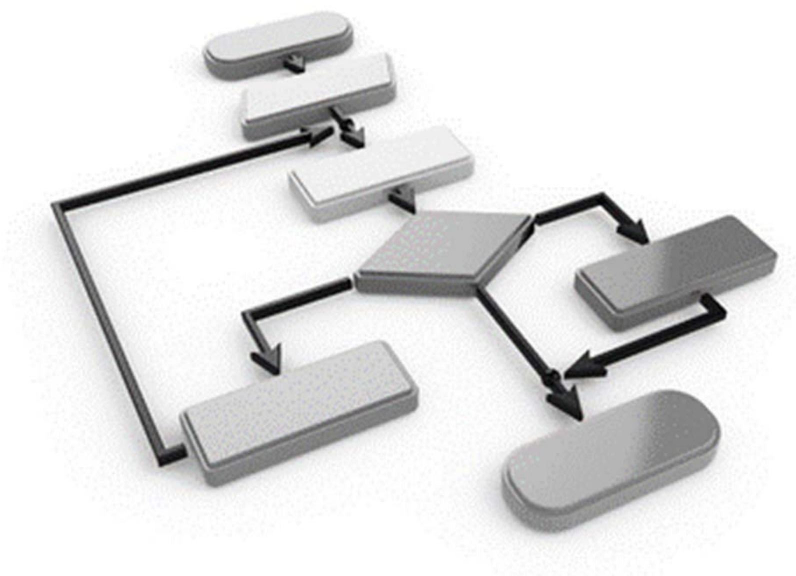


Lecture 17

Graph-Based Algorithms

CSE373: Design and Analysis of Algorithms

Definition of MST

- Let $G = (V, E)$ be a connected, undirected graph.
- For each edge (u, v) in E, we have a weight $w(u, v)$ specifying the cost (length of edge) to connect u and v.
- We wish to find a (acyclic) subset T of E that connects all of the vertices in V and whose total weight is minimized.
- Since the total weight is minimized, the subset T must be acyclic (no circuit).
- Thus, T is a tree. We call it a spanning tree.
- The problem of determining the tree T is called the minimum-spanning-tree problem.

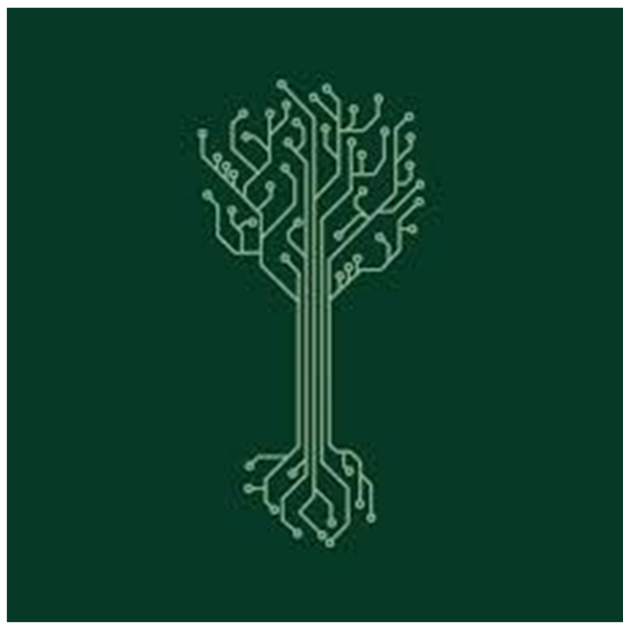
Application of MST

In the design of electronic circuitry, it is often necessary to make a set of pins electrically equivalent by wiring them together.

To interconnect n pins, we can use n-1 wires, each connecting two pins.

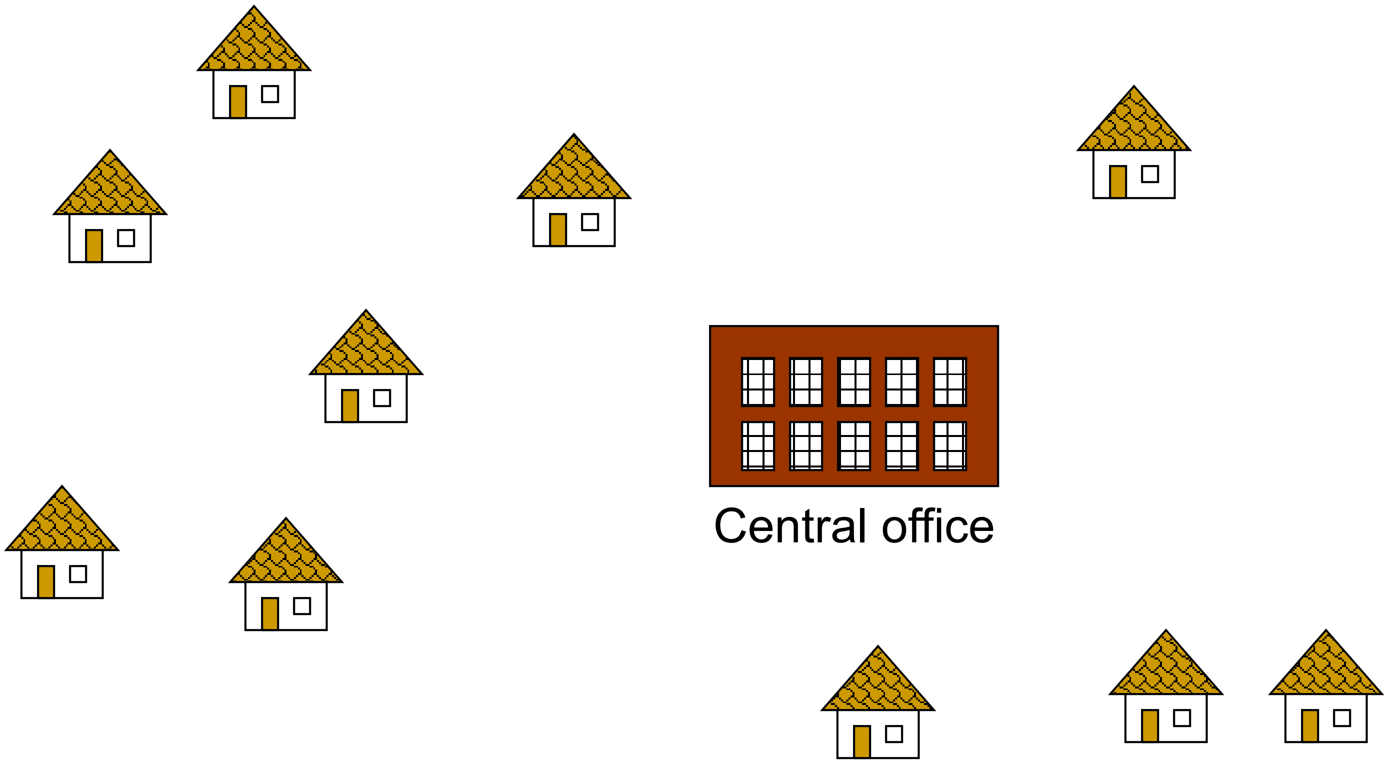
We want to minimize the total length of the wires.

Minimum Spanning Trees can be used to model this problem.



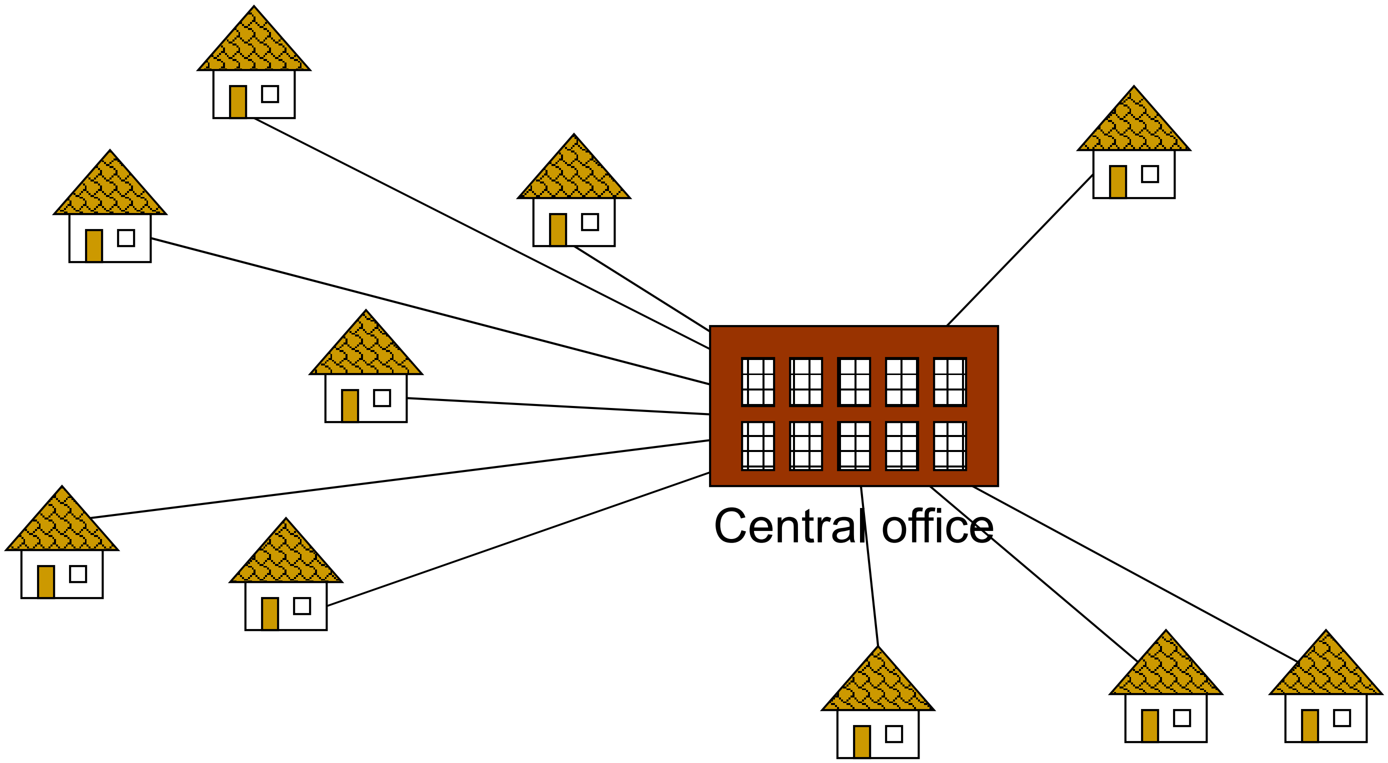
Application of MST

Problem: Laying Telephone Wire



Application of MST

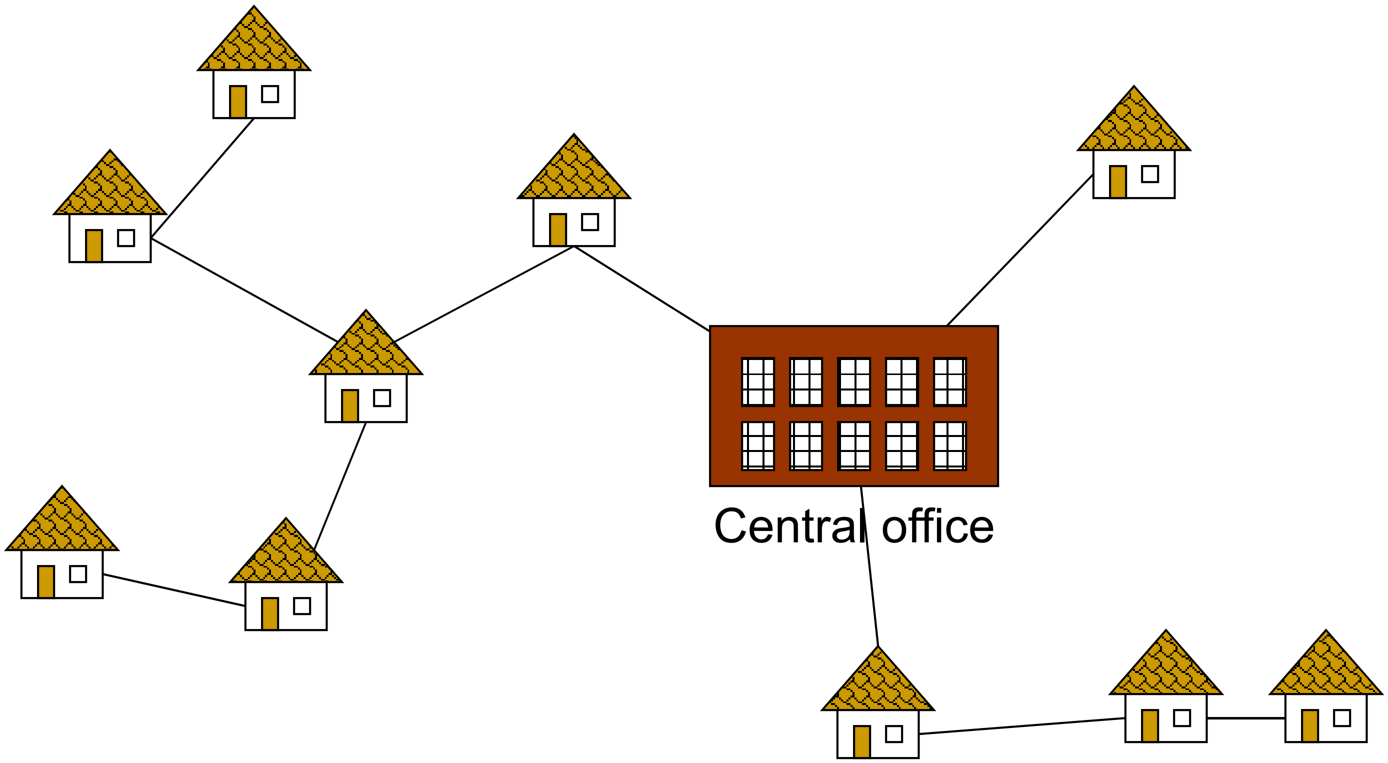
Wiring: Naïve Approach



Expensive!

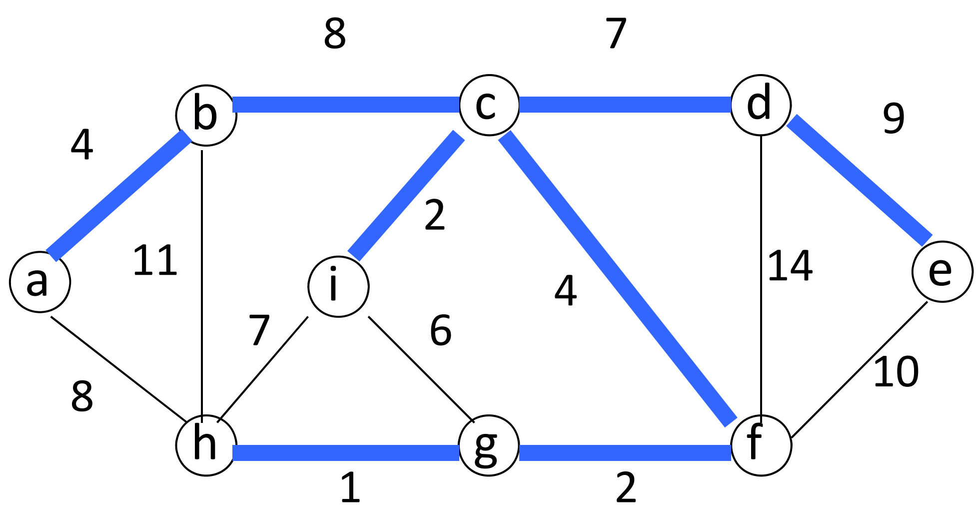
Application of MST

Wiring: Better Approach



Minimize the total length of wire connecting the customers

Here is an example of a connected graph and its minimum spanning tree:



Notice that the tree is not unique:
replacing (b,c) with (a,h) yields another spanning tree with the same minimum weight.

Growing a MST(Generic Algorithm)

Set A is always a subset of some minimum spanning tree. This property is called the invariant Property.

An edge (u, v) is a safe edge for A if adding the edge to A does not destroy the invariant.

A safe edge is just the CORRECT edge to choose to add to T.

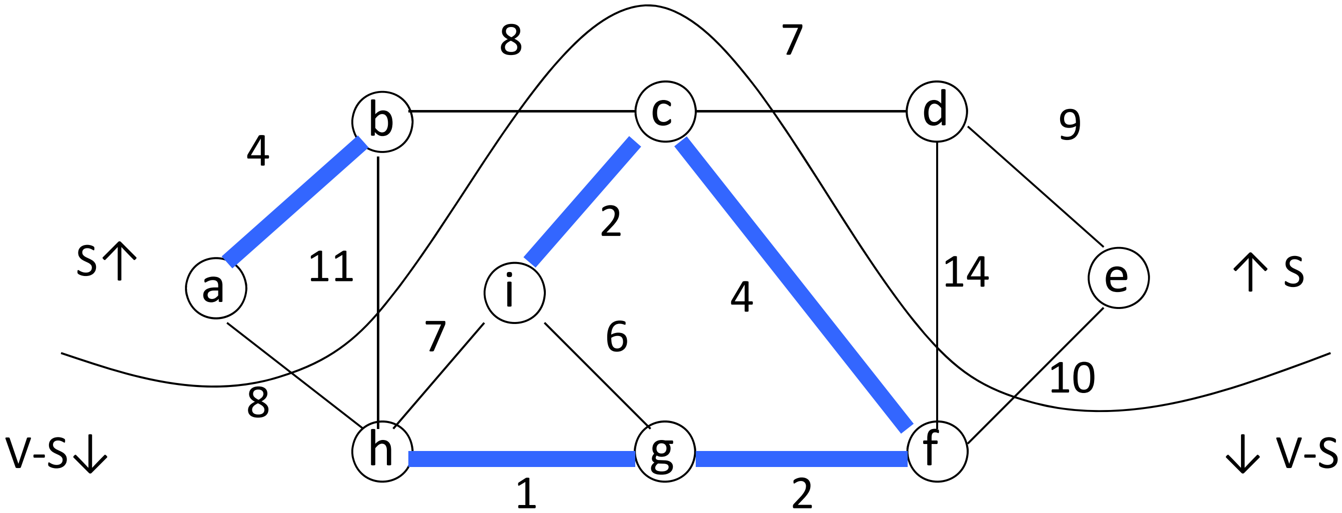
```
GENERIC_MST(G,w)
    A = Ø
    while A does not form a spanning tree
        find an edge (u,v) that is safe for A
        A = A∪{(u,v)}
    return A
```

How to Find a Safe Edge

We need some definitions and a theorem.

- A cut $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V.
- An edge crosses the cut $(S, V - S)$ if one of its endpoints is in S and the other is in $V - S$.
- An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut.

How to Find a Safe Edge



- This figure shows a cut (S,V-S) of the graph.
- The edge (d,c) is the unique light edge crossing the cut.

The Algorithms of Kruskal and Prim

- The two algorithms are elaborations of the generic algorithm.
- They each use a specific rule to determine a safe edge in line 3 of GENERIC_MST.
- In Kruskal's algorithm,
  - The set A is a forest.
  - The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.
- In Prim's algorithm,
  - The set A forms a single tree.
  - The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

Related Topics

Disjoint-Set (Chapter 21, Page 571)

- Keep a collection of sets S_1, S_2, .., S_k,
  - Each S_i is a set, e,g, $S_1=\{v_1, v_2, v_8\}$.
- Three operations
  - **Make-Set(x)** - creates a new set whose only member is **x**.
  - **Union(x, y)** – unites the sets that contain **x** and **y**, say, $\mathbf{S_x}$ and $\mathbf{S_y}$, into a new set that is the union of the two sets.
  - **Find-Set(x)** - returns a pointer to the representative of the set containing **x**.
  - Each operation takes O(log n) time.

Kruskal's Algorithm

```
MST_KRUSKAL(G,w)
    A = ∅
    for each vertex v ∈ G.V
        MAKE_SET(v)
    sort the edges of G.E into nondecreasing order by weight w
    for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
        if FIND_SET(u) ≠ FIND_SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
    return A
```

Kruskal's Algorithm

Our implementation uses a disjoint-set data structure to maintain several disjoint sets of elements.

Each set contains the vertices in a tree of the current forest.

The operation FIND_SET(u) returns a representative element from the set that contains u.

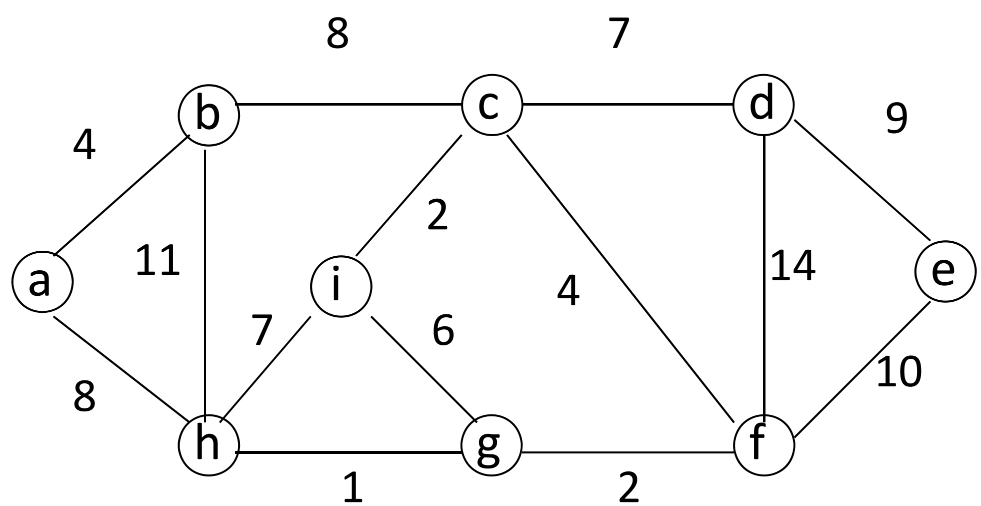
Thus, we can determine whether two vertices u and v belong to the same tree by testing whether FIND_SET(u) equals FIND_SET(v).

The combining of trees is accomplished by the UNION procedure.

Running time $O(E \lg E)$.

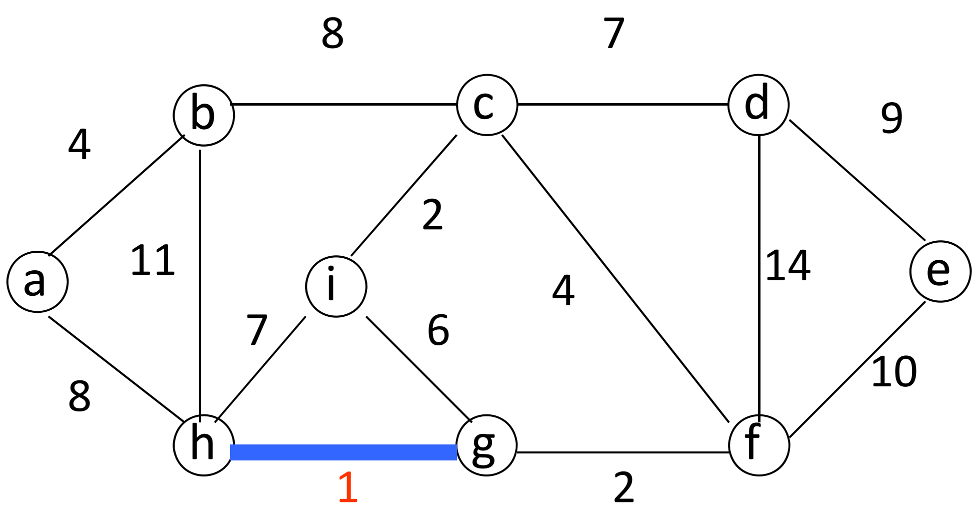
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



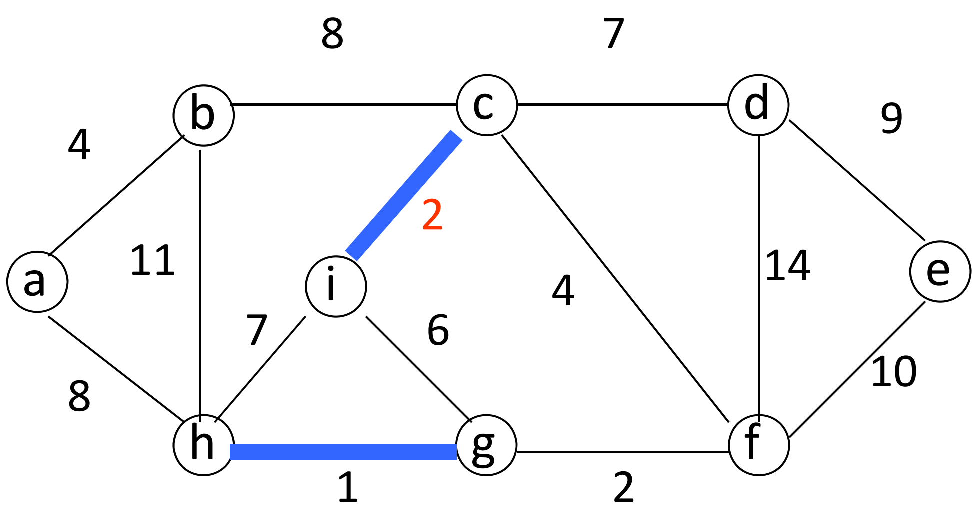
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



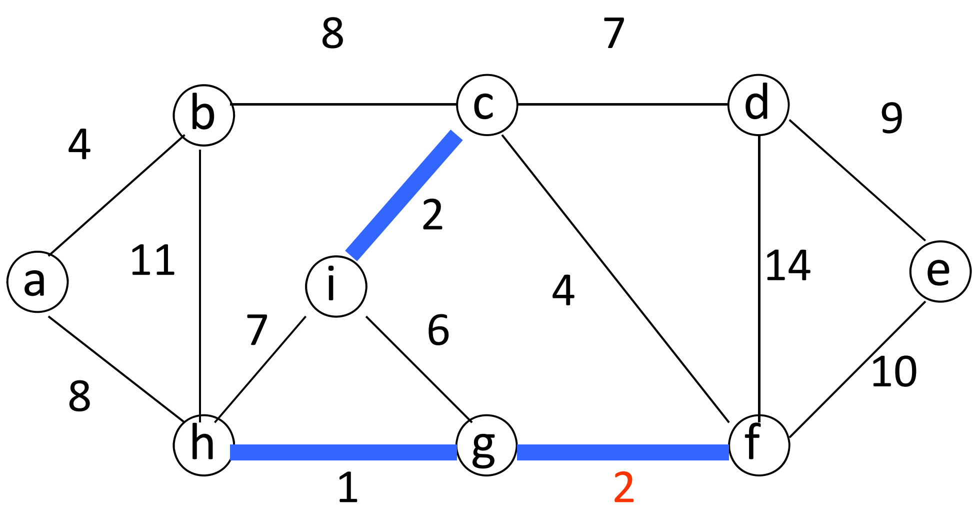
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



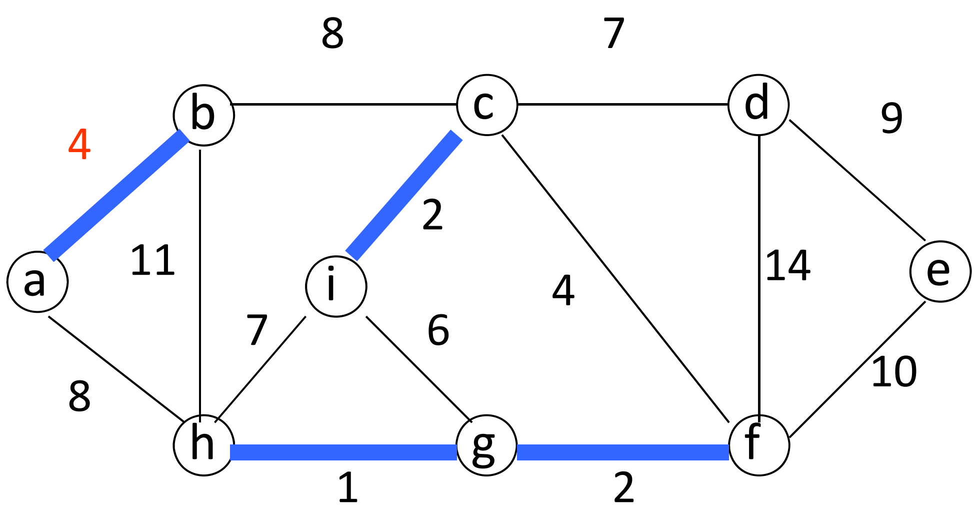
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



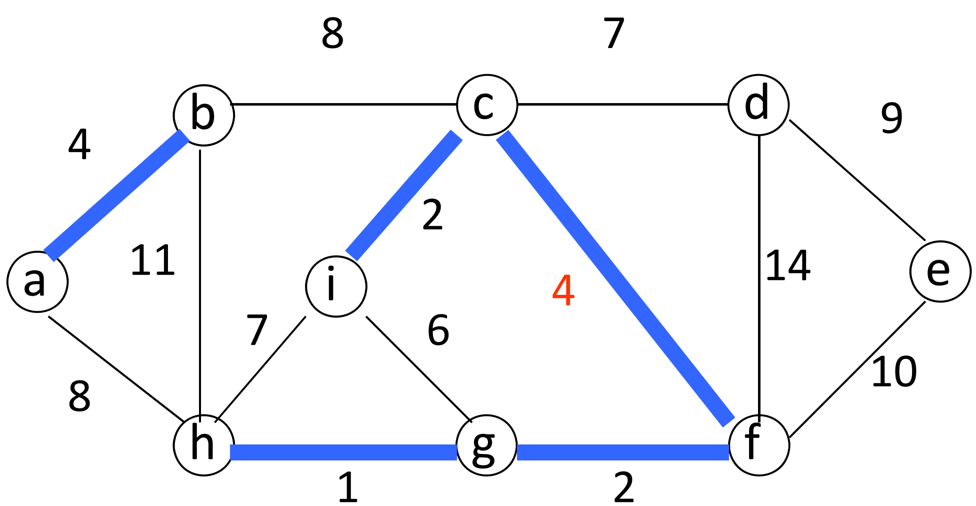
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



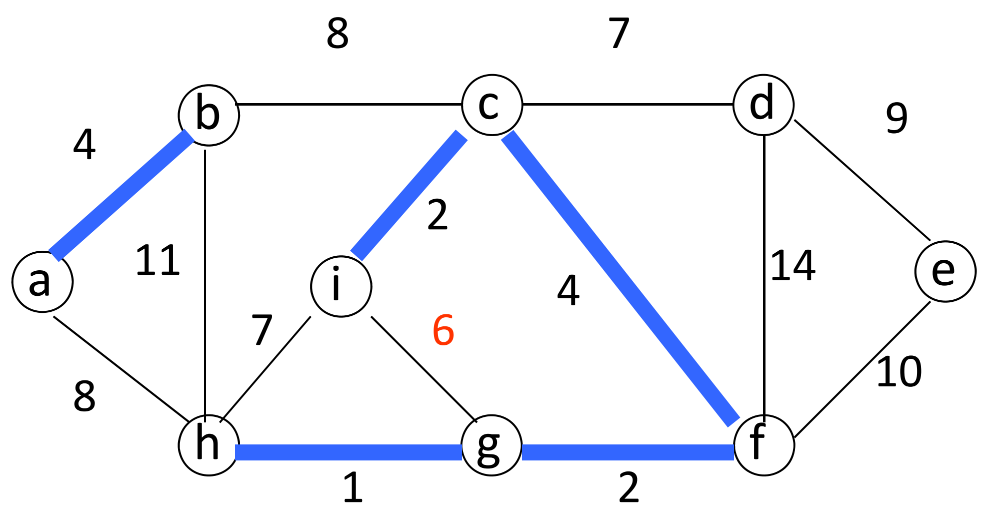
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



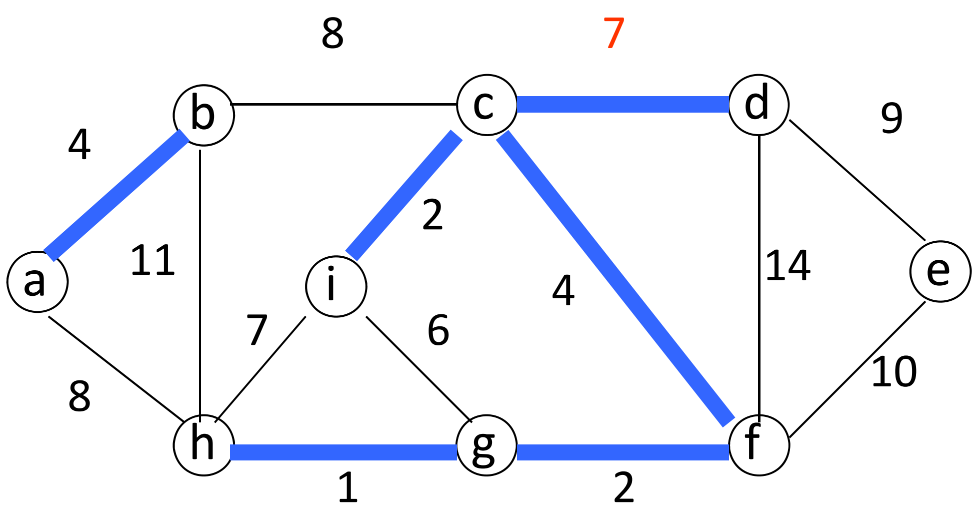
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



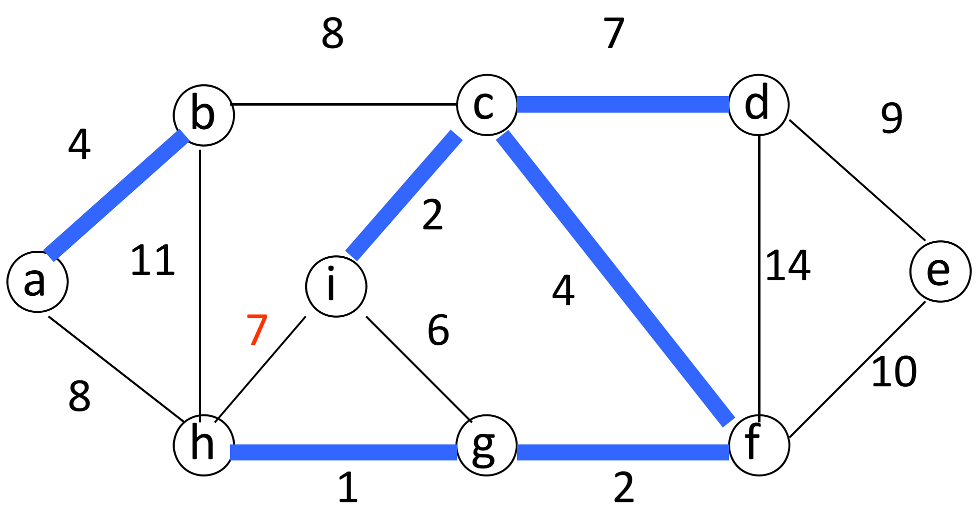
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



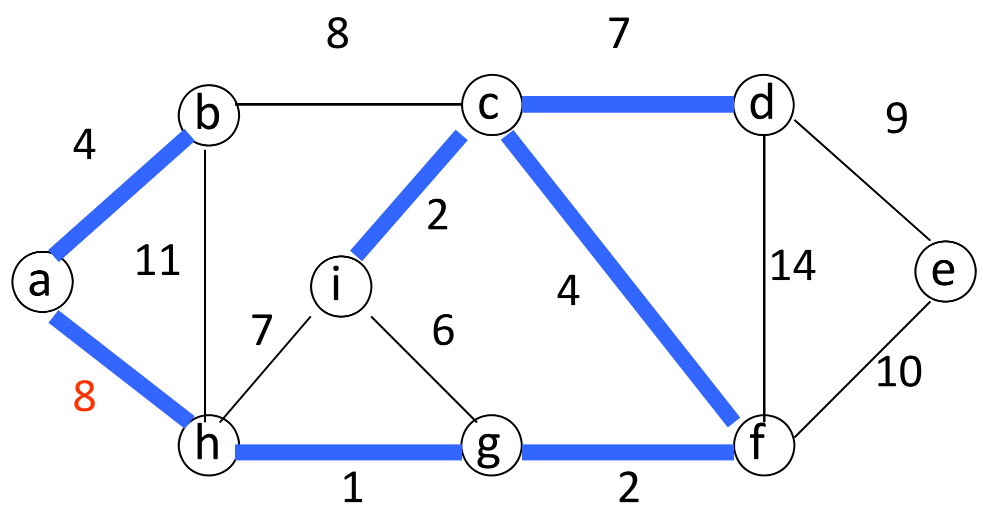
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



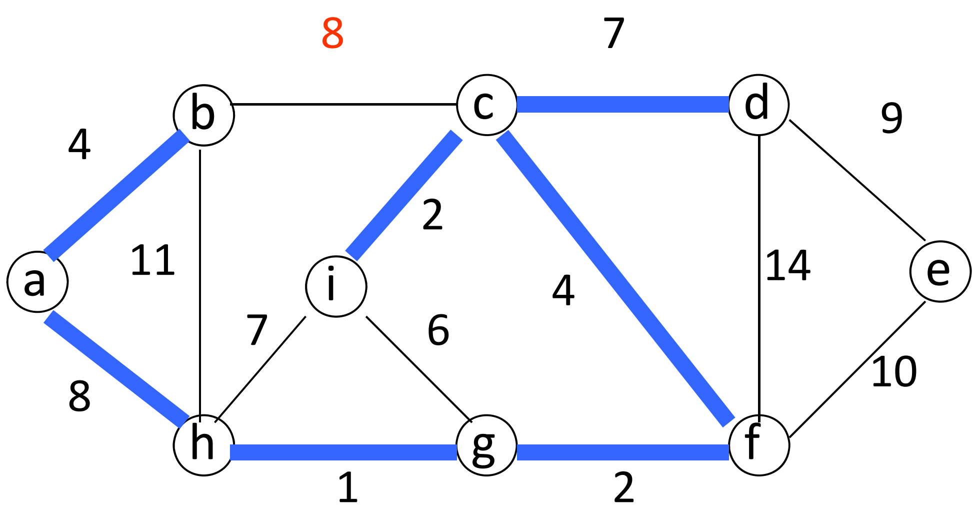
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



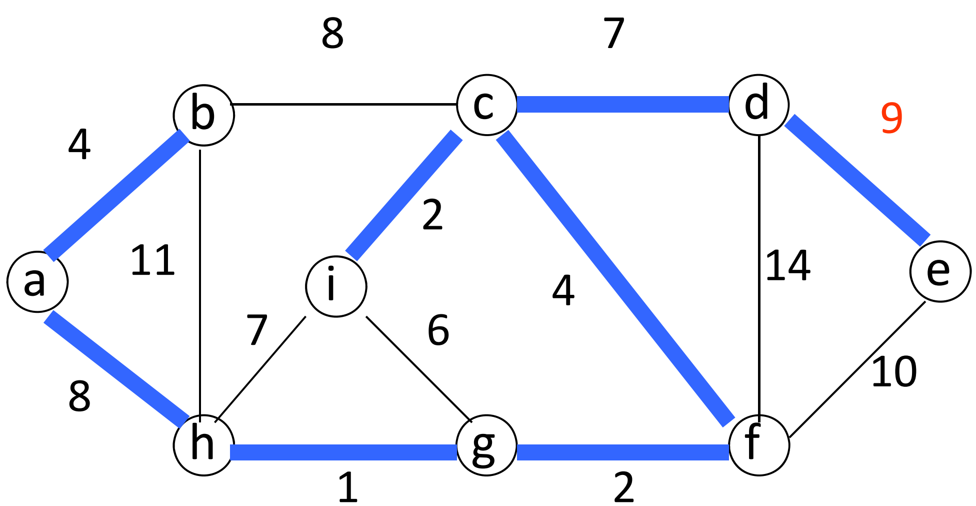
Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



Kruskal's Algorithm: Example

- The edges are considered by the algorithm in sorted order by weight.
- The edge under consideration at each step is shown with a red weight number.



Prim's Algorithm

```
MST_PRIM(G,w,r)
for each u in G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V    //Q is a min-priority queue
while Q ≠ ∅
    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

Grow the minimum spanning tree from the root vertex r.

Q is a priority queue, holding all vertices that are not in the tree now.

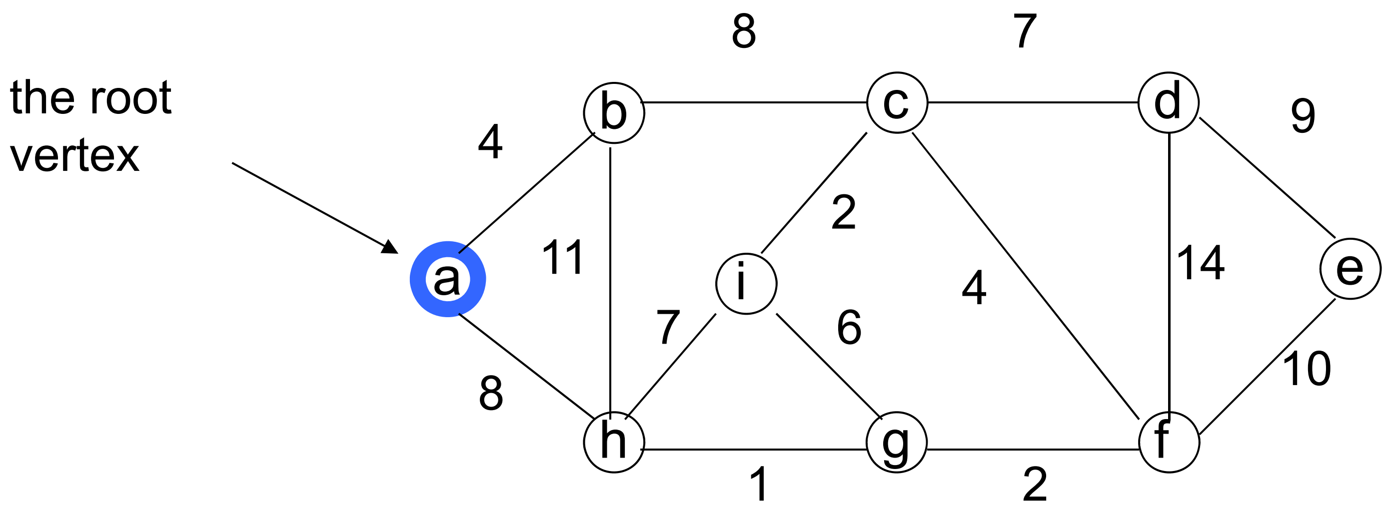
key[v] is the minimum weight of any edge connecting v to a vertex in the tree.

parent[v] names the parent of v in the tree.

When the algorithm terminates, Q is empty; the minimum spanning tree A for G is thus $A=\{(v,parent[v]):v\in V-\{r\}\}$.

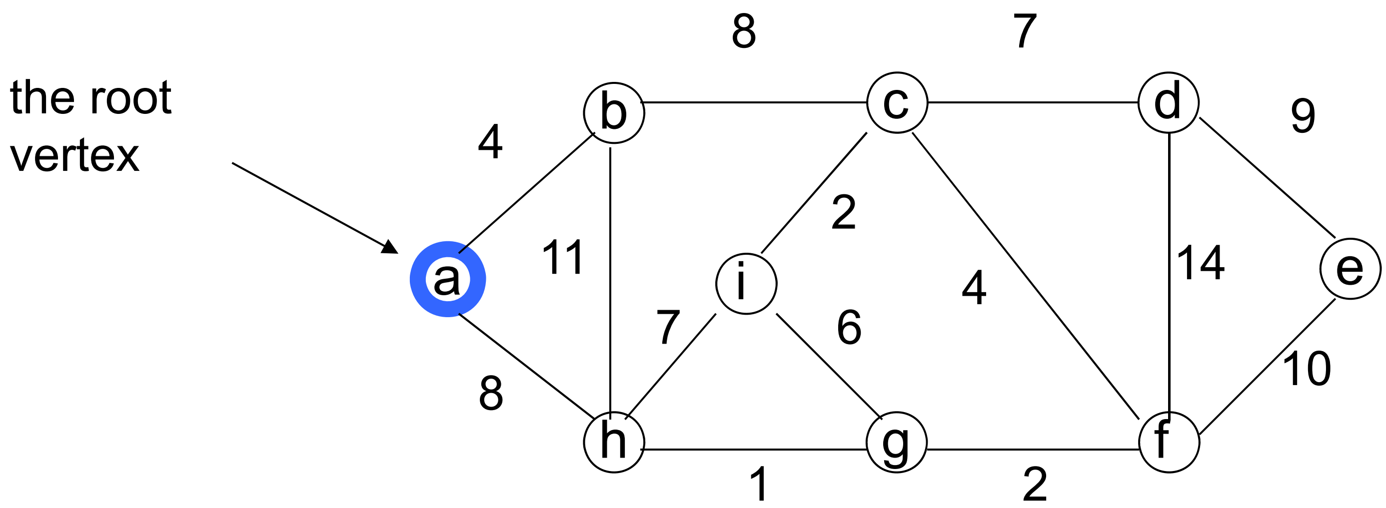
Running time: $O(||E||\lg |V|)$.

Prim's Algorithm: Example



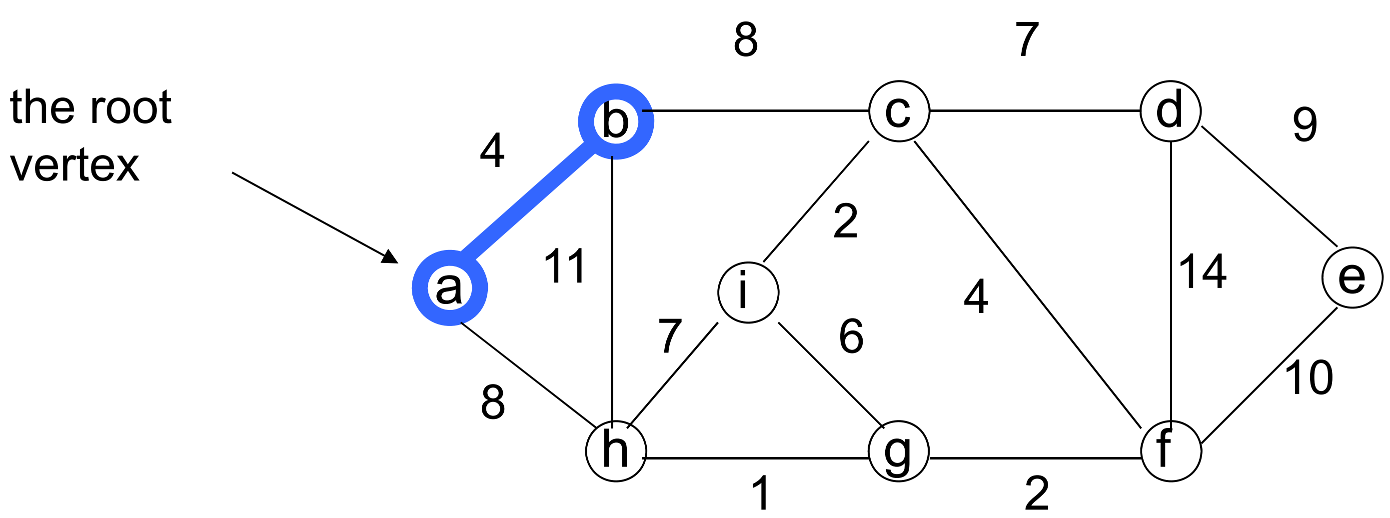
V	a	b	c	d	e	f	g	h	i
T	1	0	0	0	0	0	0	0	0
Key	0	-	-	-	-	-	-	-	-
π	-1	-	-	-	-	-	-	-	-

Prim's Algorithm: Example



V	a	b	c	d	e	f	g	h	i
T	1	**0**	0	0	0	0	0	0	0
Key	0	**4**	-	-	-	-	-	8	-
π	-1	**a**	-	-	-	-	-	a	-

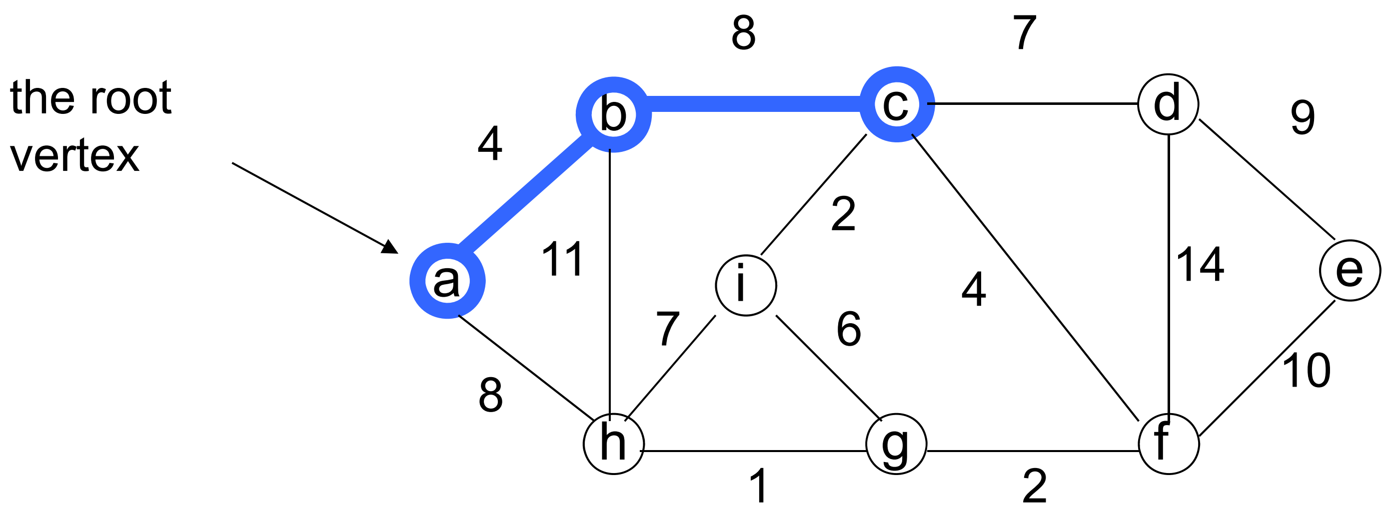
Prim's Algorithm: Example



Important: Update Key[v] only if T[v]==0

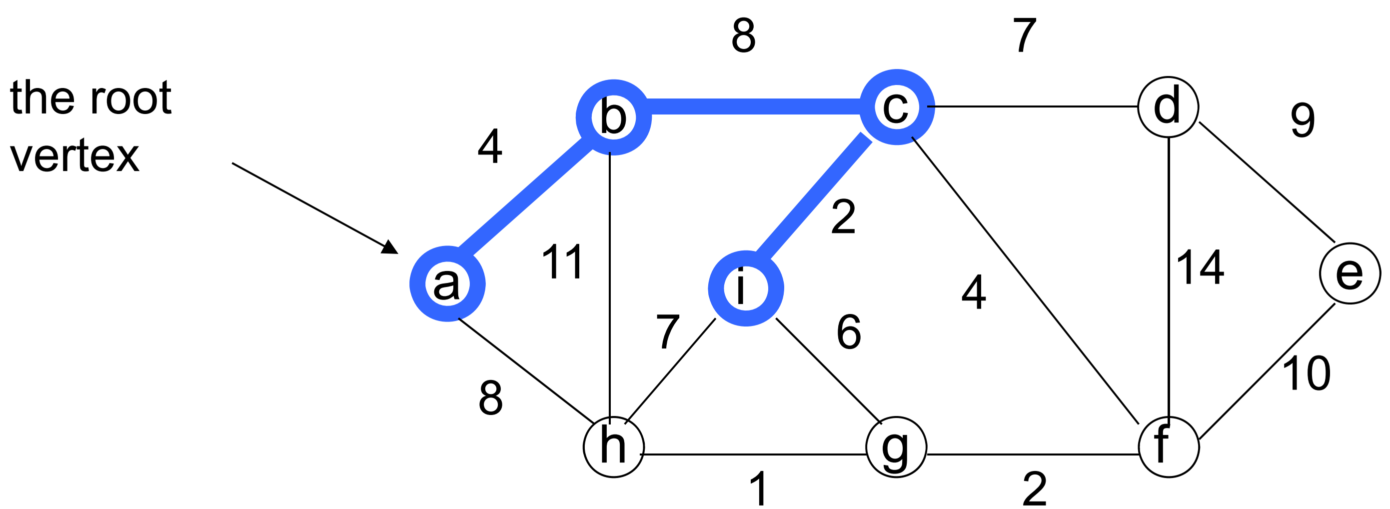
V	a	b	c	d	e	f	g	h	i
T	1	1	**0**	0	0	0	0	0	0
Key	0	4	**8**	-	-	-	-	8	-
π	-1	a	**b**	-	-	-	-	a	-

Prim's Algorithm: Example



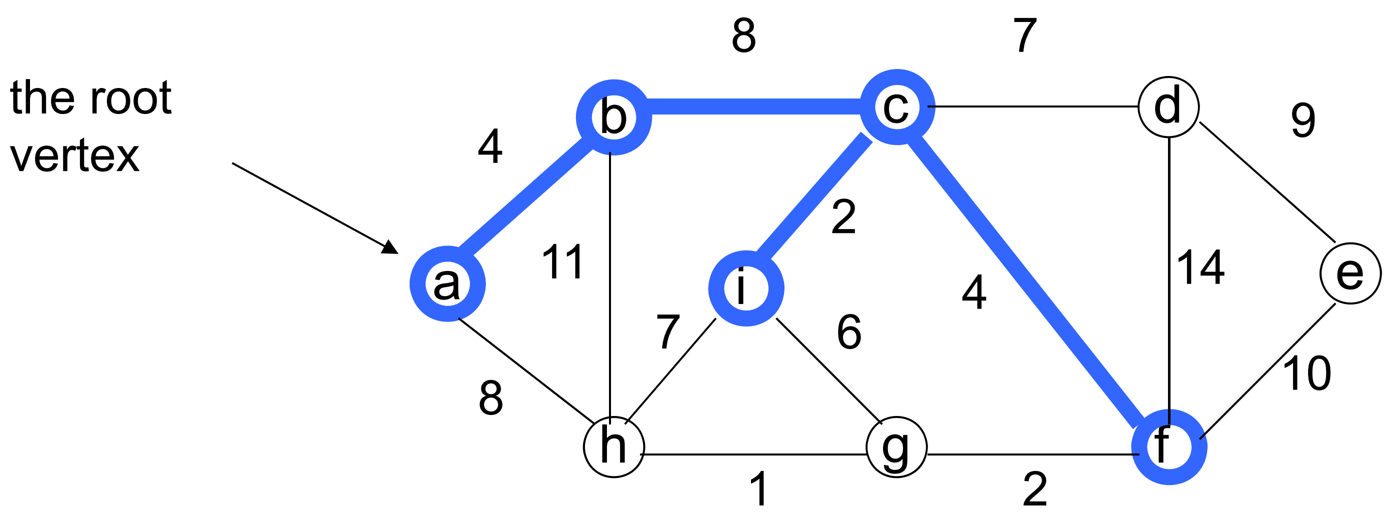
V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	0	0	0	**0**
Key	0	4	8	7	-	4	-	8	**2**
π	-1	a	b	c	-	c	-	a	**c**

Prim's Algorithm: Example



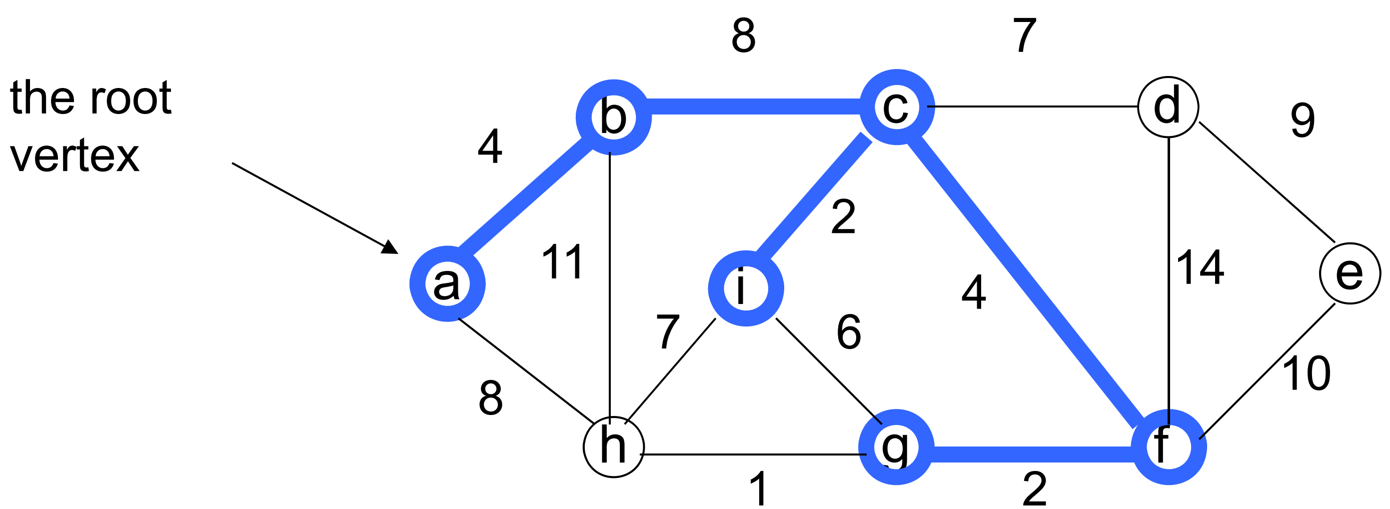
V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	**0**	0	0	1
Key	0	4	8	7	-	**4**	6	7	2
π	-1	a	b	c	-	**c**	i	i	c

Prim's Algorithm: Example



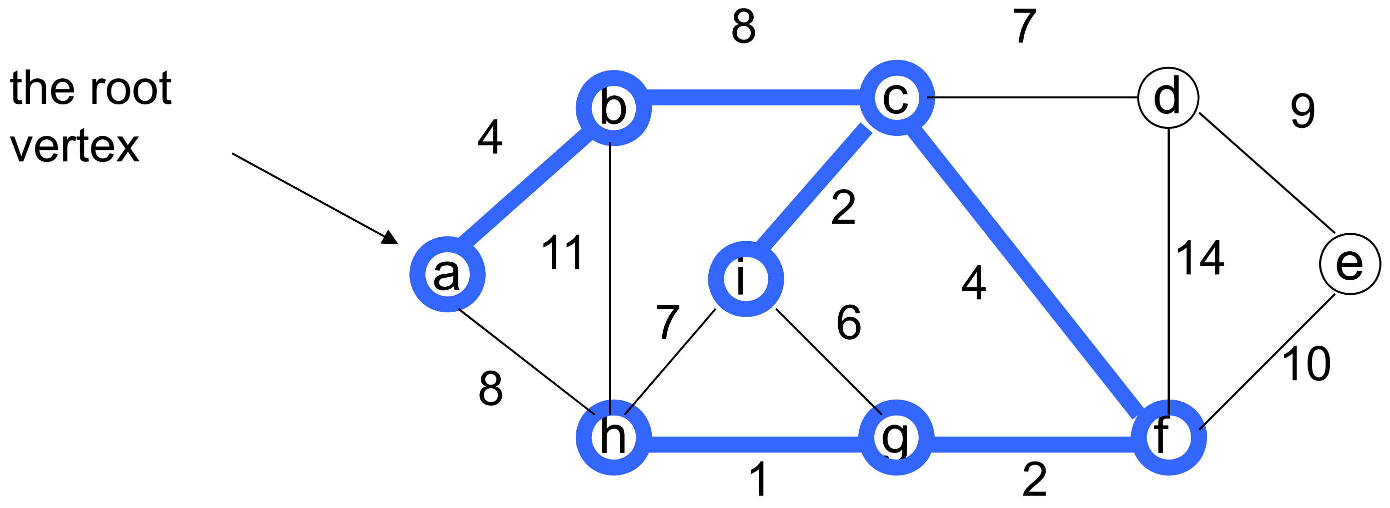
V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	**1**	**0**	0	1
Key	0	4	8	7	10	**4**	**2**	7	2
π	-1	a	b	c	f	**c**	**f**	i	c

Prim's Algorithm: Example



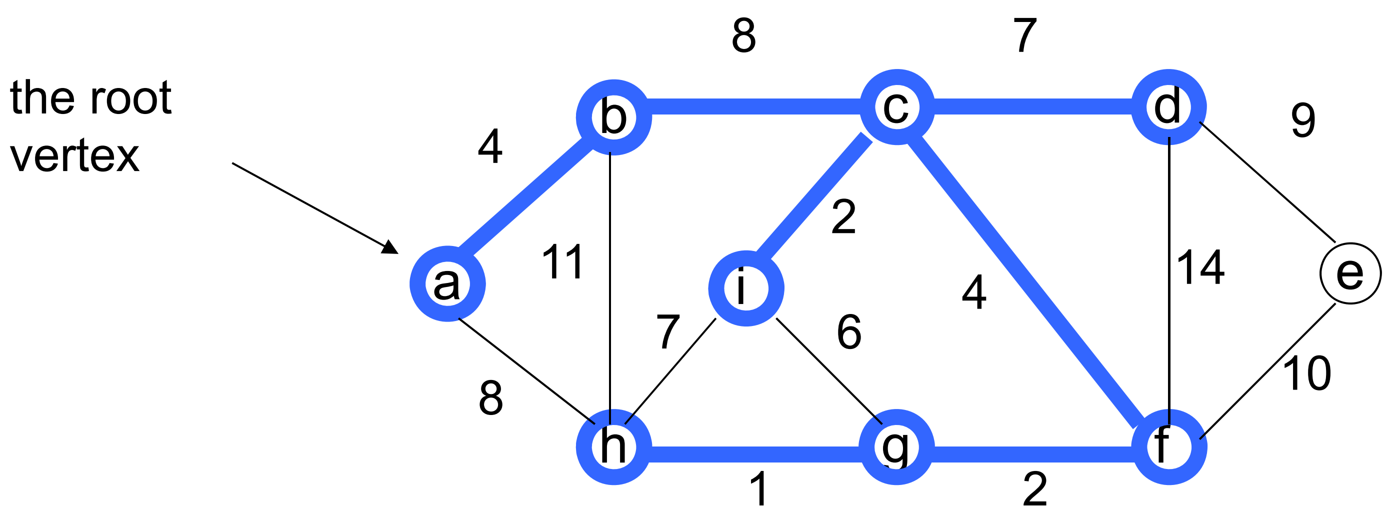
V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	1	**1**	**0**	1
Key	0	4	8	7	10	4	**2**	**1**	2
π	-1	a	b	c	f	c	**f**	**g**	c

Prim's Algorithm: Example



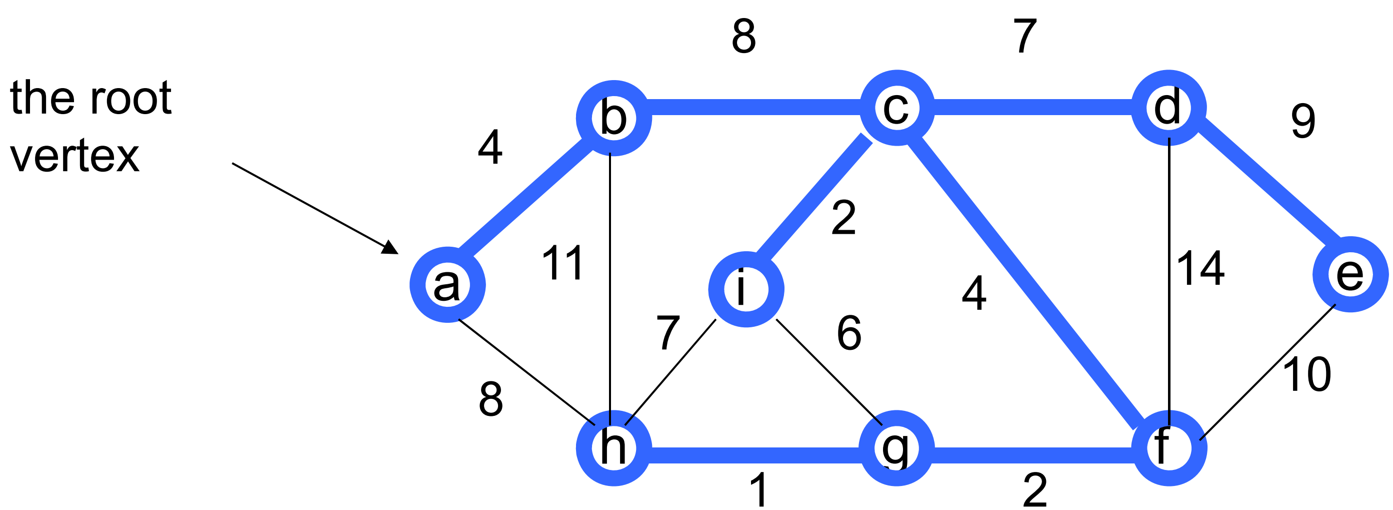
V	a	b	c	d	e	f	g	h	i
T	1	1	1	**0**	0	1	1	**1**	1
Key	0	4	8	**7**	10	4	2	**1**	2
π	-1	a	b	**c**	f	c	f	**g**	c

Prim's Algorithm: Example



V	a	b	c	d	e	f	g	h	i
T	1	1	1	**1**	**0**	1	1	1	1
Key	0	4	8	**7**	**9**	4	2	1	2
π	-1	a	b	**c**	**d**	c	f	g	c

Prim's Algorithm: Example



V	a	b	c	d	e	f	g	h	i
T	1	1	1	1	**1**	1	1	1	1
Key	0	4	8	7	**9**	4	2	1	2
π	-1	a	b	c	**d**	c	f	g	c

Complexity: Prim Algorithm

```
MST-Prim(G,w,r)
Q ← V[G]  // Q – vertices out of T
for each u ∈ Q                                    O(V)
   key[u] ← ∞
key[r] ← 0
π[r] ← NIL
while Q ≠ ∅ do                                    O(V)
  u ← ExtractMin(Q)  // making u part of T        Heap: O(lgV)
     for each v ∈ Adj[u] do                       Overall: O(E)
        if v ∈ Q and w(u,v) < key[v] then
           π[v] ← u
           key[v] ← w(u,v)                        Decrease Key: O(lgV)
```

Overall complexity: O($\boldsymbol{V}$)+O($\boldsymbol{V}$ lg $\boldsymbol{V}$+$\boldsymbol{E}$ lg $\boldsymbol{V}$) = O($\boldsymbol{E}$ lg $\boldsymbol{V}$)

Overall Complexity Analysis

$O(n^2)$

- When we don't use heap
- To find the minimum element, we traverse the "KEY" array from beginning to end
- We use adjacency matrix to update KEY.

$O(E\log V)$

- When min-heap is used to find the minimum element from "KEY".