Chapter 23
Minimum Spanning Trees

The slides for this course are based on the course textbook: Cormen, Leiserson, Rivest, and Stein, *Introduction to Algorithms*, 2nd edition, The MIT Press, McGraw-Hill, 2001.

• Many of the slides were provided by the publisher for use with the textbook. They are copyrighted, 2001.

• These slides are for classroom use only, and may be used only by students in this specific course this semester. They are NOT a substitute for reading the textbook!

Chapter 23 Topics

- Growing a minimum spanning tree
- Kruskal’s algorithm
- Prim’s algorithm

Directed Graphs

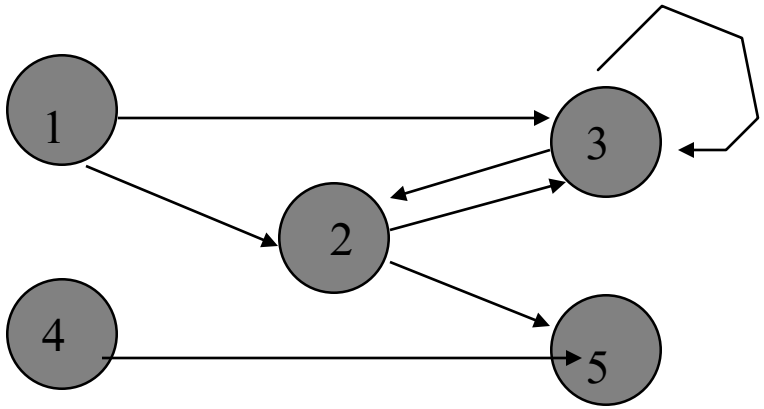
- A directed graph (or *digraph*) G is a pair (V,E), where V is a finite set and E is a binary relation on V.
  - V is the vertex set of the graph
    - $|V|$ is the number of vertices in the graph
  - E is the edge set of the graph
    - $|E|$ is the number of edges in the graph

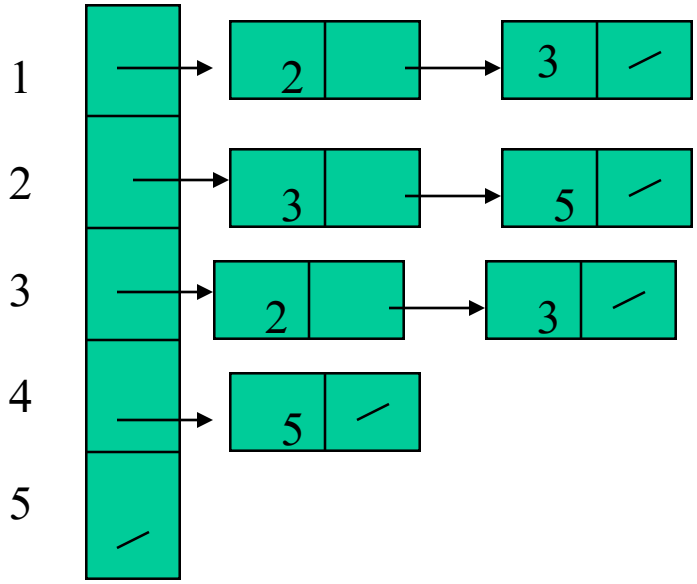
Undirected Graphs

- An undirected graph G is a pair (V,E)
  - V is the vertex set of the graph
  - E is the edge set of the graph
    - each edge is an unordered pair of vertices
  - Self loops are forbidden

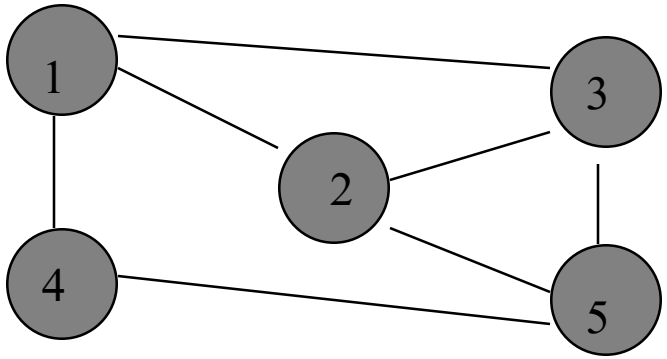
Representing a Graph

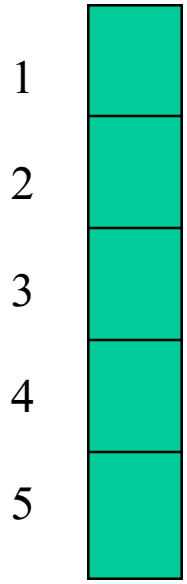
- Two standard representations
  - Adjacency lists
    - usually preferred
    - compact representation of sparse matrices
  - Adjacency matrix
    - may be preferred for a dense matrix
      - $|E|$ is close to $|V|^2$
    - or when we need to know if an edge exists between two vertices quickly

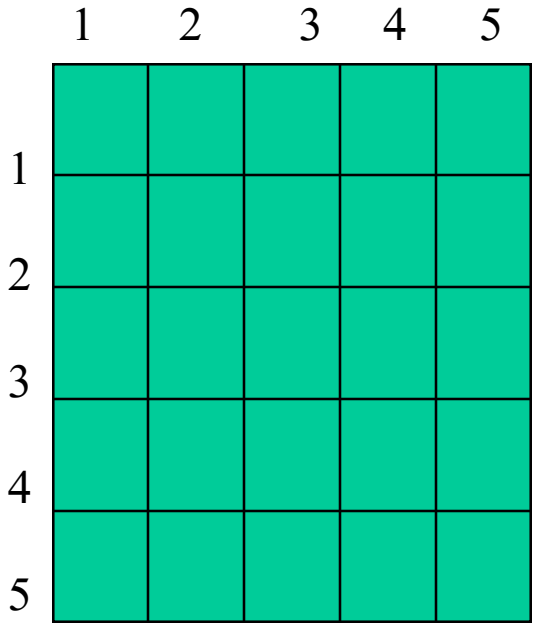




	1	2	3	4	5
1	0	1	1	0	0
2	0	0	1	0	1
3	0	1	1	0	0
4	0	0	0	0	1
5	0	0	0	0	0


1
3
2
4
5


1
2
3
4
5


1
2
3
4
5
1
2
3
4
5

Minimum Spanning Tree

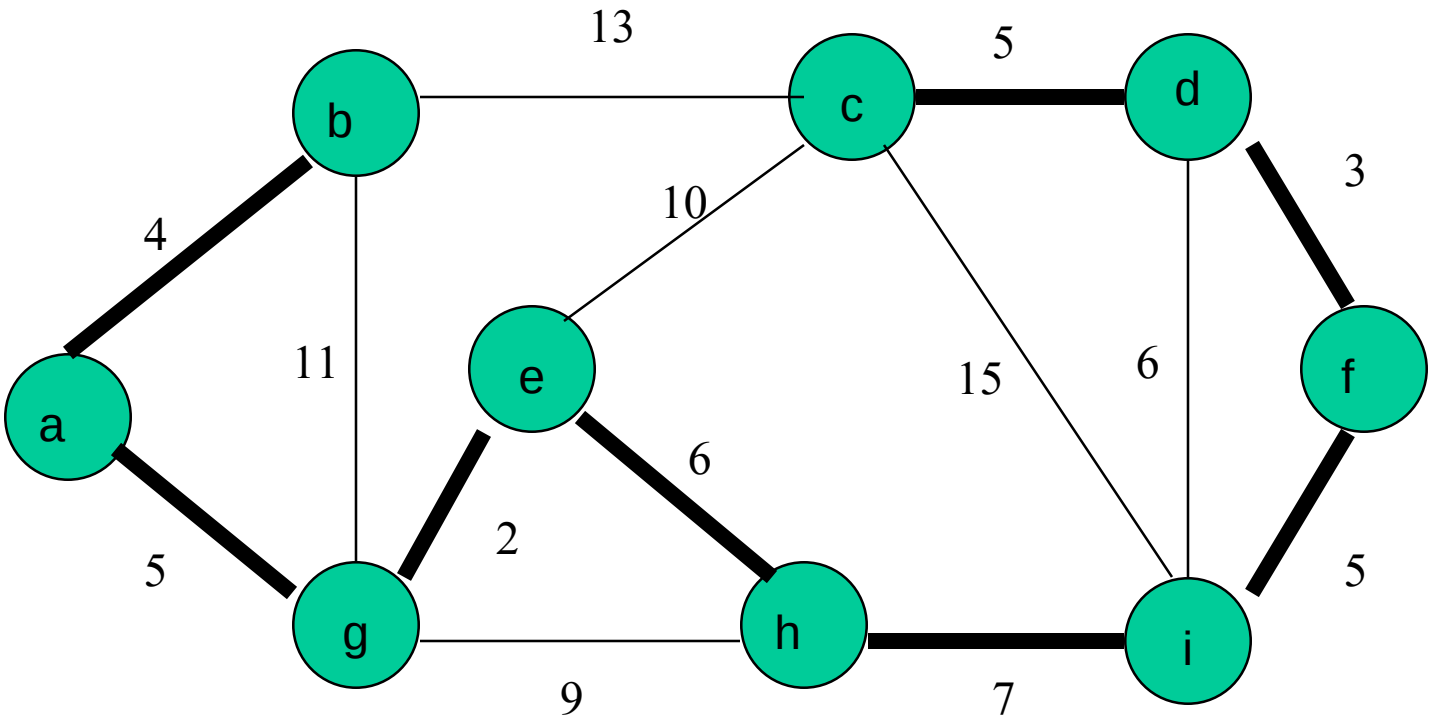
Given a connected, undirected graph $G = (V,E)$ where each edge has an associated weight (or cost or length),

- find a subset T of the edges of G such that:
  - all the nodes remain connected when only the edges in T are used, and
  - the sum of the cost of the edges is as small as possible.

Minimum Spanning Tree

- Assume a connected, undirected graph $G = (V, E)$
- with weight w(u,v) on each edge $(u,v) \in E$
- Find $T \subseteq E$ such that:
  - T connects all vertices (T is a **spanning** tree)
  - $w(T) = \sum_{(u,v) \in T} w(u,v)$ is minimized

Minimum Spanning Tree



Minimum Spanning Tree

- A spanning tree whose weight is minimum over all spanning trees is called a **minimum spanning tree**, or **MST**.
- Some properties of an MST:
  - It has $|V|$-1 edges.
  - It has no cycles.
  - It might not be unique

Greedy Strategy

- We will develop a generic greedy strategy for finding a minimum spanning tree and then look at two different algorithms that implement this strategy.
- At each step:
  - manage a set A that is a subset of a minimum spanning tree
  - find a "safe" edge (u,v) to add to the set A so that A remains a subset an MST

Growing a Minimum Spanning Tree

- Assume a connected, undirected graph $G = (V, E)$.
- Assume a weight function w: $E \rightarrow R$.
- Consider greedy generic algorithm that **grows** minimum spanning tree one edge at a time.

Generic MST Algorithm

```
GENERIC-MST(G, w)
1   A ← ∅
2   while A does not form a spanning tree do
3       find an edge (u, v) that is safe for A
4       A ← A ∪ {(u, v)}
5   return A
```

Terminology

- A **cut** (S, V-S) of an undirected graph $G = (V, E)$ is a partition of V.
  - Let $C = \{S_i\}$ be a collection of nonempty sets.

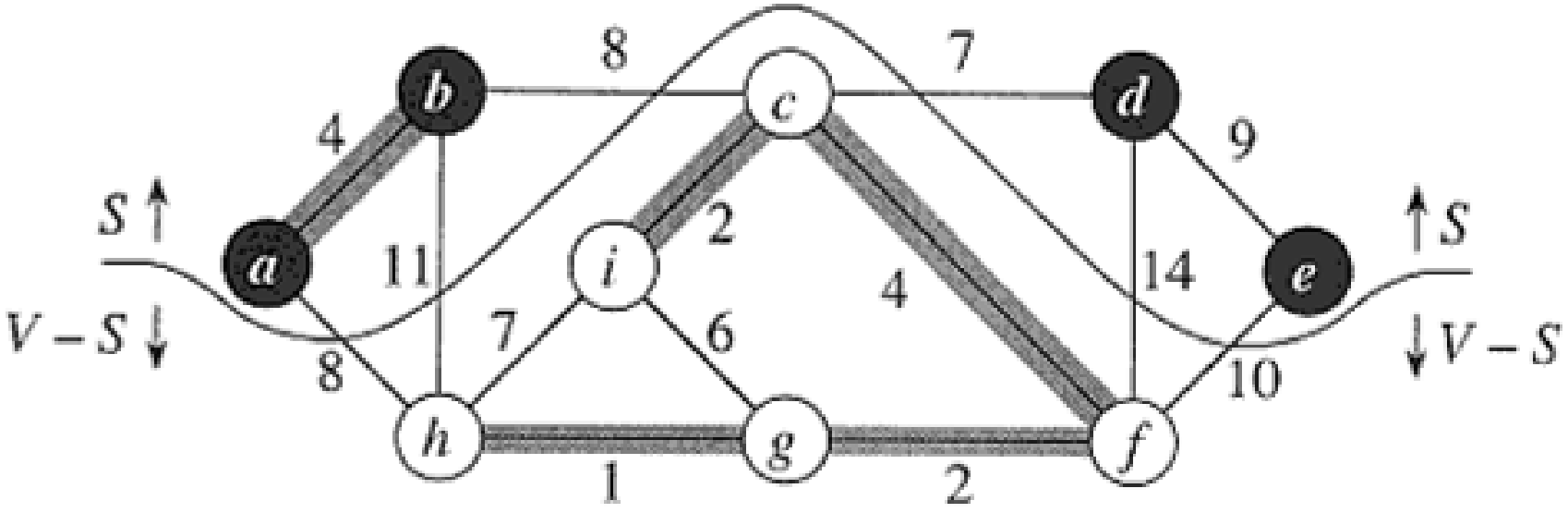
    C forms a **partition** of a set S if:
    - the sets are pairwise disjoint, and
    - their union is S.
  - i.e., each element of S appears in exactly one S_i (where $S_i \in C$).

Terminology (continued)

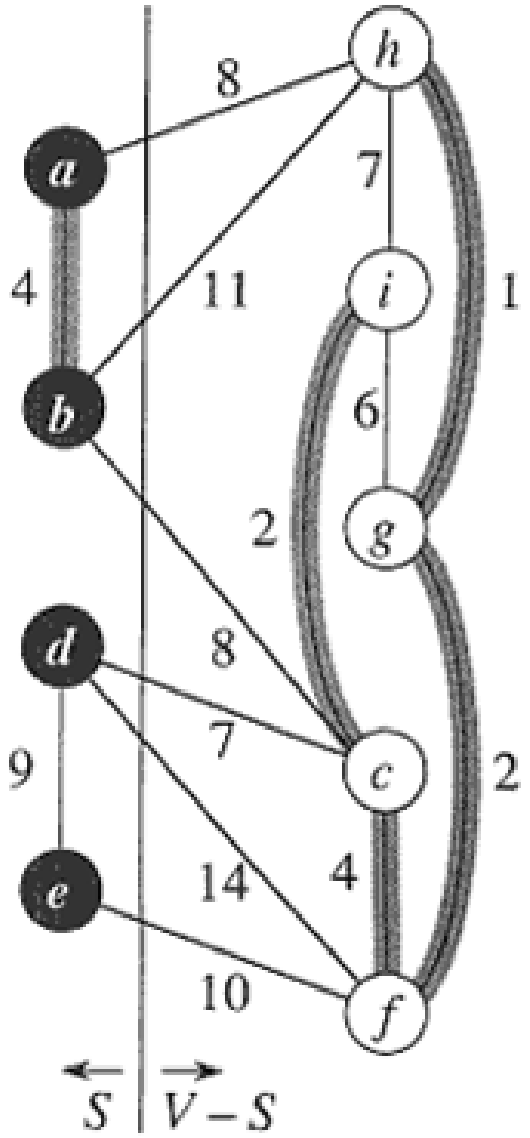
- Edge $(u, v) \in E$ **crosses** the cut $(S, V\text{-}S)$ if one of its endpoints is in S and the other is in $V\text{-}S$.
- A cut **respects** a set A of edges if no edge in A crosses the cut.
- An edge is a **light edge** crossing a cut if its weight is the minimum of any edge crossing the cut (may not be unique).
- More generally, an edge is a **light edge** satisfying a given property if its weight is the minimum of any edge satisfying the property.

Light edge:Any edge crossing a cut is light edge if its weight is the minimum of all the edge crossing the cut. Light edge is defined with respect to a particular Cut.

A “cut”



A “cut”



Safe Edges Theorem (23.1)

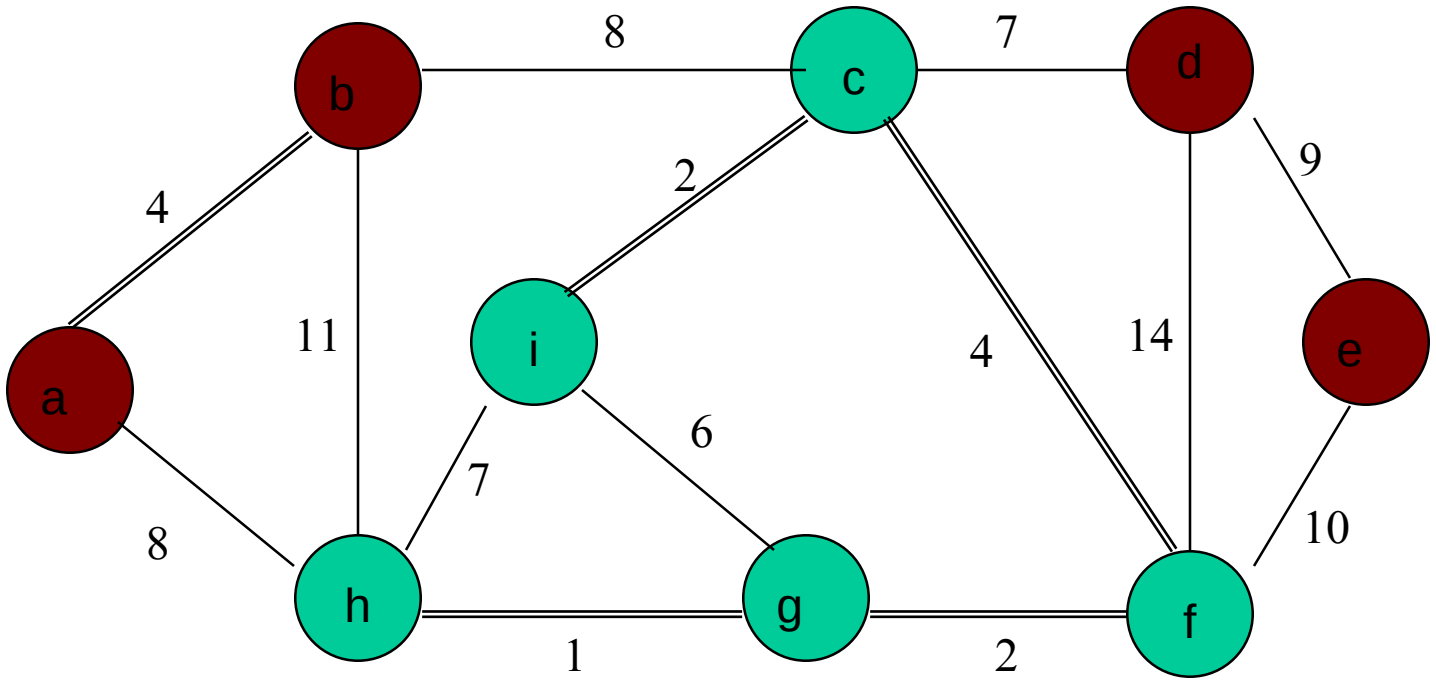
An edge that may be added to A without violating the invariant that A is a subset of some minimum spanning tree

- Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E.
- Let A be a subset of E that is included in some minimum spanning tree for G.
- Let $(S, V\text{-}S)$ be any cut of G that **respects** A (that is, no edge in A crosses the cut).
- Let (u, v) be a **light edge** crossing $(S, V\text{-}S)$ (i.e., its weight is the minimum of any edge crossing the cut).

Then edge (u, v) is **safe** for A.

Maroon = vertices in set S Double lines = edges in set A

Green = vertices in set $(V - S)$



The edges connecting maroon with green vertices *cross the cut*.

Edge (d, c) is a *light edge* crossing the cut.

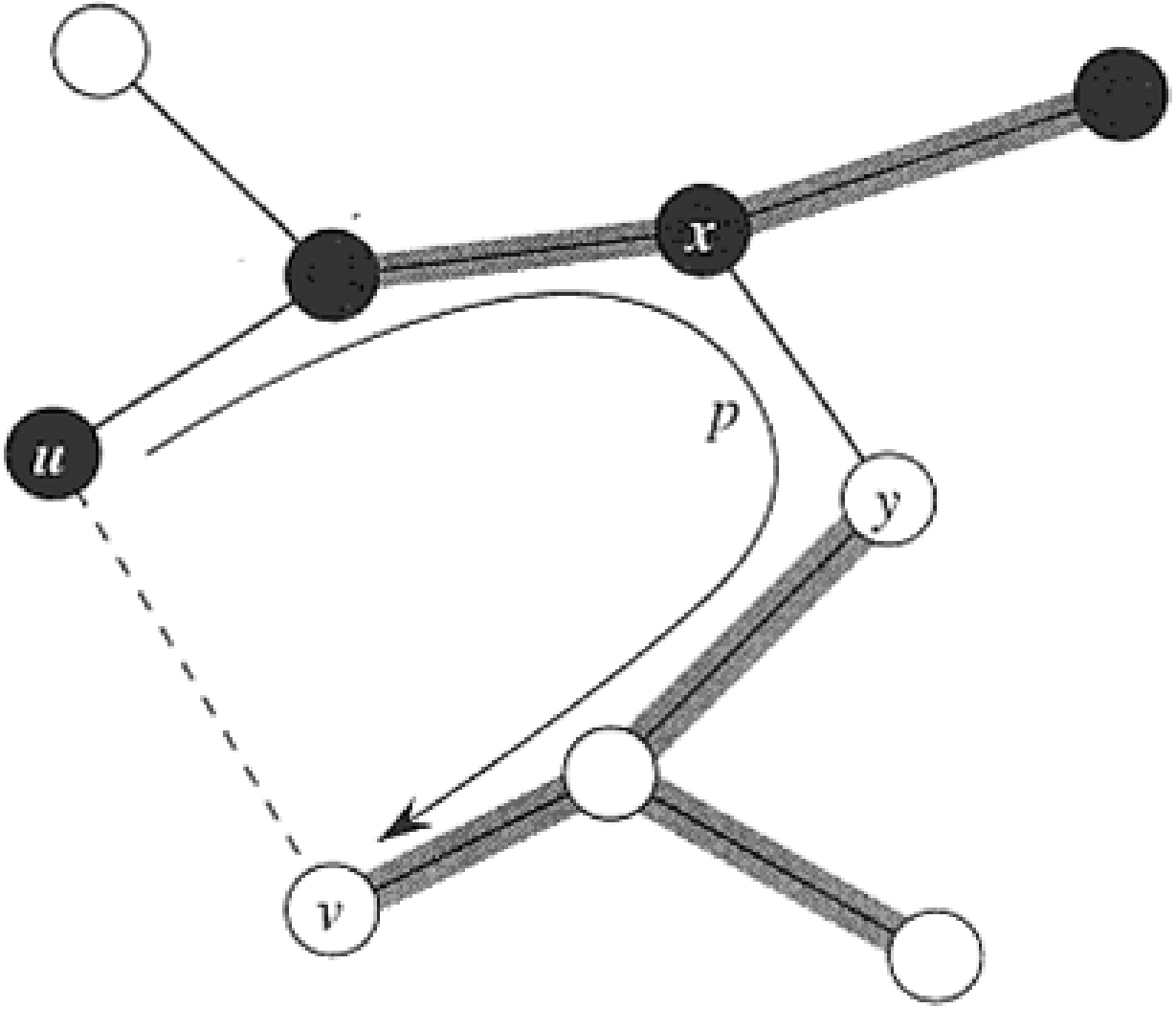
The cut $(S, V\text{-}S)$ *respects* A; i.e., no edge of A crosses the cut.

Proof of Safe Edges Theorem

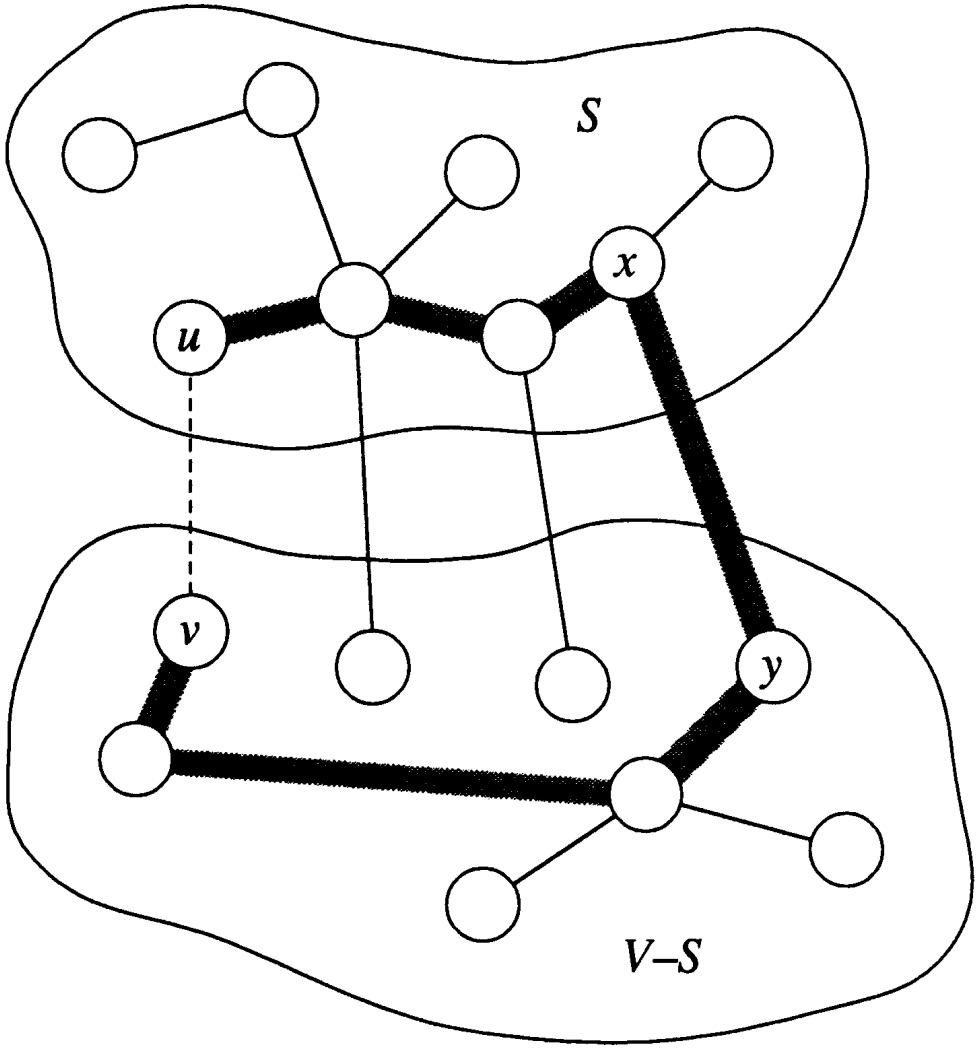
Let T be a minimum spanning tree that includes A, and assume that T does not contain the light edge (u,v), since if it does, we are done. (Why?)

We will construct another spanning tree T' that includes $A \cup \{(u,v)\}$ by using a cut and paste technique, thereby showing that (u,v) is a safe edge for A.

Proof of Safe Edges Theorem



Proof of Safe Edges Theorem



Proof continued

The edge (u,v) forms a cycle with the edges on the path p from u to v in T. (Why?)

Since u and v are on opposite sides of the cut (S, V-S), there is at least one edge in T on the path p that also crosses the cut. (Why?)

Let (x,y) be any such edge. The edge (x,y) is not in A. (Why?)

Since (x,y) is on the unique path from u to v in T, removing (x,y) breaks T into two components. (Why?)

Proof continued

Adding (u,v) reconnects them to form a new spanning tree $T' = T - \{(x,y)\} \cup \{(u,v)\}$. (Why?)

Now we need to show that T' is a MST (not just a ST).

- (u,v) is a light edge crossing $(S, V\text{-}S)$
- (x,y) also crosses this cut
- therefore $w(u,v) <= w(x,y)$
- therefore $w(T') = w(T) - w(x,y) + w(u,v)$
  $<= w(T)$

But T is an MST, so $w(T) <= w(T')$ so T' must also be an MST.

Proof continued

We have shown that we can take an MST T and construct another MST T' from it by substituting (u,v) for (x,y).

Now we need to show that (u,v) is a safe edge for A.

$A \subseteq T'$, since $A \subseteq T$ and $(x,y) \notin A$;

therefore, $A \cup \{(u,v)\} \subseteq T'$

therefore, T' is an MST

therefore (u,v) is safe for A

Corollary 23.2

Let $G = (V,E)$ be a connected, undirected graph with a real-valued weight function w defined on E.

Let A be a subset of E that is included in some MST for G, and let C be a connected component (tree) in the forest $G_A = (V,A)$.

If (u,v) is a light edge connecting C to some other component in G_A, then (u,v) is safe for A.

Proof: The cut $(C, V\text{-}C)$ respects A, and (u,v) is therefore a light edge for this cut.

Two MST Algorithms

- Two famous greedy algorithms use the generic algorithm but pick safe edges differently
- Kruskal's algorithm
  - A is a forest
  - always add the edge of minimum weight that does not introduce a cycle
- Prim's algorithm
  - A is a tree
  - always add a minimum weight edge to the existing tree

Disjoint Sets

- Initially each vertex is a connected component
- Represent each connected component as a set. The sets are disjoint.
- When considering an edge, determine if the two endpoints are in the same connected component (disjoint set).

Kruskal's Algorithm

Let edge (u, v) be the edge of least weight that connects two trees C_1 and C_2.

Since (u, v) is a light edge connecting C_1 to some other tree, it is a safe edge for C.

Implementing Kruskal's Algorithm

- Initially, the MST has $|V|$ vertices and 0 edges ($A = \varnothing$)
- While A is not an MST
  - Find the cheapest edge not yet considered
    - If you add it to A, would you induce a cycle?
    - If not, add it to A

Kruskal's Algorithm

```
MST-Kruskal (G, w)
1  A ← Ø
2  for each vertex v ∈V[G] do
3     MAKE-SET(v)
4  sort the edges of E into
   nondecreasing order by weight
5  for each (u, v) ∈ E, taken in
   nondecreasing order do
6     if FIND-SET(u) ≠ FIND-SET(v)
7        then A ← A ∪ {(u, v)}
8             UNION(u, v)
9  return A
```

Analysis of Kruskal's Algorithm

Initialization $O(V)$

Sort of edges $O(E \lg E)$

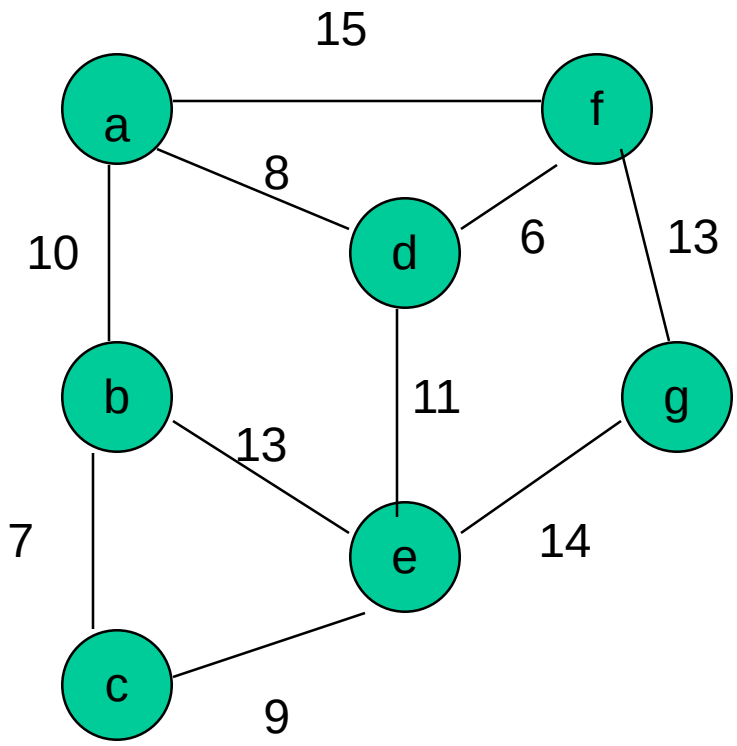
$O(E)$ disjoint forest operations:

total time $O(E\alpha(E,V))$

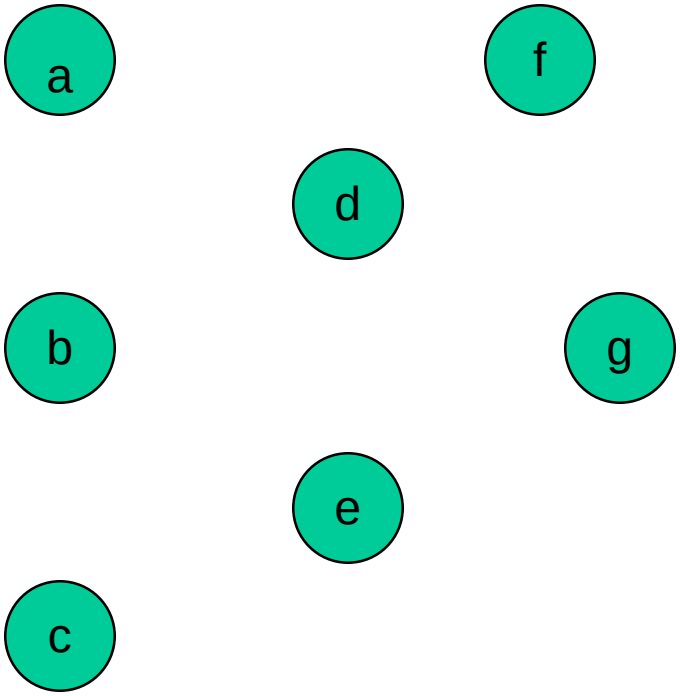
$\alpha(E,V) = O(\lg E) = O(\lg V)$

Total time $O(E \lg E)$ or $O(E \lg V)$

Here is the complete graph



Beginning Kruskal's algorithm . . .

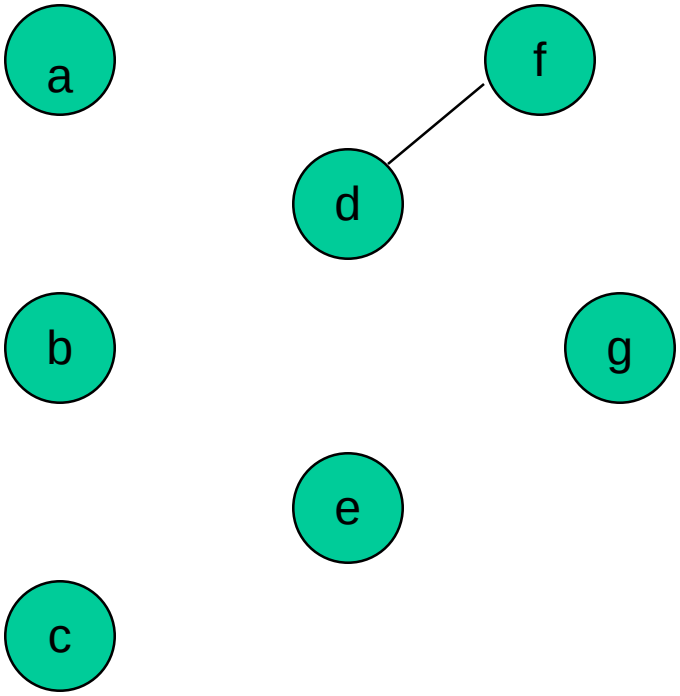


d f	6
b c	7
a d	8
c e	9
b e	10
d e	11
f g	13
a b	13
e g	14
a f	15

Set of edges in MST:
A = { }

Sorted set of all edges

First step

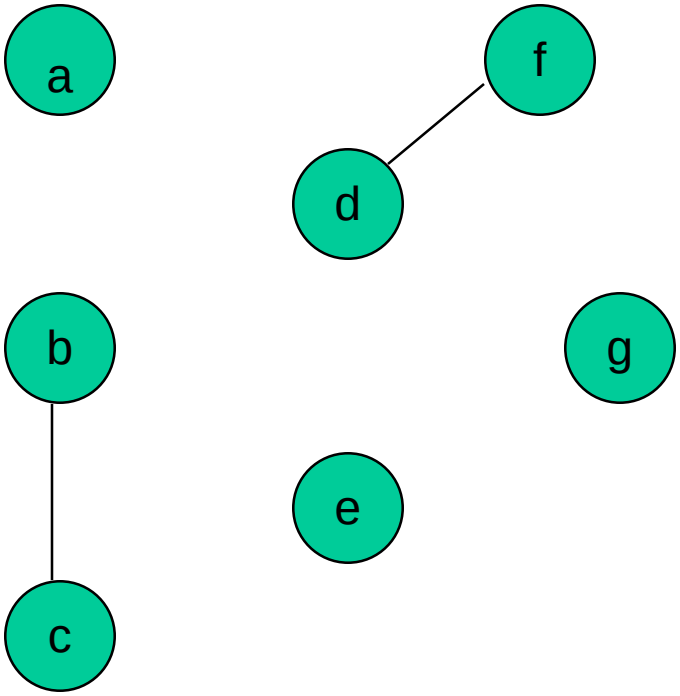


b c	7
a d	8
c e	9
b e	10
d e	11
f g	13
a b	13
e g	14
a f	15

Set of edges in MST:
A = {(d,f)}

Sorted set of edges
with (d,f) removed

Second step

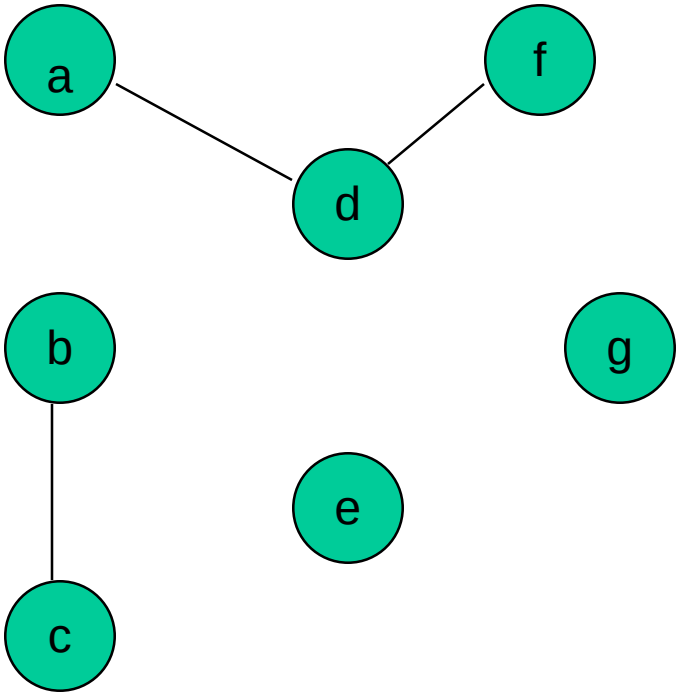


a d	8
c e	9
b e	10
d e	11
f g	13
a b	13
e g	14
a f	15

Set of edges in MST:
A = {(d,f), (b,c)}

Sorted set of edges
with (b,c) removed

Third step

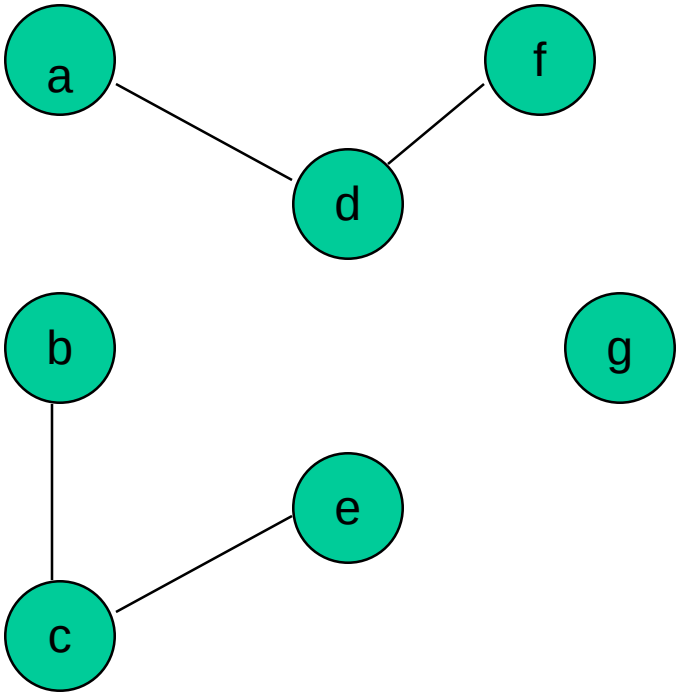


c e	9
b e	10
d e	11
f g	13
a b	13
e g	14
a f	15

Set of edges in MST:
A = {(d,f), (b,c), (a,d)}

Sorted set of edges
with (a,d) removed

Fourth step

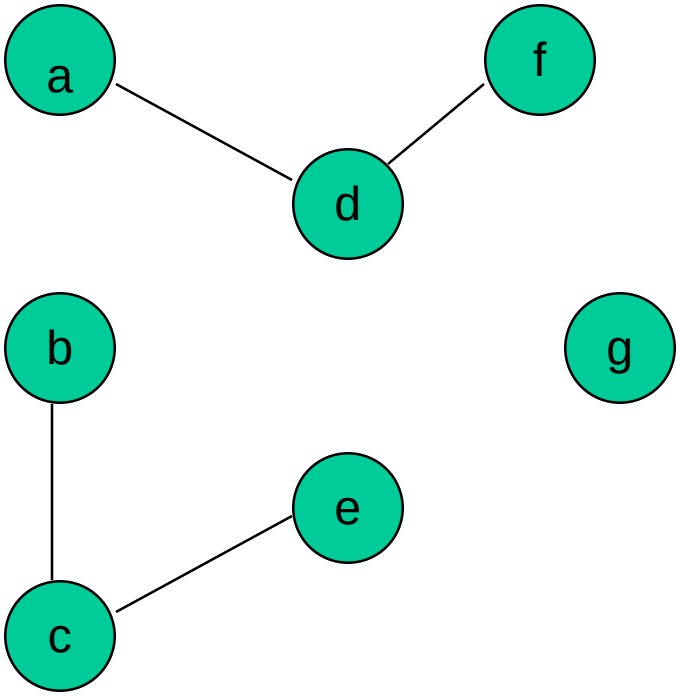


b e	10
d e	11
f g	13
a b	13
e g	14
a f	15

Set of edges in MST:
A = {(d,f), (b,c), (a,d), (c,e)}

Sorted set of edges
with (c,e) removed

Fifth step



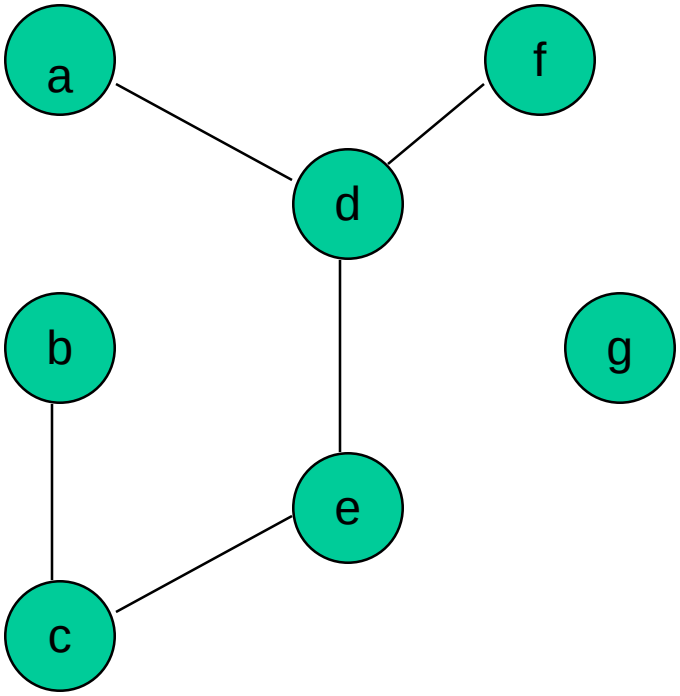
Edge (b,e) would create a cycle, so we don't add it to A.

d e	11
f g	13
a b	13
e g	14
a f	15

Set of edges in MST:
A = {(d,f), (b,c), (a,d), (c,e)}

Sorted set of edges with (b,e) removed

Sixth step

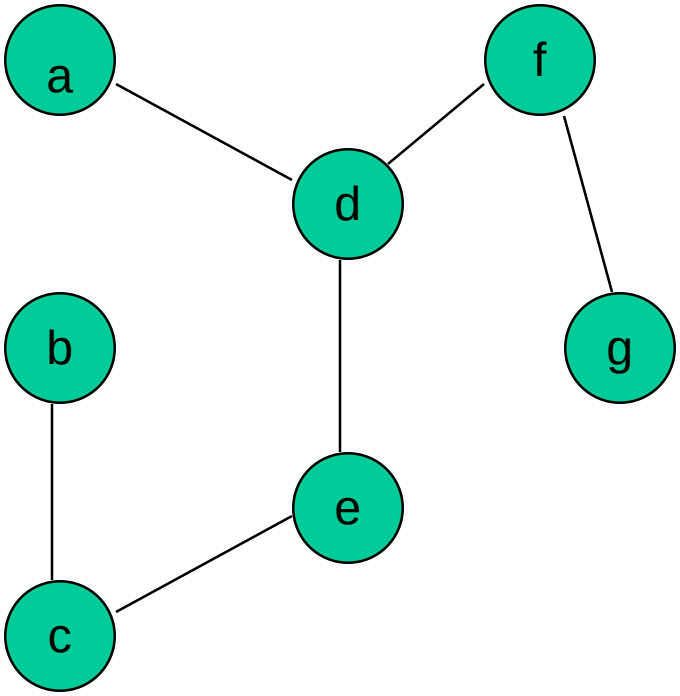


f g	13
a b	13
e g	14
a f	15

Set of edges in MST:
A = {(d,f), (b,c), (a,d), (c,e), (d,e)}

Sorted set of edges
with (d,e) removed

Seventh step



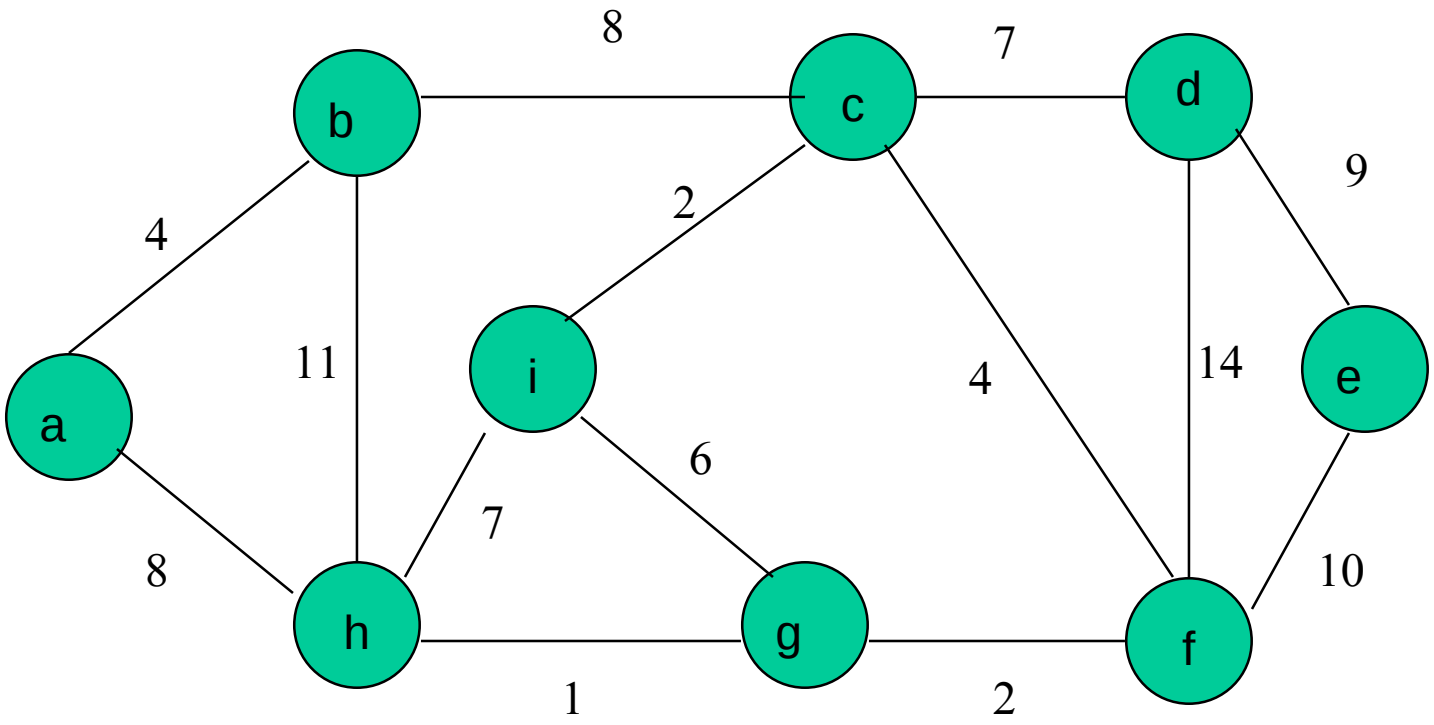
a b	13
e g	14
a f	15

MST found!

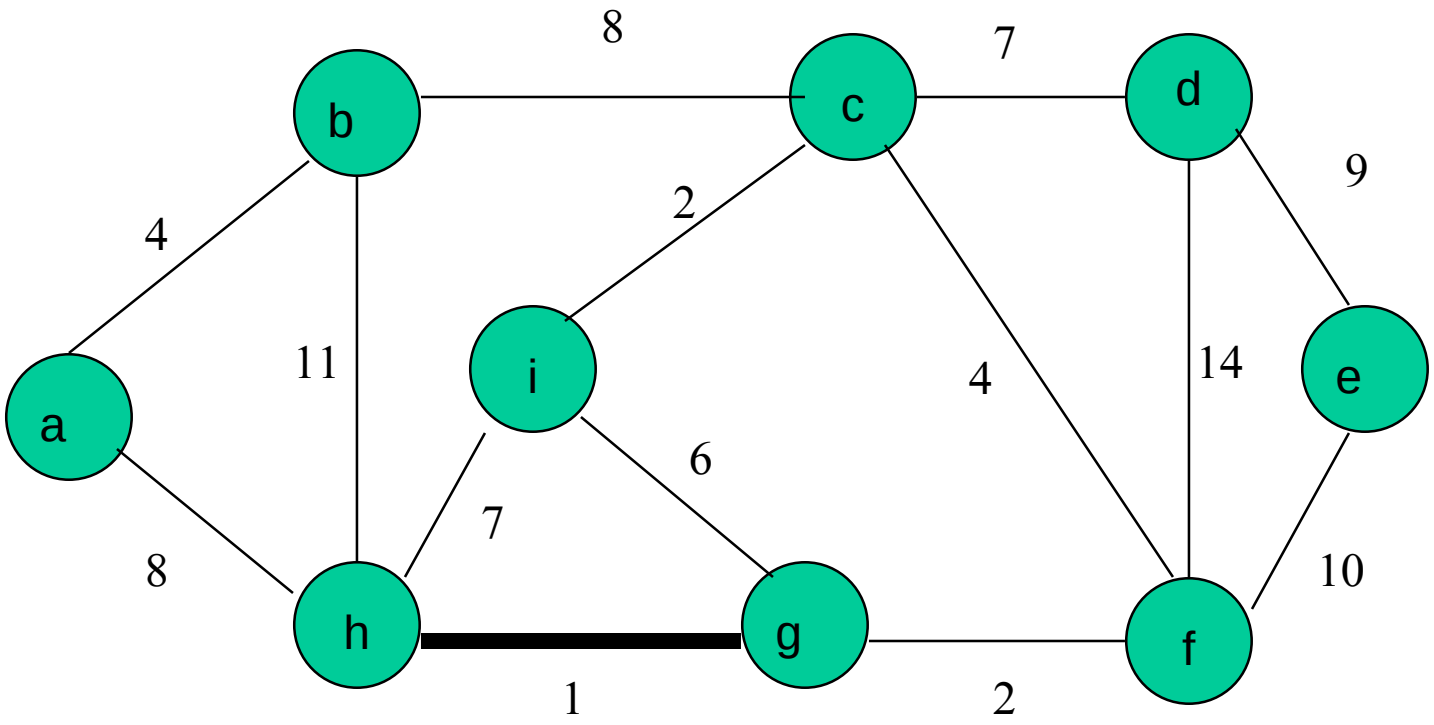
Set of edges in MST:
A = {(d,f), (b,c), (a,d), (c,e), (d,e), (f,g)}

Sorted set of edges with (f,g) removed

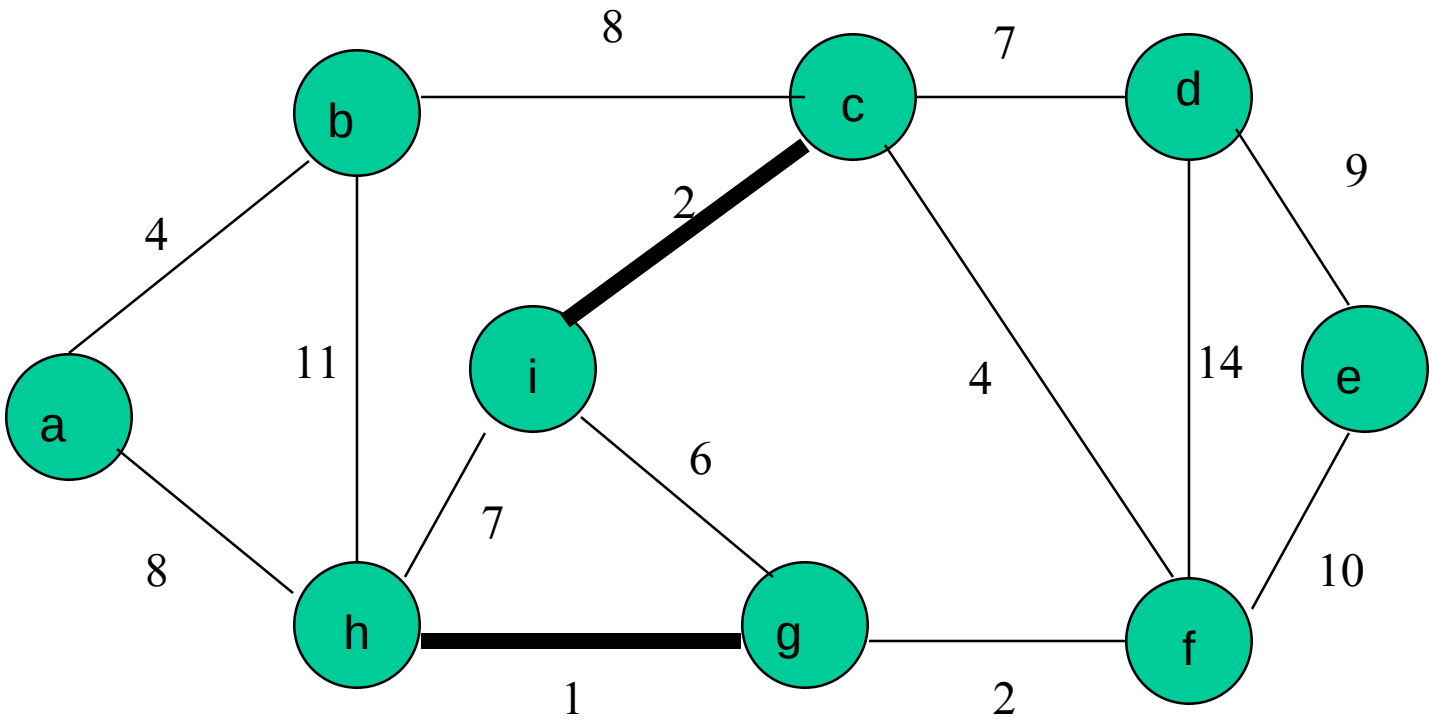
Example of Kruskal's Algorithm



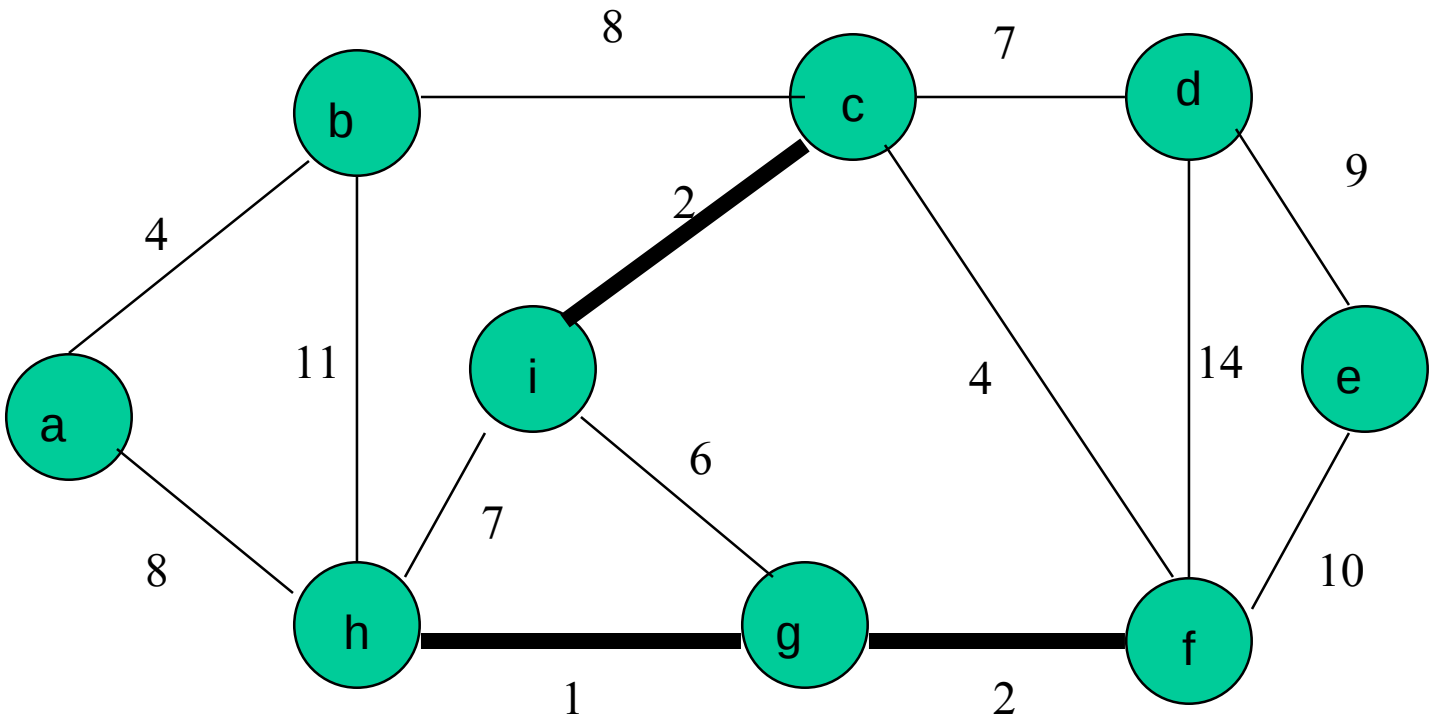
Example of Kruskal's Algorithm



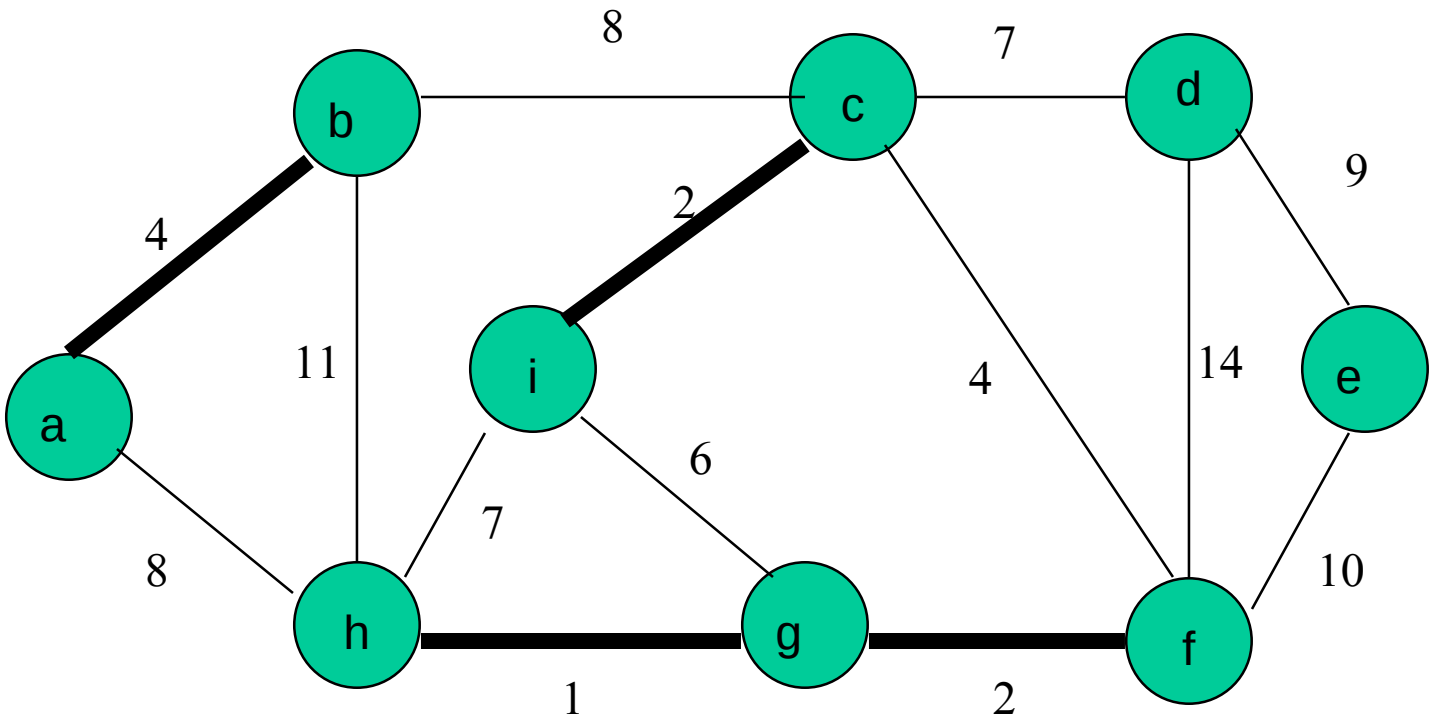
Example of Kruskal's Algorithm



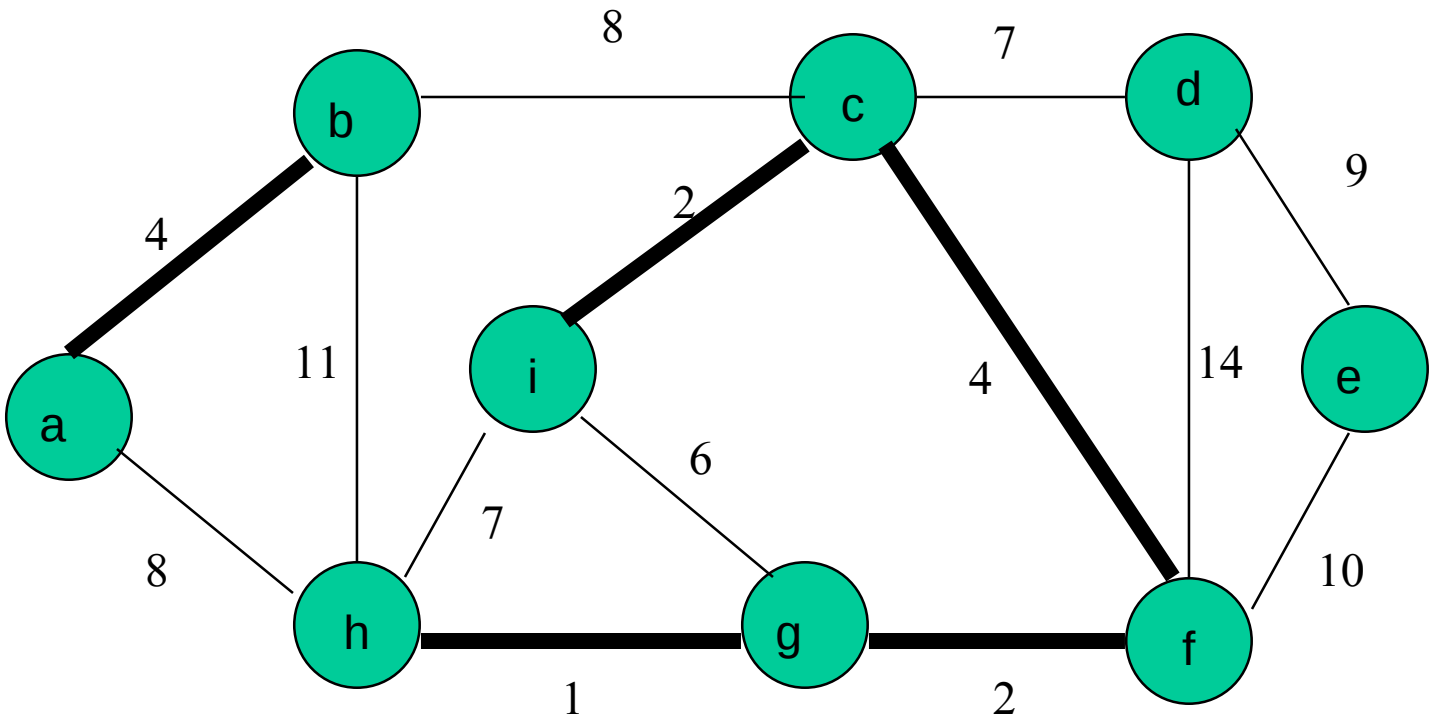
Example of Kruskal's Algorithm



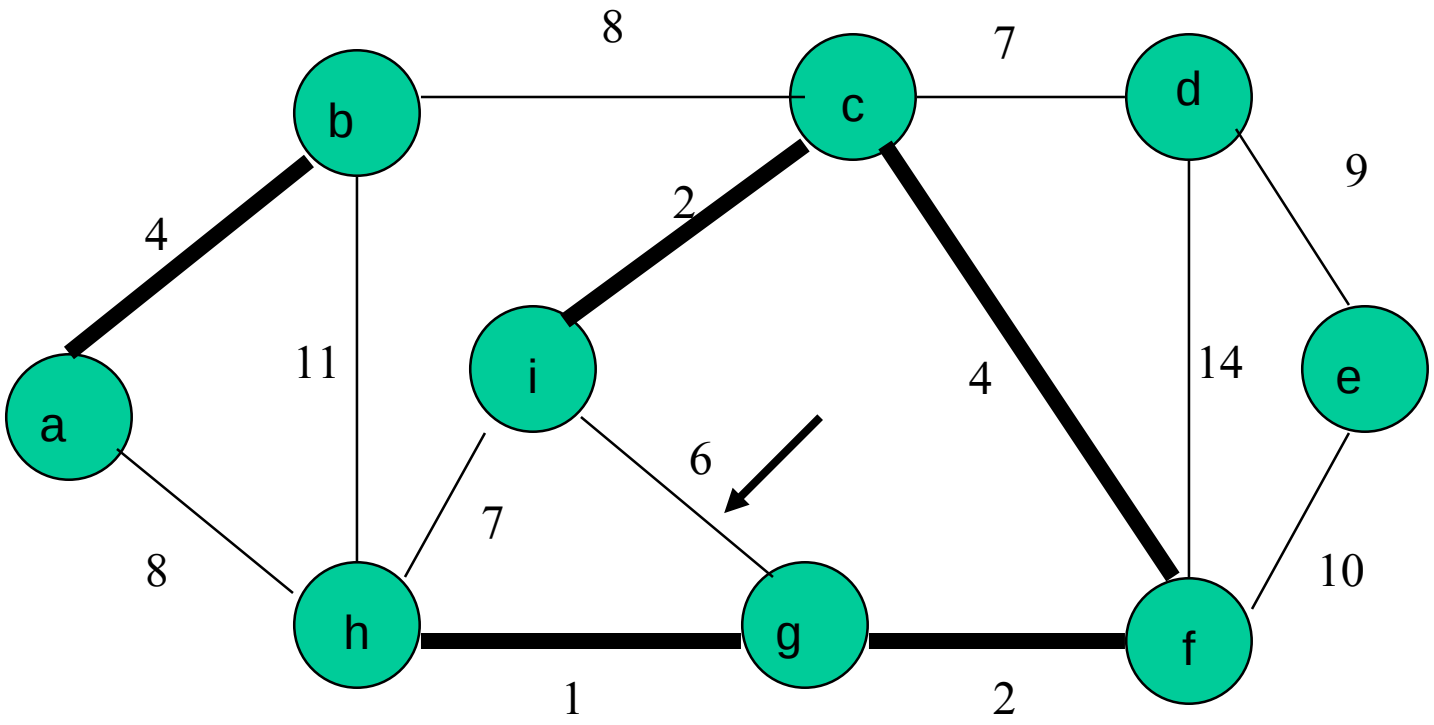
Example of Kruskal's Algorithm



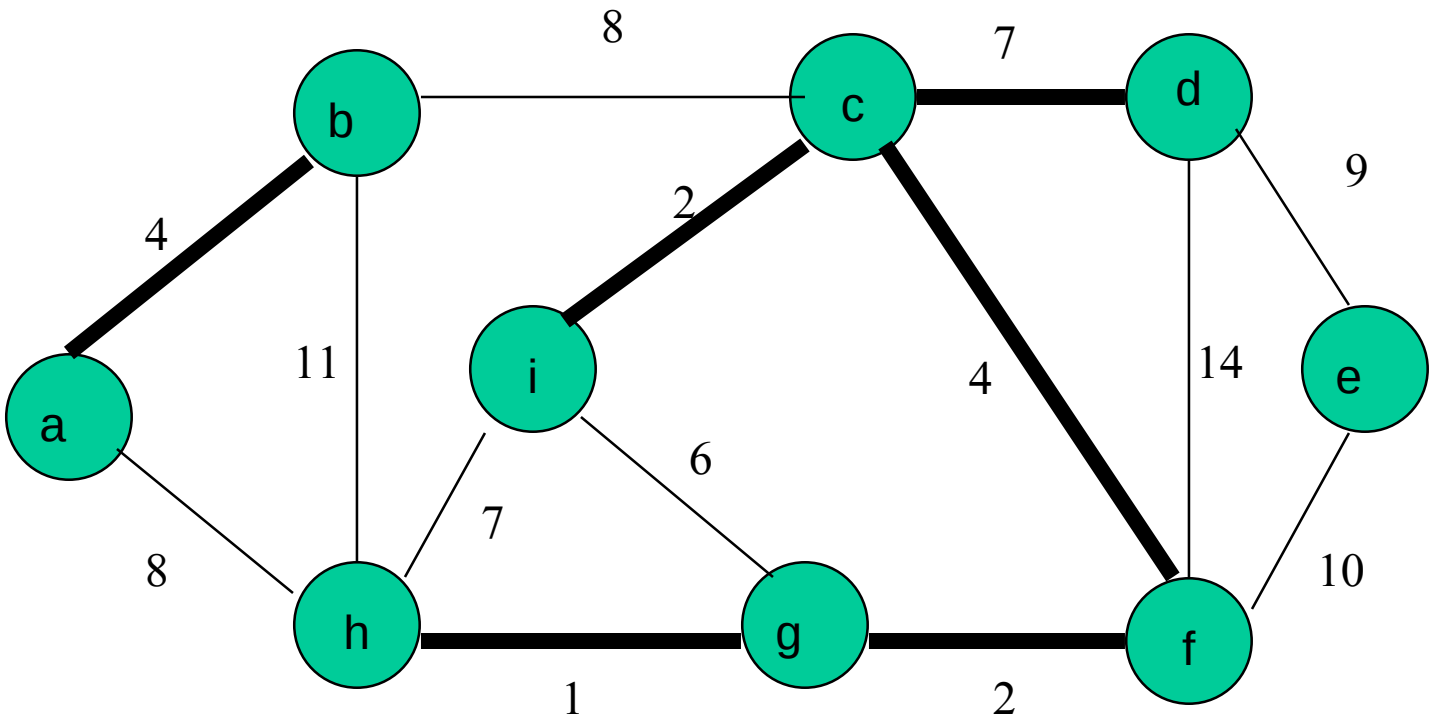
Example of Kruskal's Algorithm



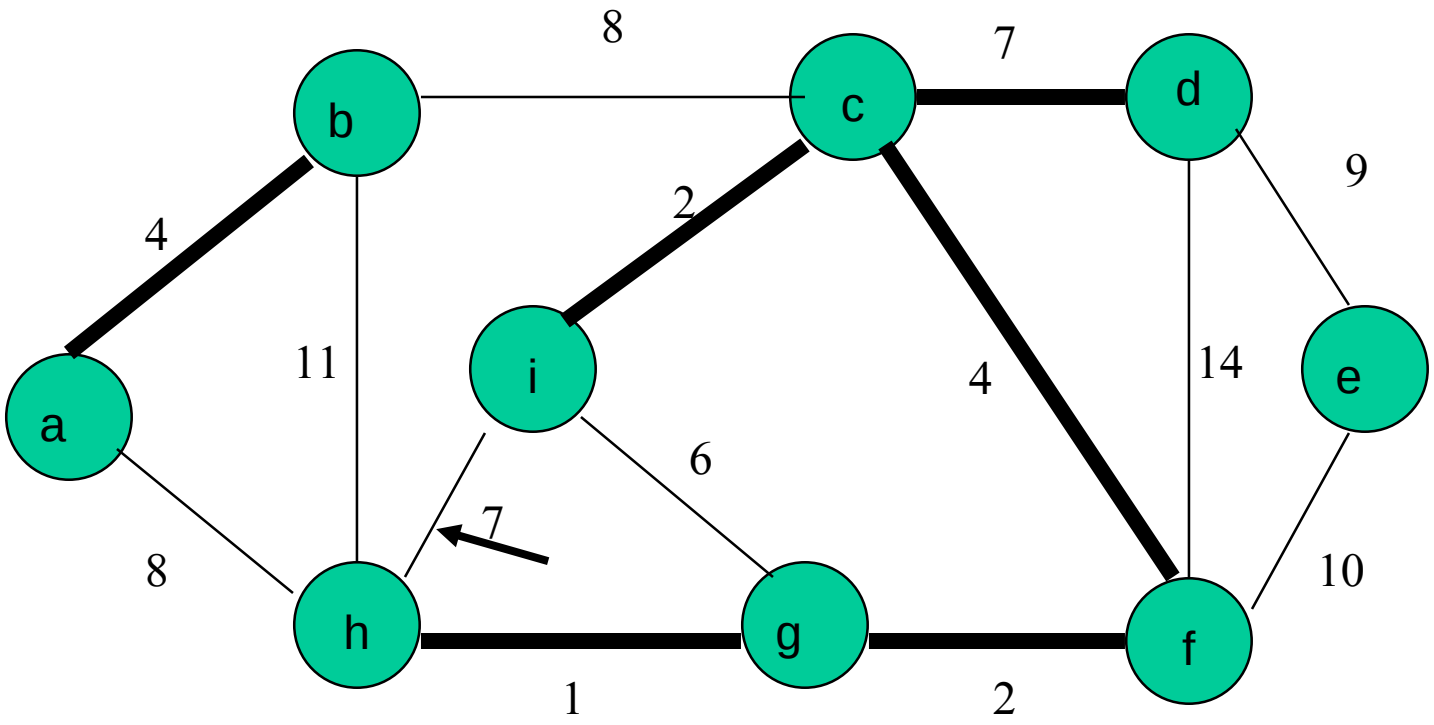
Example of Kruskal's Algorithm



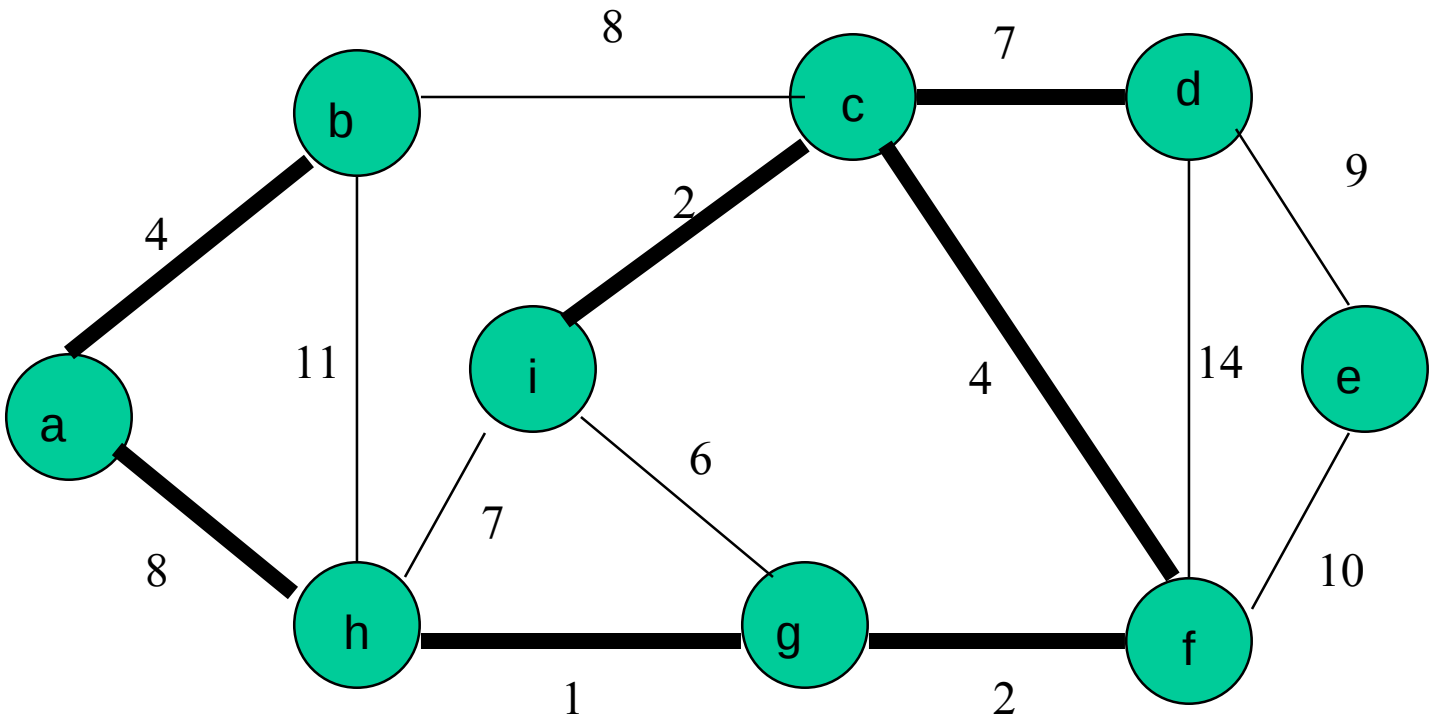
Example of Kruskal's Algorithm



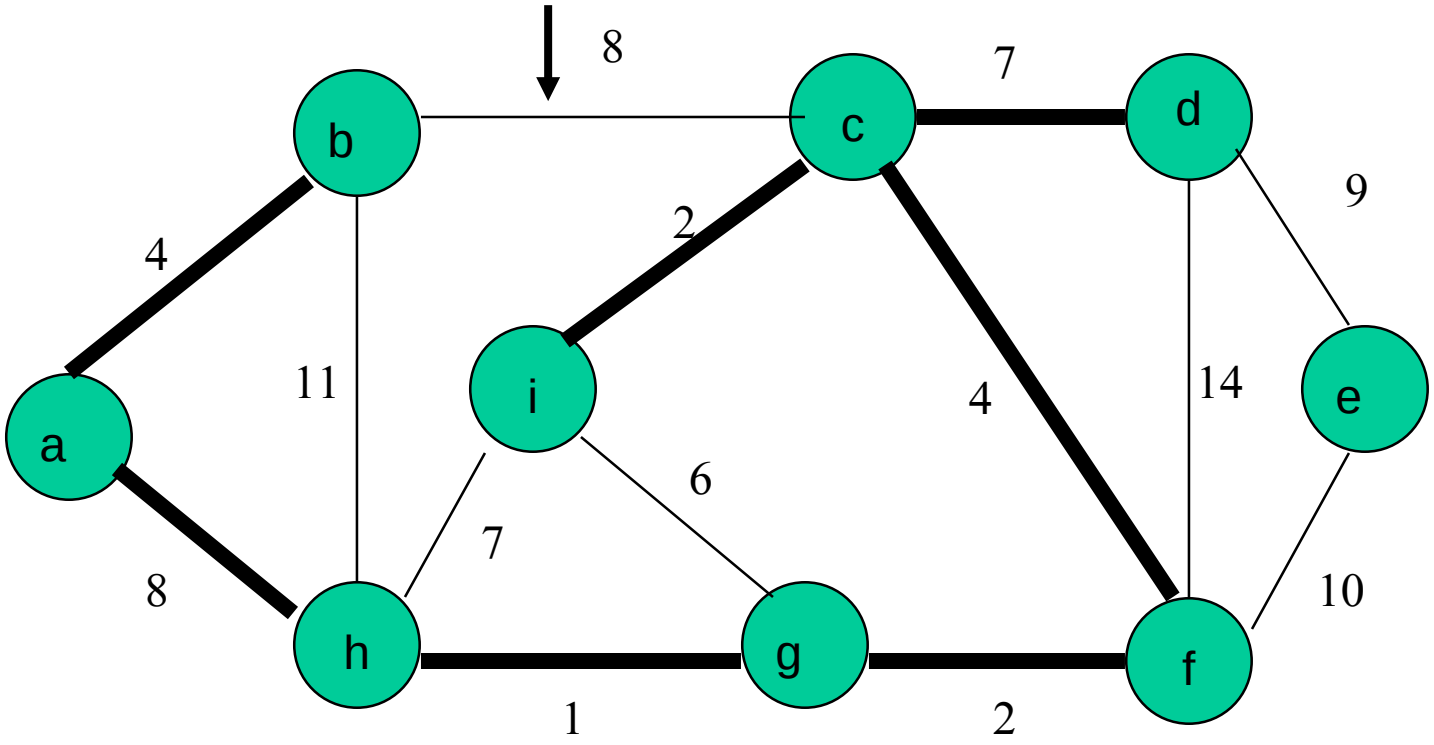
Example of Kruskal's Algorithm



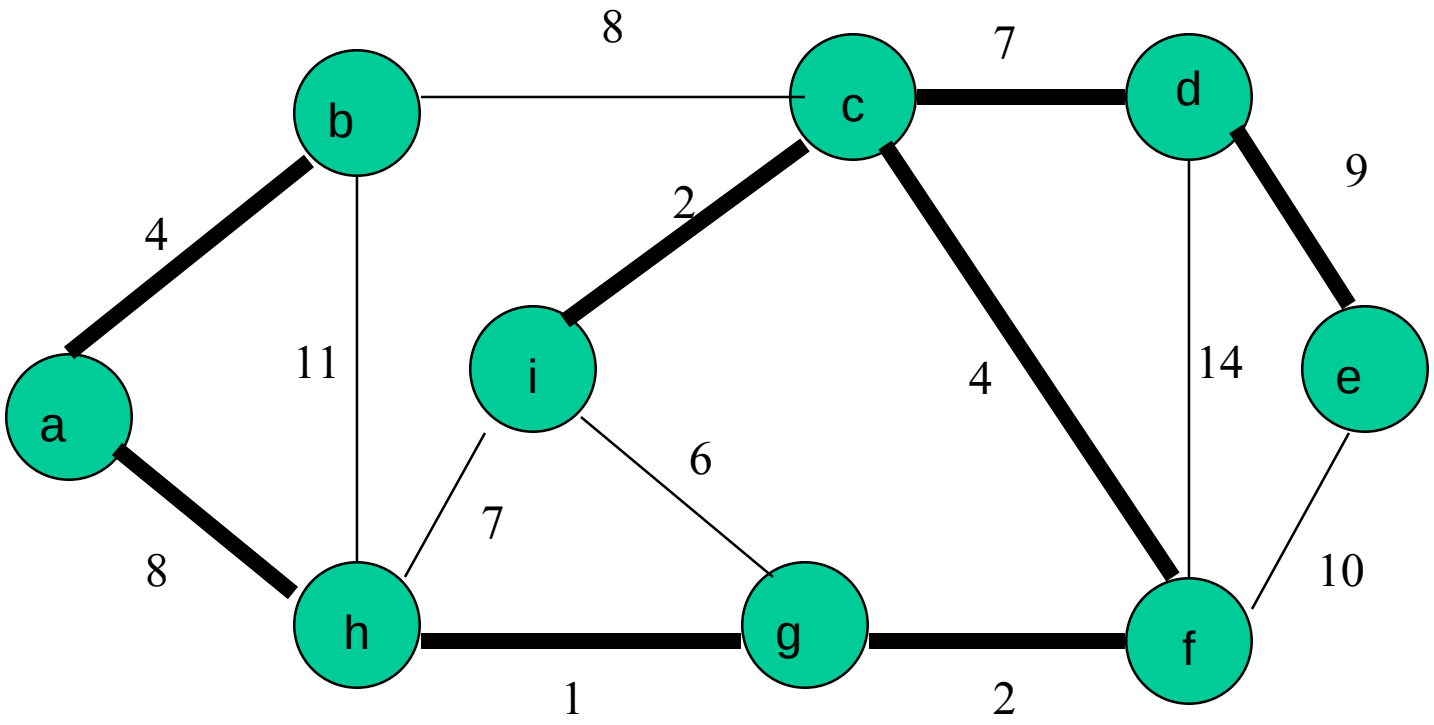
Example of Kruskal's Algorithm



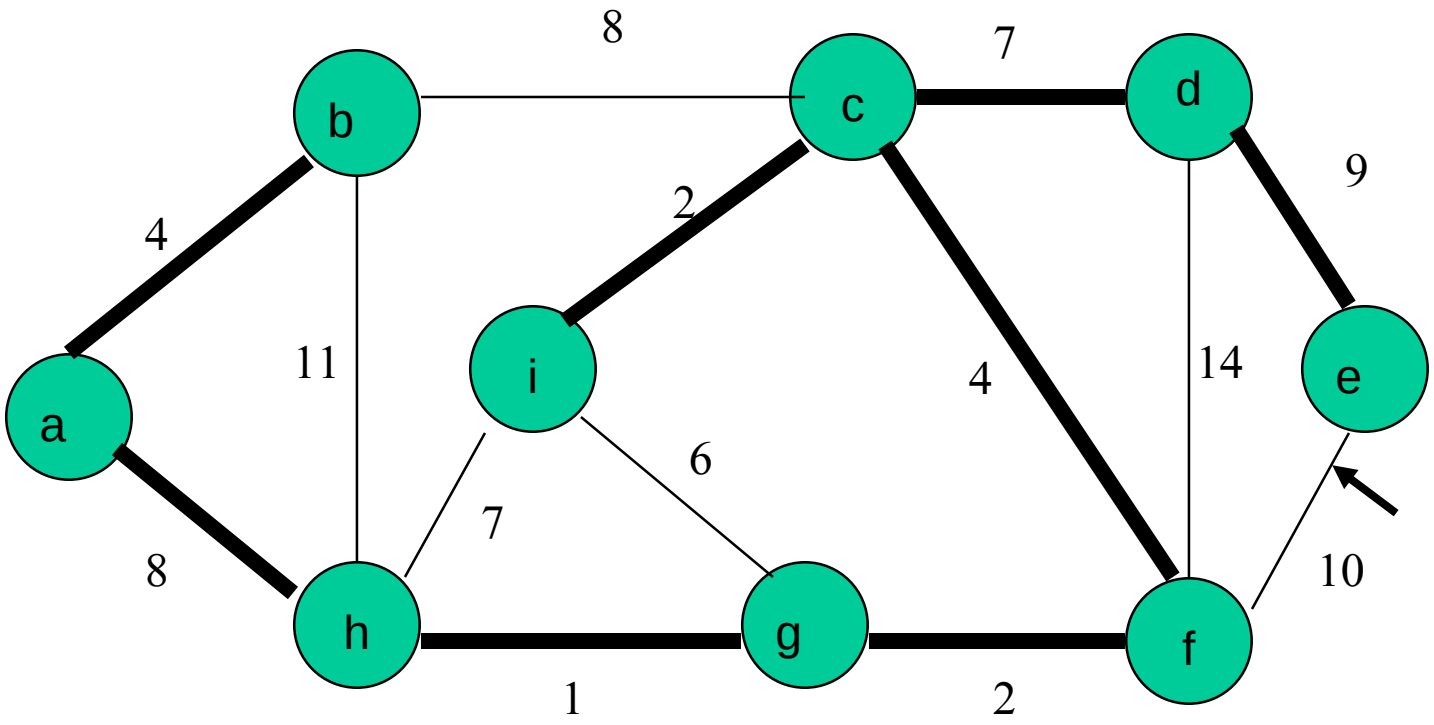
Example of Kruskal's Algorithm



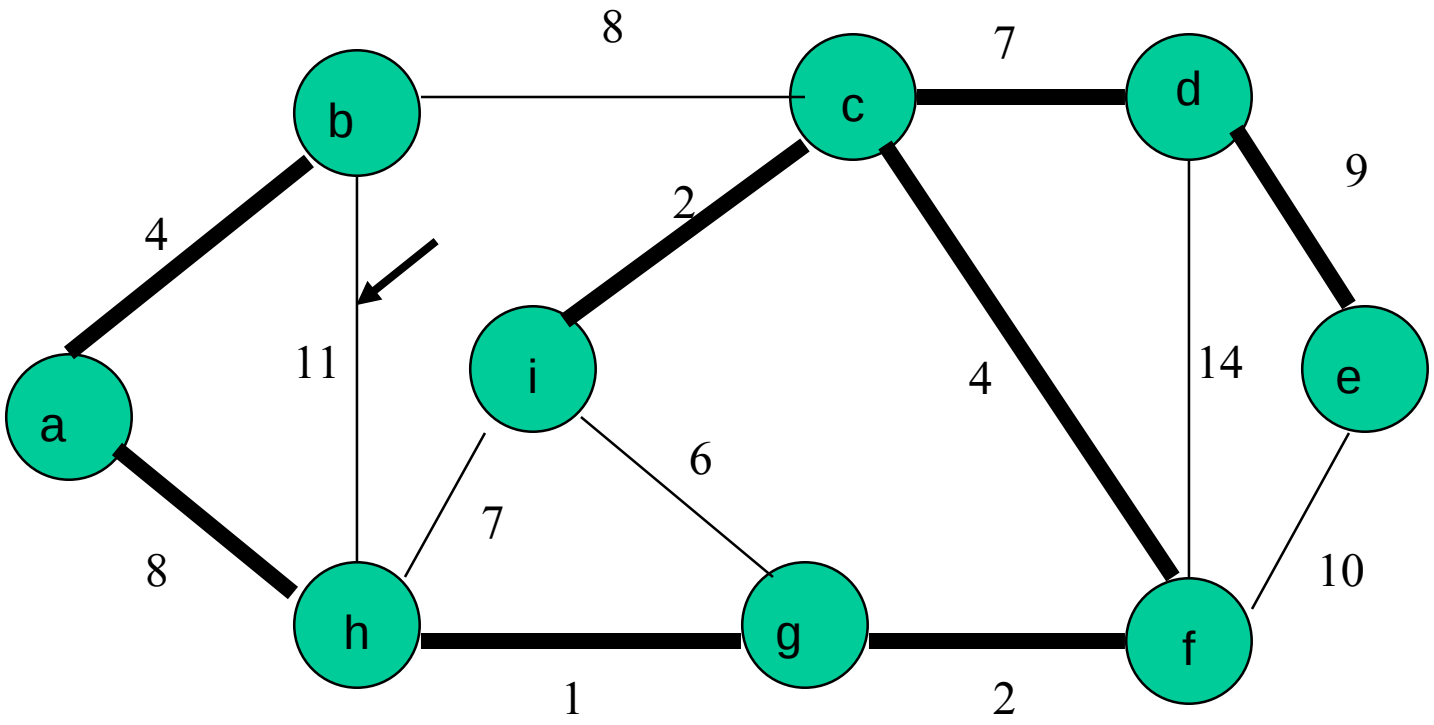
Example of Kruskal's Algorithm



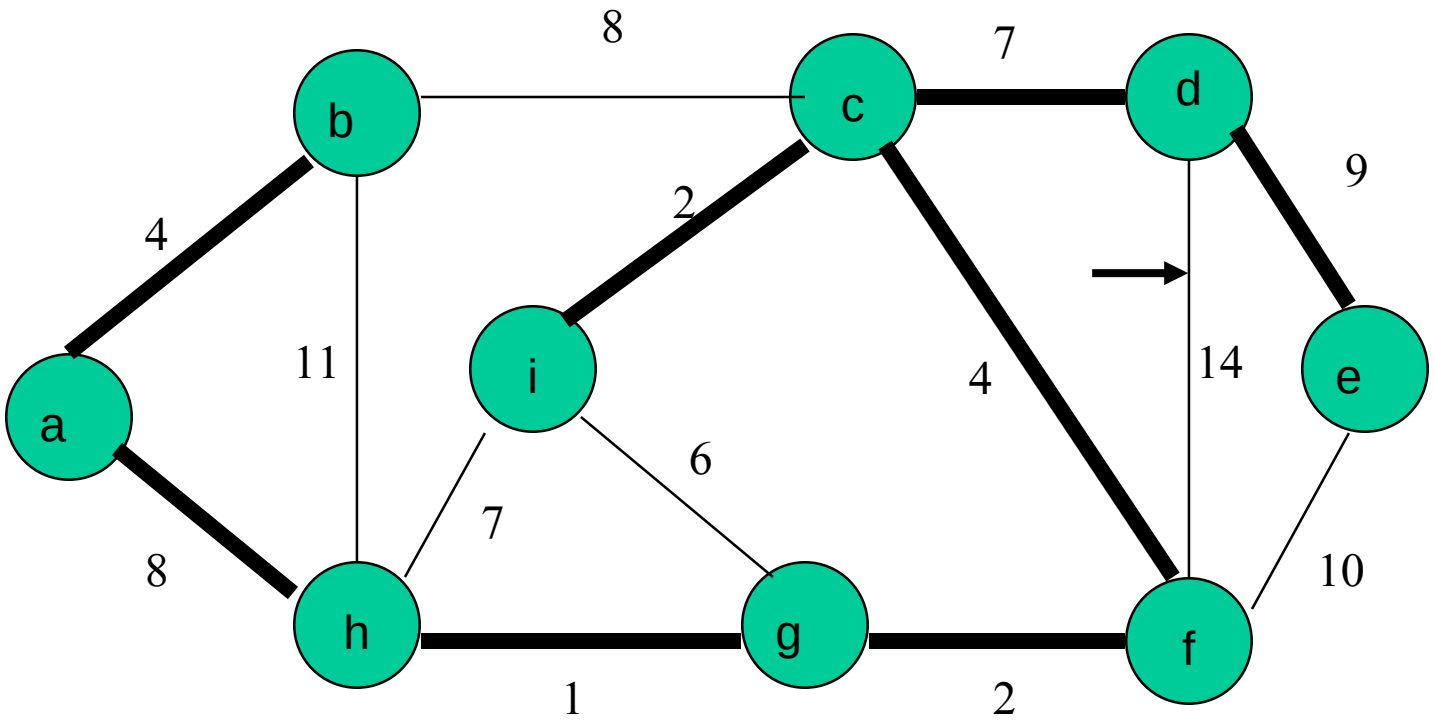
Example of Kruskal's Algorithm



Example of Kruskal's Algorithm



Example of Kruskal's Algorithm



Prim's Algorithm

Prim's algorithm is another instantiation of the generic minimum-spanning-tree algorithm.

Prim's algorithm maintains a tree structure from start to finish; at each step, the edge added always connects a single new vertex to the tree.

The edge chosen is always the safe edge that adds the smallest amount to the tree's weight.

Prim's Algorithm

Let G be a connected graph.

Let r be the vertex that will serve as the root of the minimum cost spanning tree.

At each step, add to the tree A a light edge that connects A to an isolated vertex of $G_A = (V, A)$.

By our corollary, this adds only safe edges to A.

Choose the edge that adds the minimum amount possible to the tree's weight.

Implementation of Prim's Algorithm

Management of a priority queue is the critical step in Prim's algorithm.

We can use a binary heap for the queue.

Implementation of Prim's Algorithm

r is the vertex that serves as the root of the MST.

All vertices that are not in *A* (the growing MST) are in a min-priority queue, *Q*.

Each vertex *v* in the queue has a key field; *key[v]* is the minimum weight of any edge connecting *v* to a vertex in the tree. If there is no edge connecting *v* to a vertex in the tree, *key[v]* = ∞.

$A = \{(v, \pi(v)): v \in V - \{r\} - Q\}$

$\pi(v)$ names the parent of v in the tree.

Prim's Algorithm

```
MST-PRIM (G, w, r)
1  for each u ∈ V[G] do
2      key[u] ← ∞
3      π[u] ← NIL      // π[u] is the parent of u
4  key[r] ← 0
5  Q ← V[G]
6  while Q ≠ ∅ do
7      u ← EXTRACT-MIN(Q)
8      for each v ∈ Adj[u] do
9          if v ∈ Q and w(u, v) < key[v]
10             then π[v] ← u
11                  key[v] ← w(u, v)
```

Prim's Algorithm

`MST-PRIM (G, w, r)`

Prim's function is called with three parameters:

G – the graph of vertices and edges

w – the set of weights on the edges

r – the vertex that will serve as the root of the MST.

Note that the weight of an edge is not the same as the *key* of a vertex.

Prim's Algorithm

```
MST-PRIM (G, w, r)
1   for each u ∈ V[G] do
2       key[u] ← ∞
3       π[u] ← NIL
4   key[r] ← 0
5   Q ← V[G]
```

In lines 1 through 5:
The key of each vertex is set to ∞
The parent of each vertex is set to NIL.
The key of the root vertex, *r*, is set to 0, so it will be the first vertex processed.
The min-priority queue is initialized to contain all the vertices.

Prim's Algorithm

```
6   while Q ≠ ∅ do
7       u ← EXTRACT-MIN(Q)
8       for each v ∈ Adj[u] do
9           if v ∈ Q and w(u, v) < key[v]
10              then π[v] ← u
11                   key[v] ← w(u, v)
```

The *while* loop in line 6 executes until the queue is empty.

In line 7, the top node in the min-priority queue is assigned to *u*.

The first time through line 7, the top node is *r*. Thereafter, the top node is a vertex incident on a light edge crossing the cut (V – Q, Q).

The *for* loop in lines 8-11 updates the *key* and π fields of every vertex *V* adjacent to *u* and in *Q* (and therefore not in *A*)

Prim's Algorithm

```
6   while Q ≠ ∅ do
7       u ← EXTRACT-MIN(Q)
8       for each v ∈ Adj[u] do
9           if v ∈ Q and w(u, v) < key[v]
10              then π[v] ← u
11                   key[v] ← w(u, v)
```

As the *key* for a vertex is changed in line 11, the min-priority heap has to be re-heapified, costing $O(\log_2 V)$.

Note that, initially, all keys (except the key for *r*), will be ∞. Only the vertices *v* adjacent to *u* (the top vertex in the min-priority heap) will have their key values changed to *w(u, v)*. These weights will be less than ∞, so the adjacent *v* vertices will bubble up to the top of the heap, making these adjacent vertices available to be considered for the MST.

Analysis of Prim's Algorithm

Implement Q as a binary heap:

BUILD-MIN-HEAP (lines 1-5)	O(V)
while loop in line 6 is executed	|V| times
Extract MIN	O(lg V)
for loop within *while* loop	O(E/V)
test for membership in Q in line 9	O(1)
assignment in line 11 has implicit DECREASE-KEY	O(lg V)

Total time: O(V lg V) + O(E lg V) = O(E lg V)

Analysis of Prim's Algorithm

Be careful with the analysis! The cost of the *for* loop within *while* loop is O(E/V) because:

The sum of the lengths of all of the adjacency lists in line 8 that have to be examined is 2|E|.

Thus, the total number of times the *for* loop is executed within the *while* loop is O(E).

Therefore, the average number of times the *for* loop is executed each trip through the *while* loop is E/V.

Since the total number of times the *for* loop is executed is O(E), the total cost of lines 8-11 is O(E lg V).

Analysis of Prim's Algorithm

The test for membership in Q in line 9 can be implemented by keeping a bit for each vertex that tells whether or not this vertex is in Q. This bit is updated when the vertex is removed from Q.

The cost of this operation is O(1).

The assignment in line 11 is:

`key[v]` ← `w(u, v)`

This requires that a DECREASE-KEY operation be performed, which includes a re-heapification of the heap. This costs O(lg V).

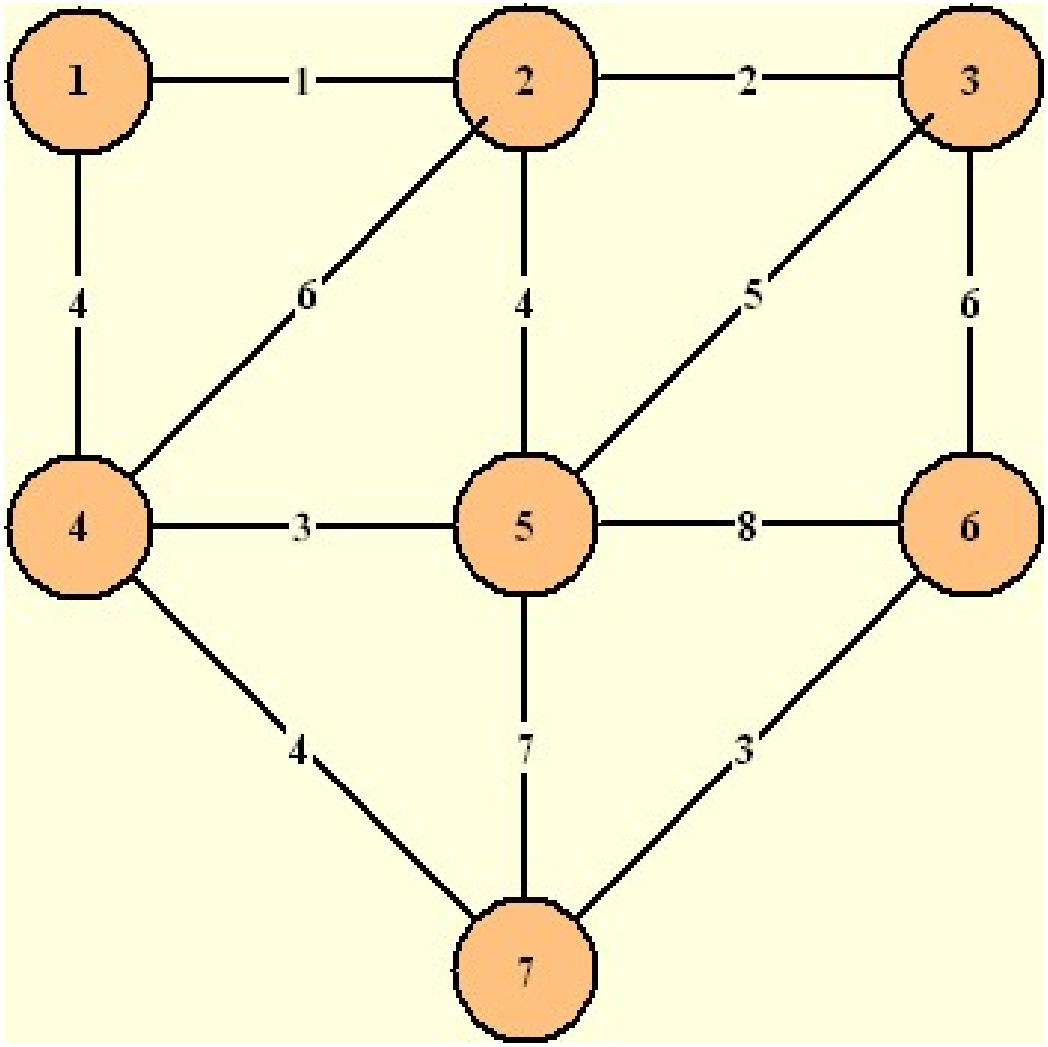
Analysis of Prim's Algorithm

Theoretically, the asymptotic running time of Prim's algorithm can be improved to $O(E + V \lg V)$ by using Fibonacci heaps.

(See Chapter 20 for more information on Fibonacci heaps.)

In practice, binary heaps are usually better.

Example of Prim's Algorithm

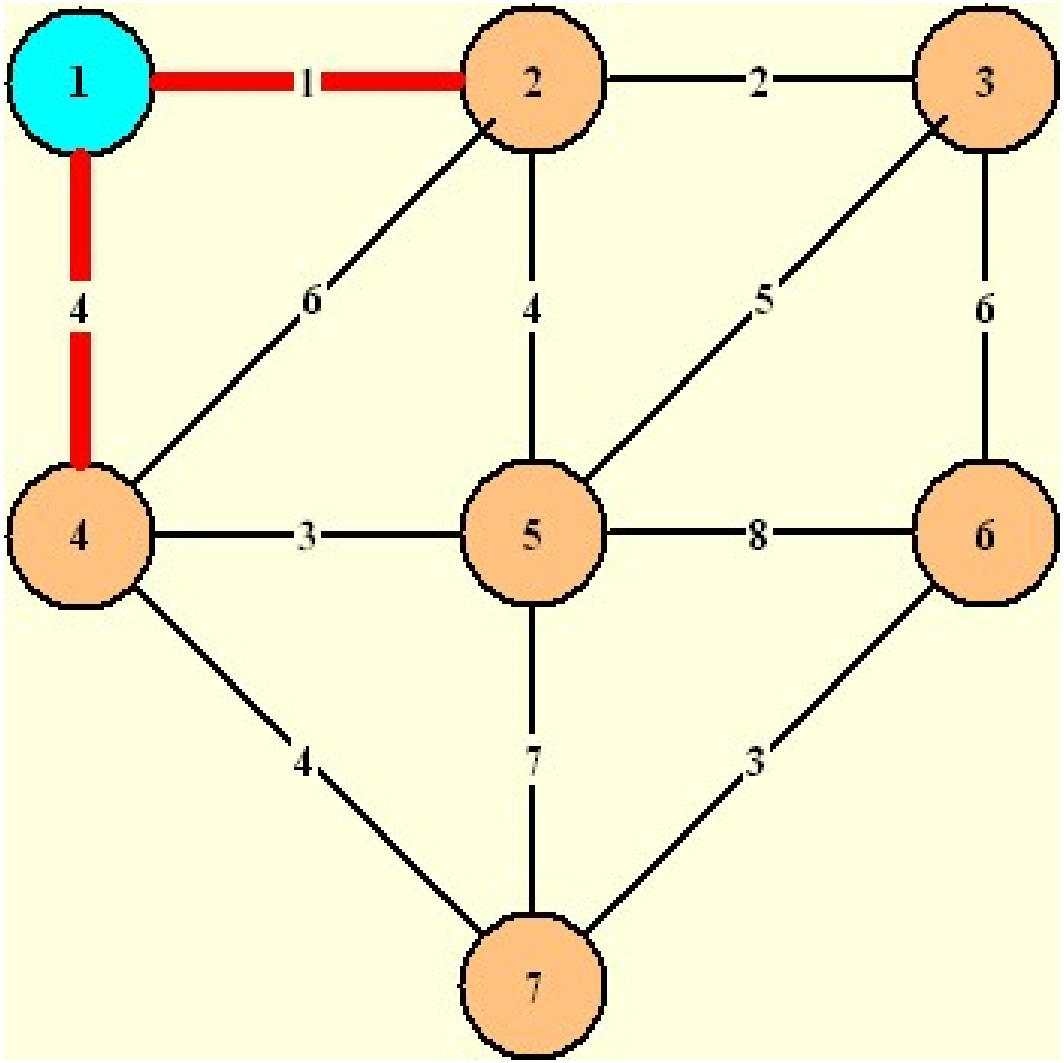


Step 1:
B is initialized with vertex 1 (could be any vertex).
T is empty.

B	T	Edges considered with weights	Selected edge
[1]	[ ]		

Prim example from:
http://www.ecf.utoronto.ca/apsc/courses/ece242/2003fall/notes/MCST_examples//notes_graph.htm

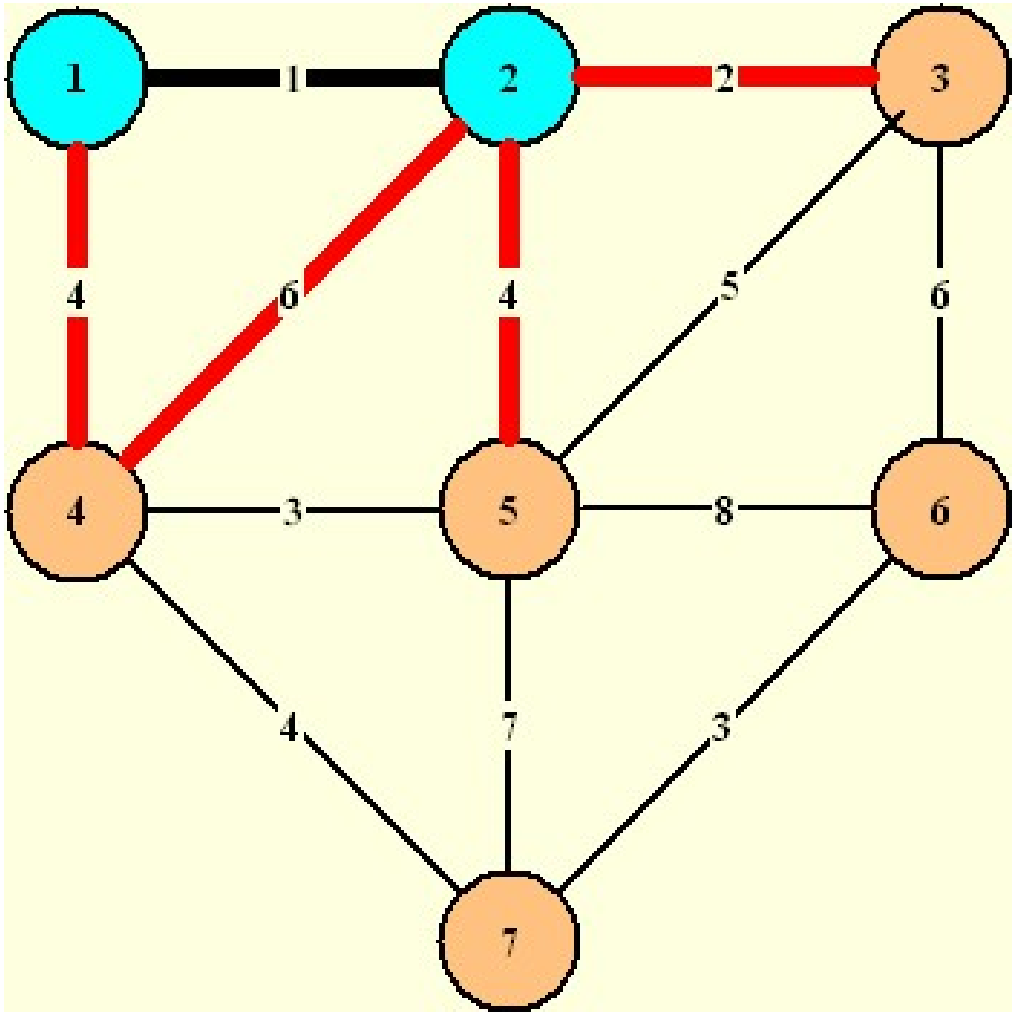
Example of Prim's Algorithm



Step 2:
Examine edges from vertex 1.
Choose (1,2) as least weight; add 2 to B; (1,2) to T.

B	T	Edges considered with weights	Selected edge
[1]	[ ]	(1,2) - 1 (1,4) - 4	(1,2)

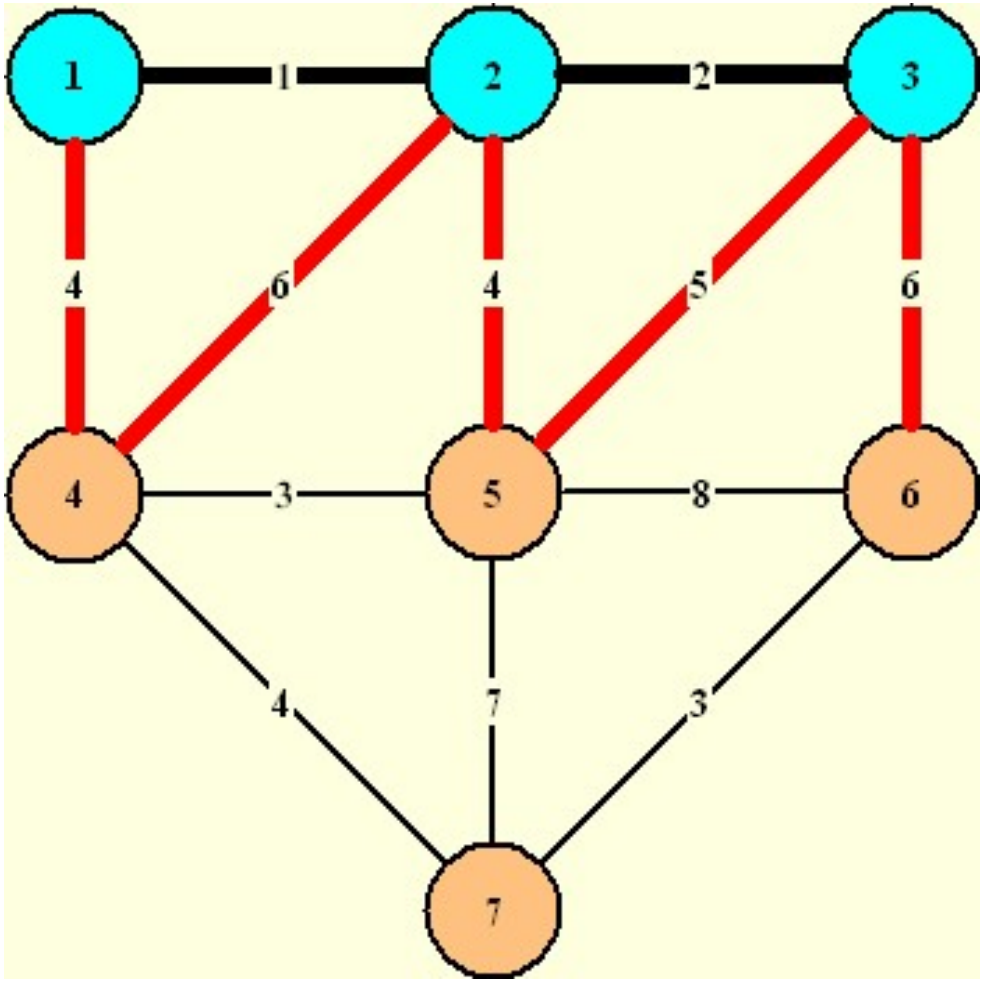
Example of Prim's Algorithm



Step 3:
Examine edges from vertices 1, 2.
Choose (2,3) as least weight; add 3 to B; (2,3) to T.

B	T	Edges considered with weights	Selected edge
[1]	[ ]		(1,2)
[1,2]	[(1,2)]	(1,4) - 4 (2,3) - 2 (2,4) - 6 (2,5) - 4	(2,3)

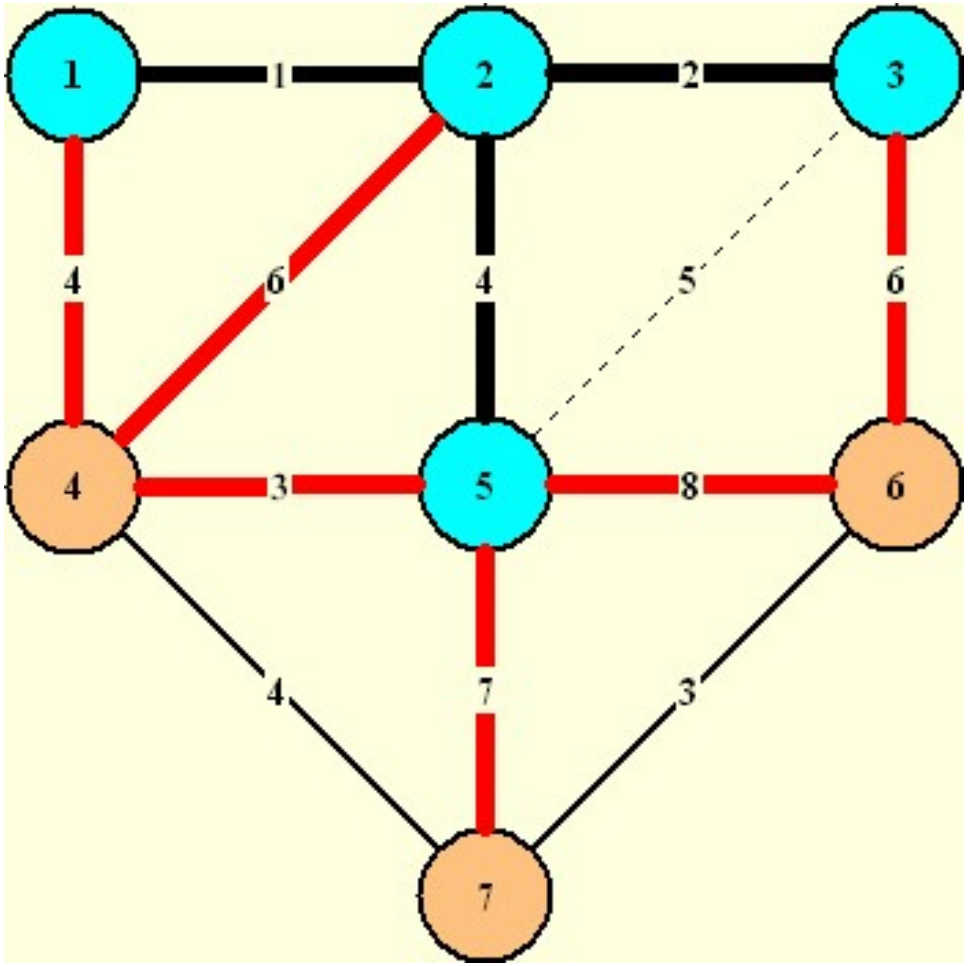
Example of Prim's Algorithm



Step 4:
Examine edges from vertices 1, 2, 3.
Choose (2,5) as least weight; add 5 to B; (2,5) to T. Could have chosen (1,4).

B	T	Edges considered with weights	Selected edge
[1]	[ ]		(1,2)
[1,2]	[(1,2)]		(2,3)
[1,2,3]	[(1,2), (2,3)]	(1,4) - 4 (2,4) - 6 (2,5) - 4 (3,5) - 5 (3,6) - 6	(2,5)

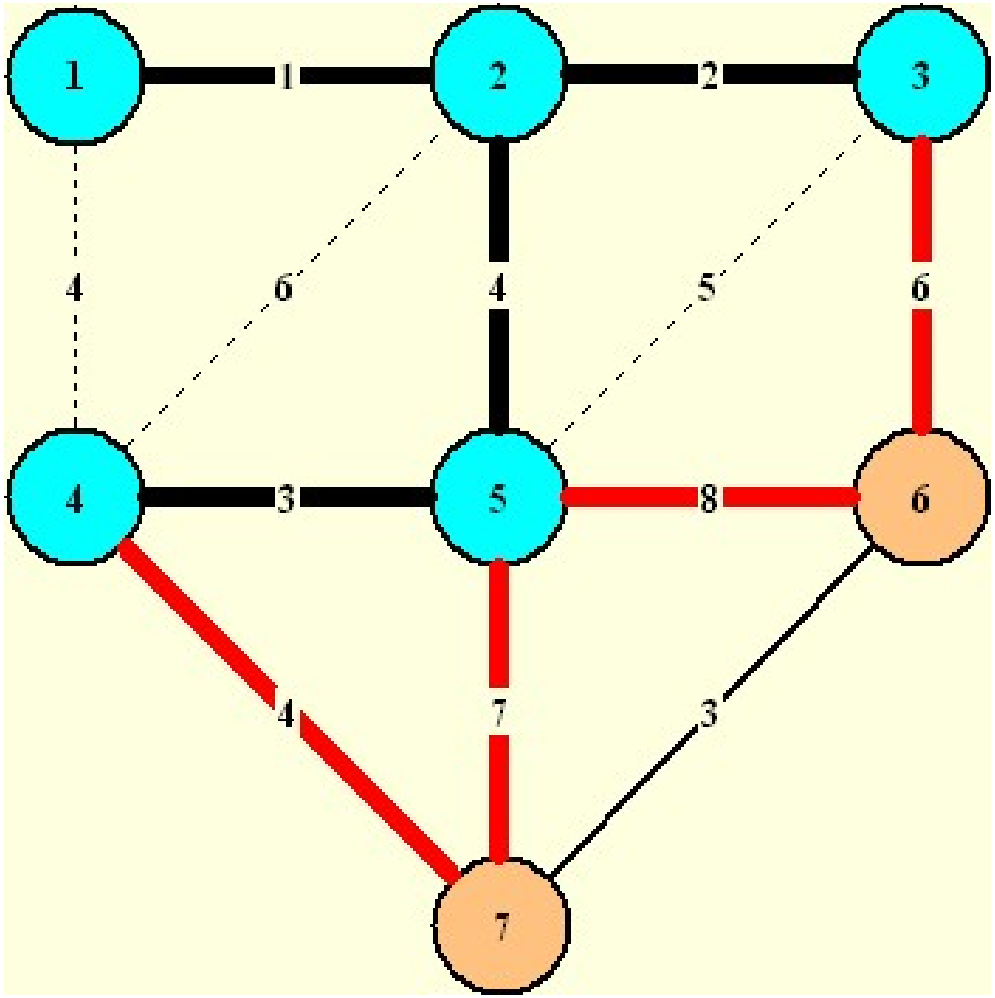
Example of Prim's Algorithm



Step 5:
Examine edges from vertices 1, 2, 3, 5.
Choose (4,5) as least weight; add 4 to B; (4,5) to T.

B	T	Edges considered with weights	Selected edge
[1]	[ ]		(1,2)
[1,2]	[(1,2)]		(2,3)
[1,2,3]	[(1,2), (2,3)]		(2,5)
[1,2,3,5]	[(1,2), (2,3), (2,5)]	(1,4) - 4 (2,4) - 6 (3,6) - 6 (4,5) - 3 (5,6) - 8 (5,7) - 7	(4,5)

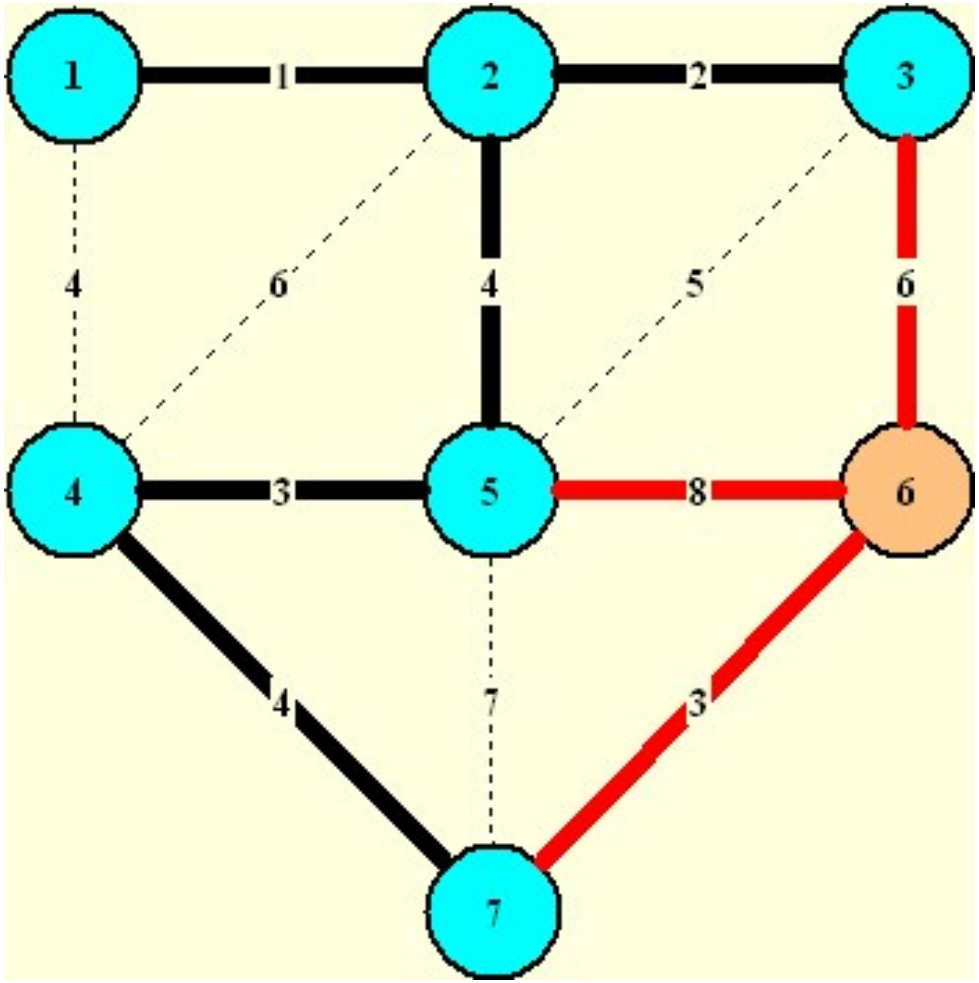
Example of Prim's Algorithm



Step 6:
Examine edges from vertices 1, 2, 3, 4, 5.
Choose (4,7) as least weight; add 7 to B; (4,7) to T.

B	T	Edges considered with weights	Selected edge
[1]	[ ]		(1,2)
[1,2]	[(1,2)]		(2,3)
[1,2,3]	[(1,2), (2,3)]		(2,5)
[1,2,3,5]	[(1,2), (2,3), (2,5)]		(4,5)
[1,2,3,4,5]	[(1,2), (2,3), (2,5), (4,5)]	(3,6) - 6 (4,7) - 4 (5,6) - 8 (5,7) - 7	(4,7)

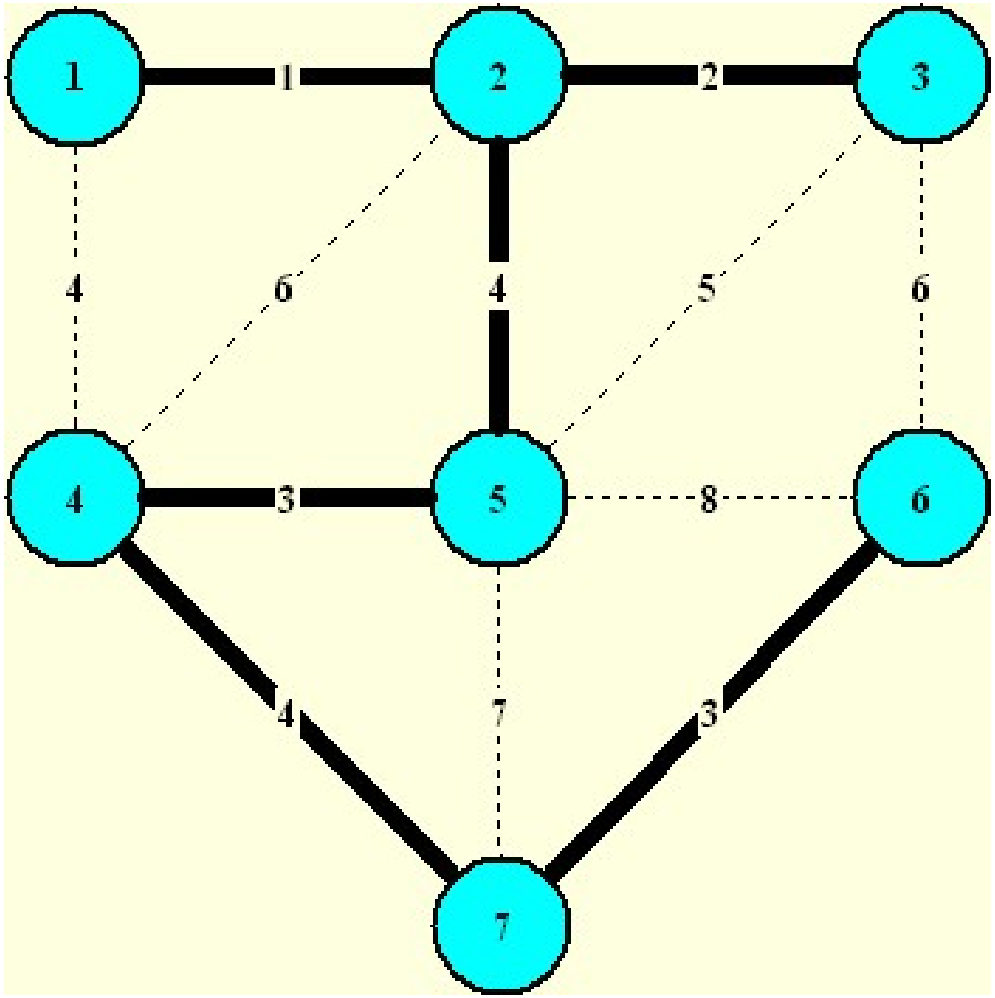
Example of Prim's Algorithm



Step 7:
Examine edges from vertices 1, 2, 3, 4, 5, 7.
Choose (6,7) as least weight; add 6 to B; (6,7) to T.

B	T	Edges considered with weights	Selected edge
[1]	[ ]		(1,2)
[1,2]	[(1,2)]		(2,3)
[1,2,3]	[(1,2), (2,3)]		(2,5)
[1,2,3,5]	[(1,2), (2,3), (2,5)]		(4,5)
[1,2,3,4, 5]	[(1,2), (2,3), (2,5), (4,5)]		(4,7)
[1,2,3,4, 5,7]	[(1,2), (2,3), (2,5), (4,5), (4,7)]	(3,6) - 6 (5,6) - 8 (6,7) - 3	(6,7)

Example of Prim's Algorithm



Step 8:
Set B contains all nodes in N. Edges in T form a minimal spanning tree. Halt.

B	T	Edges considered with weights	Selected edge
[1]	[ ]		(1,2)
[1,2]	[(1,2)]		(2,3)
[1,2,3]	[(1,2), (2,3)]		(2,5)
[1,2,3,5]	[(1,2), (2,3), (2,5)]		(4,5)
[1,2,3,4,5]	[(1,2), (2,3), (2,5), (4,5)]		(4,7)
[1,2,3,4,5,7]	[(1,2), (2,3), (2,5), (4,5), (4,7)]		(6,7)
[1,2,3,4,5,6,7]	[(1,2), (2,3), (2,5), (4,5), (4,7), (6,7)]	B = N, so halt	

Conclusion

Finding minimum spanning trees is an important process.

Running times of the two algorithms for finding MSTs:

Kruskal's = O(E lg E)

Prim's = O(E lg V)

(asymptotically the same)