CS2040S
Data Structures and Algorithms

All about minimum spanning trees…

(Part 2)

Roadmap

Minimum Spanning Trees

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

Variations:

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

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

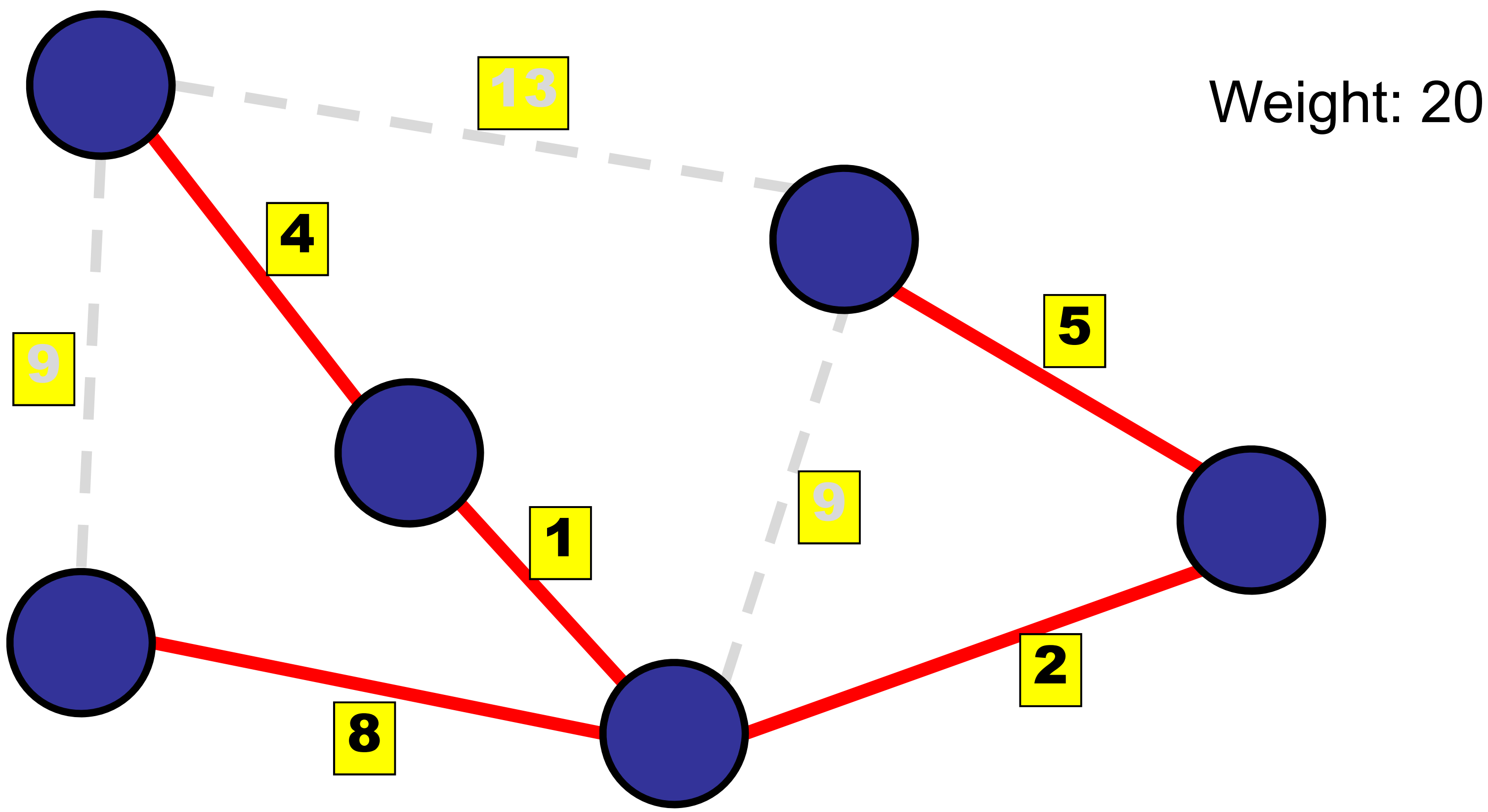
Roadmap

Minimum Spanning Trees

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

Minimum Spanning Tree

Definition: a spanning tree with minimum weight



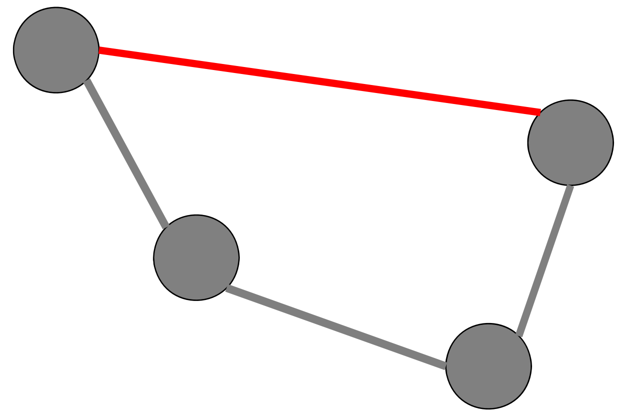
Property of MST

- No cycles in a MST
- If you cut an MST, the two pieces are both MSTs.
- Cycle property
  - For every cycle, the maximum weight edge is not in the MST.
- Cut property
  - For every cut D, the minimum weight edge that crosses the cut is in the MST.

Generic MST Algorithm

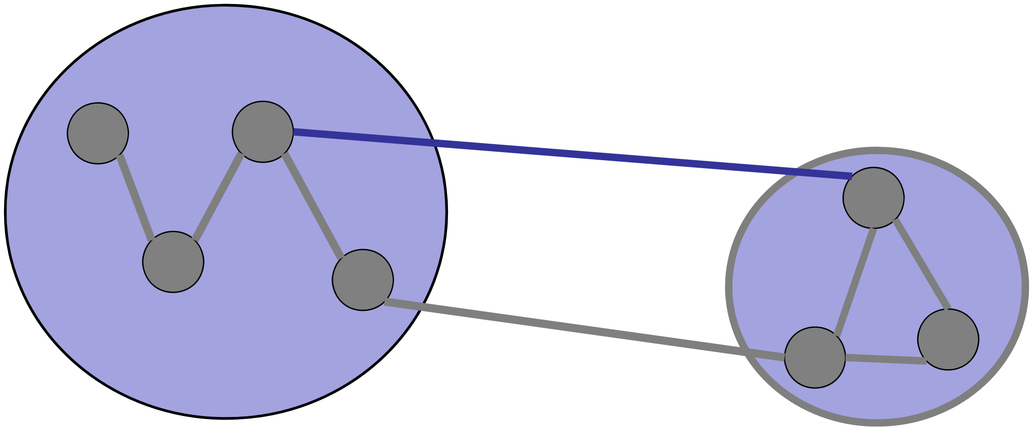
Red rule:

If C is a cycle with no red arcs, then color the max-weight edge in C red.



Blue rule:

If D is a cut with no blue arcs, then color the min-weight edge in D blue.



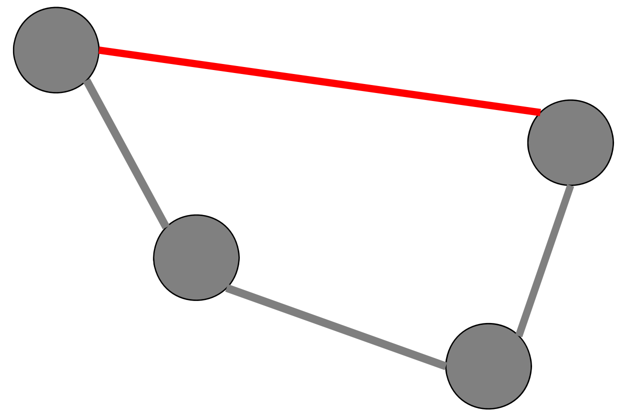
Generic MST Algorithm

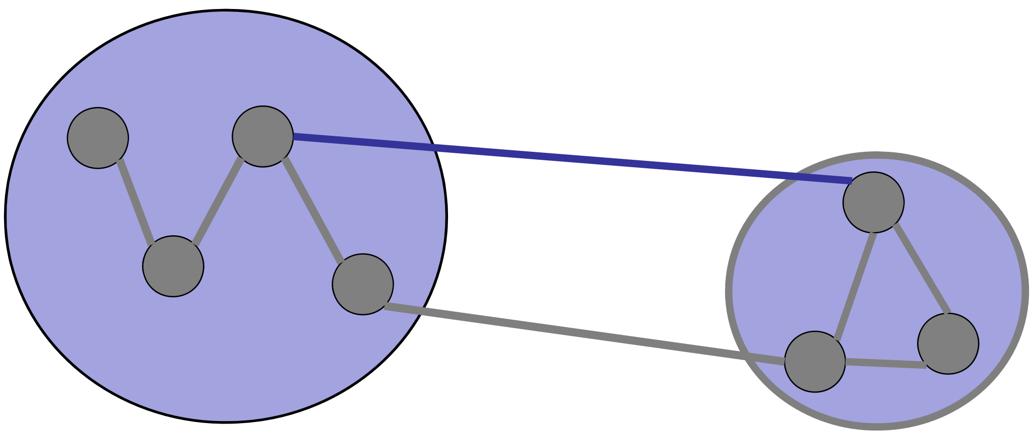
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

until no more edges can be colored.





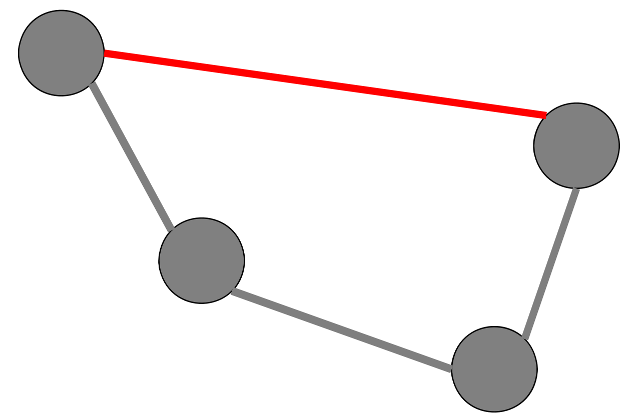
Generic MST Algorithm

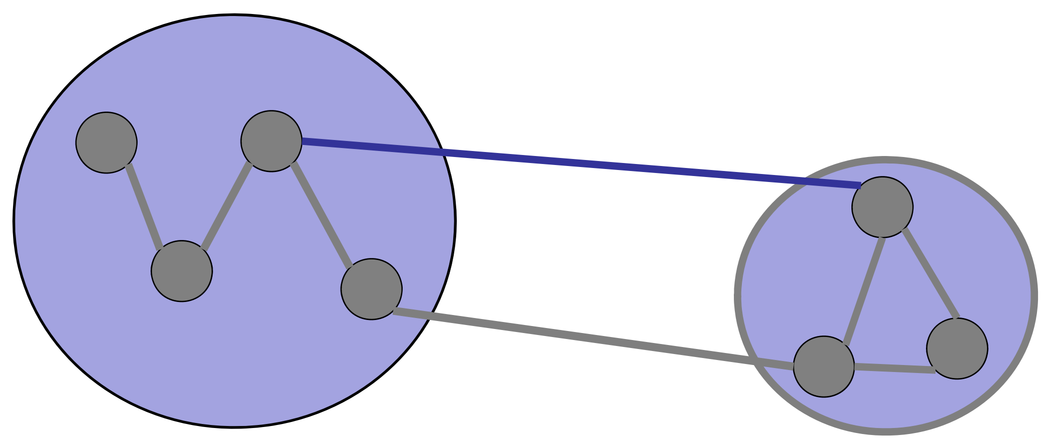
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

until no more edges can be colored.





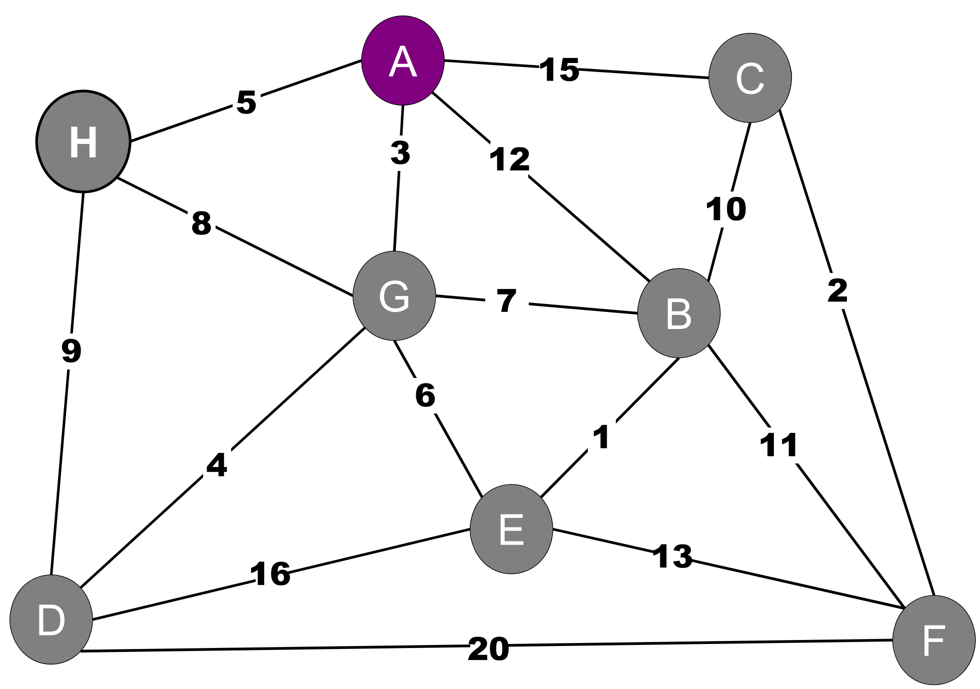
Roadmap

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)

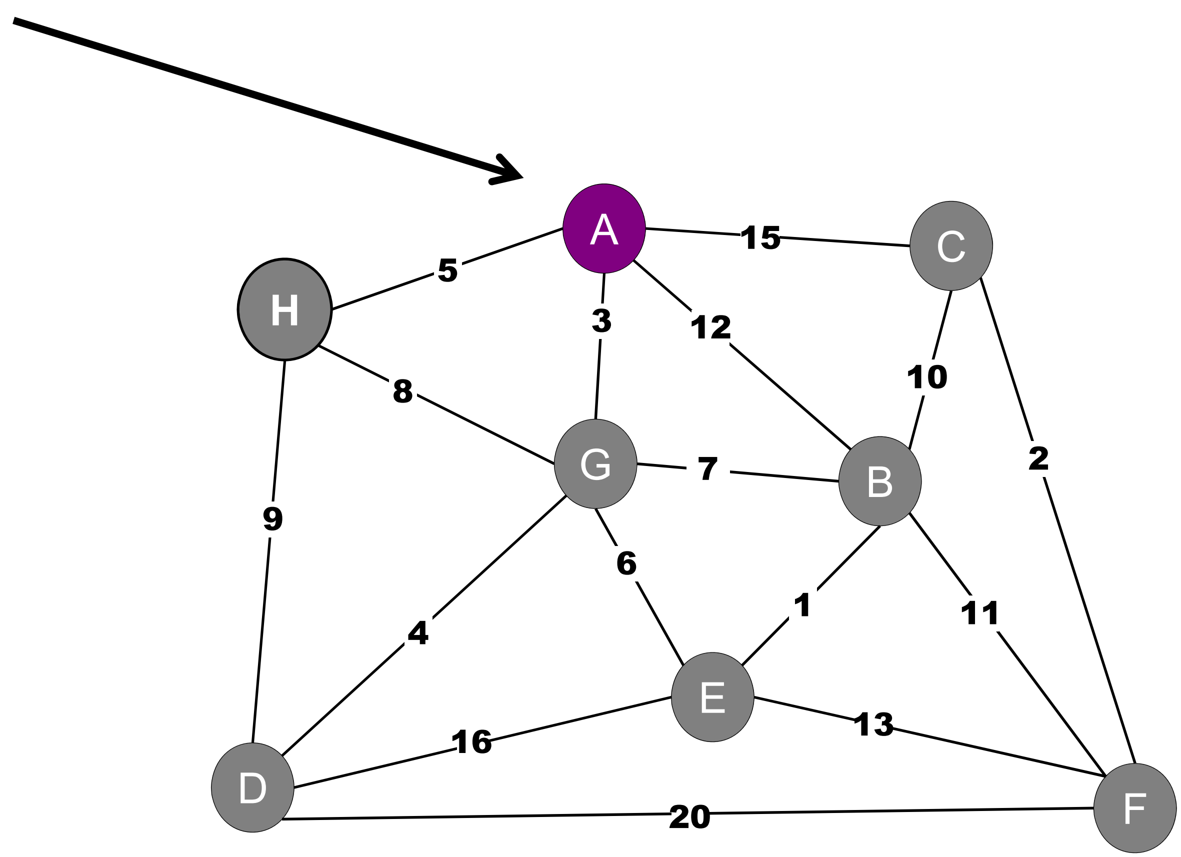


Prim's Algorithm

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

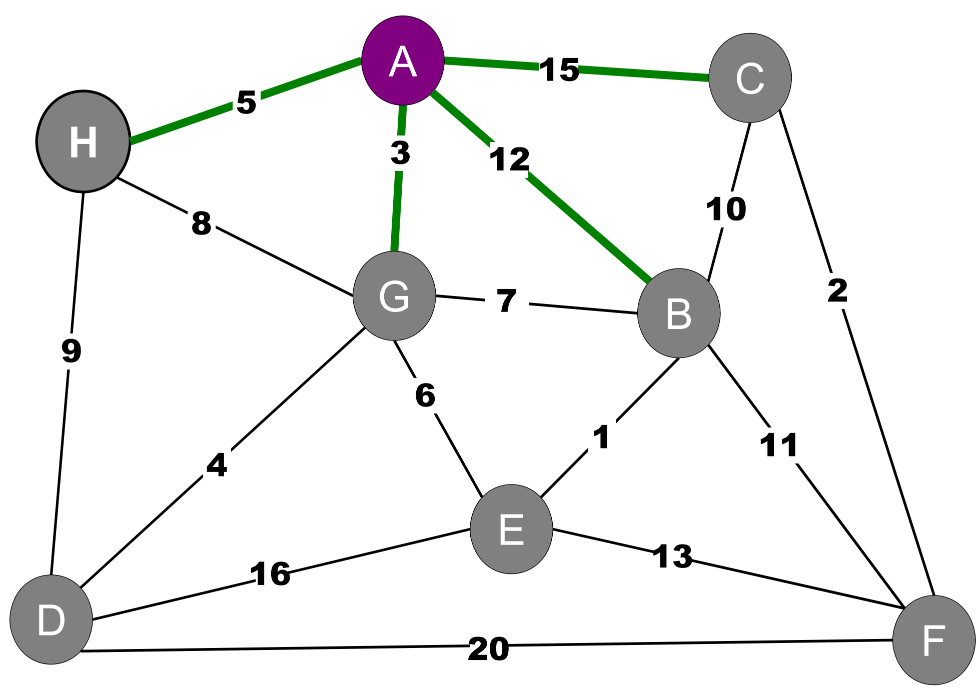
Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

– S : set of nodes connected by blue edges.

– Initially: S = {A}

– Identify cut: {S, V–S}

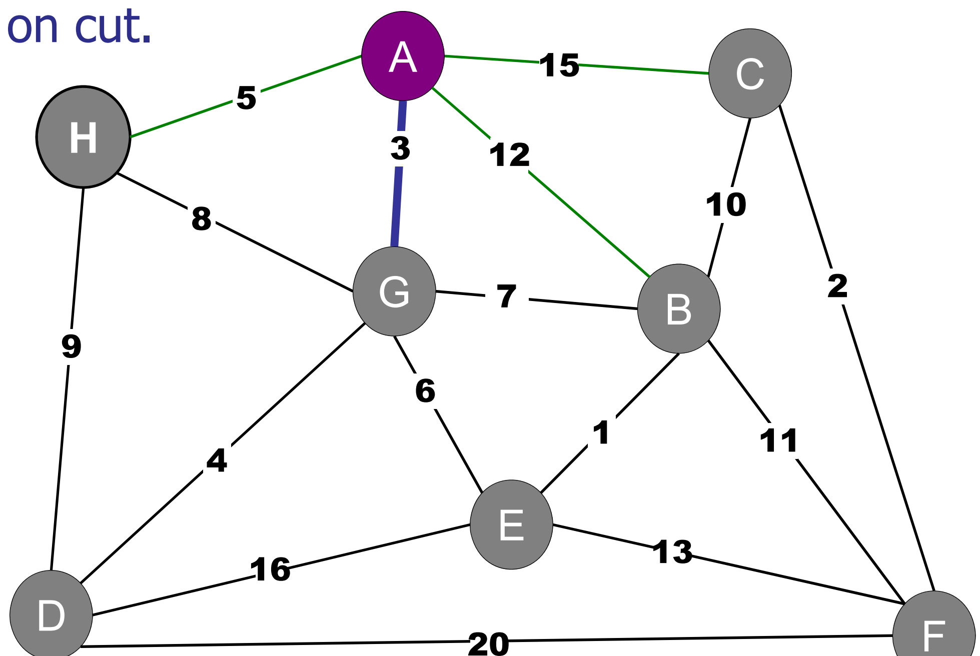


Prim’s Algorithm

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.

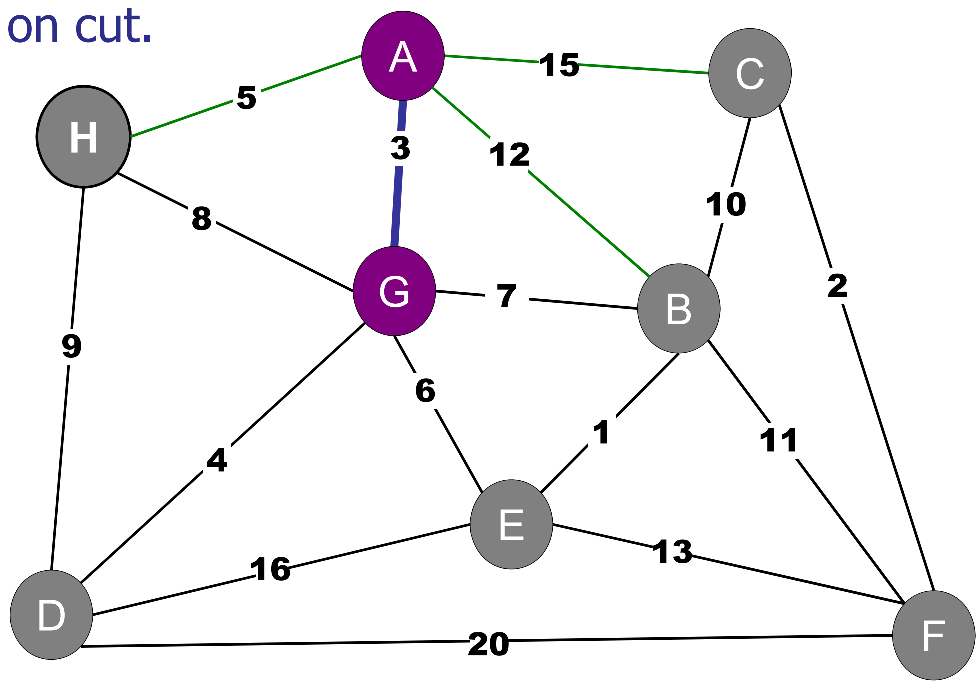


Prim's Algorithm

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.

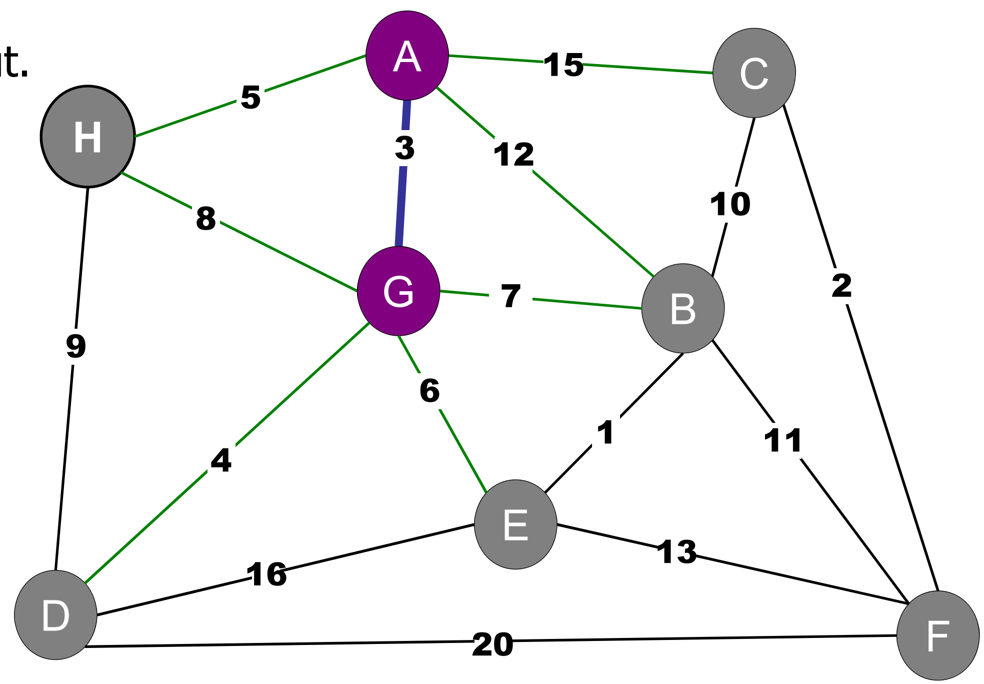


Prim's Algorithm

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.

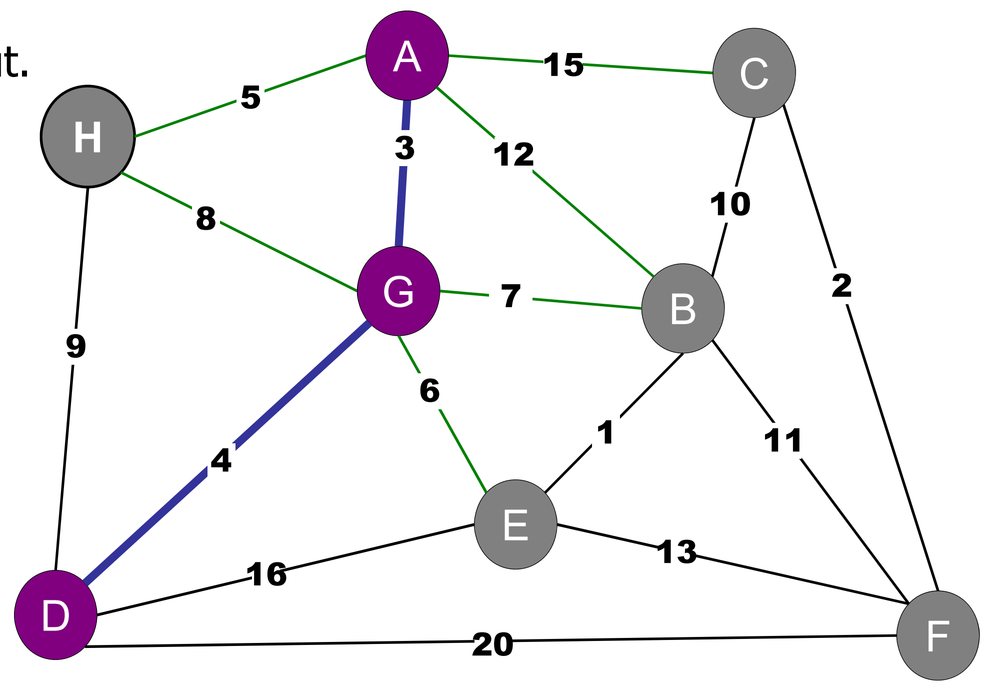


Prim’s Algorithm

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.

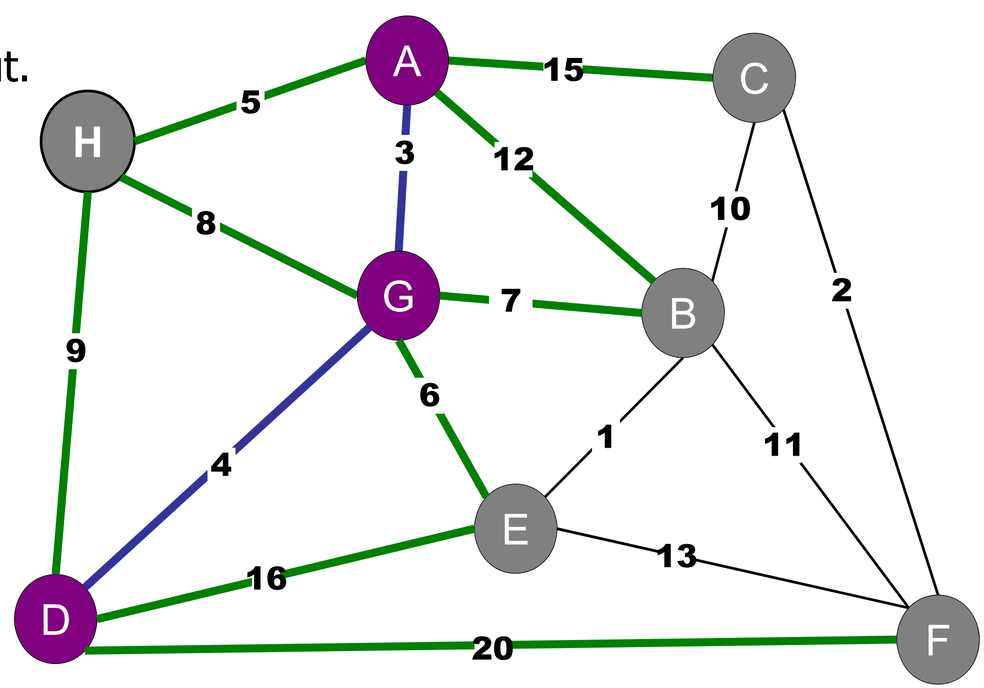


Prim's Algorithm

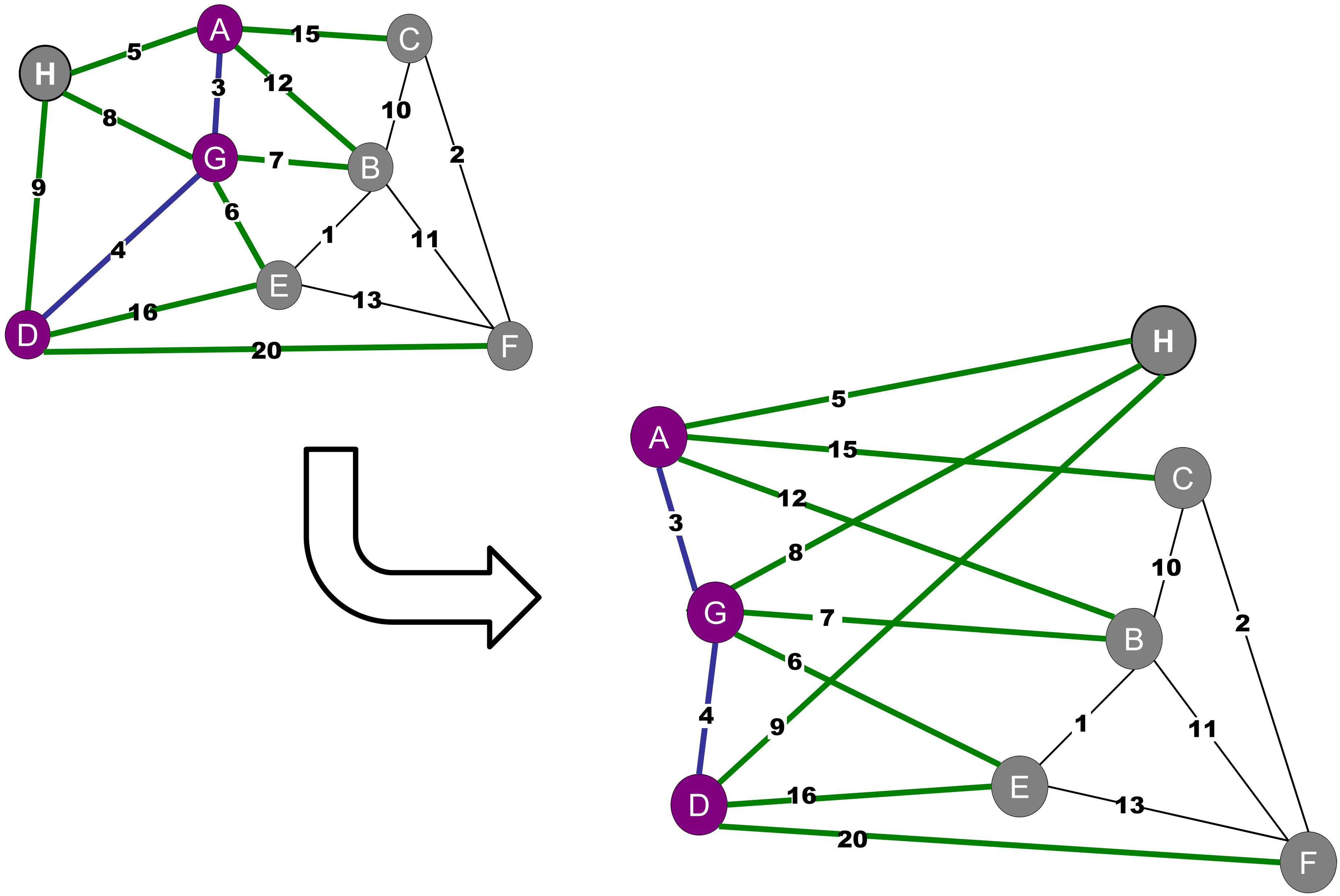
Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

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

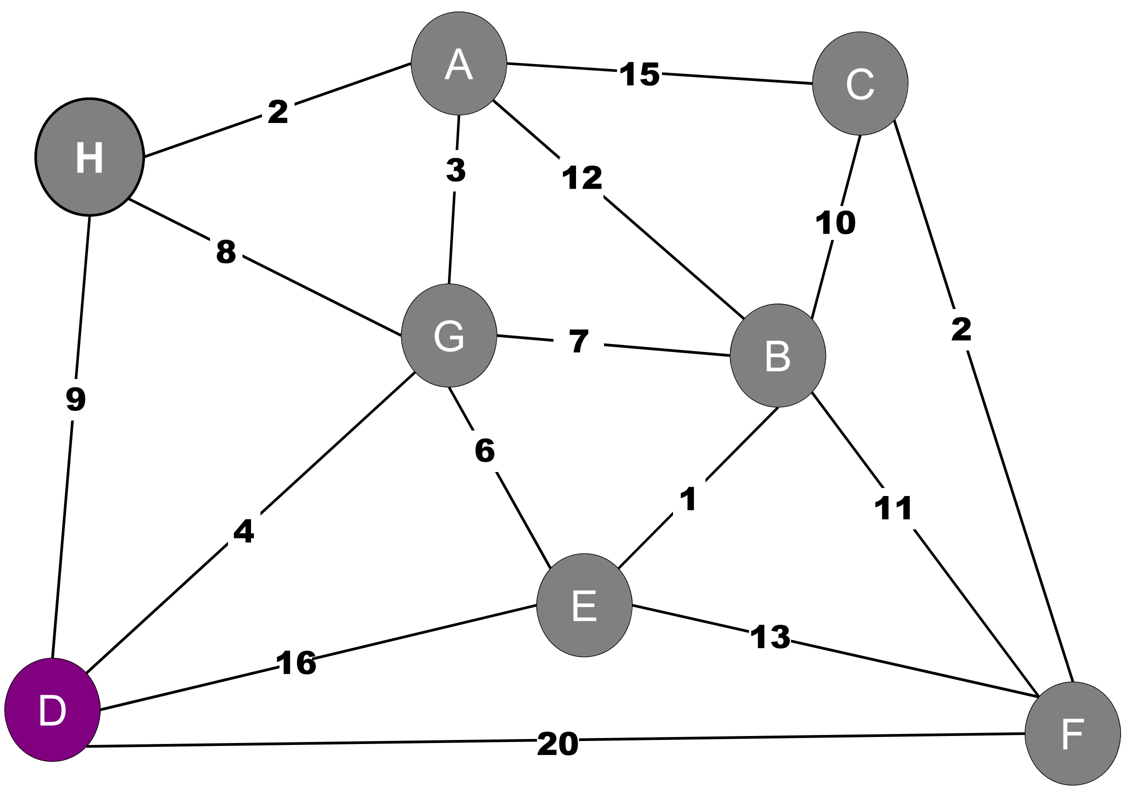
```
HashMap<Node,Node> parent = new HashMap<Node,Node>();
parent.put(start, null);
```

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);
          }
     }
}
```

<u>Assume for today (only):</u>
decreaseKey does nothing if new weight is larger than old weight

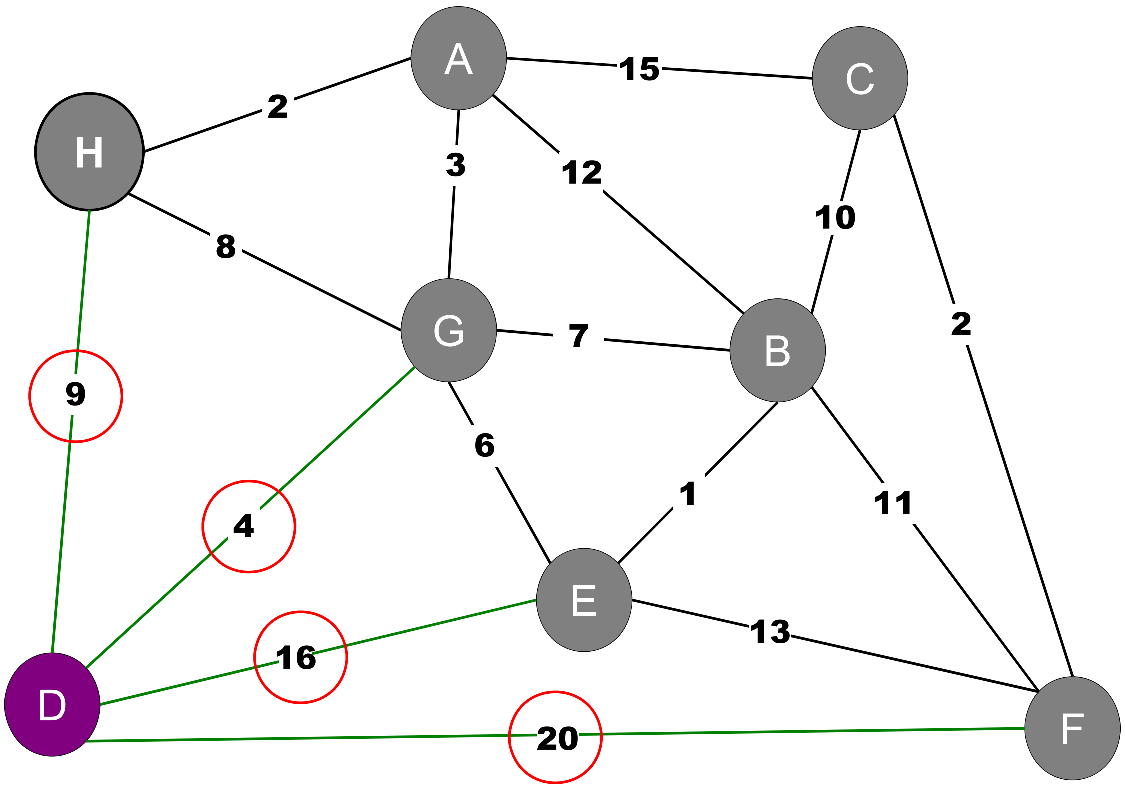
Prim's Example



Vertex	Weight
D	0

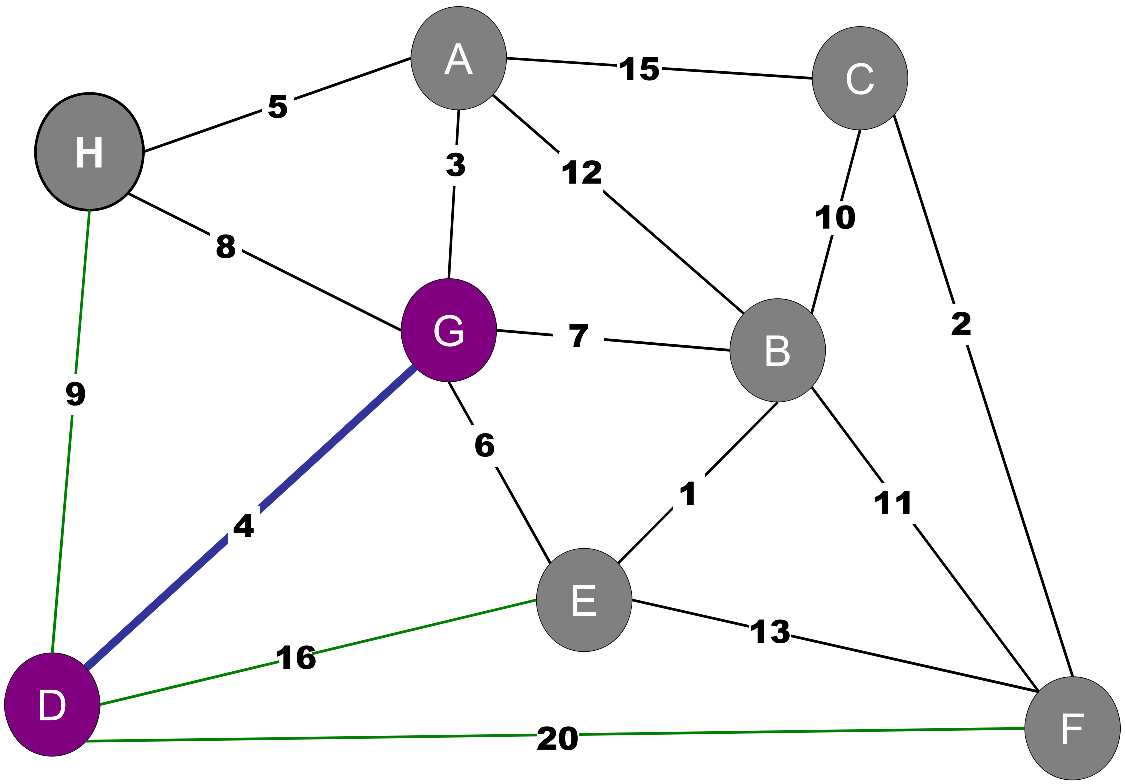
Not drawing infinite weight nodes in PQ.

Prim's Example



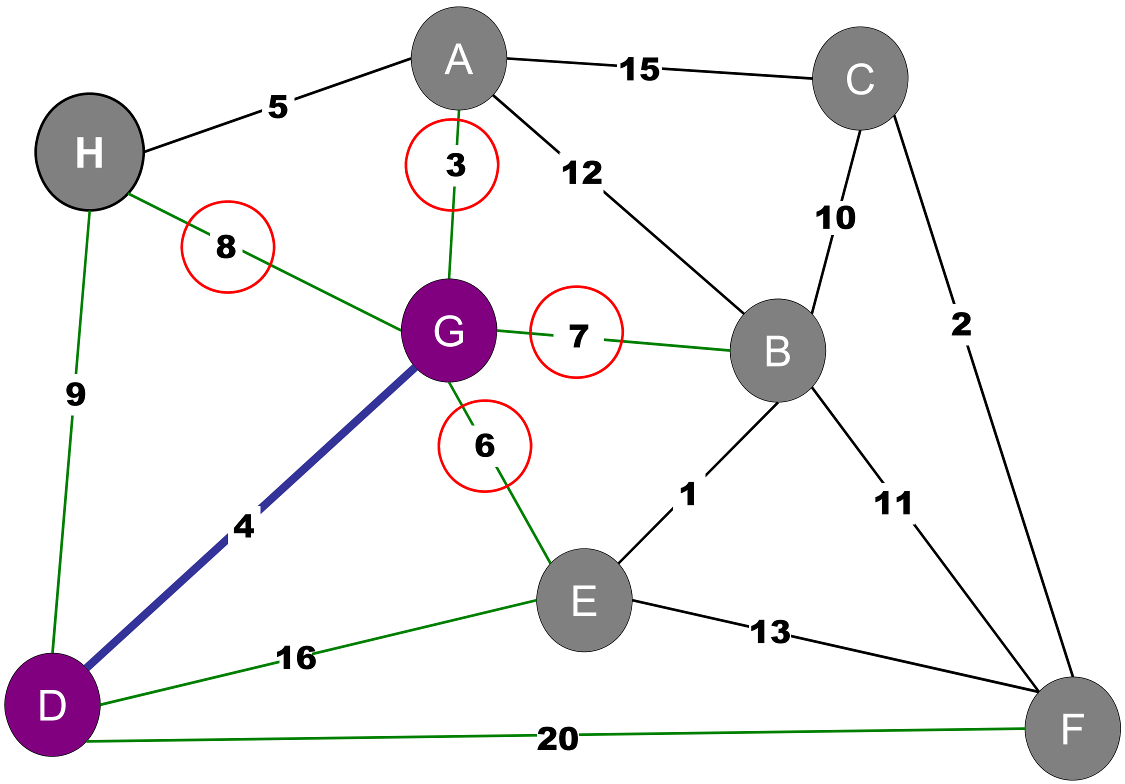
Vertex	Weight
G	4
H	9
E	16
F	20

Prim's Example



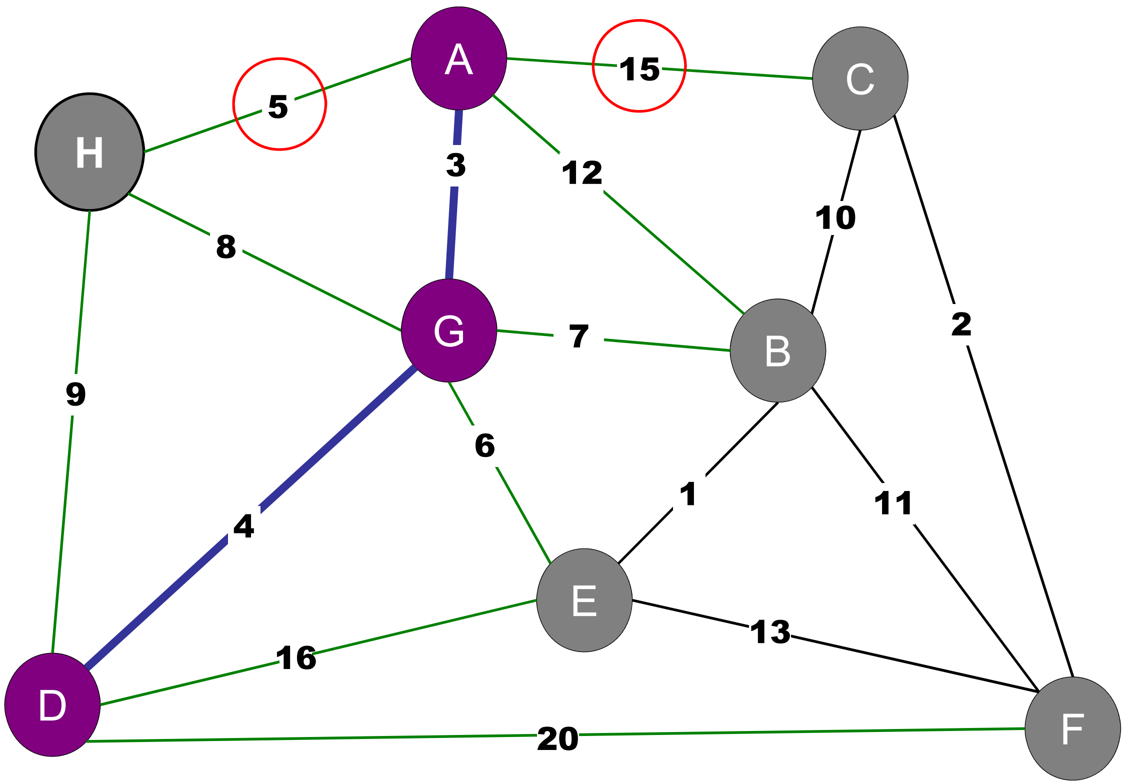
Vertex	Weight
H	9
E	16
F	20

Prim's Example



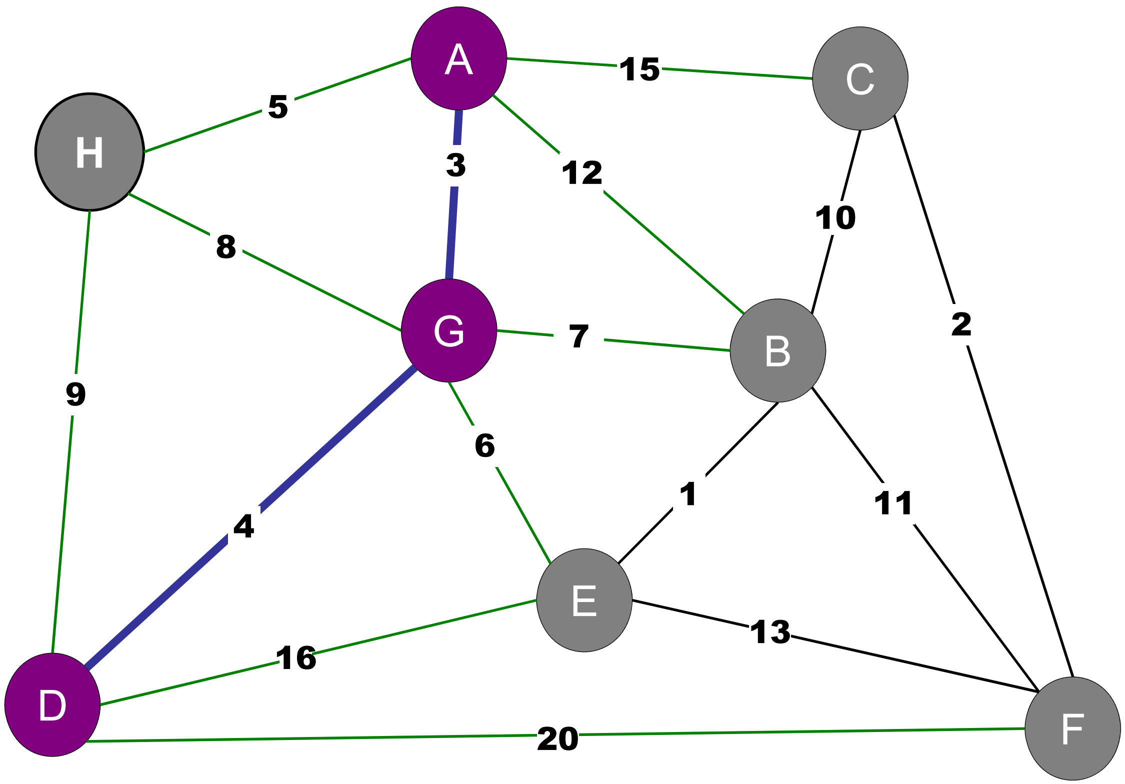
Vertex	Weight
A	**3**
E	**16->6**
B	**7**
H	**9->8**
F	20

Prim's Example



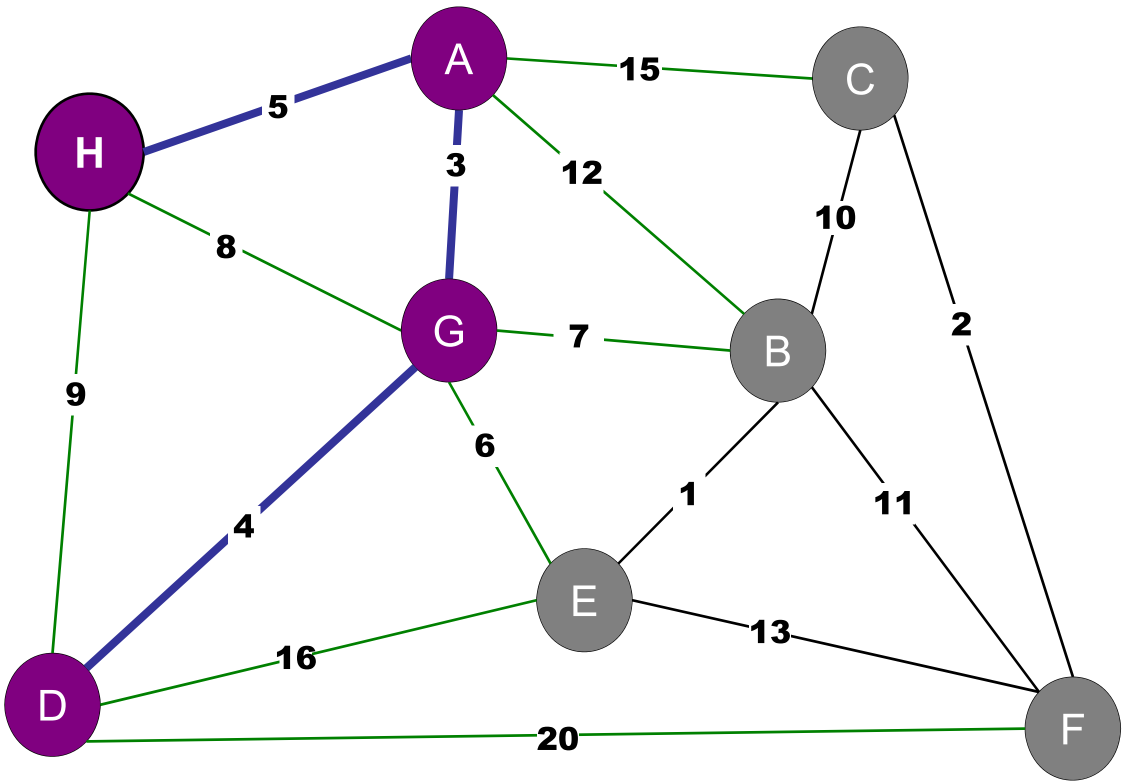
Vertex	Weight
H	**8->5**
E	6
B	7
C	**15**
F	20

Prim’s Example



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

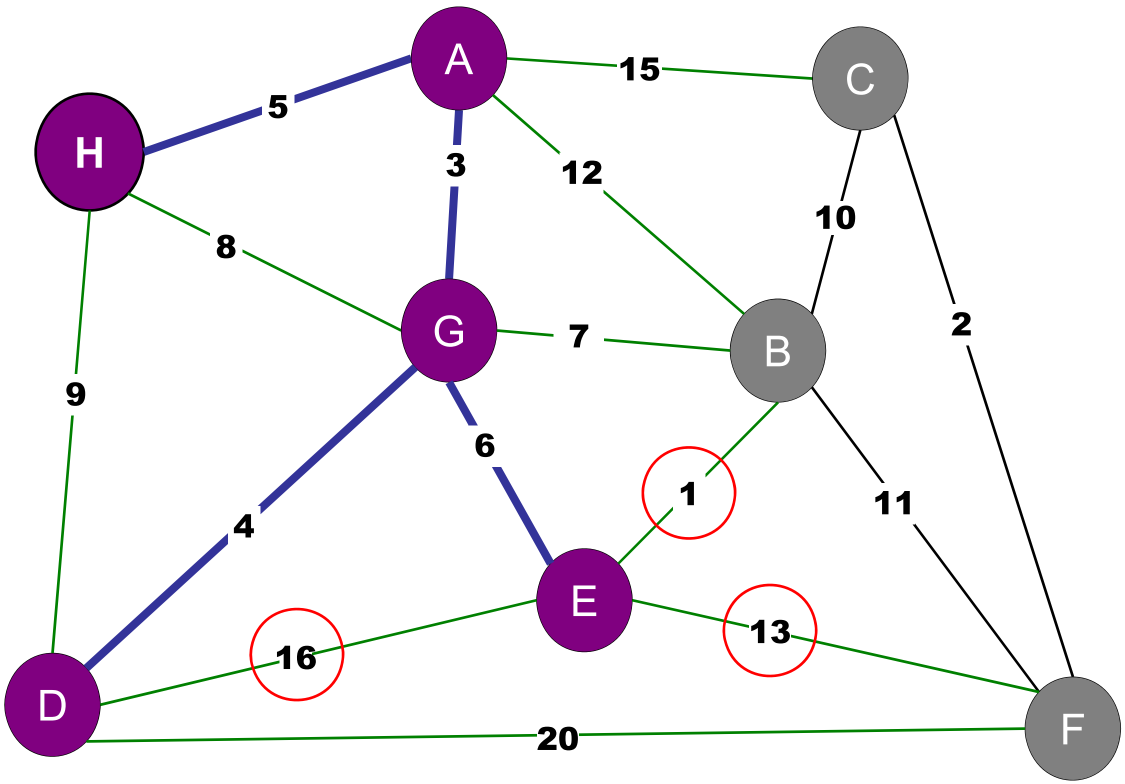
Prim’s Example



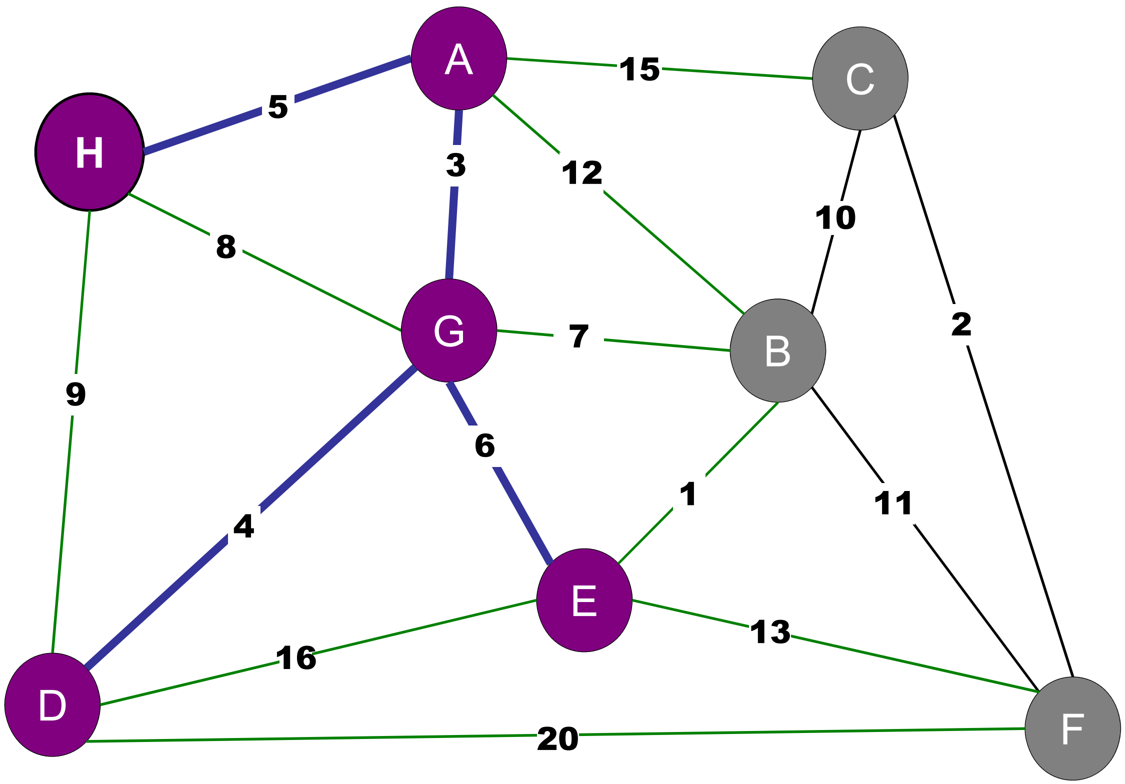
Vertex	Weight
E	6
B	7
C	15
F	20

Prim's Example

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



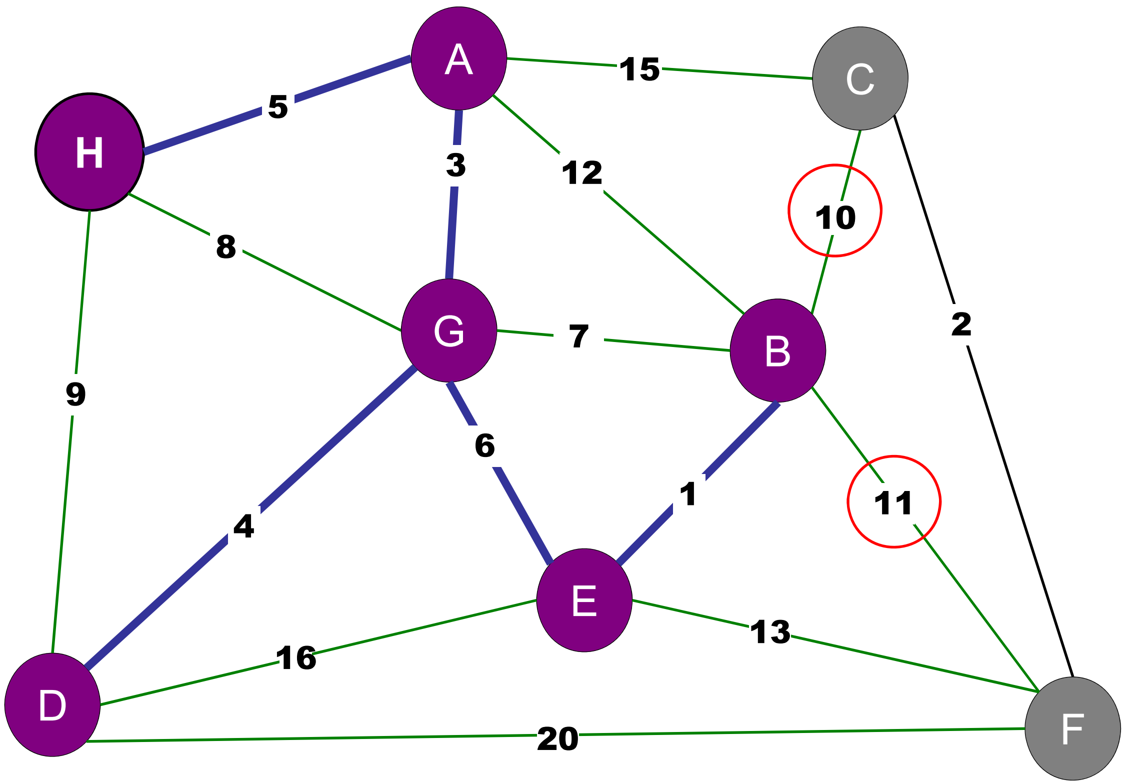
Prim's Example



Vertex	Weight
B	1
C	15
F	13

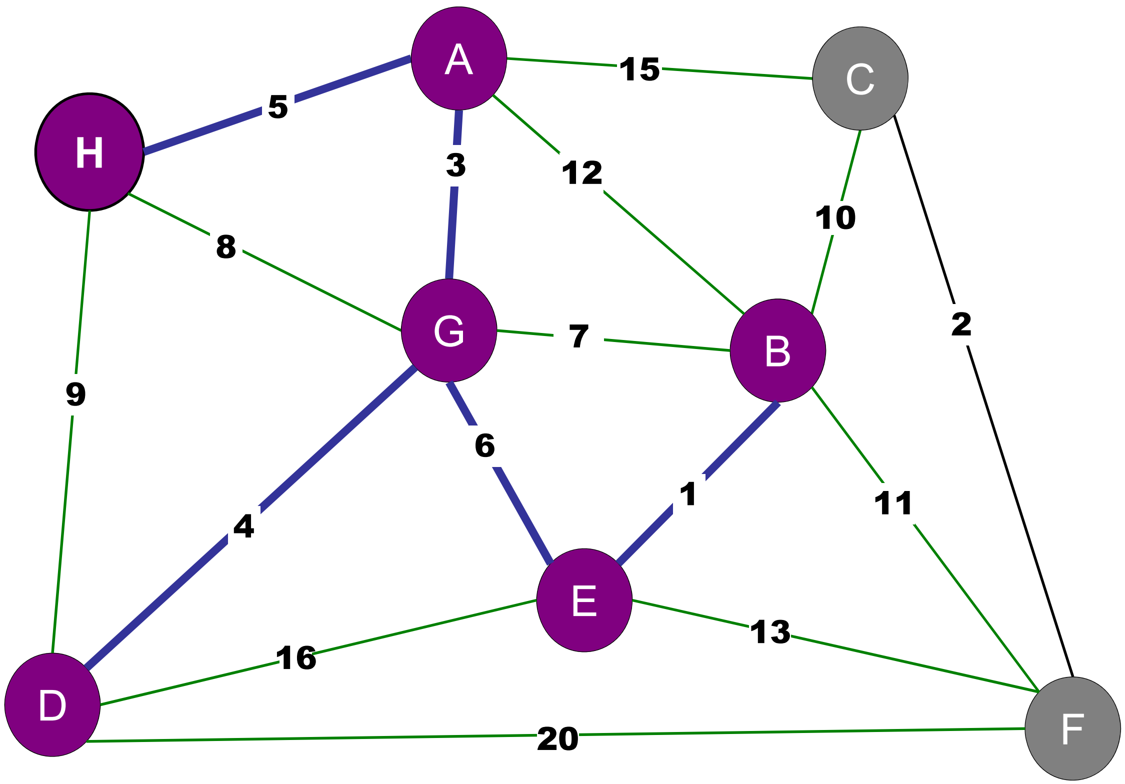
Prim's Example

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



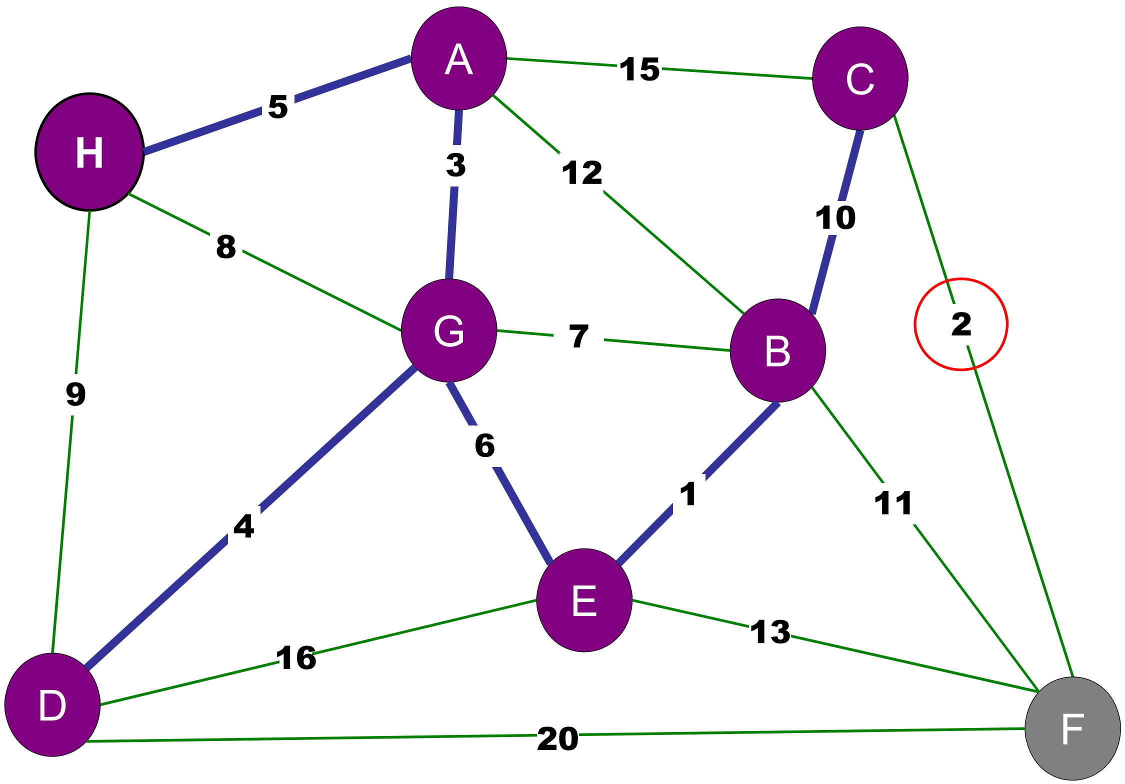
Prim’s Example

Vertex	Weight
C	10
F	11



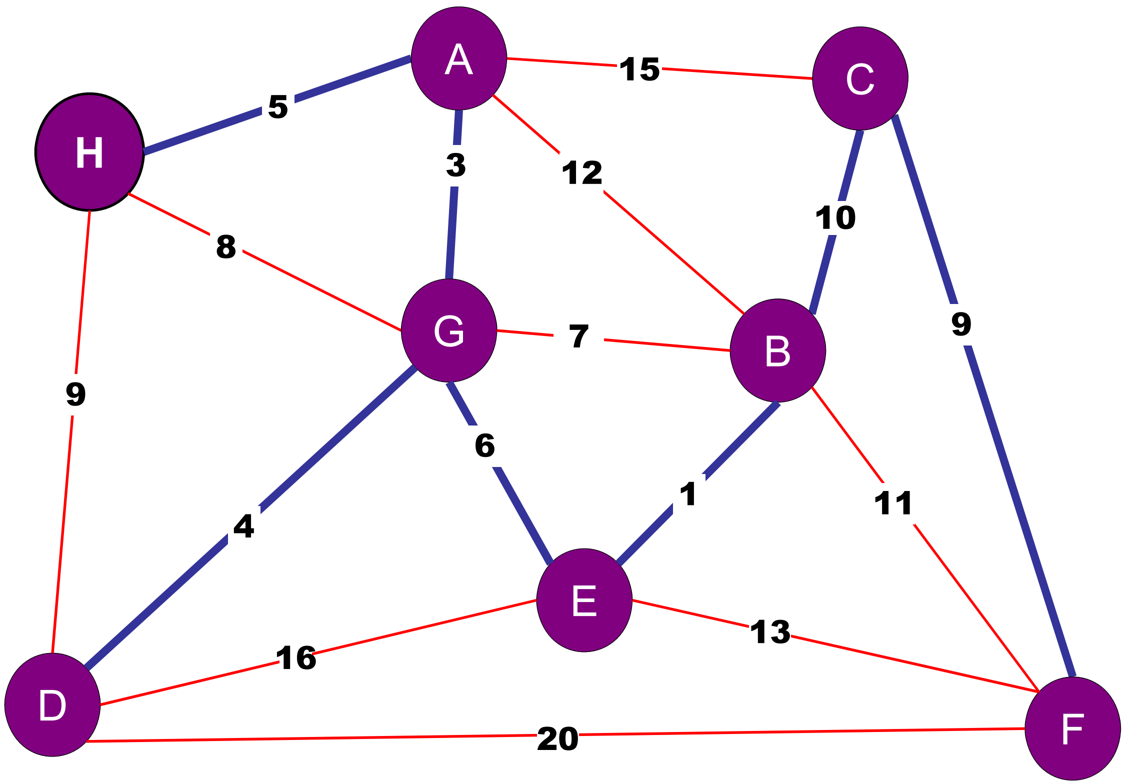
Prim's Example

Vertex	Weight
F	11->2



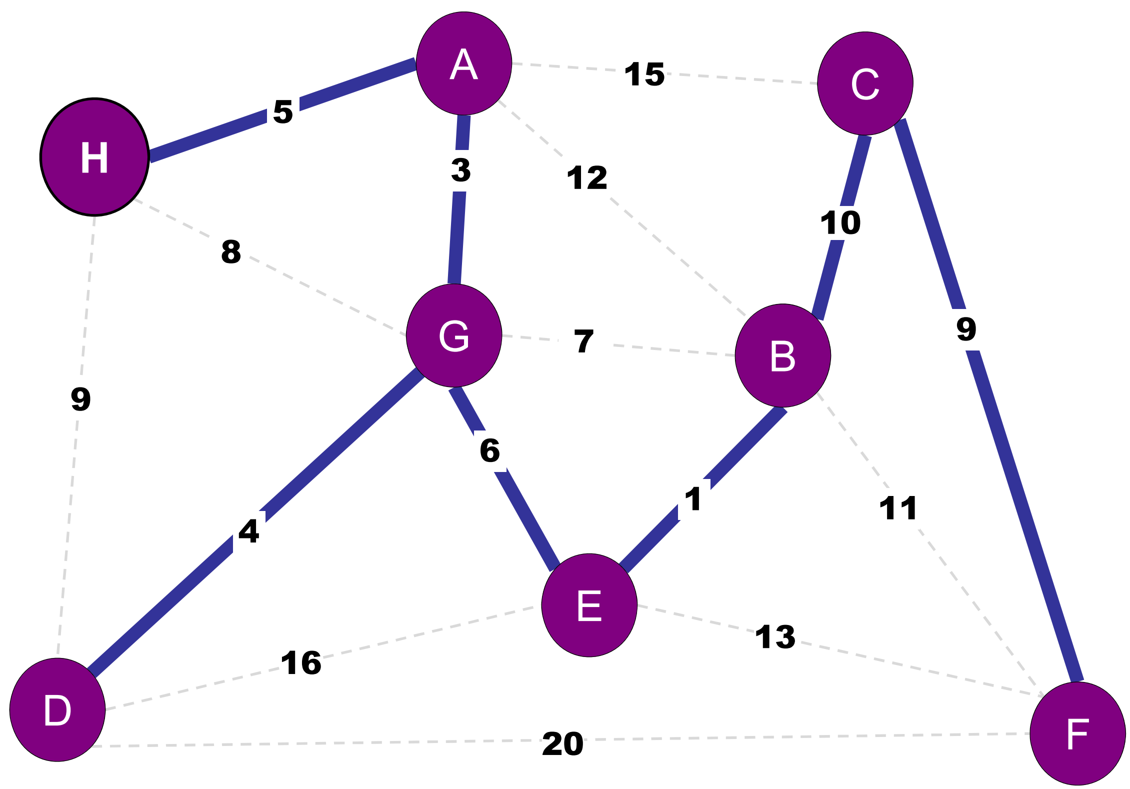
Prim's Example

Vertex	Weight



Prim’s Example

Vertex	Weight

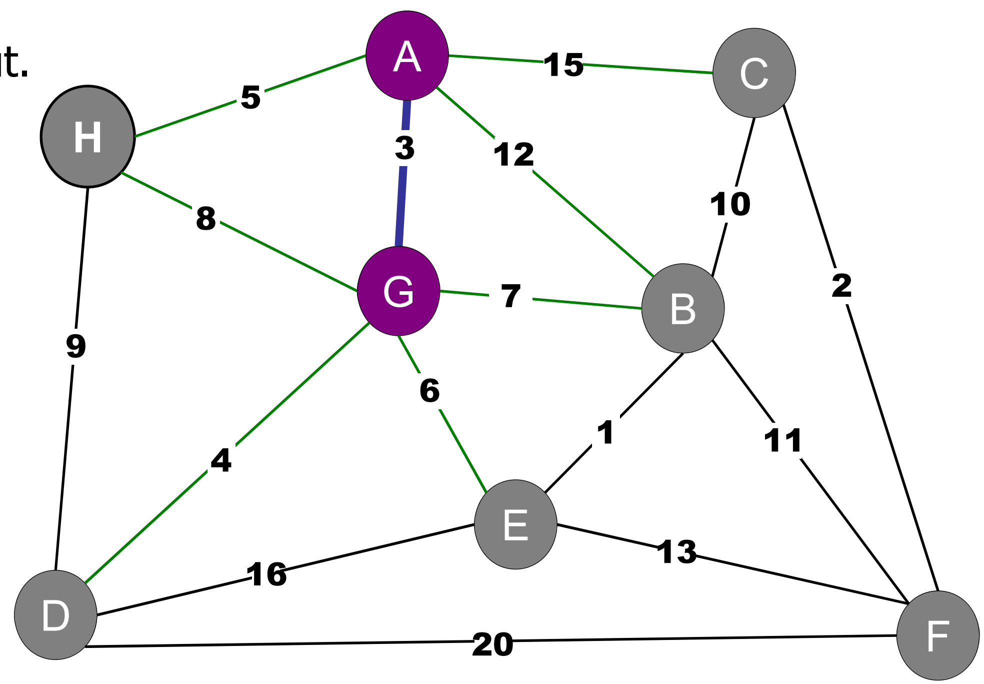


Prim's Algorithm

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 (BLUE RULE).
- 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)

ARCHIPELAGO
is open

Prim's Algorithm

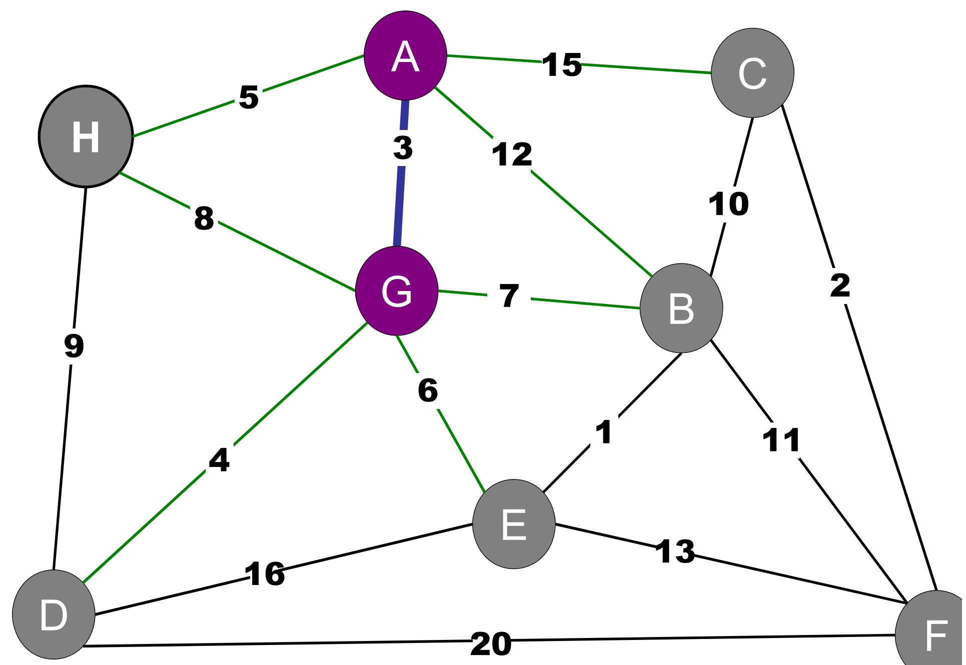
Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

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

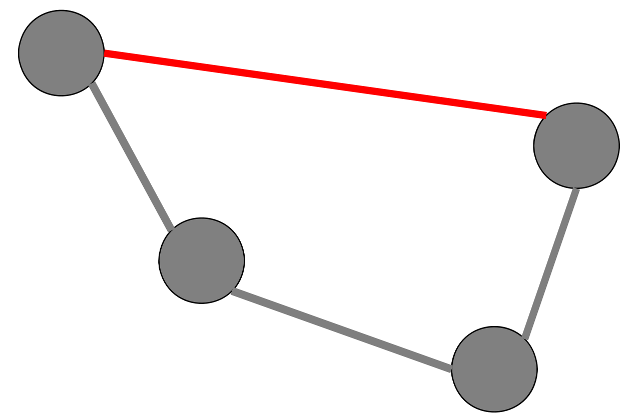
Generic MST Algorithm

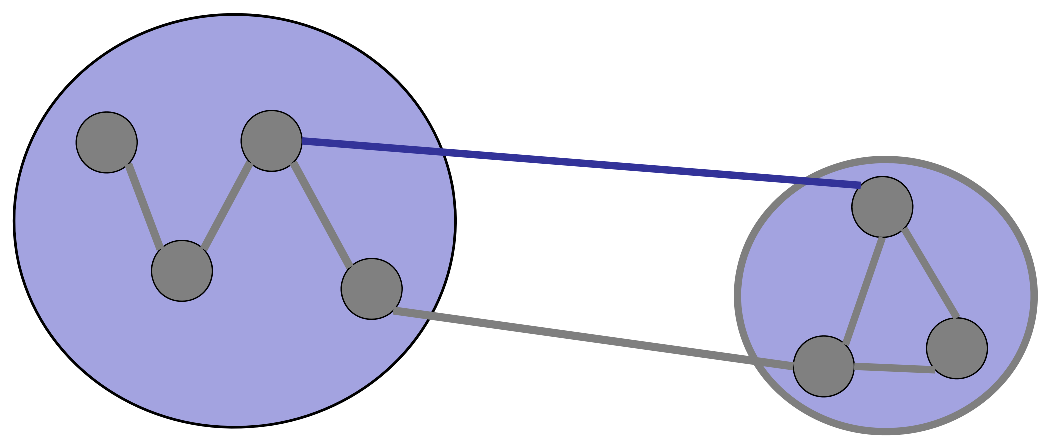
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

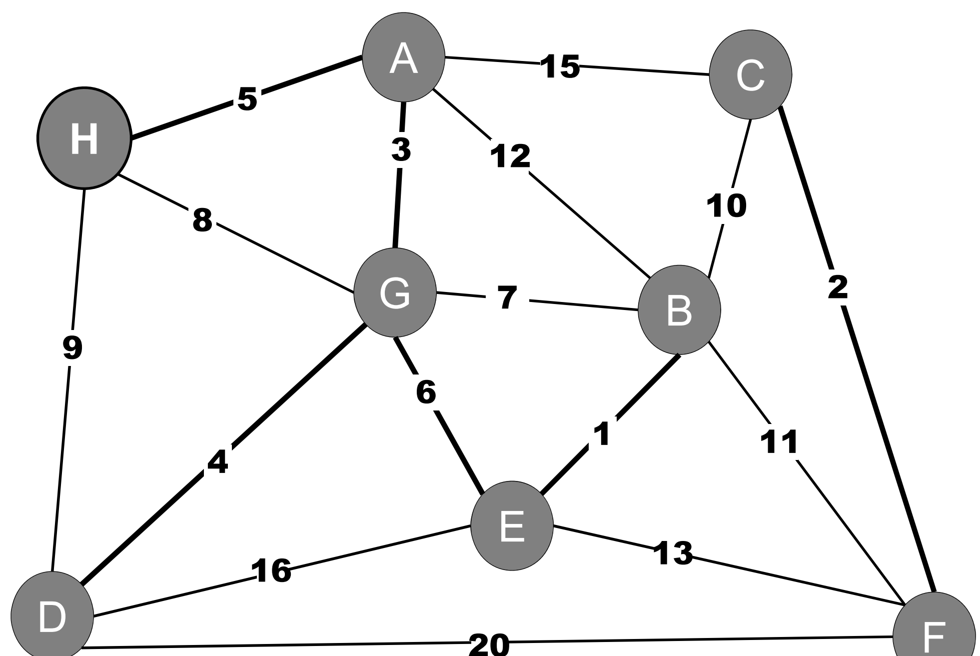
until no more edges can be colored.





Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

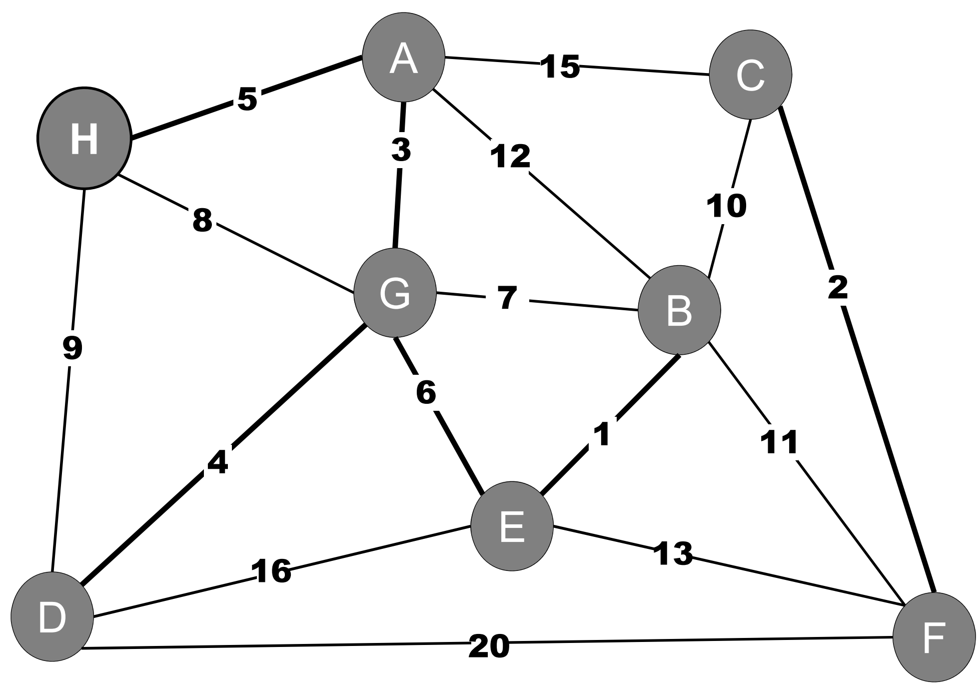


Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

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



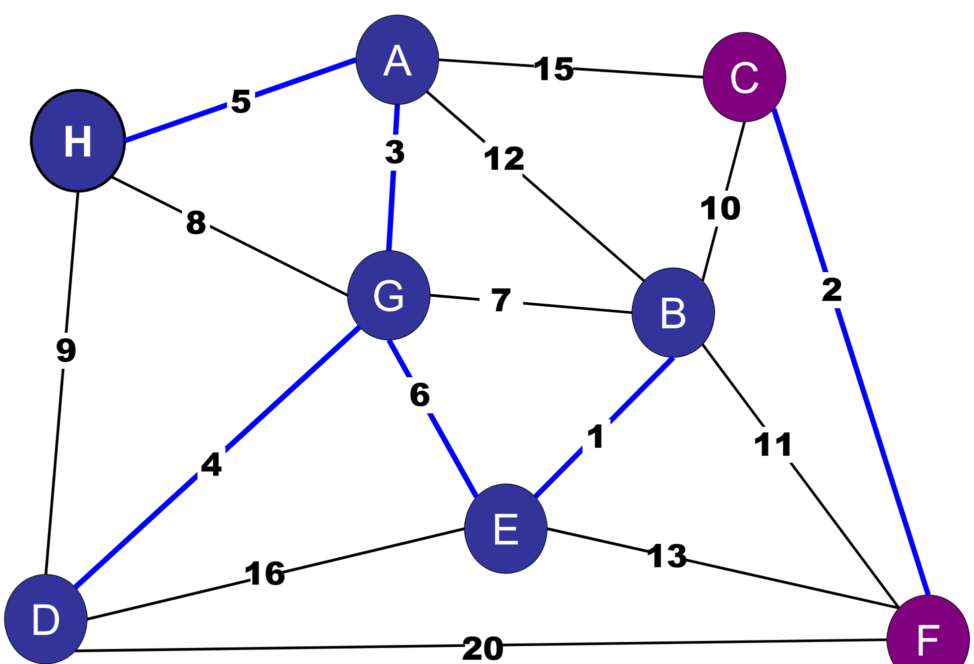
Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

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

Must be the heaviest edge on the cycle!

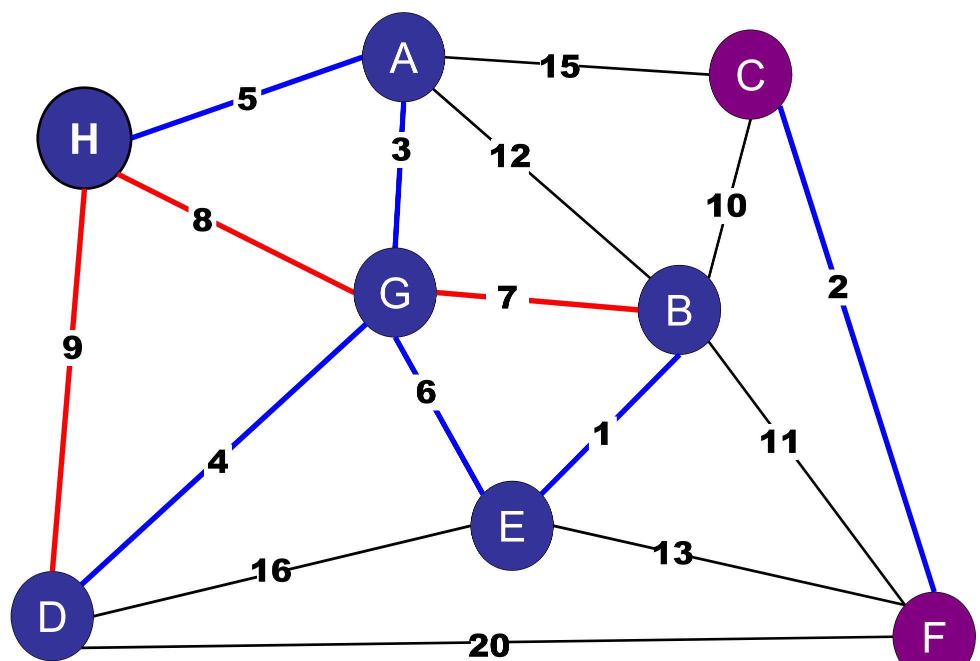


Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

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

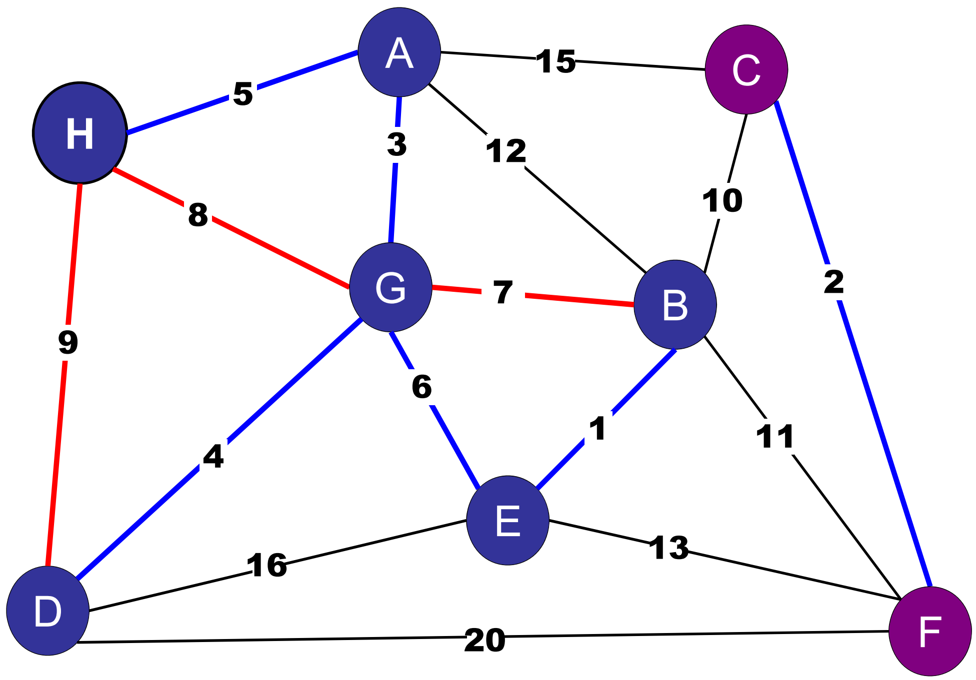


Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

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



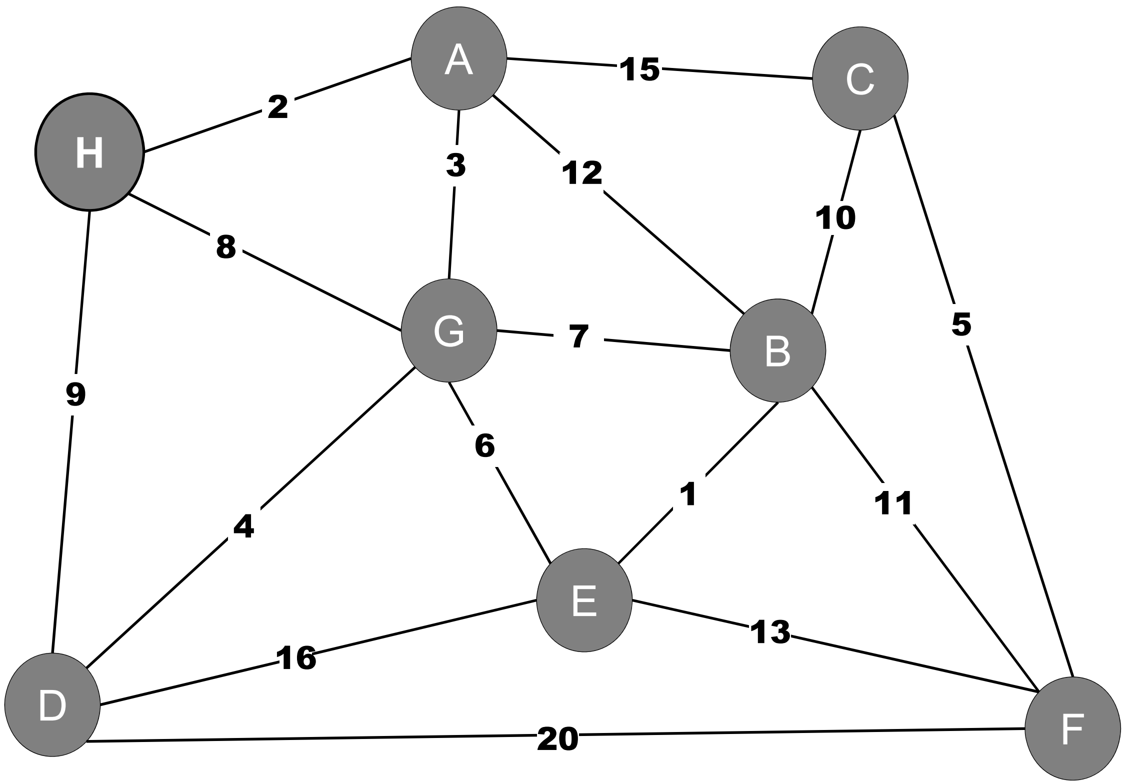
Data structure:

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

Kruskal's Algorithm

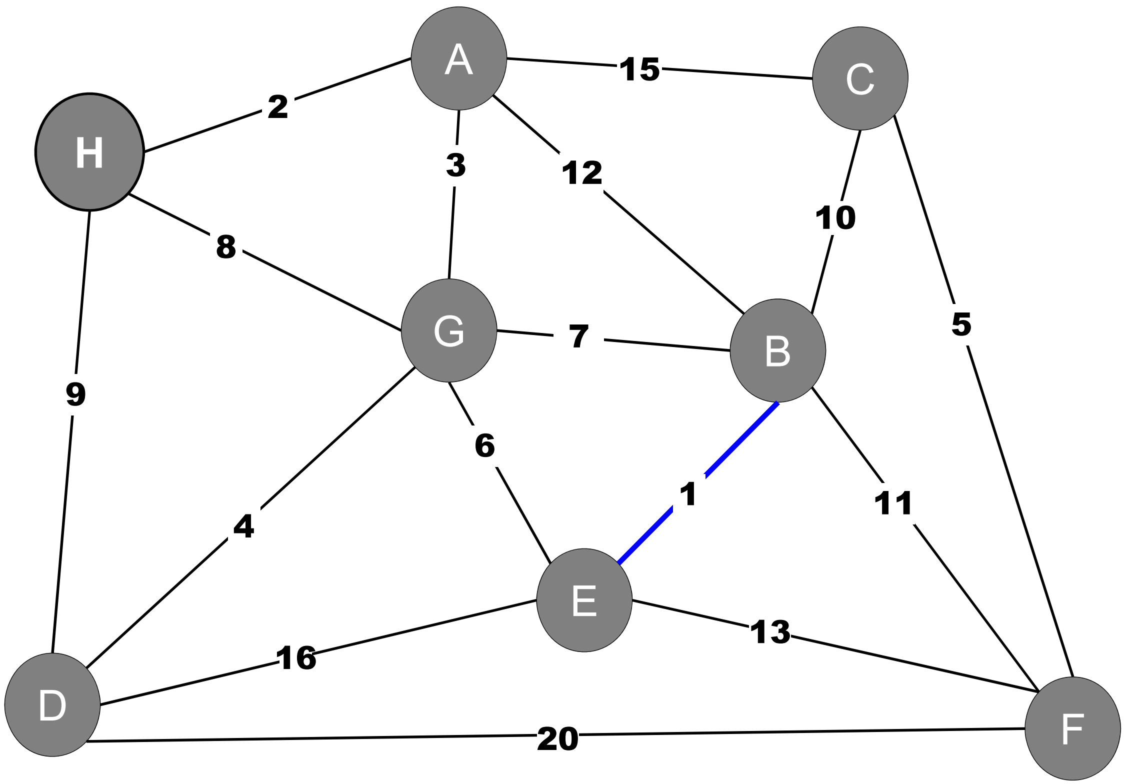
```
// Sort edges and initialize
Edge[] sortedEdges = sort(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());


// 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



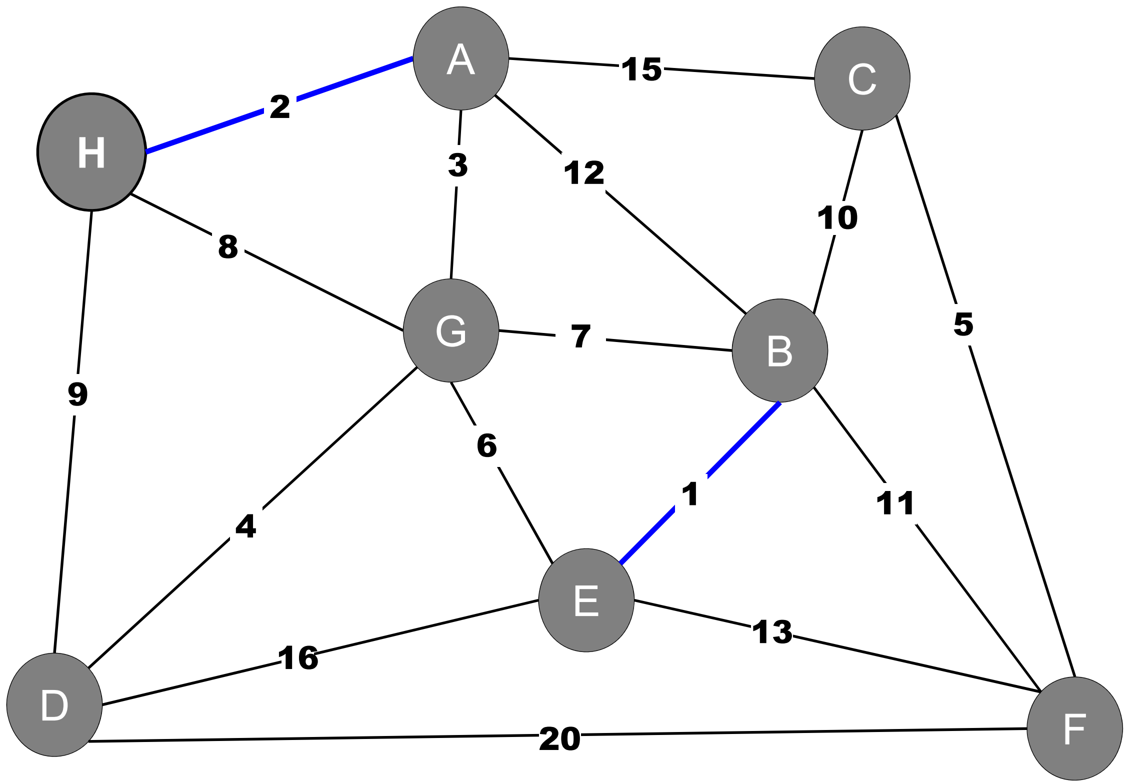
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



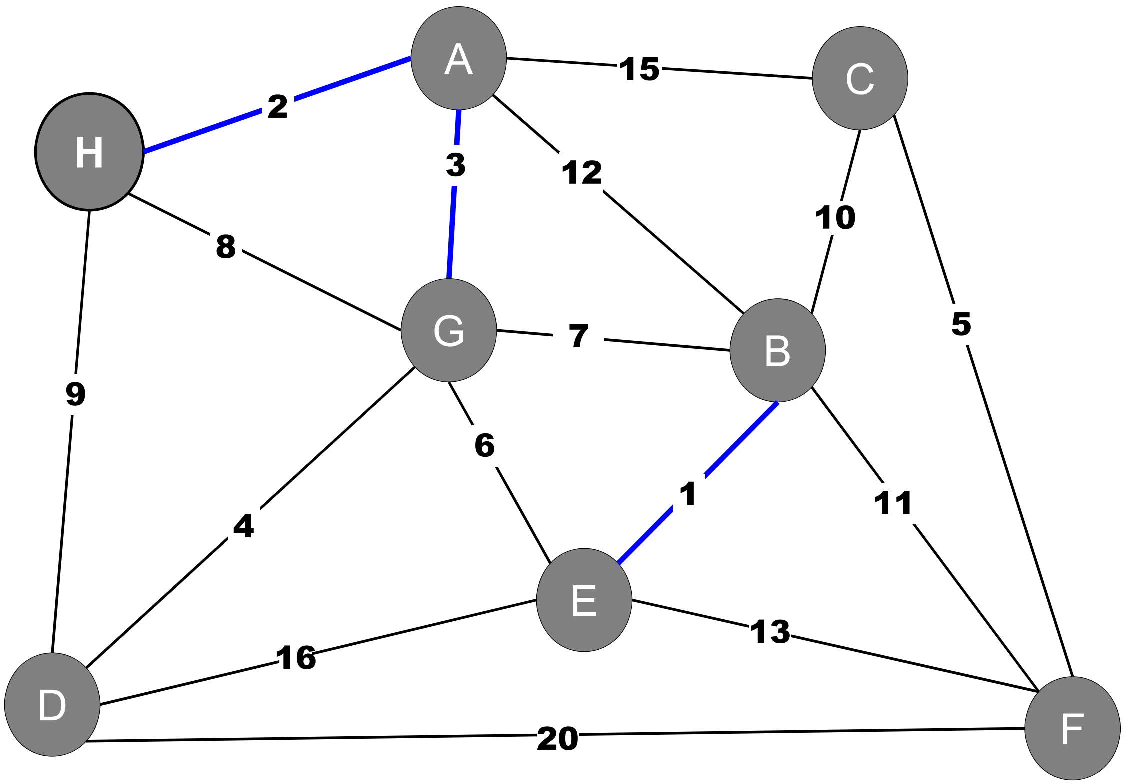
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal’s Example



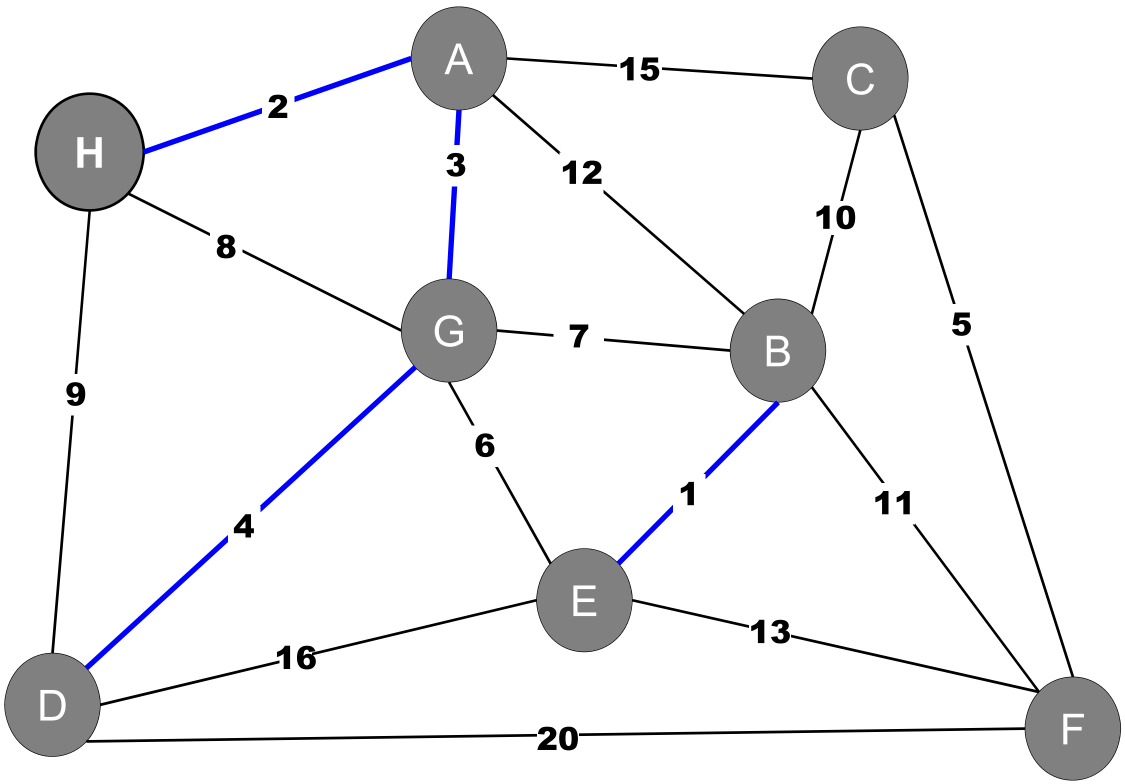
Weight	Edge
1	(E,B)
2	**(C,F)**
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



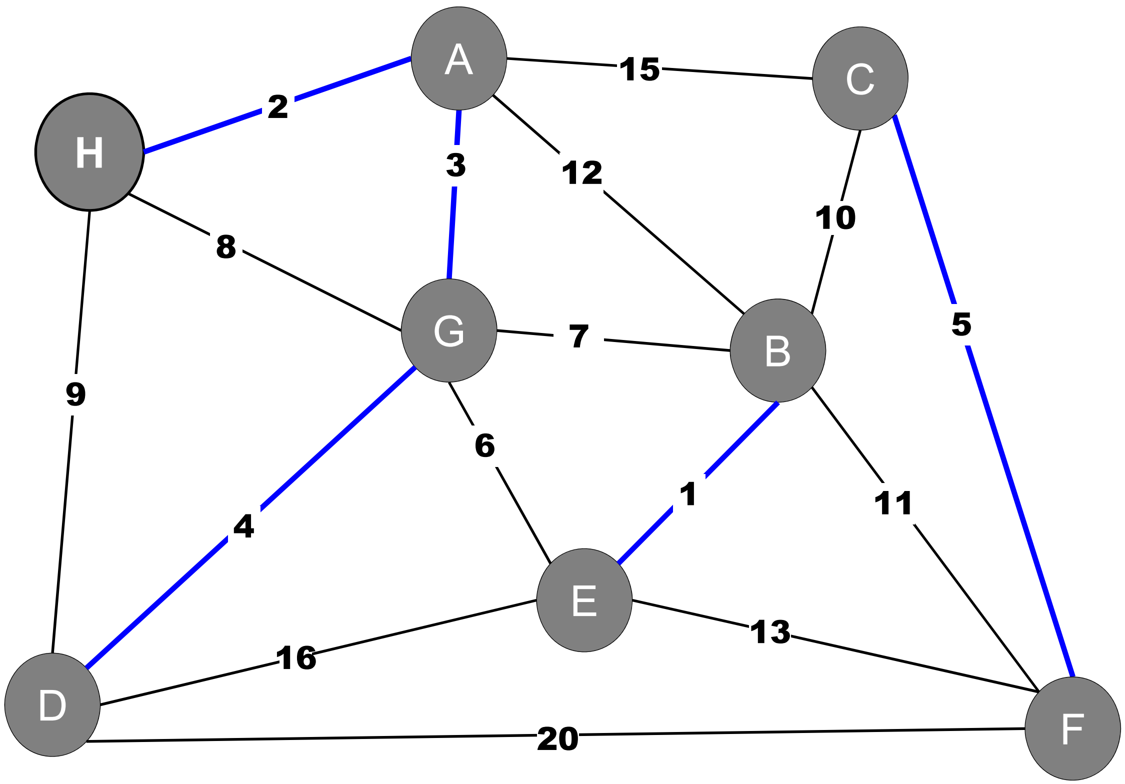
Weight	Edge
1	(E,B)
2	(C,F)
3	**(A,G)**
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal’s Example



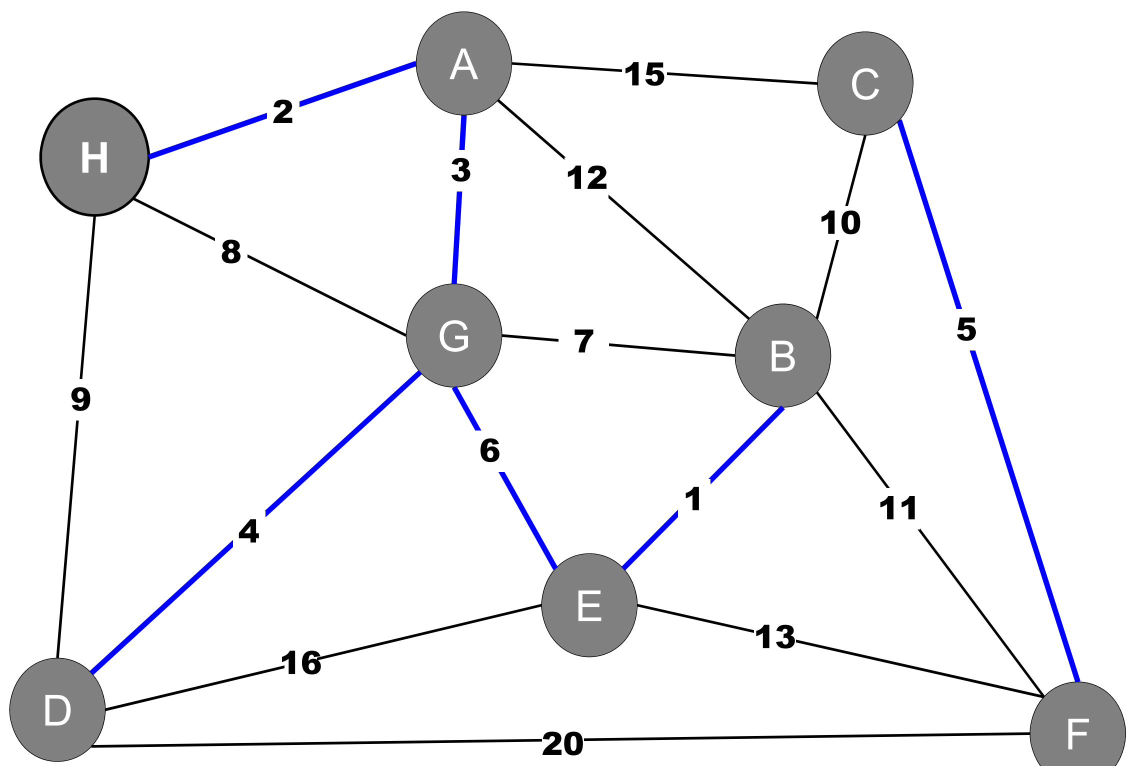
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	**(D,G)**
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



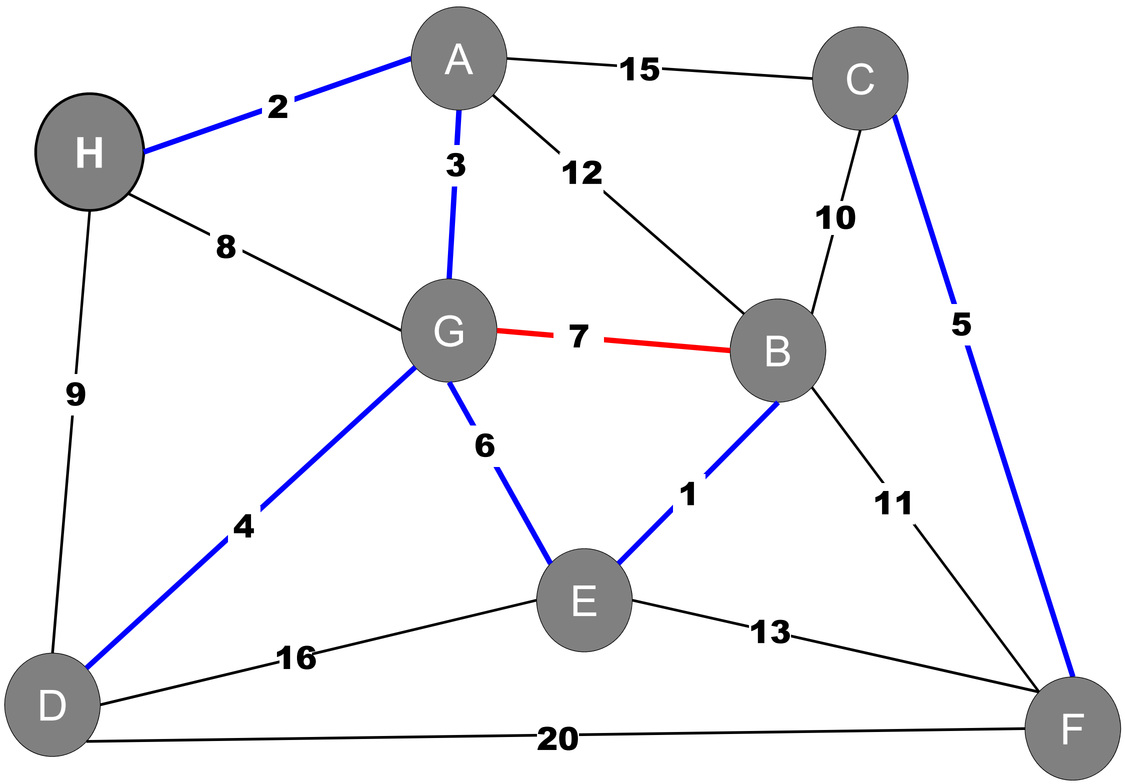
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	**(C,F)**
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



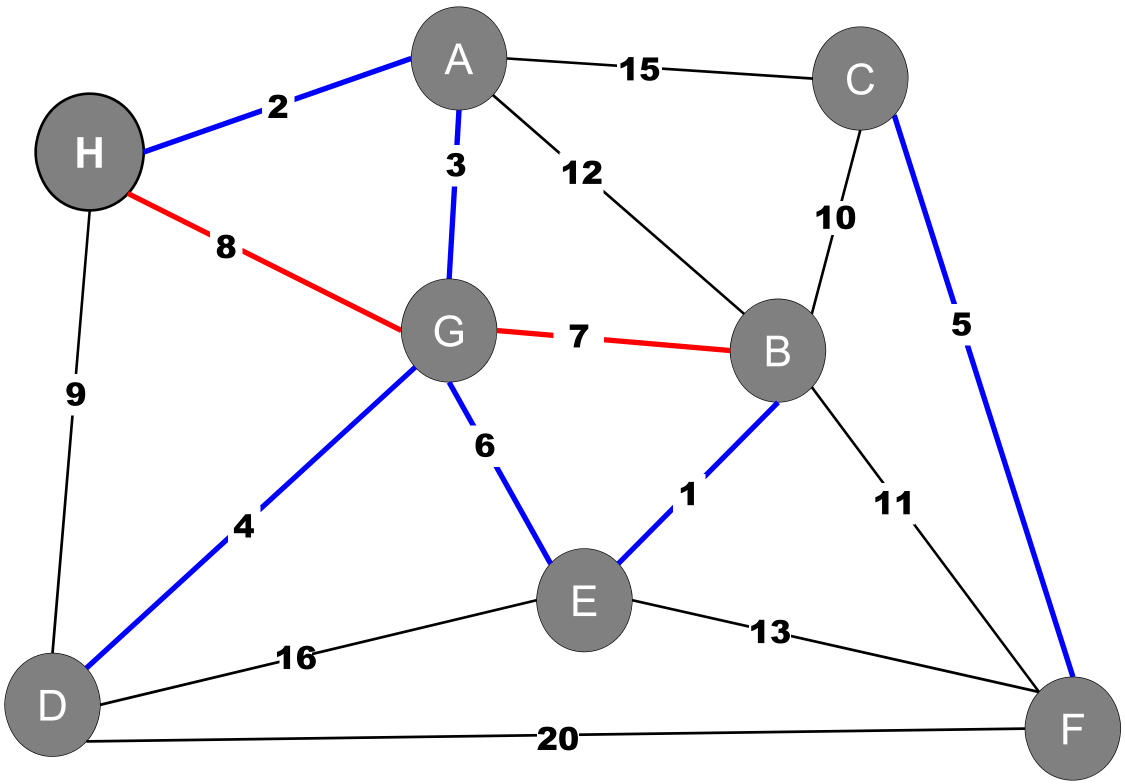
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	**(E,G)**
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



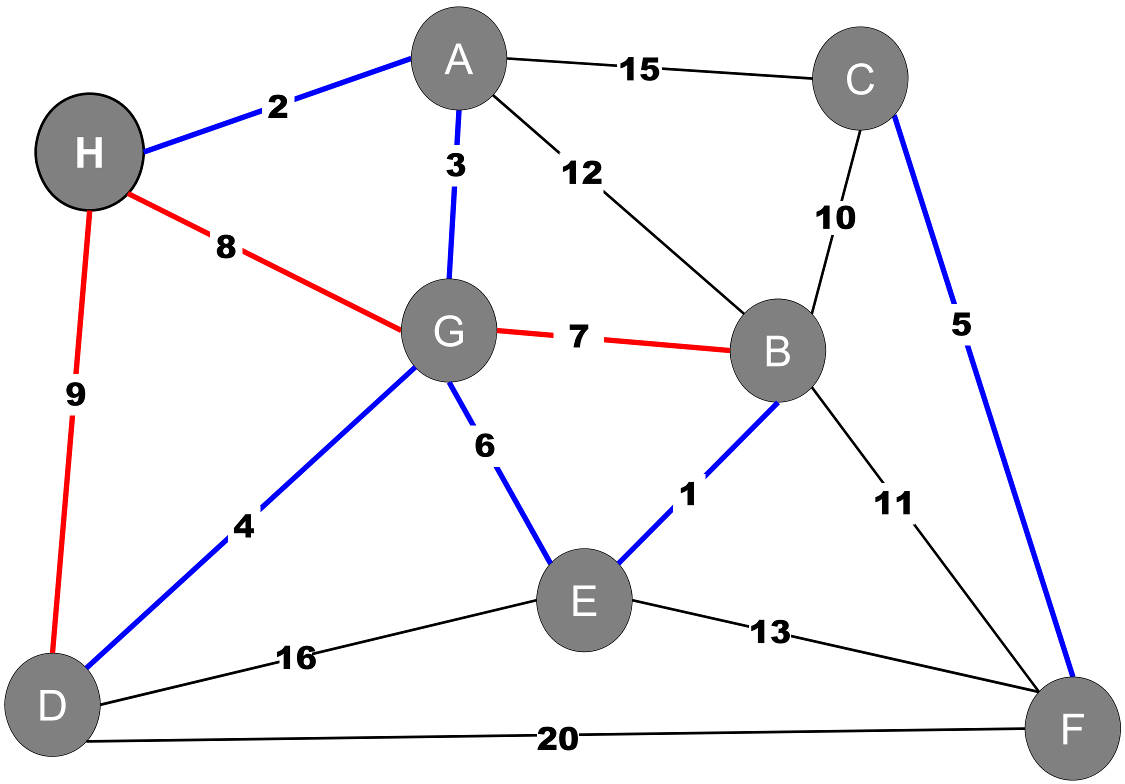
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	**(B,G)**
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal's Example



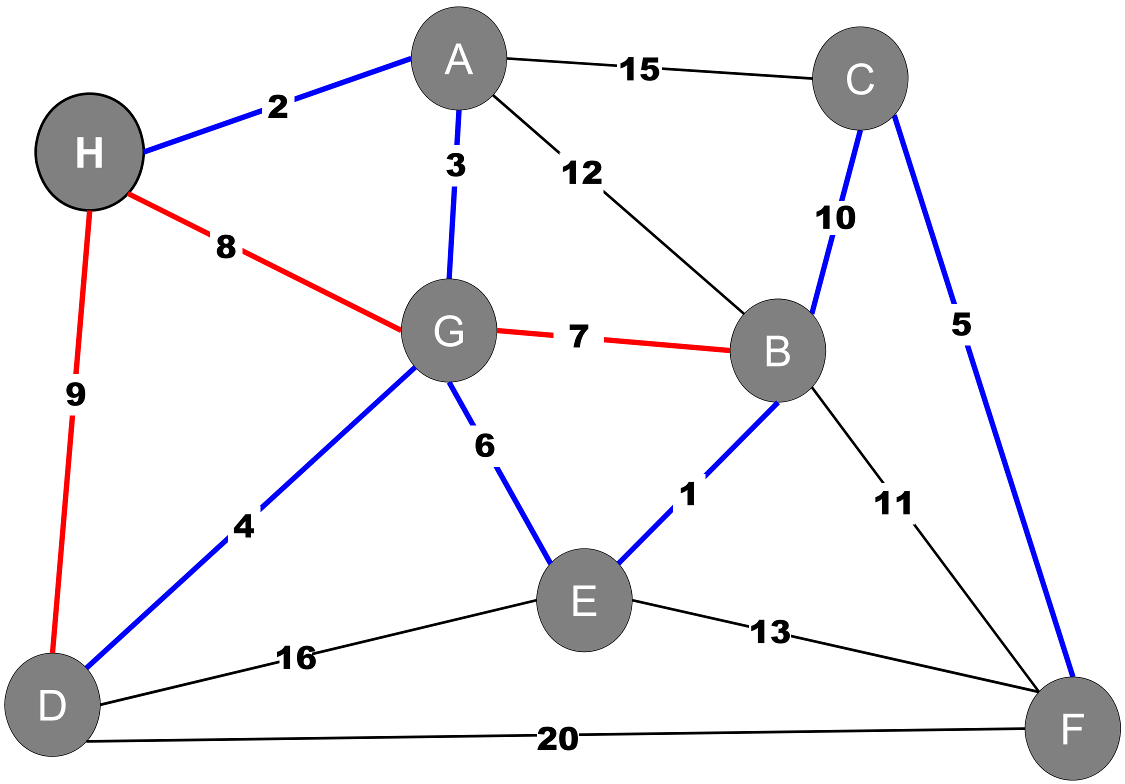
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	**(G,H)**
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Kruskal’s Example



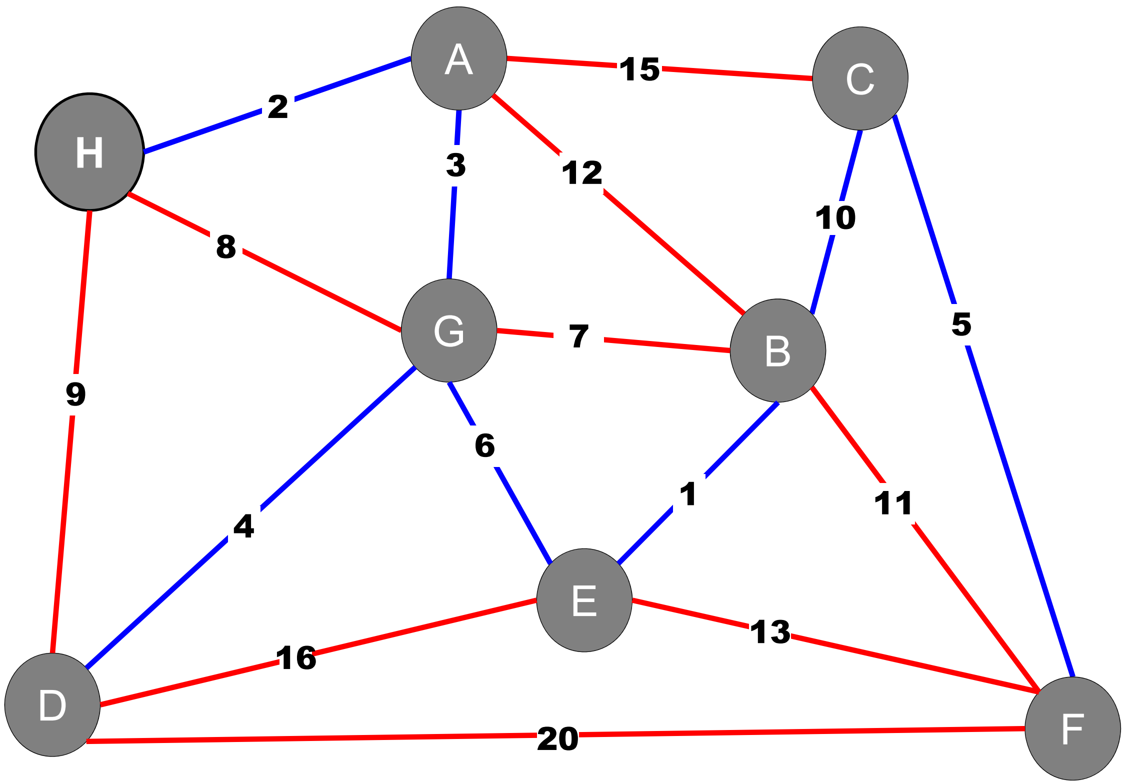
Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	**(D,G)**
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

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

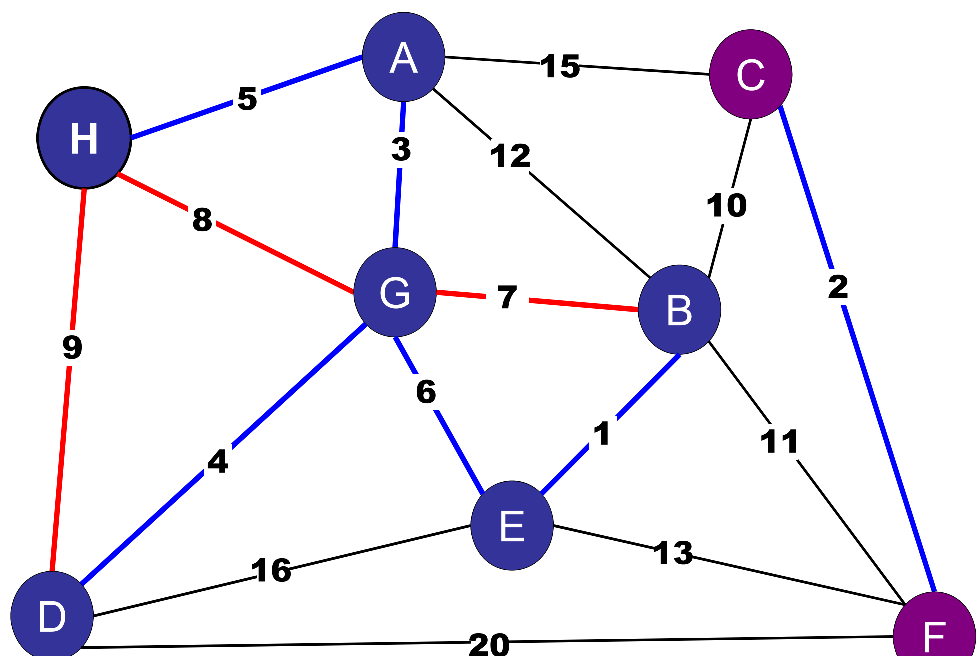
Kruskal's Algorithm. (Kruskal 1956)

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.



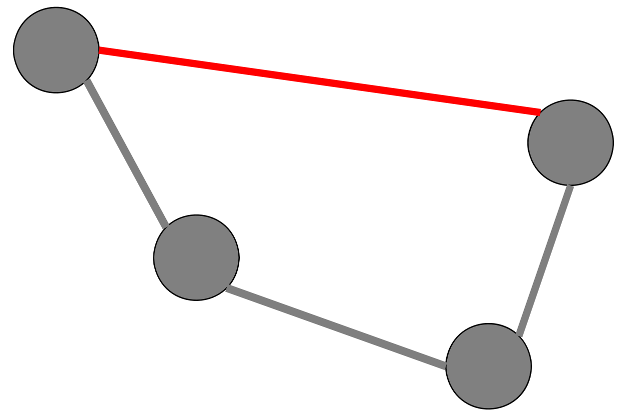
Generic MST Algorithm

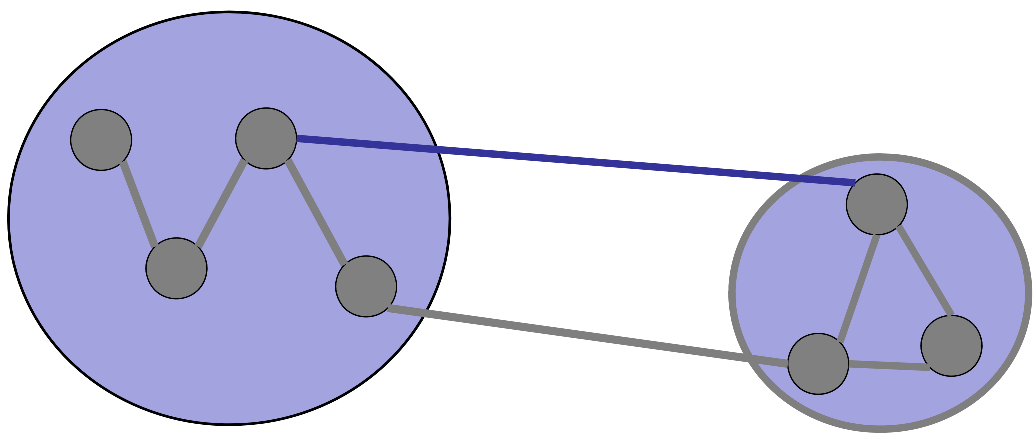
Greedy Algorithm:

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)

ARCHIPELAGO

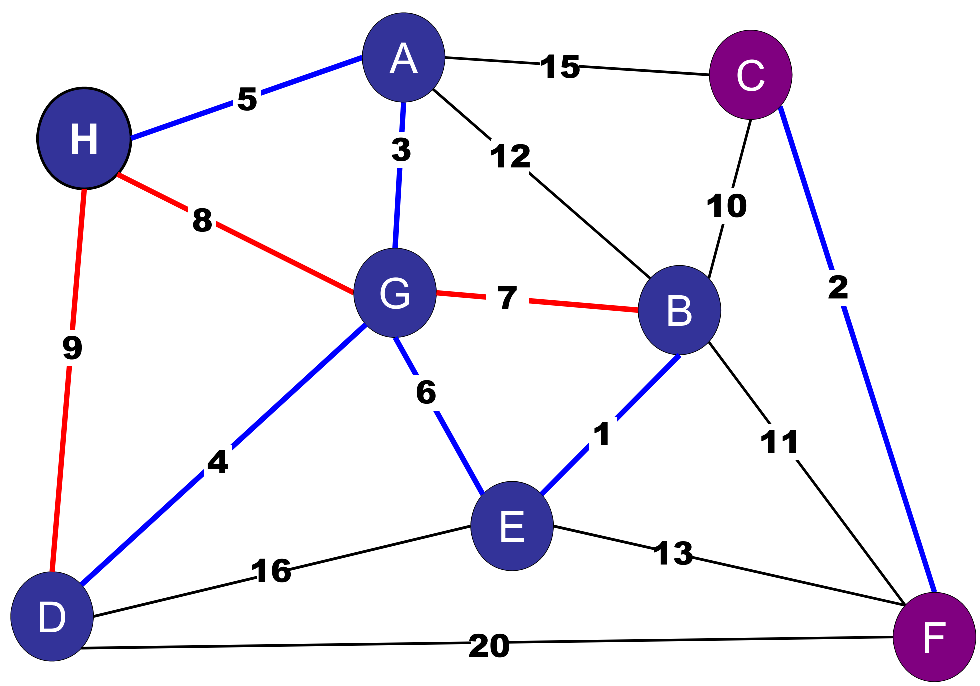
is open

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: $O(\alpha(n))$ or O(log V)
  - Union: $O(\alpha(n))$ 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.

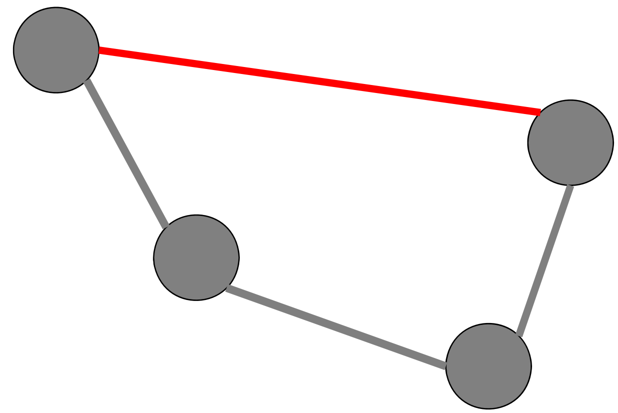
Generic MST Algorithm

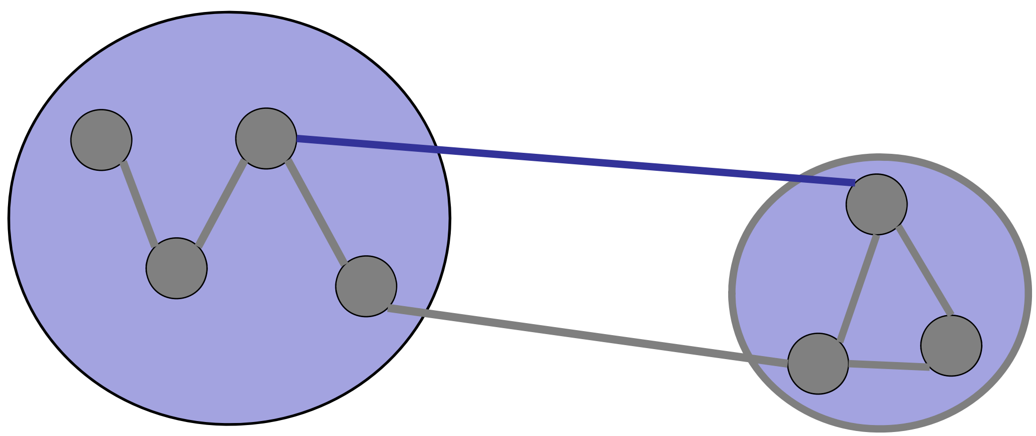
Greedy Algorithm:

Repeat:

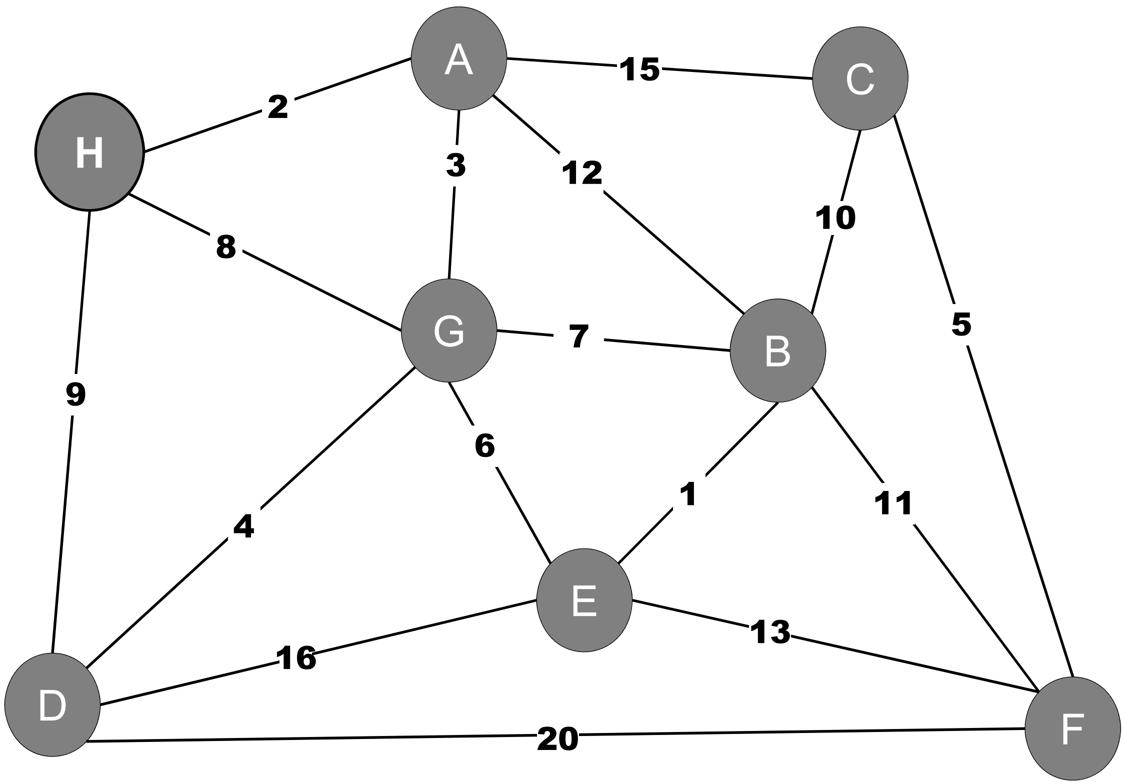
Apply red rule or blue rule to an arbitrary edge.

until no more edges can be colored.





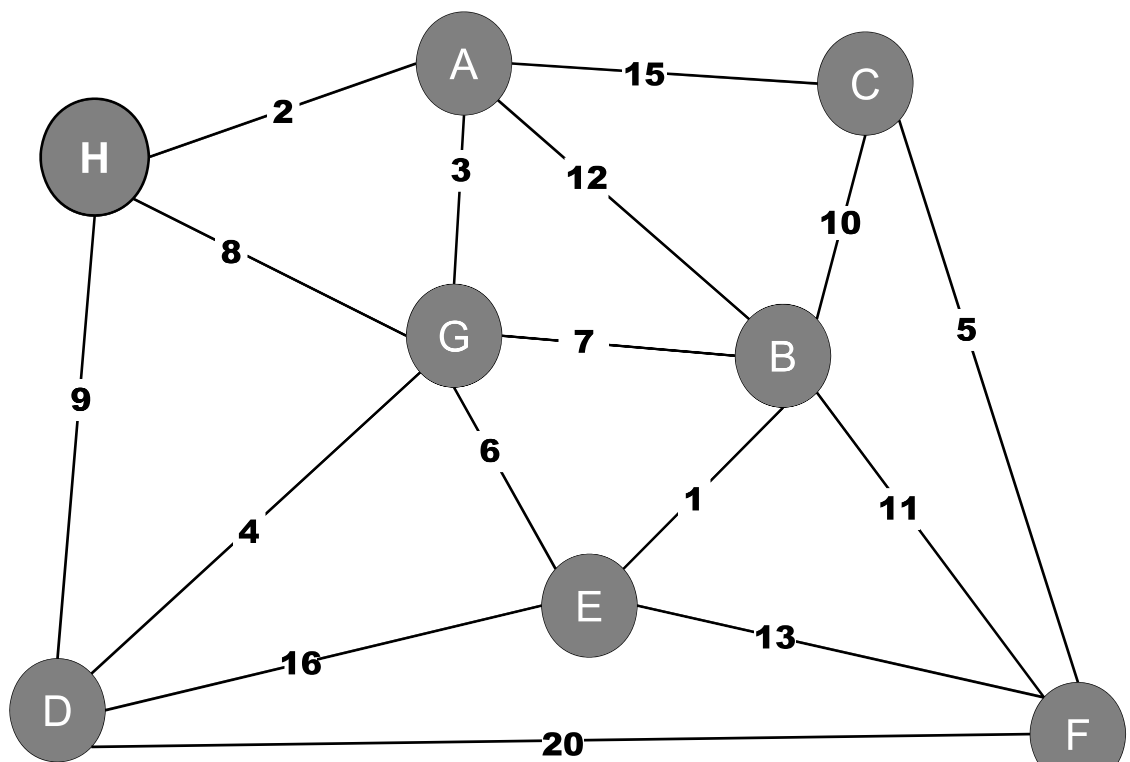
Boruvka's Example



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)

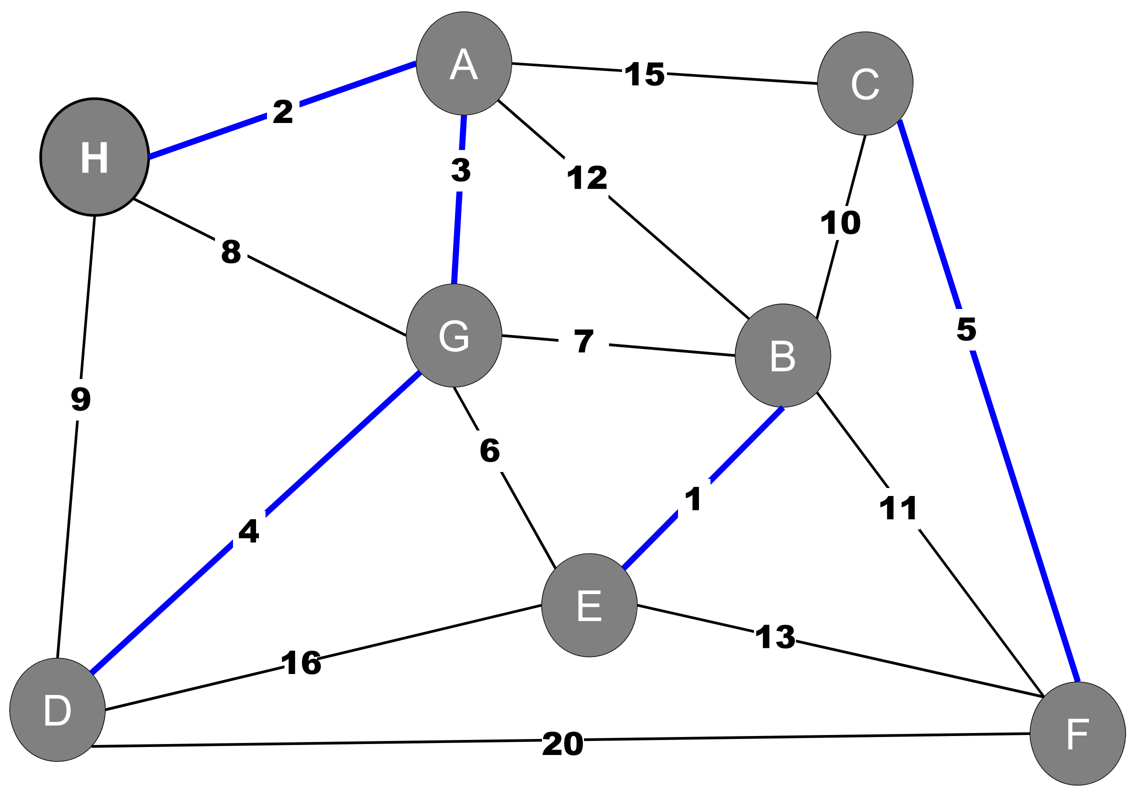
Boruvka's Example



The minimum adjacent edge!

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

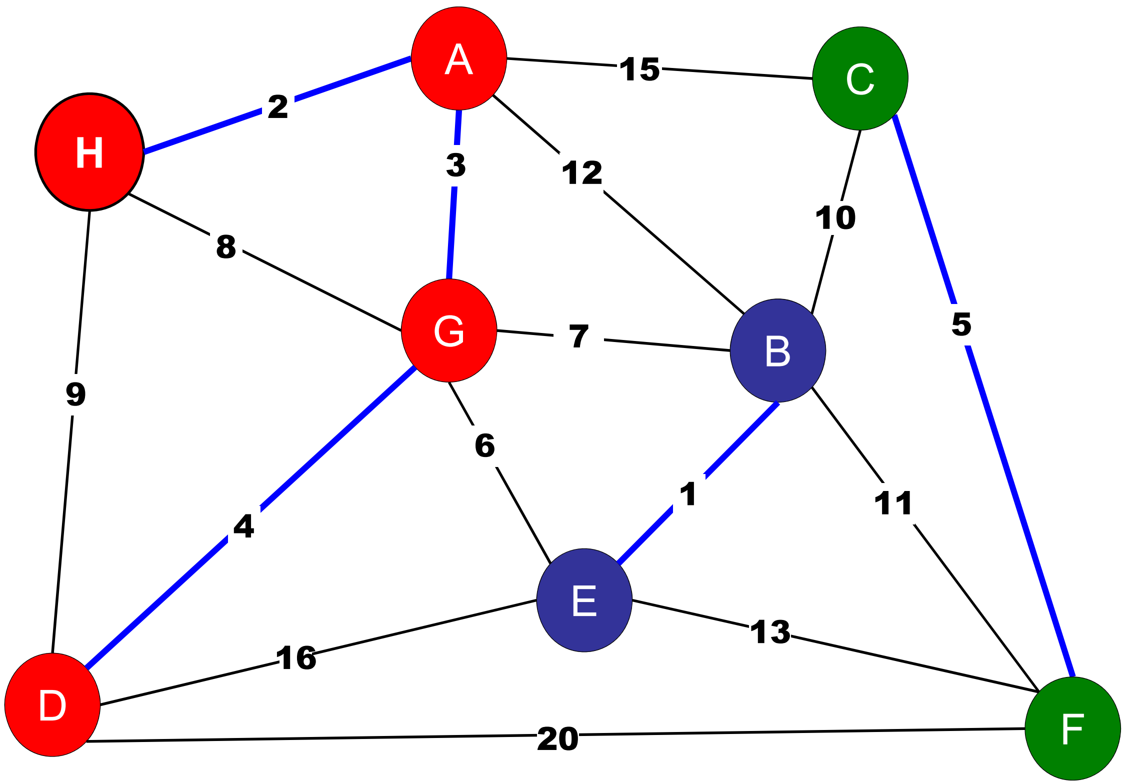


For every node: add minimum adjacent edge.

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

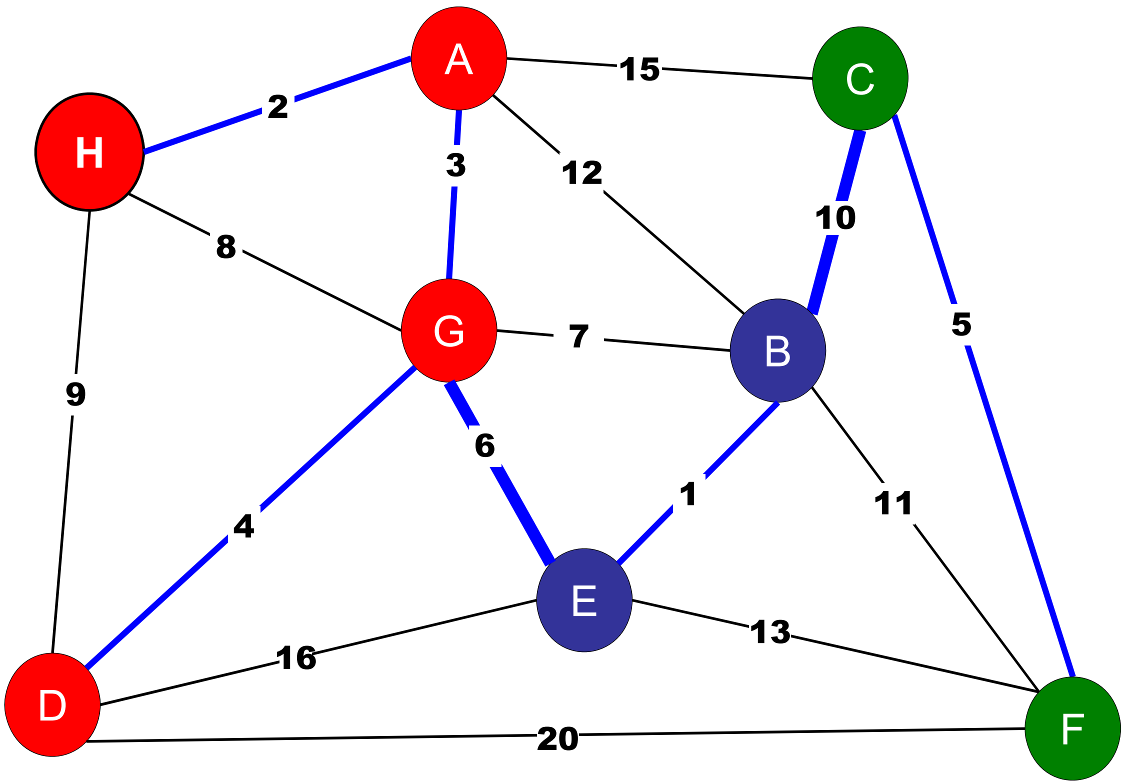


Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Look at connected components…

At most n/2 connected components.

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:

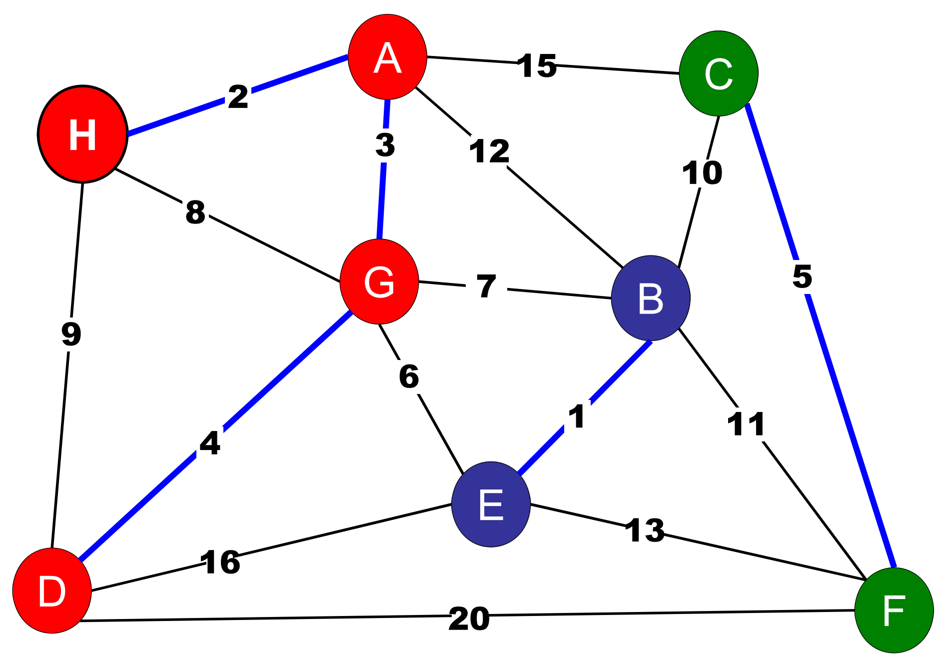
– Create n connected components, one for each node in the graph.

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:

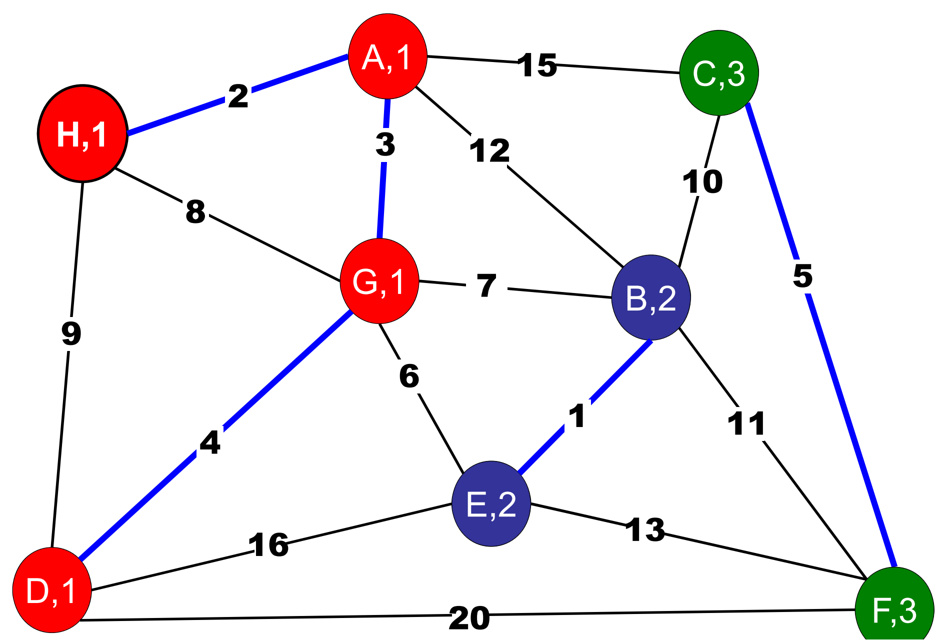
Check if edge connects two components.

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.

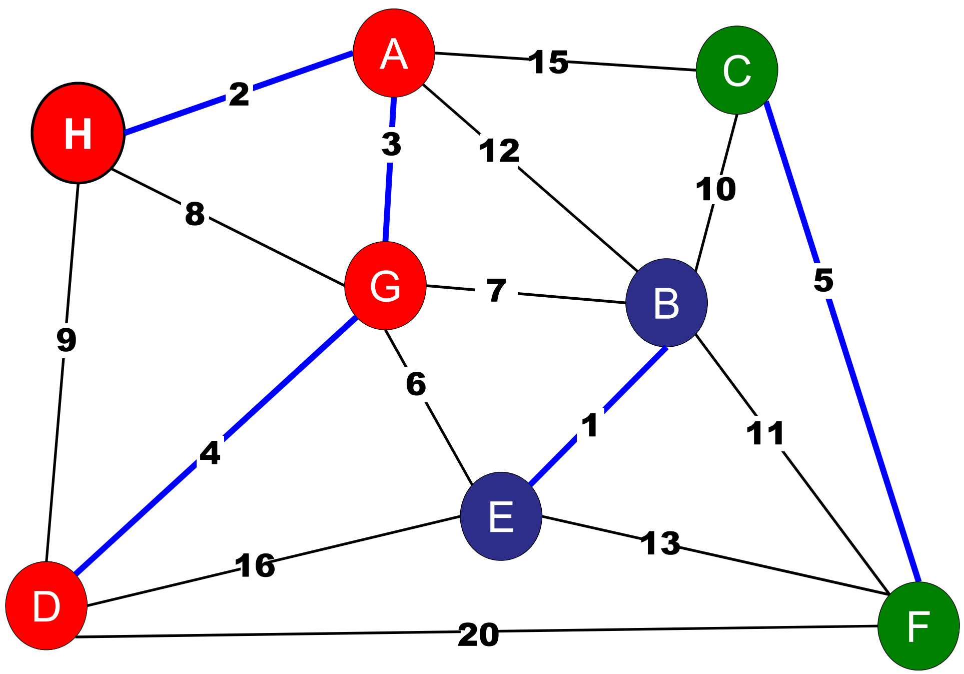
Remember minimum cost edge connected to each component.

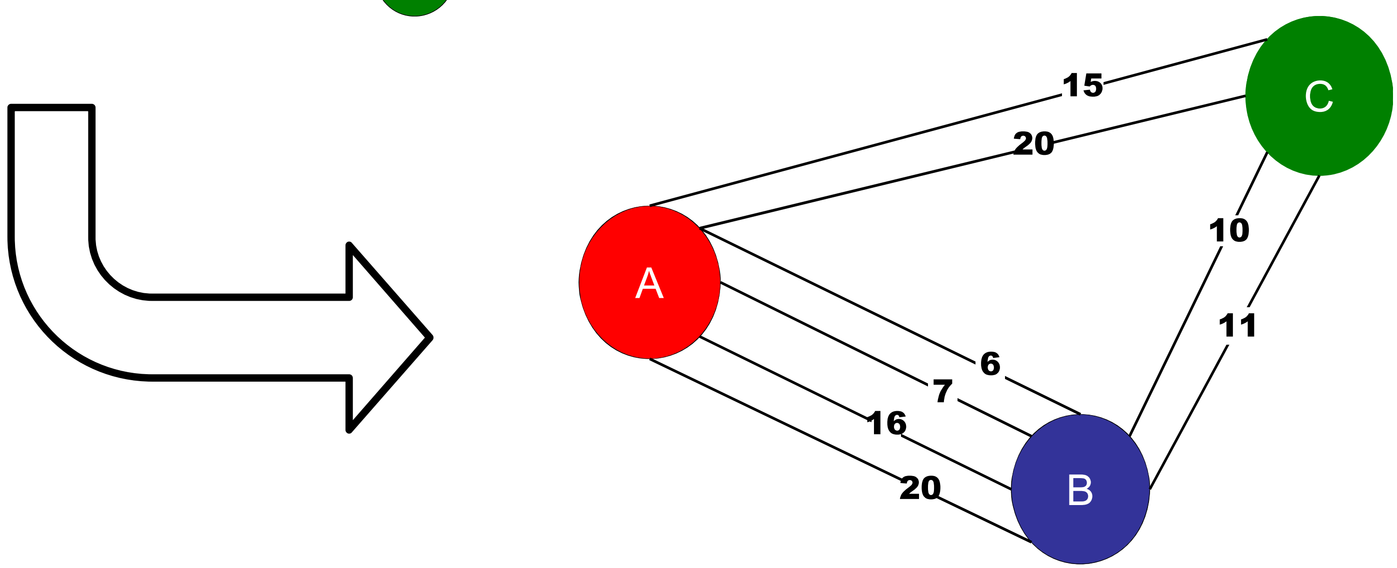
Scan every node: O(V)
Compute new component ids.

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

Why does it have a lot of parallelism?

Each connected component can perform a Boruvka step (mostly) independently:

Each component can search for its own minimum weight outgoing edge.
➔ Many edges found in parallel.

Merging requires coordination between components.
➔ Key bottleneck

Lots of parallelism in searching for edges and updating connected components.

Boruvka's Algorithm

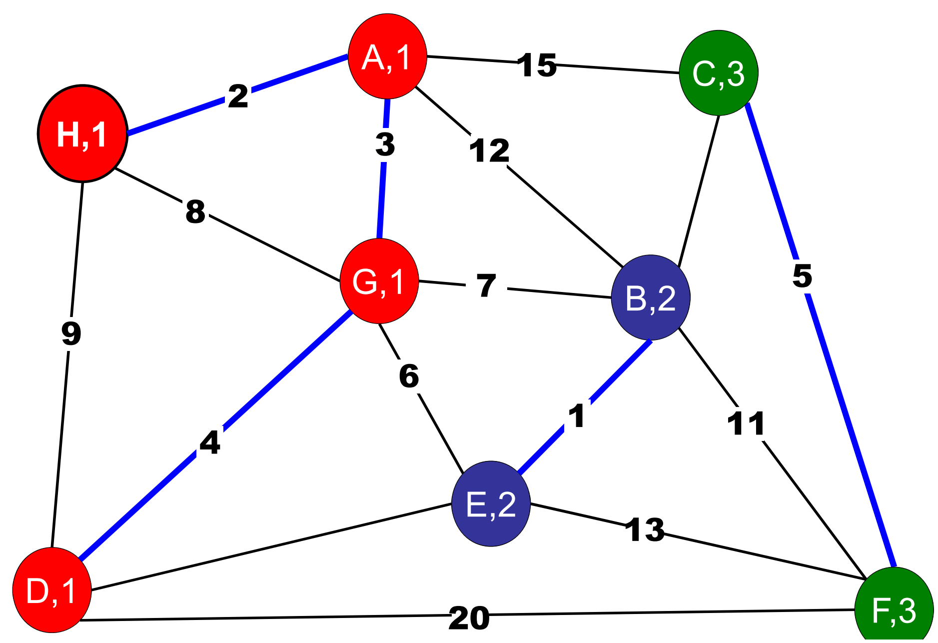
Boruvka's Algorithm

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)
- Uses a "soft heap" (!?)

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

- Chazelle (2000)

Holy grail and major open problem: O(m)

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

Roadmap

Last time: Minimum Spanning Trees

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

Today: Variations

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

MST Variants

What if all the edges have the same weight?

How fast can you find an MST?

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

ARCHIPELAGO

is open

MST Variants

What if all the edges have the same weight?

- Depth-First-Search or Breadth-First-Search

If all edge-weights are 2, what is the **cost** of a MST?

1. V-1
2. V
3. 2(V-1)
4. 2V
5. E-V
6. E

ARCHIPELAGO

is open

MST Variants

What if all the edges have the same weight?

- Depth-First-Search or Breadth-First-Search
- An MST contains exactly (V-1) edges.
- Every spanning tree contains (V-1) edges!
- Thus, any spanning tree you find with DFS/BFS is a minimum spanning tree.

Kruskal's Variants

What if all the edges have weights from {1..10}?

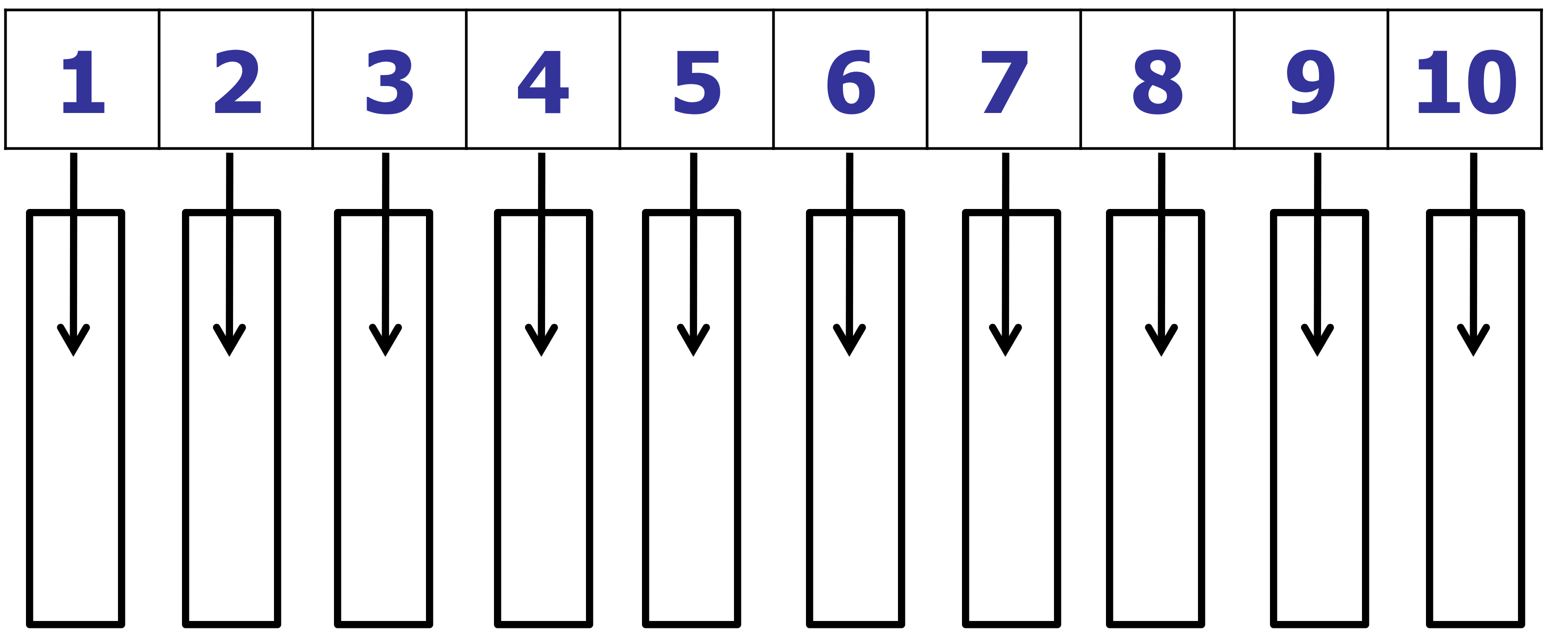
ARCHIPELAGO

is open

Kruskal’s Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10 to sort



slot A[j] holds a linked list of edges of weight j

Kruskal's Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10

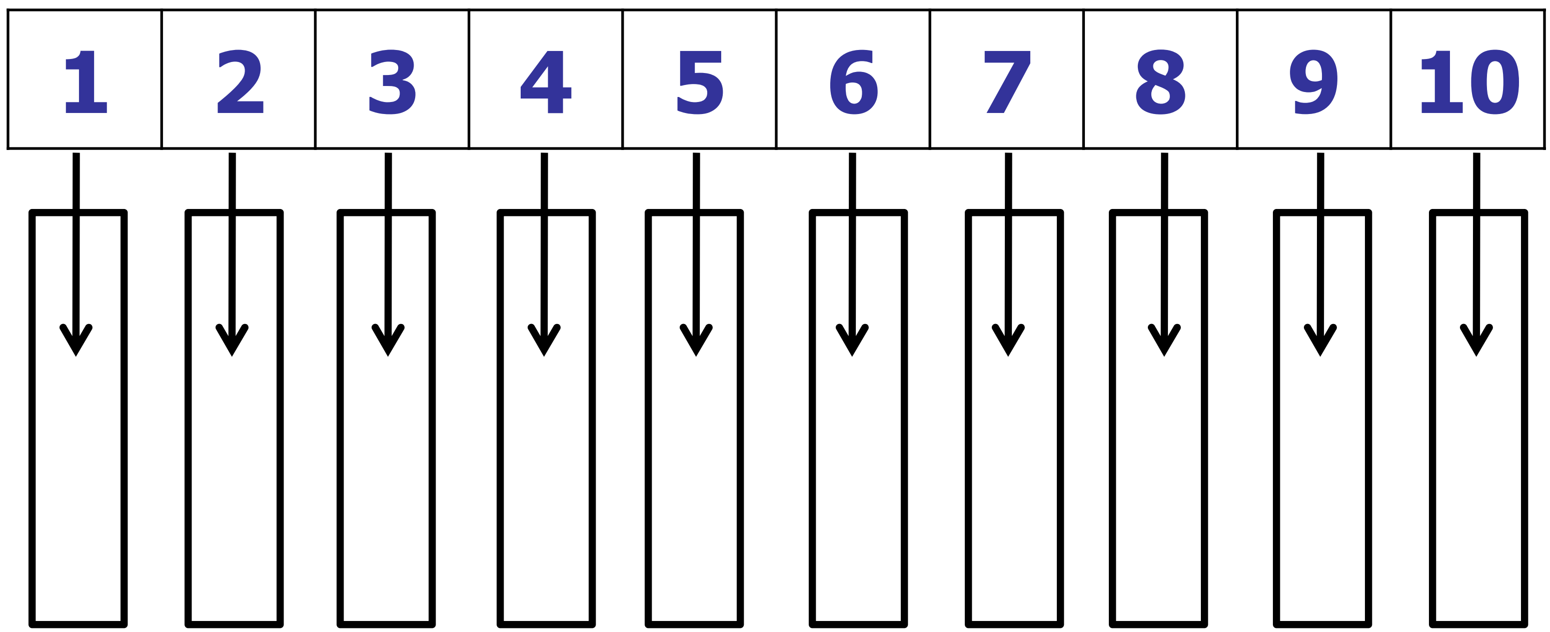
- Putting edges in array of linked lists: O(E)
- Iterating over all edges in ascending order: O(E)
- For each edge:
  - Checking whether to add an edge: O(α)
  - Union two components: O(α)

Total: O(αE)

Prim's Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10 as a Priority Queue

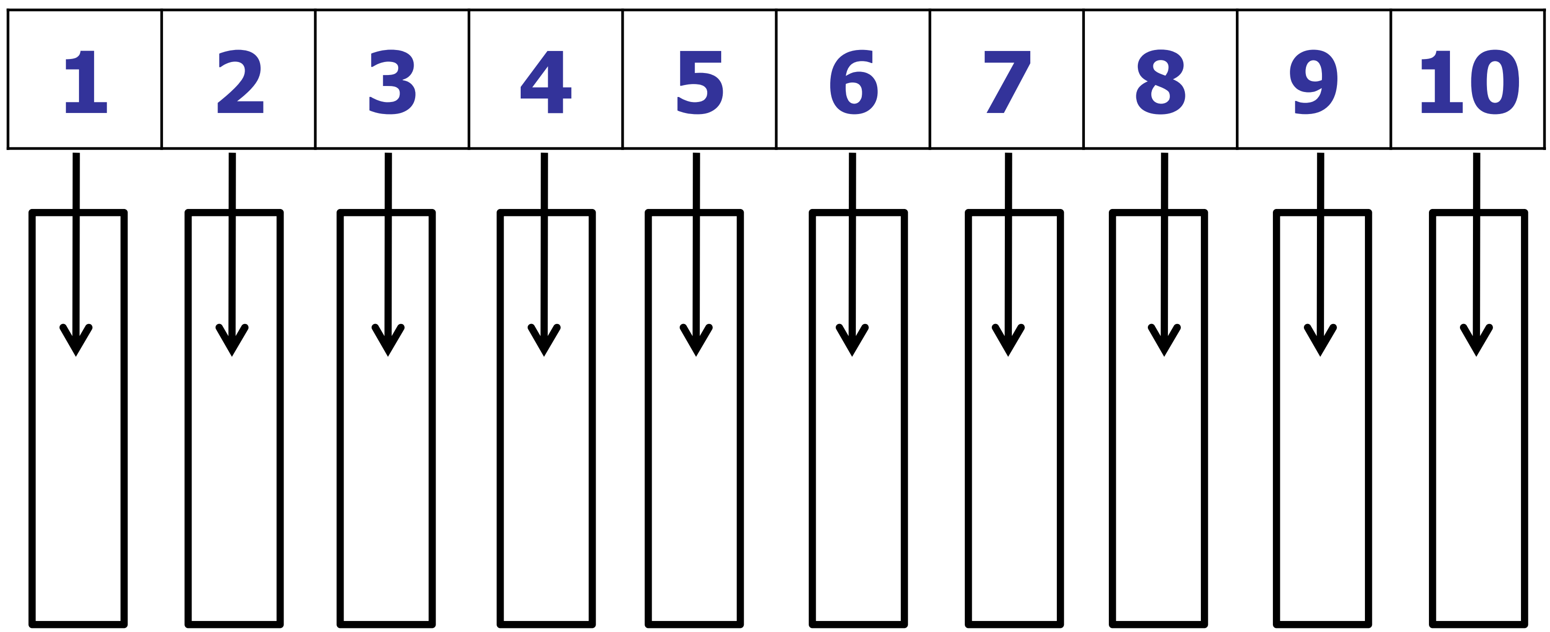


slot A[j] holds a linked list of **nodes** of weight j

Prim's Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10 as a Priority Queue



decreaseKey: move node to new linked list

What is the running time of (modified) Prim's if all the edge weights are in {1..10}?

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

ARCHIPELAGO

is open

Prim's Variants

What if all the edges have weights from {1..10}?

Implement Priority Queue:

- Use an array of size 10 to implement
- Insert: put node in correct list
- Remove: lookup node (e.g., in hash table) and remove from liked list.
- ExtractMin: Remove from the minimum bucket.
- DecreaseKey: lookup node (e.g., in hash table) and move to correct liked list.

Prim's Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10

- Inserting/Removing nodes from PQ: O(V)
- decreaseKey: O(E)

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

Prim’s Variants

What if all the edges have weights from {1..10}?

Implement Priority Queue….

Why does this fail for Dijkstra’s Algorithm?

Roadmap

Today: Minimum Spanning Trees

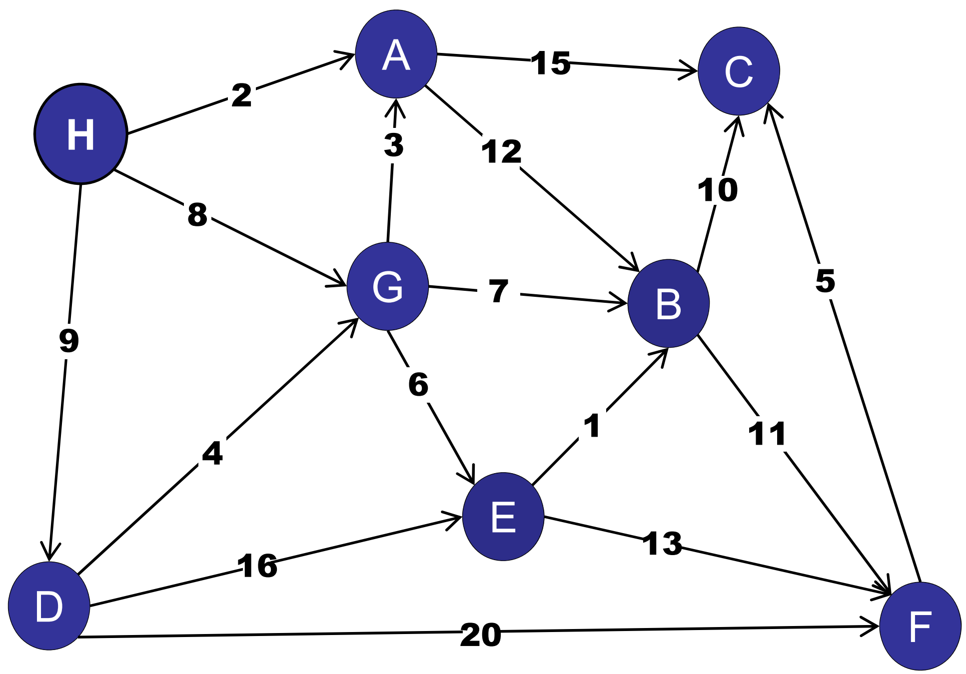
- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm

Variations:

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

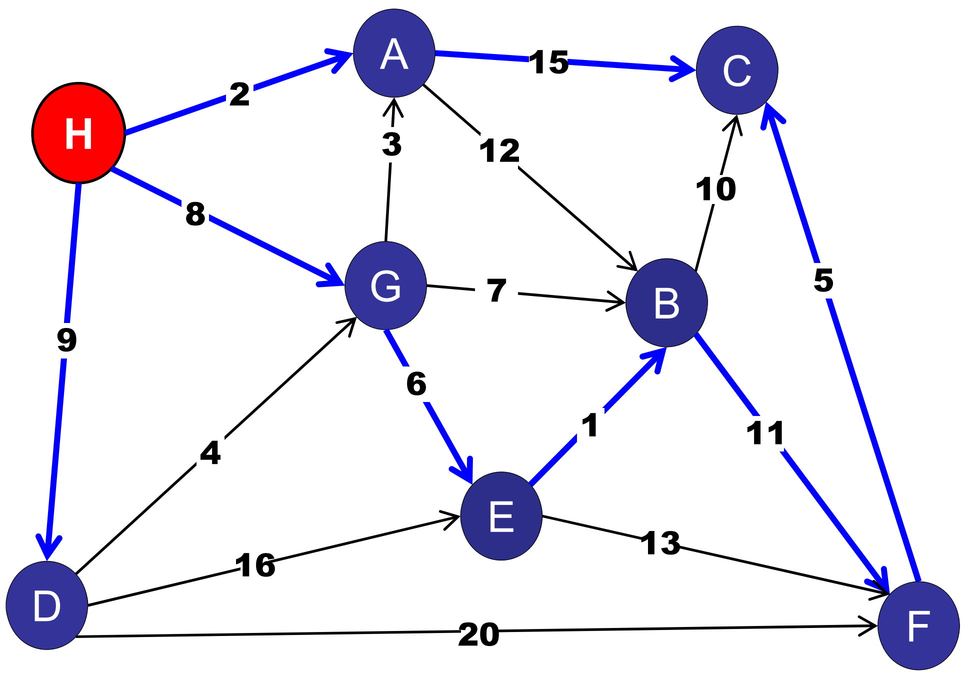
Directed Minimum Spanning Tree

What if the edges are directed?



Directed Minimum Spanning Tree

A rooted spanning tree:



Every node is reachable on a path from the root.

No cycles.

Directed Minimum Spanning Tree

Harder problem:

- Cut property does not hold.
- Cycle property does not hold.
- Generic MST algorithm does not work.

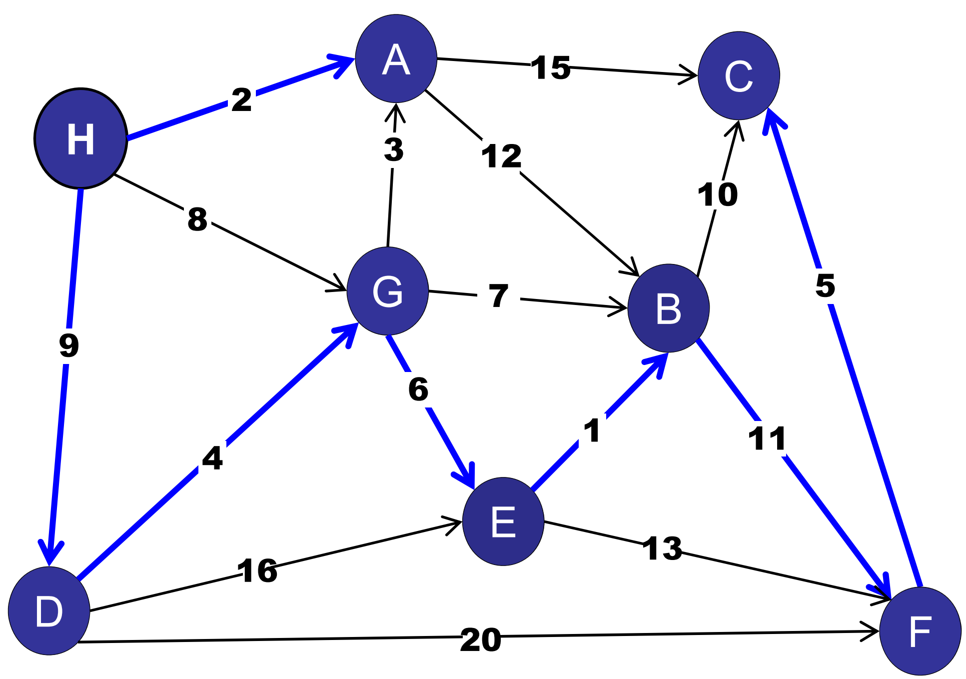
Prim's, Kruskal's, Boruvka's do not work.

See CS3230 / CS4234 for more details…

Exercise: Draw a directed graph where the cut property or cycle property is violated.

Directed Minimum Spanning Tree

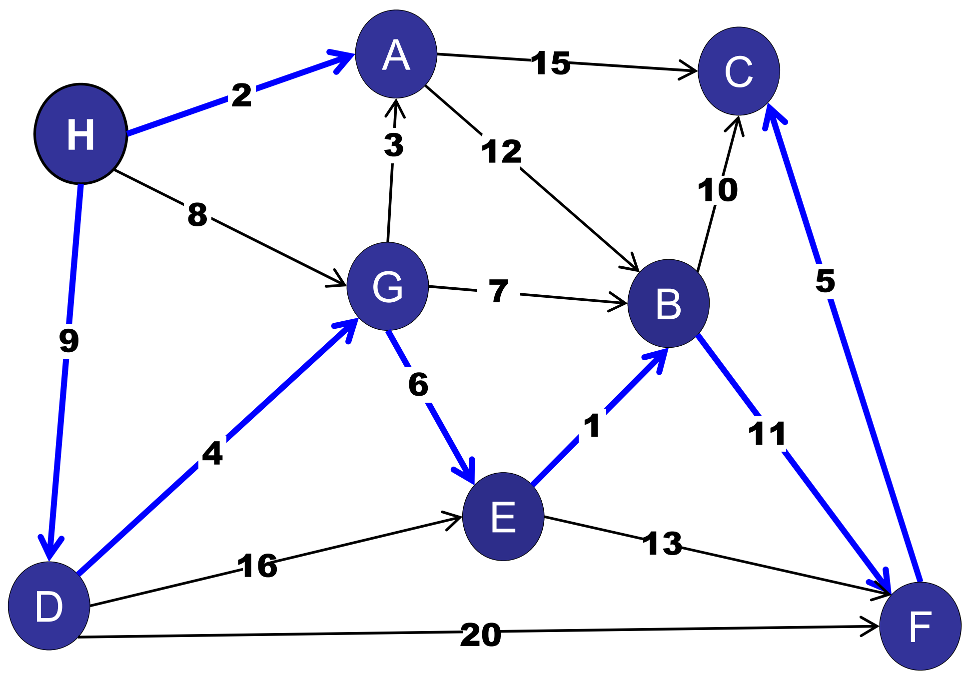
Special case: a directed acyclic graph with one “root”:



Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.



Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.

Observations:

- No cycles (since acyclic graph).
- Each edge is chosen only once.

Tree:
V nodes
V – 1 edges
No cycles

Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.

Observations:

- No cycles (since acyclic graph).
- Each edge is chosen only once.

Tree:
V nodes
V – 1 edges
No cycles

- Every node has to have at least one incoming edge in the MST, so this is the minimum spanning tree.

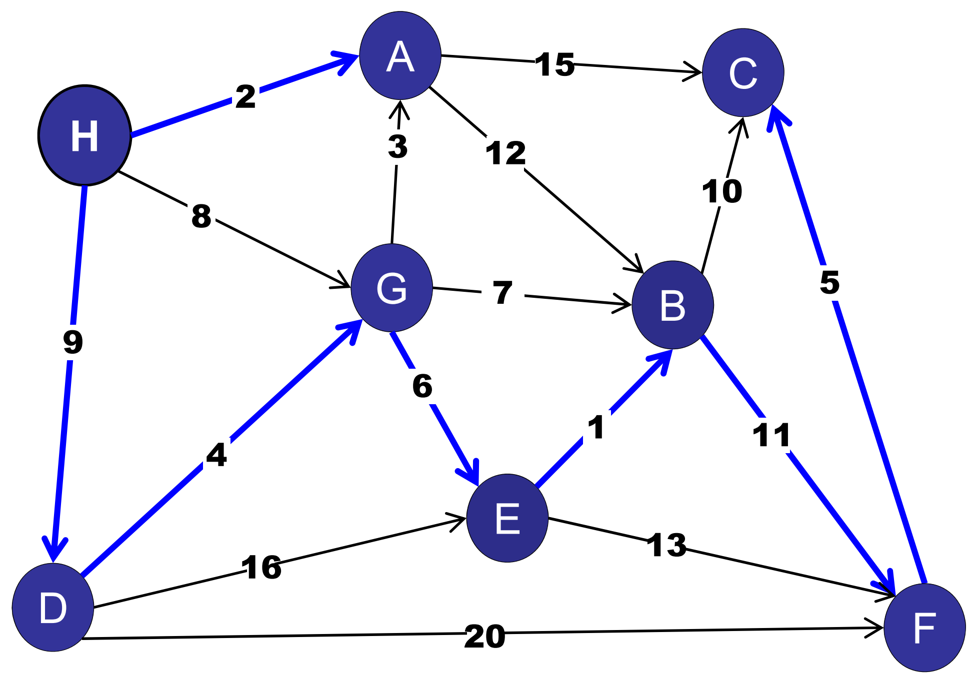
Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.

Conclusion: Minimum Spanning Tree

O(E) time



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

Maximum Spanning Tree

A MaxST is a spanning tree of maximum weight.

How do you find a MaxST?

Maximum Spanning Tree

Reweighting a spanning tree:

- What happens if you add a constant k to the weight of every edge?

ARCHIPELAGO

is open

Kruskal's Algorithm

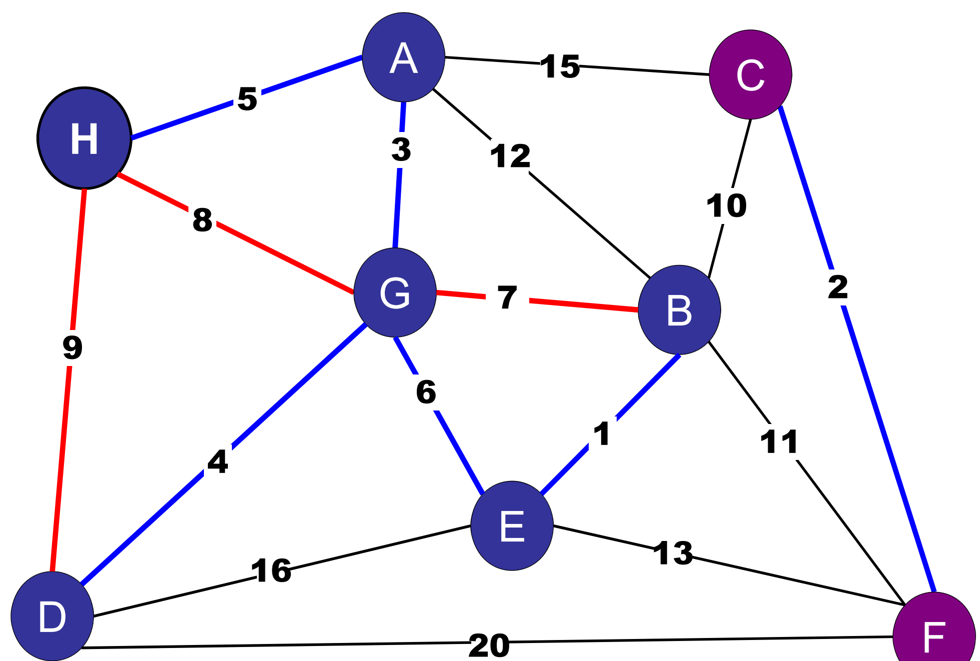
Kruskal's Algorithm. (Kruskal 1956)

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.

What matters?

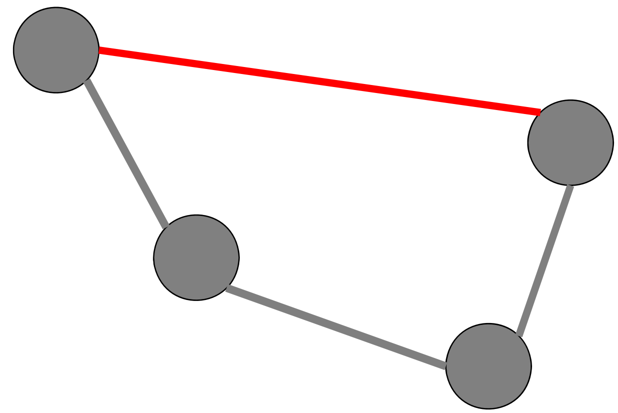
- Relative edge weights.
- Absolute edge weights have no impact.



Generic MST Algorithm

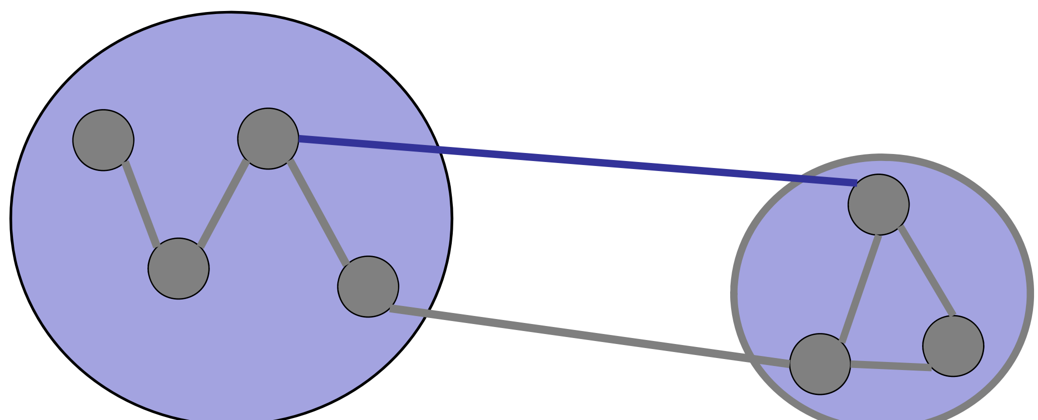
Red rule:

If C is a cycle with no red arcs, then color the max-weight edge in C red.



Blue rule:

If D is a cut with no blue arcs, then color the min-weight edge in D blue.



Maximum Spanning Tree

Reweighting a spanning tree:

- What happens if you add a constant k to the weight of every edge?

No change!

We can add or subtract weights without effecting the MST.

(Very *different* from shortest paths…)

Maximum Spanning Tree

MST with negative weights?

Maximum Spanning Tree

MST with negative weights?

No problem!

1. Reweight MST by adding a big enough value to each edge so that it is positive.

2. Actually, no need to reweight. Only relative edge weights matter, so negative weights have no bad impact.

Maximum Spanning Tree

A MaxST is a spanning tree of maximum weight.

How do you find a MaxST?

Easy!

1. Multiply each edge weight by -1.
2. Run MST algorithm.
3. MST that is "most negative" is the maximum.

Maximum Spanning Tree

A MaxST is a spanning tree of maximum weight.

How do you find a MaxST?

Or… run Kruskal's in reverse.

Roadmap

Last time: Minimum Spanning Trees

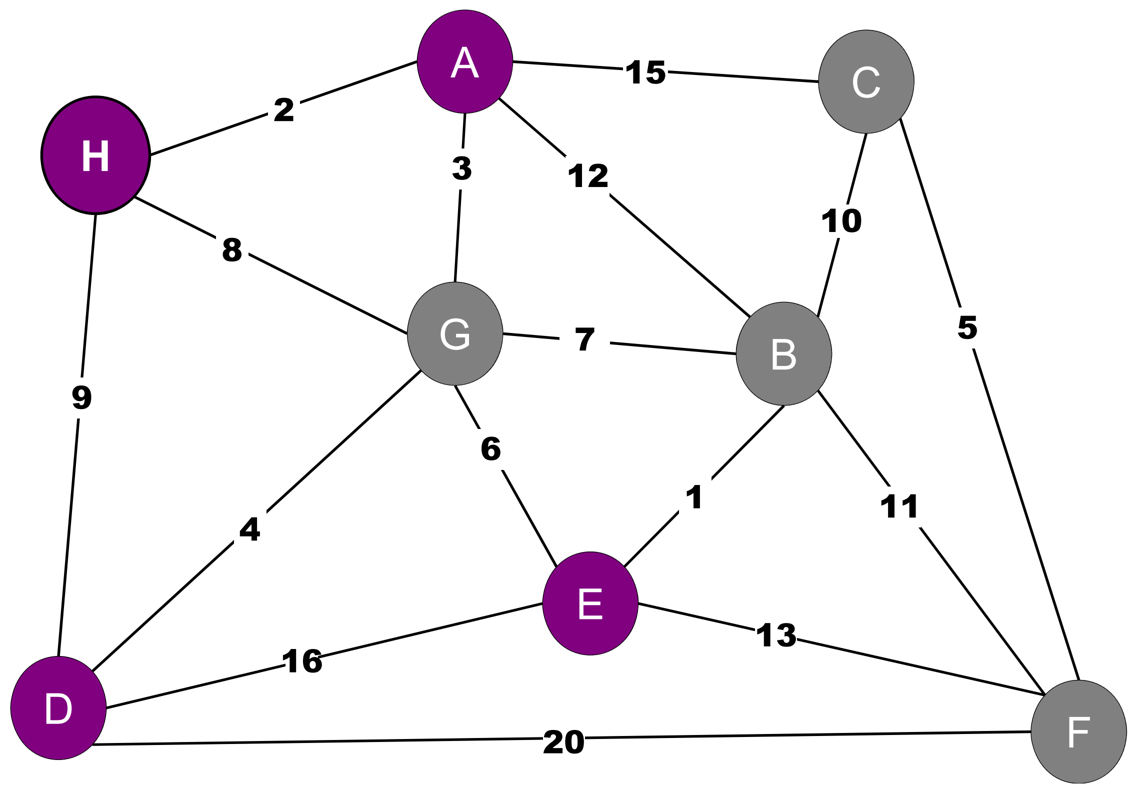
- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm

Today: Variations

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

Steiner Tree Problem

What if I want a minimum spanning tree of a subset of the vertices?

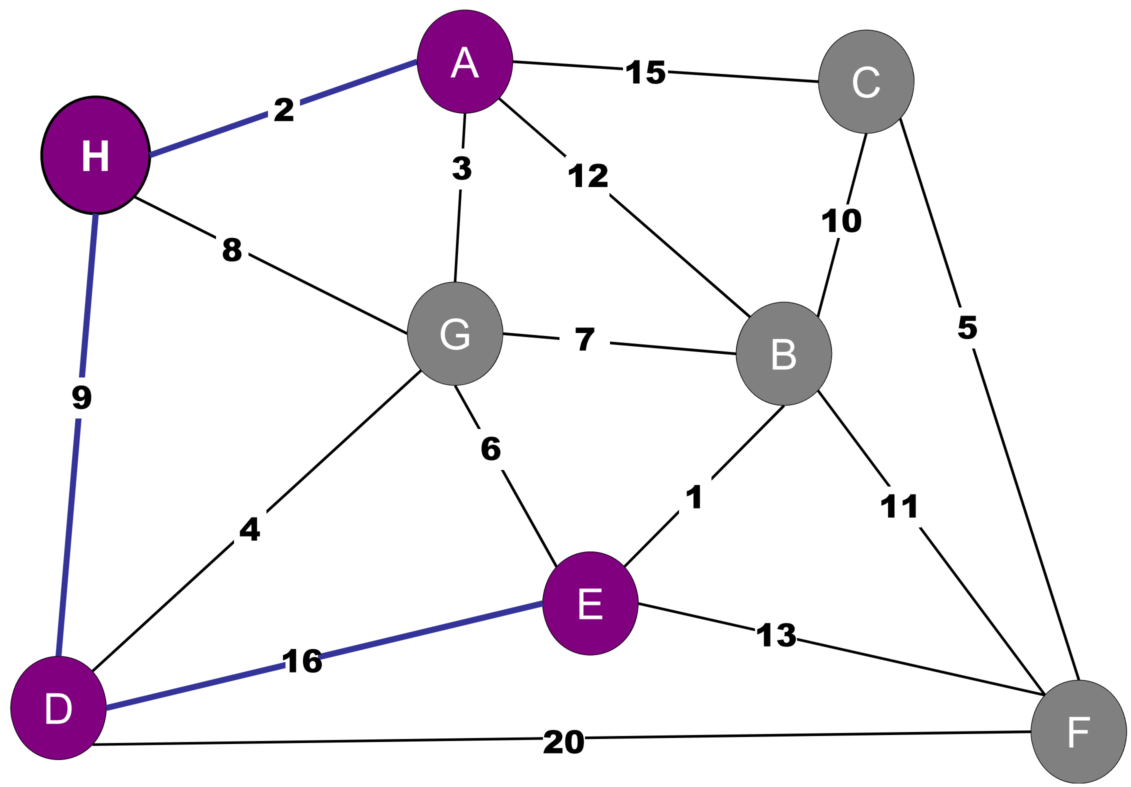


Steiner Tree Problem

What if I want a minimum spanning tree of a subset of the vertices?

1. Just use the sub-graph.

weight = 27

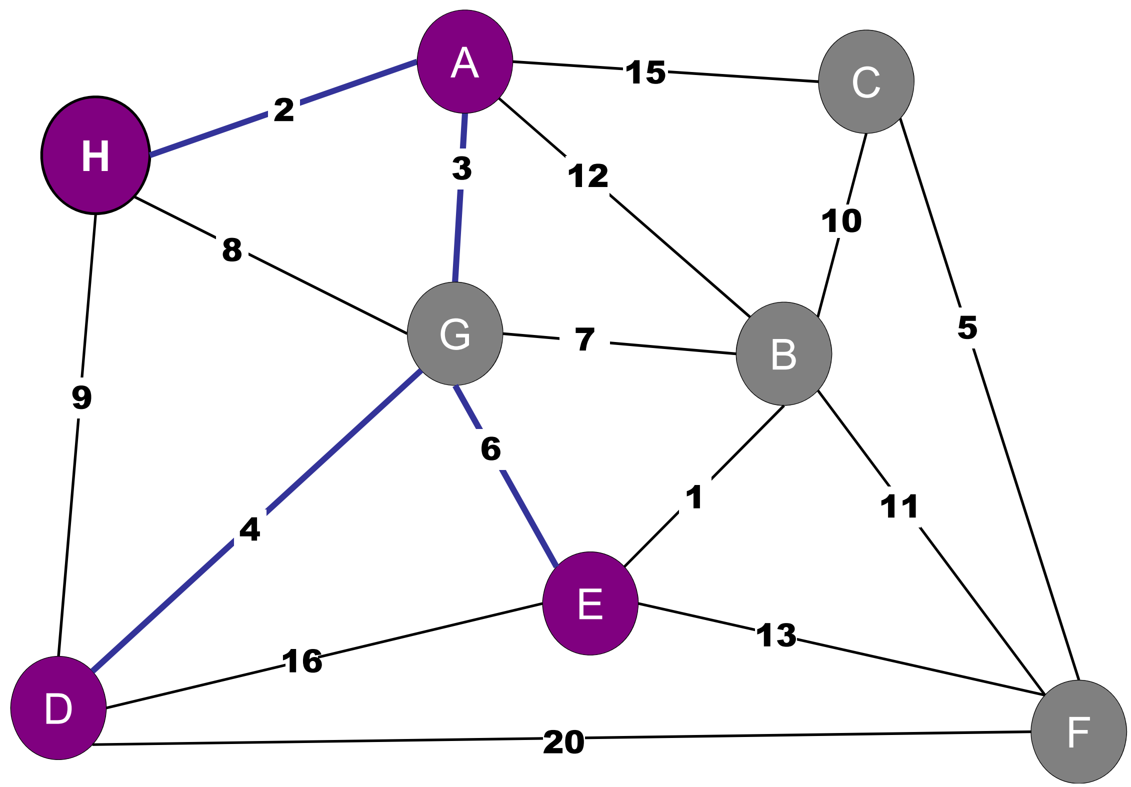


Steiner Tree Problem

What if I want a minimum spanning tree of a subset of the vertices?

1. Just use the sub-graph.
2. Use other nodes.

weight = 15

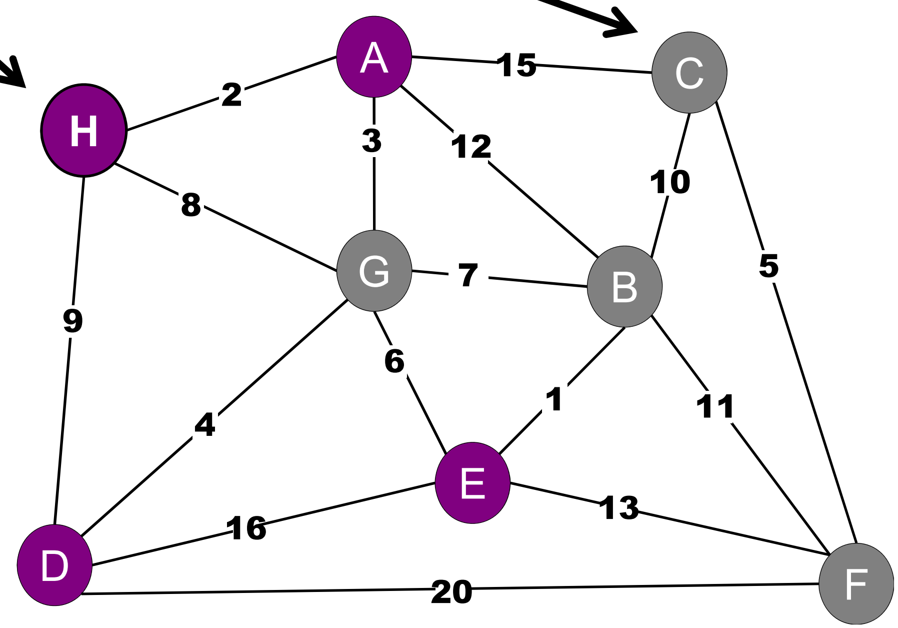


Steiner Tree Problem

What is the minimum spanning tree of a subset of the vertices?

- Steiner nodes
- Required nodes

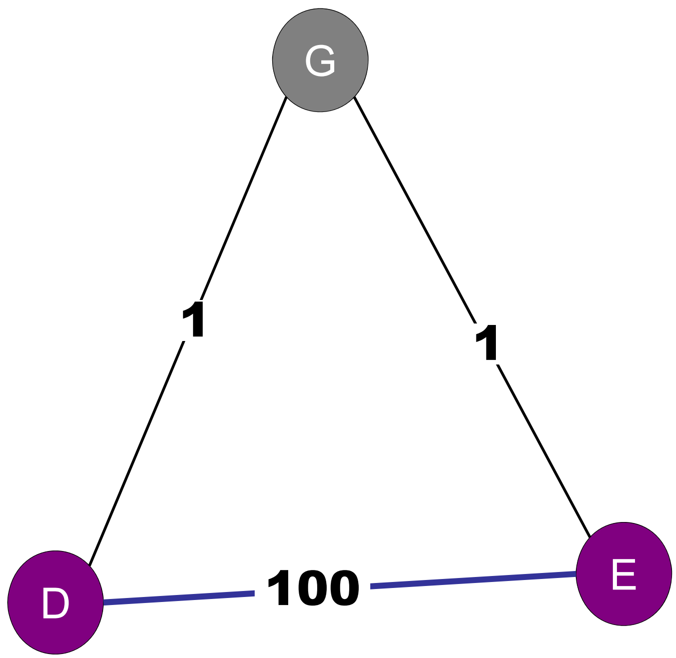
Find spanning tree of required vertices. You may include Steiner vertices, optionally.



Steiner Tree Problem

Just calculate MST doesn't work:

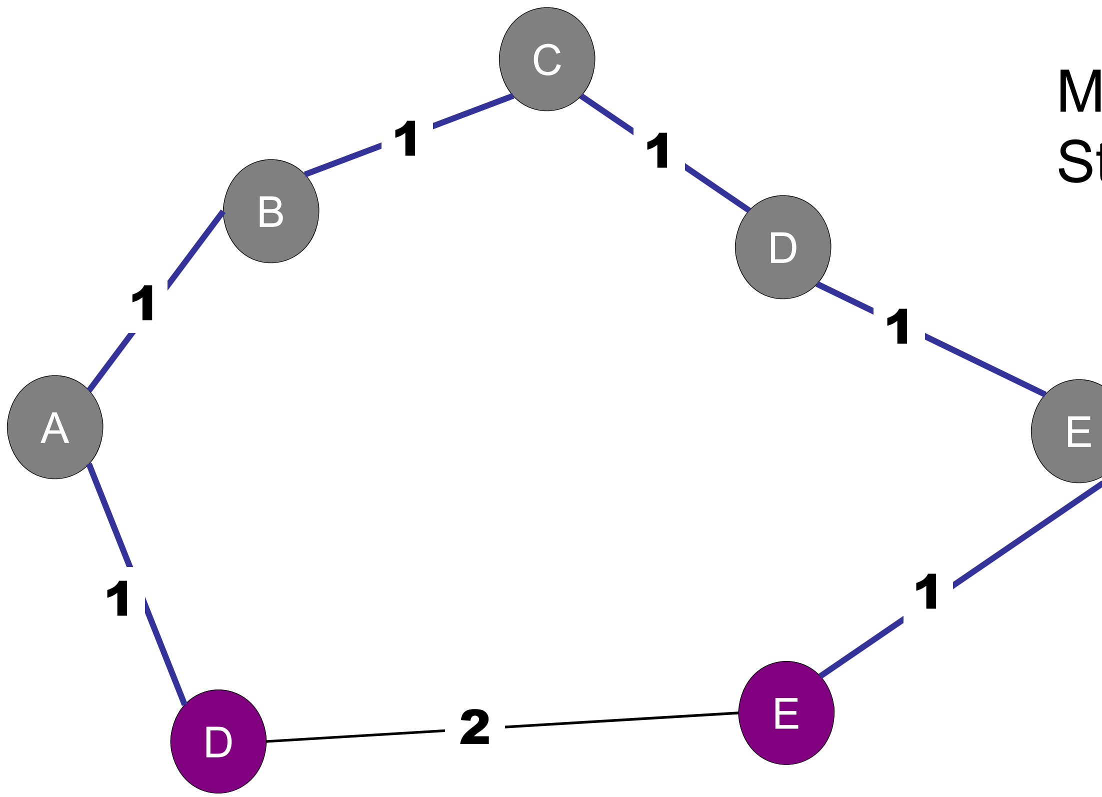
1. Calculate MST with no Steiner nodes.



Steiner Tree Problem

Just calculate MST doesn't work:

2. Calculate MST with all Steiner nodes.



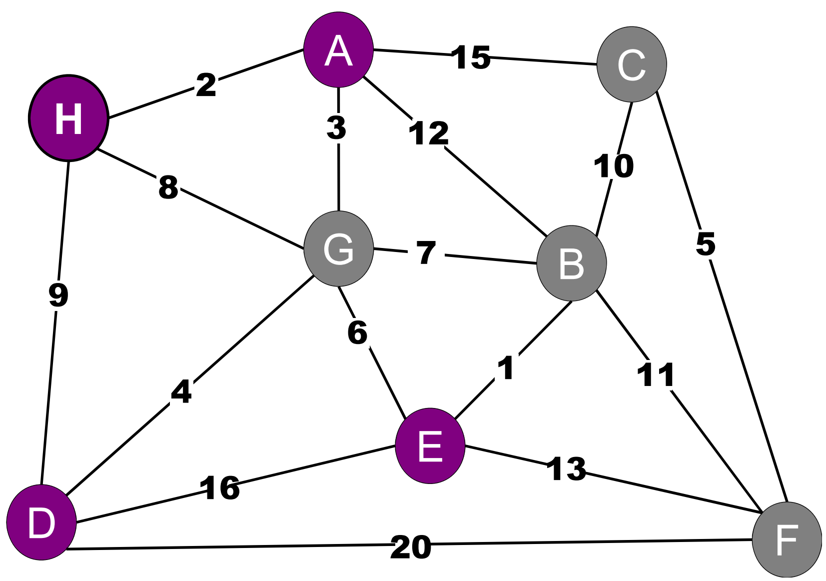
MST = 6
Steiner Tree = 2

Steiner Tree Problem

What is the minimum spanning tree of a subset of the vertices?

Bad News: NP-Hard

No efficient (polynomial) time algorithm (unless P = NP).



Steiner Tree Problem

What is the minimum spanning tree of a subset of the vertices?

Good News: Efficient approximation algorithms

Algorithm SteinerMST guarantees:

- OPT(G) = minimum cost Steiner Tree
- T = output of SteinerMST
- T < 2*OPT(G)

Steiner Tree Problem

Algorithm SteinerMST guarantees:

- OPT(G) = minimum cost Steiner Tree
- T = output of SteinerMST
- T < 2*OPT(G)

Example:

- Optimal Steiner Tree has cost 50.
- Our algorithm always outputs a solution with cost < 100.

Steiner Tree Problem

Algorithm SteinerMST:

1. For every pair of required vertices (v,w), calculate the shortest path from (v to w).
   - Use Dijkstra V times.
   - Or wait until we cover All-Pairs-Shortest-Paths next time.

Steiner Tree Problem

Example: Step 1

Shortest Paths:

(A,H) = 2

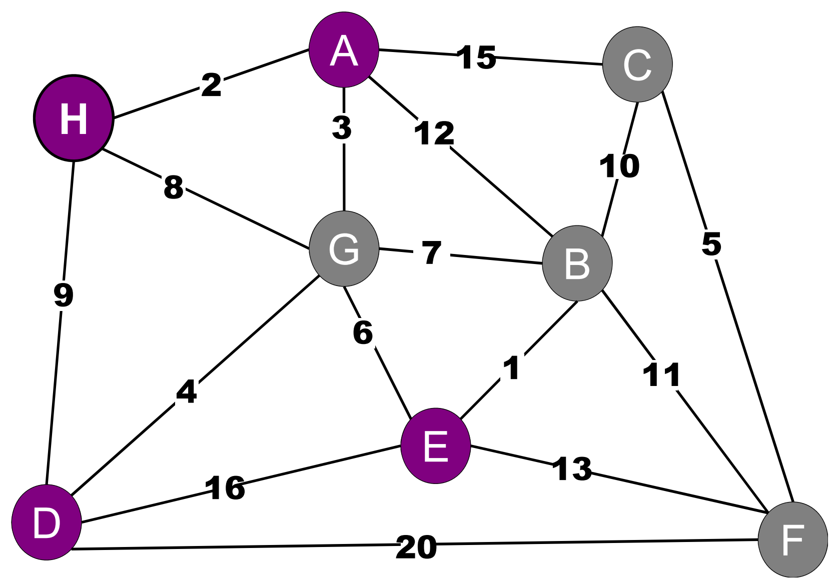
(A,D) = 7

(A,E) = 9

(H,D) = 9

(H,E) = 11

(D,E) = 10



Steiner Tree Problem

Algorithm SteinerMST:

1. For every required vertex (v,w), calculate the shortest path from (v to w).
2. Construct new graph on required nodes.
   - V = required nodes
   - E = shortest path distances

Steiner Tree Problem

Example: Step 2

Shortest Paths:

(A,H) = 2

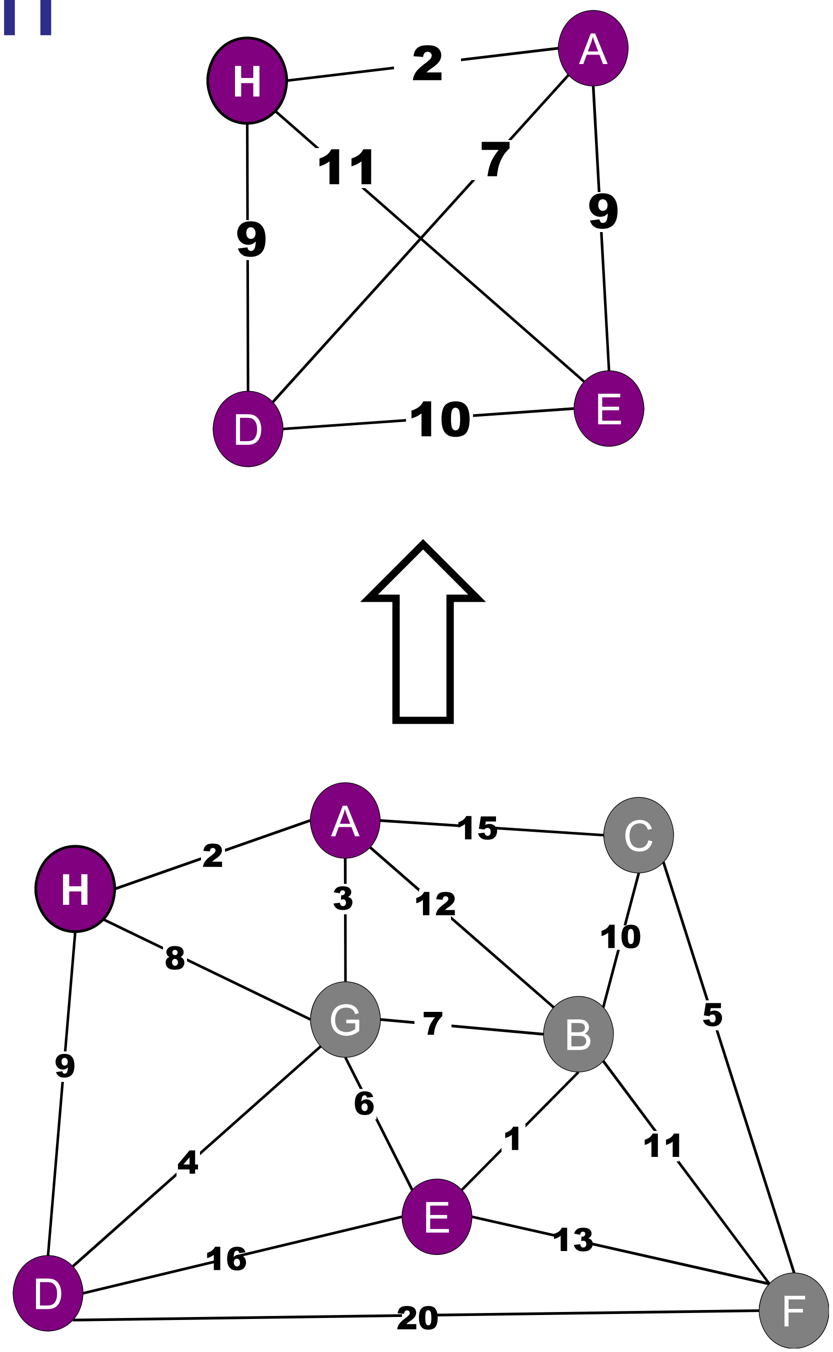
(A,D) = 7

(A,E) = 9

(H,D) = 9

(H,E) = 11

(D,E) = 10



Steiner Tree Problem

Algorithm SteinerMST:

1. For every required vertex (v,w), calculate the shortest path from (v to w).
2. Construct new graph on required nodes.
3. Run MST on new graph.
   - Use Prim’s or Kruskal’s or Boruvka’s
   - MST gives edges on new graph

Steiner Tree Problem

Example: Step 3

Shortest Paths:

(A,H) = 2

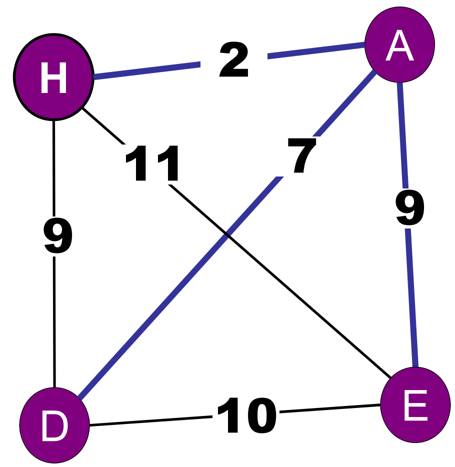
(A,D) = 7

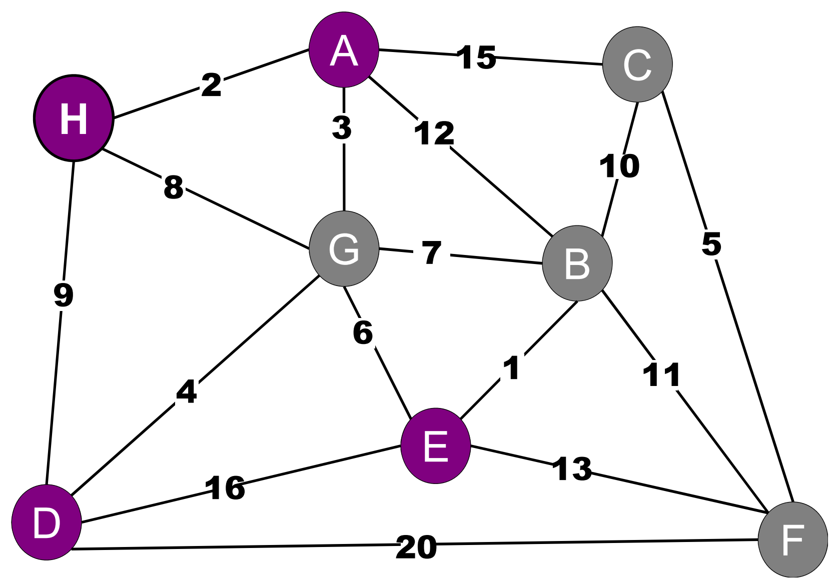
(A,E) = 9

(H,D) = 9

(H,E) = 11

(D,E) = 10





Steiner Tree Problem

Algorithm SteinerMST:

1. For every required vertex (v,w), calculate the shortest path from (v to w).
2. Construct new graph on required nodes.
3. Run MST on new graph.
4. Map new edges back to original graph.
   - Use shortest path discovered in Step 1.
   - Add these edges to Steiner MST.
   - Remove duplicates.

Steiner Tree Problem

Example: Step 4

Shortest Paths:

(A,H) = 2

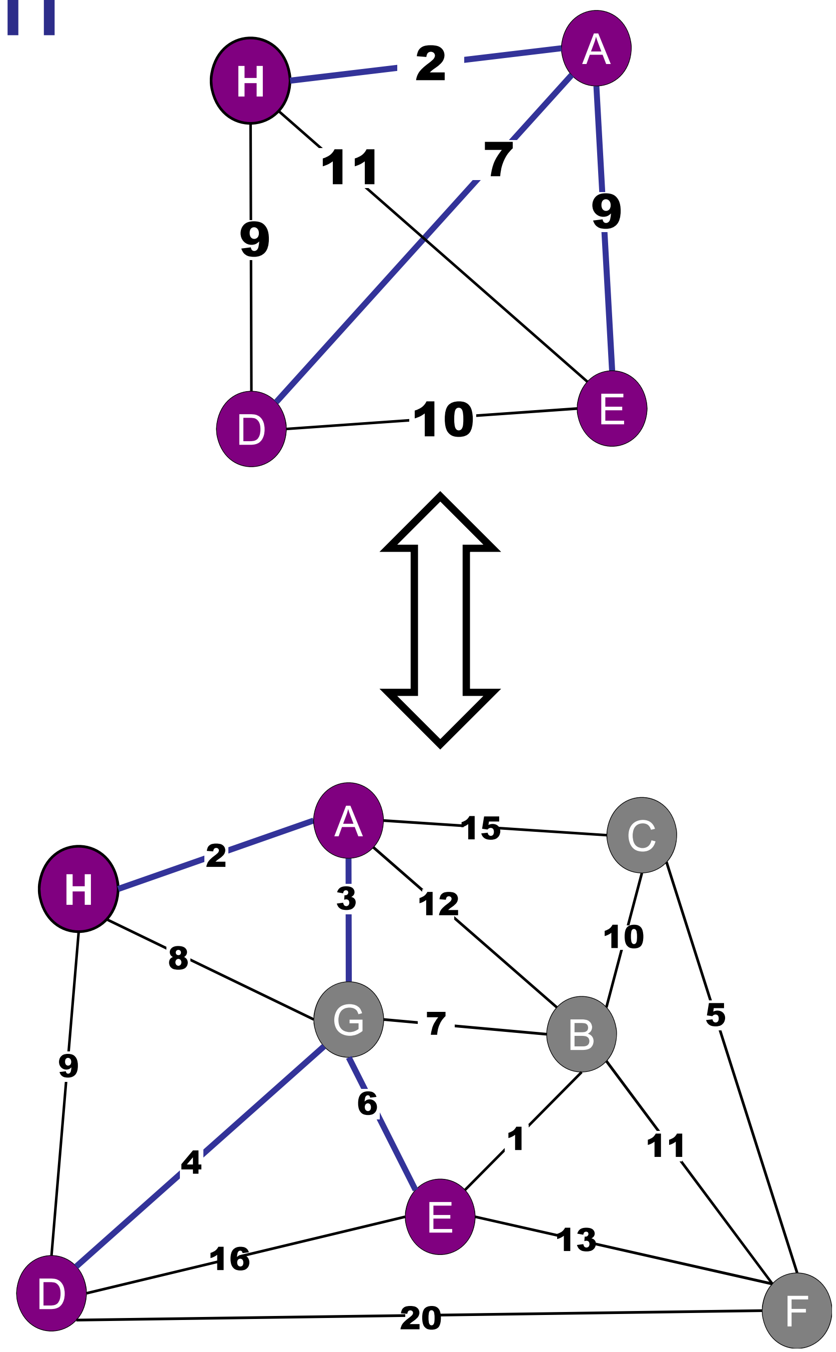
(A,D) = 7

(A,E) = 9

(H,D) = 9

(H,E) = 11

(D,E) = 10



Steiner Tree Problem

Algorithm SteinerMST:

1. For every required vertex (v,w), calculate the shortest path from (v to w).
2. Construct new graph on required nodes.
3. Run MST on new graph.
4. Map new edges back to original graph.

Note: Does NOT guarantee optimal
Steiner tree.

Steiner Tree Problem

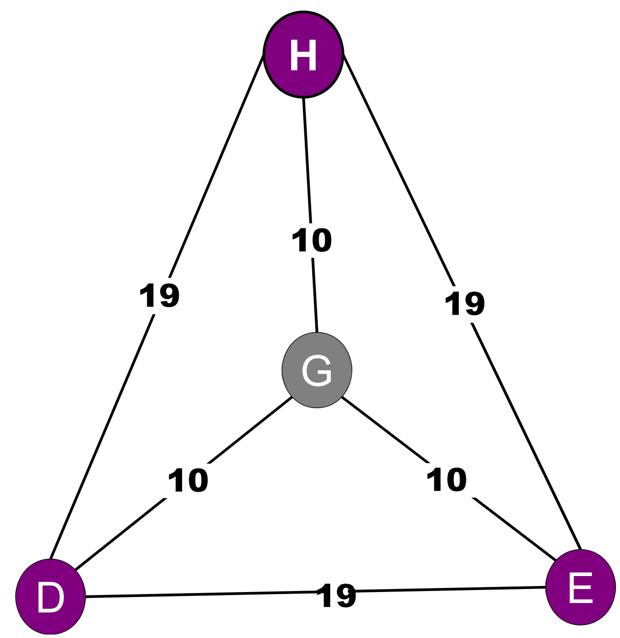
Example:

Shortest Paths:

(D,H) = 19

(D,E) = 19

(E,H) = 19



Steiner Tree Problem

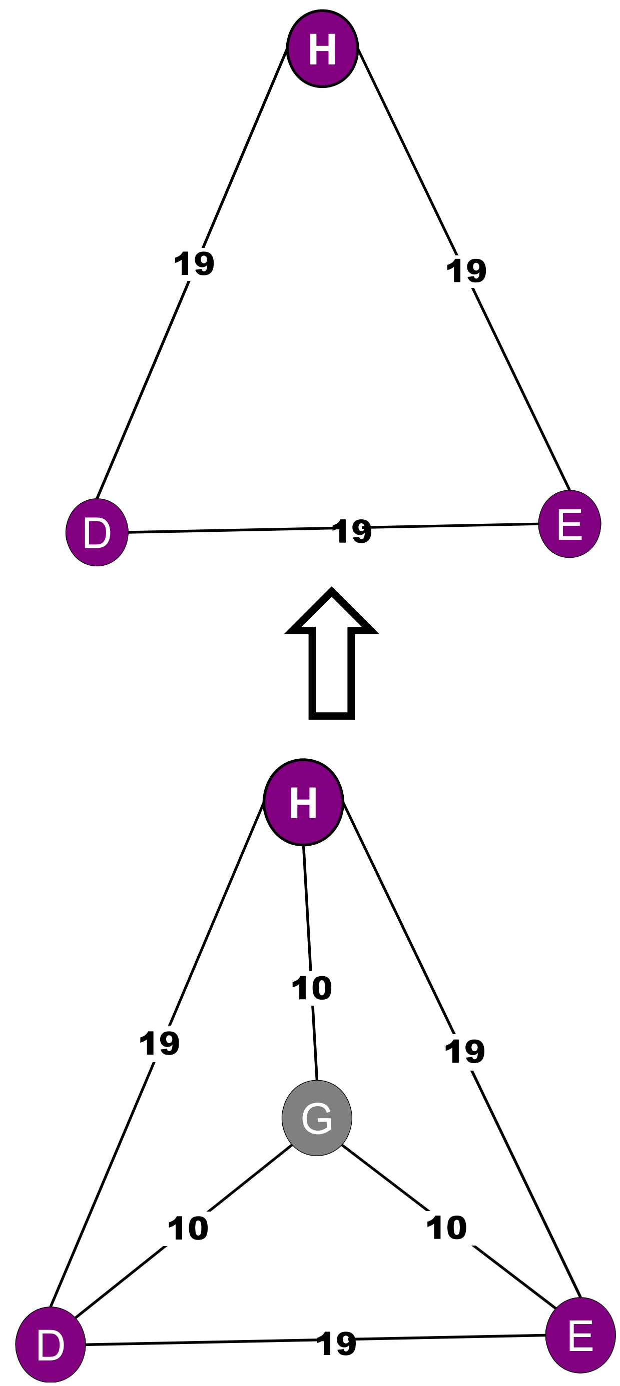
Example:

Shortest Paths:

(D,H) = 19

(D,E) = 19

(E,H) = 19



Steiner Tree Problem

Example:

Shortest Paths:

(D,H) = 19

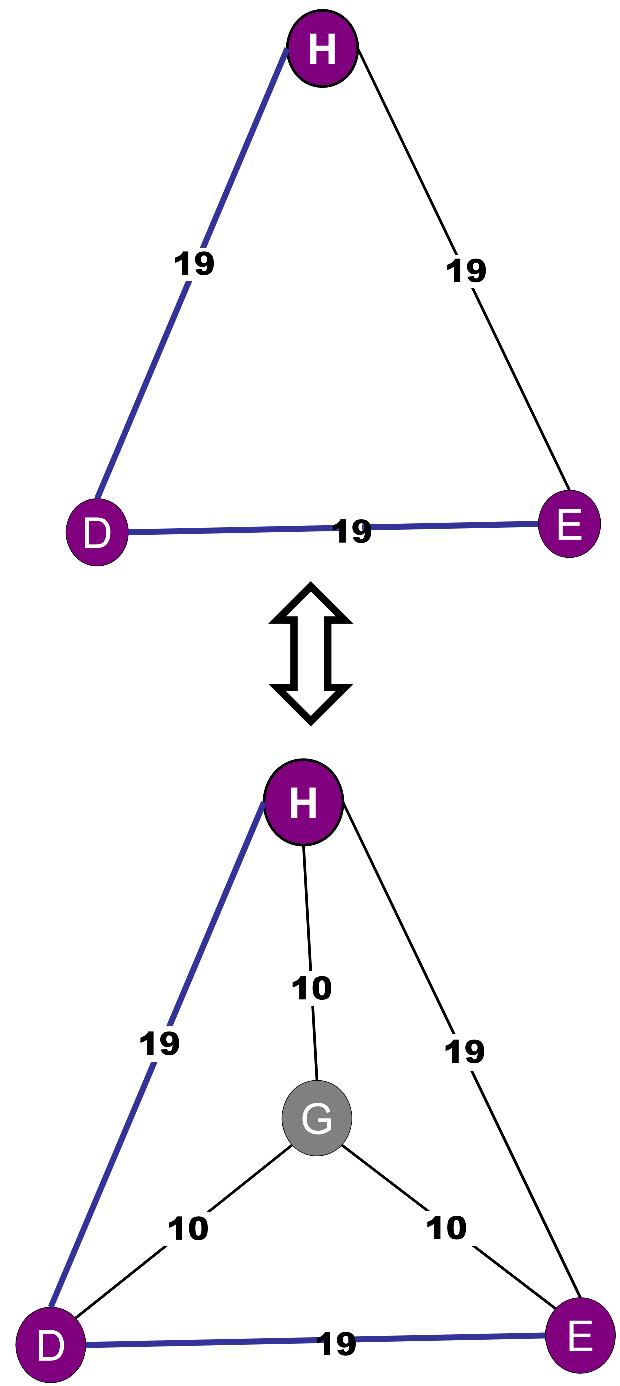
(D,E) = 19

(E,H) = 19

Cost = 38:

OPT Steiner = 30

Challenge: bigger gap!



Steiner Tree Problem

Algorithm SteinerMST:

1. For every required vertex (v,w), calculate the shortest path from (v to w).
2. Construct new graph on required nodes.
3. Run MST on new graph.
4. Map new edges back to original graph.

Note: Does NOT guarantee optimal Steiner tree.

Steiner Tree Problem

Algorithm SteinerMST:

1. Let O be OPT (Steiner) tree.

   Let T be SteinerMST tree.

Steiner Tree Problem

Algorithm SteinerMST:

1. Let O be OPT (Steiner) tree.

   Let T be SteinerMST tree.

2. Let D = DFS on O.

   cost(D) = 2*OPT.

Traverse each edge exactly twice!

Steiner Tree Problem

Example: Step 3

Shortest Paths:

(A,H) = 2

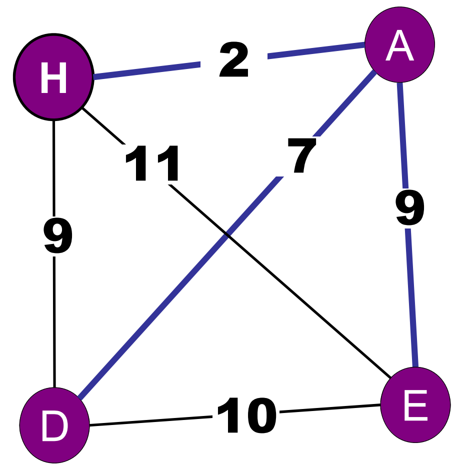
(A,D) = 7

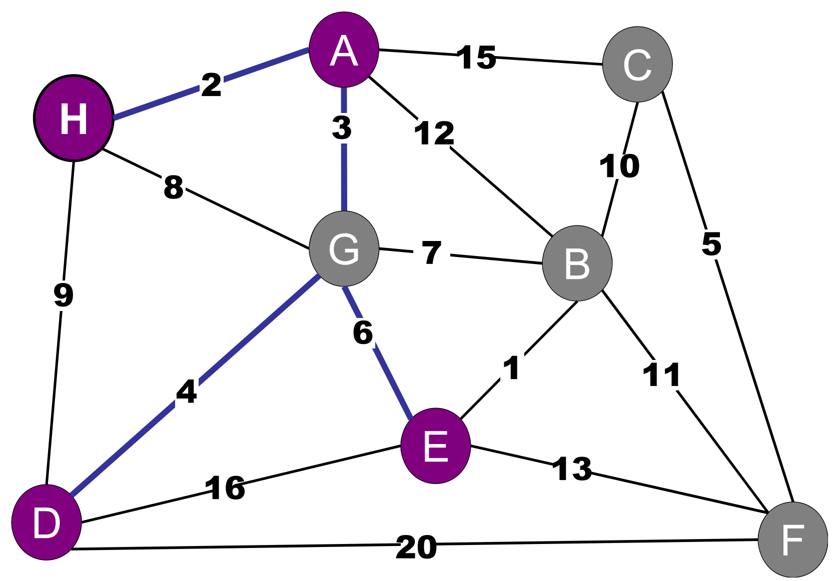
(A,E) = 9

(H,D) = 9

(H,E) = 11

(D,E) = 10





Steiner Tree Problem

Algorithm SteinerMST:

1. Let O be OPT tree.

   Let T be SteinerMST tree.
2. Let D = DFS on O.

   cost(D) = 2*OPT.
3. D = {H, A, G, D, G, E, G, A, H}

Steiner Tree Problem

Algorithm SteinerMST:

1. Let O be OPT tree.

   Let T be SteinerMST tree.

2. Let D = DFS on O.

   cost(D) = 2*OPT.

3. D = {H, A, G, D, G, E, G, A, H}

4. cost(D) = w(H,A) + w(A,G) + … + w(A,H)

Steiner Tree Problem

Algorithm SteinerMST:

1. Let O be OPT tree.

   Let T be SteinerMST tree.
2. Let D = DFS on O.

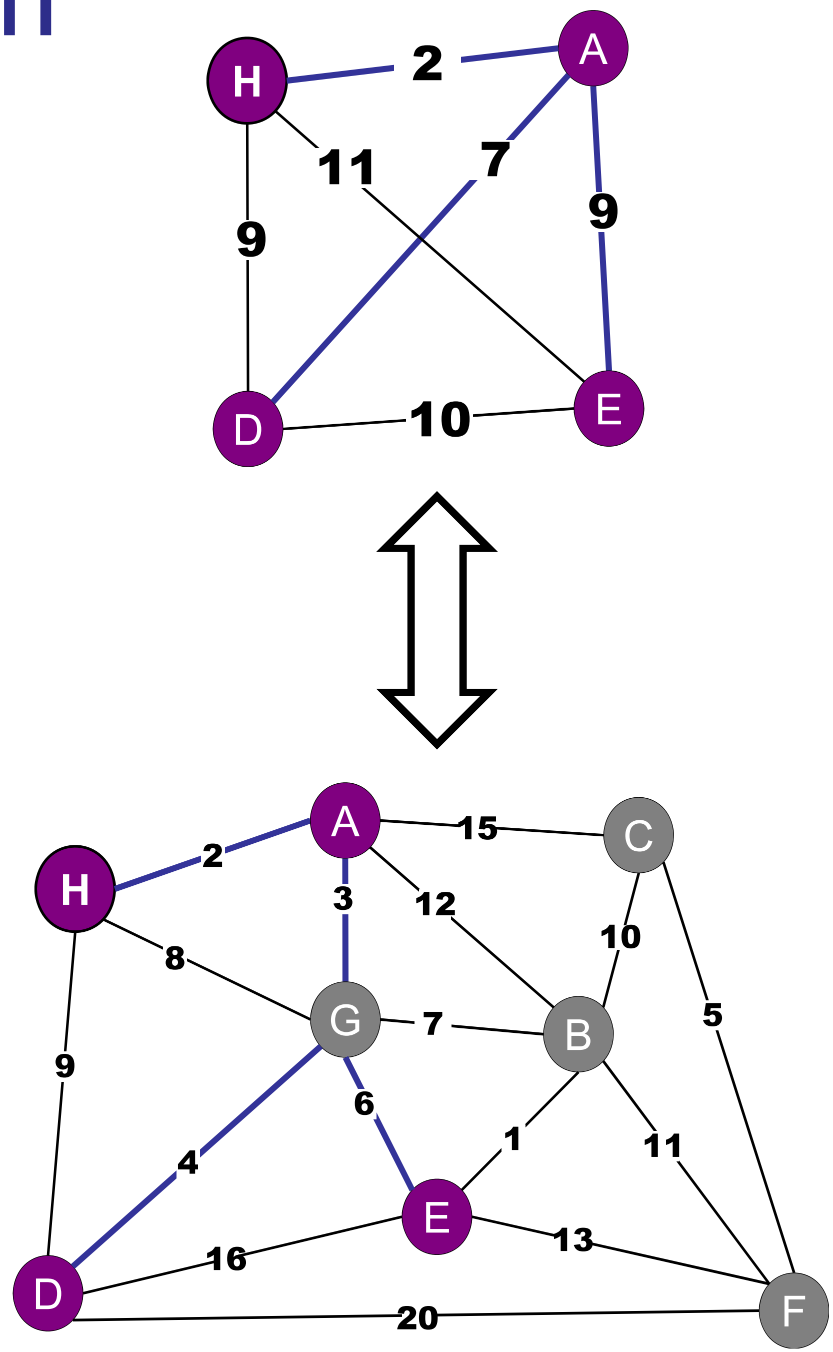
   cost(D) = 2*OPT.
3. D = {H, A, G, D, G, E, G, A, H}
4. cost(D) = w(H,A) + w(A,G) + ... + w(A,H)
5. Skip Steiner Nodes: D’ = {H, A, D, E, A, H}

Steiner Tree Problem

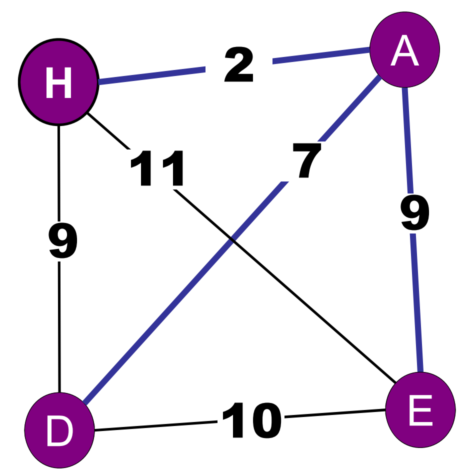
D′ = {H, A, D, E, A, H}

D′ = set of edges on shortest path graph

D′ = spanning subgraph of shortest path graph



Steiner Tree Problem



D′ = {H, A, D, E, A, H}

D′ = set of edges on shortest path graph

D′ = spanning subgraph of shortest path graph

cost(D′) = cost of traversing shortest paths

<= cost(D) <= 2*OPT

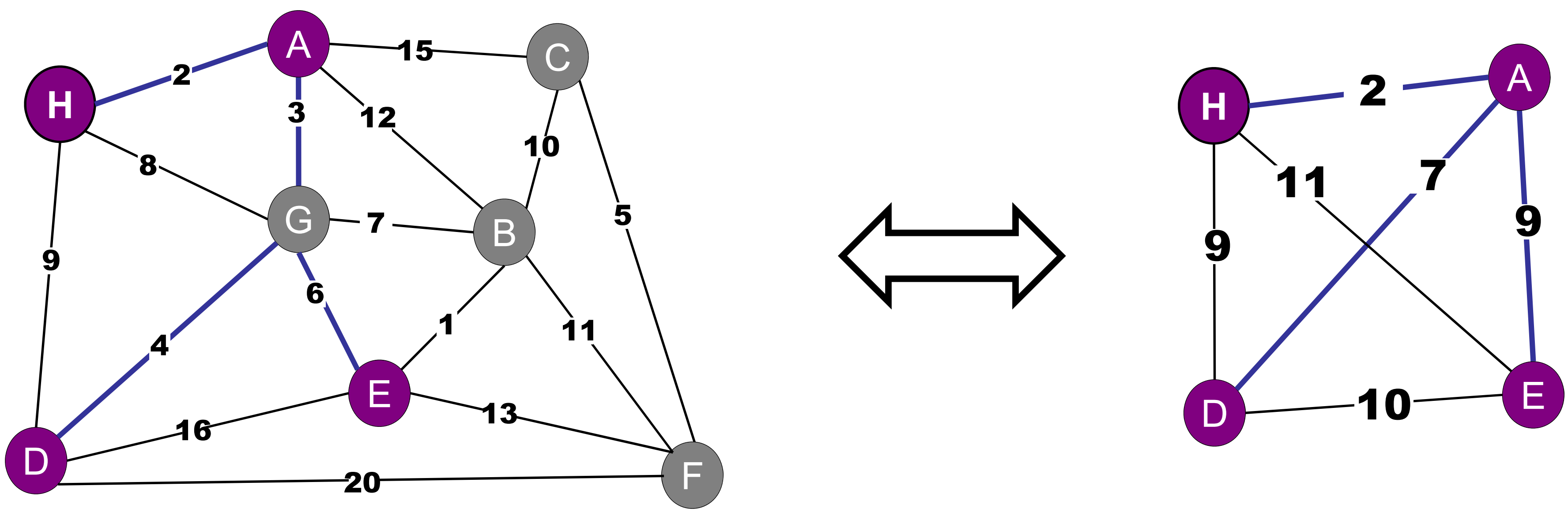
Steiner Tree Problem

Algorithm SteinerMST:

6. cost(D’) = cost of traversing shortest paths

   <= cost(D) <= 2*OPT.

7. D’ spans shortest path graph
8. cost(T) = cost(MST) < cost(D’) <= 2*OPT



Steiner Tree Problem

Algorithm SteinerMST:

1. For every required vertex (v,w), calculate the shortest path from (v to w).
2. Construct new graph on required nodes.
3. Run MST on new graph.
4. Map new edges back to original graph.

Note: Does NOT guarantee optimal Steiner tree.
Best known approximation: 1.55

Roadmap

Last time: Minimum Spanning Trees

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

Today: Variations:

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