CS2040S
Data Structures and Algorithms
(e-learning edition)

All about minimum spanning trees…

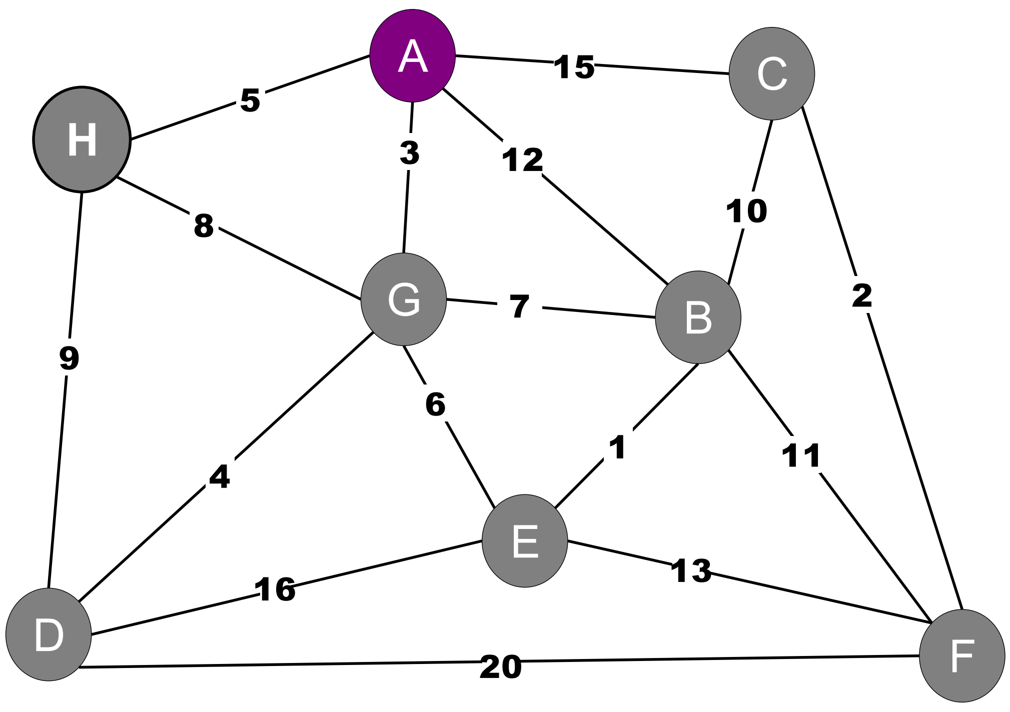
Roadmap

Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- **Prim’s Algorithm**
- Kruskal’s Algorithm
- Boruvka’s Algorithm
- Variations

Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

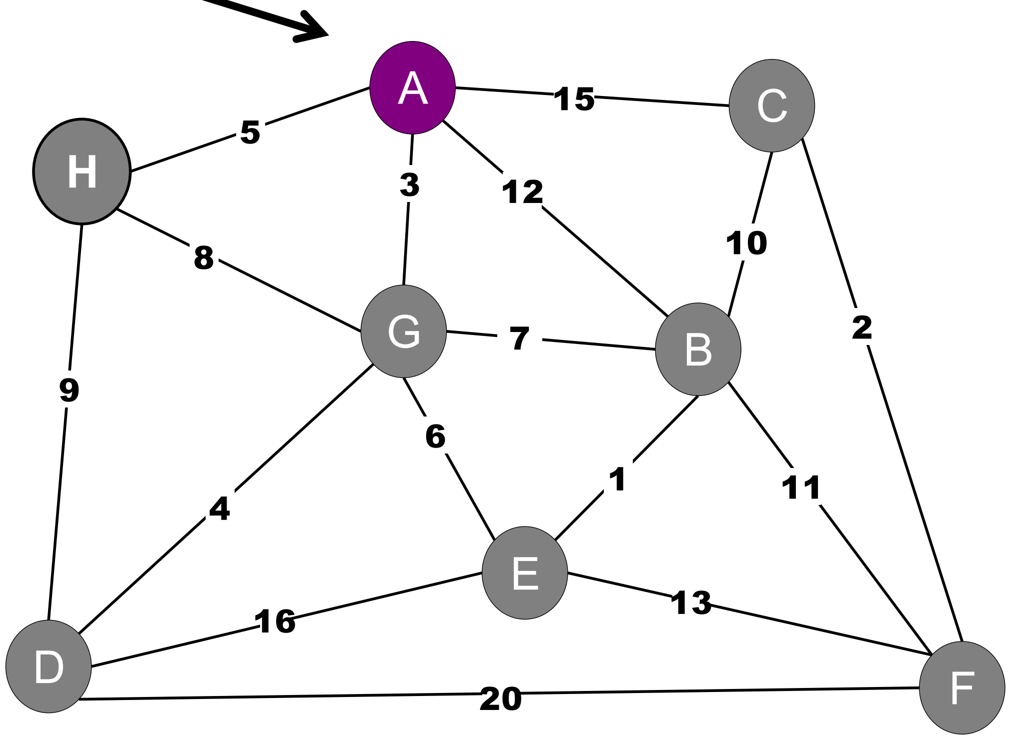


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}

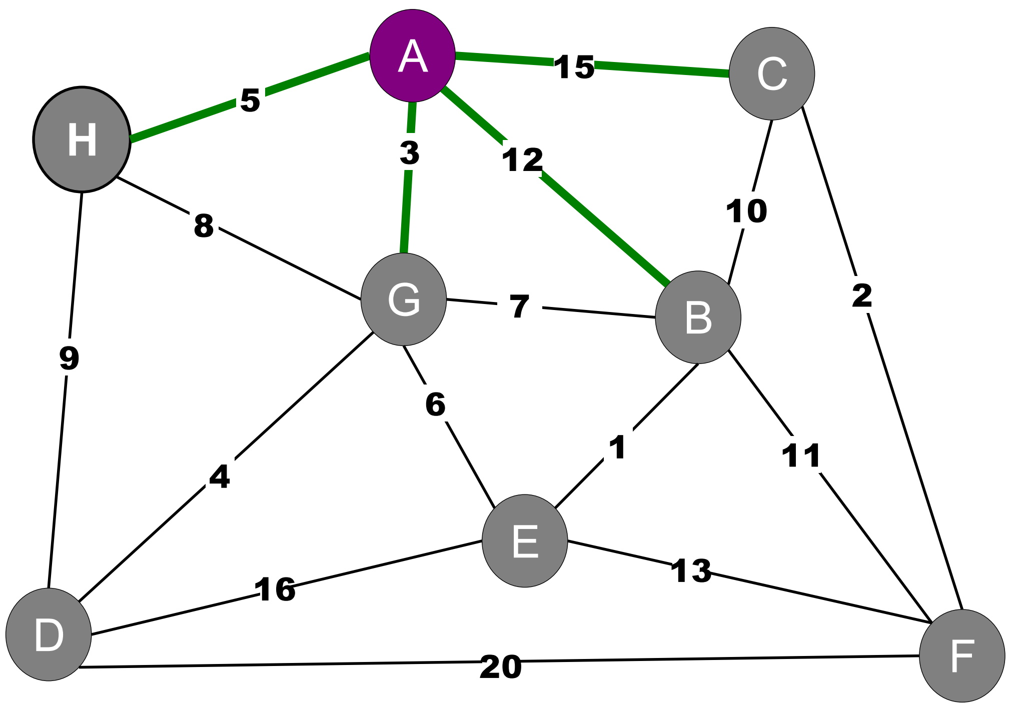


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Identify cut: {S, V–S}

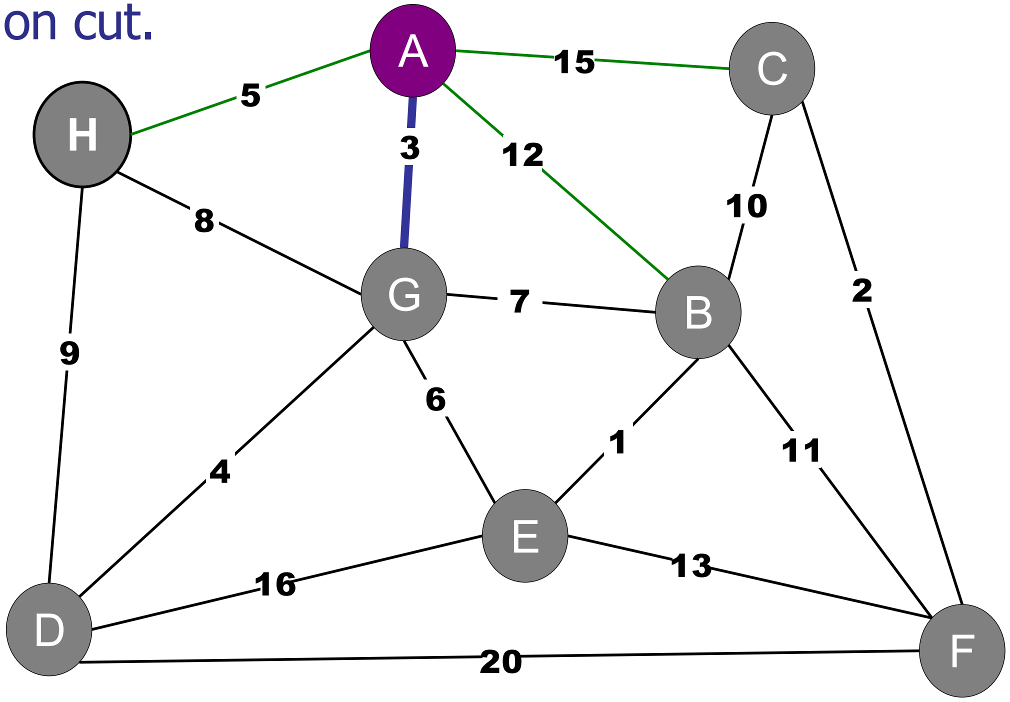


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Identify cut: {S, V–S}
- Find minimum weight edge on cut.

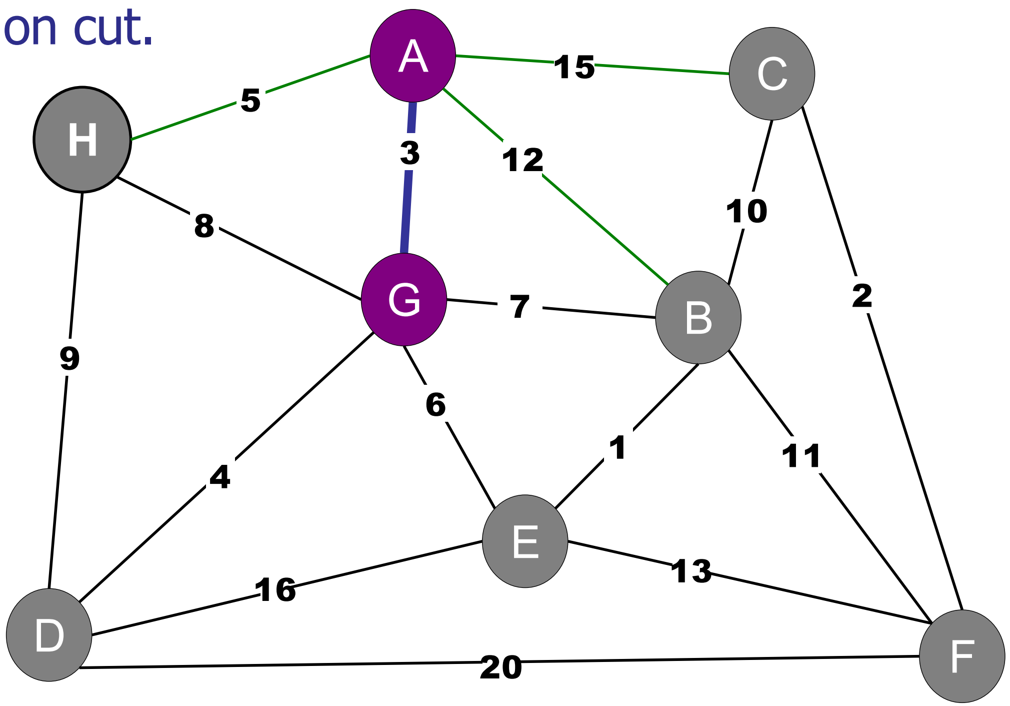


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Identify cut: {S, V–S}
- Find minimum weight edge on cut.
- Add new node to S.

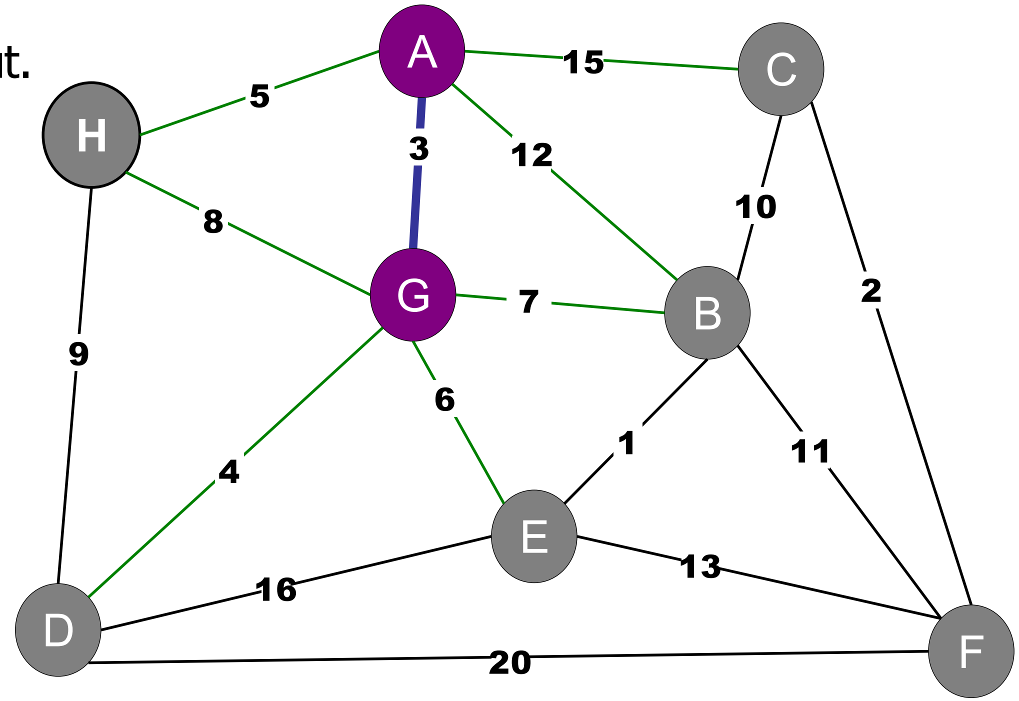


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.

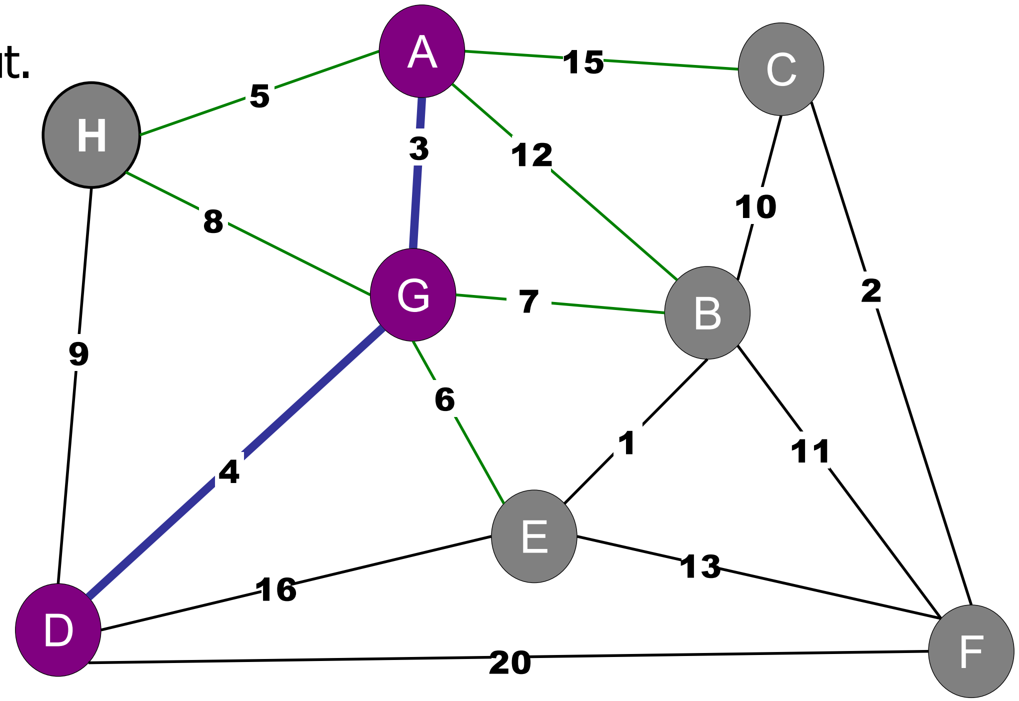


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.

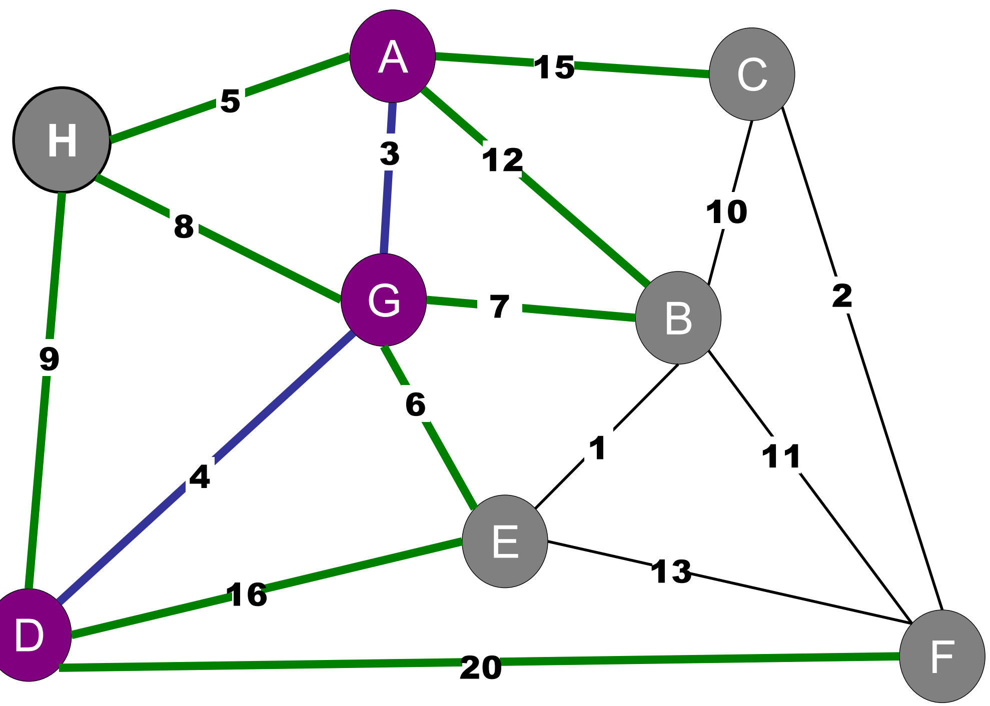


Prim's Algorithm

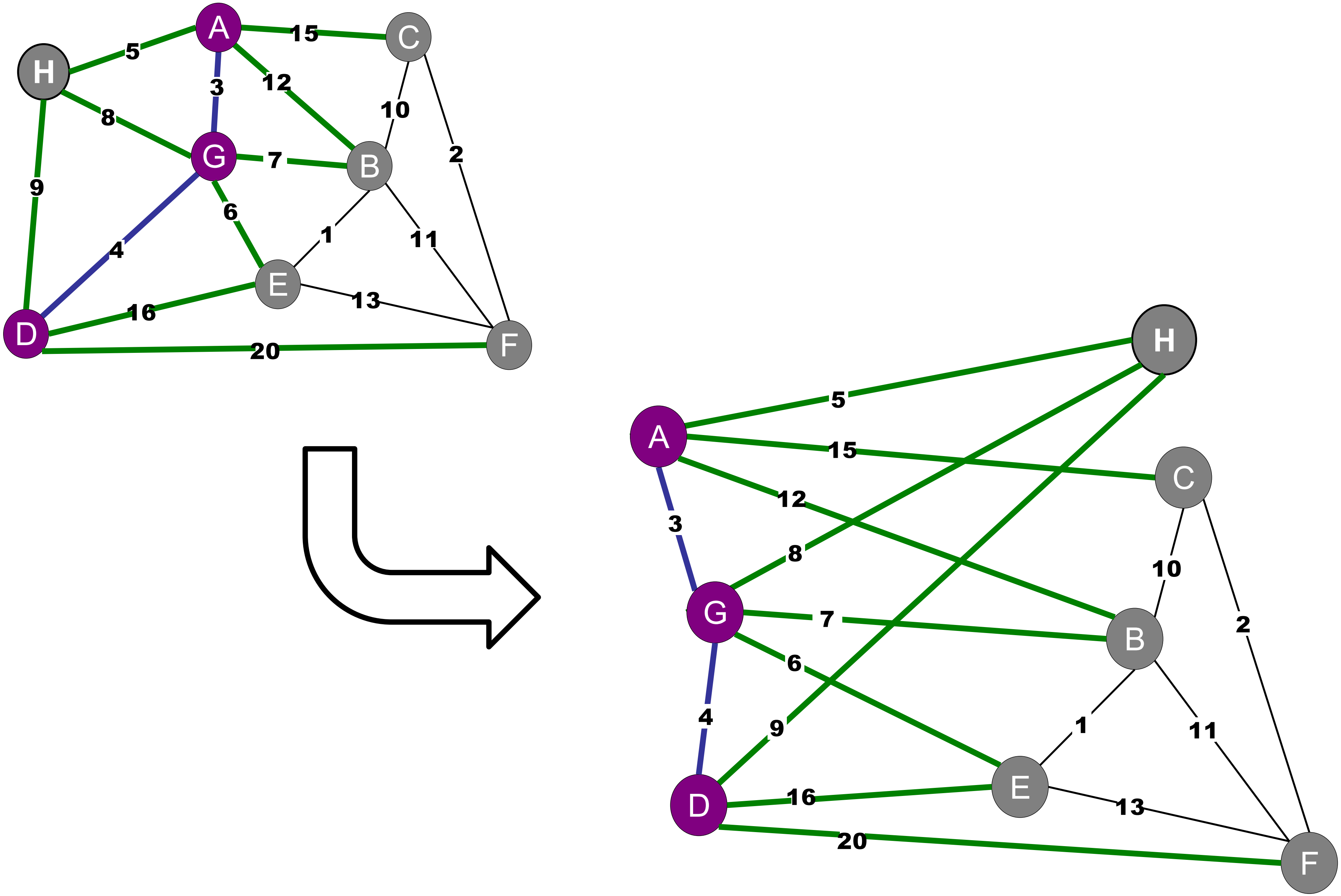
Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.



Prim's Algorithm



How do we find the lightest edge on a cut?

✓ 1. Priority Queue
2. Union-Find
3. Max-flow / Min-cut
4. BFS
5. DFS

Prim's Algorithm: Initialization

// Initialize priority queue

```
PriorityQueue pq = new PriorityQueue();
for (Node v : G.V()) {
        pq.insert(v, INFTY);
}
pq.decreaseKey(start, 0);
```

// Initialize set S

```
HashSet<Node> S = new HashSet<Node>();
S.put(start);
```

// Initialize parent hash table

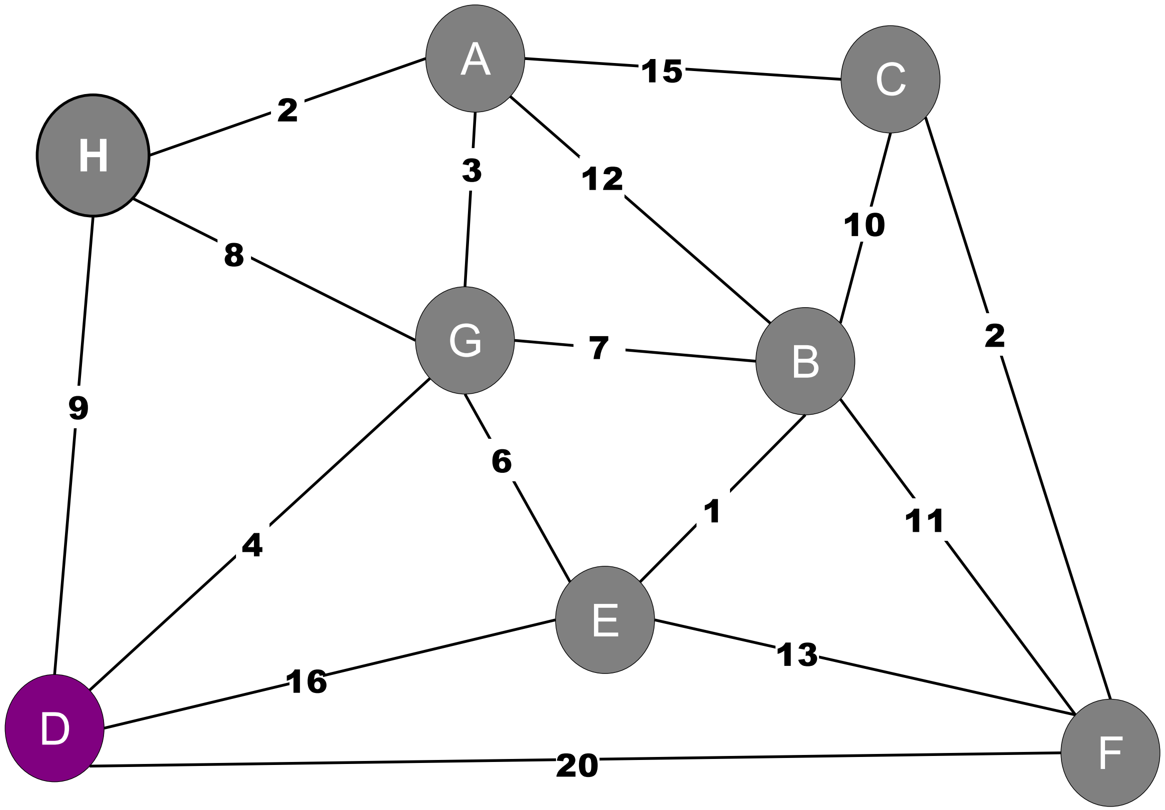
```
HashMap<Node,Node> parent = new HashMap<Node,Node>();
parent.put(start, null);
```

Prim's Algorithm

```
while (!pq.isEmpty()){
      Node v = pq.deleteMin();
      S.put(v);
      for each (Edge e : v.edgeList()){
            Node w = e.otherNode(v);
            if (!S.get(w)) {
                  pq.decreaseKey(w, e.getWeight());
                  parent.put(w, v);
            }
      }
}
```

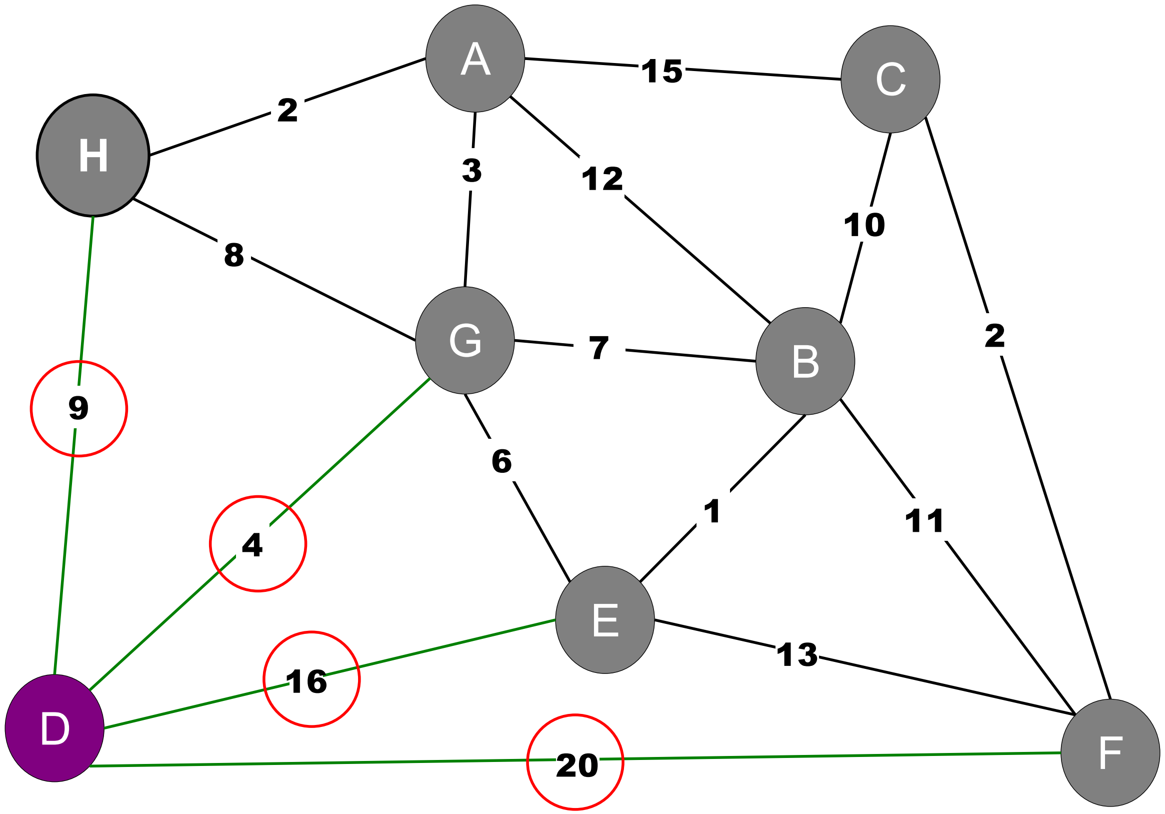
if wt decreased

Prim’s Example



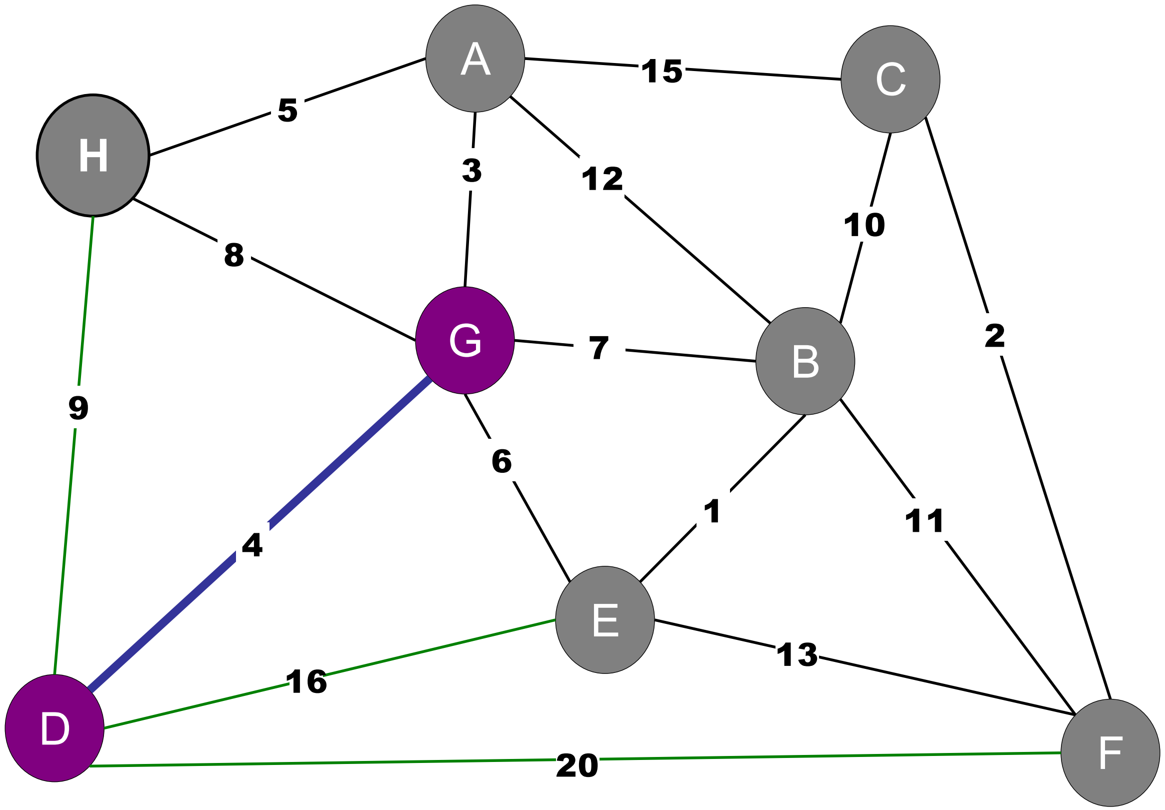
Vertex	Weight
D	0

Prim’s Example



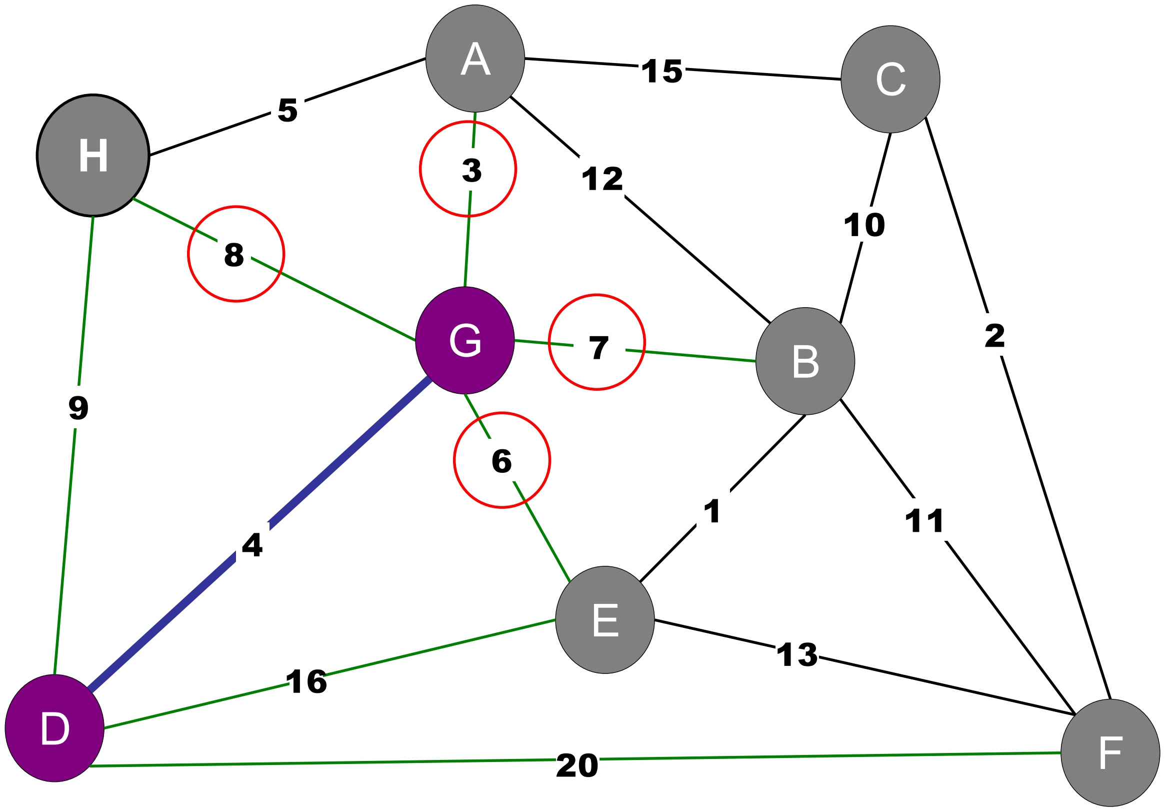
Vertex	Weight
G	4
H	9
E	16
F	20

Prim's Example



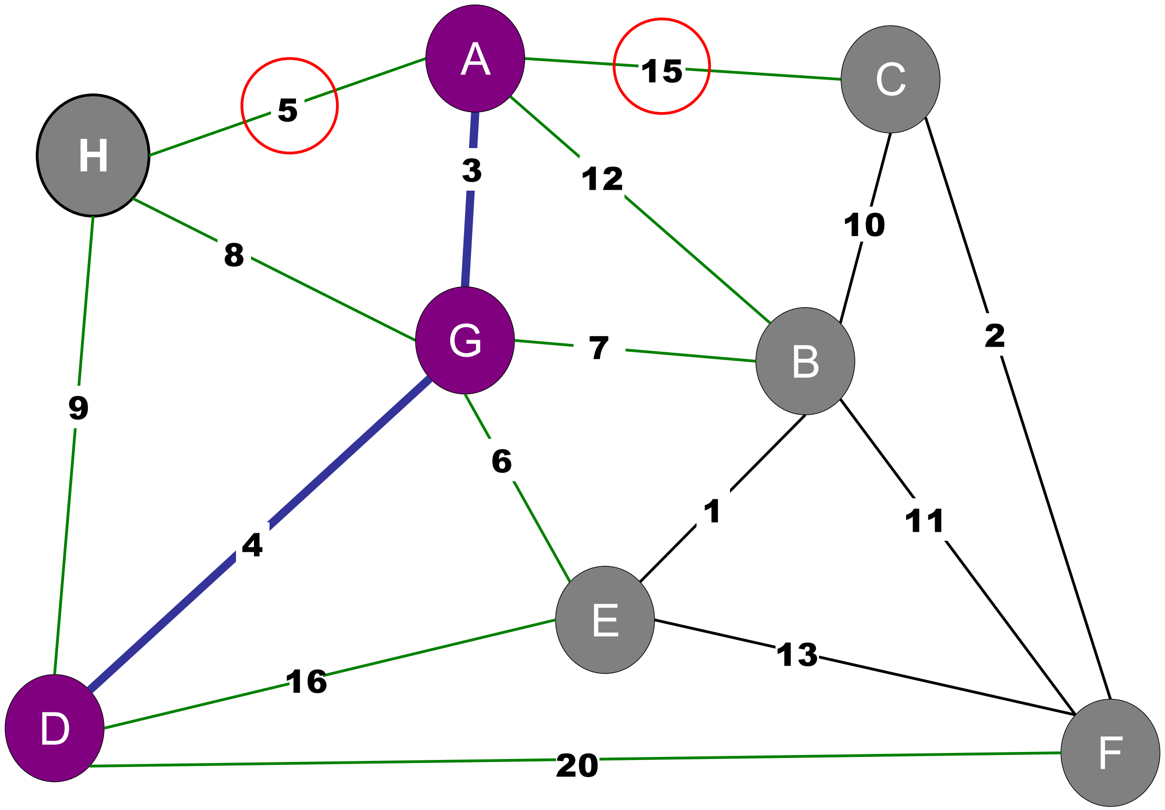
Vertex	Weight
H	9
E	16
F	20

Prim's Example



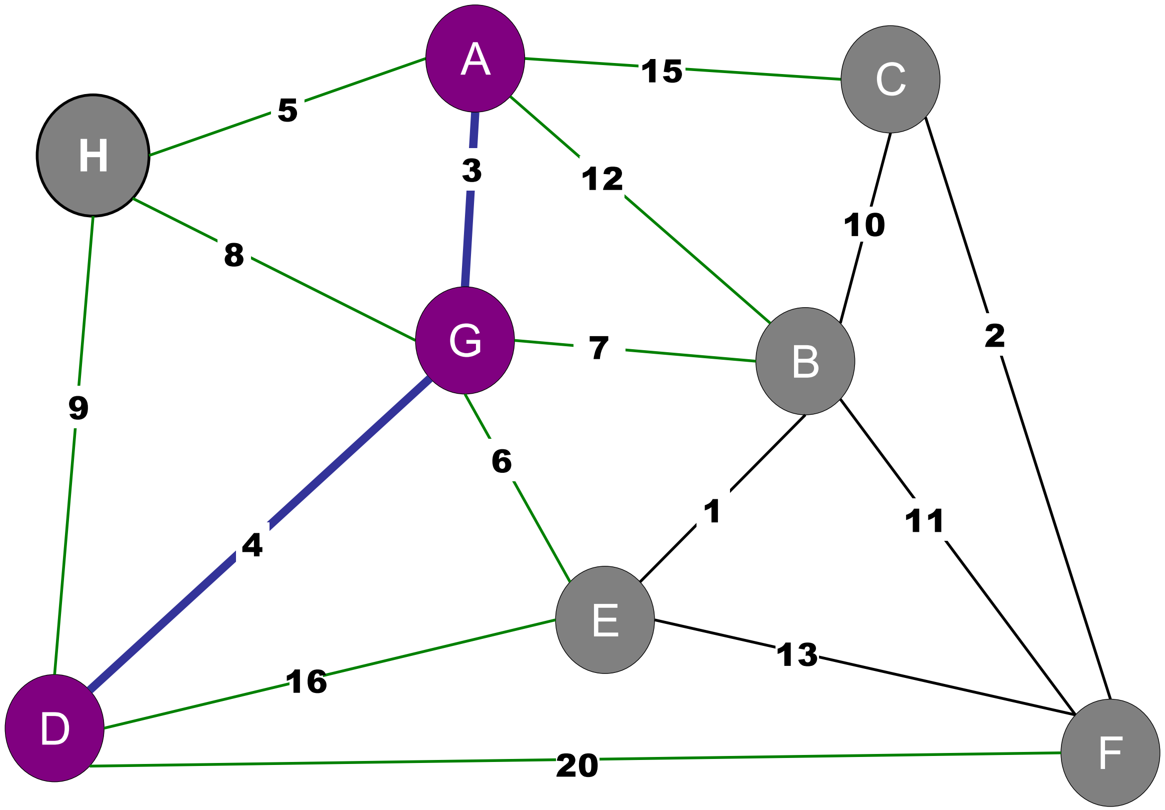
Vertex	Weight
A	3
E	16->6
B	7
H	9->8
F	20

Prim's Example



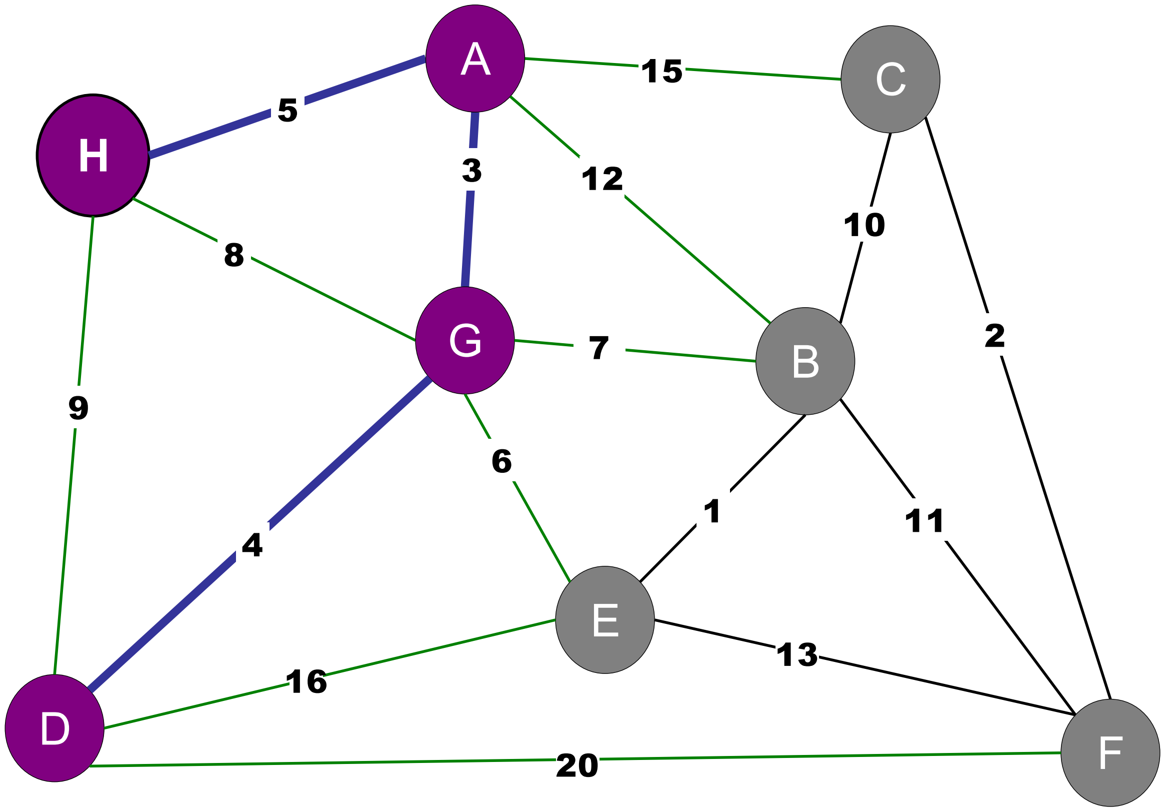
Vertex	Weight
H	**8->5**
E	6
B	7
C	**15**
F	20

Prim's Example



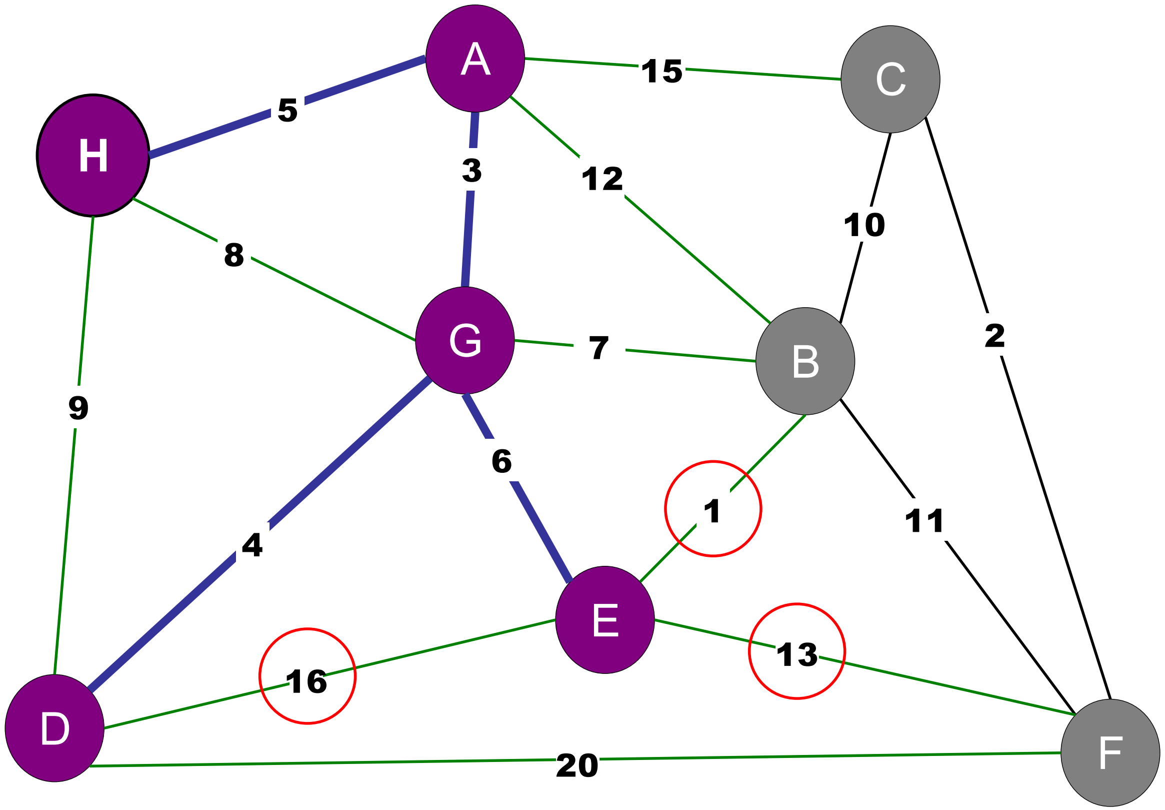
Vertex	Weight
H	5
E	6
B	7
C	15
F	20

Prim's Example



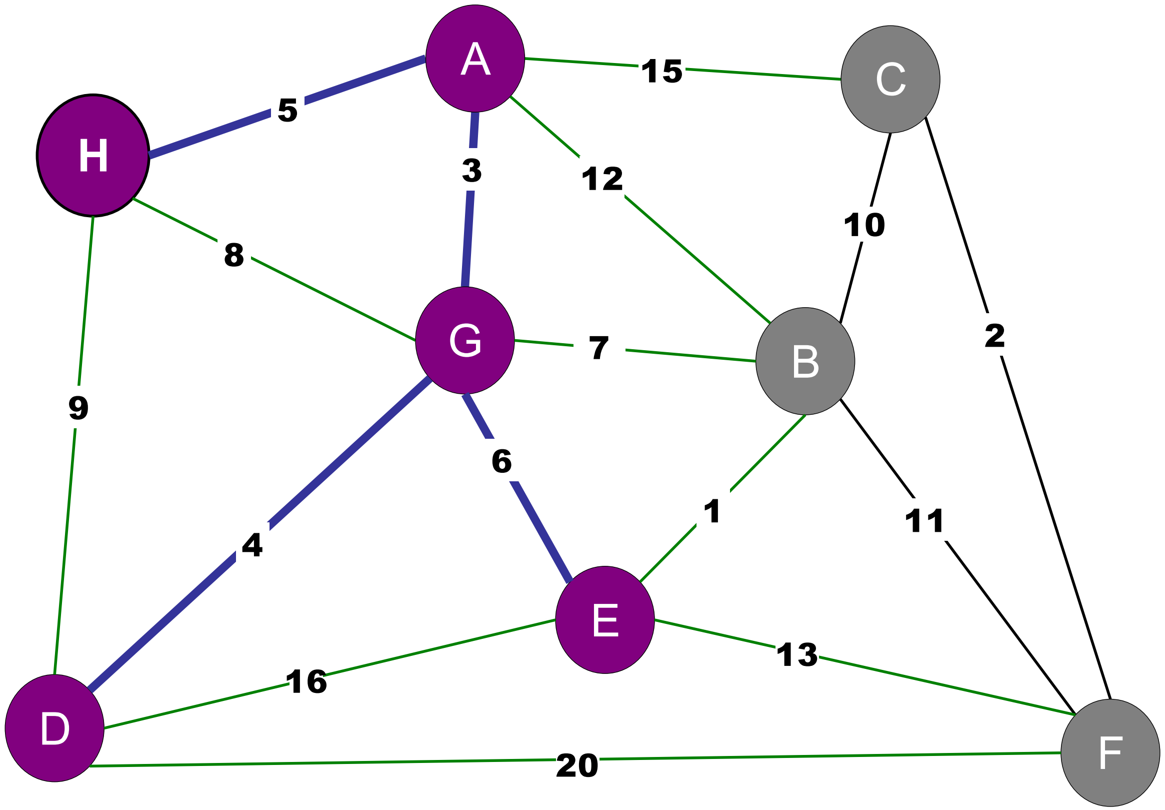
Vertex	Weight
E	6
B	7
C	15
F	20

Prim's Example



Vertex	Weight
B	**7->1**
C	15
F	**20->13**

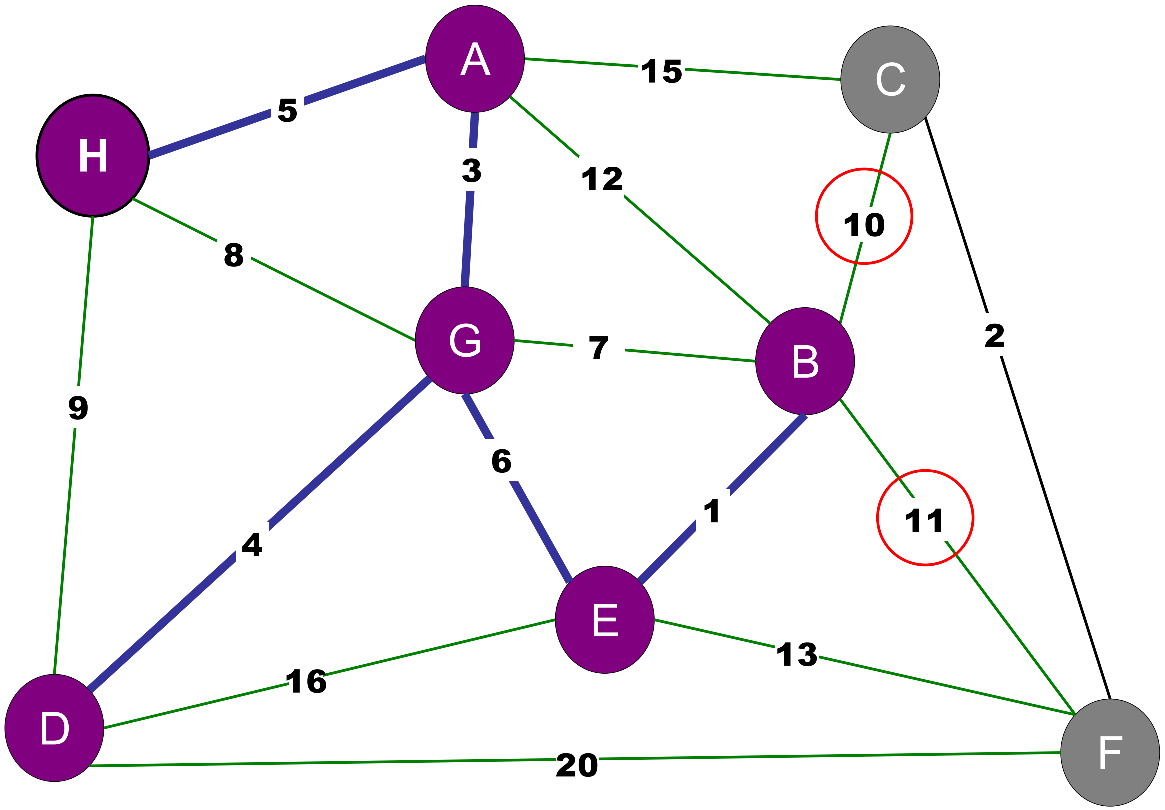
Prim’s Example



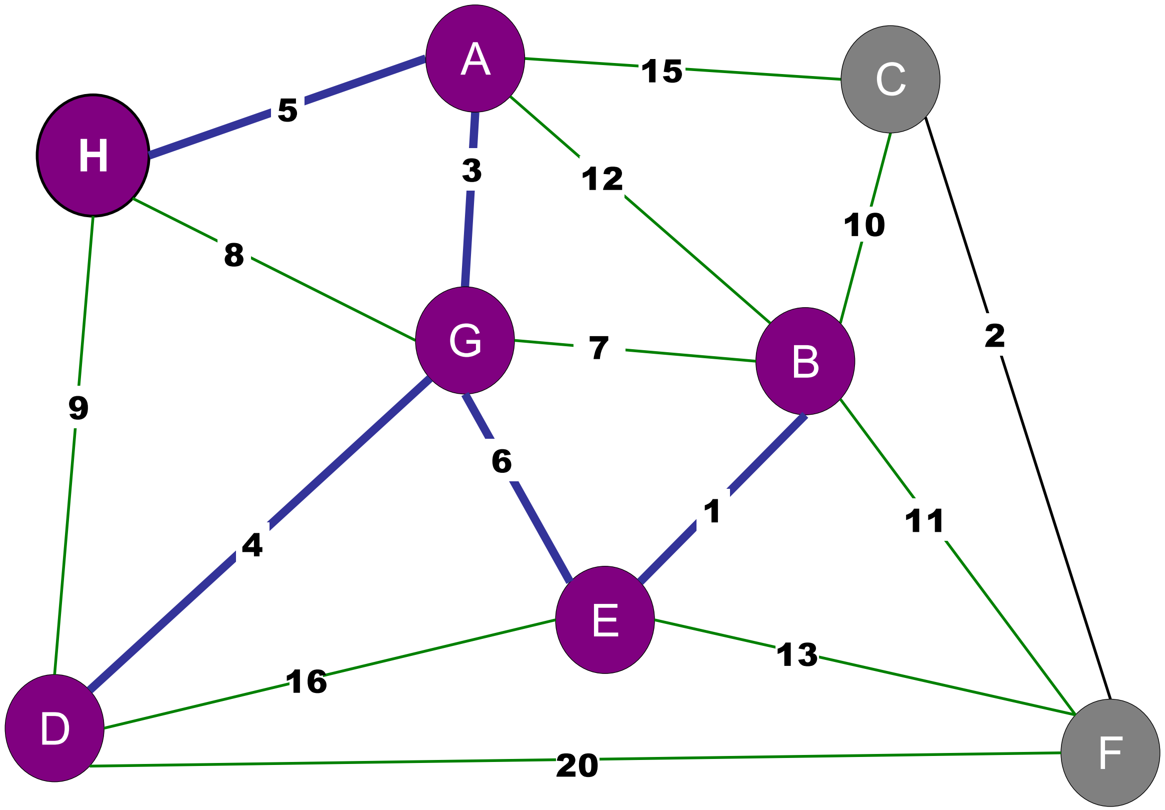
Vertex	Weight
B	1
C	15
F	13

Prim’s Example

Vertex	Weight
C	**15->10**
F	**13->11**



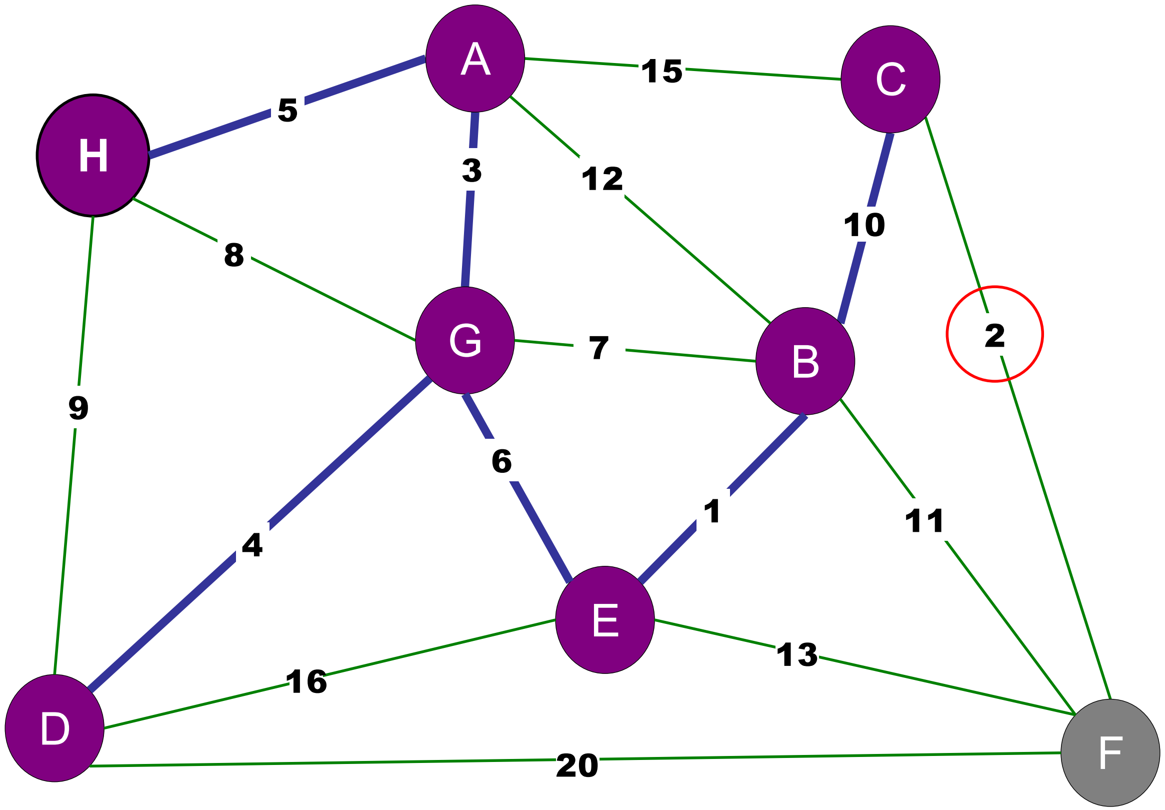
Prim's Example



Vertex	Weight
C	10
F	11

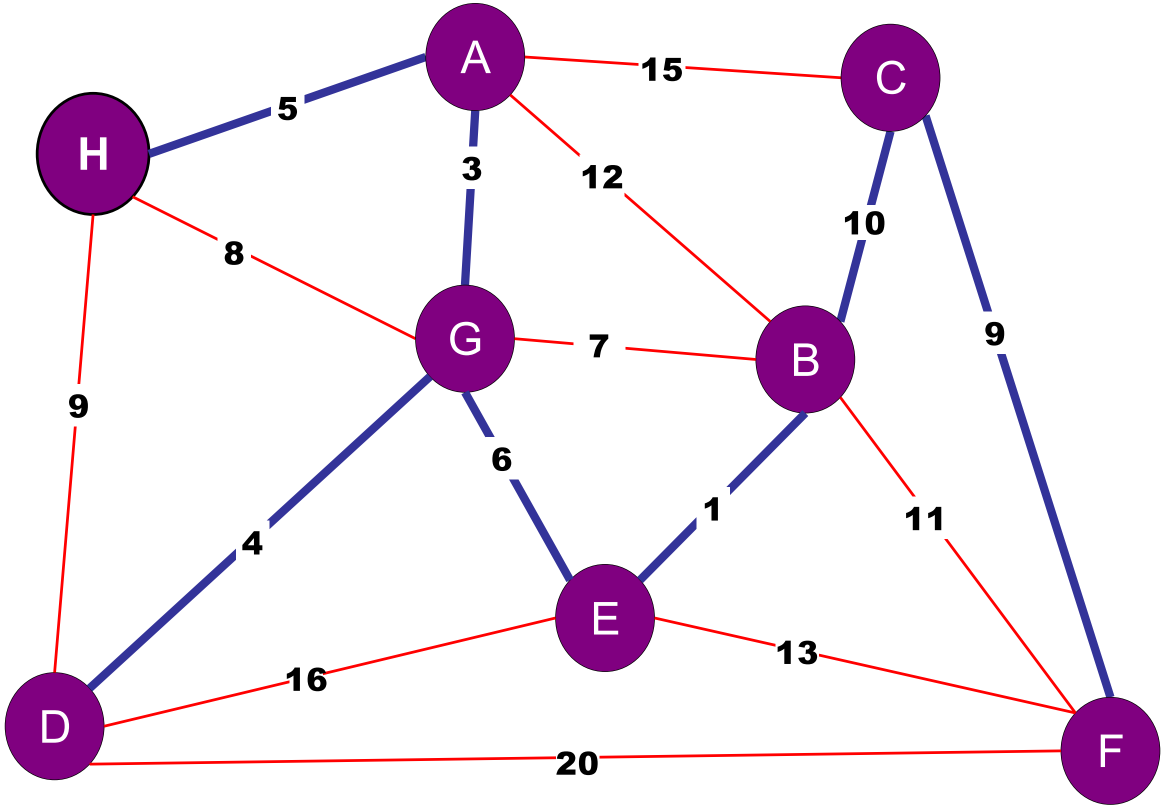
Prim’s Example

Vertex	Weight
F	11->2



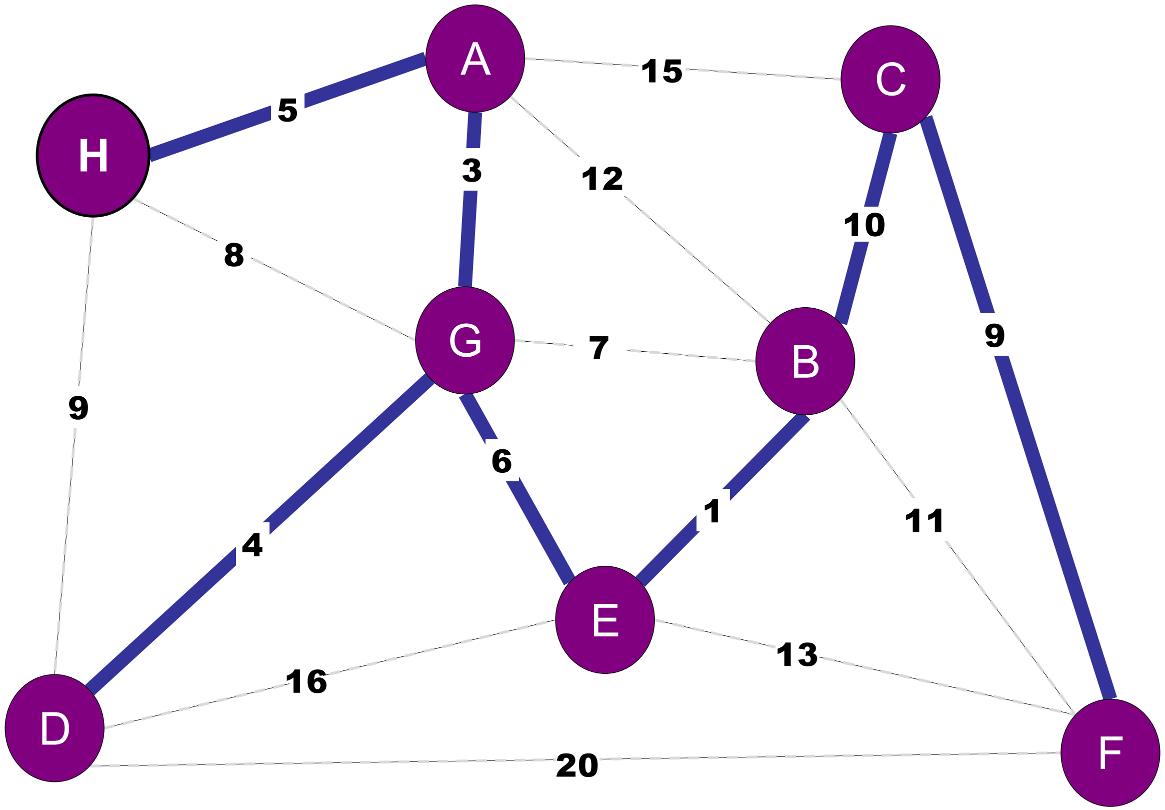
Prim’s Example

Vertex	Weight



Prim's Example

Vertex	Weight

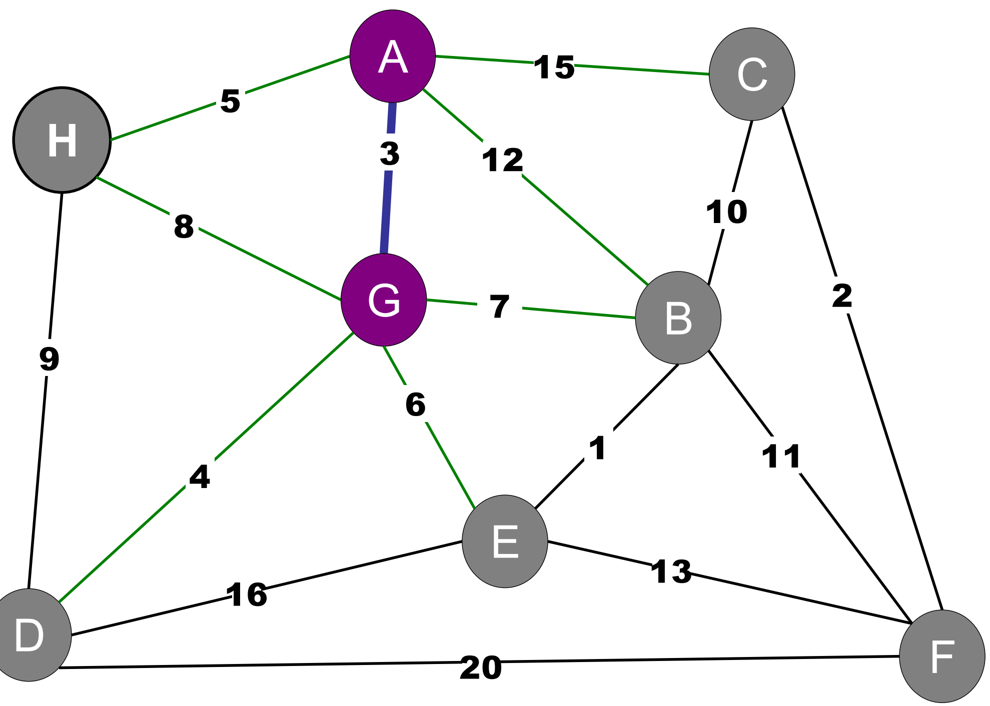


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.



Proof:

- Each added edge is the lightest on some cut.
- Hence each edge is in the MST.

What is the running time of Prim's Algorithm, using a binary heap?

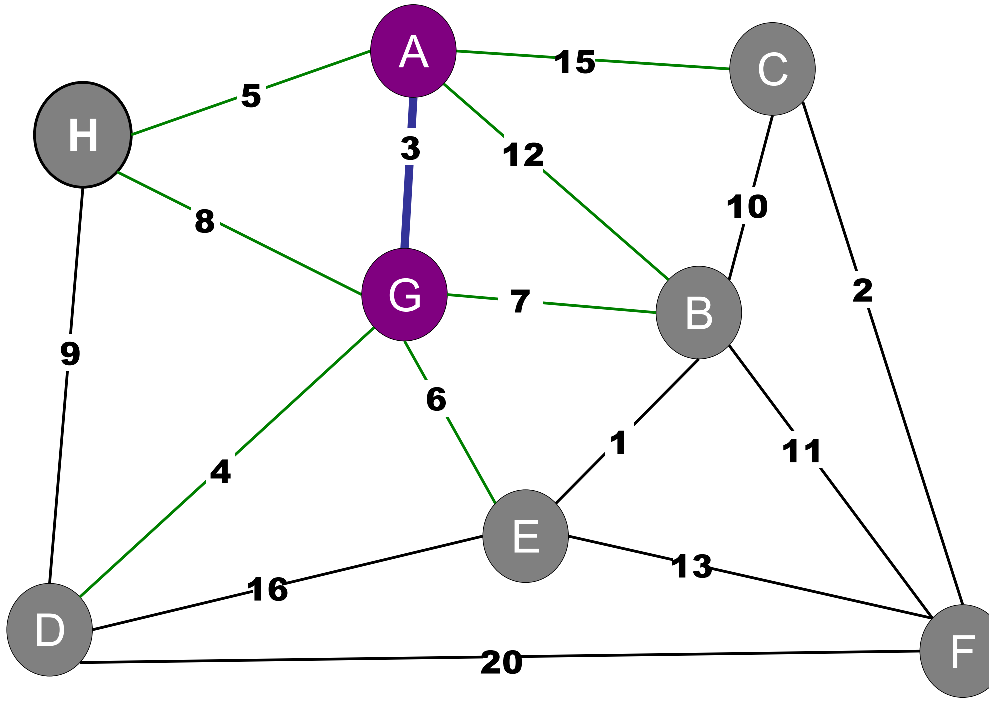
1. O(V)
2. O(E)
3. O(E log V)
4. O(V log E)
5. O(EV)

Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.



Analysis:

- Each vertex added/removed once from the priority queue: O(V log V)
- Each edge => one decreaseKey: O(E log V).

Two Algorithms

Prim's Algorithm.

Basic idea:

- Maintain a set of visited nodes.
- Greedily grow the set by adding node connected via the lightest edge.
- Use Priority Queue to order nodes by edge weight.

Dijkstra's Algorithm.

Basic idea:

- Maintain a set of visited nodes.
- Greedily grow the set by adding neighboring node that is closest to the source.
- Use Priority Queue to order nodes by distance.

Roadmap

Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- Prim’s Algorithm
- **Kruskal’s Algorithm**
- Boruvka’s Algorithm
- Variations

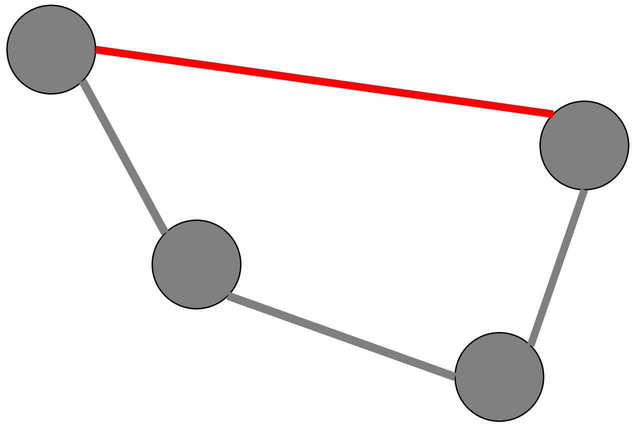
Generic MST Algorithm

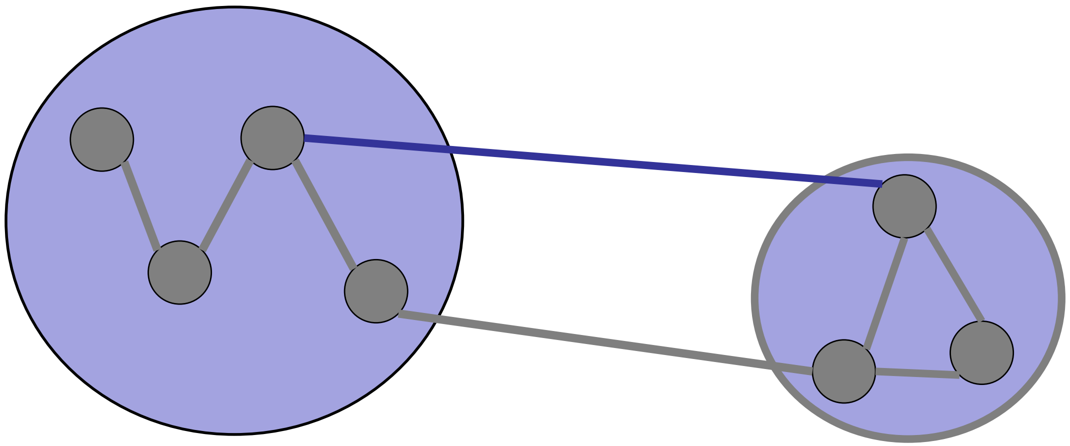
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

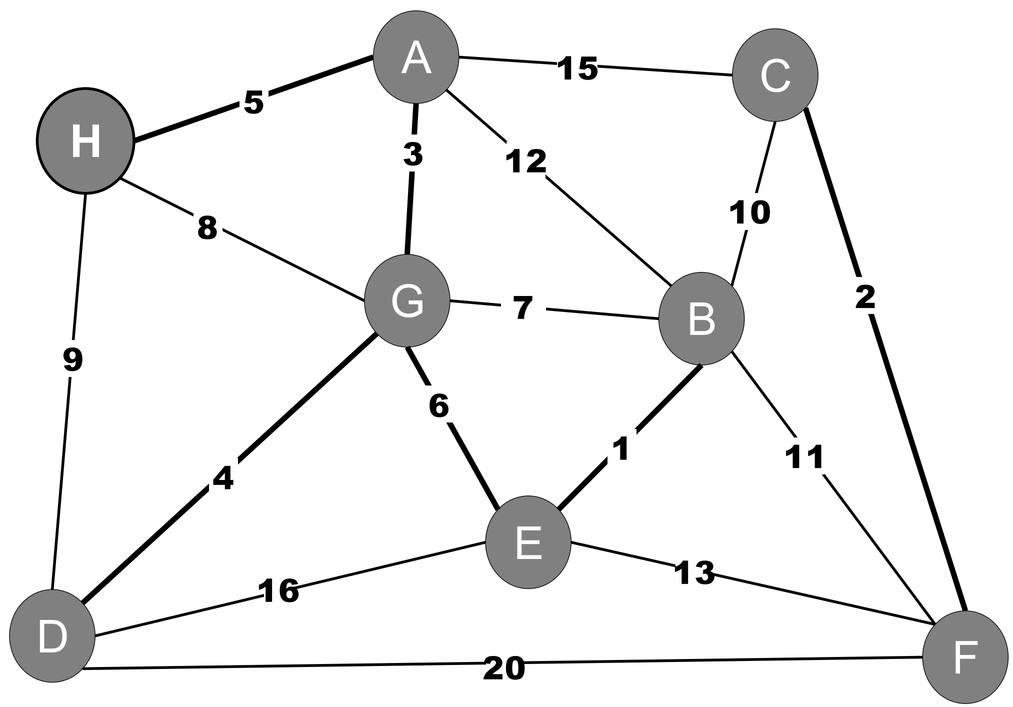
until no more edges can be colored.





Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

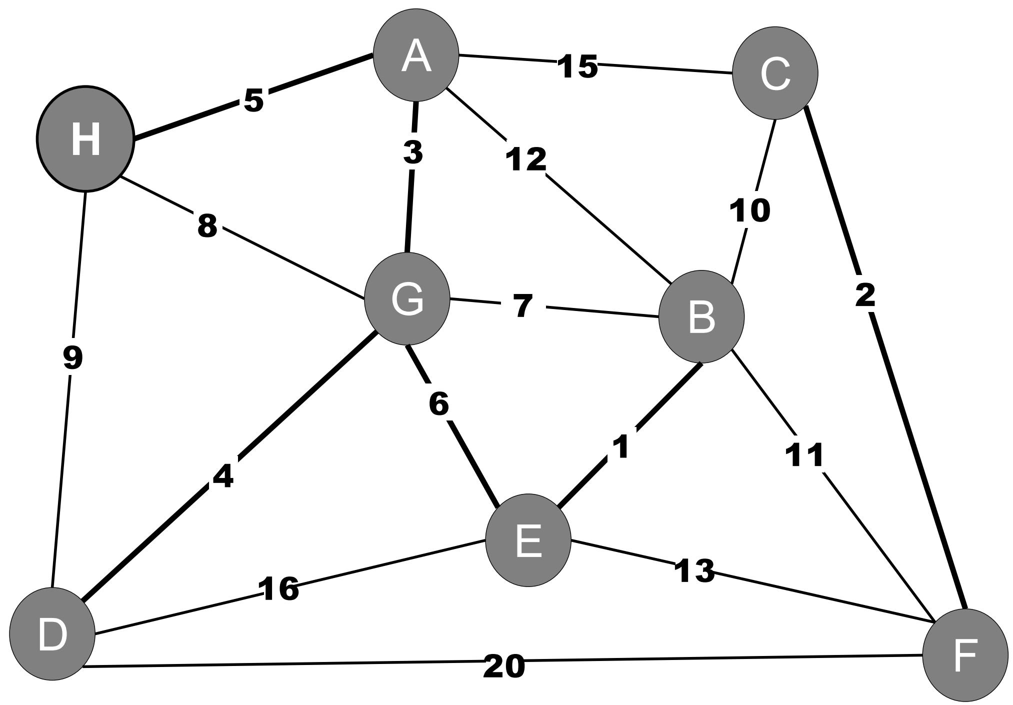


Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

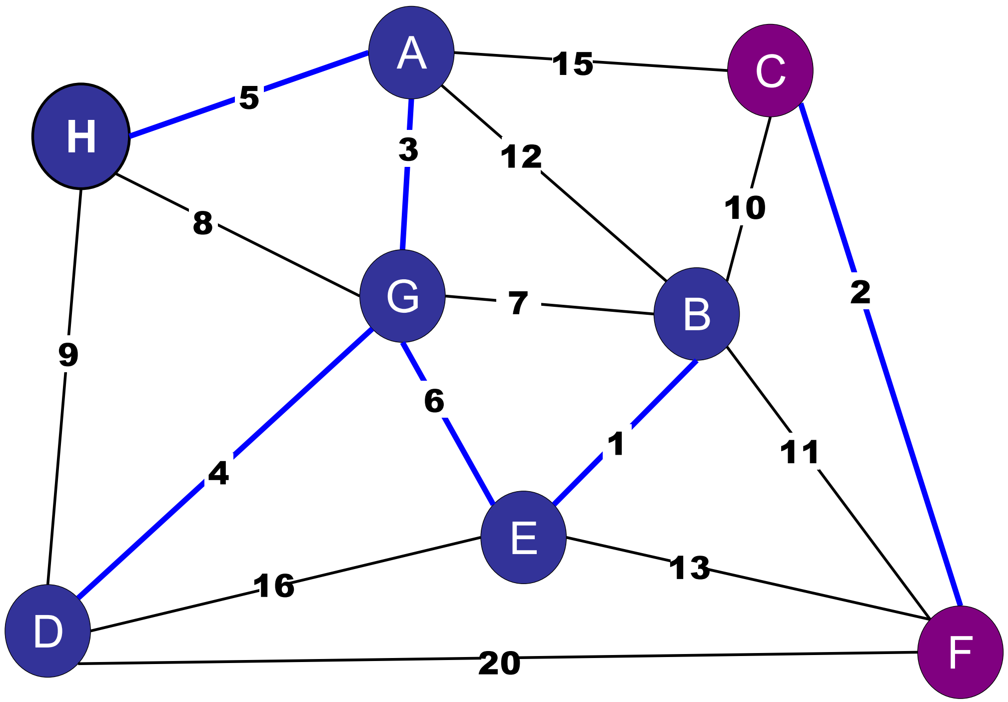


Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

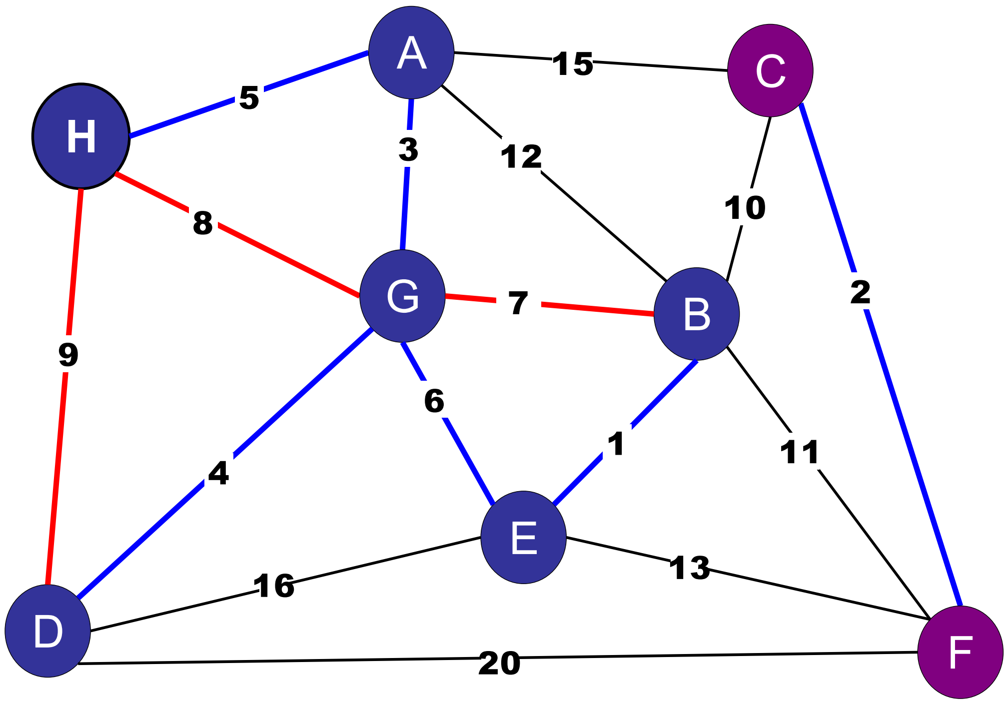


Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.



Kruskal's Algorithm

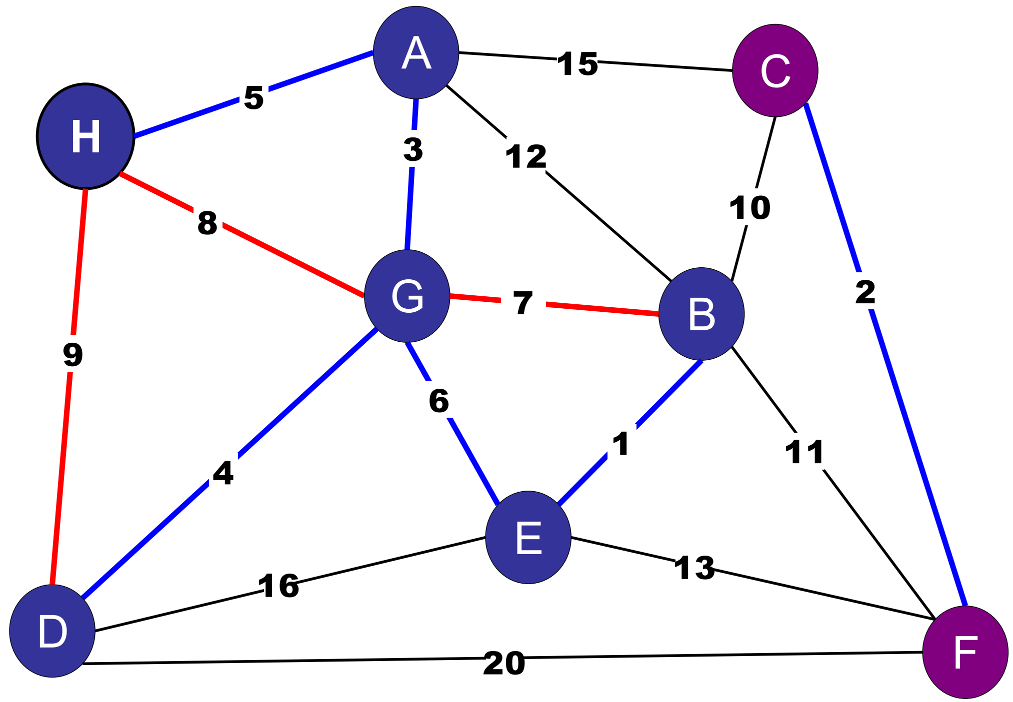
Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

Data structure:

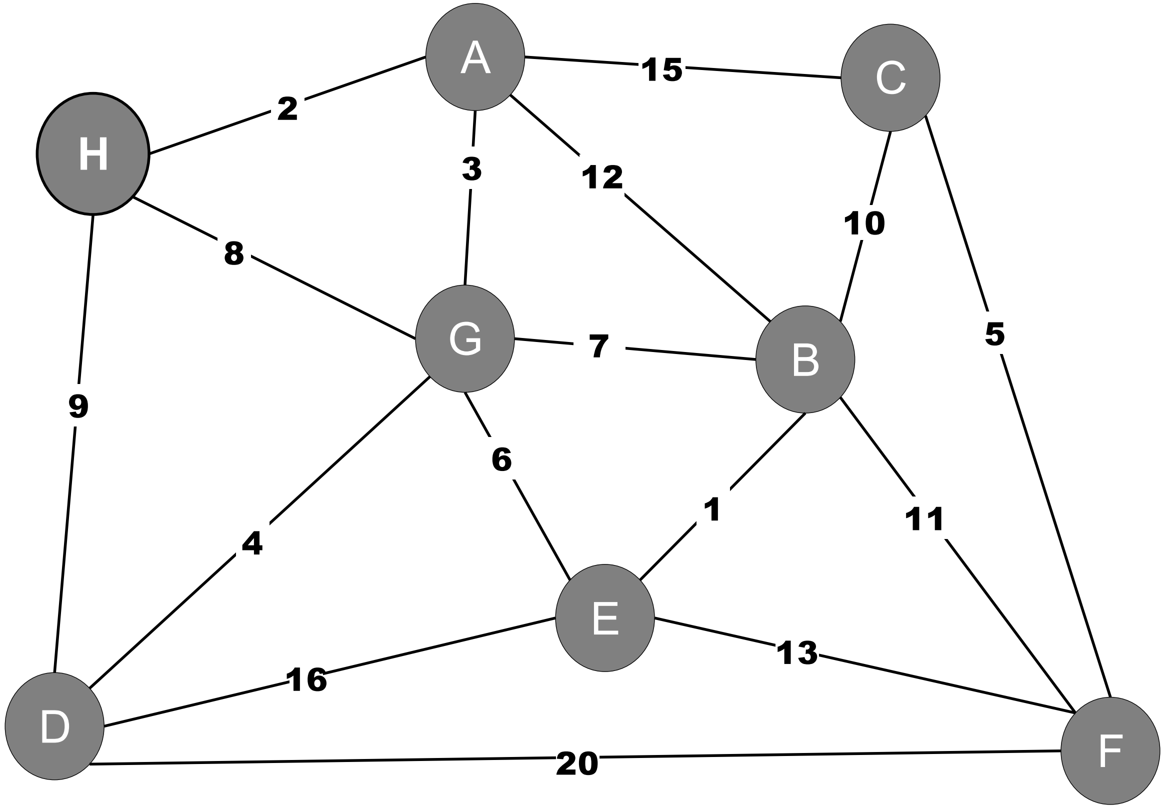
- Union-Find
- Connect two nodes if they are in the same blue tree.



Kruskal's Algorithm

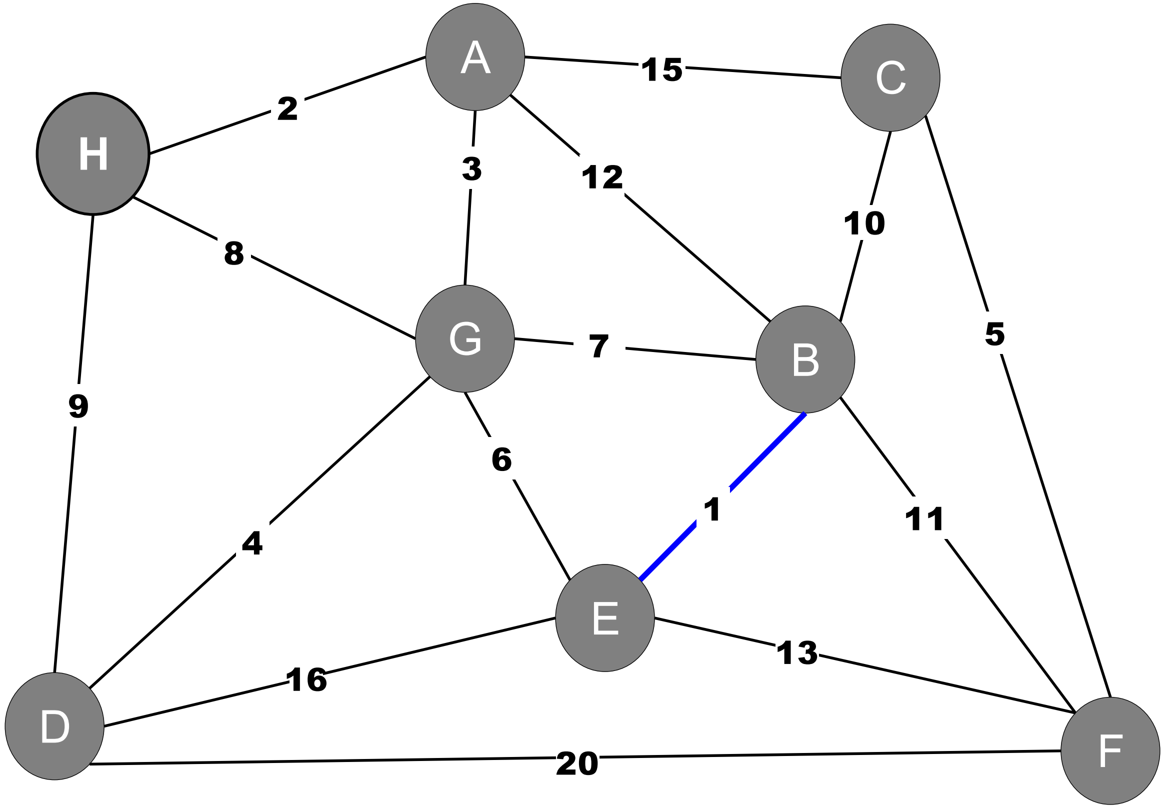
```
// Sort edges and initialize
Edge[] sortedEdges = sort(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());


// Iterate through all the edges, in order
for (int i=0; i<sortedEdges.length; i++) {
        Edge e = sortedEdges[i]; // get edge
        Node v = e.one(); // get node endpoints
        Node w = e.two();

        if (!uf.find(v,w)) { // in the same tree?
                mstEdges.add(e); // save edge
                uf.union(v,w); // combine trees
        }
}
```

Kruskal's Example



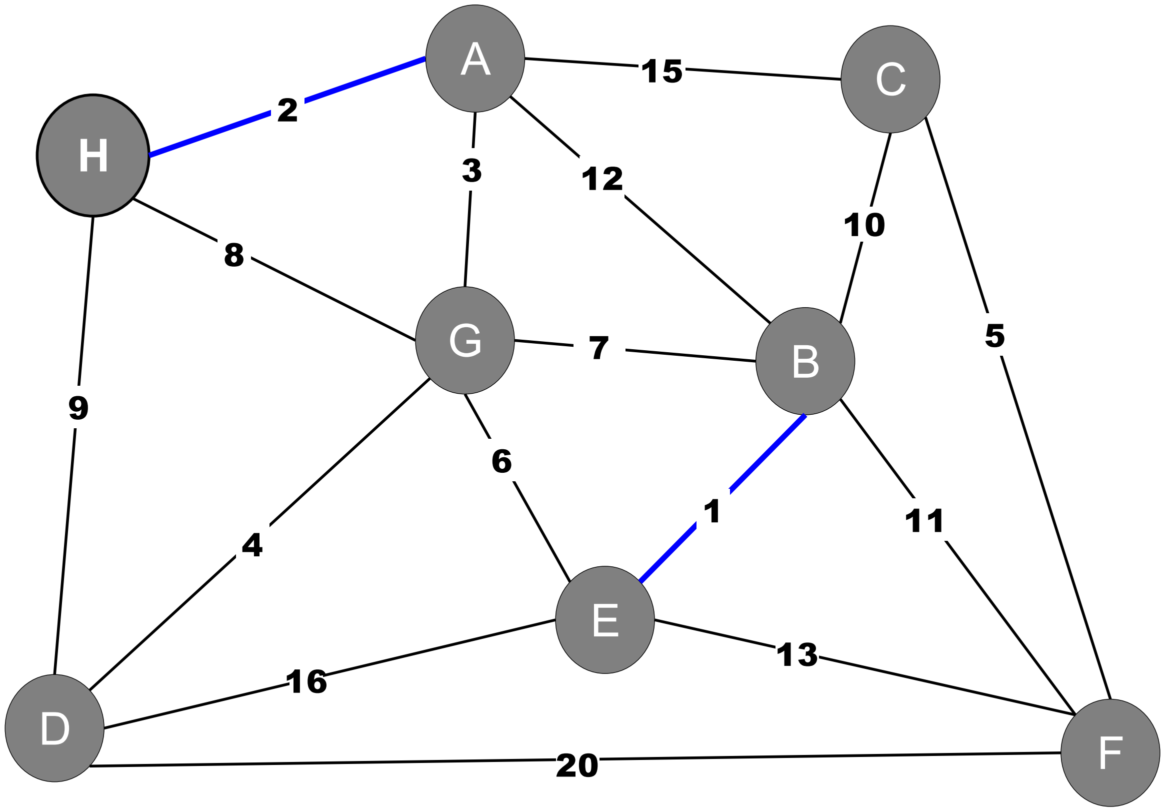
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



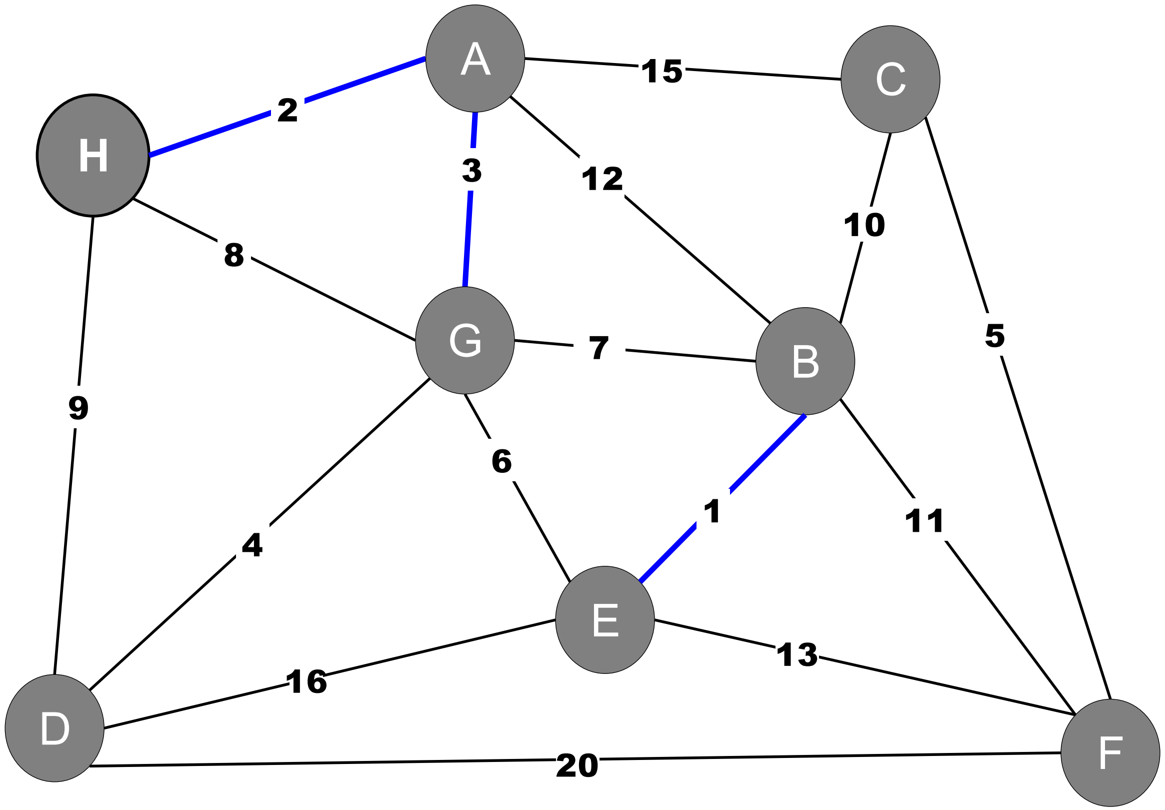
Weight	Edge
1	**(E,B)**
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



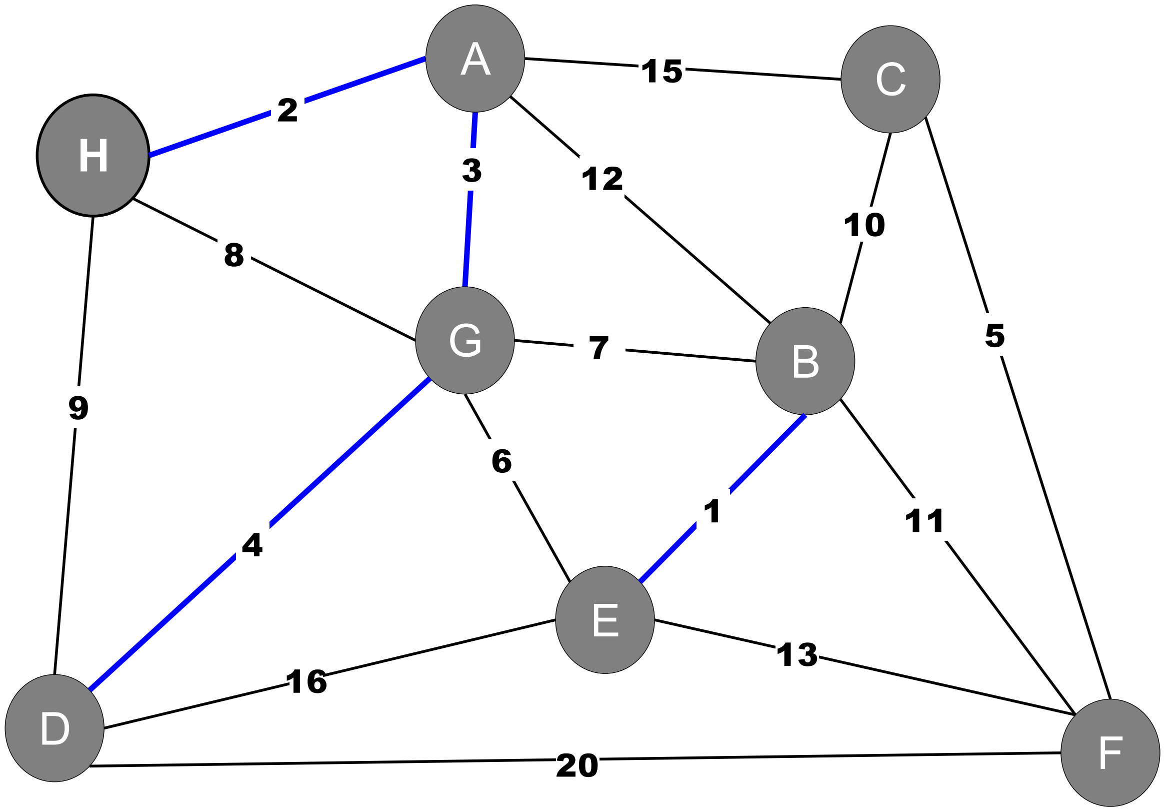
Weight	Edge
1	(E,B)
2	**(C,F)**
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal’s Example



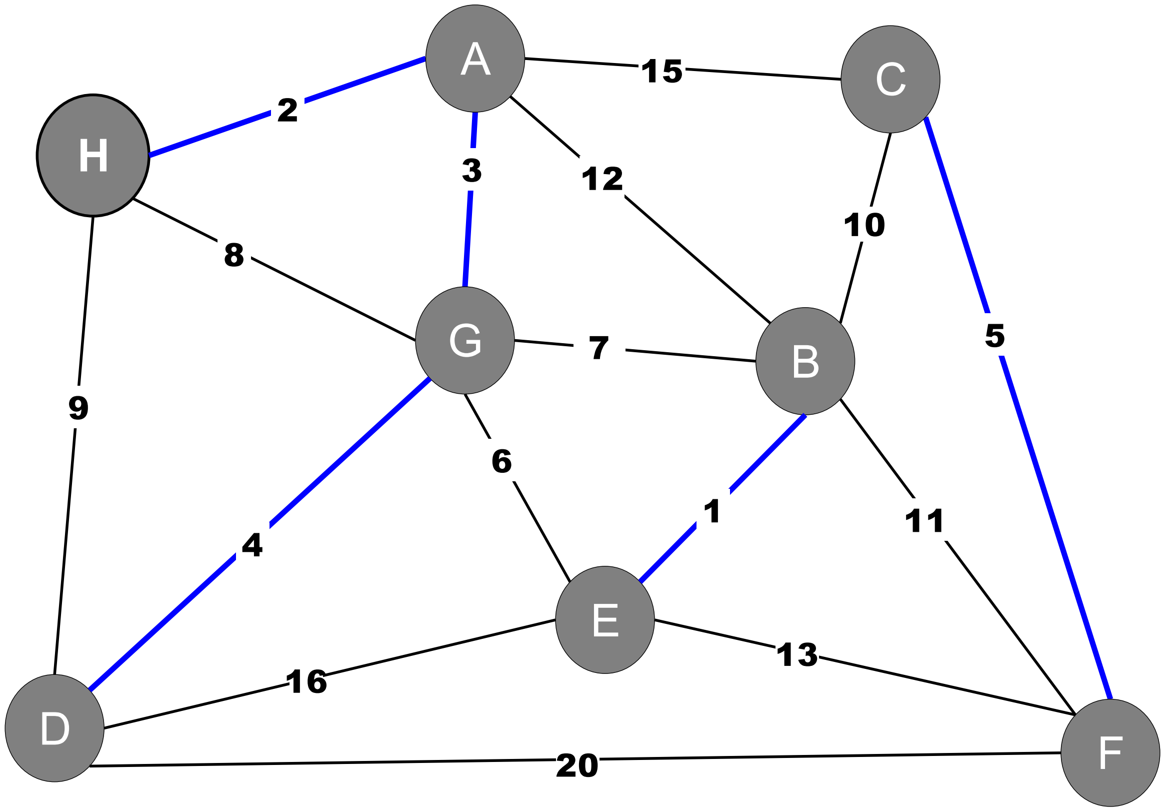
Weight	Edge
1	(E,B)
2	(C,F)
3	**(A,G)**
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



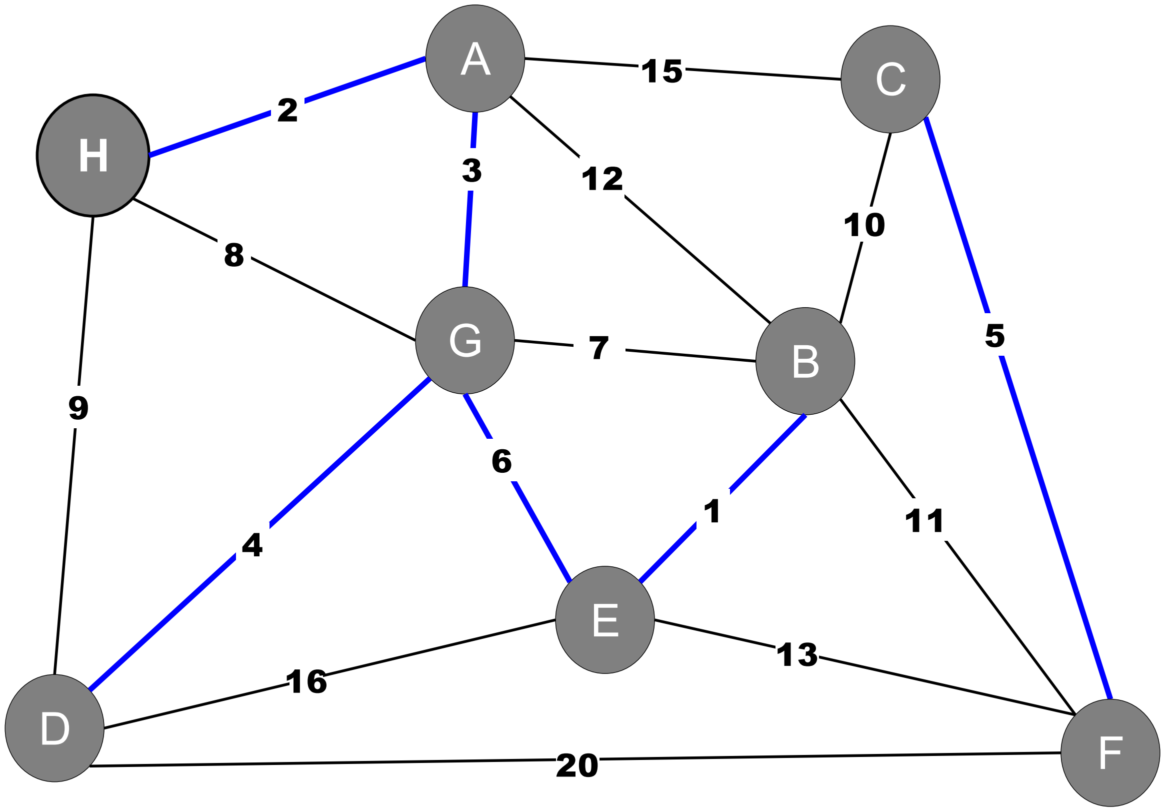
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	**(D,G)**
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



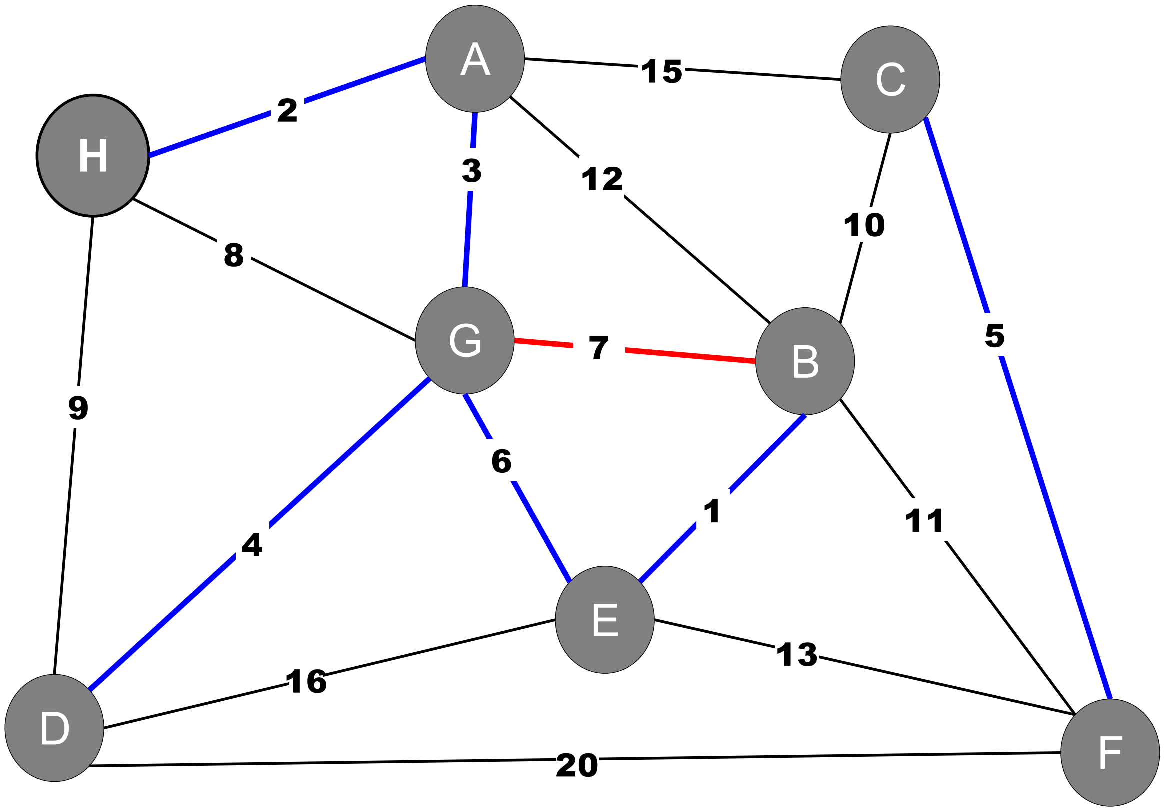
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	**(C,F)**
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal’s Example



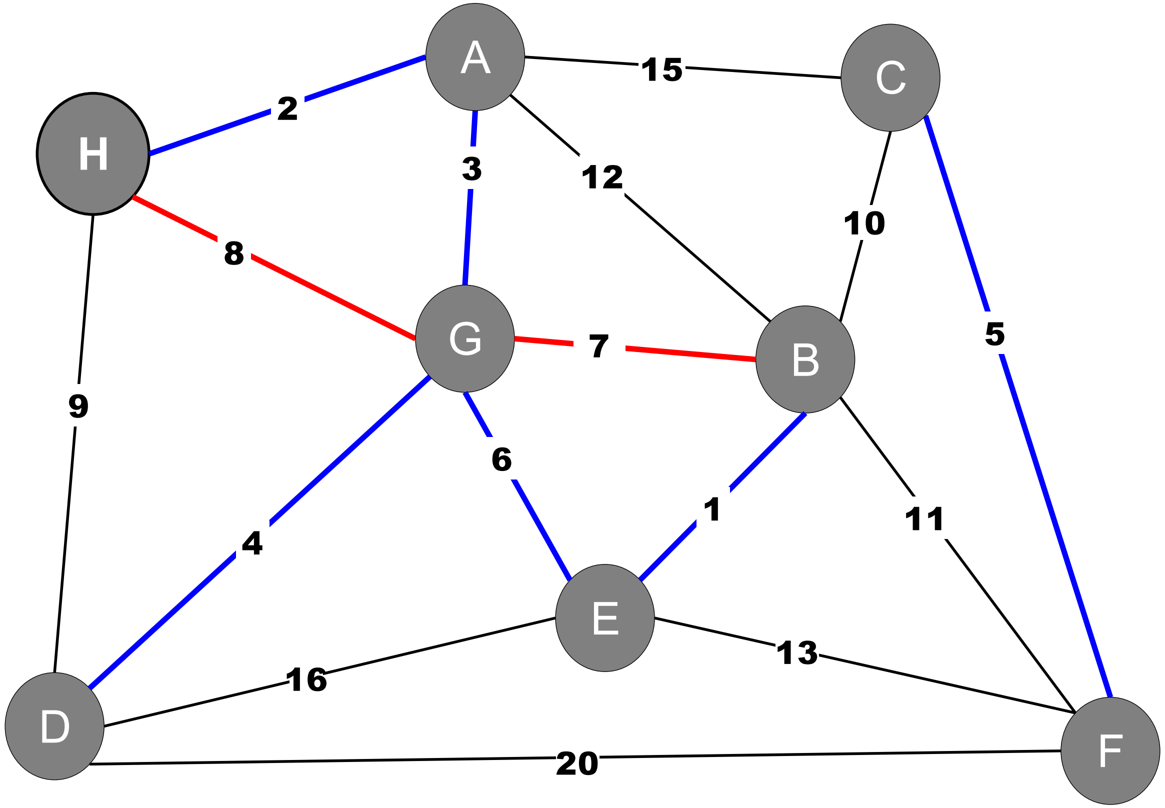
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	**(E,G)**
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



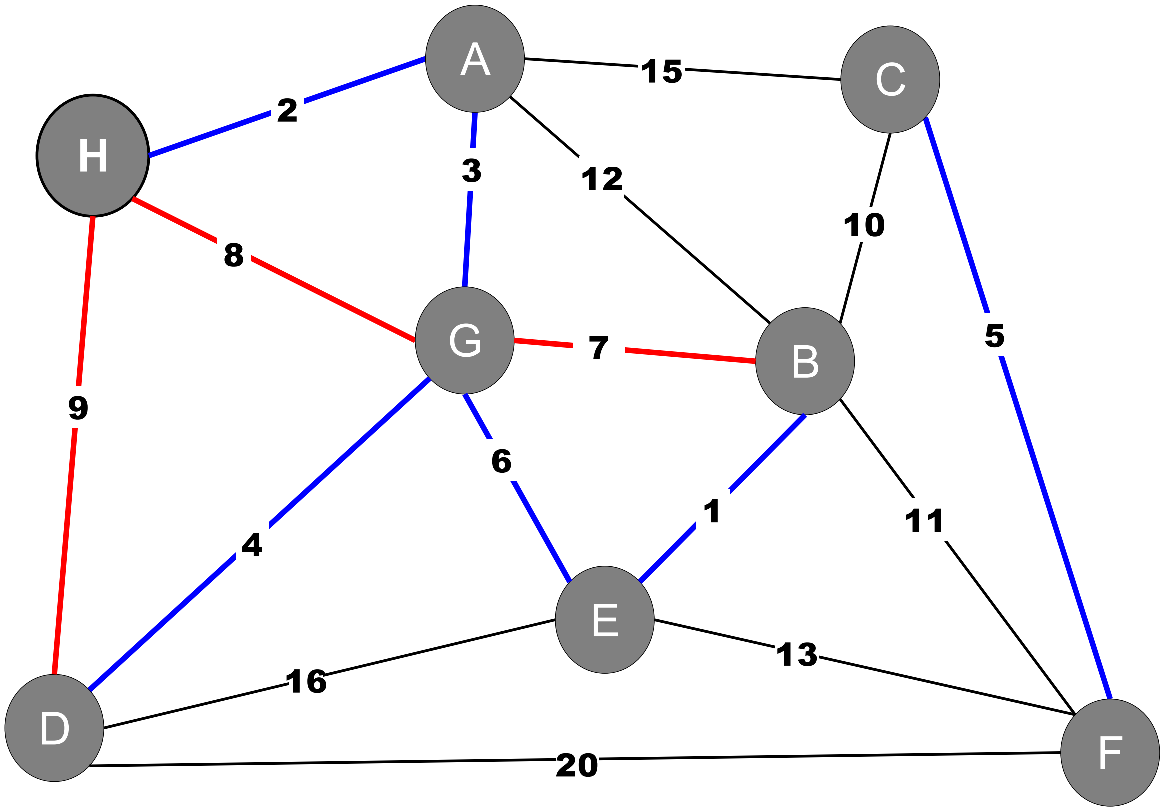
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	**(B,G)**
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



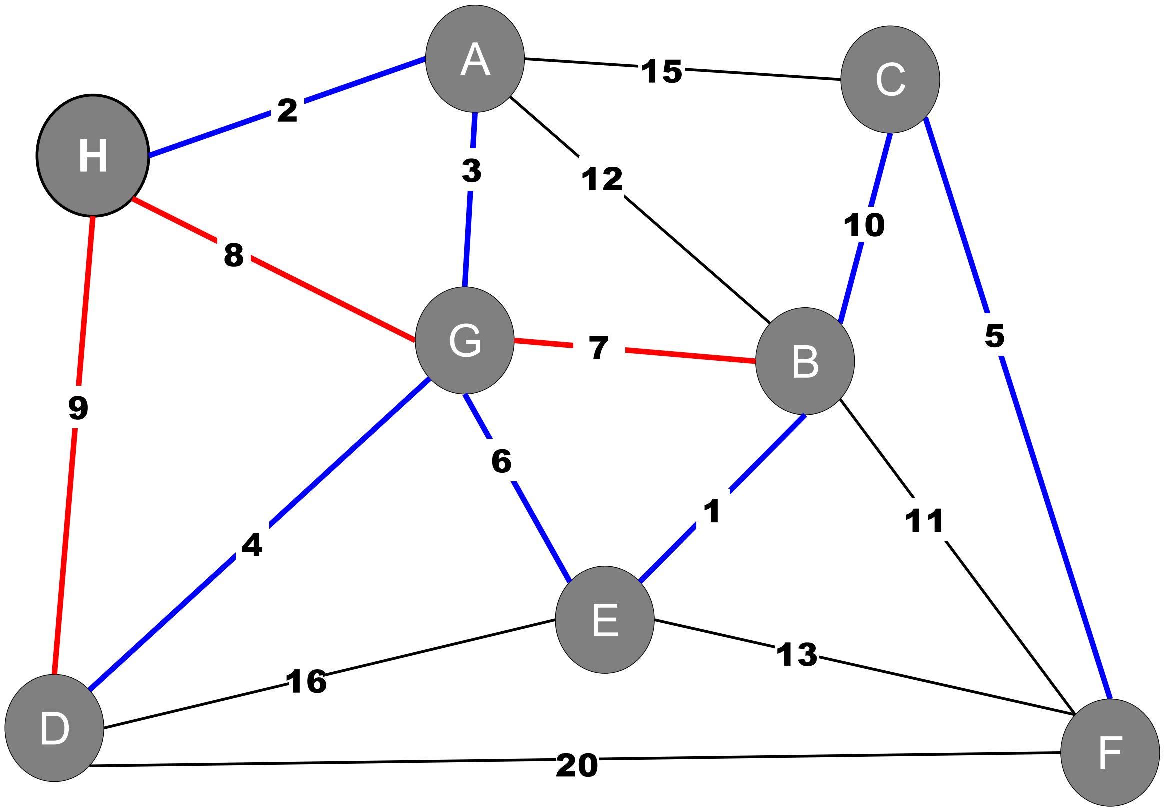
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	**(G,H)**
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



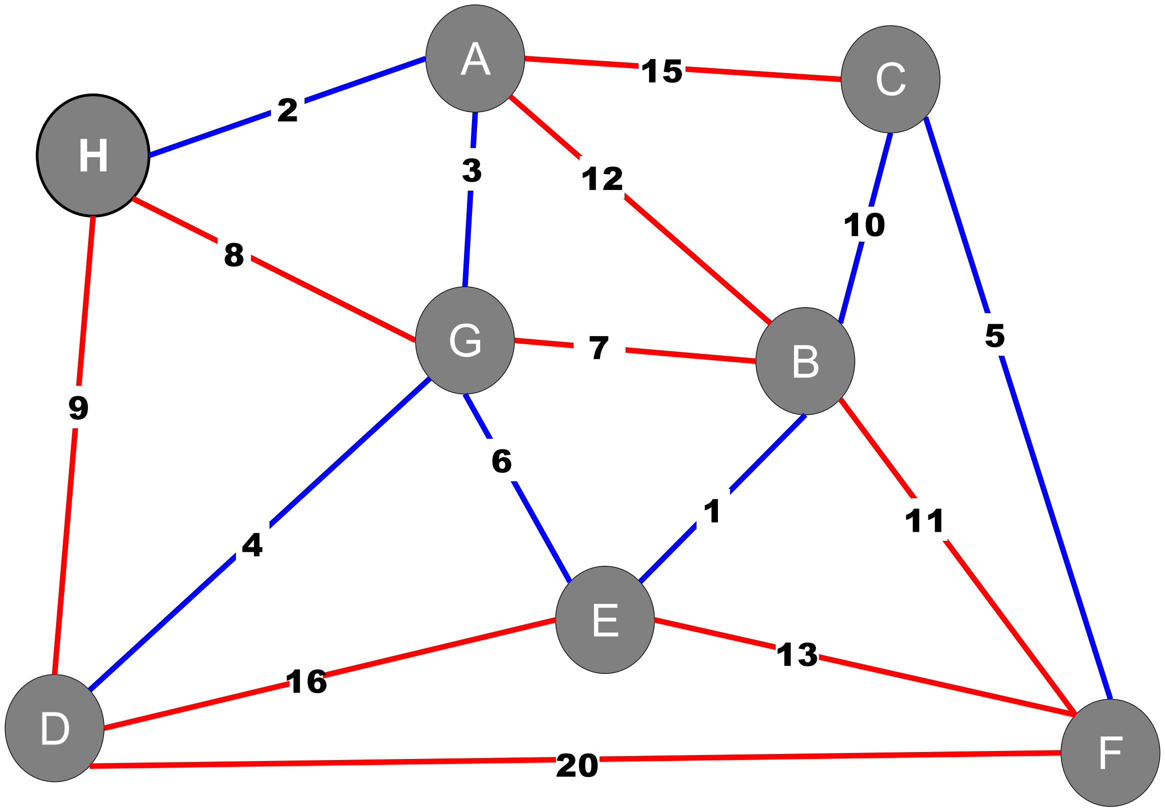
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	**(D,G)**
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Algorithm

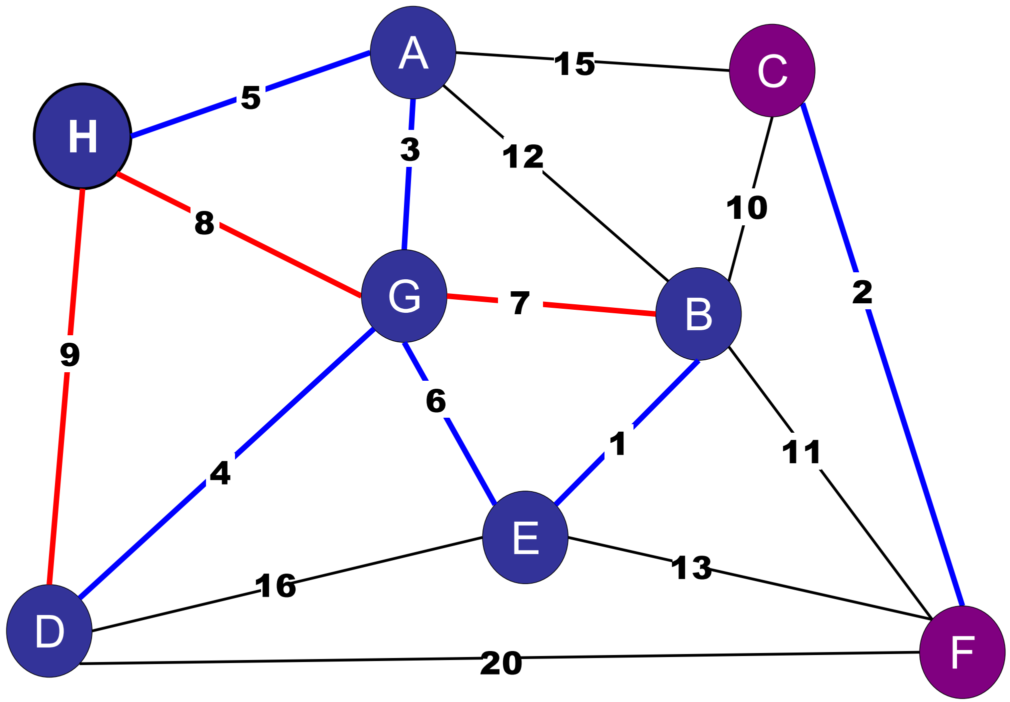
Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

Proof:

- Each added edge crosses a cut.
- Each edge is the lightest edge across the cut: all other lighter edges across the cut have already been considered.



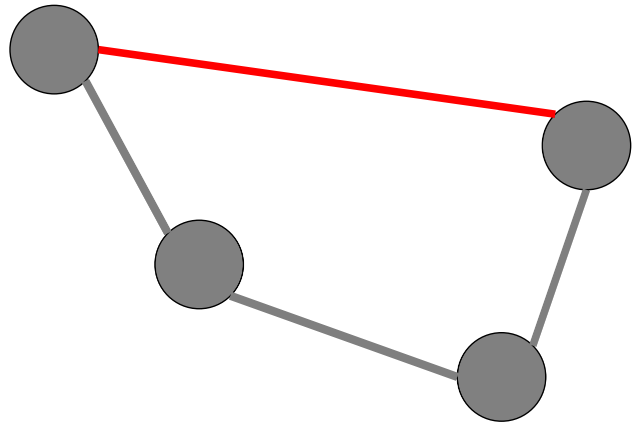
Generic MST Algorithm

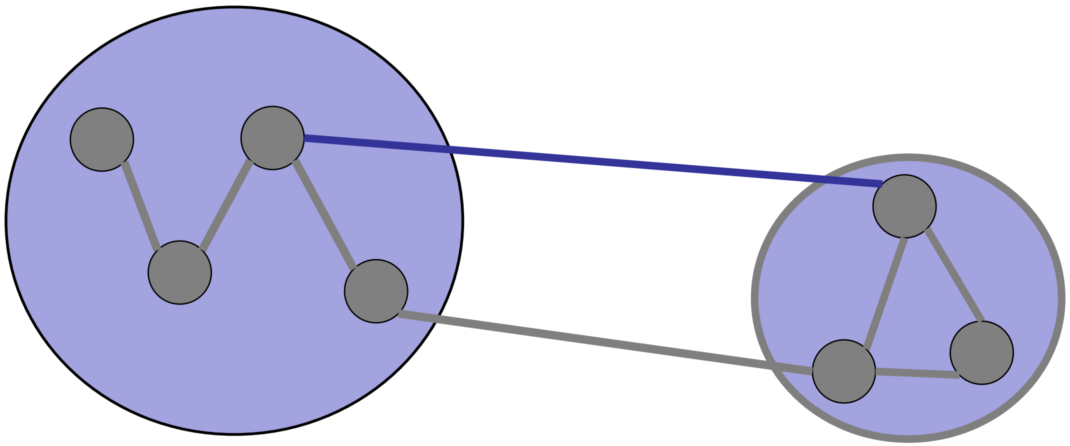
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

until no more edges can be colored.





What is the running time of Kruskal's Algorithm on a connected graph?

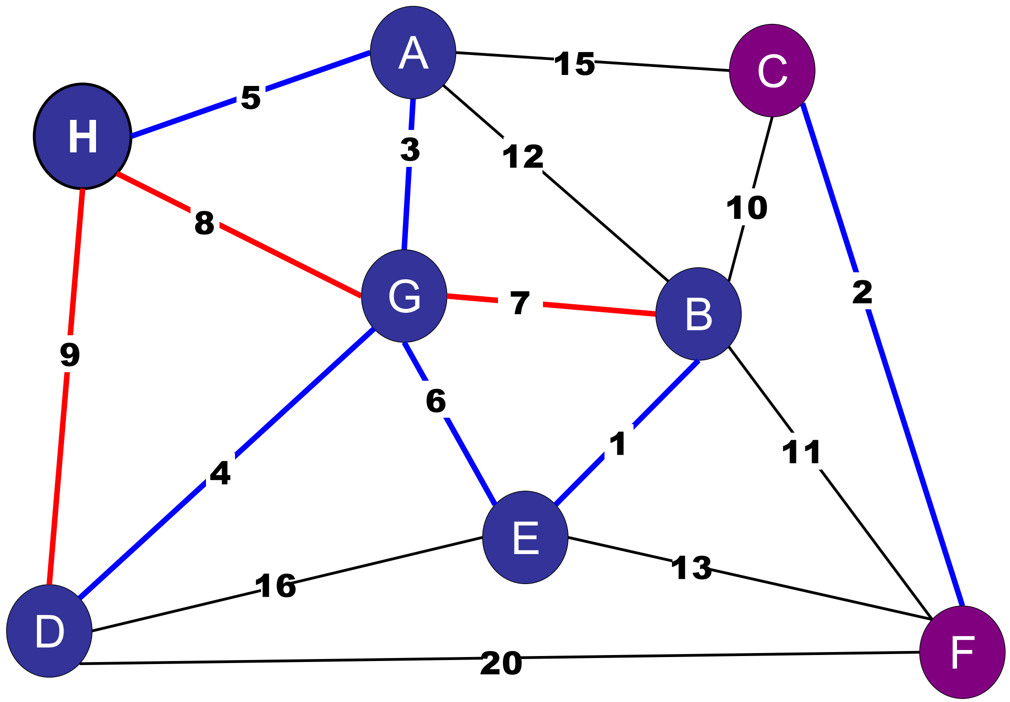
1. O(V)
2. O(E)
3. O(E α)
4. O(V α)
5. O(E log V)
6. O(V log E)

Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.



Performance:

- Sorting: O(E log E) = O(E log V)
- For E edges:
  - Find: O(α(n)) or O(log V)
  - Union: O(α(n)) or O(log V)

Roadmap

Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- Prim's Algorithm
- Kruskal's Algorithm
- **Boruvka's Algorithm**
- Variations

MST Algorithms

Classic:

– Prim’s Algorithm
– Kruskal’s Algorithm

Modern requirements:

– Parallelizable
– Faster in “good” graphs (e.g., planar graphs)
– Flexible

Boruvka’s Algorithm

Origin: 1926

- Otakar Boruvka
- Improve the electrical network of Moravia

Based on generic algorithm:

- Repeat: add all “obvious” blue edges.
- Very simple, very flexible.

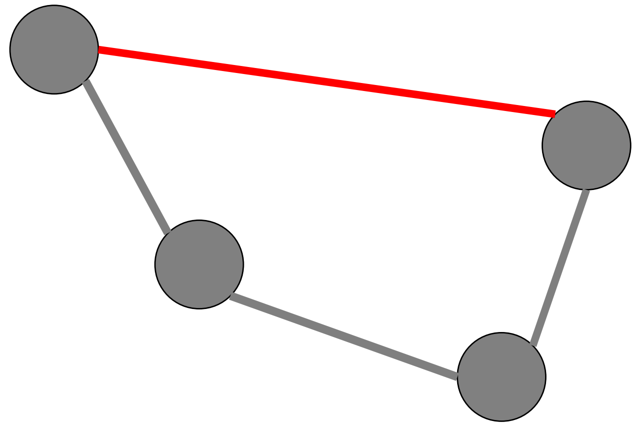
Generic MST Algorithm

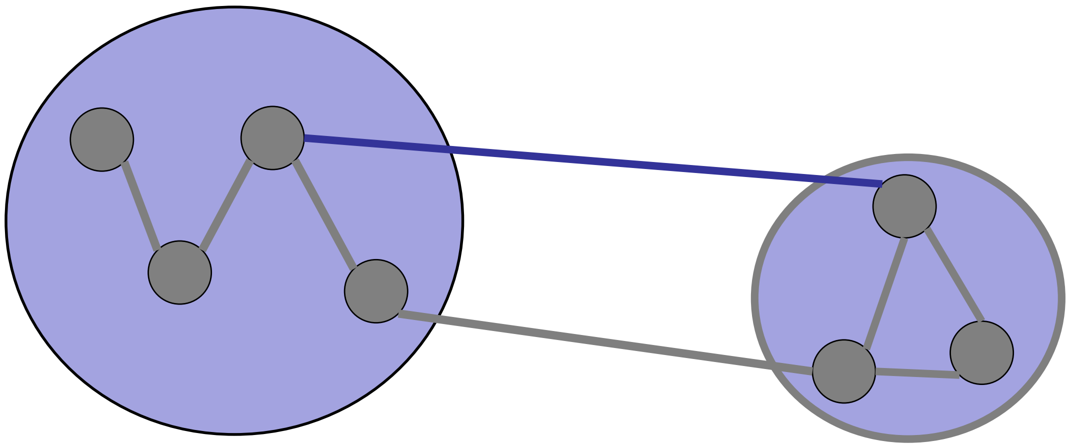
Greedy Algorithm:

Repeat:

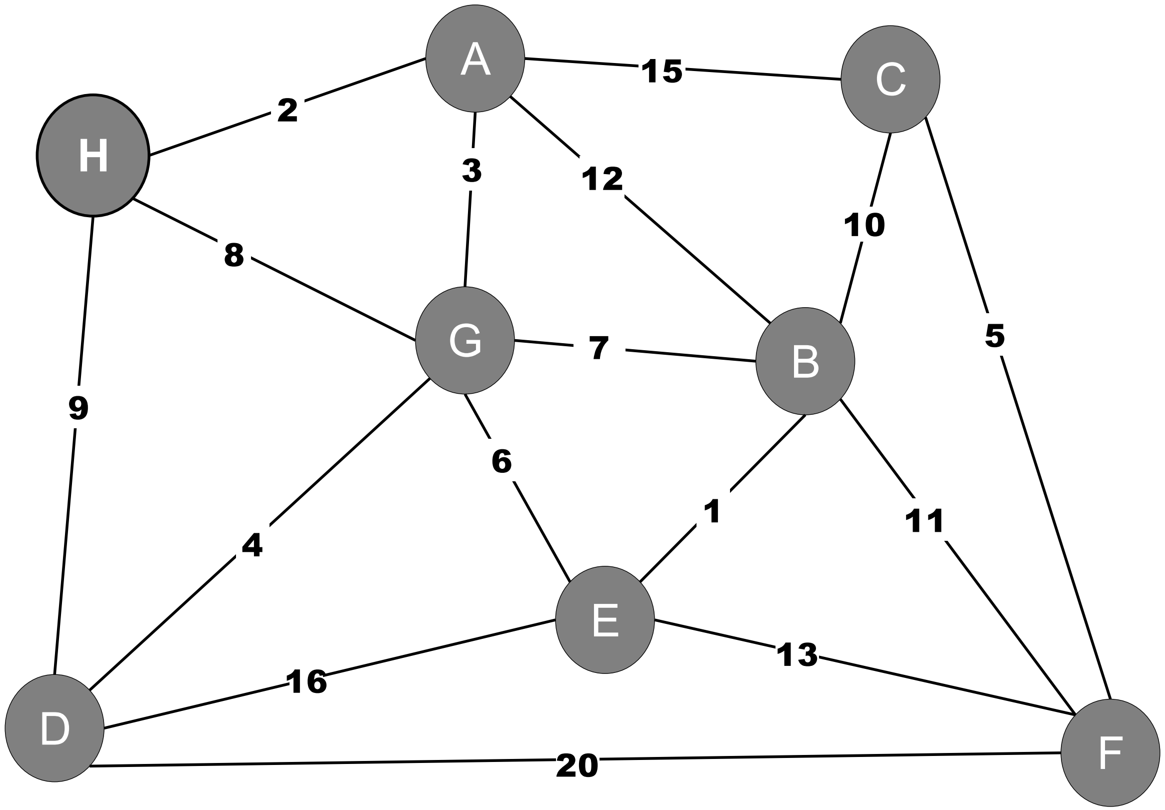
Apply red rule or blue rule to an arbitrary edge.

until no more edges can be colored.





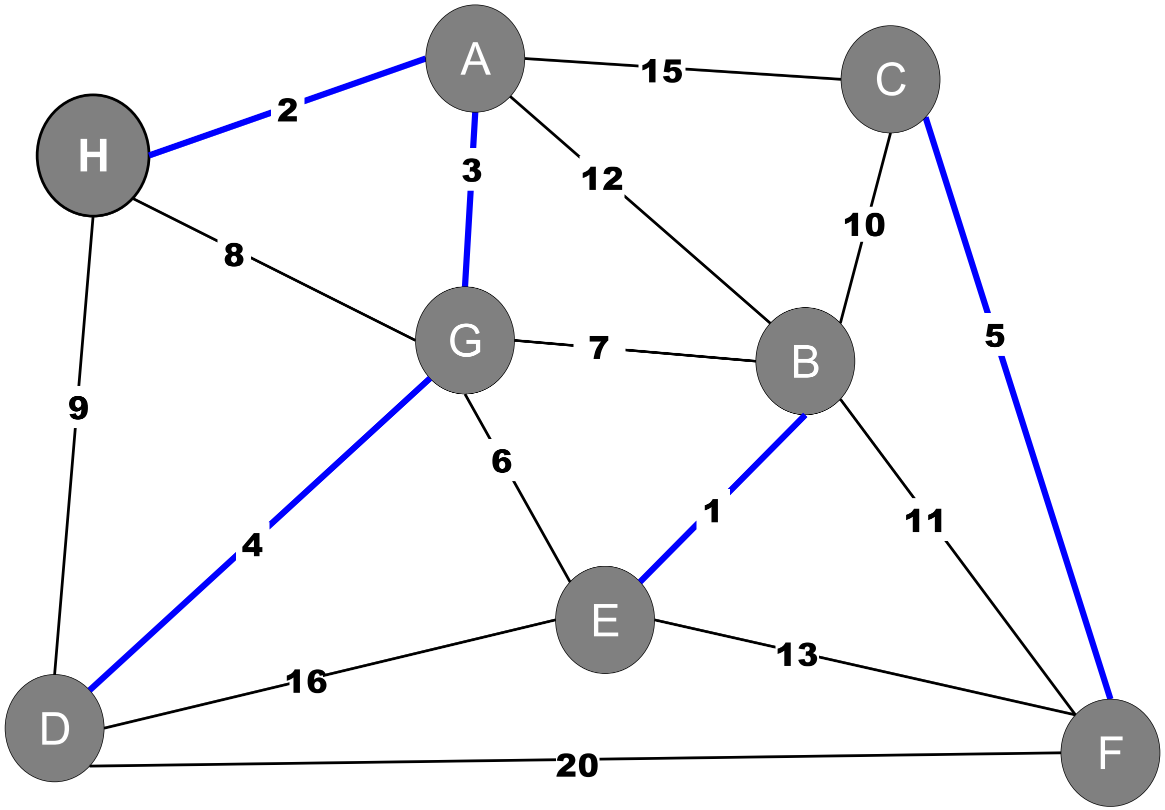
Boruvka's Example



Which edges are "obviously" in the MST?

Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Boruvka's Example

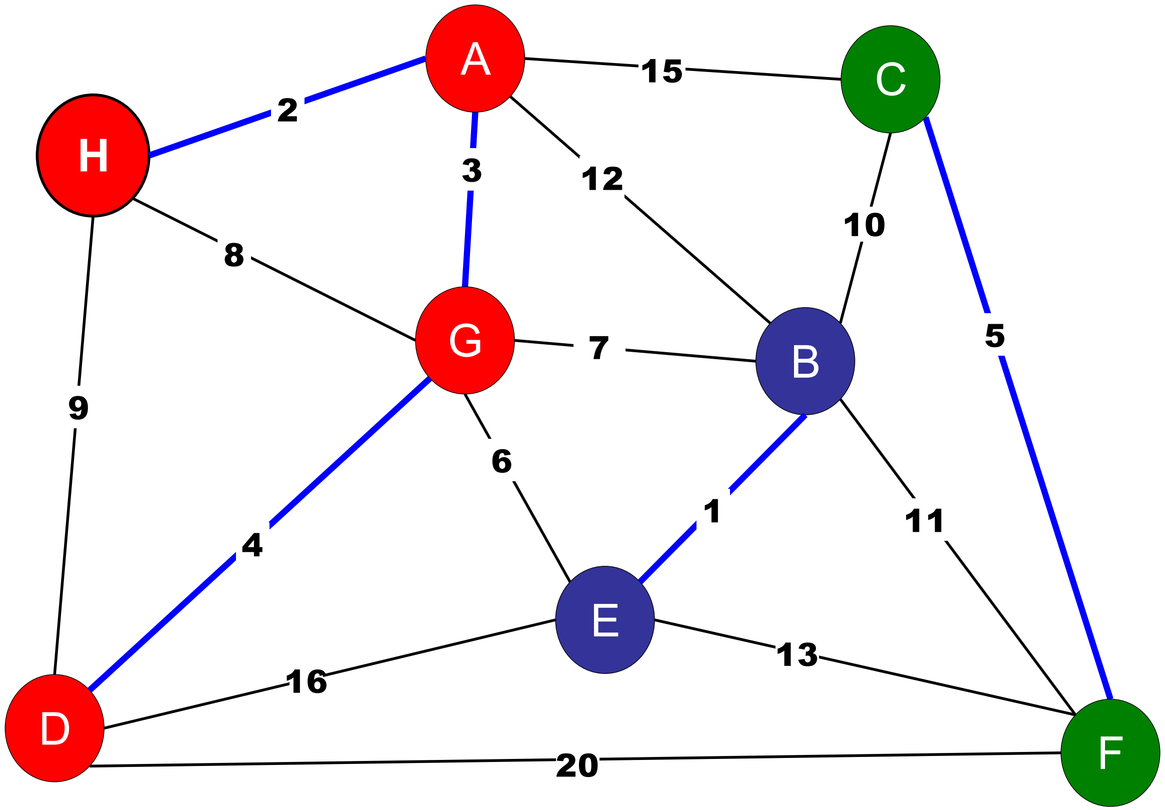


For every node: add minimum adjacent edge.

Add at least n/2 edges.

Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Boruvka's Example

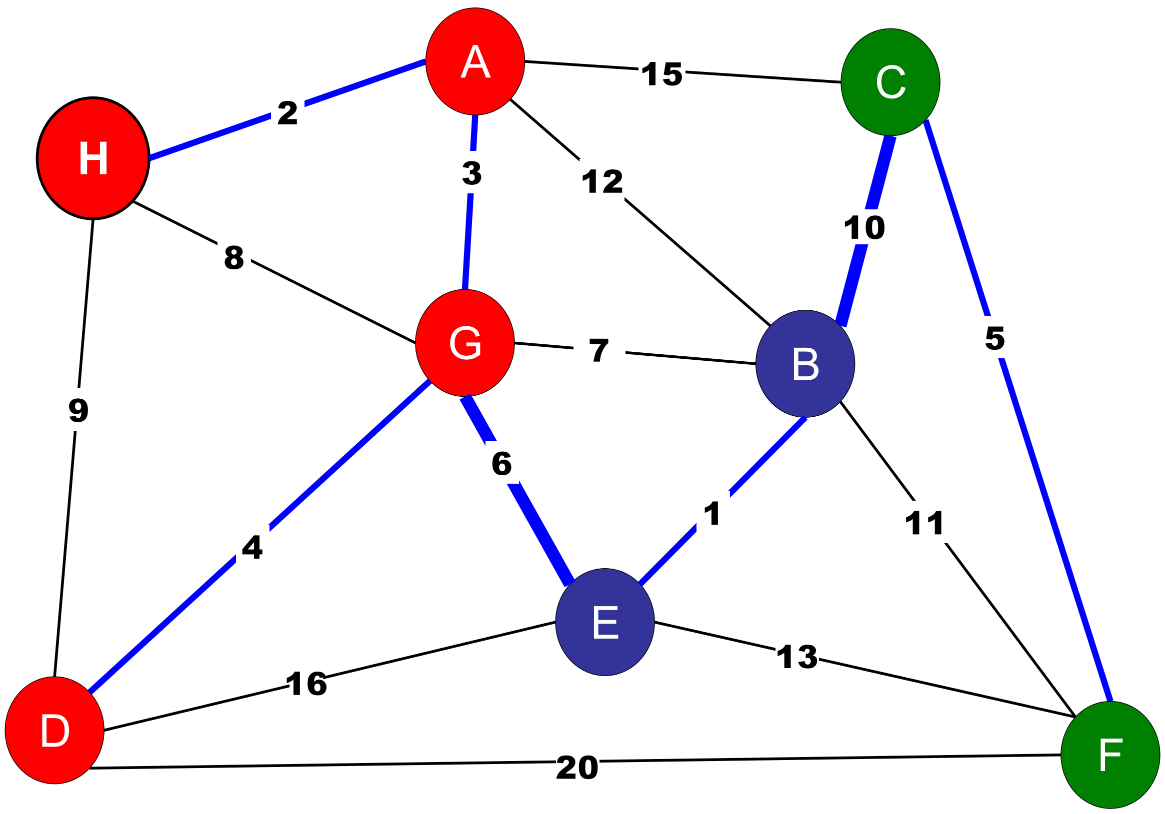


Look at connected components…

At most n/2 connected components.

Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Boruvka's Example



Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Repeat: for every connected components, add minimum outgoing edge.

Boruvka's Algorithm

Boruvka's Algorithm

Initially:

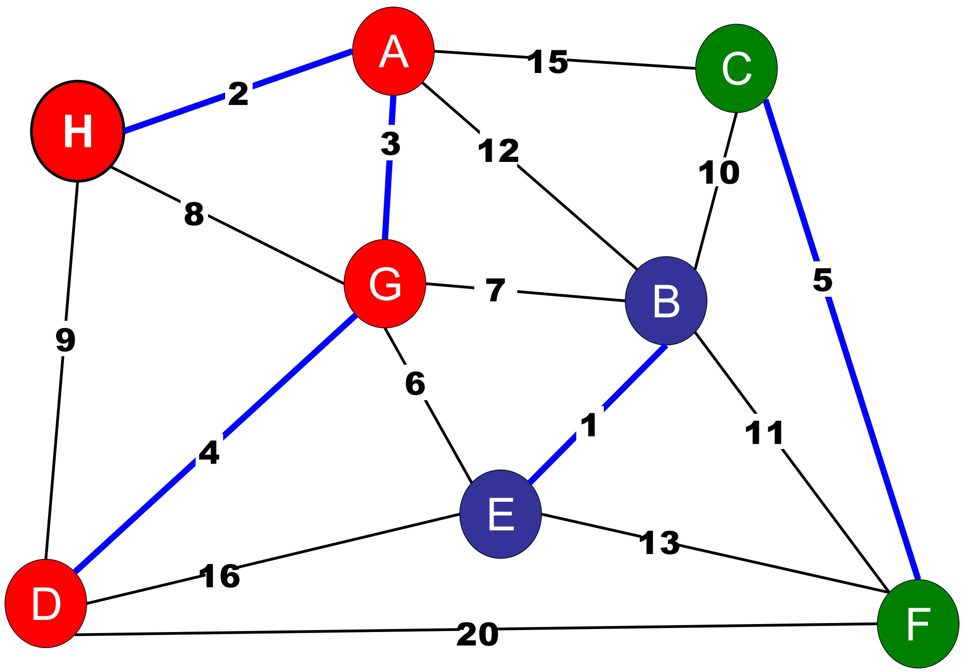
– Create n connected components, one for each node in the graph.

One "Boruvka" Step:

– For each connected component, search for the minimum weight outgoing edge.

– Add selected edges.

– Merge connected components.



Boruvka's Algorithm

Boruvka's Algorithm

Initially:

- Create n connected components, one for each node in the graph.

For each node: store a component identifier.

One "Boruvka" Step:

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.

Boruvka's Algorithm

For each node: store a component identifier.

Boruvka's Algorithm

Initially:

- Create n connected components, one for each node in the graph.

DFS or BFS:

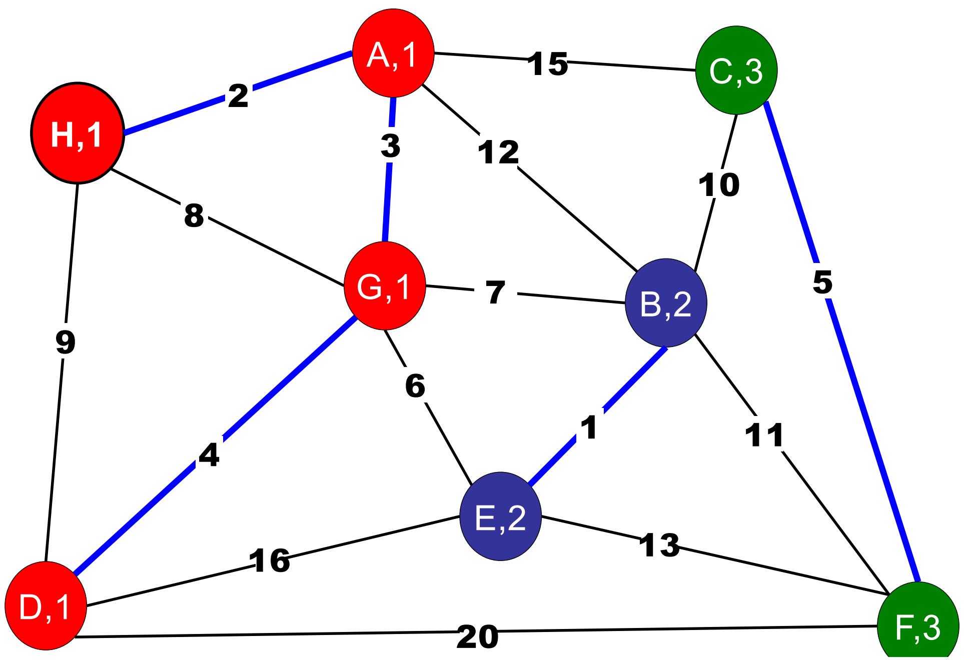
Check if edge connects two components.

Remember minimum cost edge connected to each component.

One "Boruvka" Step:

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.

Component	1	2	3
Min cost edge	(G,E), 6	(G,E), 6	(B,C), 10
To be merged	1 and 2	1 and 2	2 and 3



Boruvka's Algorithm

Boruvka's Algorithm

Initially:

- Create n connected components, one for each node in the graph.

One "Boruvka" Step:

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.

Component	1	2	3
Min cost edge	(G,E), 6	(G,E), 6	(B,C), 10
To be merged	1 and 2	1 and 2	2 and 3
New ID:	1	1	1

For each node: store a component identifier.

DFS or BFS:

Check if edge connects two components.

Remember minimum cost edge connected to each component.

Scan every node:

Compute new component ids.

Update component ids.

Mark added edges.

Boruvka's Algorithm

Boruvka's Algorithm

Initially:

- Create n connected components, one for each node in the graph.

One "Boruvka" Step: O(V+E)

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.

For each node: O(V)
store a component identifier.

DFS or BFS: O(V + E)
Check if edge connects two components.

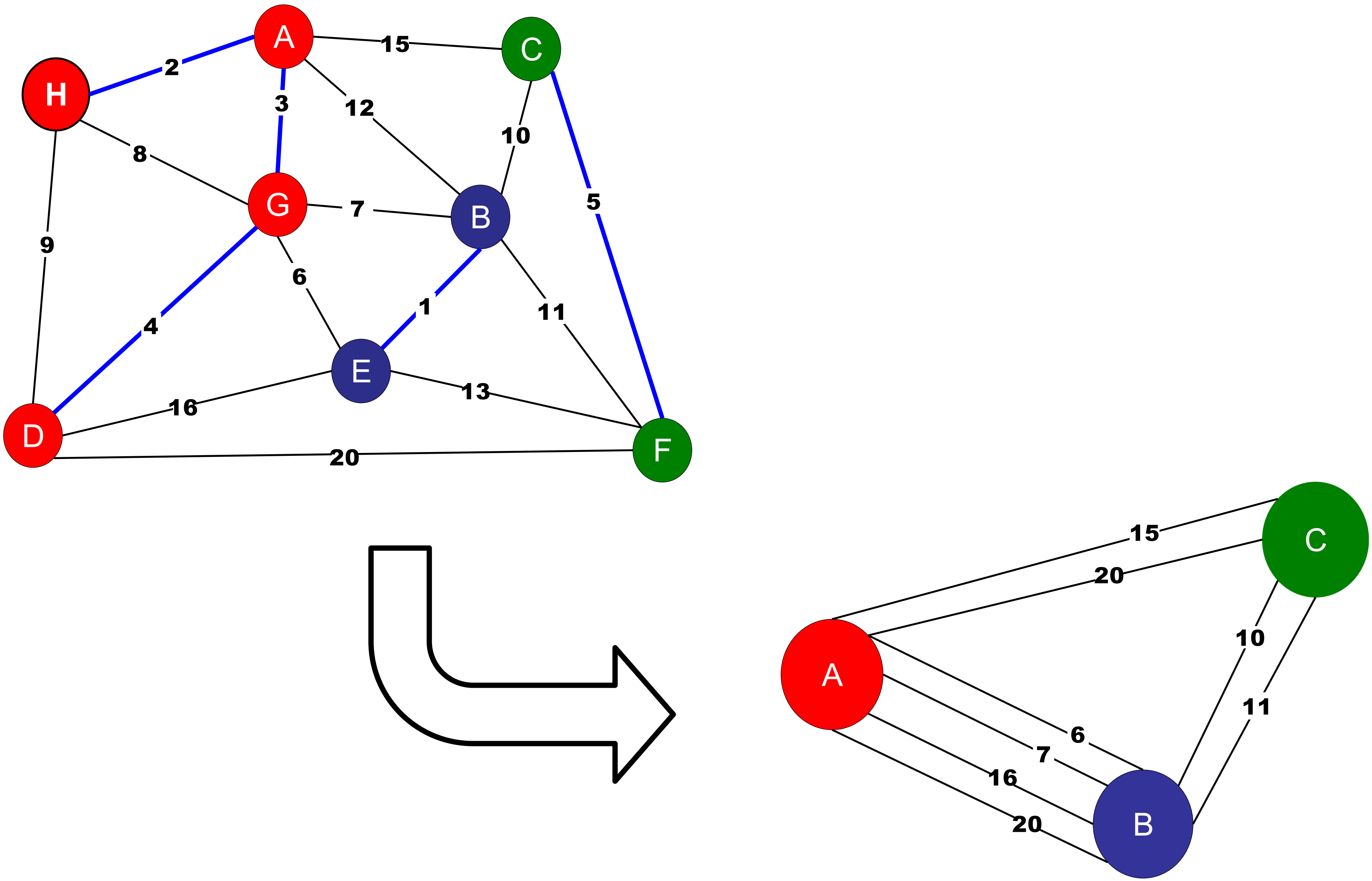
Remember minimum cost edge connected to each component.

Scan every node: O(V)
Compute new component ids.

Update component ids.

Mark added edges.

Boruvka's Example: Contraction



Boruvka's Algorithm

Boruvka's Algorithm

Initially:

- Create n connected components, one for each node in the graph.

In each "Boruvka" Step: O(V+E)

- Assume k components, initially.
- At least k/2 edges added.

Count edges:

Each component adds one edge.

Some choose same edge.

Each edge is chosen by at most two different components.

Boruvka's Algorithm

Boruvka's Algorithm

Initially:

- Create n connected components, one for each node in the graph.

In each "Boruvka" Step: O(V+E)

- Assume k components, initially.
- At least k/2 edges added.
- At least k/2 components merge.

Merging:
Each edge merges two components

Boruvka's Algorithm

Boruvka's Algorithm

Initially:

- Create n connected components, one for each node in the graph.

In each "Boruvka" Step: O(V+E)

- Assume k components, initially.
- At least k/2 edges added.
- At least k/2 components merge.
- At end, at most k/2 components remain.

Boruvka's Algorithm

Boruvka's Algorithm

Initially:

n components

At each step:

k components ➔ k/2 components.

Termination:

1 component

Conclusion:

At most O(log V) Boruvka steps.

Boruvka's Algorithm

Boruvka's Algorithm

Initially:

n components

At each step:

k components ➔ k/2 components.

Termination:

1 component

Conclusion:

At most O(log V) Boruvka steps.

Total time:

O((E+V)log V) = O(E log V)

Boruvka's Algorithm

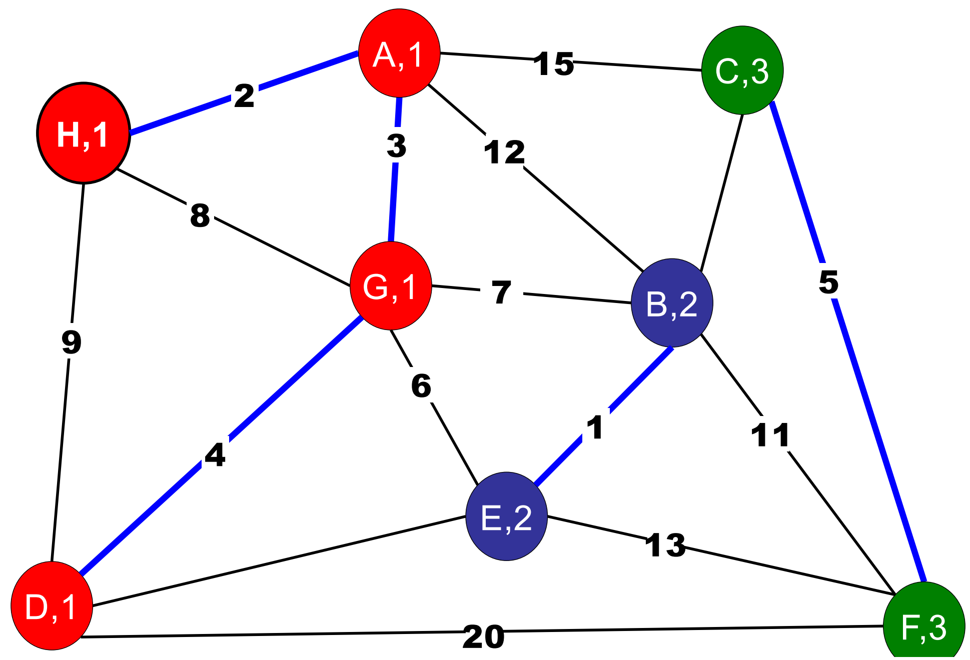
Boruvka's Algorithm

Initially:

- Create n connected components, one for each node in the graph.

One "Boruvka" Step: O(V+E)

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.



Roadmap

So far:

Minimum Spanning Trees

- Prim's Algorith
- Kruskal's Algorithm
- Boruvka's Algorithm

Minimum Spanning Tree Summary

Classic greedy algorithms: O(E log V)

- Prim's (Priority Queue)
- Kruskal's (Union-Find)
- Boruvka's

Best known: $O(m\ \alpha(m, n))$

- Chazelle (2000)

Holy grail and major open problem: O(m)

Minimum Spanning Tree Summary

Classic greedy algorithms: O(E log V)

- Prim's (Priority Queue)
- Kruskal's (Union-Find)
- Boruvka's

Best known: O(m α(m, n))

- Chazelle (2000)

Holy grail and major open problem: O(m)

- Randomized: Karger-Klein-Tarjan (1995)
- Verification: Dixon-Rauch-Tarjan (1992)

Roadmap

Today: Minimum Spanning Trees

- Prim’s Algorithm
- Kruskal’s Algorithm
- Boruvka’s Algorithm

Variations:

- Constant weight edges
- Bounded integer edge weights
- Directed graphs
- Maximum Spanning Tree
- Steiner Tree