorithm



1. **for** each u ∈ G.V
2.   u.key = ∞
3.   u.π=NIL
4. r.key = 0
5. Q = G.V

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

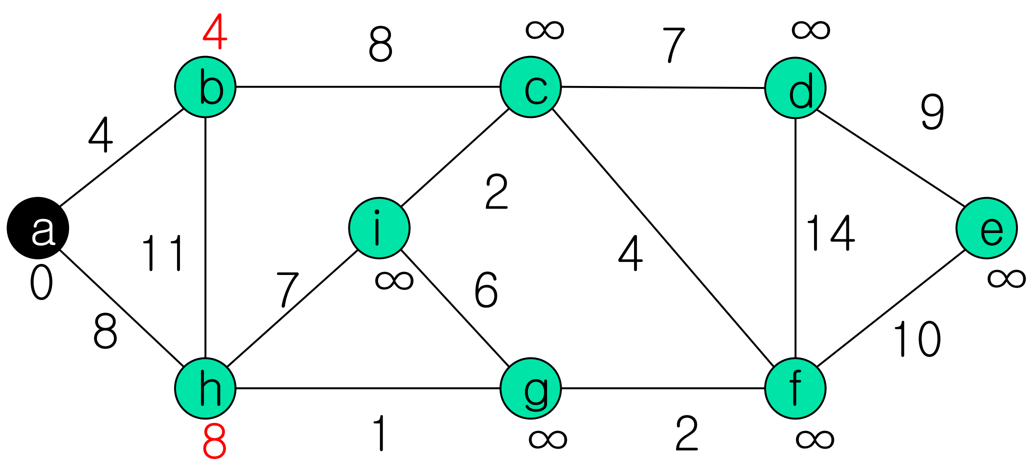
Prim’s Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

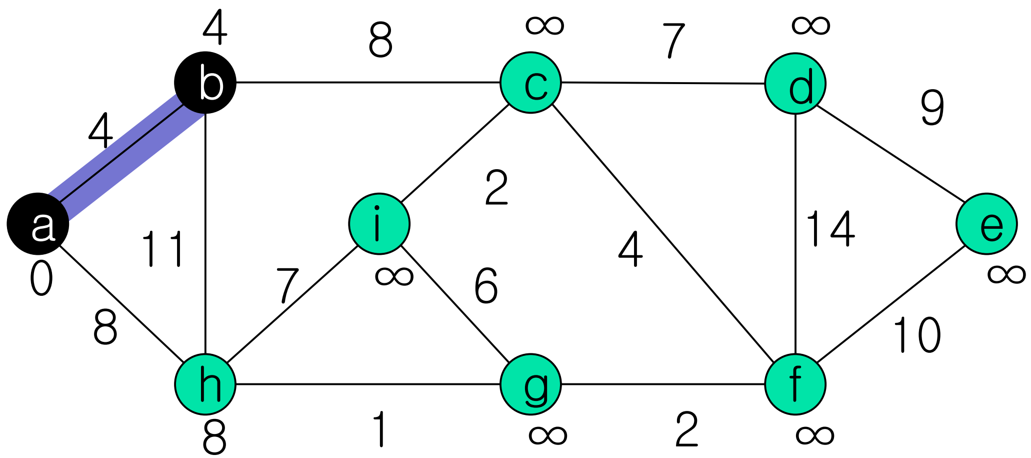
Prim’s Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

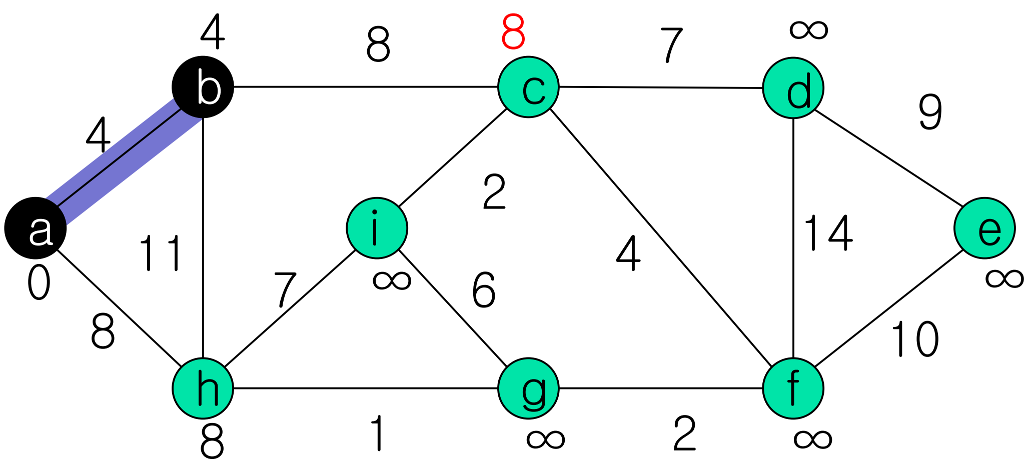
Prim's Algorithm



7. u = Extract-Min(Q)
8. for each v ∈ G.Adj[u]
9. if v ∈ Q and w(u,v) < v.key
10. v.π = u
11. v.key = w(u,v)

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

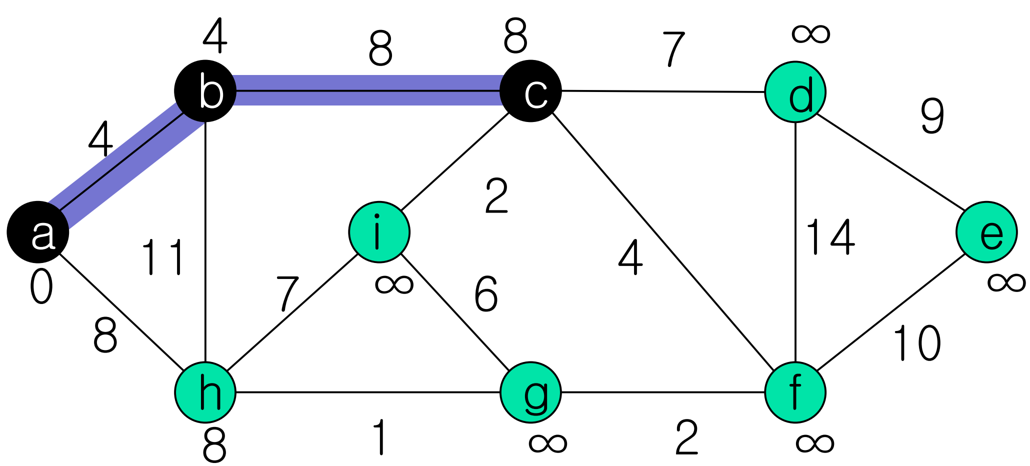
Prim’s Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

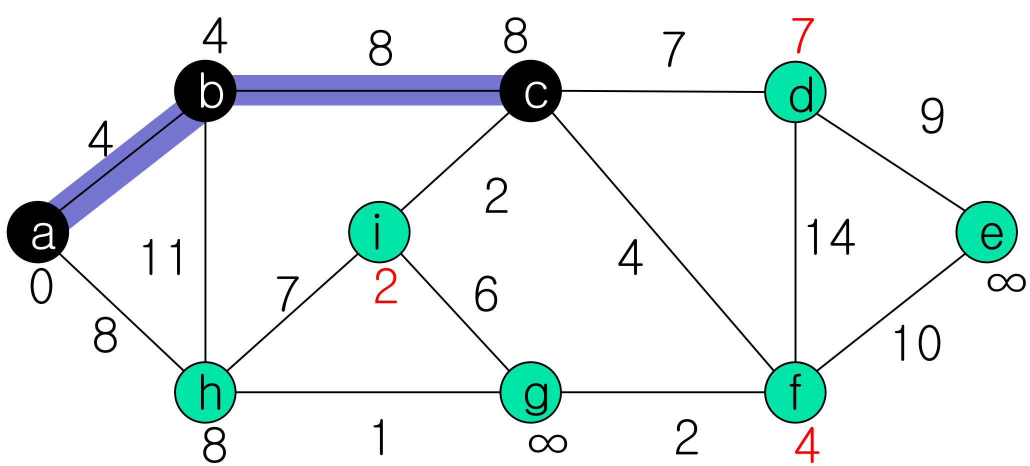
Prim’s Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

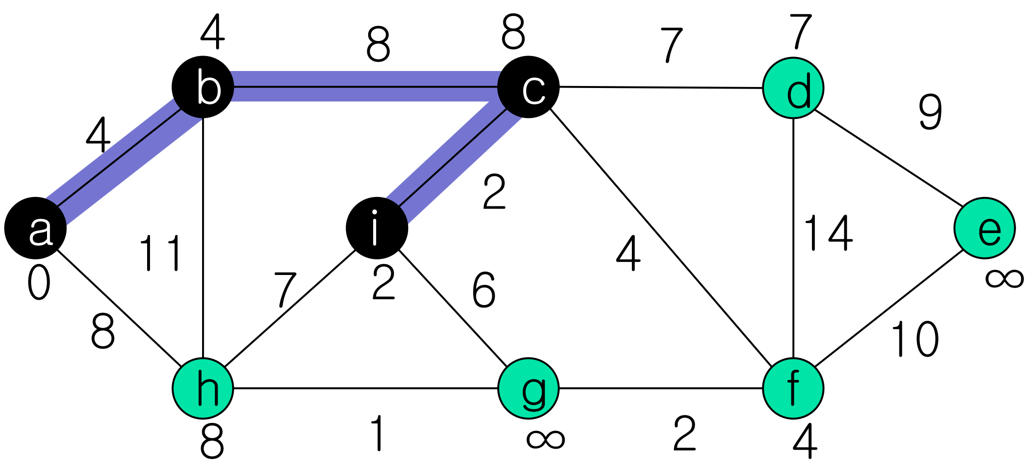
Prim’s Algorithm




```
7.      u = Extract-Min(Q)
8.          for each v ∈ G.Adj[u]
9.              if v ∈ Q and w(u,v) < v.key
10.                 v.π = u
11.                 v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

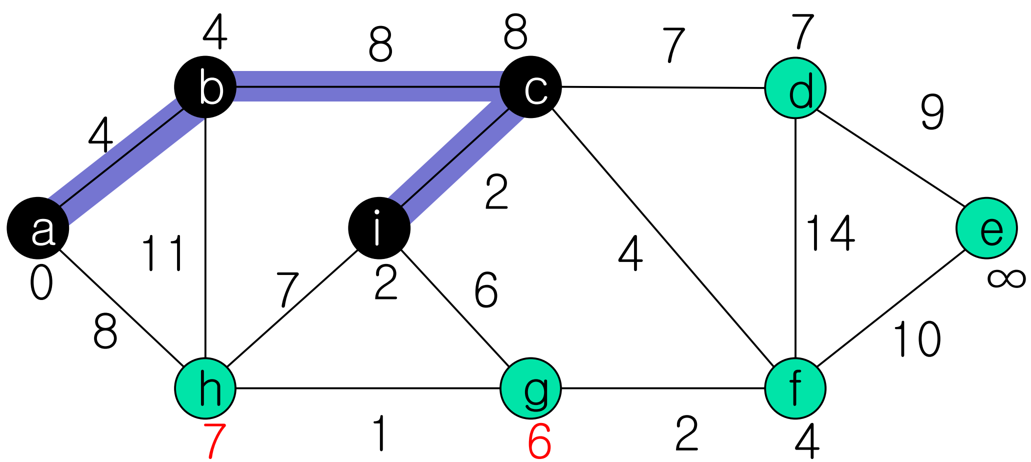
Prim's Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

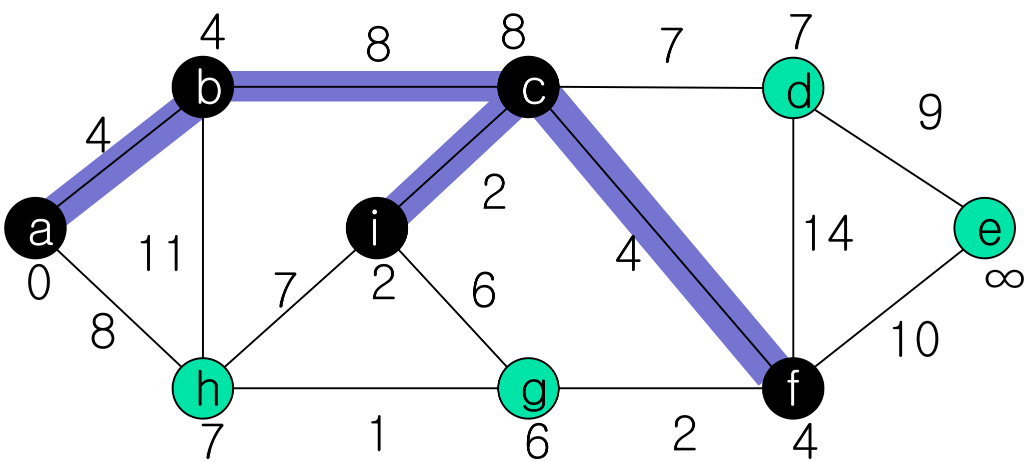
Prim's Algorithm



7. u = Extract-Min(Q)
8. for each v ∈ G.Adj[u]
9. if v ∈ Q and w(u,v) < v.key
10. v.π = u
11. v.key = w(u,v)

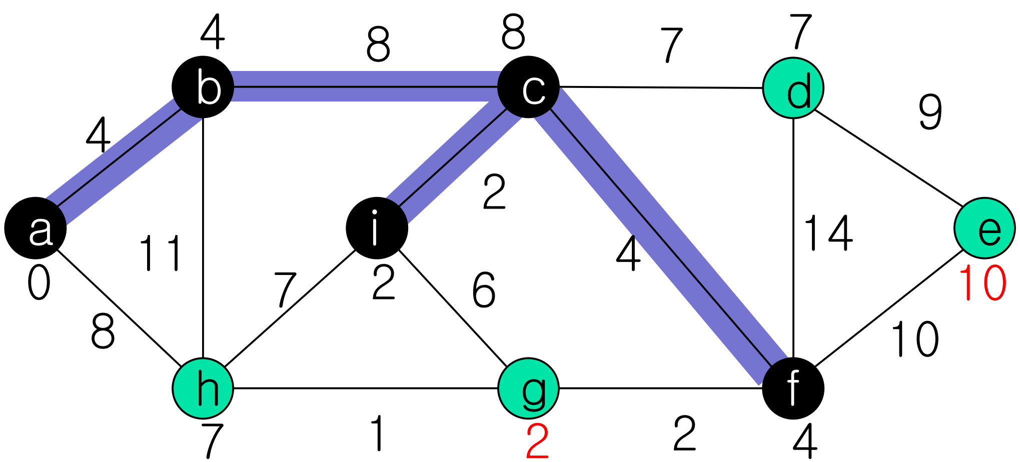
Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Prim's Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

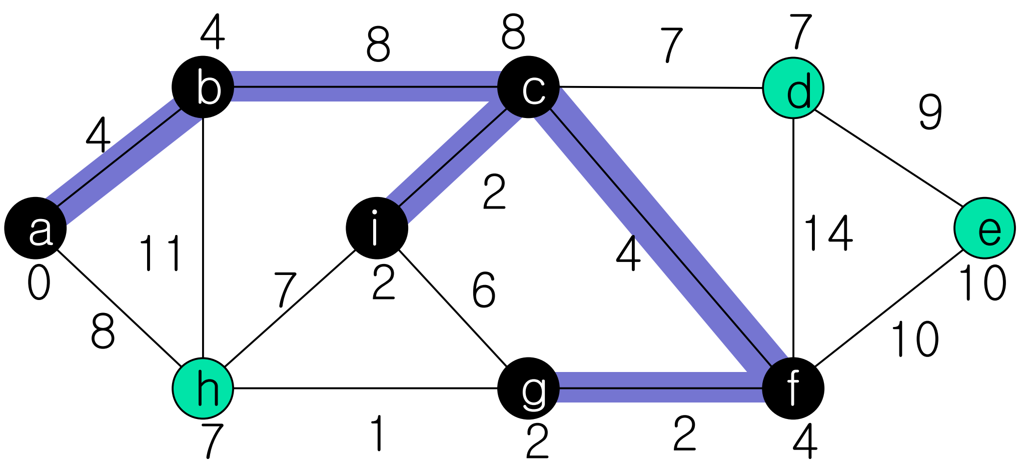
Prim's Algorithm




```
7.      u = Extract-Min(Q)
8.        for each v ∈ G.Adj[u]
9.          if v ∈ Q and w(u,v) < v.key
10.           v.π = u
11.           v.key = w(u,v)
```


Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

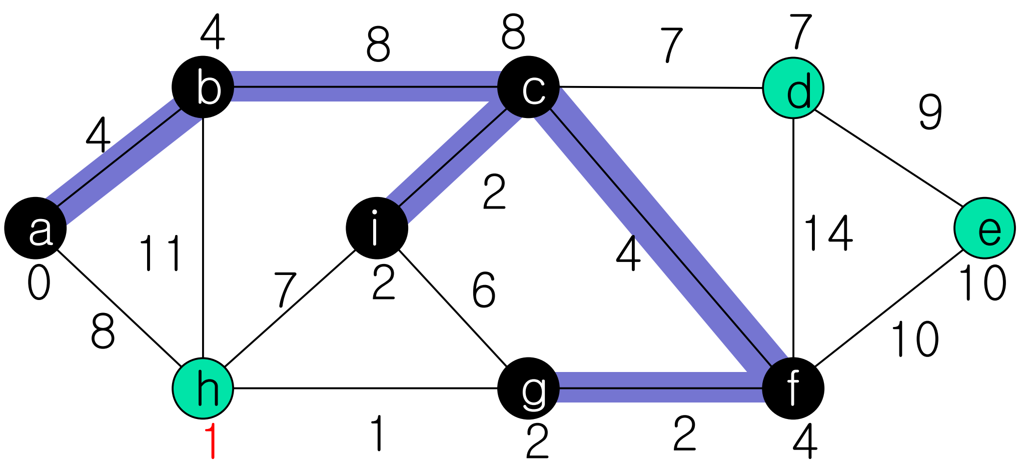
Prim's Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

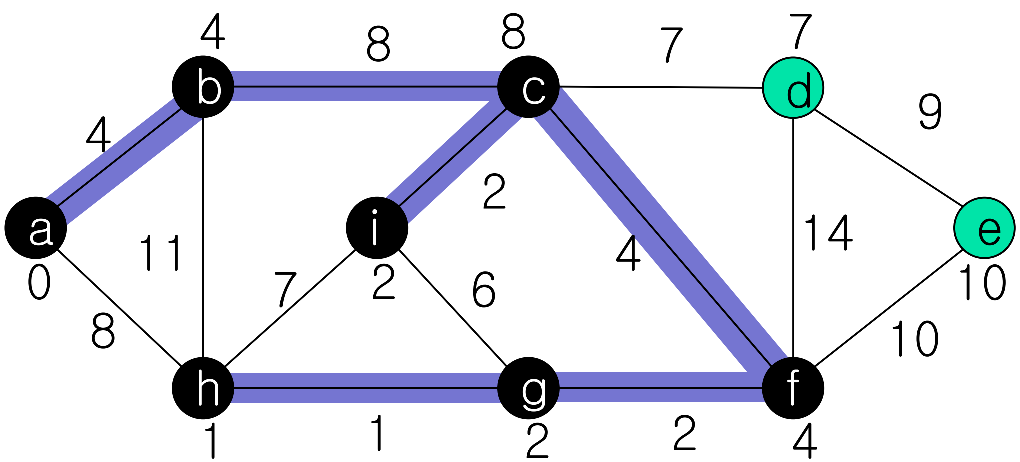
Prim’s Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

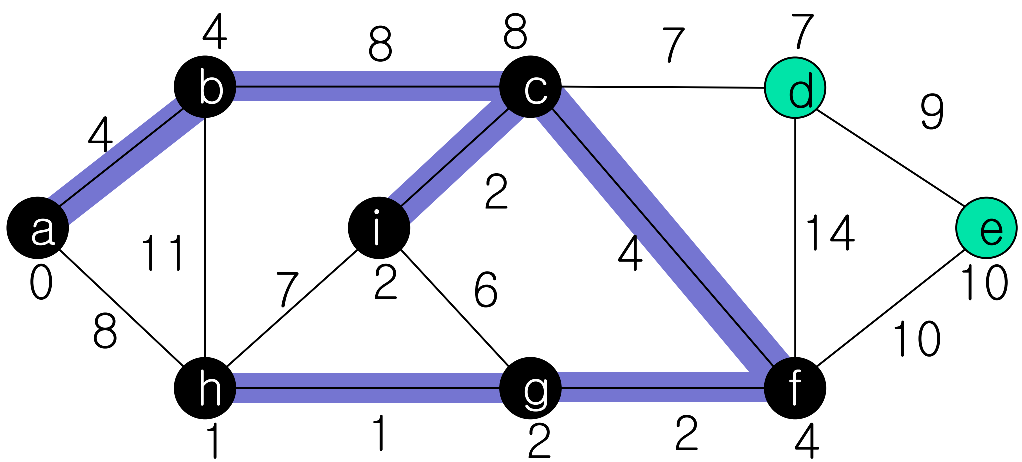
Prim’s Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

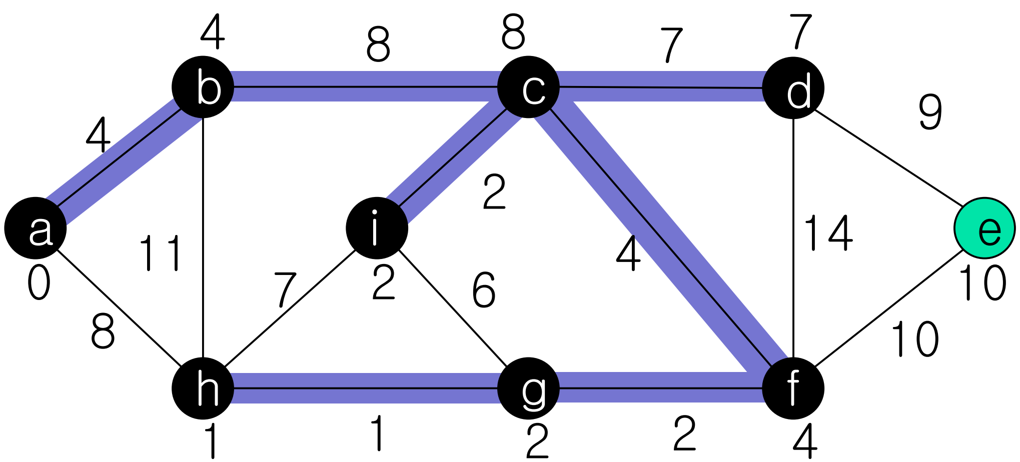
Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Prim's Algorithm




```
7.        u = Extract-Min(Q)
8.          for each v ∊ G.Adj[u]
9.              if v ∊ Q and w(u,v) < v.key
10.                 v.π = u
11.                 v.key = w(u,v)
```

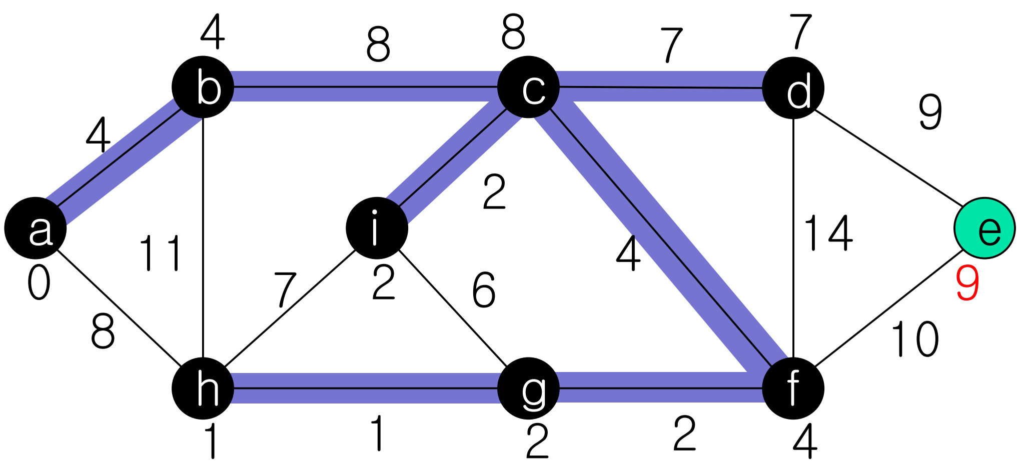
Prim's Algorithm



7. u = Extract-Min(Q)
8. for each v ∈ G.Adj[u]
9. if v ∈ Q and w(u,v) < v.key
10. v.π = u
11. v.key = w(u,v)

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

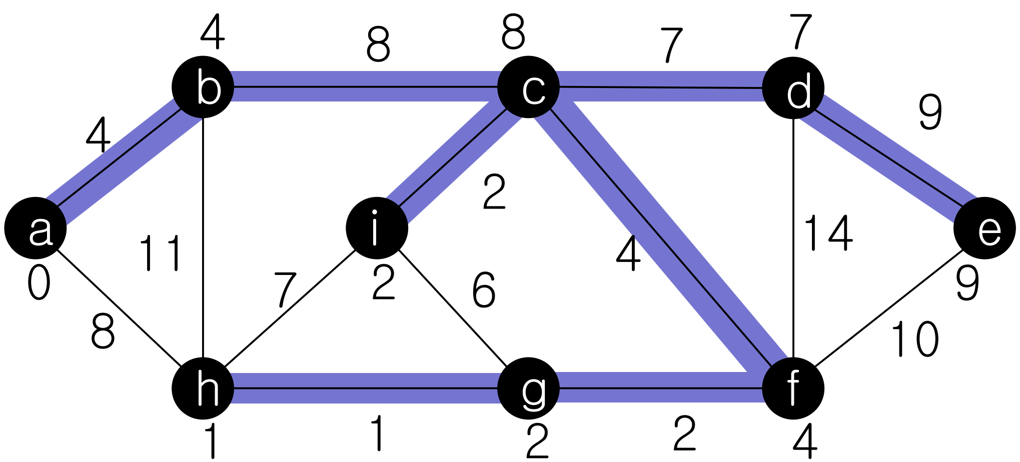
Prim's Algorithm



7. u = Extract-Min(Q)
8. for each v ∊ G.Adj[u]
9. if v ∊ Q and w(u,v) < v.key
10. v.π = u
11. v.key = w(u,v)

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

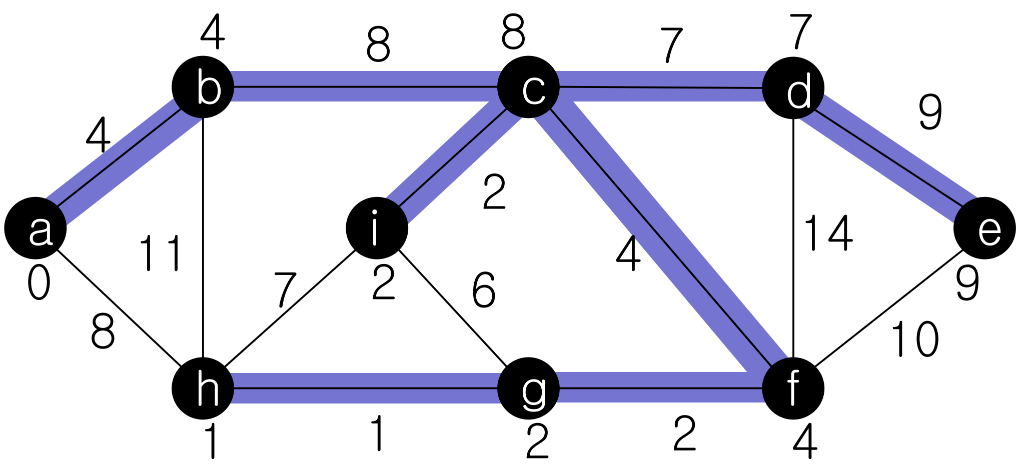
Prim's Algorithm




```
u = Extract-Min(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v)
```

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Prim's Algorithm



6. **while** Q ≠ ∅
7. u = Extract-Min(Q)

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Running Time of Prim's Algorithm

- The running time of Prim's algorithm depends on how we implement the min-priority queue Q.
- If we implement Q as a binary min-heap,
  - EXTRACT-MIN takes O(lg |V|) time.
  - DECREASE-KEY takes O(lg |V|) time.
- If we implement Q as a simple array,
  - EXTRACT-MIN takes O(|V|) time.
  - DECREASE-KEY O(1) time.
- If we implement Q as a Fibonacci heap,
  - EXTRACT-MIN takes O(lg V|) amortized time.
  - DECREASE-KEY O(1) amortized time.

Prim's Algorithm

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π=NIL
    r.key = 0
    Q = G.V
    while Q ≠ Ø
        u = Extract-Min(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u,v) < v.key
                v.π = u
                v.key = w(u,v)
```

$O(|V|)$

Q: Implement as a binary min-heap

$O(|V| \lg |V|)$

$O(|E| \lg |V|)$
DECREASE-KEY $O(E)$ times

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Prim’s Algorithm

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π=NIL
    r.key = 0
    Q = G.V
    while Q ≠ Ø
        u = Extract-Min(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u,v) < v.key
                v.π = u
                v.key = w(u,v)
```

$O(|V|)$

Q: Implement as an array

$O(|V|^2)$

$O(|E|)$

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Prim's Algorithm

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π=NIL
    r.key = 0
    Q = G.V
    while Q ≠ Ø
        u = Extract-Min(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u,v) < v.key
                v.π = u
                v.key = w(u,v)
```

O(|V|)

Q: Implement as a Fibonacci mean-heap

O(|V| lg |V|)

O(|E|)

Thomas T. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, MIT Press, Cambridge, MA, 2009

Minimum-Cost Spanning Trees

- Cost of a spanning tree
  - Sum of the costs (weights) of the edges in the spanning tree
- Min-cost spanning tree
  - A spanning tree of least cost
- Greedy method
  - At each stage, make the best decision possible at the time
    - Based on either a least cost or a highest profit criterion
  - Make sure the decision will result in a feasible solution
    - Satisfy the constraints of the problem
- To construct min-cost spanning trees
  - Best decision : least-cost
  - Constraints
    - Use only edges within the graph
    - Use exactly n-1 edges
    - May not use edges that produce a cycle

Kruskal's Algorithm

- Procedure
  - Build a min-cost spanning tree T by adding edges to T one at a time
  - Select edges for inclusion in T in nondecreasing order of their cost
  - Edge is added to T if it does not form a cycle

Kruskal's Algorithm (Cont.)

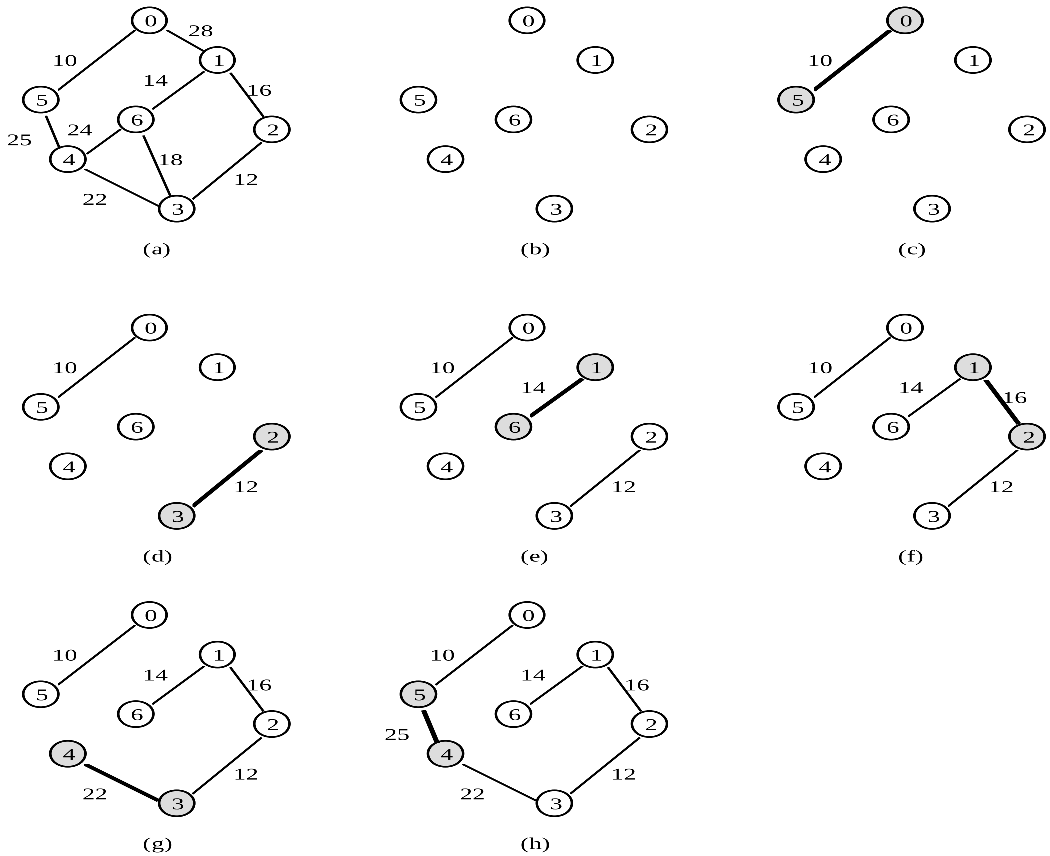


Figure 6.23 : Stages in Kruskal's algorithm

Kruskal's Algorithm (Cont.)

```
T = Φ;
while( (T contains less than n-1 edges) && (E not empty) ) {
      choose an edge (v,w) from E of the lowest cost;
      delete (v,w) from E;
      if( (v,w) does not create a cycle in T ) add (v,w) to T;
      else discard (v,w);
}
if (T contains fewer than n-1 edge) cout << "no spanning tree" << endl;
```

Program 6.6: Kruskal's algorithm

- Time Complexity
  - When we use a min heap to determine the lowest cost edge , O(eloge)

Prim's Algorithm

- Property
  - At all times during the algorithm the set of selected edges forms a tree
- Procedure
  - Begin with a tree T that contains a single vertex
  - Add a least-cost edge (u,v) to T such that $T \cup \{(u,v)\}$ is also a tree
  - Repeat until T contains n-1 edges

Prim's Algorithm (Cont.)

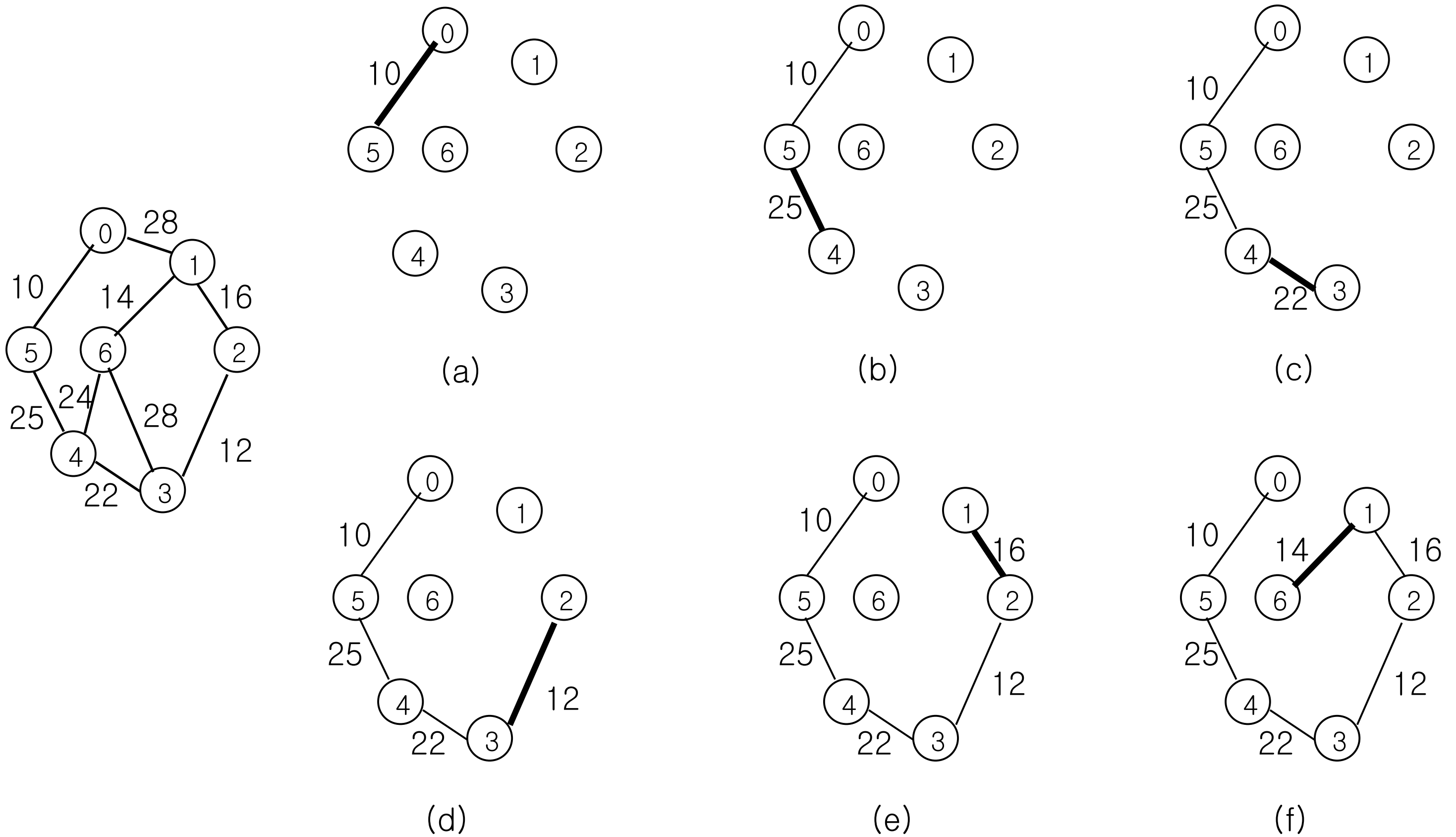


Figure 6.24: Stages in Prim's algorithm

Prim's Algorithm (Cont.)

```
// Assume that G has at least one vertex.
TV = { 0 };  // start with vertex 0 and no edges
for(T = Φ; T contains fewer than n-1 edges; add (u,v) to T)
{
     Let (u,v) be a least-cost edge such that u ∈ TV and v !∈ TV;
     if(there is no such edge) break;
     add v to TV;
}
if(T contains fewer than n-1 edges) cout << “no spanning tree” << endl;
```

Program 6.7: Prim's algorithm

Sollin's Algorithm

- Select several edges at each stage
- At the start of a stage, the selected edges, together with all n graph vertices, form a spanning forest at each stage
- During a stage, select one minimum cost edge for each tree in this forest
- The selected edges are added to the spanning tree being constructed

Sollin's Algorithm (Cont.)

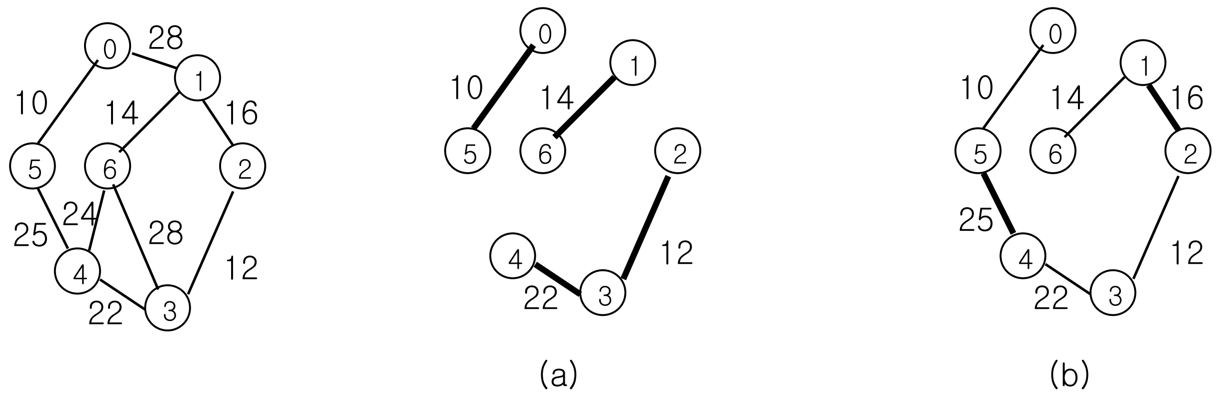


Figure 6.25: Stages in Sollin's algorithm

Single-source Shortest Paths

Single-source Shortest-Paths Problem

- Given a weighted directed graph G=(V,E) with weight function w:E→R mapping edges to real-valued weights, find the minimum-weight path from a given source vertex s to another vertex v.
  - The weight w(p) of a path p=<$v_0, v_1, \ldots, v_k$> is the sum of the weights of its constituent edges: $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$
  - The shortest-path weight δ(s,v) from s to v by

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

Optimal Substructure Property

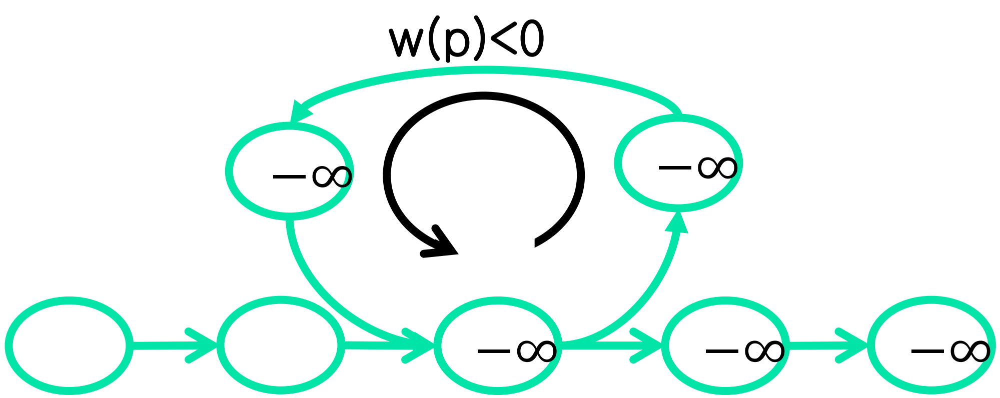
- A shortest path between two vertices contains other shortest paths within it.
- Lemma 24.1 (Subpaths of shortest paths are shortest paths)
  - Consider a weighted graph G, with weight function $w:E \rightarrow R$,
  - Let $p=<v_0,v_1,v_2,v_4,\ldots,v_{k-1},v_k>$ be a shortest path from vertex v_0 to vertex v_k.
  - For any i and j such that $1 \leq i \leq j \leq k$, let $p_{ij}=<v_i,v_{i+1},\ldots,v_j>$ be the subpath of p from vertex v_i to vertex v_j.
  - Then, p_{ij} is a shortest path from v_i to v_j.
- Proof is straightforward.

Shortest Path Properties

- Some instances of the single-source shortest-paths problem may include edges whose weights are negative.
- If the graph $G=(V,E)$ contains no negative weight cycles reachable from the source s, then for all $v \in V$, the shortest-path weight $\delta(s,v)$ remains well defined, even if it has a negative value.
- If the graph contains a negative-weight cycle reachable from s, however, shortest-path weights are not well defined.

Shortest Path Properties

- No path from s to a vertex on the cycle can be a shortest path—we can always find a path with lower weight by following the proposed shortest path and then traversing the negative-weight cycle.
- If there is a negative-weight cycle on some path from s to v, we define $\delta(s,v) = -\infty$.

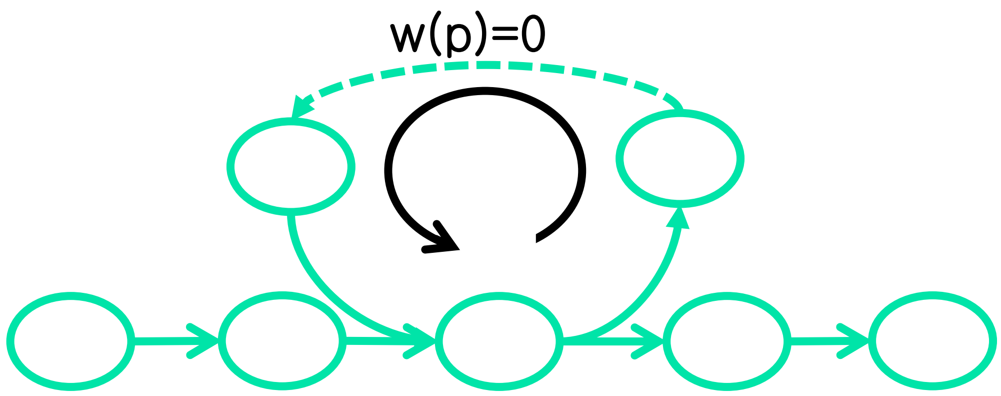


Can a Shortest Path Contain a Cycle?

- As we have just seen, it cannot contain a negative-weight cycle.
- Nor can it contain a positive-weight cycle, since removing the cycle from the path produces a path with the same source and destination vertices and a lower path weight.
- We can remove a 0-weight cycle from any path to produce another path whose weight is the same.
- Thus, if there is a shortest path from a source vertex s to a destination vertex that contains a 0-weight cycle, then there is another shortest path from s to without this cycle.

Shortest Path Properties

- Without loss of generality, we can assume that when we are finding shortest paths, they have no cycles.
- Since any acyclic path in a graph G=(V,E), contains at most |V| distinct vertices, it also contains at most |V|-1 edges.
- If have only 0-weight cycles, we can restrict our attention to shortest paths of at most |V|-1 edges.



Representing Shortest Paths

- We represent shortest paths similarly to how we represented breadth-first trees.
- Shortest-paths tree:
  - Predecessor subgraph $G_\pi = (V_\pi, E_\pi)$
    - $V_\pi=\{v\in V : v.\pi \neq NIL \}\cup\{s\}$
    - $E_\pi=\{(v.\pi,v) \in E : v \in V_\pi - \{s\}$
  - V_π is the set of vertices of G with non-NIL predecessors, plus the source s.
  - E_π is the the set of edges induced by the π values for vertices in V_π.
  - G_π forms a rooted tree with root s containing a shortest path from the source s to every vertex that is reachable from s.

A Shortest-path Tree

- Let G=(V, E) be a weighted, directed graph with weight function $w: E \rightarrow R$.
- Assume that G contains no negative-weight cycles reachable from the source vertex $s \in V$, so that shortest paths are well defined.
- A shortest-path tree rooted at s is a directed subgraph G'=(V', E'), where $V' \subset V$ and $E' \subset E$, such that
  - V' is the set of vertices reachable from s in G.
  - G' forms a rooted tree with root s.
  - 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.

Relaxation

- The algorithms in this chapter use the technique of relaxation.
- For all $v \in V$, we maintain an attribute $v.d$ which is an upper bound on the weight of a shortest path from source s to v.
- We call $v.d$ a shortest-path estimate.

INITIALIZE-SINGLE-SOURCE(G,s)

1. **for** each vertex $v \in G.V$
2. $\quad v.d = \infty$
3. $\quad v.\pi = \text{NIL}$
4. $s.d = 0$

Relaxation

- The process of relaxing an edge (u,v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating v.d and $v.\pi$.
- The following code performs a relaxation step on edge (u,v) in O(1) time.

RELAX(u, v, w)

1. **if** v.d>u.d + w(u,v)
2. v.d = u.d + w(u,v)
3. $v.\pi$ = u

Relaxation

- Each algorithm in this chapter calls INITIALIZE-SINGLE-SOURCE and then repeatedly relaxes edges.
- Moreover, relaxation is the only means by which shortest path estimates and predecessors change.
- The algorithms in this chapter differ in how many times they relax each edge and the order in which they relax edges.
  - The Bellman-Ford algorithm relaxes each edge |V|-1 times.
  - Dijkstra’s algorithm for directed acyclic graphs relaxes each edge exactly once.

Initialize

INITIALIZE-SINGLE-SOURCE(G,s)

1. **for** each vertex v $\in$ G.V
2. v.d = ∞
3. v.π = NIL
4. s.d = 0

Relaxation

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