CS473-Algorithms I

Lecture ?

The Algorithms of Kruskal and Prim

The Algorithms of Kruskal and Prim

Both algorithms use a specific rule to:

Determine a safe-edge in the Generic MST algoritm.

In Kruskal's algorithm, the set A is a forest

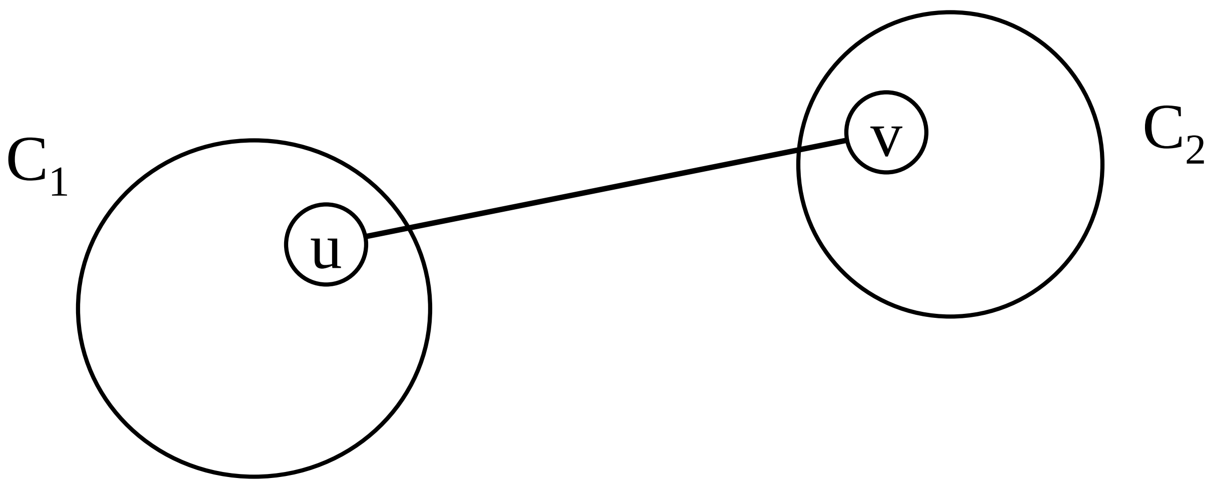
The Safe-Edge is always a Least-Weight edge in the graph that connects two distinct components (trees).

In Prim's algorithm, the set A forms a single tree

The Safe-Edge is always a Least-Weight edge in the graph that connects the tree to a vertex not in tree.

Kruskal's Algorithm

- Kruskal's algorithm is based directly on the Generic-MST
- It finds a Safe-Edge to add to the growing forest, by finding an edge (u,v) of Least-Weight of all edges that connect any two trees in the forest
- Let C_1 & C_2 denote two trees that are connected by (u,v)



Kruskal's Algorithm

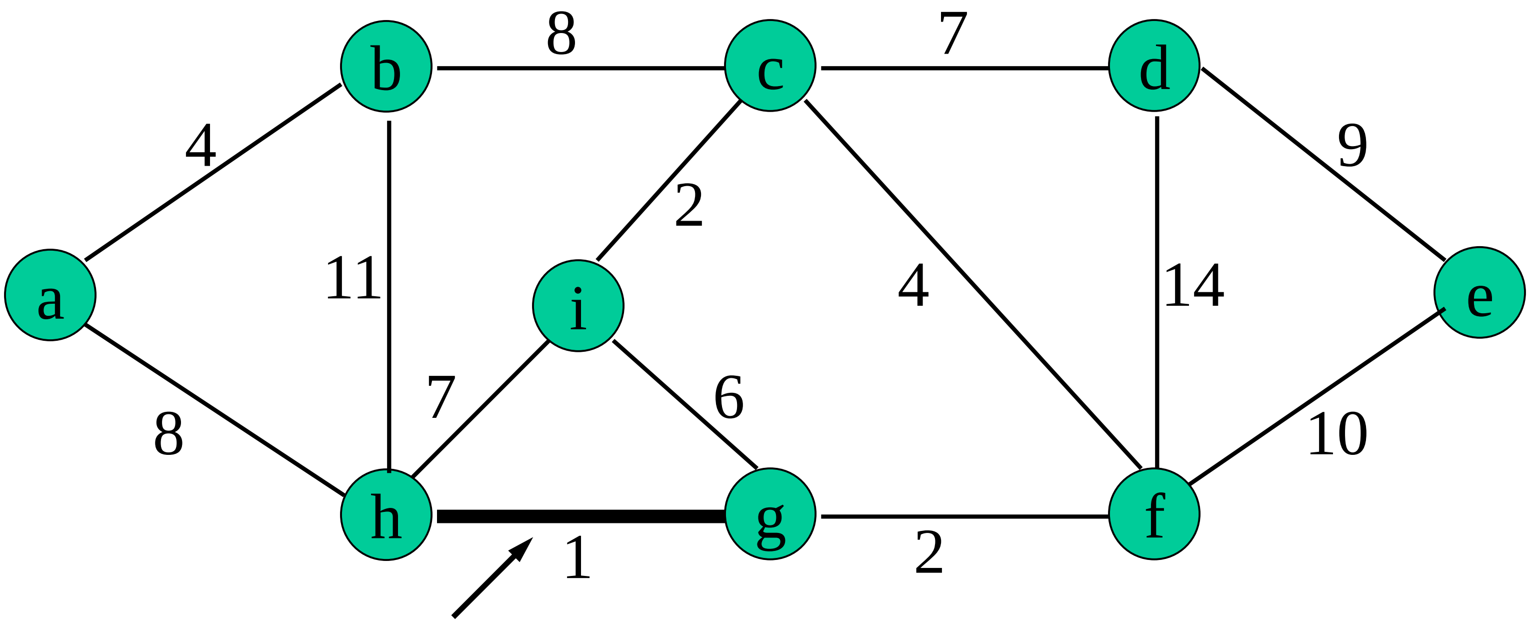
- Since (*u*,*v*) must be a light-edge connecting C_1 to some other tree, the Corollary implies that (*u*,*v*) is a Safe-Edge for C_1.

- Kruskal's algorithm is a greedy algorithm

  Because at each step it adds to the forest an edge of least possible weight.

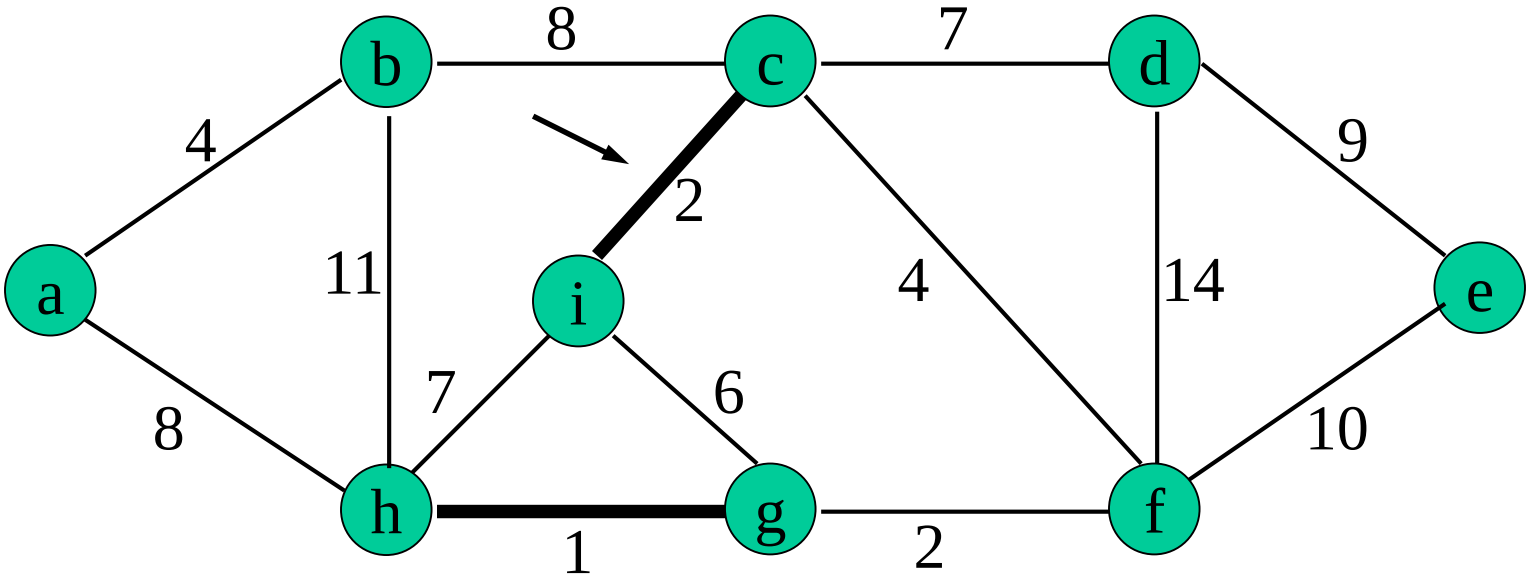
The Execution of Kruskal's Algorithm

(a)

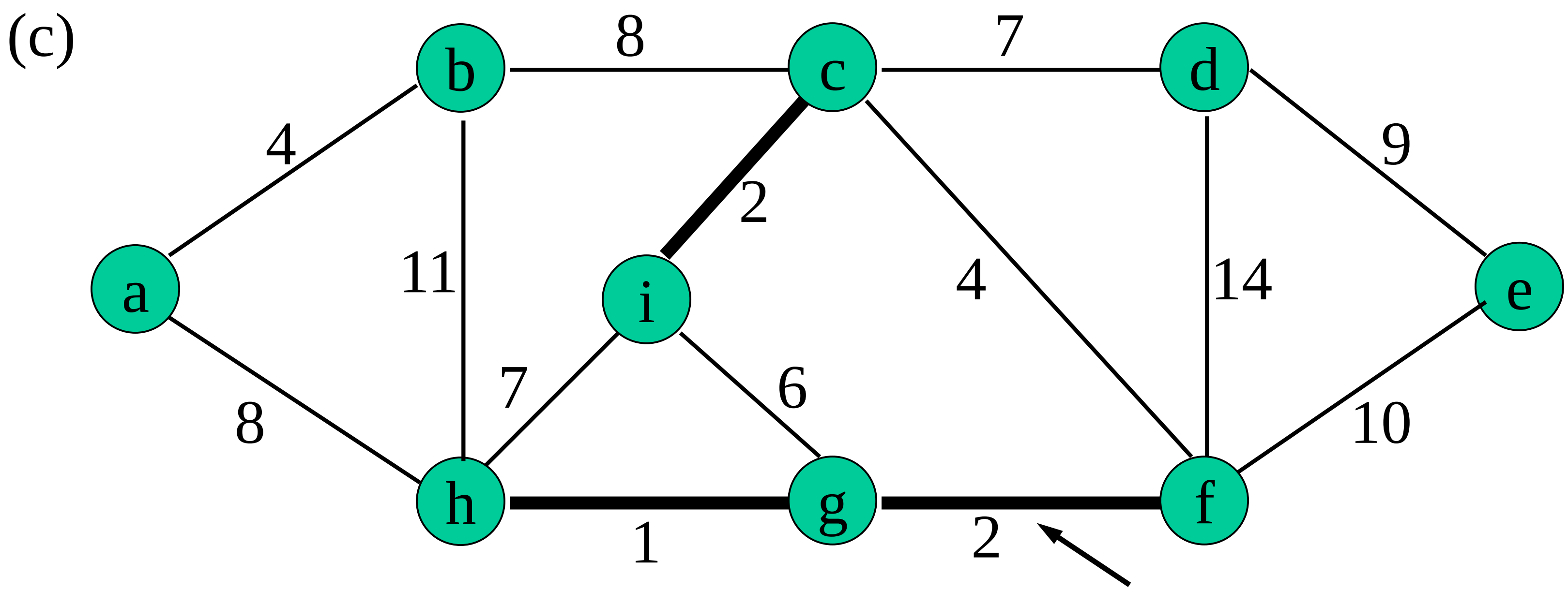


The Execution of Kruskal's Algorithm

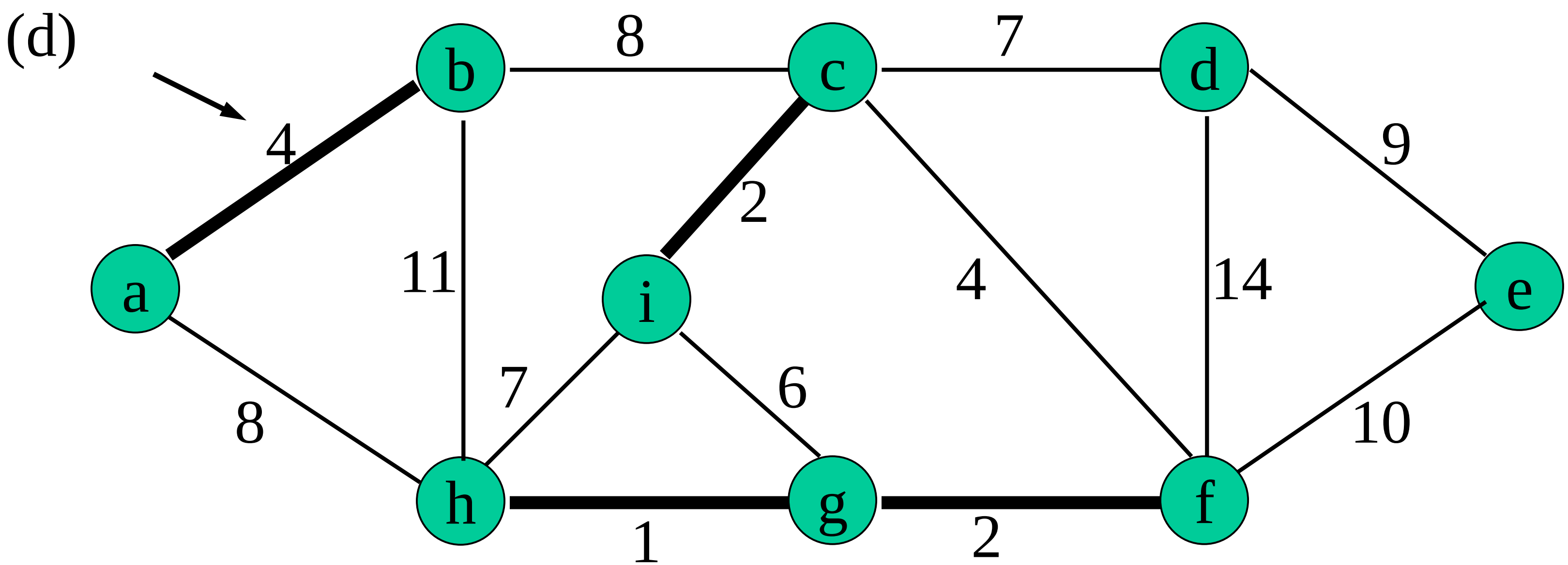
(b)



The Execution of Kruskal's Algorithm

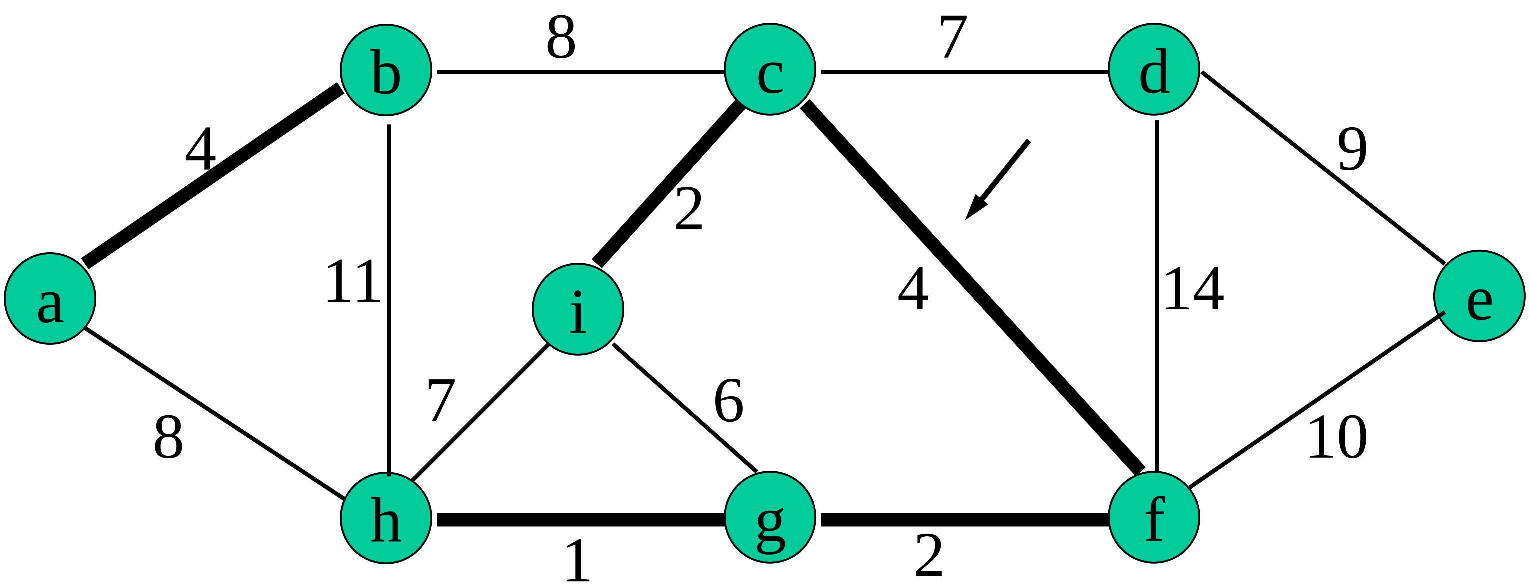


The Execution of Kruskal's Algorithm



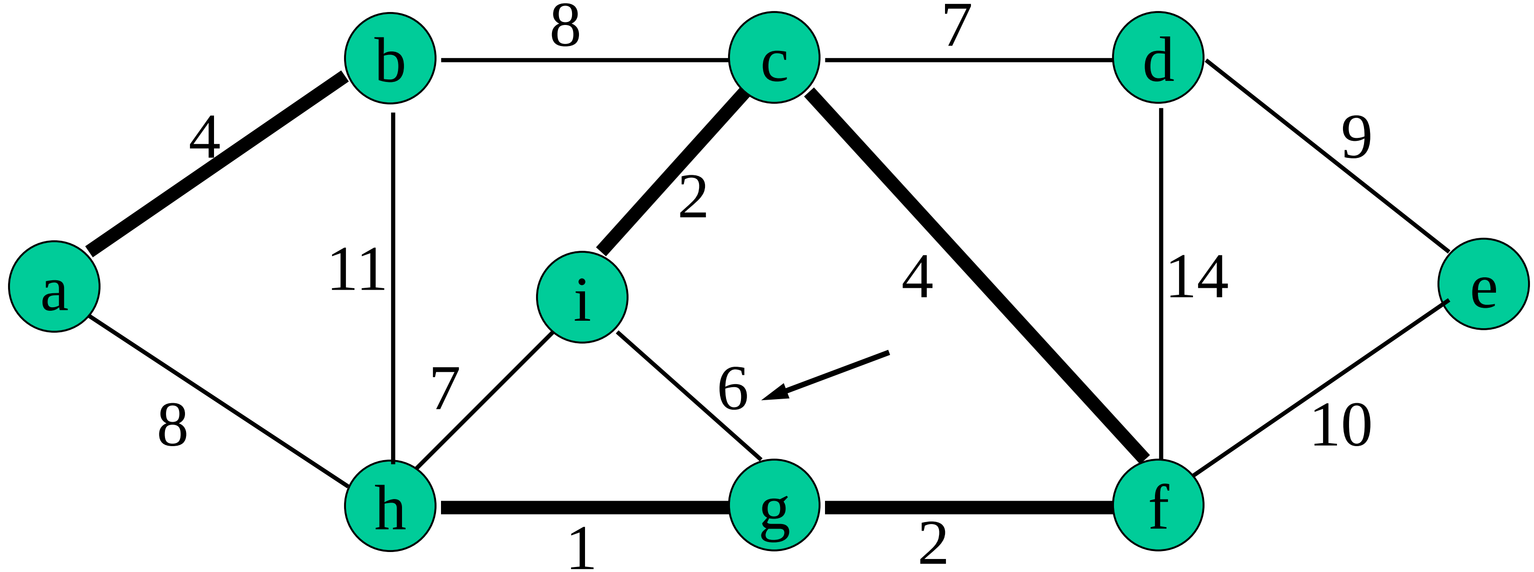
The Execution of Kruskal's Algorithm

(e)



The Execution of Kruskal's Algorithm

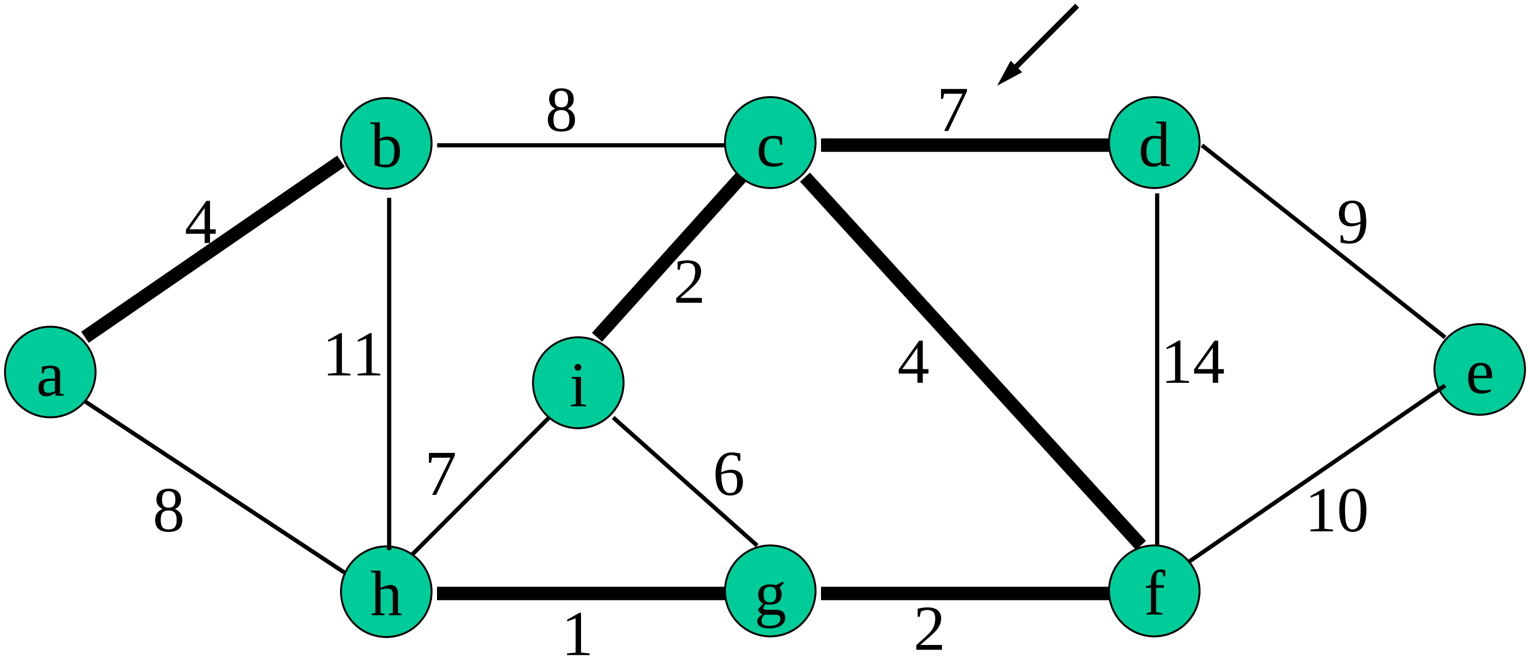
(f)



(g,i) discarded

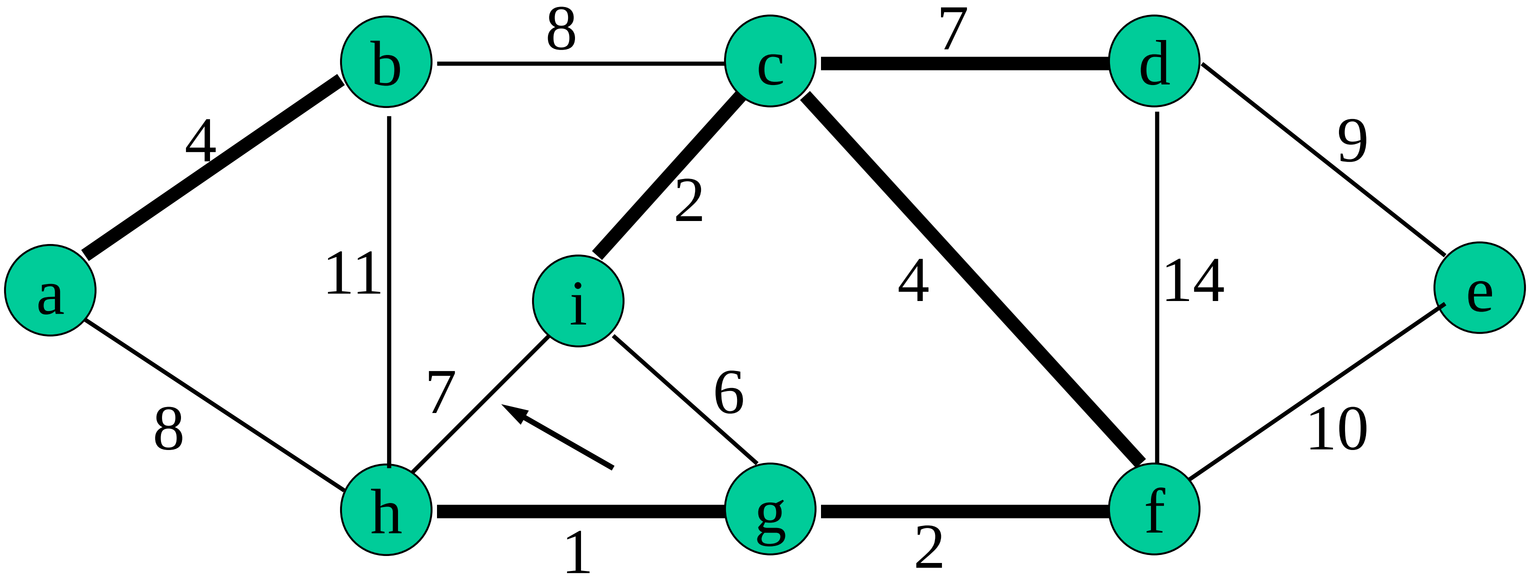
The Execution of Kruskal's Algorithm

(g)



The Execution of Kruskal's Algorithm

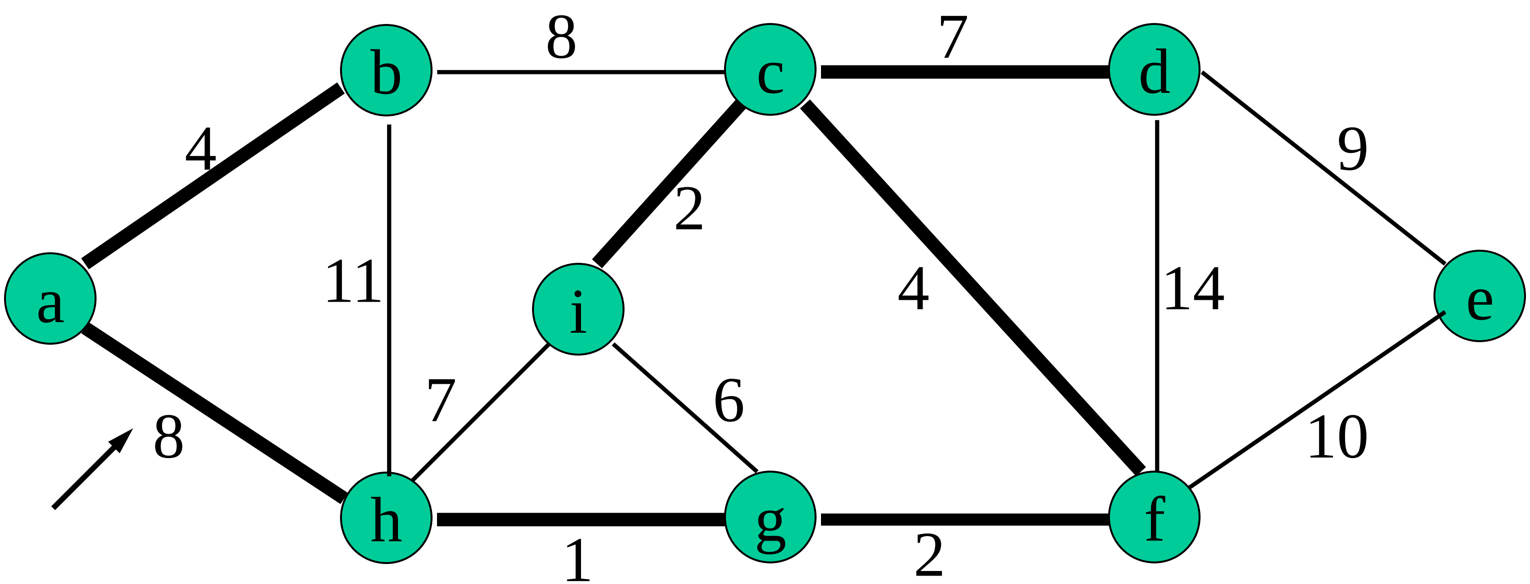
(h)



(h,i) discarded

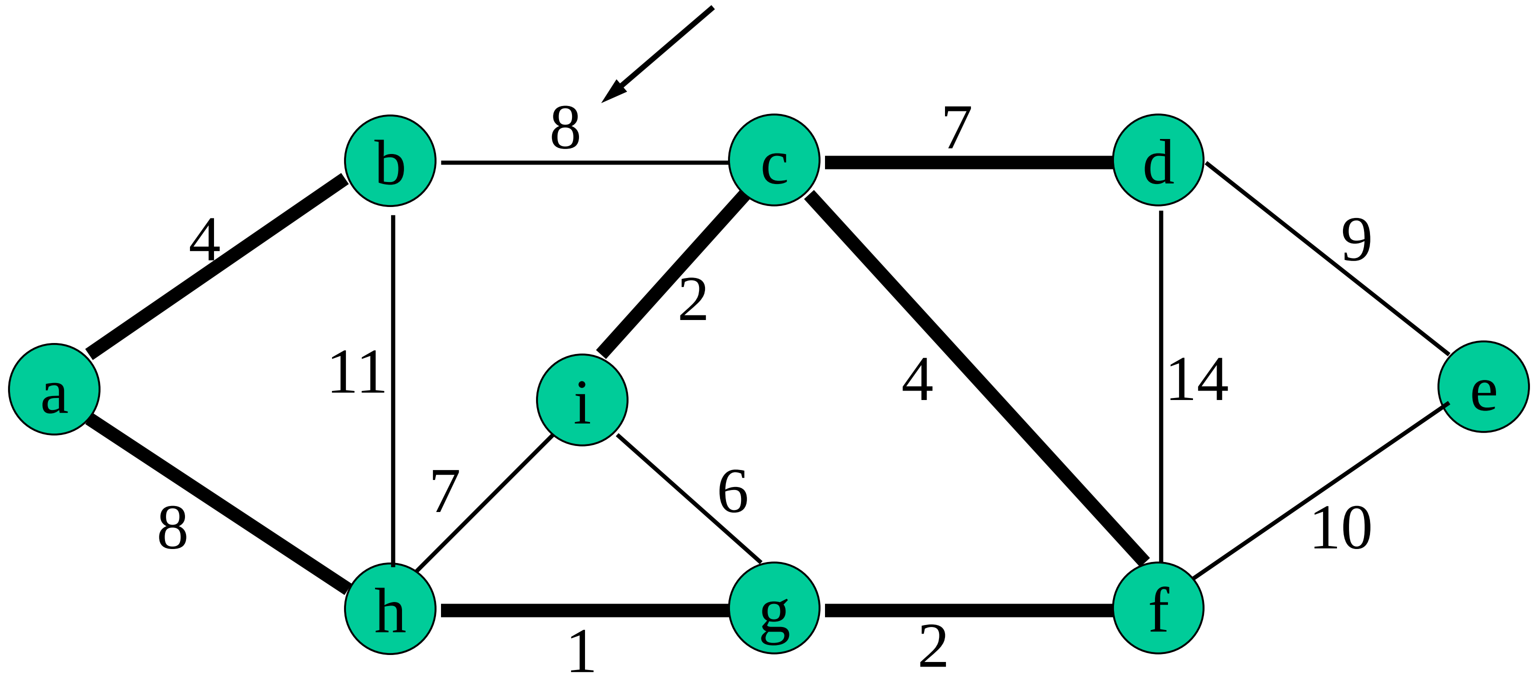
The Execution of Kruskal’s Algorithm

(i)



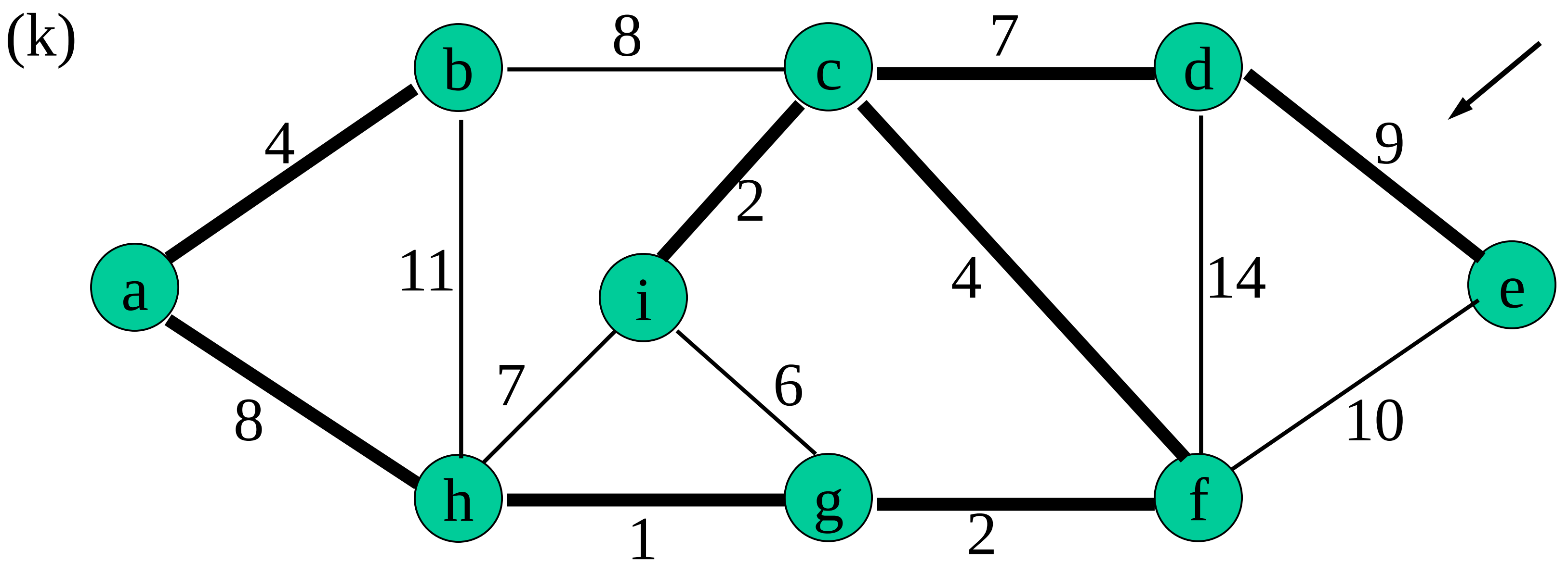
The Execution of Kruskal's Algorithm

(j)



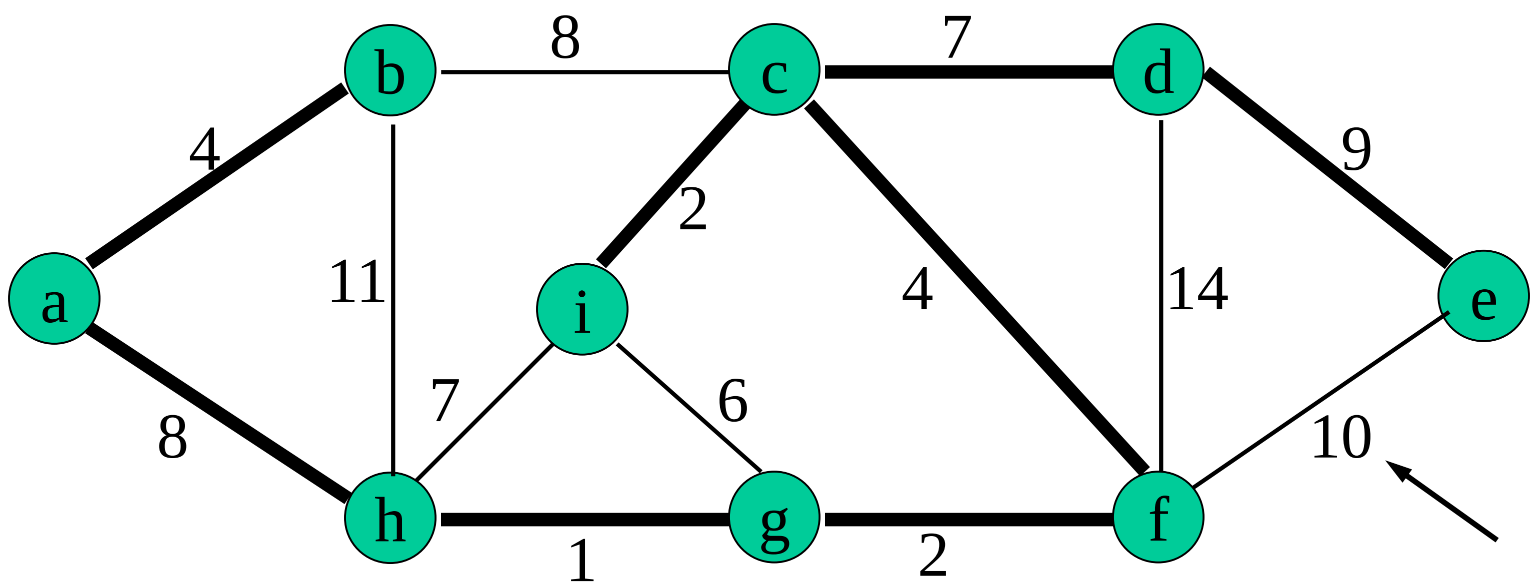
(b,c) discarded

The Execution of Kruskal's Algorithm



The Execution of Kruskal's Algorithm

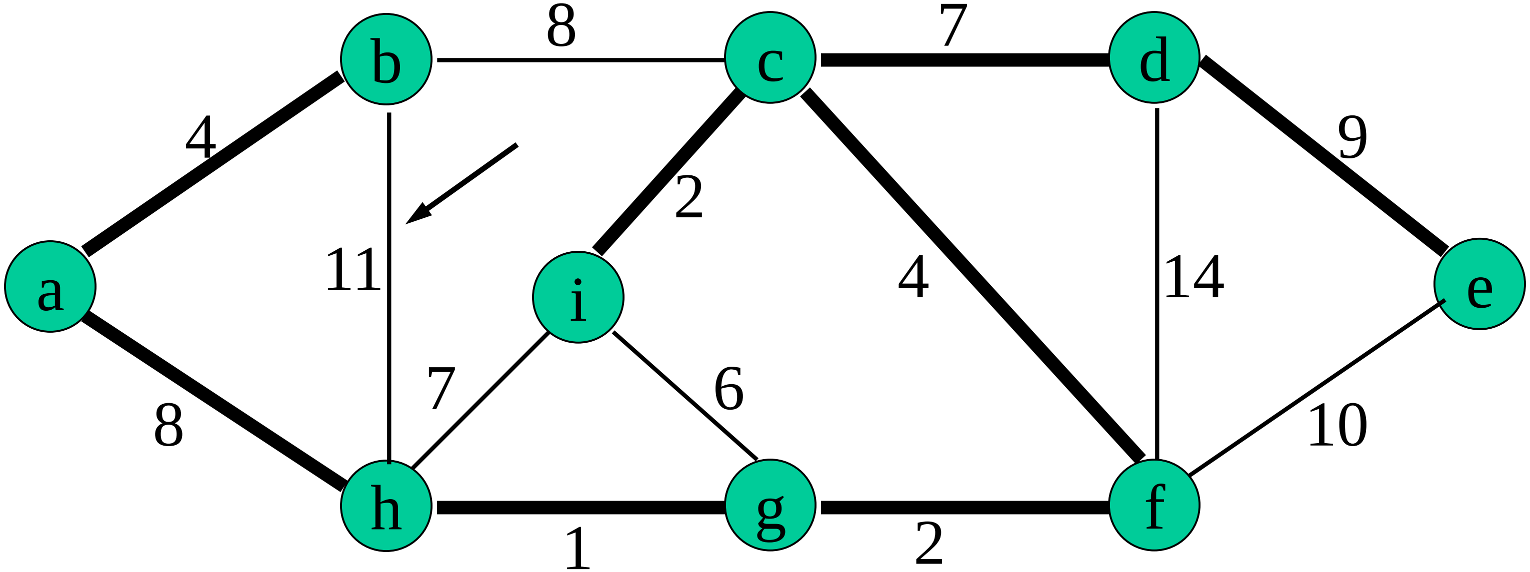
(l)



(e,f) discarded

The Execution of Kruskal's Algorithm

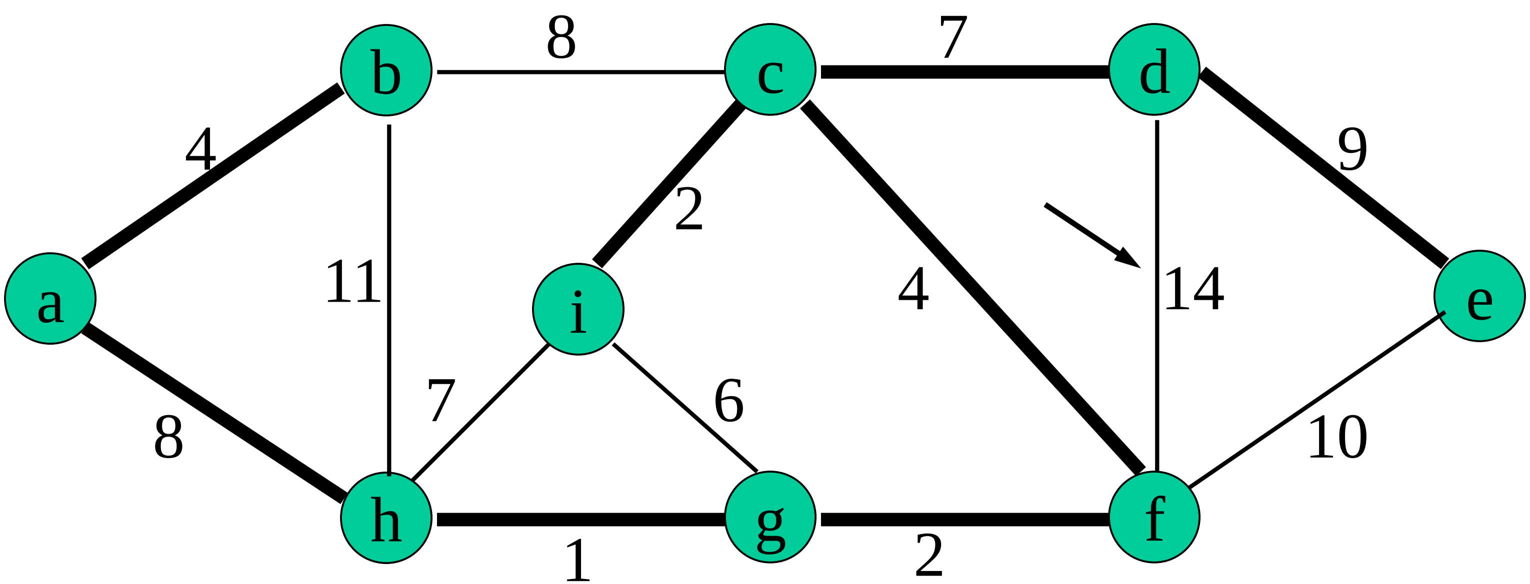
(m)



(b,h) discarded

The Execution of Kruskal's Algorithm

(n)



(d,f) discarded

Kruskal's Algorithm

- Our implementation of Kruskal's Algorithm uses a Disjoint-Set Data Structure to maintain several disjoint set of elements
- Each set contains the vertices of a tree of the current forest

Kruskal's Algorithm

MST-KRUSKAL (G, ω)

    $A \leftarrow \emptyset$

    for each vertex $v \in V[G]$ do

        MAKE-SET (v)

    SORT the edges of E by nondecreasing weight ω

    for each edge $(u,v) \in E$ in nondecreasing order do

        if FIND-SET(u) $\neq$ FIND-SET(v) then

            $A \leftarrow A \cup \{(u,v)\}$

            UNION (u,v)

    return A

end

Kruskal's Algorithm

- The comparison FIND-SET(u) $\neq$ FIND-SET(v) checks whether the endpoints u & v belong to the same tree
- If they do, then the edge (u,v) cannot be added to the tree without creating a cycle, and the edge is discarded
- Otherwise, the two vertices belong to different trees, and the edge is added to A

Running Time of Kruskal's Algorithm

- The running time for a graph G= (V, E) depends on the implementation of the disjoint-set data structure.
- Use the Disjoint-Set-Forest implementation with the Union-By-Rank and Path-Compression heuristics.
- Since it is the asymptotically fastest implementation known

Initialization (first for-loop) takes time O (V)
Sorting takes time O (E lg E) time

Running Time of Kruskal's Algorithm

- There are O (E) operations on the disjoint-set forest which in total take $O (E \alpha (E, V))$ time where α is the Functional Inverse of Ackerman's Function

- Since $\alpha (E, V) = O (\lg E)$

The total running time is $O (E \lg E)$.

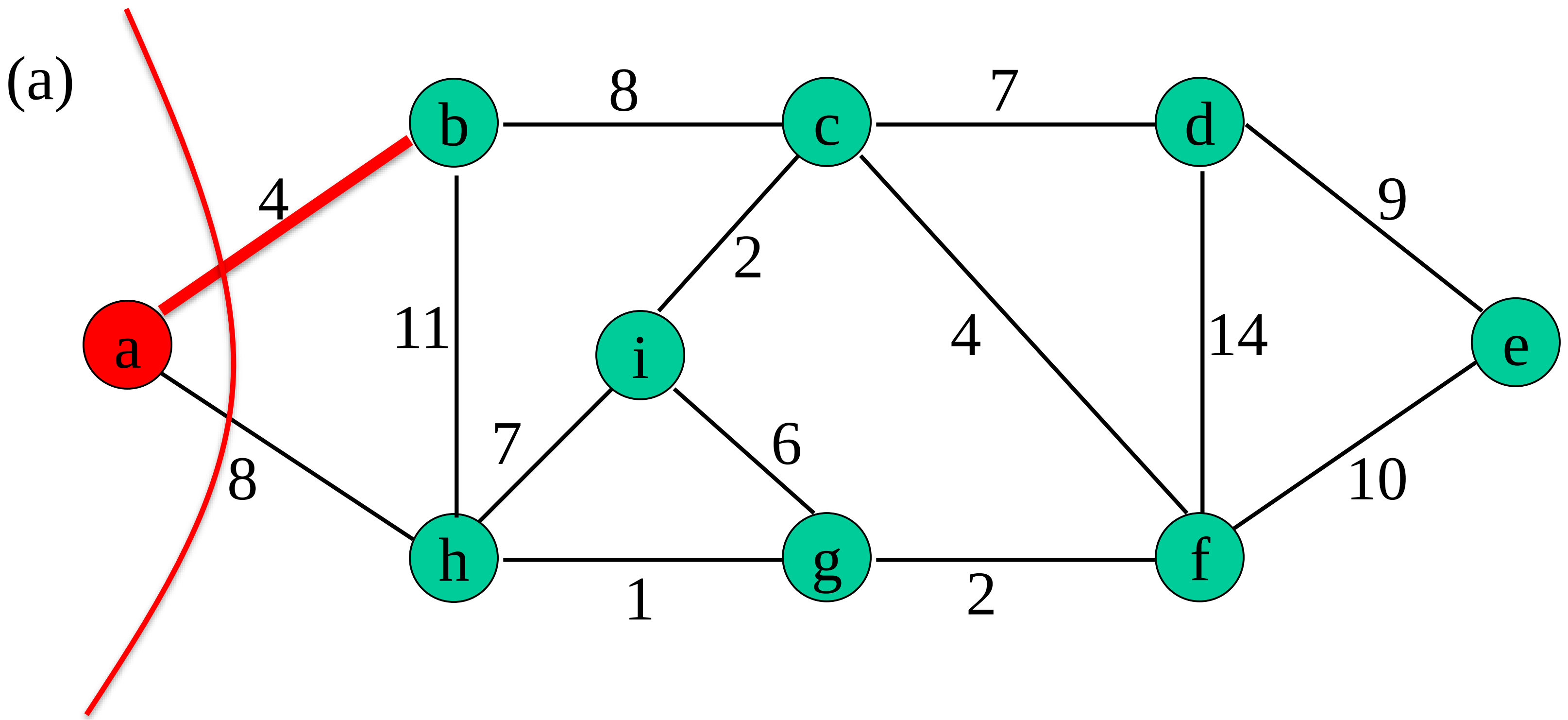
Prim's Algorithm

- Prim's algorithm is also a special case of Generic-MST algorithm
- The edges in the set A always form a single tree
- The tree starts from an arbitrary vertex v and grows until the tree spans all the vertices in V
- At each step, a light-edge connecting a vertex in A to a vertex in V - A is added to the tree A
- Hence, the Corollary implies that Prim's algorithm adds safe-edges to A at each step.

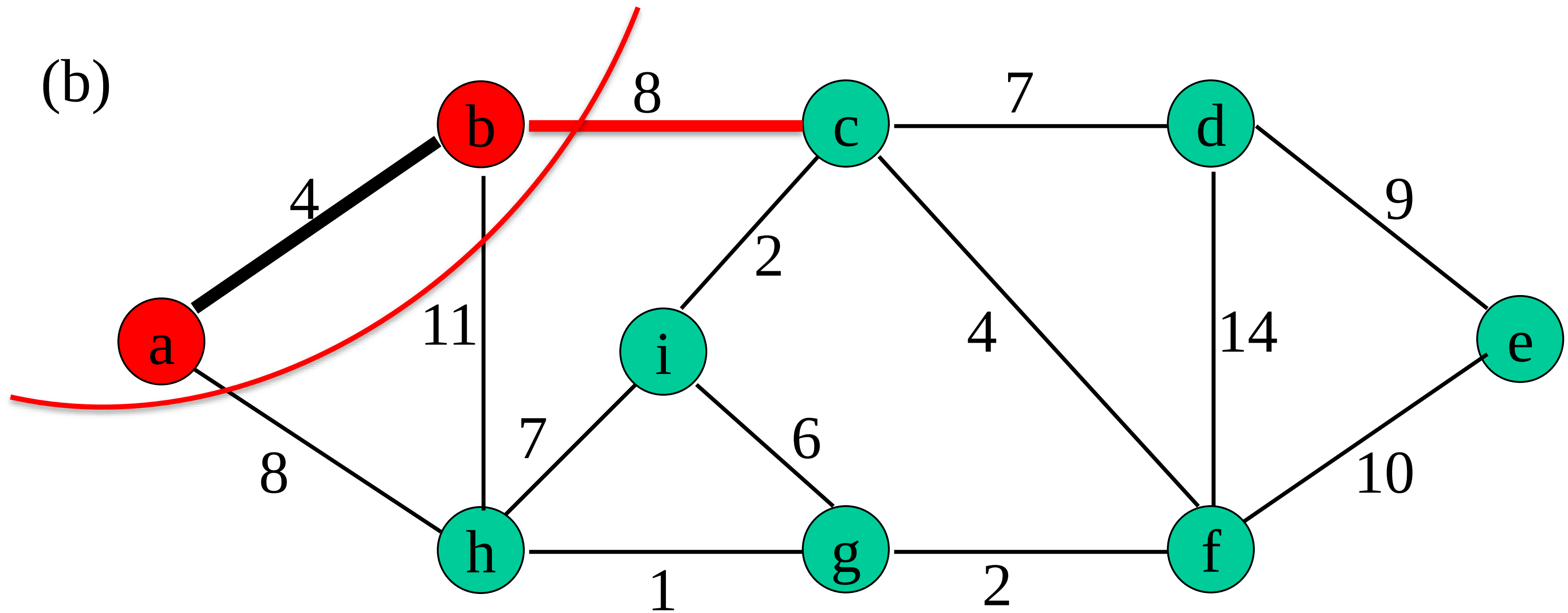
Prim’s Algorithm

- *This strategy is greedy*
- The tree is augmented at each step with an edge that contributes the minimum amount possible to the tree’s weight.

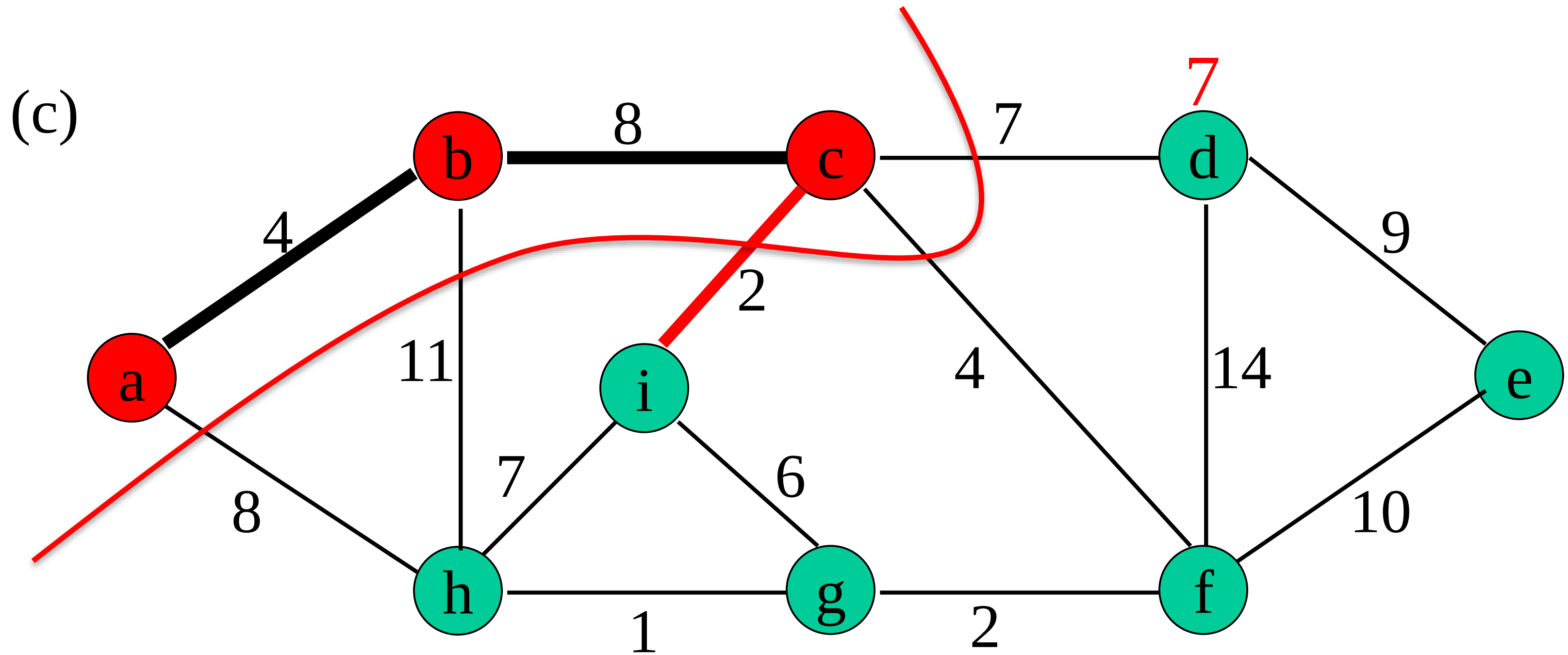
The Execution of Prim's Algorithm



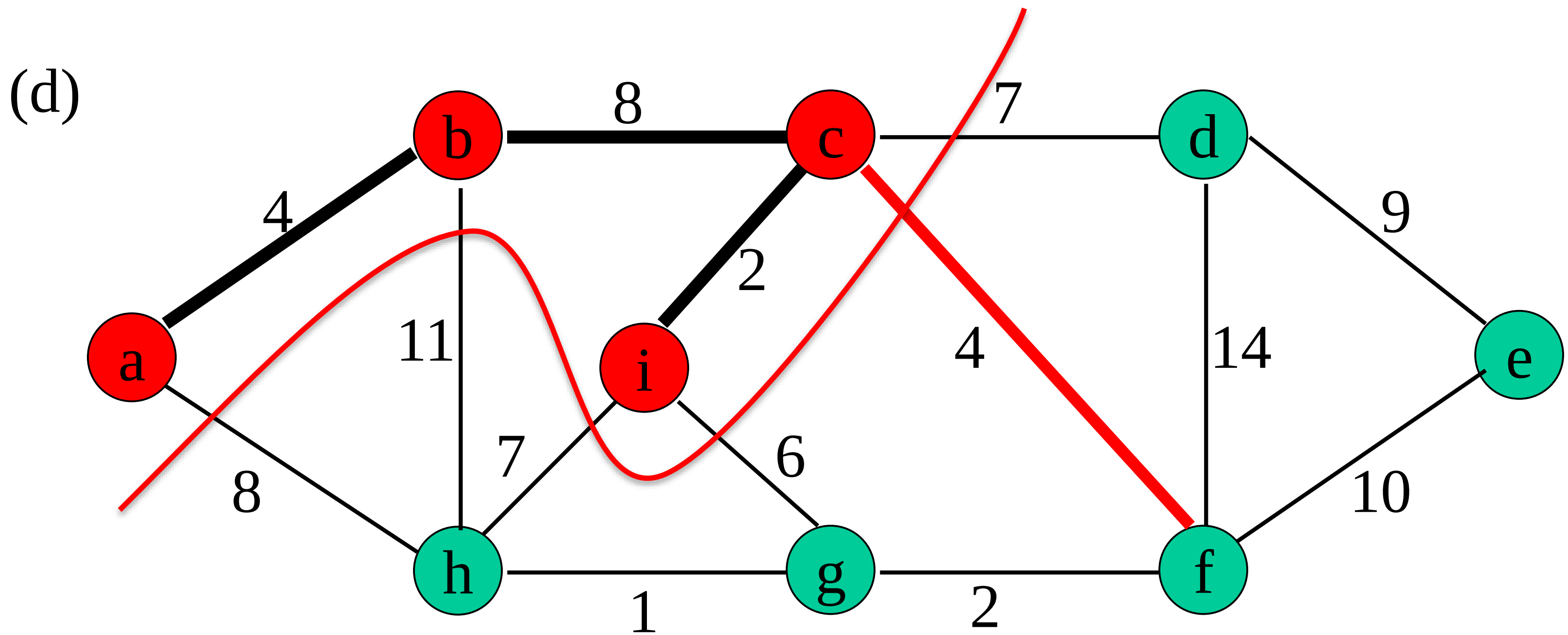
The Execution of Prim's Algorithm



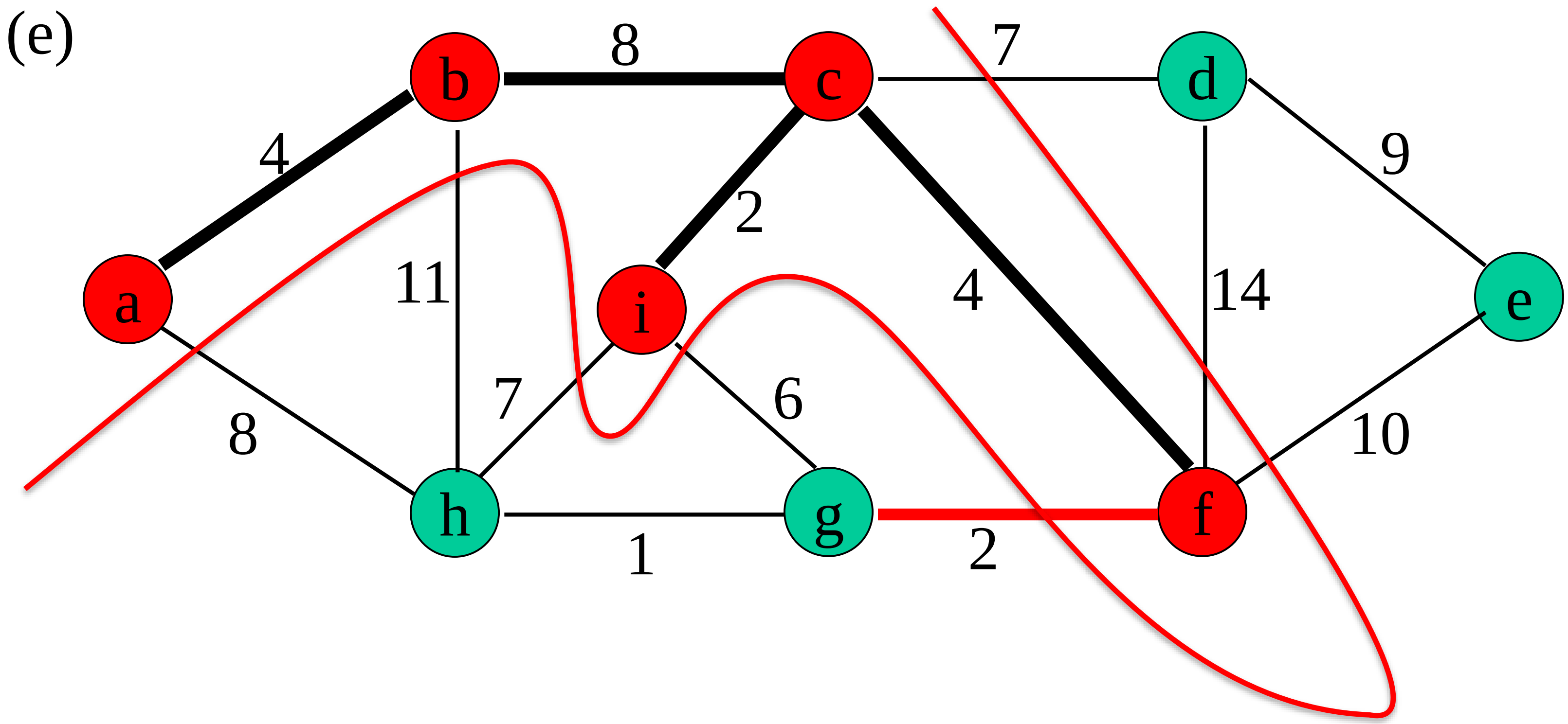
The Execution of Prim's Algorithm



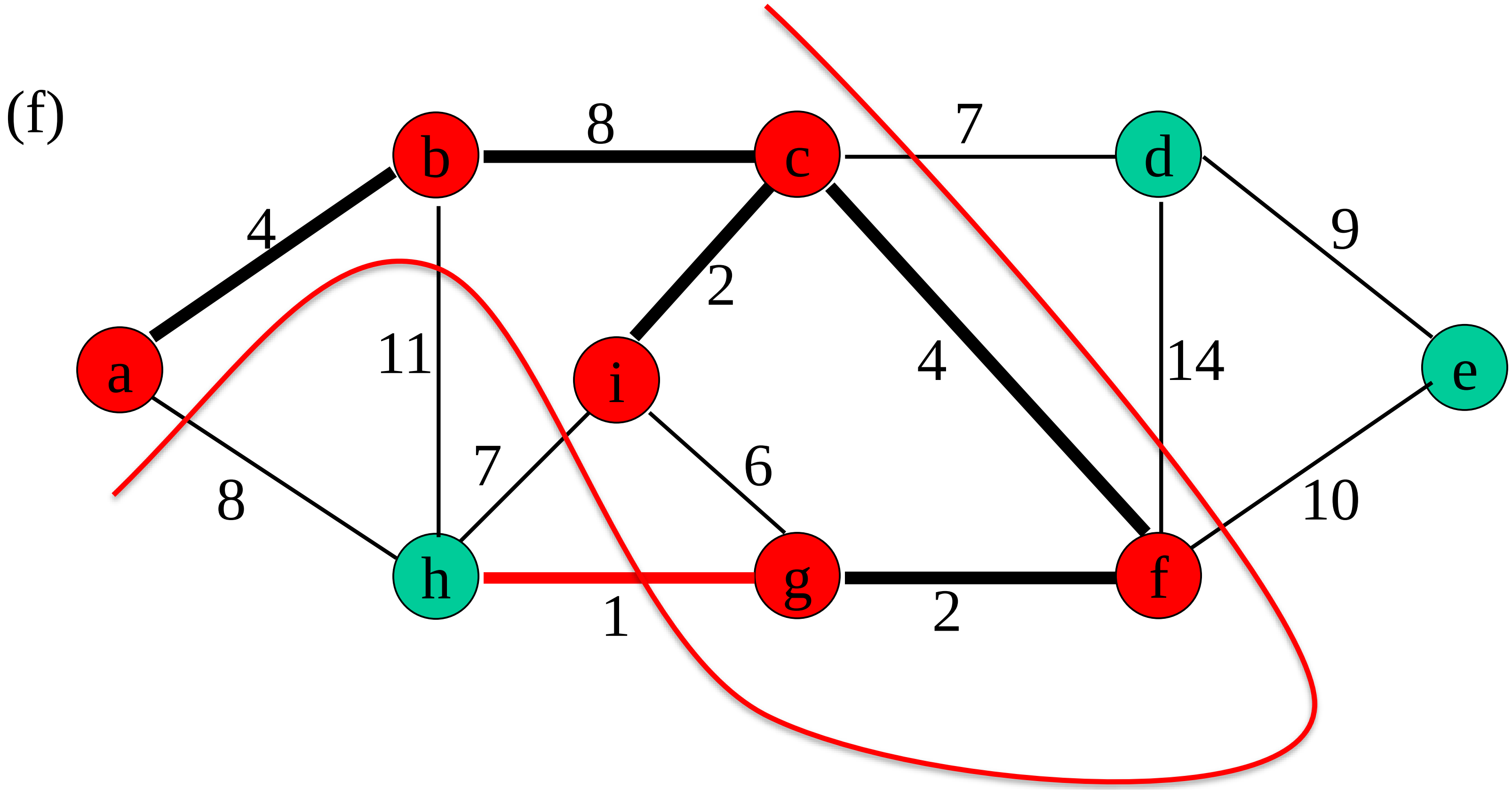
The Execution of Prim's Algorithm



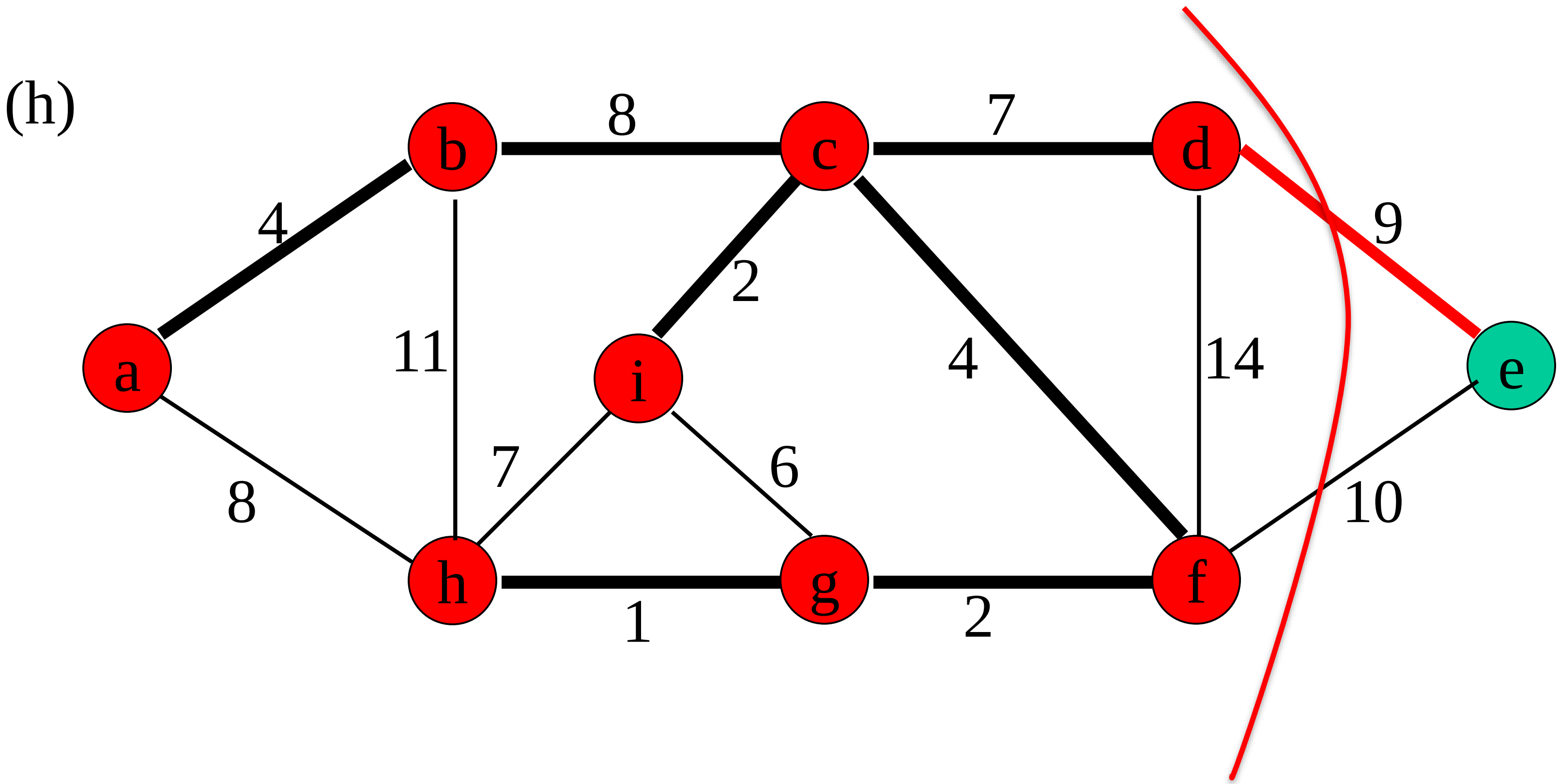
The Execution of Prim’s Algorithm



The Execution of Prim's Algorithm

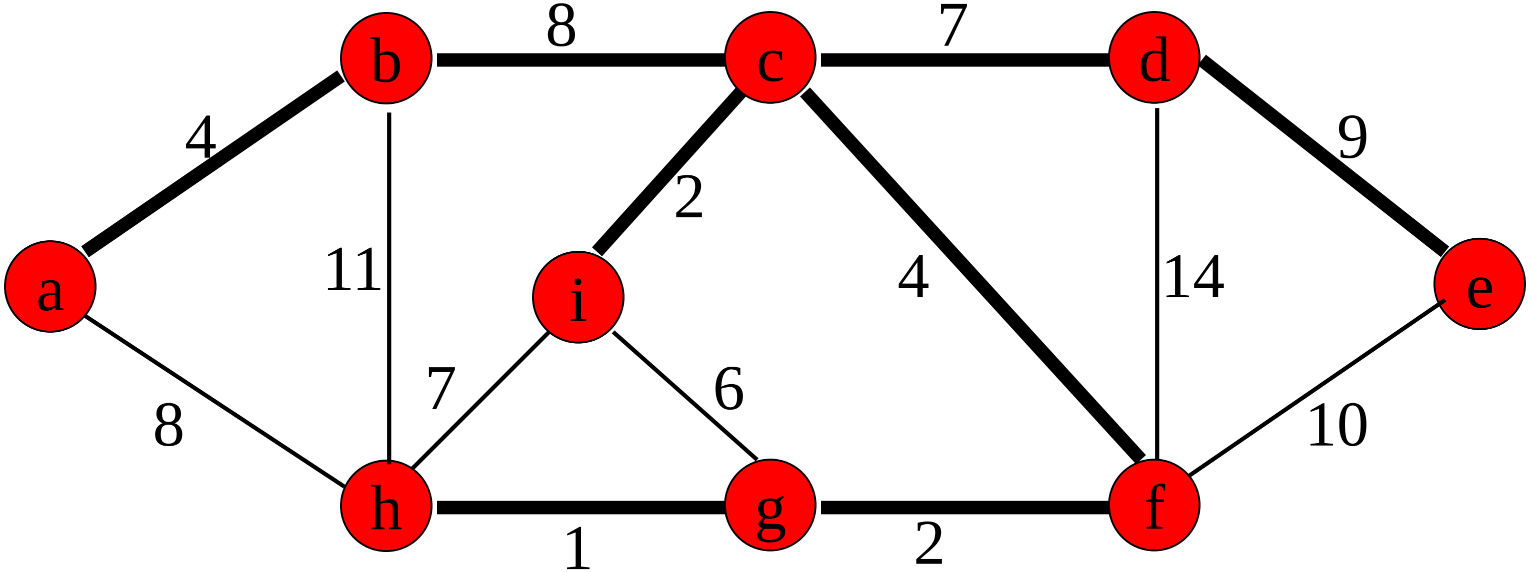


The Execution of Prim's Algorithm



The Execution of Prim's Algorithm

(i)



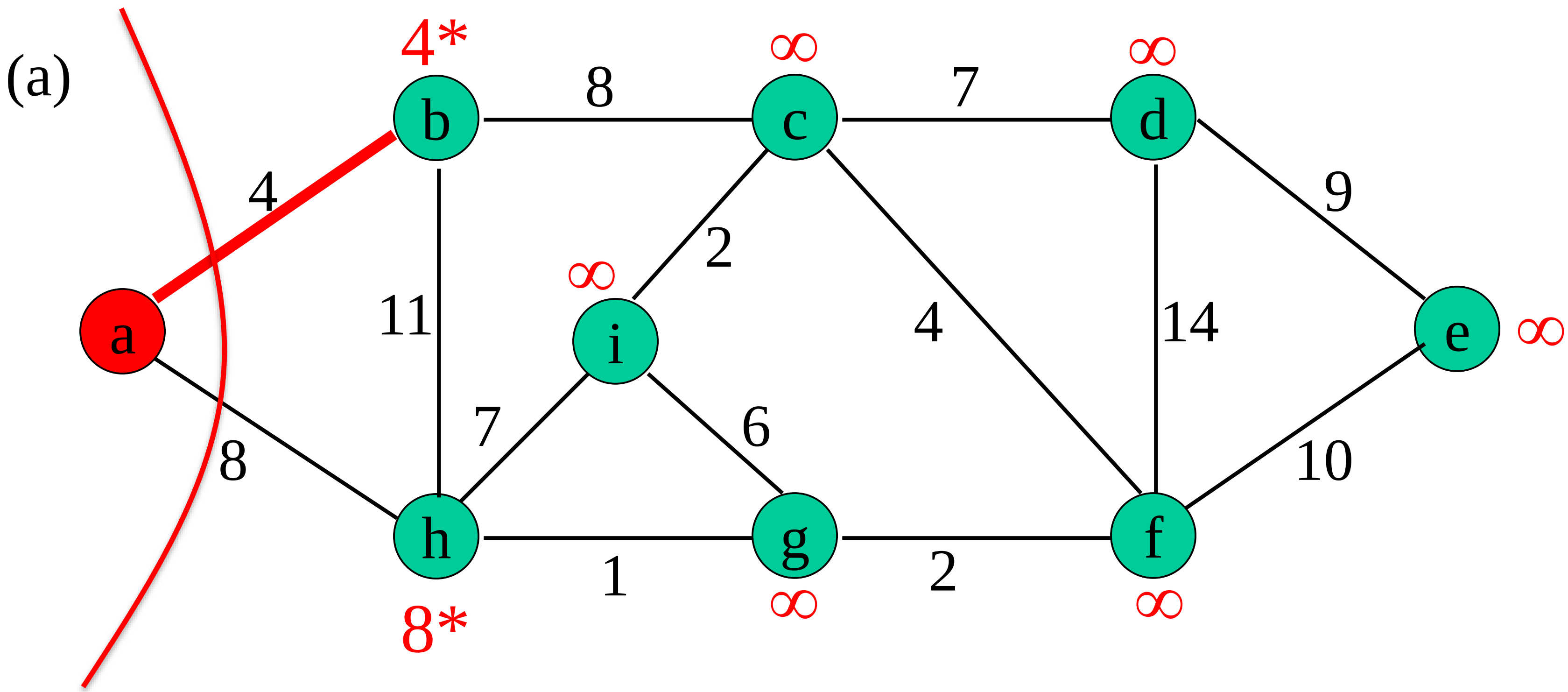
Implementation of Prim's Algorithm

- The key to implementing Prim's algorithm efficiently is to make it easy to select a new edge to be added to A

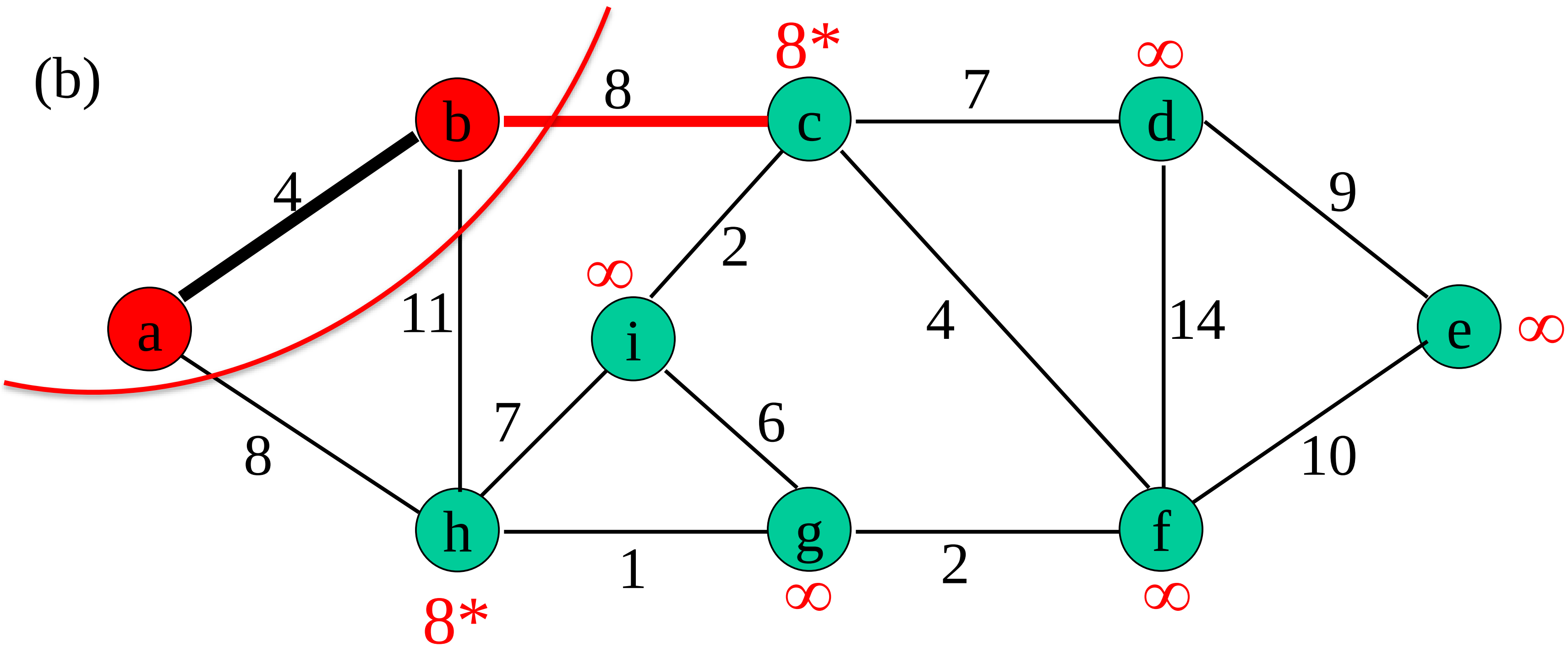
- All vertices that are not in the tree reside in a priority queue Q based on a key field.

- For each vertex *v*, key[v] is the minimum weight of any edge connecting *v* to a vertex in the tree
  key[v] = ∞ if there is no such edge.

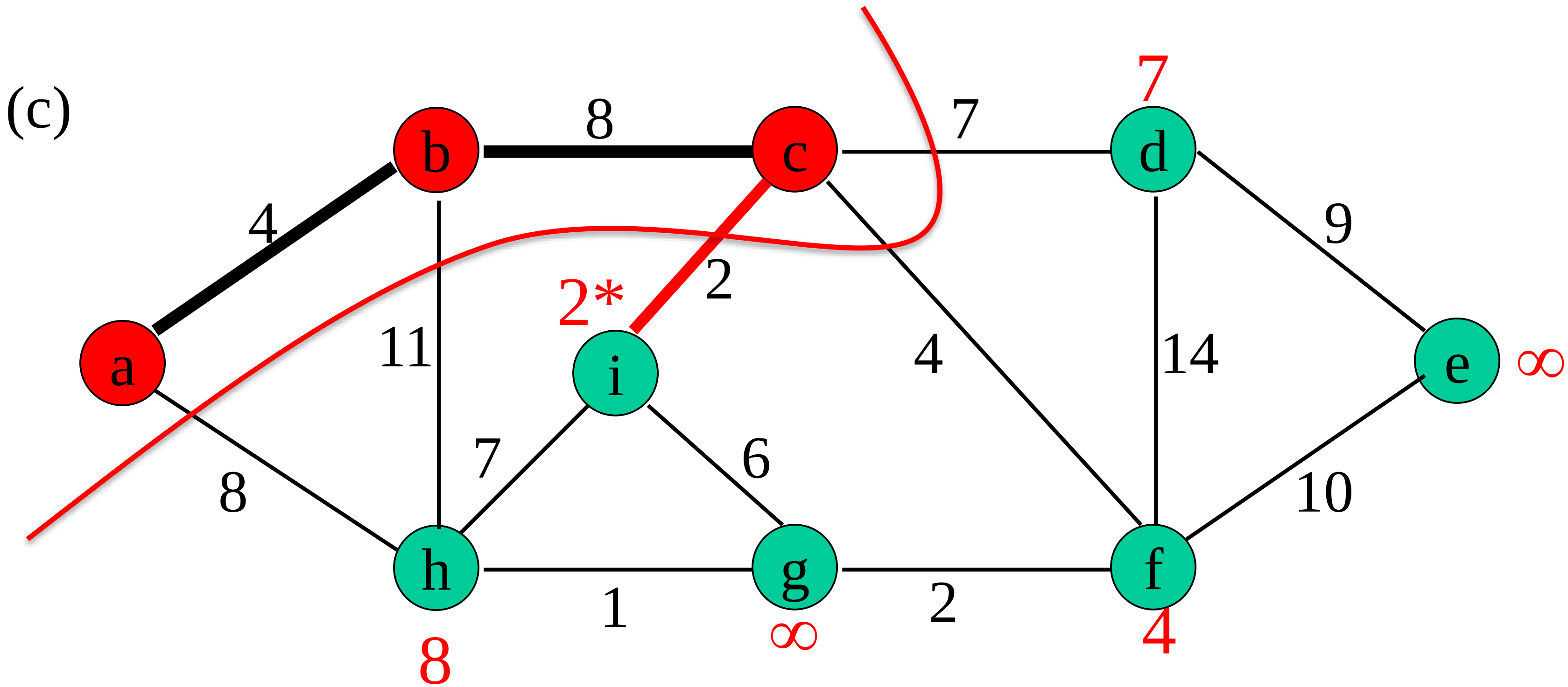
The Execution of Prim's Algorithm



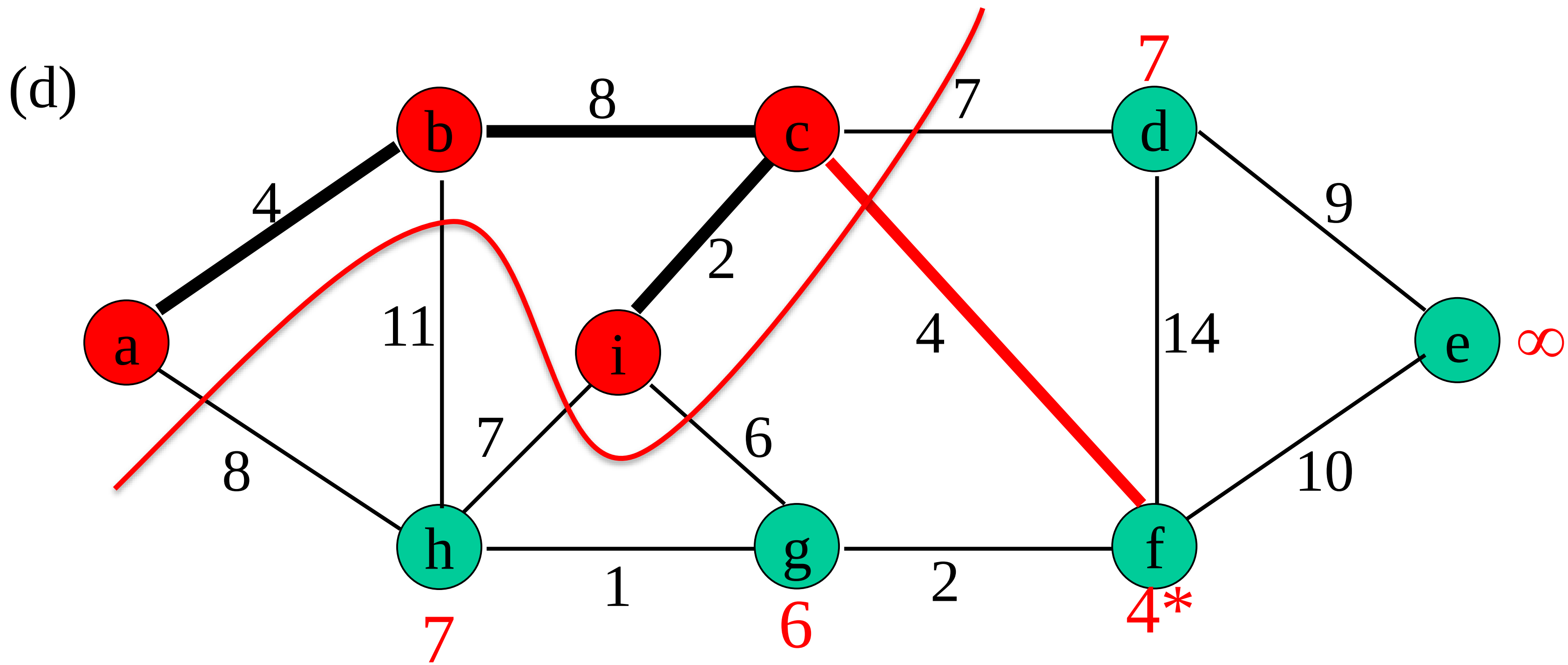
The Execution of Prim’s Algorithm



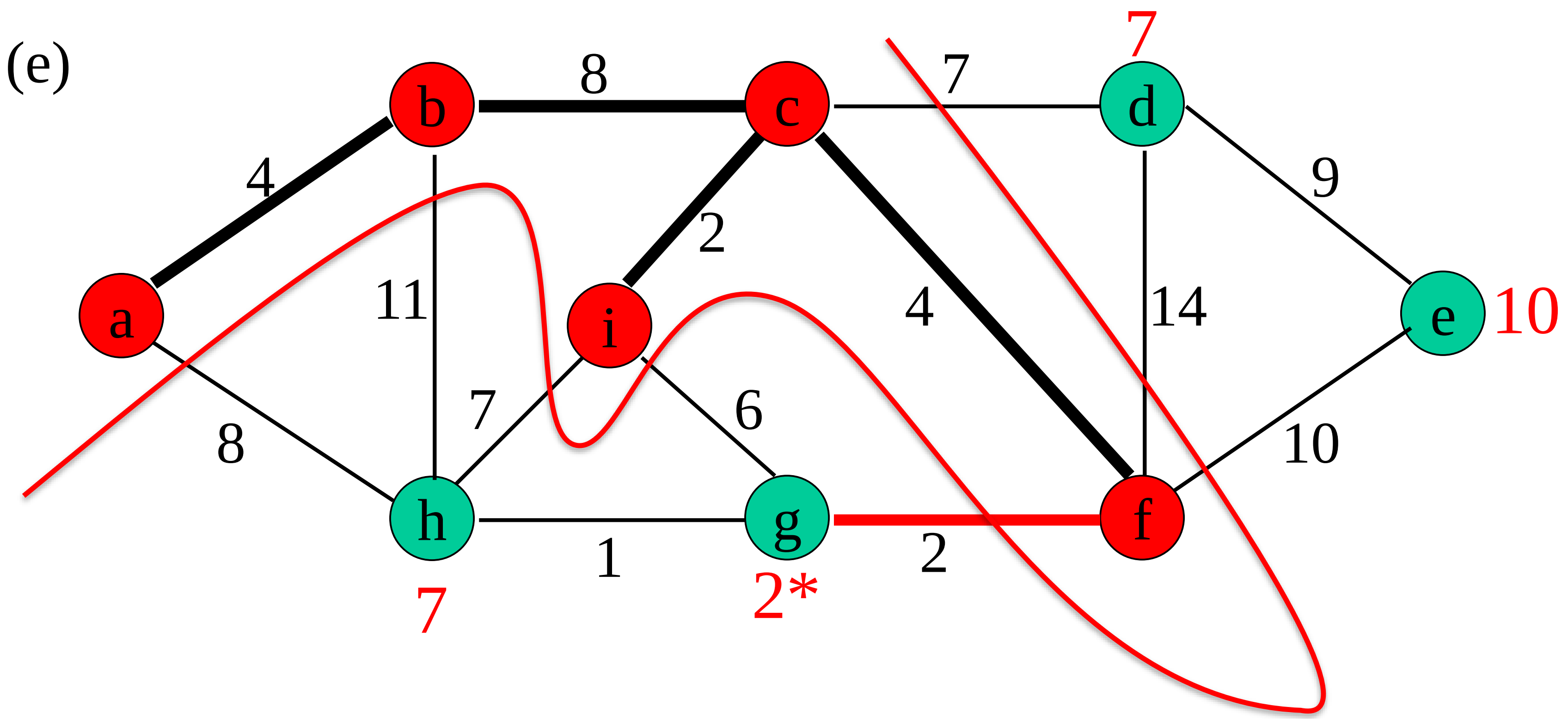
The Execution of Prim's Algorithm



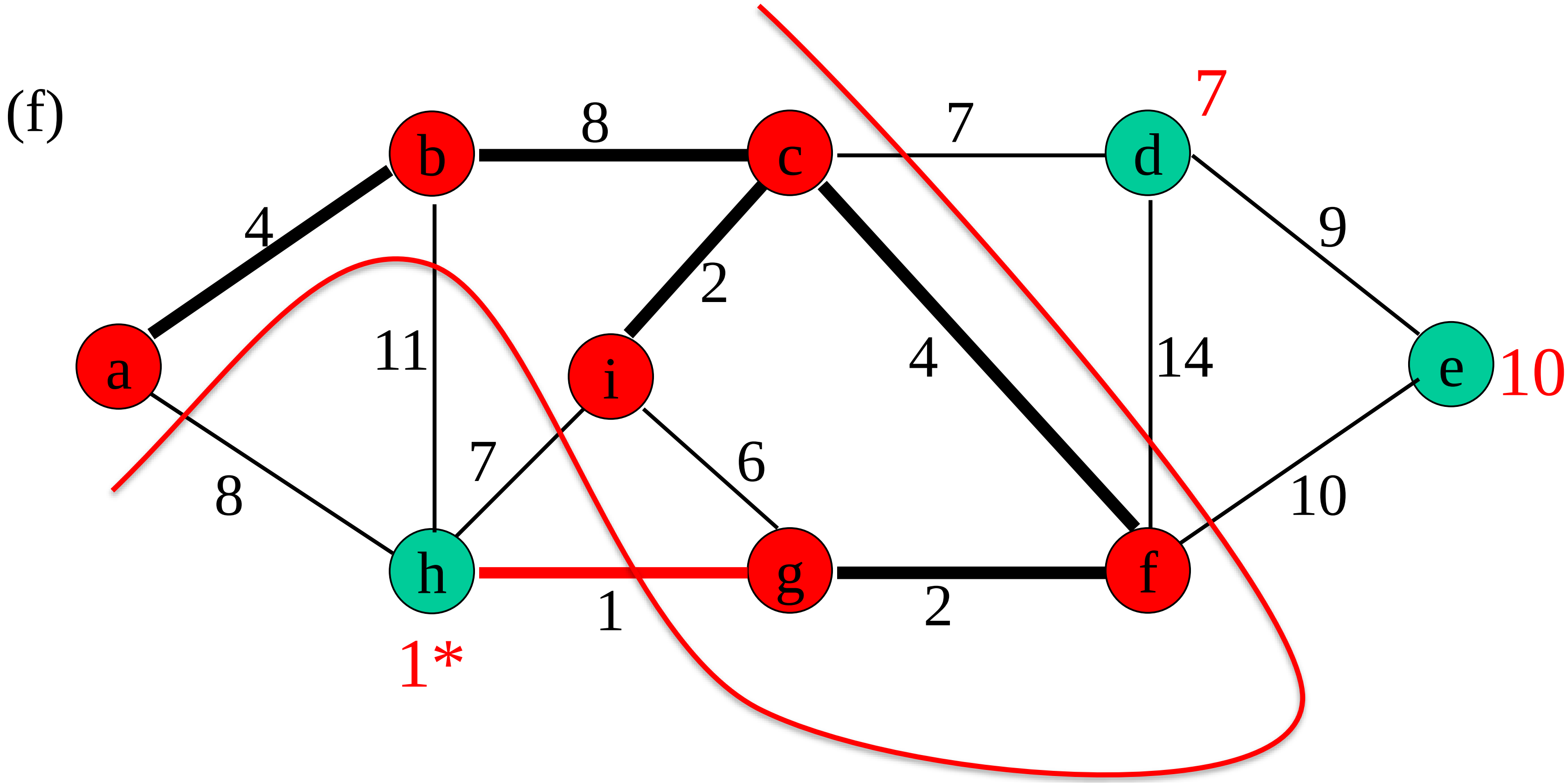
The Execution of Prim's Algorithm



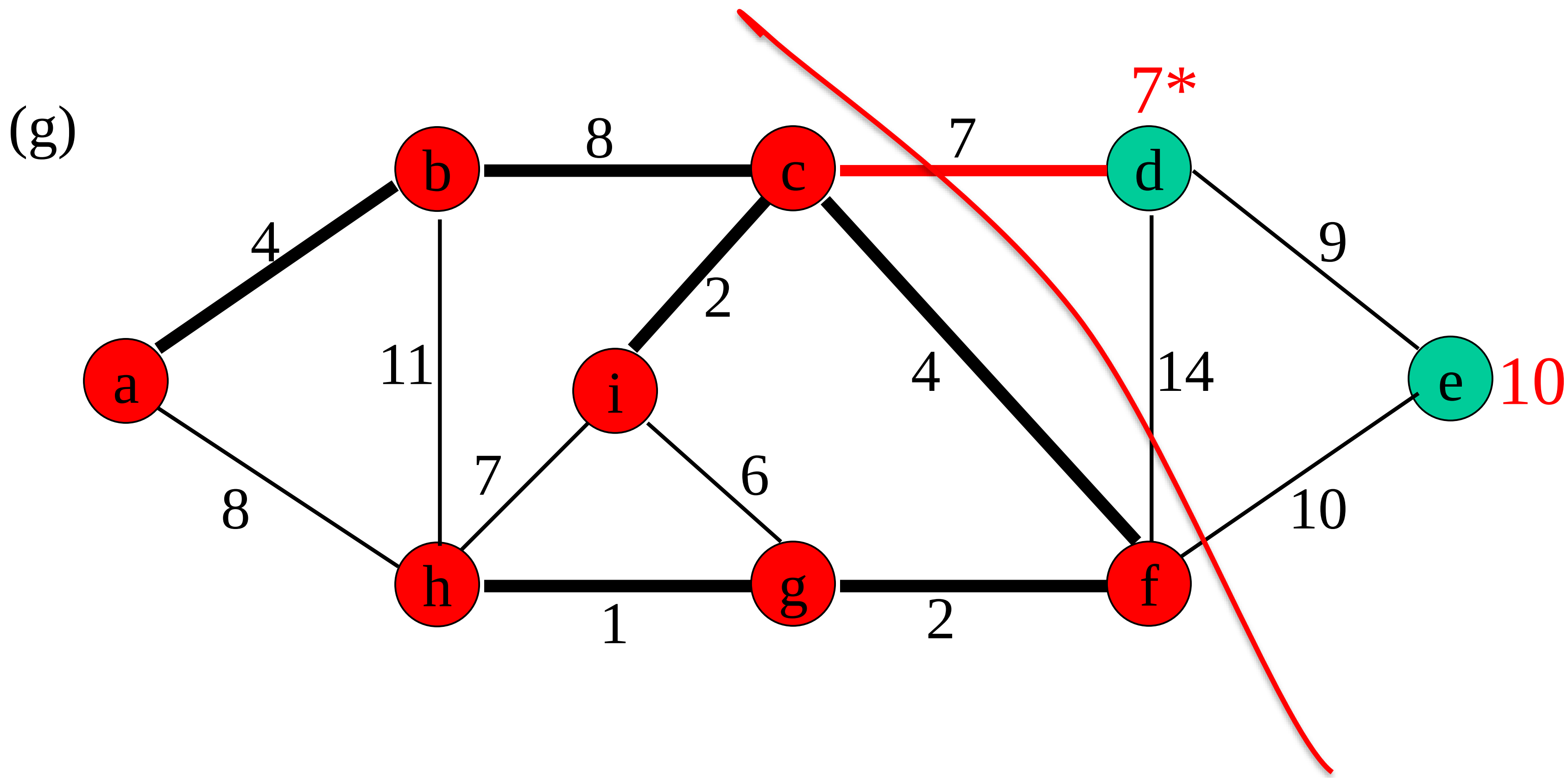
The Execution of Prim’s Algorithm



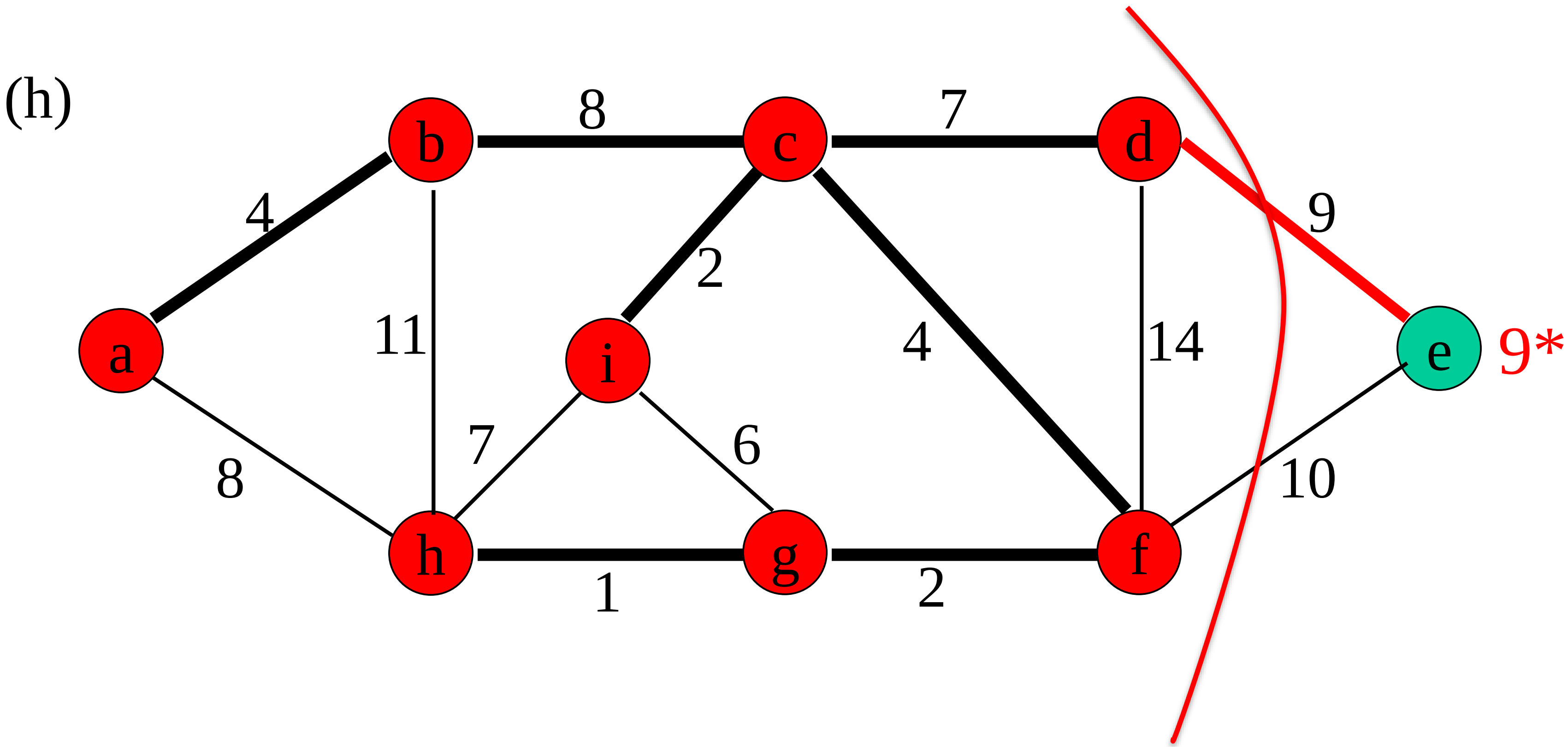
The Execution of Prim's Algorithm



The Execution of Prim's Algorithm

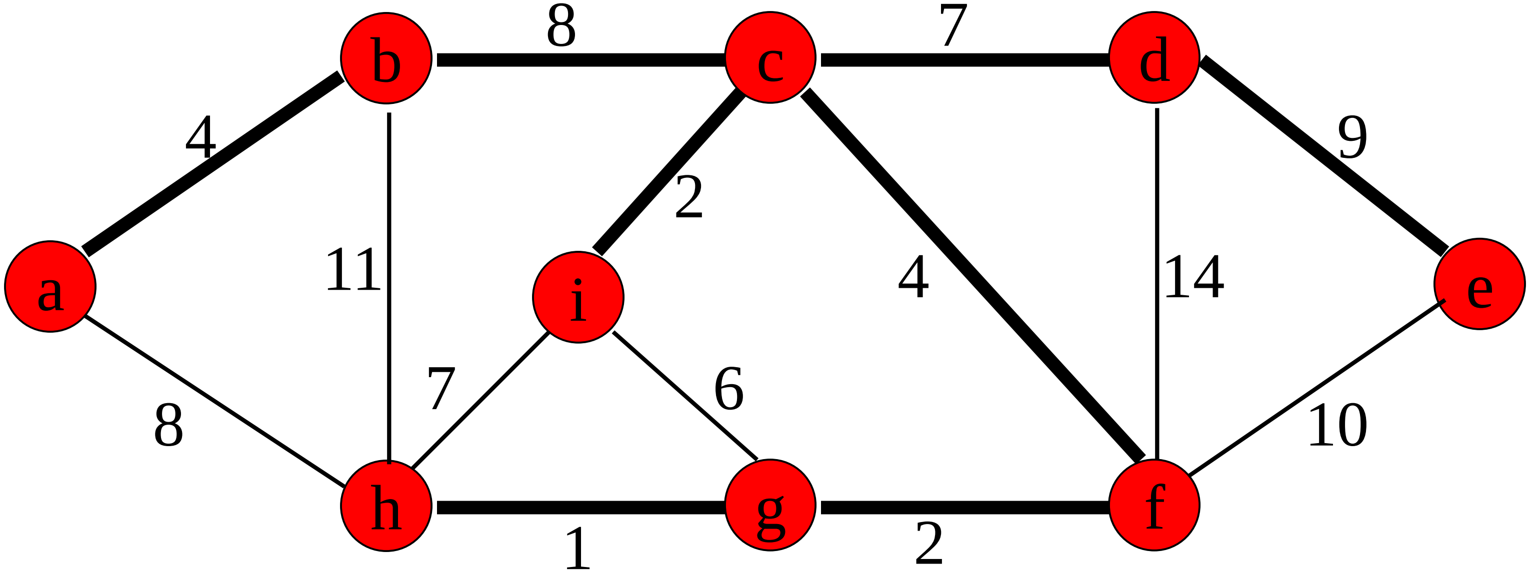


The Execution of Prim's Algorithm

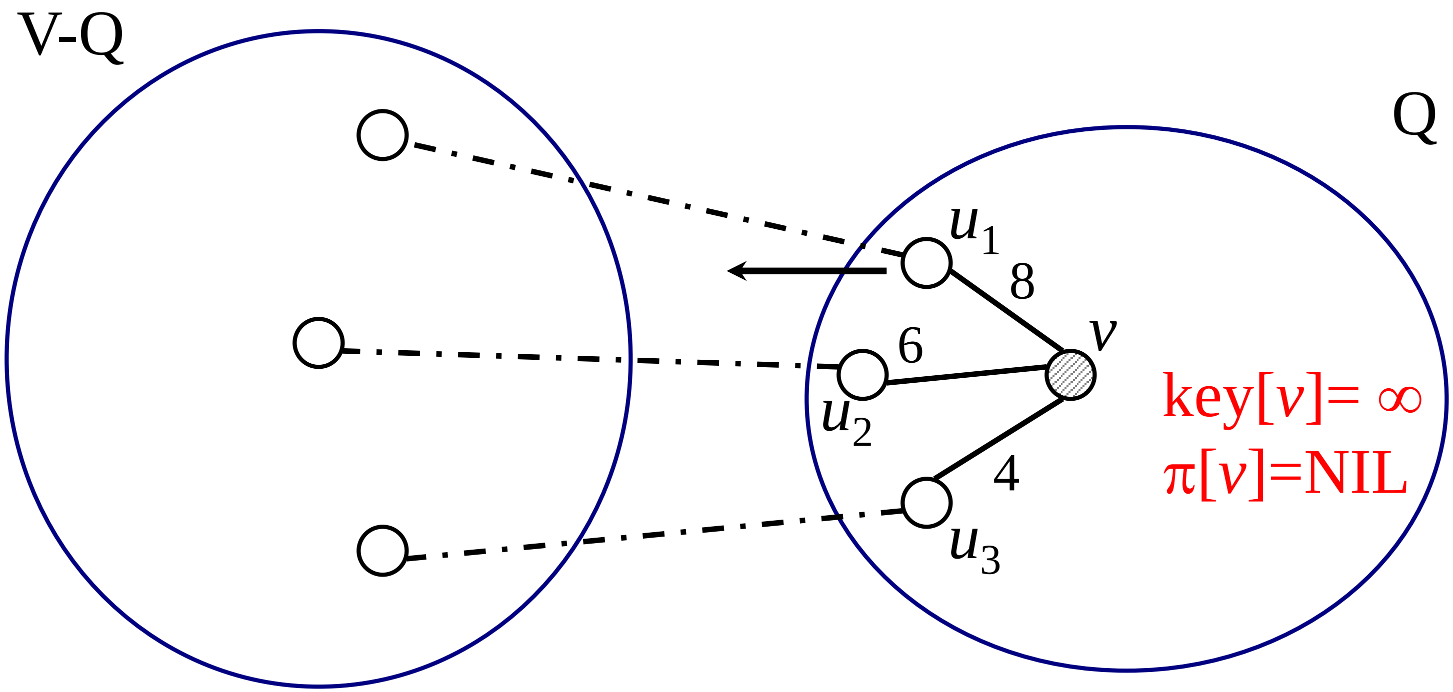


The Execution of Prim’s Algorithm

(i)

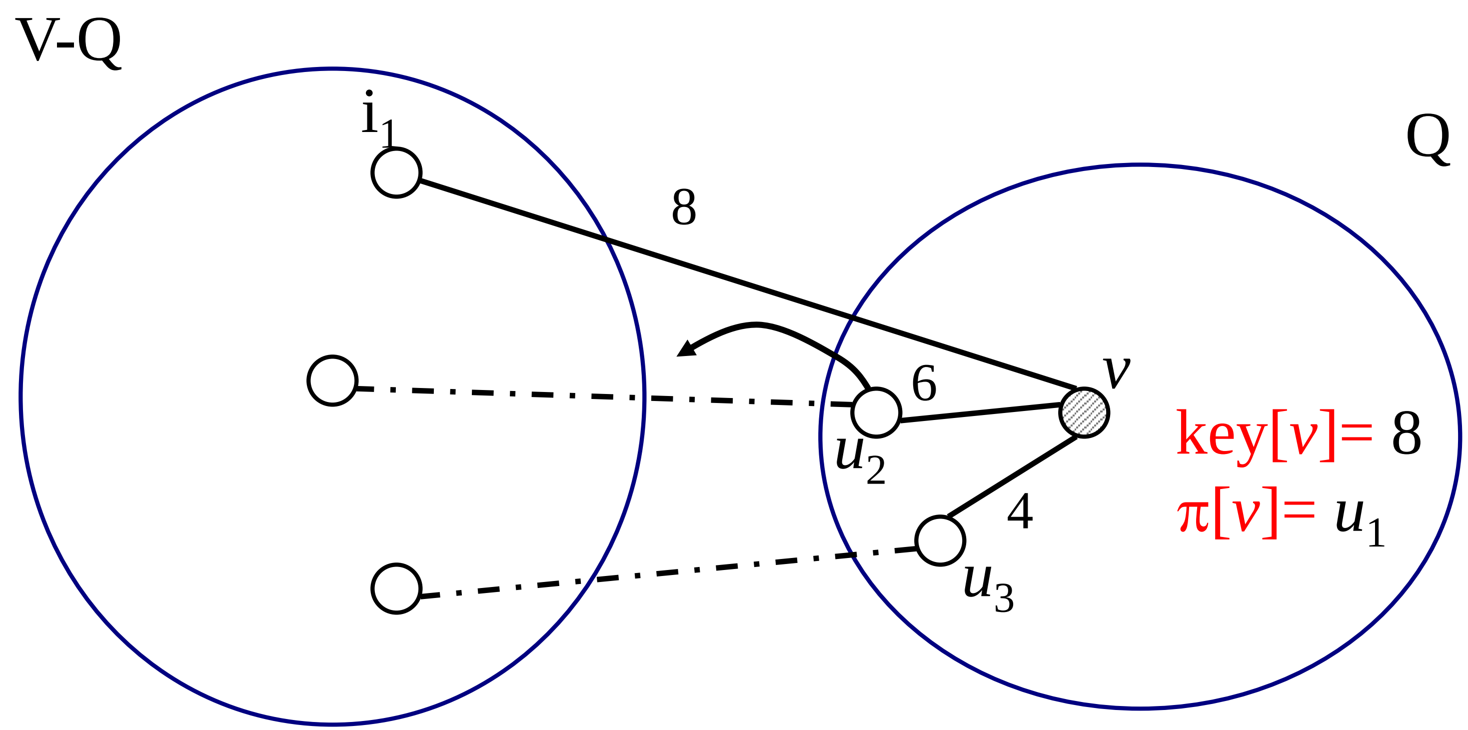


Prim's Algorithm



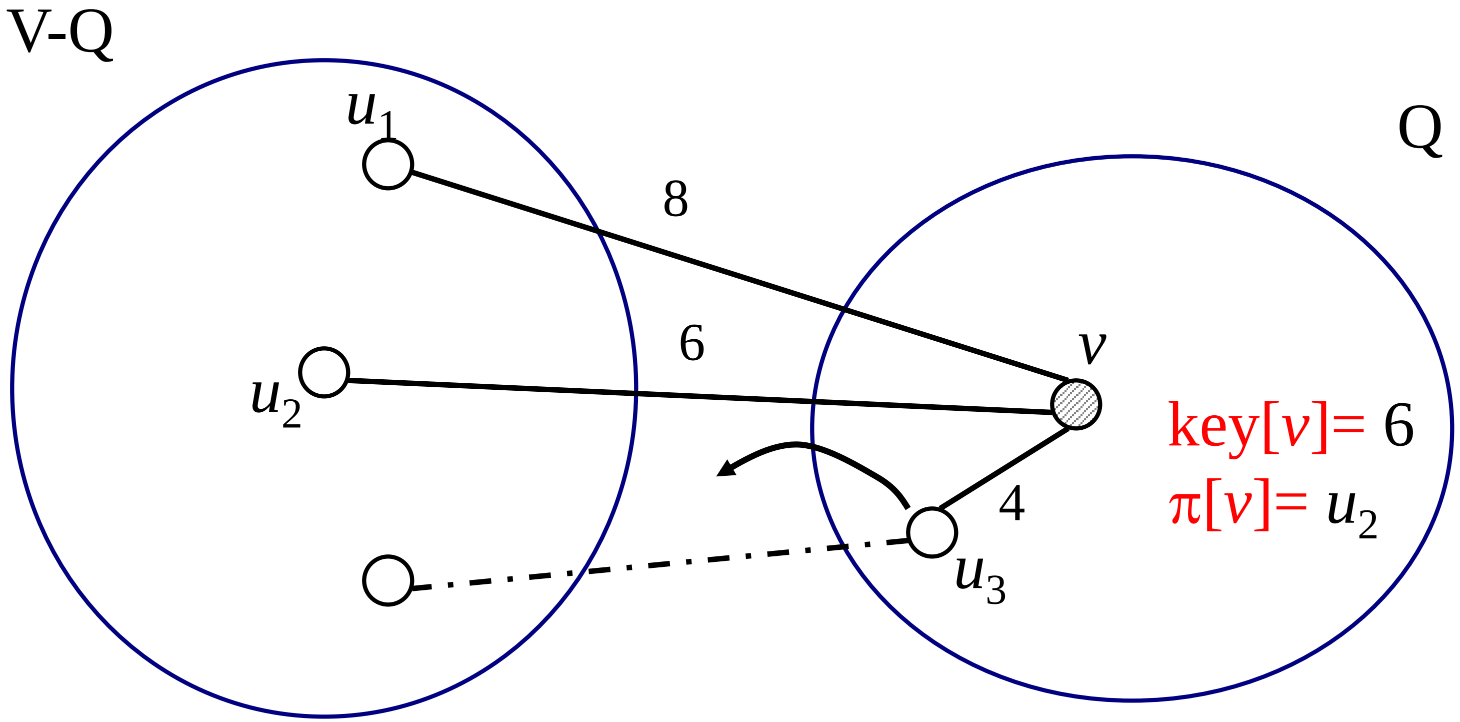
Prim's Algorithm

Vertex u_1 moves from Q to V-Q thru EXTRACT-MIN



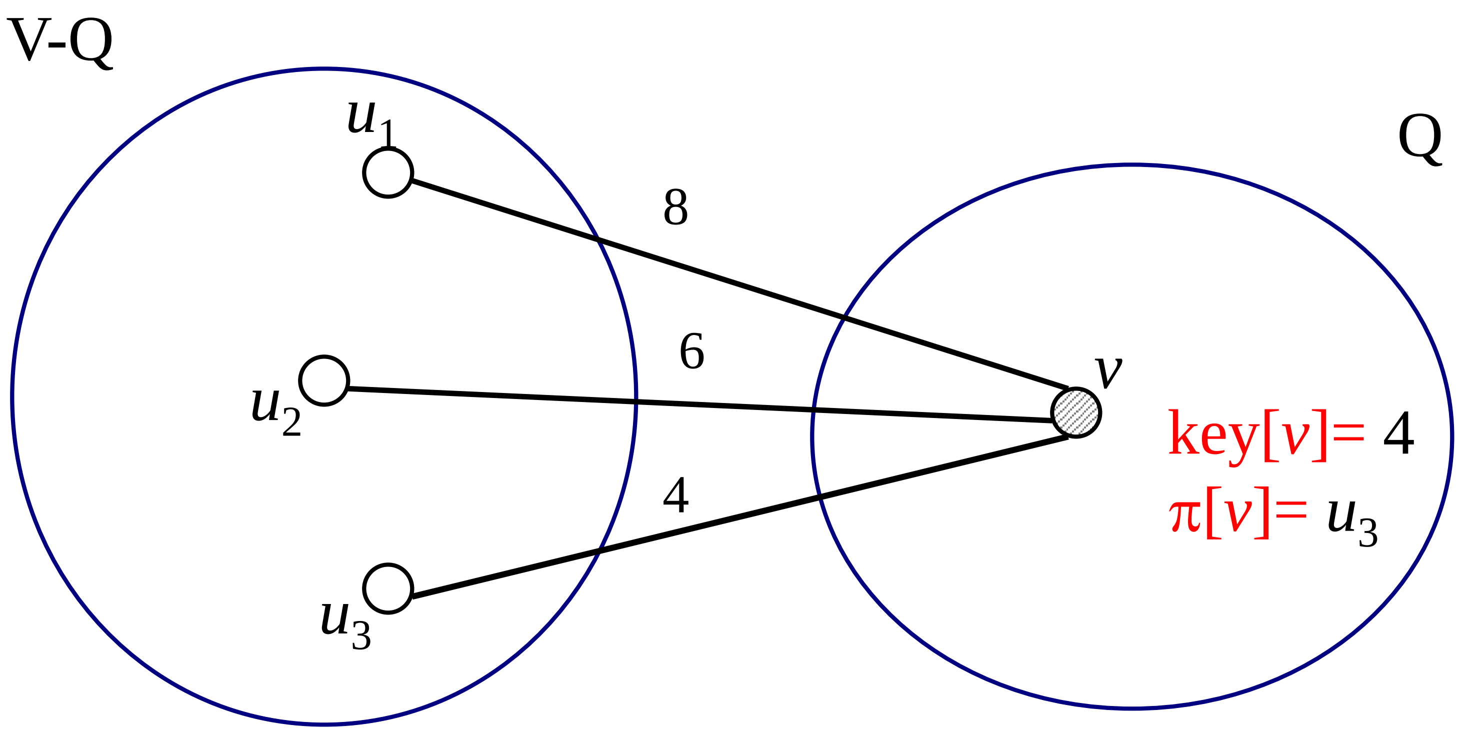
Prim's Algorithm

Vertex u_2 moves from Q to V-Q thru EXTRACT-MIN



Prim's Algorithm

Vertex u_3 moves from Q to V-Q thru EXTRACT-MIN



Prim's Algorithm

- For each vertex v we maintain two fields:

  key [v] : Min. weight of any edge connecting v to a vertex in the tree.

  key [v] = ∞ if there is no such edge

  π [v] : Points to the parent of v in the tree.

- During the algorithm, the set A in Generic-MST is maintained as

  $A = \{ (v, \pi[v]) : v \in V - \{r\} - Q \}$, where r is a random start vertex.

- When the algorithm terminates, the priority queue is empty.

  The MST A for G is thus $A = \{ (v, \pi[v]) : v \in V - \{r\} \}$

Prim's Algorithm

```
MST-PRIM (G, ω, r)
    Q ← V[G]
    for  each u ∈ Q do
         key[u] ← ∞
    key[r] ← 0
    π [r] ← NIL
    BUILD-MIN-HEAP (Q)
    while   Q ≠ Ø do
        u ← EXTRACT-MIN (Q)
        for  each v ∈ Adj [u]  do
             if   v ∈ Q and ω(u, v) < key[v] then
                  π [v] ← u
                  DECREASE-KEY (Q, v, ω(u, v) )
                  /* key[v] ← ω(u, v) */
end
```

Prim's Algorithm

- Through the algorithm, the set V - Q contains the vertices in the tree being grown.

- $u \leftarrow$ EXTRACT-MIN (Q) identifies a vertex $u \in$ Q incident on a light edge crossing the cut (V-Q, Q) with the exception of the first iteration, in which $u = r$

- Removing u from the set Q adds it to the set V - Q of vertices in the tree

Prim's Algorithm

- The inner for-loop updates the key & π fields of every vertex v adjacent to u but not in the tree

- This updating maintains the invariants

$$key[v] \leftarrow \omega(v, \pi[v]), \text{ and}$$

$(v, \pi[v])$ is a light-edge connecting v to the tree

Running Time of Prim's Algorithm

- The performance of Prim's algorithm depends on how we implement the priority queue
- If Q is implemented as a binary heap

Use BUILD-MIN-HEAP procedure to perform the initialization in O (V) time

while-loop is executed |V| times

each EXTRACT-MIN operation takes O (lgV) time

Therefore, the total time for all calls EXTRACT-MIN is O (V lg V)

Running Time of Prim's Algorithm

- The inner for-loop is executed O(E) times altogether since the sum of the lengths of all adjacency lists is 2|E|

- Within the for-loop

The membership test $v \in$ Q can be implemented in constant time by keeping a bit for each vertex whether or not it is in Q and updating the bit when vertex is removed from Q

The assigment key[v] $\leftarrow \omega(u, v)$ involves a DECREASE-KEY operation on the heap which can be implemented in O(lg V) time

Running Time of Prim's Algorithm

- Thus, the total time for Prim's algorithm is

$$O(V \lg V + E \lg V) = O (E \lg V)$$

- The asymptotic running time of Prim's algorithm can be improved by using FIBONACCI HEAPS
- If |V| elements are organized into a fibonacci heap we can perform:

An EXTRACT-MIN operation in O(lgV) amortized time

A DECREASE-KEY operation (line 11) in O(1) amortized time

Running Time of Prim's Algorithm

The asymptotic running time of Prim's algorithm can be improved by using FIBONACCI HEAPS

If |V| elements are organized into a fibonacci heap we can perform:

An EXTRACT-MIN operation in O(lgV) amortized time

A DECREASE-KEY operation in O(1) amortized time

Hence, if we use FIBONACCI-HEAP to implement the priority queue Q the running time of Prim's algorithm improves to:

$O(E + V \lg V)$