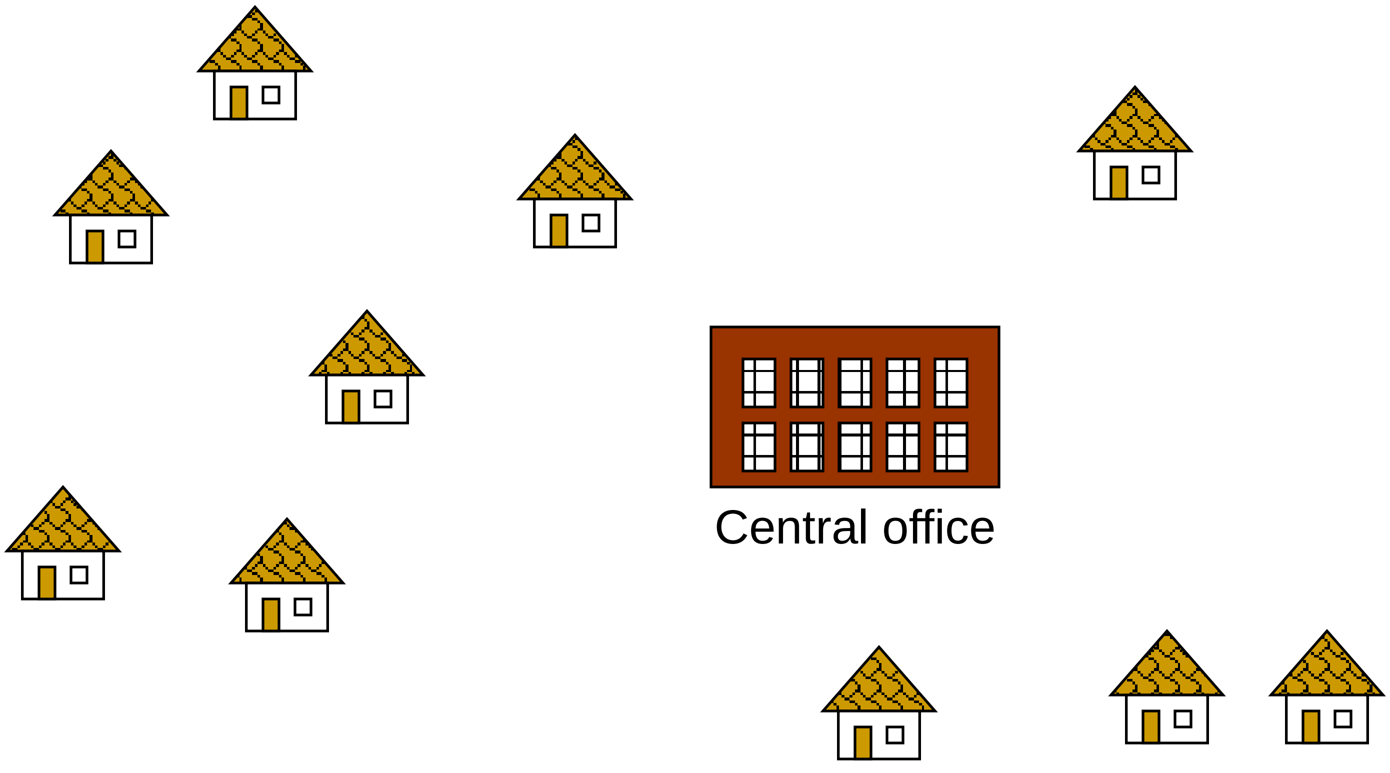
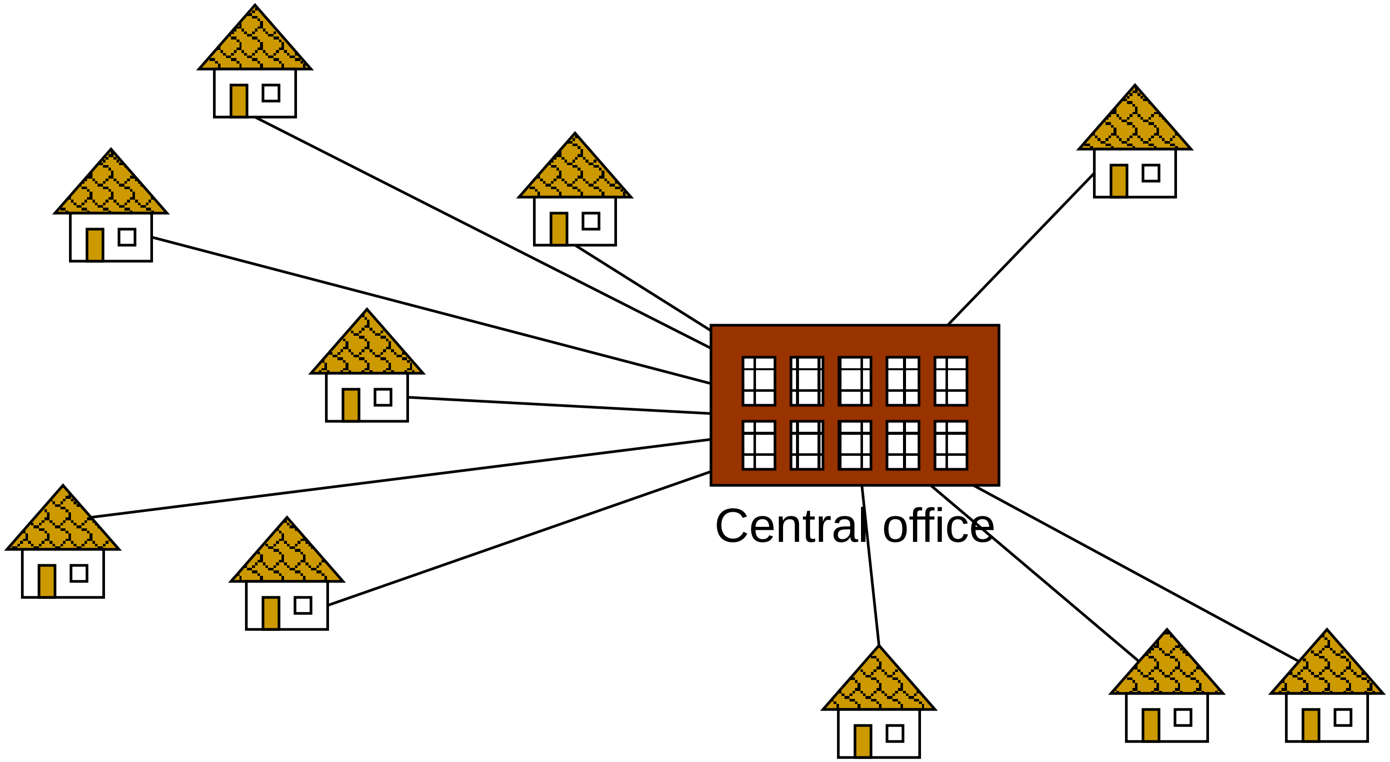
CSE 301
Combinatorial Optimization

Minimum Spanning Tree

Problem: Laying Telephone Wire

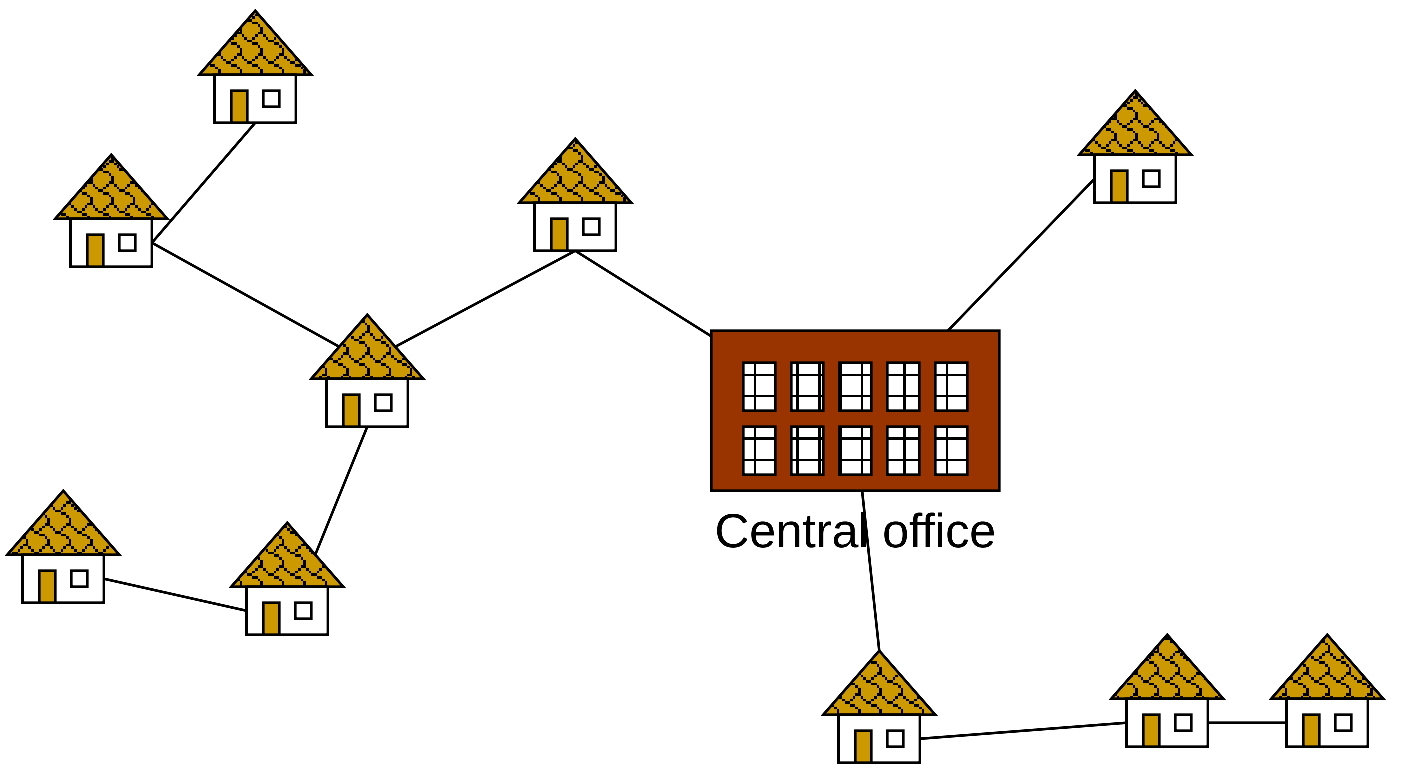


Wiring: Naïve Approach



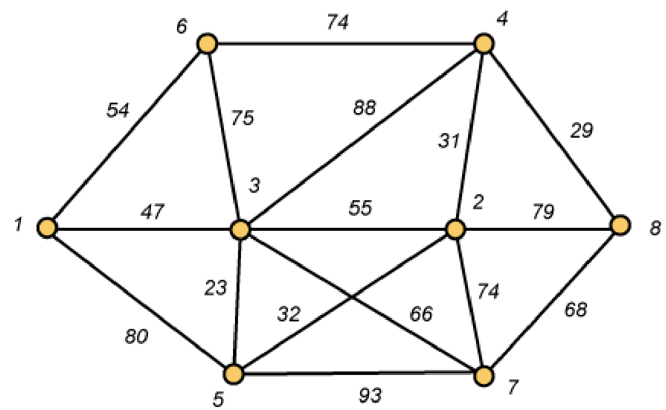
Expensive!

Wiring: Better Approach



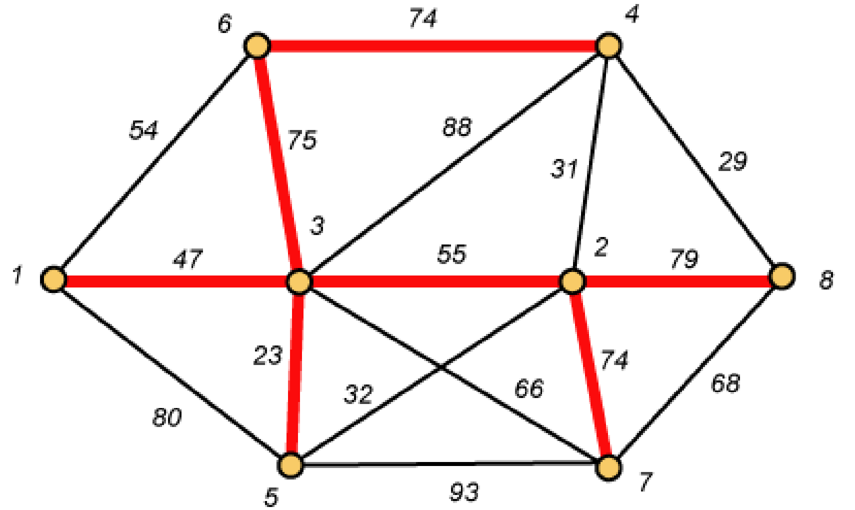
Minimize the total length of wire connecting the customers

A Networking Problem



Problem: The vertices represent 8 regional data centers which need to be connected with high-speed data lines. Feasibility studies show that the links illustrated above are possible, and the cost in millions of dollars is shown next to the link. Which links should be constructed to enable full communication (with relays allowed) and keep the total cost minimal.

Links Will Form a Spanning Tree

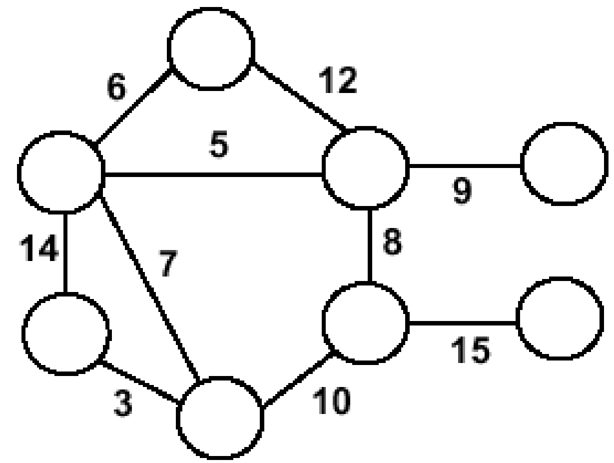


Cost (T) = 47 + 23 + 75 + 74 + 55 + 74 + 79

= 427

Minimum Spanning Trees

- Undirected, connected graph $G = (V,E)$
- Weight function $W: E \rightarrow R$ (assigning cost or length or other values to edges)



- Spanning tree: tree that connects all the vertices (above?)
- Minimum spanning tree: tree that connects all the vertices and minimizes

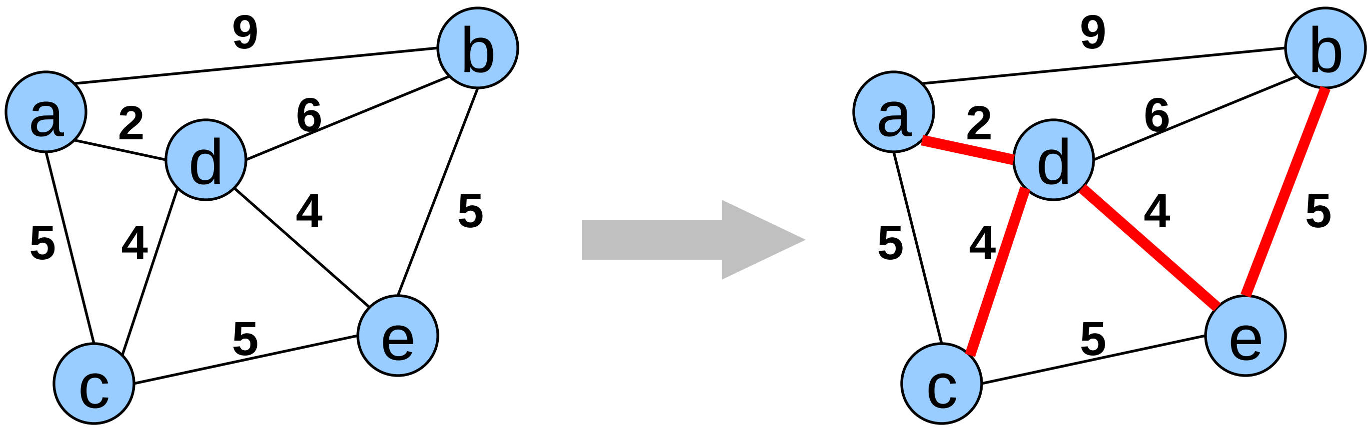
$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

Minimum Spanning Tree (MST)

A **minimum spanning tree** is a subgraph of an undirected weighted graph $\boldsymbol{G}$, such that

- it is a tree (i.e., it is acyclic)
- it covers all the vertices $\boldsymbol{V}$
  - contains $|\boldsymbol{V}|$ **-** $\boldsymbol{1}$ edges
- the total cost associated with tree edges is the minimum among all possible spanning trees
- not necessarily unique

How Can We Generate a MST?



Greedy Choice

We will show two ways to build a minimum spanning tree.

- A MST can be grown from the current spanning tree by adding the nearest vertex and the edge connecting the nearest vertex to the MST. (Prim's algorithm)

- A MST can be grown from a forest of spanning trees by adding the smallest edge connecting two spanning trees. (Kruskal's algorithm)

Notation

- Tree-vertices: in the tree constructed so far
- Non-tree vertices: rest of vertices

Prim’s Selection rule

- Select the minimum weight edge between a tree-node and a non-tree node and add to the tree

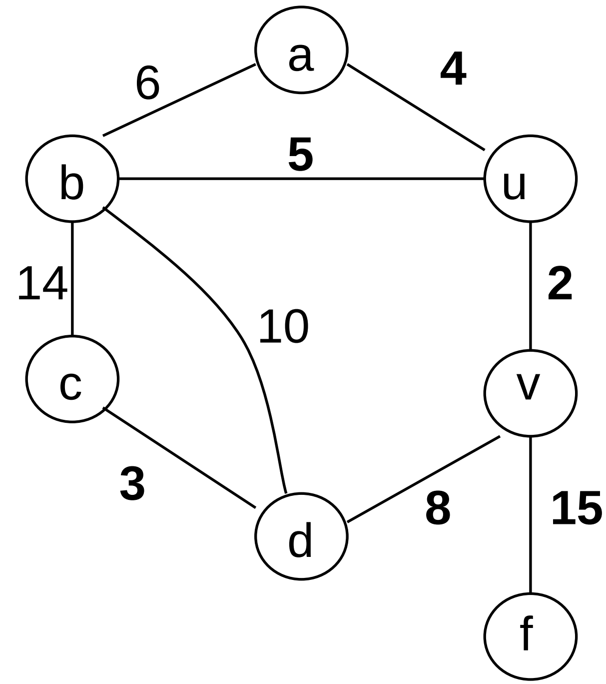
The Prim algorithm Main Idea

Select a vertex to be a tree-node

while (there are non-tree vertices) {
  if there is no edge connecting a tree node with a non-tree node
    return "no spanning tree"

  select an edge of minimum weight between a tree node and a non-tree node

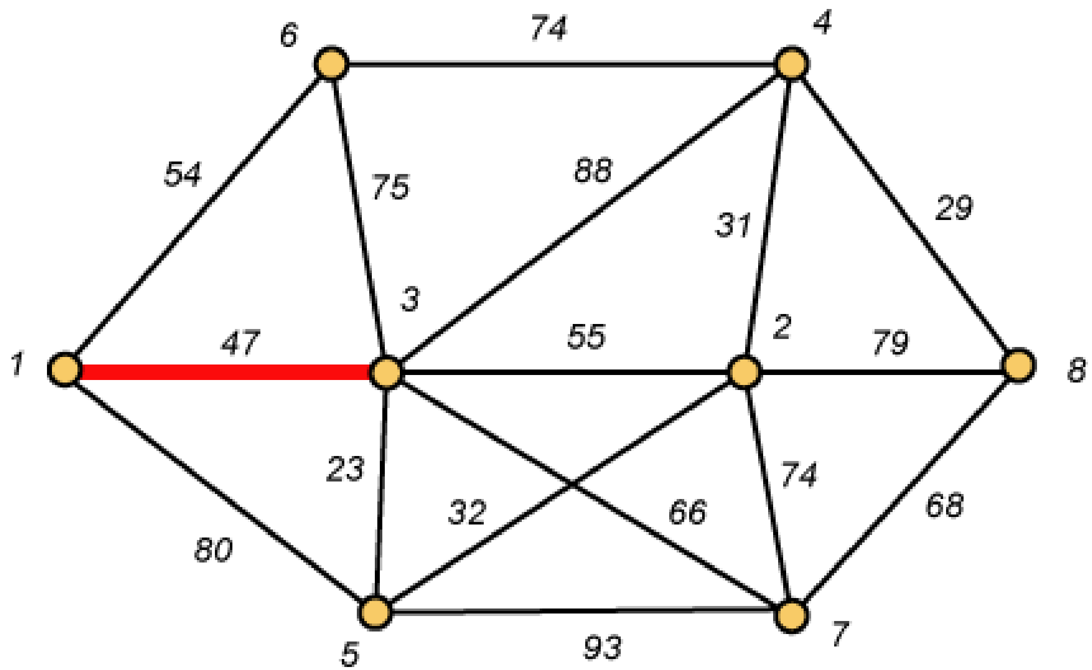
  add the selected edge and its new vertex to the tree
}
return tree



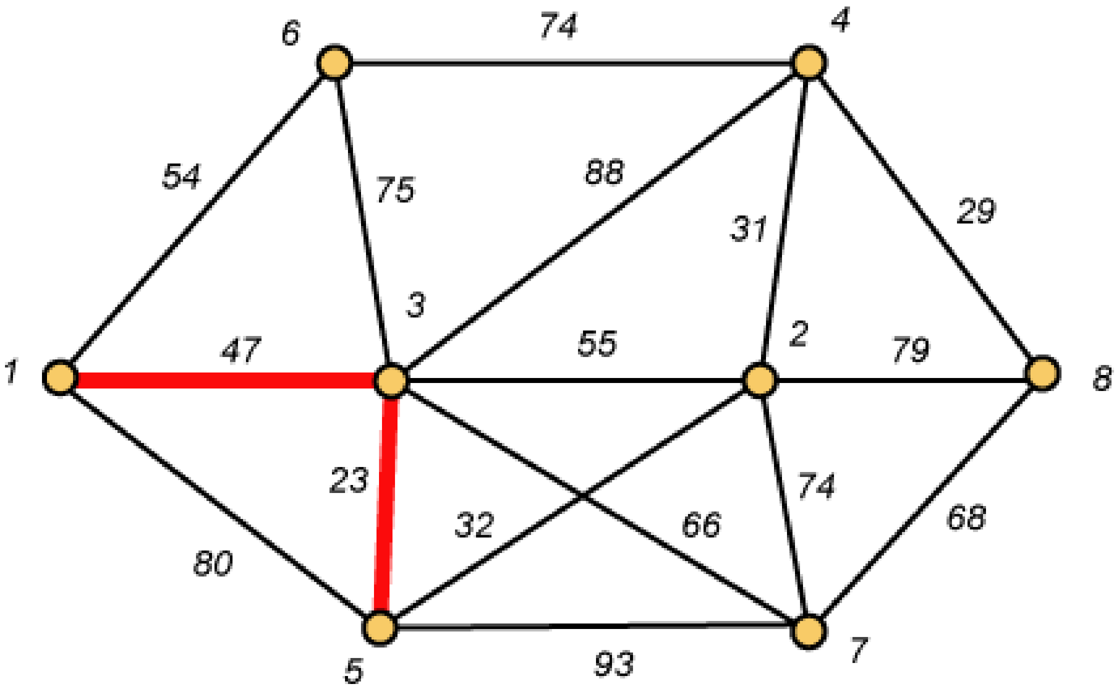
Prim’s Algorithm

- Vertex based algorithm
- Grows one tree T, **one vertex at a time**

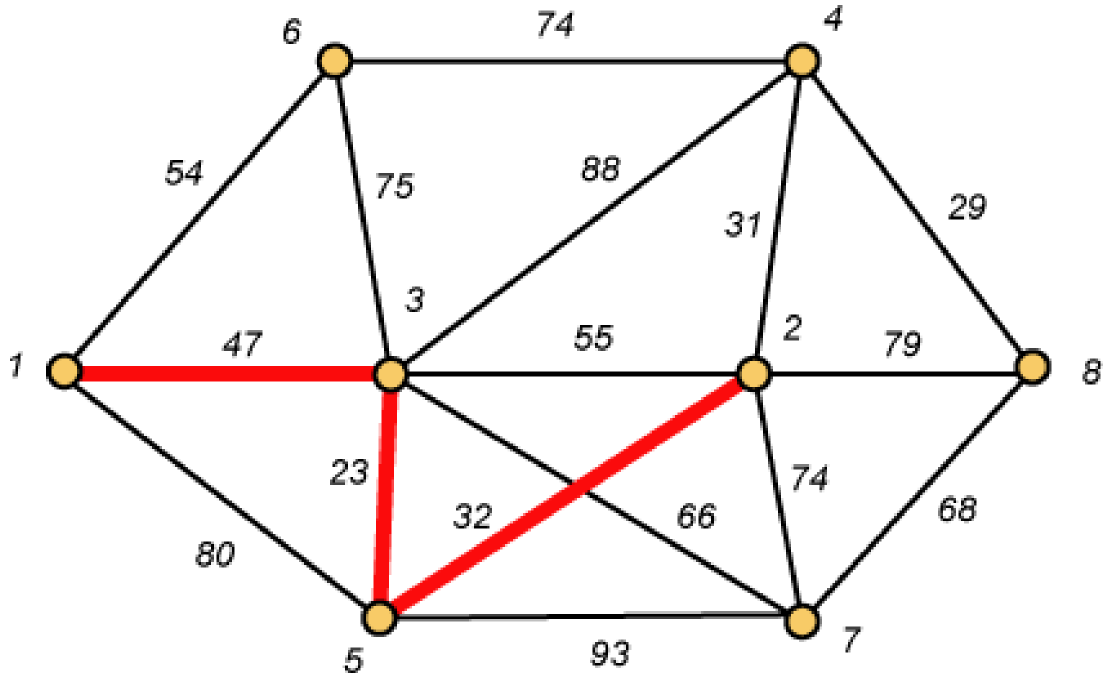
Prim – Step 1



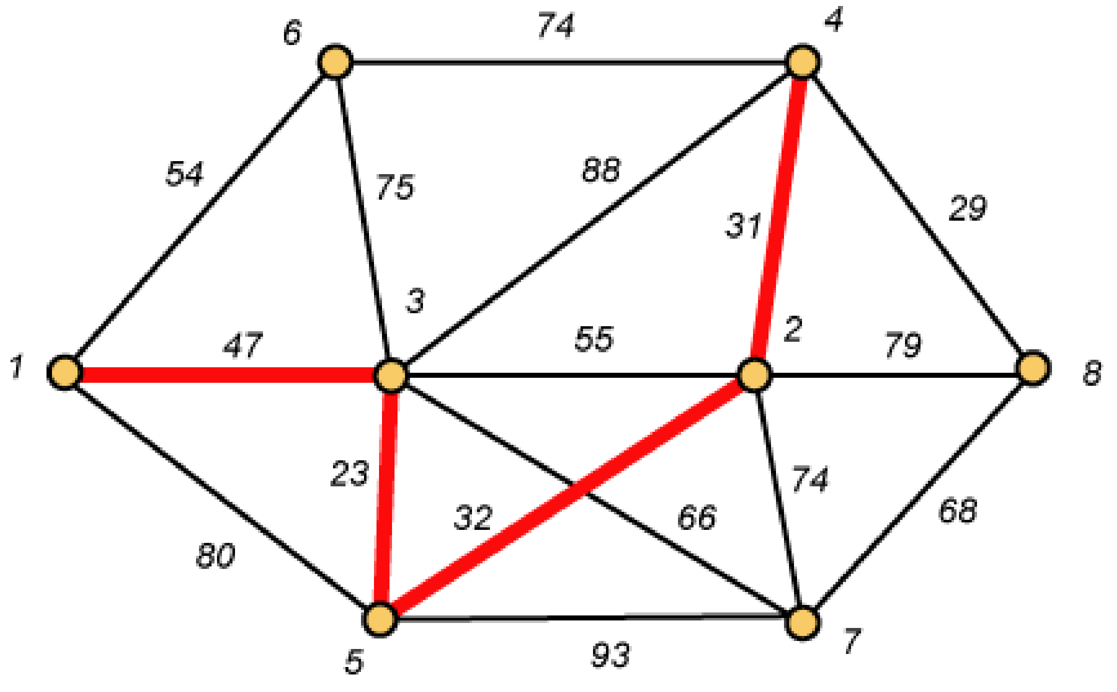
Prim – Step 2



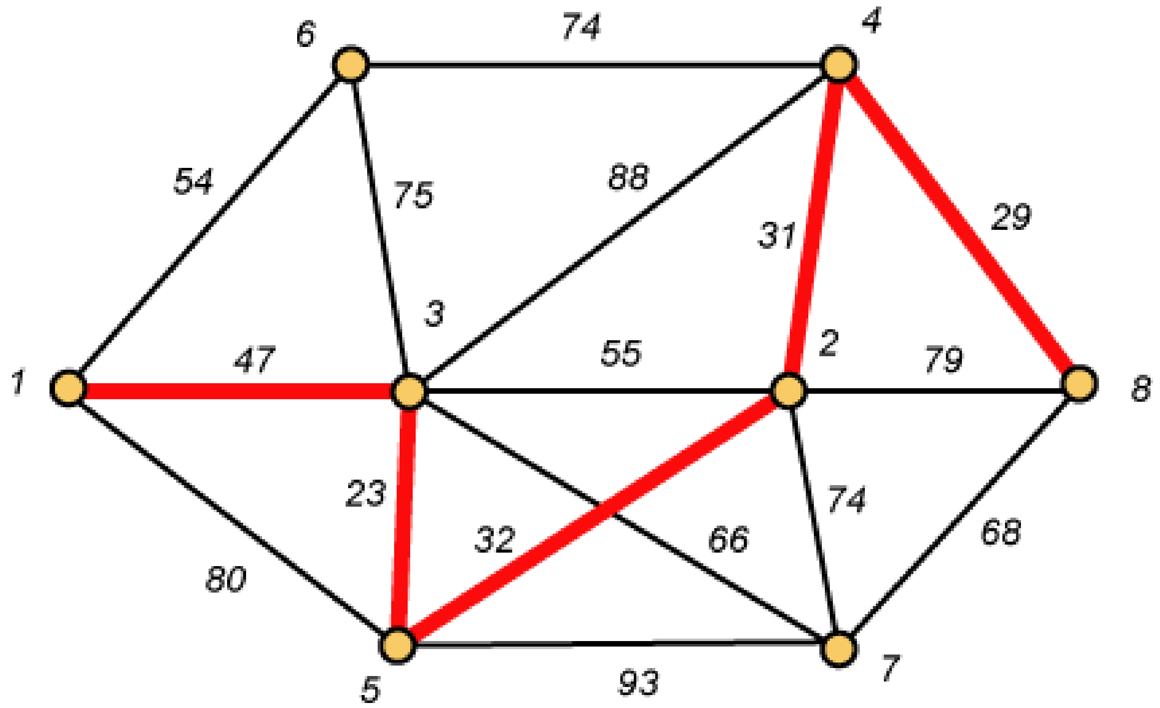
Prim – Step 3



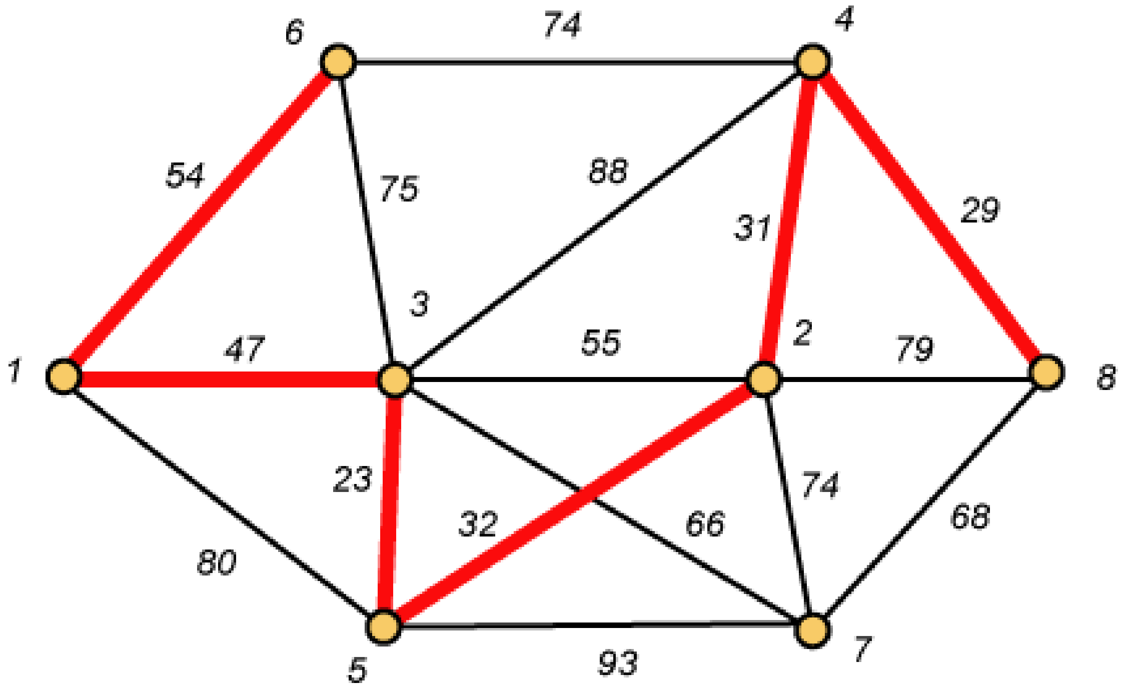
Prim – Step 4



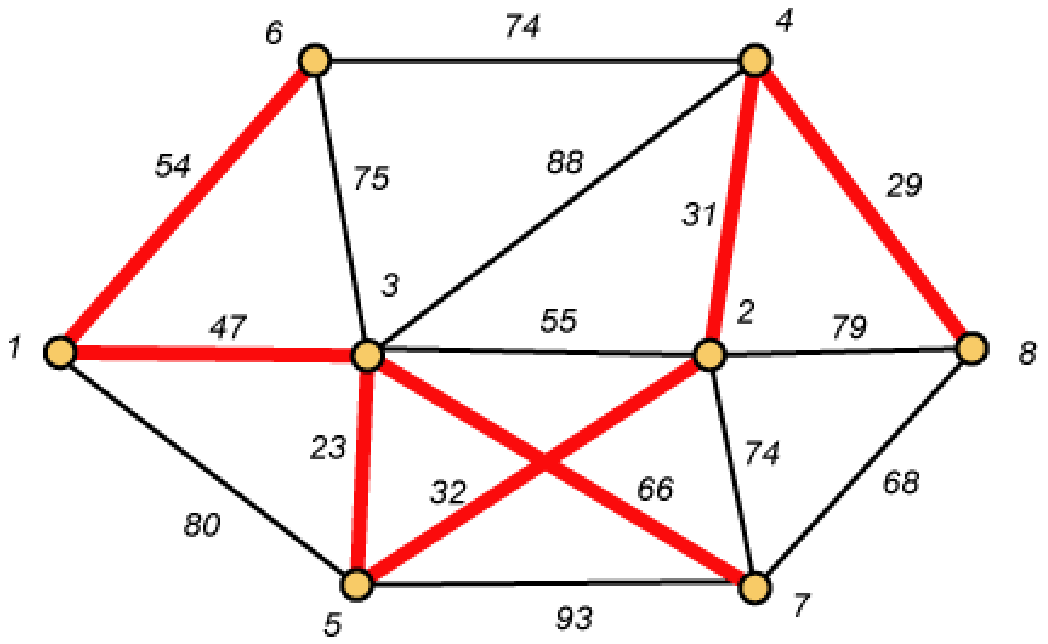
Prim – Step 5



Prim – Step 6



Prim – Step 7 *Done!!*



Weight (T) = 23 + 29 + 31 + 32 + 47 + 54 + 66 = **282**

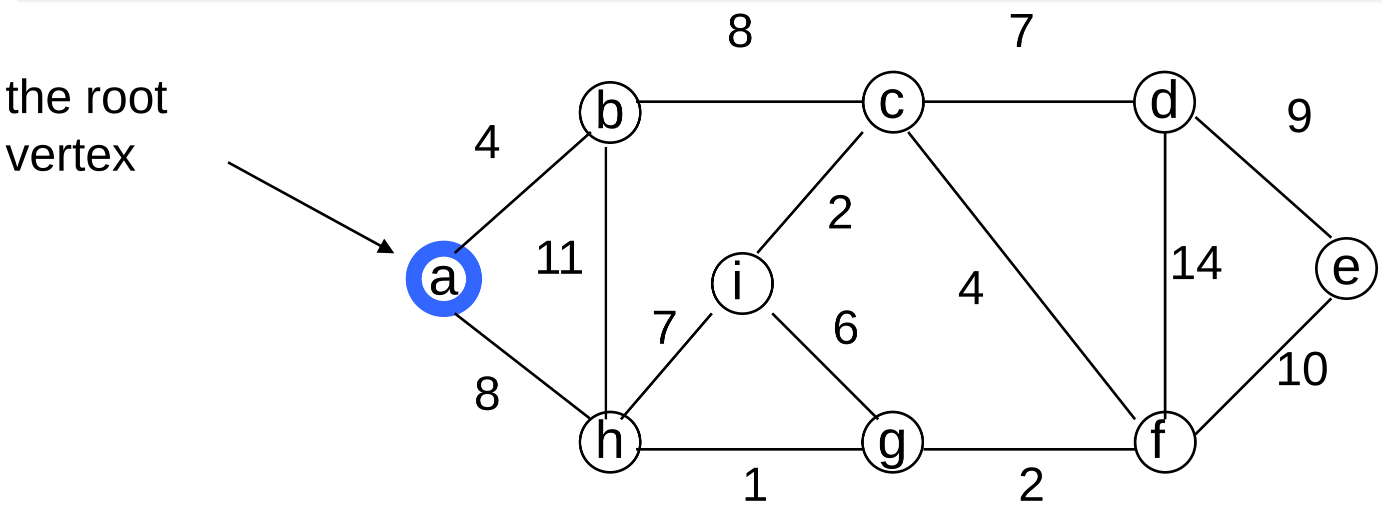
Prim Algorithm (2)

```
MST-Prim(G,w,r)
Q ← V[G]  // Q - vertices out of T
for each u ∈ Q
   key[u] ← ∞
key[r] ← 0       // r is the first tree node, let r=1
π[r] ← NIL
while Q ≠ ∅ do
  u ← ExtractMin(Q)  // making u part of T
     for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
           π[v] ← u
           key[v] ← w(u,v)
```

Prim Algorithm:Variables

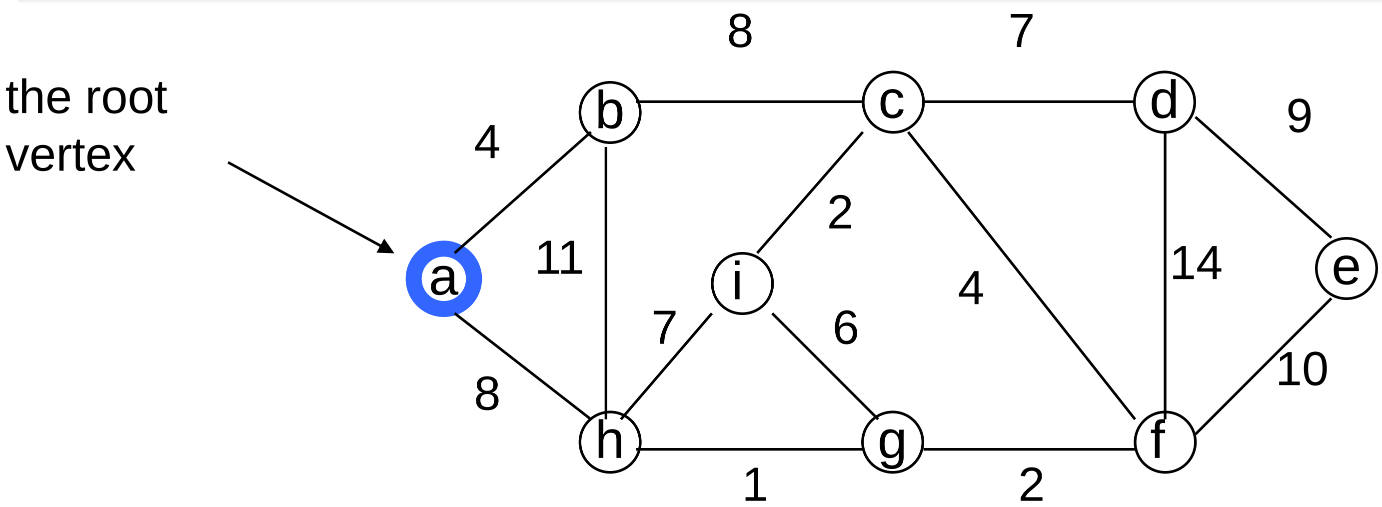
- r:
  - Grow the minimum spanning tree from the **root vertex "r"**.
- Q:
  - is a priority queue, holding all vertices that are not in the tree now.
- key[v]:
  - is the minimum weight of any edge connecting v to a vertex in the tree.
- π [v]:
  - names the parent of v in the tree.
- T[v] –
  - Vertex v is already included in MST if T[v]==1, otherwise, it is not included yet.

The execution of Prim's algorithm(moderate part)



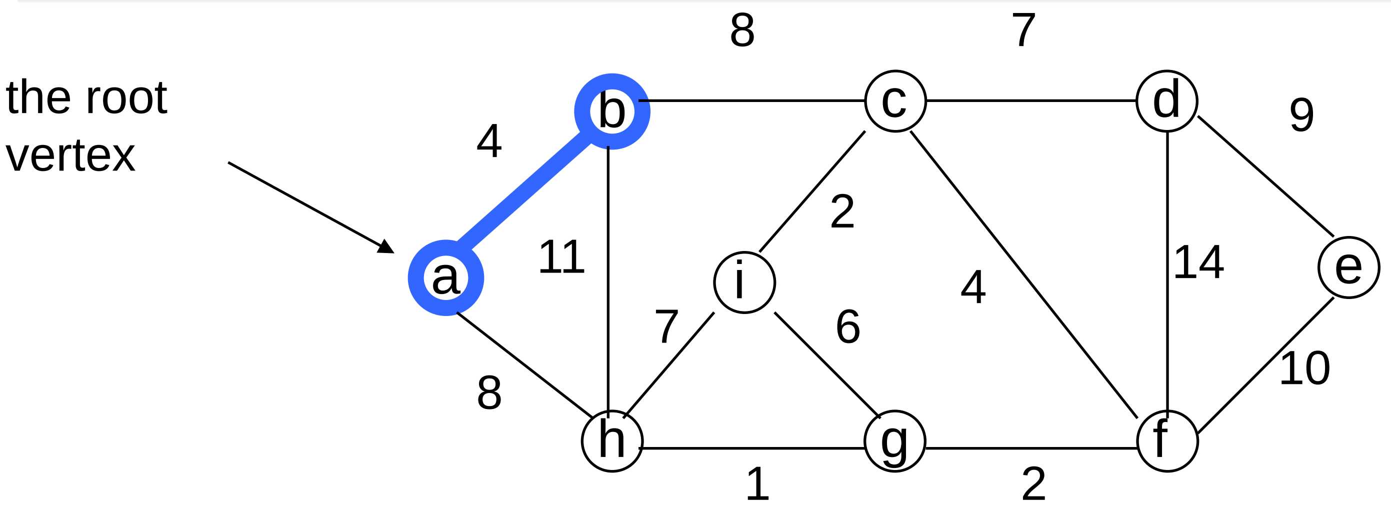
V	a	b	c	d	e	f	g	h	i
T	1	0	0	0	0	0	0	0	0
Key	0	-	-	-	-	-	-	-	-
π	-1	-	-	-	-	-	-	-	-

The execution of Prim's algorithm(moderate part)



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	-

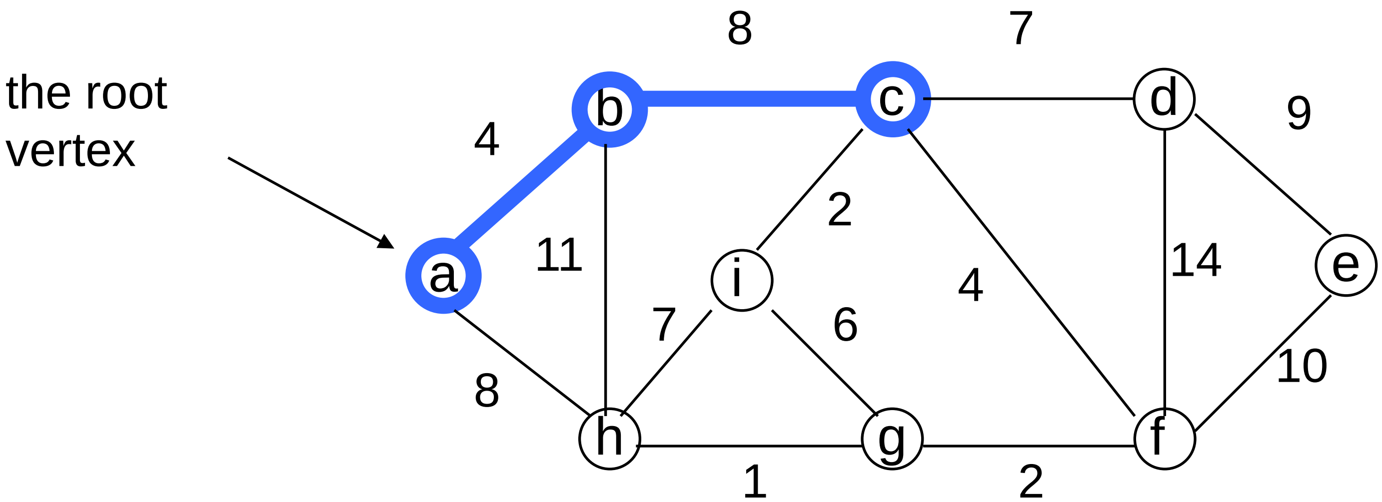
The execution of Prim's algorithm(moderate part)



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

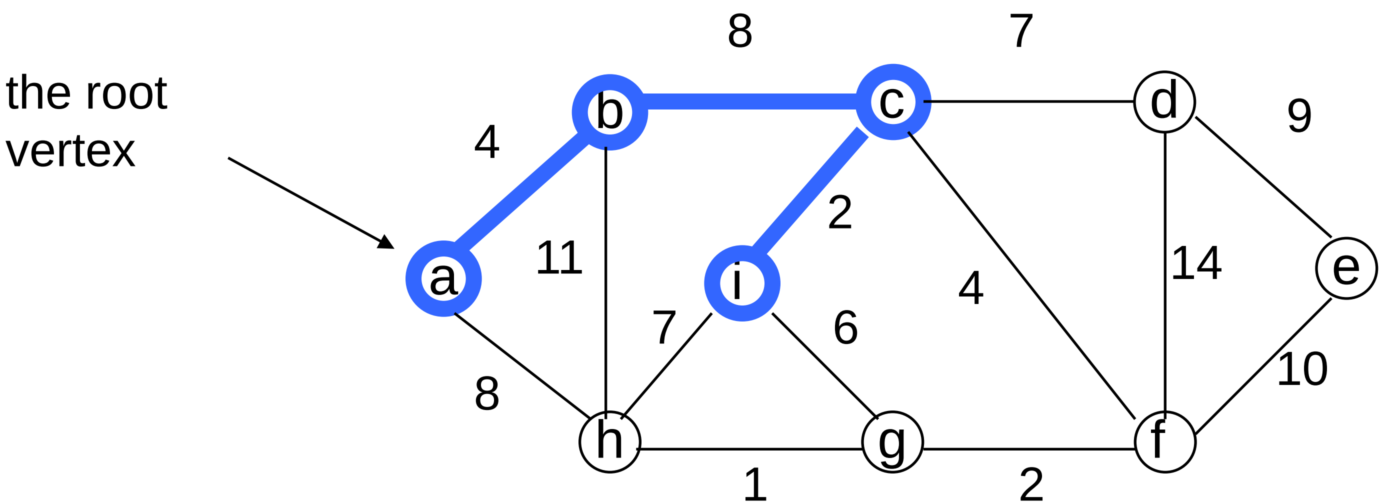
V	a	b	c	d	e	f	g	h	i
T	1	1	**0**	0	0	0	0	0	0
Key	0	4	**8**	-	-	-	-	8	-
π	-1	a	**b**	-	-	-	-	a	-

The execution of Prim's algorithm(moderate part)



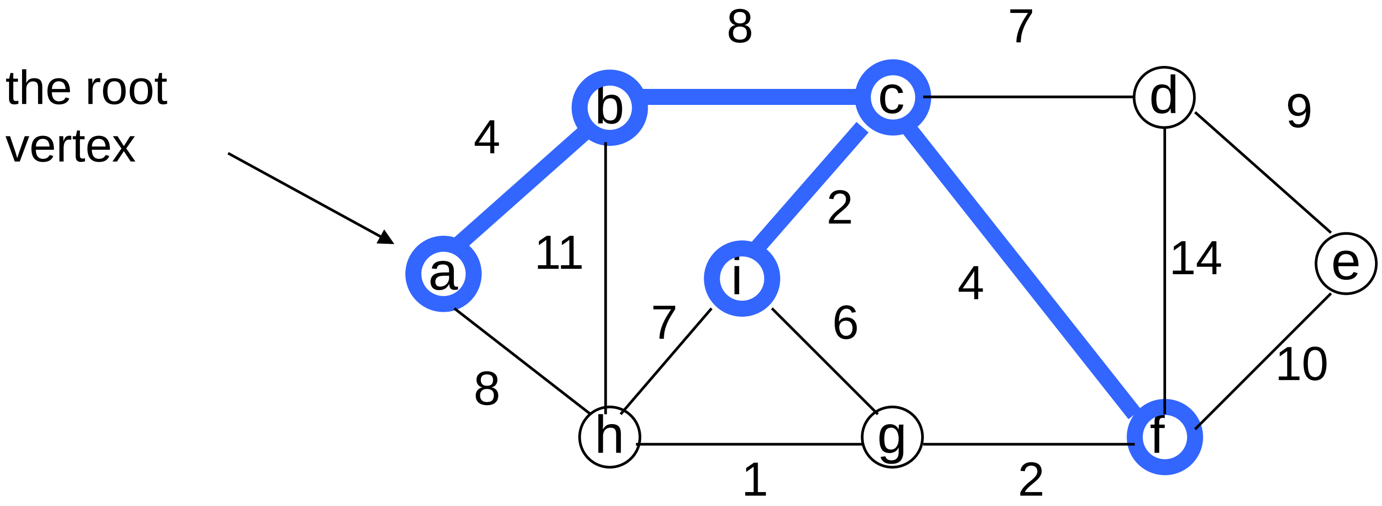
V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	0	0	0	**0**
Key	0	4	8	7	-	4	-	8	**2**
π	-1	a	b	c	-	c	-	a	**c**

The execution of Prim's algorithm(moderate part)



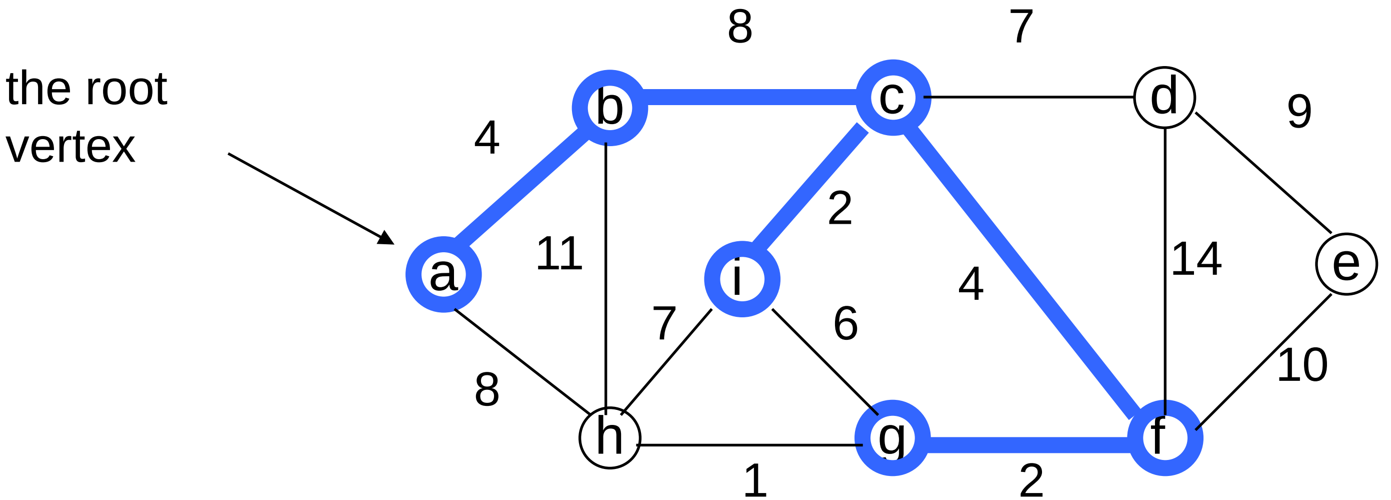
V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	**0**	0	0	1
Key	0	4	8	7	-	**4**	6	7	2
π	-1	a	b	c	-	**c**	i	i	c

The execution of Prim's algorithm(moderate part)



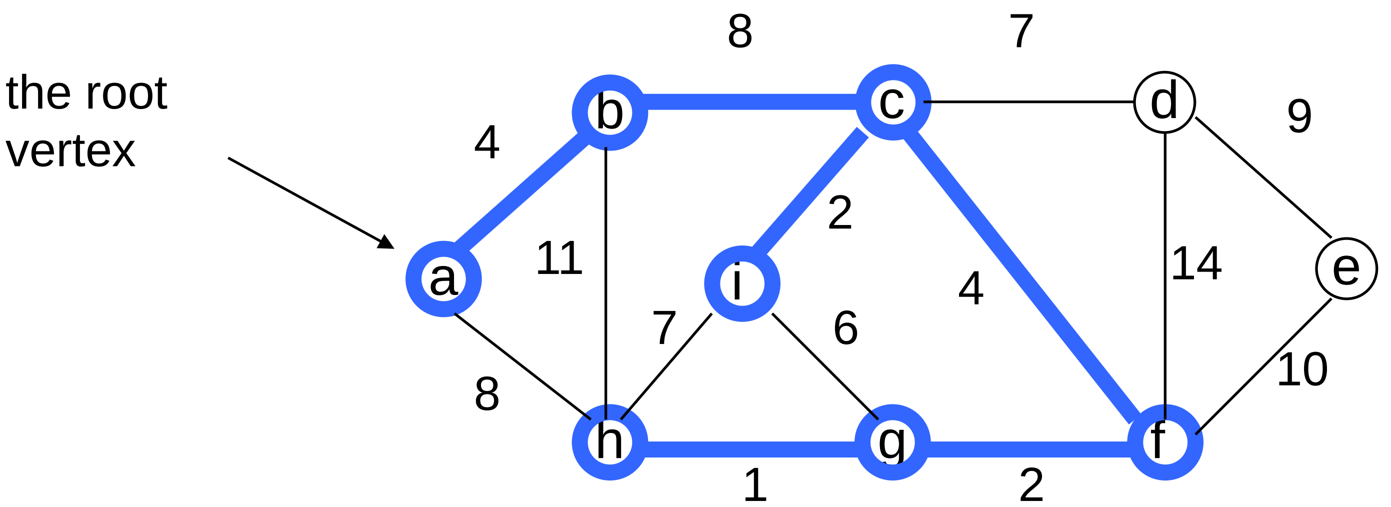
V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	**1**	**0**	0	1
Key	0	4	8	7	10	**4**	**2**	7	2
π	-1	a	b	c	f	**c**	**f**	i	c

The execution of Prim's algorithm(moderate part)



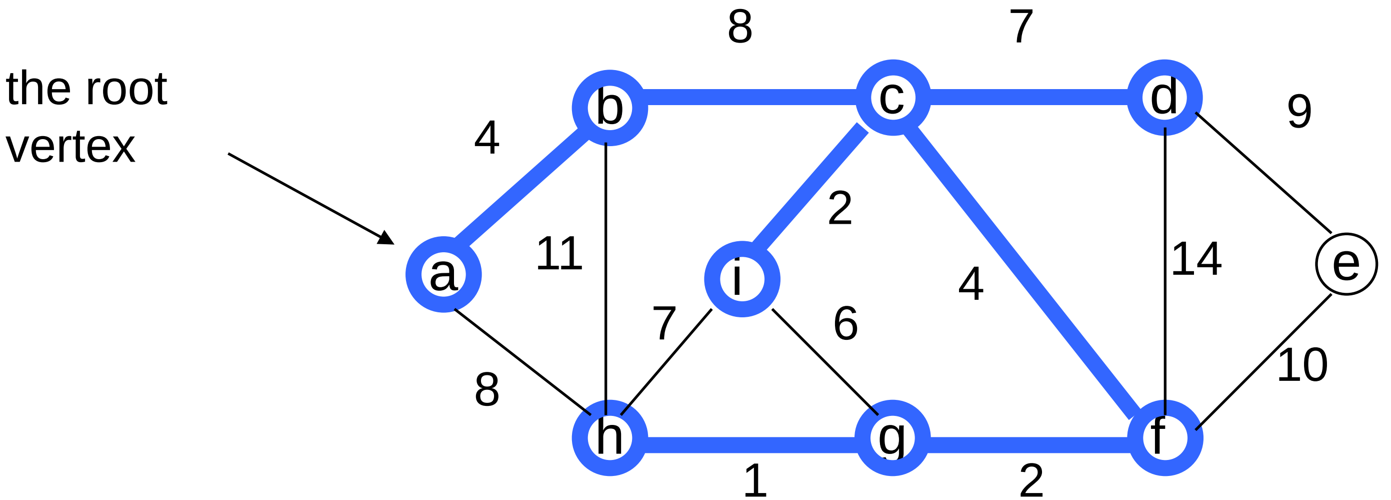
V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	1	**1**	**0**	1
Key	0	4	8	7	10	4	**2**	**1**	2
π	-1	a	b	c	f	c	**f**	**g**	c

The execution of Prim's algorithm(moderate part)



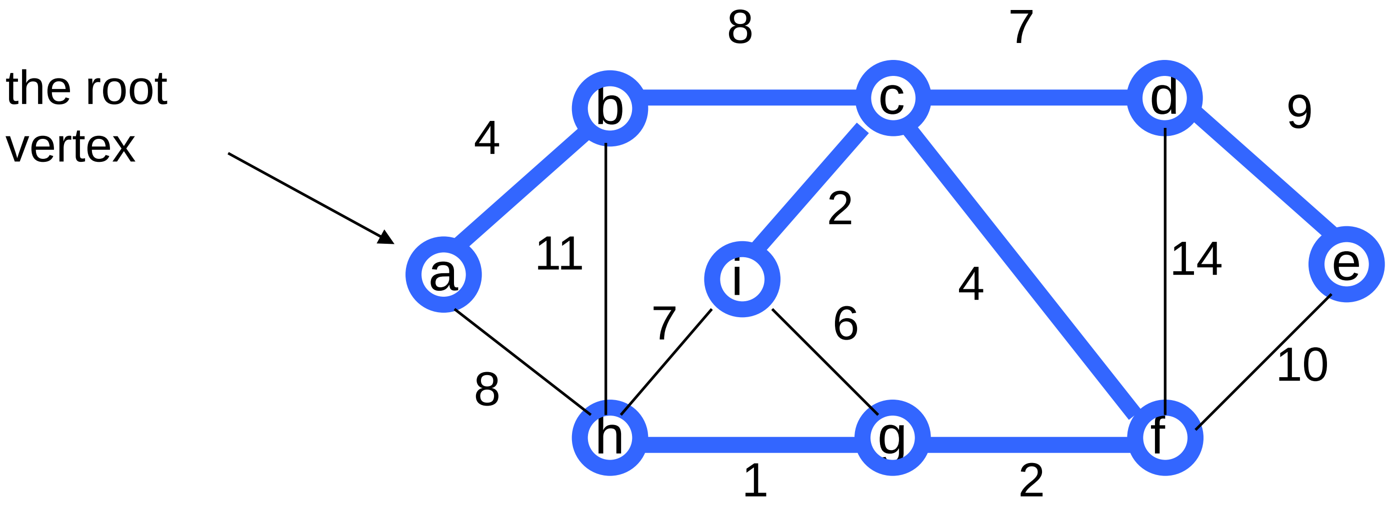
V	a	b	c	d	e	f	g	h	i
T	1	1	1	**0**	0	1	1	**1**	1
Key	0	4	8	**7**	10	4	2	**1**	2
π	-1	a	b	**c**	f	c	f	**g**	c

The execution of Prim's algorithm(moderate part)



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

The execution of Prim's algorithm(moderate part)



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(***V***)+O(***V*** lg ***V***+***E*** lg ***V***) = O(***E*** lg ***V***)

Overall Complexity Analysis

- $O(V^2)$
  - 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.
- $O(ElogV)$
  - When min-heap is used to find the minimum element from “KEY”.
- $O(E+VlogV)$
  - When fibonacci heap is used to find the minimum element from “KEY”.