Minimum Spanning Tree: Prim’s & Kruskal’s Algorithms

Prim’s Algorithm

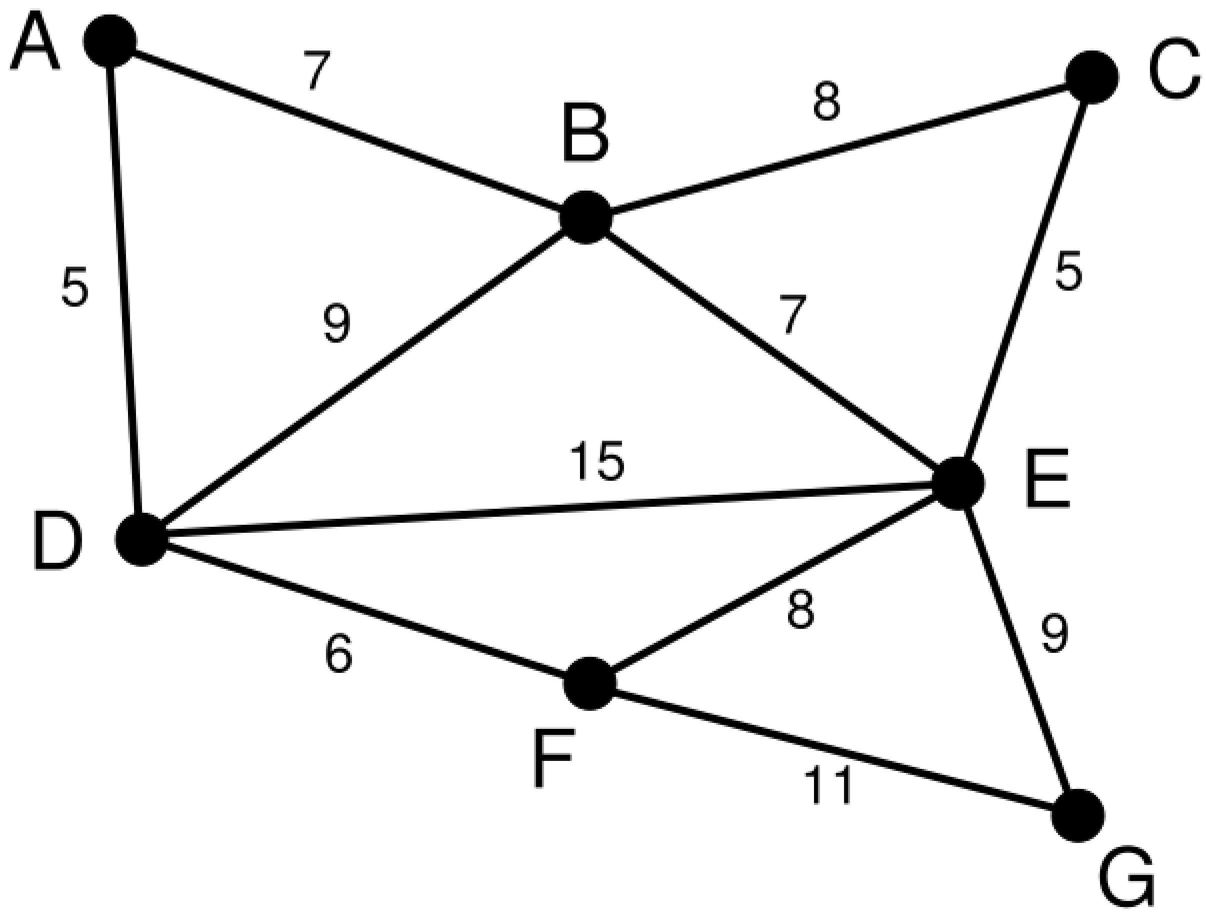
- Prim's algorithm finds a minimum spanning tree (MST) for a connected weighted graph.
- MST = subset of edges that forms a tree including every vertex, such that total weight of all edges is minimized

- Vojtěch Jarník: *O jistém problému minimálním* [About a certain minimal problem], Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57-63. (in Czech)
- Robert C. Prim: *Shortest connection networks and some generalisations*. In: *Bell System Technical Journal*, 36 (1957), pp. 1389–1401
- Rediscovered by Edsger Dijkstra in 1959
- *aka DJP algorithm, Jarník algorithm, Prim-Jarník algorithm*

Prim’s Algorithm

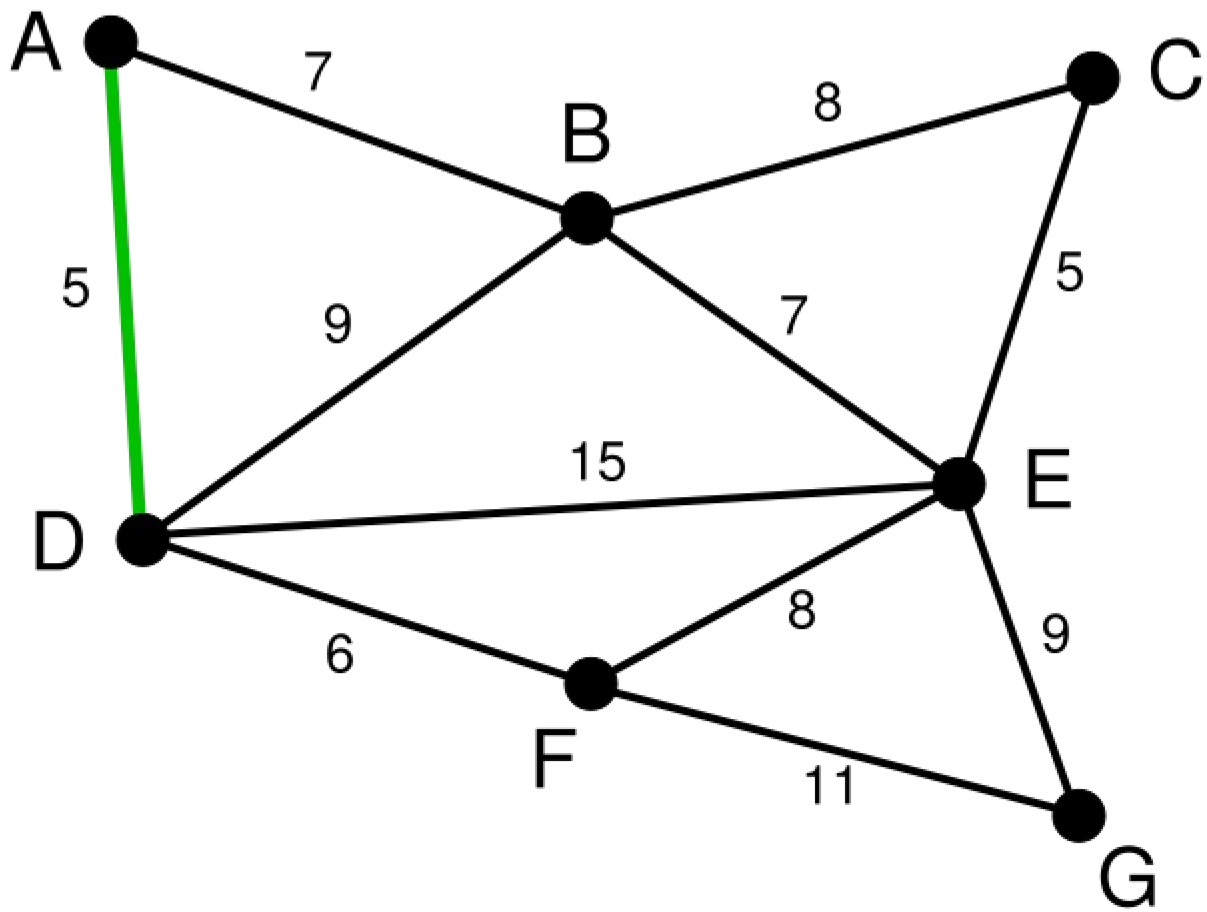
```
for each vertex in graph
   set min_distance of vertex to ∞
   set parent of vertex to null
   set minimum_adjacency_list of vertex to empty list
   set is_in_Q of vertex to true
set distance of initial vertex to zero
add to minimum-heap Q all vertices in graph.

while latest_addition = remove minimum in Q
    set is_in_Q of latest_addition to false
    add latest_addition to (minimum_adjacency_list of (parent of latest_addition))
    add (parent of latest_addition) to (minimum_adjacency_list of latest_addition)

    for each adjacent of latest_addition
    if (is_in_Q of adjacent) and (weight-function(latest_addition, adjacent) <
         min_distance of adjacent)
        set parent of adjacent to latest_addition
        set min_distance of adjacent to weight-function(latest_addition, adjacent)

update adjacent in Q, order by min_distance
```

Prim's Algorithm 1



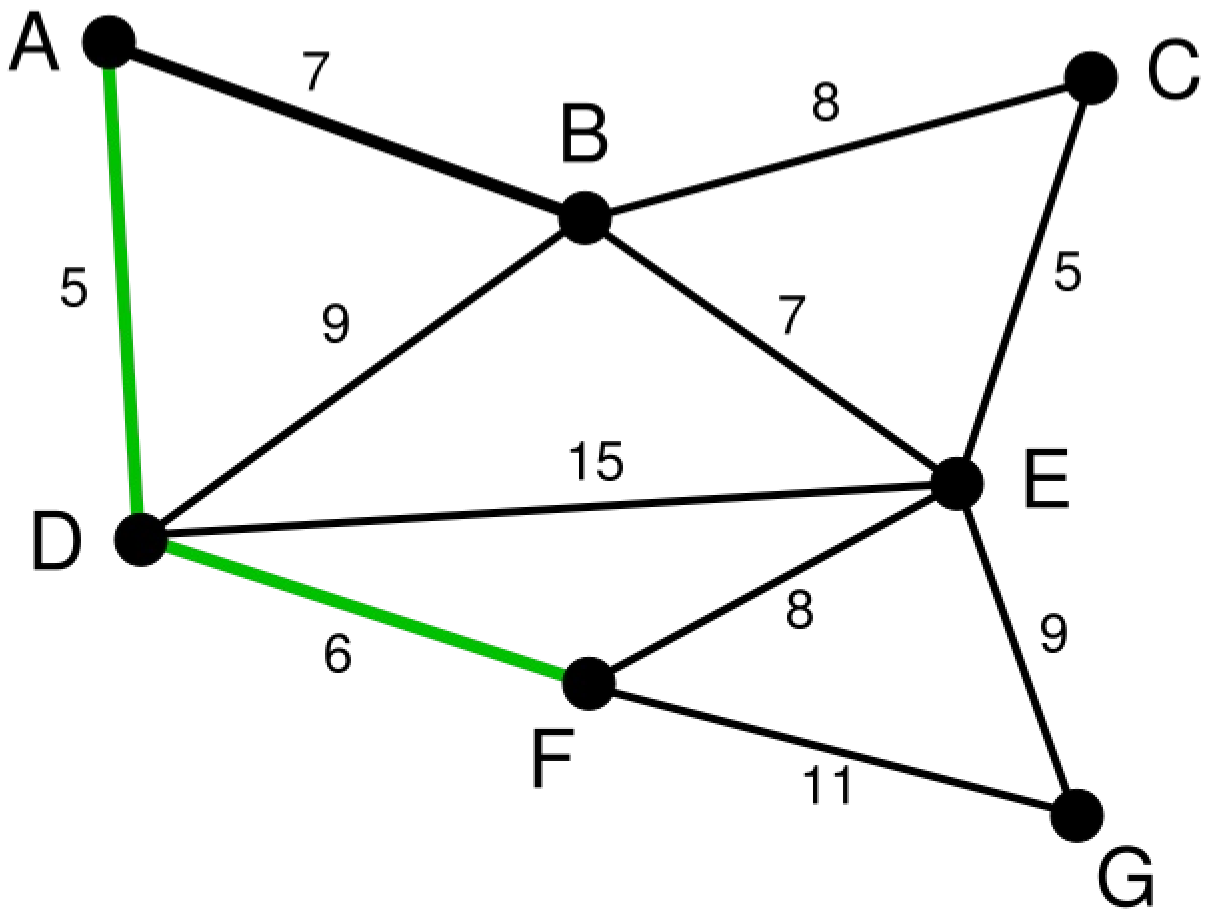
Not Seen	Fringe	Solution
C, G	A, B, E, F	D

Prim's Algorithm 2



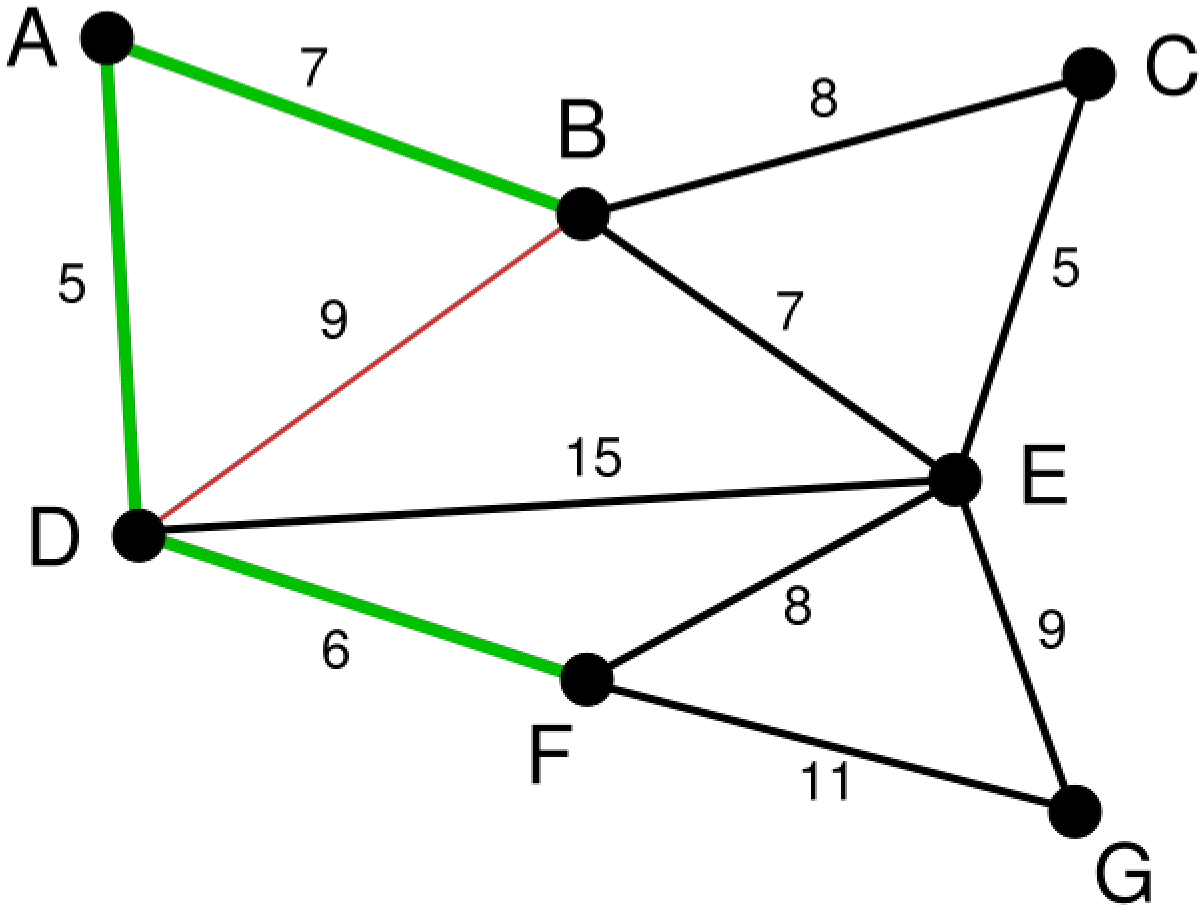
Not Seen	Fringe	Solution
C, G	B, E, F	A, D

Prim's Algorithm 3



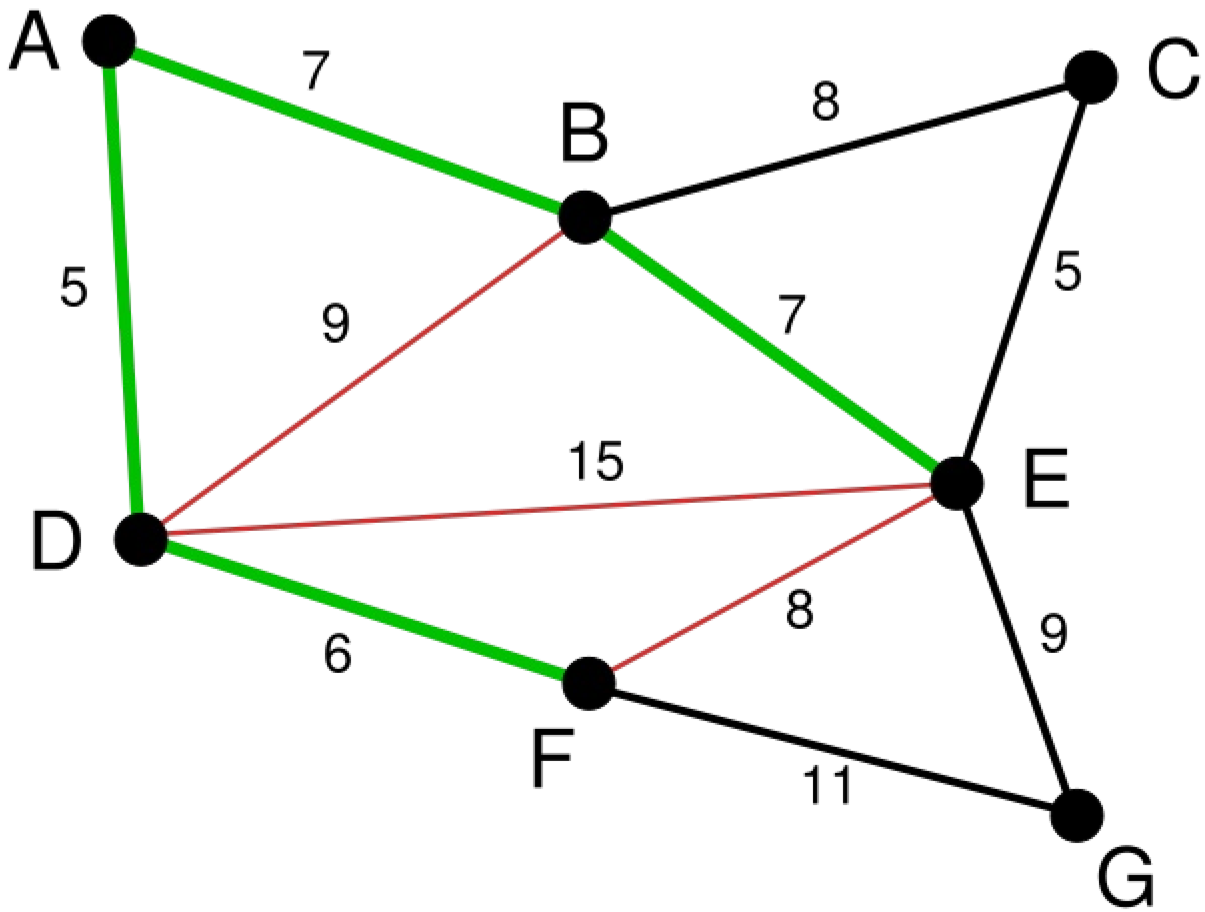
Not Seen	Fringe	Solution
C	B, E, G	A, D, F

Prim’s Algorithm 4



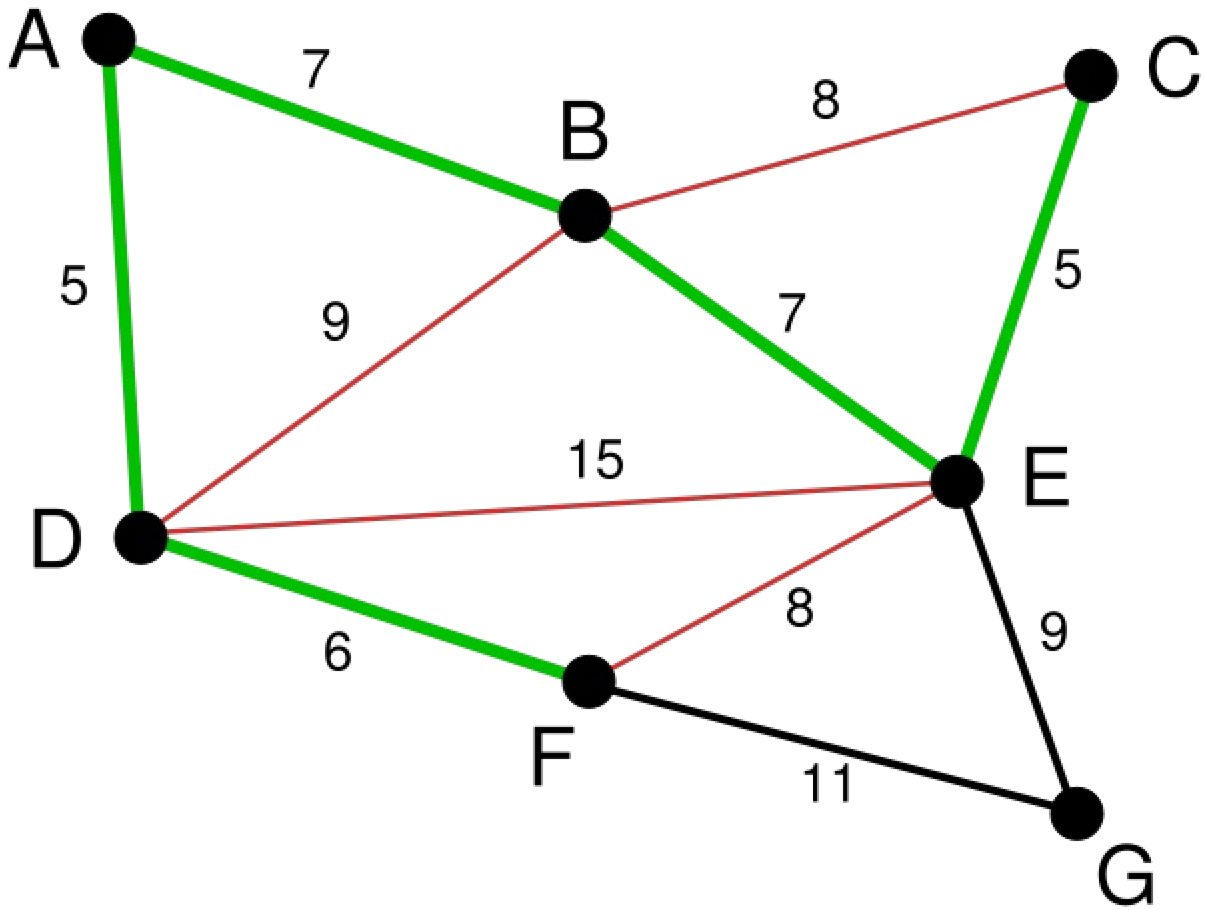
Not Seen	Fringe	Solution
—	C, E, G	A, D, F, B

Prim's Algorithm 5



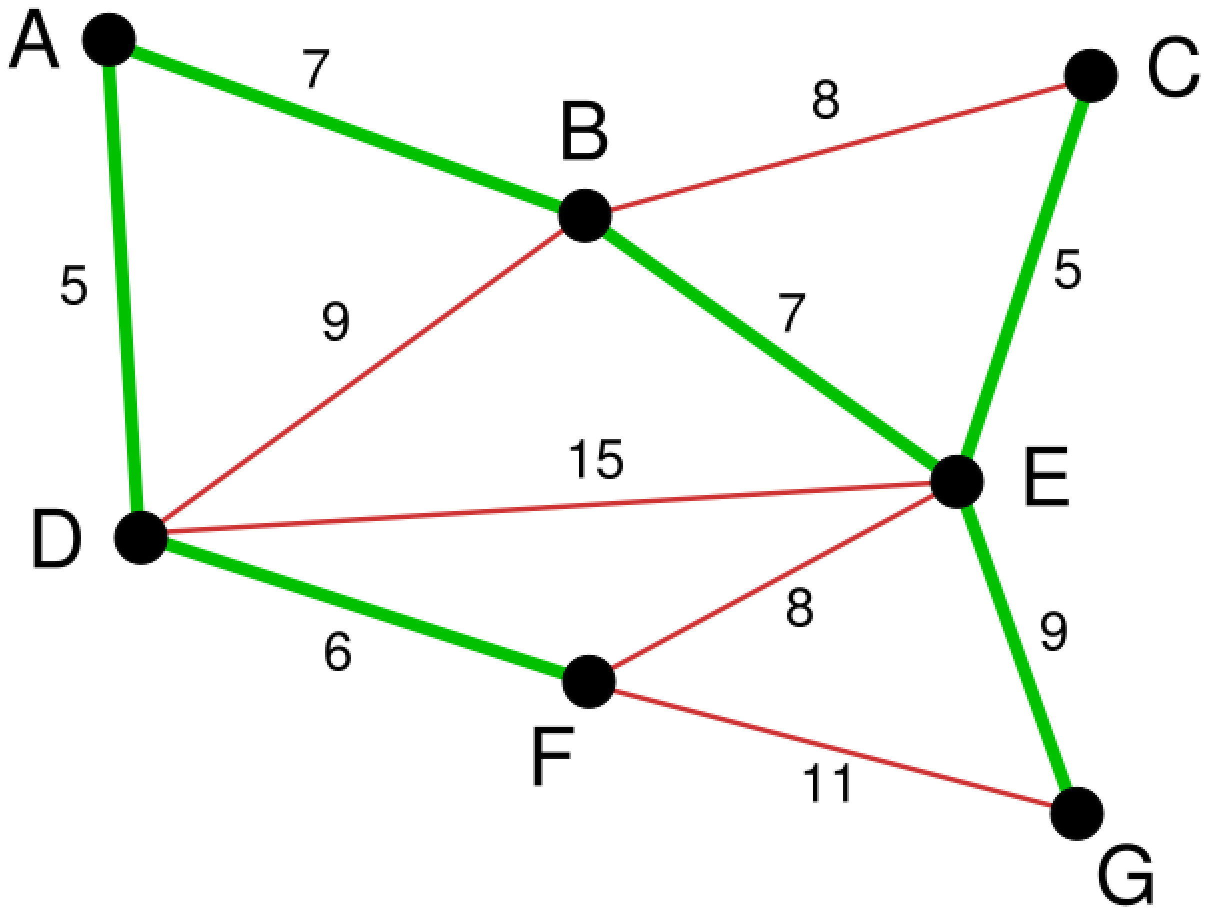
Not Seen	Fringe	Solution
—	C, G	A, D, F, B, E

Prim’s Algorithm 6



Not Seen	Fringe	Solution
—	G	A, D, F, B, E, C

Prim's Algorithm 7



Not Seen	Fringe	Solution
—	—	A, D, F, B, E, C, G

Prim's Algorithm

```
public class Algorithms
{
    public static Graph PrimsAlgorithm(Graph g, int s)
    {
        int n = g.NumberOfVertices;
        Entry[] table = new Entry[n];
        for (int v = 0; v < n; ++v)
            table[v] = new Entry(false,
                int.MaxValue, int.MaxValue);
        table[s].distance = 0;
        PriorityQueue queue = new BinaryHeap(g.NumberOfEdges);
        queue.Enqueue(new Association(0, g.GetVertex(s)));

while (!queue.IsEmpty)
{
    Association assoc = (Association)queue.DequeueMin();
    Vertex v0 = (Vertex)assoc.Value;
    if (!table[v0.Number].known)
    {
        table[v0.Number].known = true;
        foreach (Edge e in v0.EmanatingEdges)
        {
            Vertex v1 = e.MateOf(v0);
            int d = (int)e.Weight;
            if (!table[v1.Number].known &&
                table[v1.Number].distance>d)
            {
                table[v1.Number].distance = d;
                table[v1.Number].predecessor = v0.Number;
                queue.Enqueue(new Association(d, v1));
            }
        }
    }
}

        Graph result = new GraphAsLists(n);
        for (int v = 0; v < n; ++v)
            result.AddVertex(v);
        for (int v = 0; v < n; ++v)
            if (v != s)
                result.AddEdge(v, table[v].predecessor);
        return result;
    }
}
```

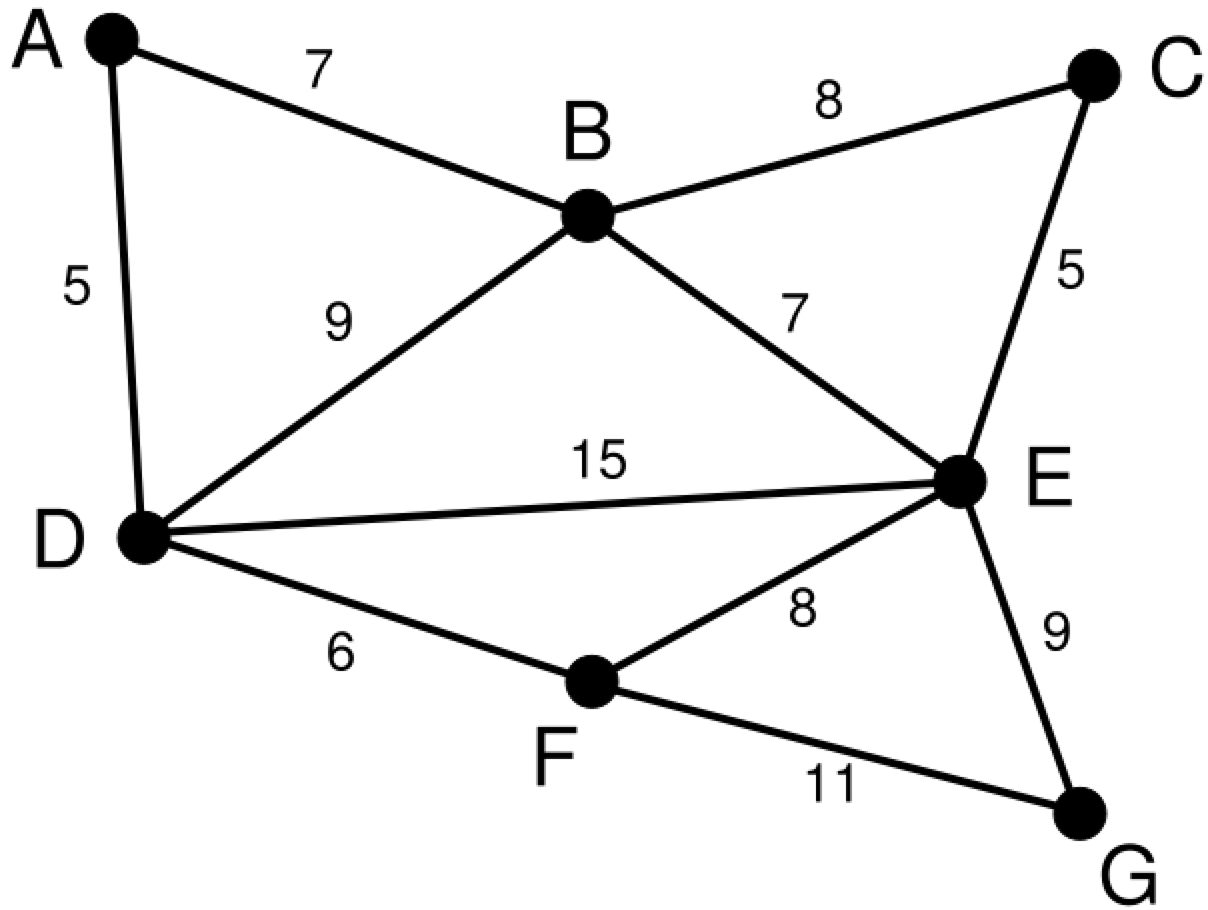
Kruskal’s Algorithm

- Kruskal's algorithm finds a minimum spanning tree (MST) for a connected weighted graph.
- MST = subset of edges that forms a tree including every vertex, such that total weight of all edges is minimized
- If the graph is not connected, the minimum spanning forest will be found, i.e., MST for each connected component.
  - This is because all edges are examined in order of weight.
- Kruskal’s is a greedy algorithm
- Joseph. B. Kruskal: *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*. In: *Proceedings of the American Mathematical Society*, Vol 7, No. 1 (Feb, 1956), pp. 48–50

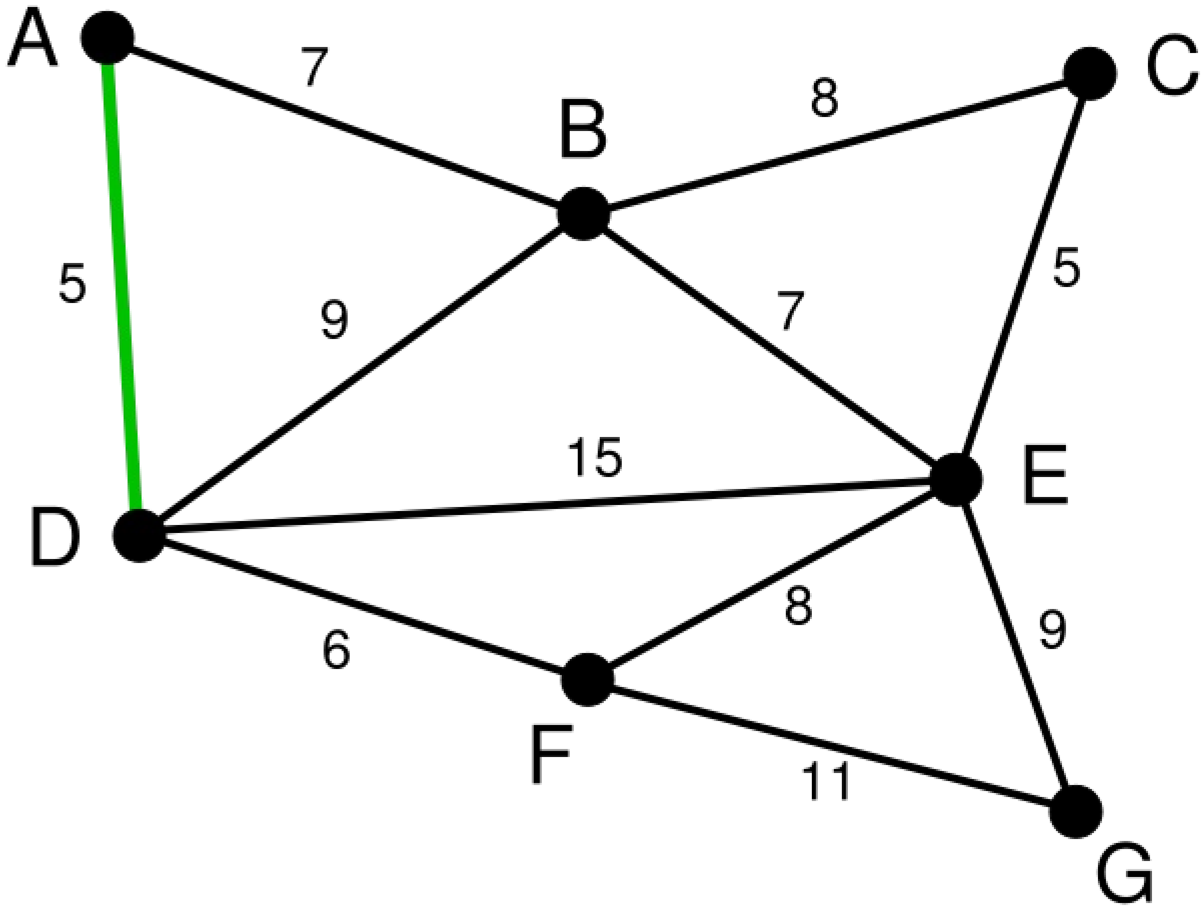
Kruskal's Algorithm

```
function Kruskal(G)
  for each vertex v in G do
    Define an elementary cluster C(v) ← {v}.

  Initialize a priority queue Q to contain all edges in G, using the weights as keys.
  Define a tree T ← Ø       //T will ultimately contain the edges of the MST

  // n is total number of vertices
  while T has fewer than n-1 edges do

     (u,v) ← Q.removeMin()

     // prevent cycles in T. add edge u,v only if T does not already contain an edge
  consisting of u and v.
     // Note that the cluster contains more than one vertex only if an edge containing
        a pair of the vertices has been added to the tree.

    Let C(v) be the cluster containing v, and let C(u) be the cluster containing u.
    if C(v) ≠ C(u) then
      Add edge (v,u) to T.
      Merge C(v) and C(u) into one cluster, that is, union C(v) and C(u).

  return tree T
```

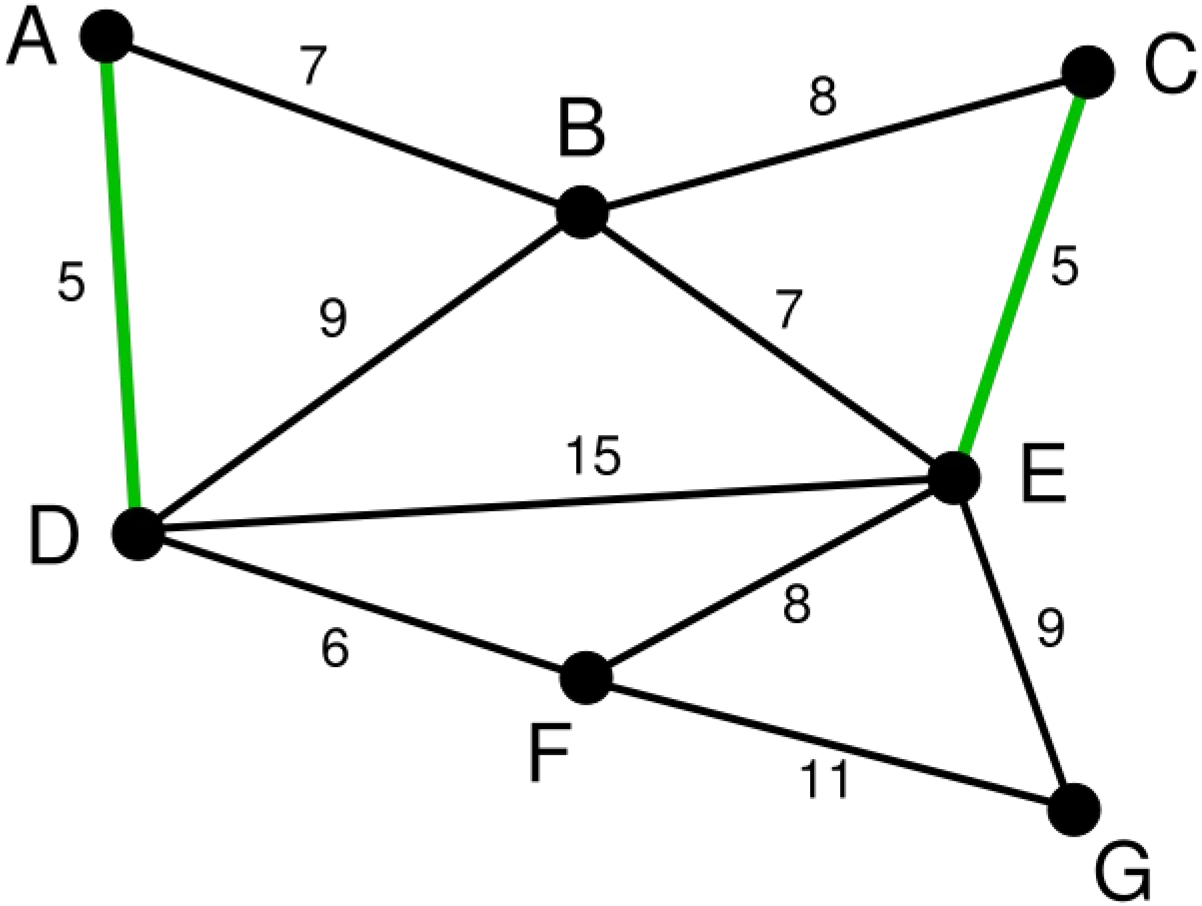
Kruskal’s Algorithm 1



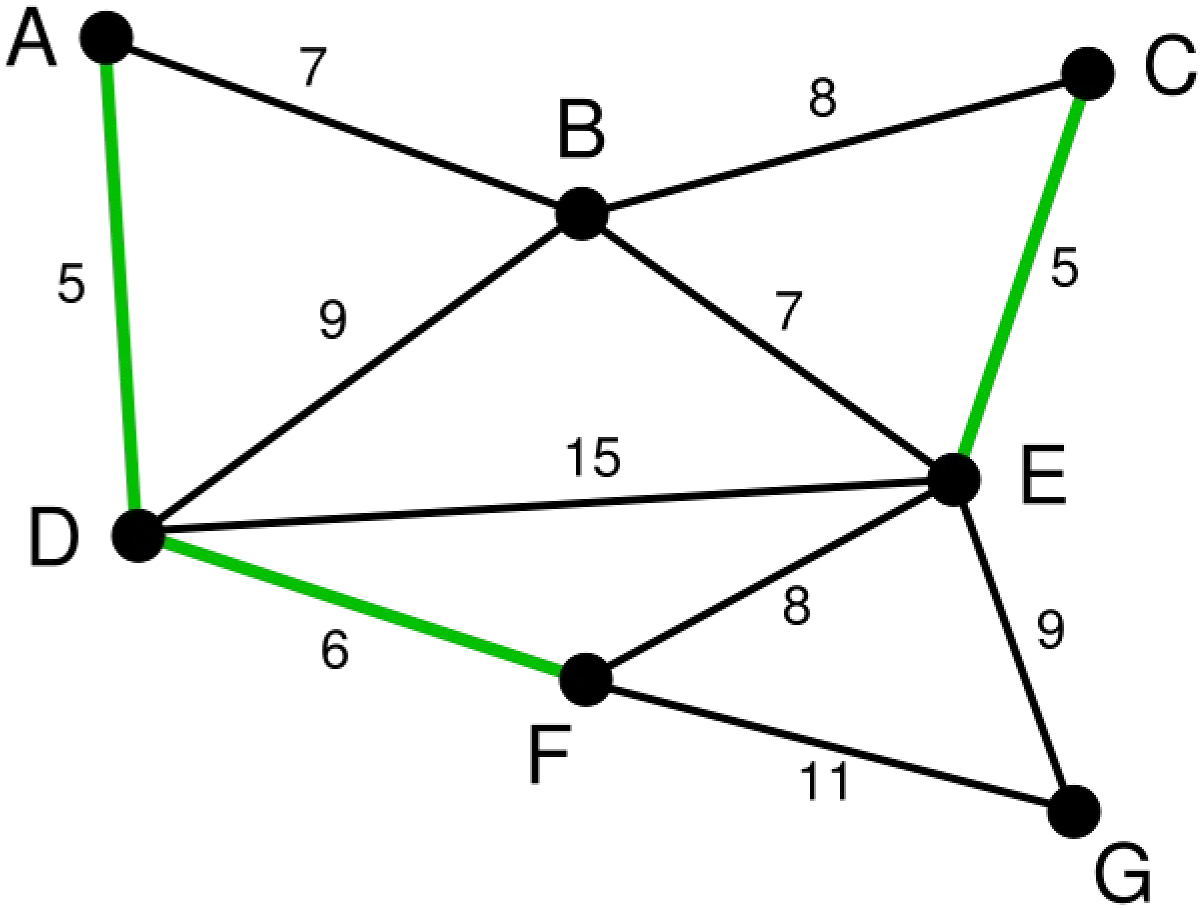
Kruskal’s Algorithm 2



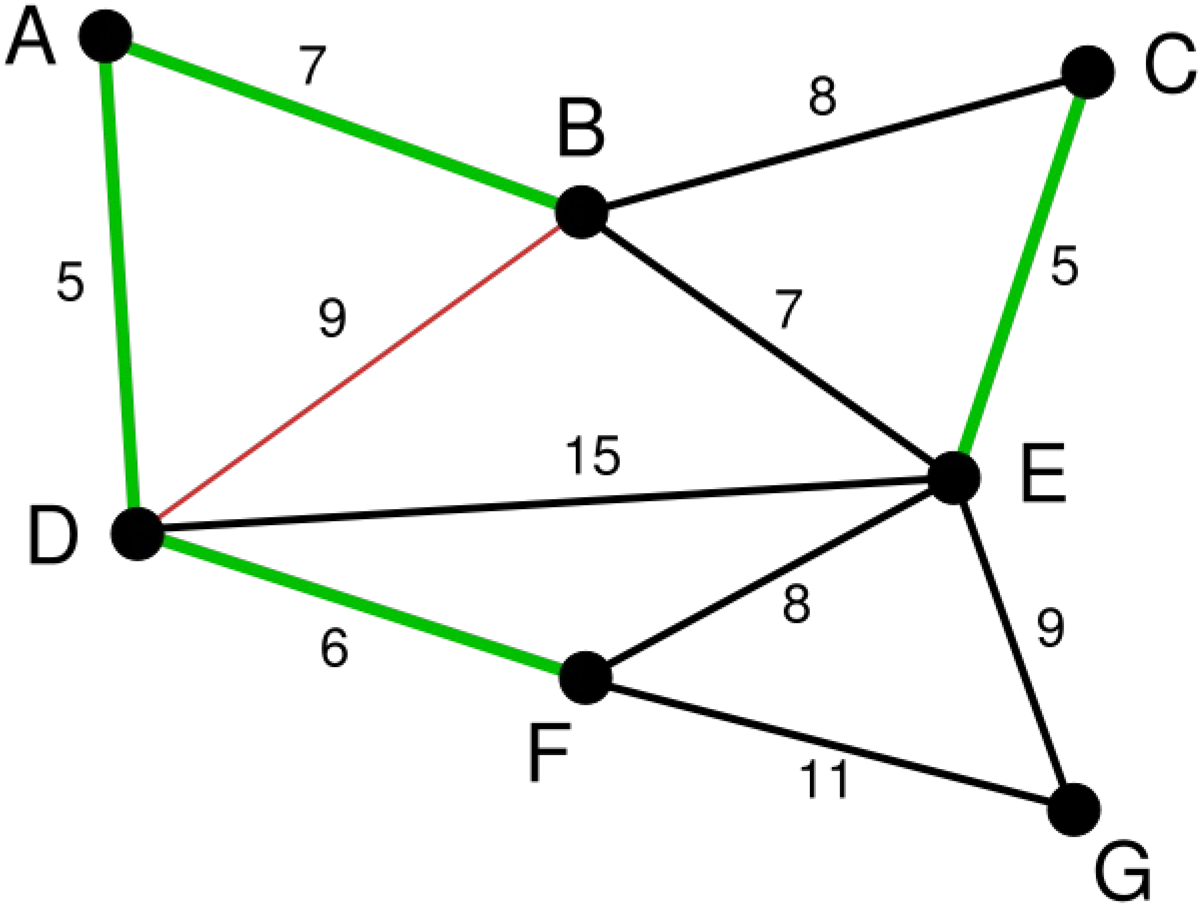
Kruskal’s Algorithm 3



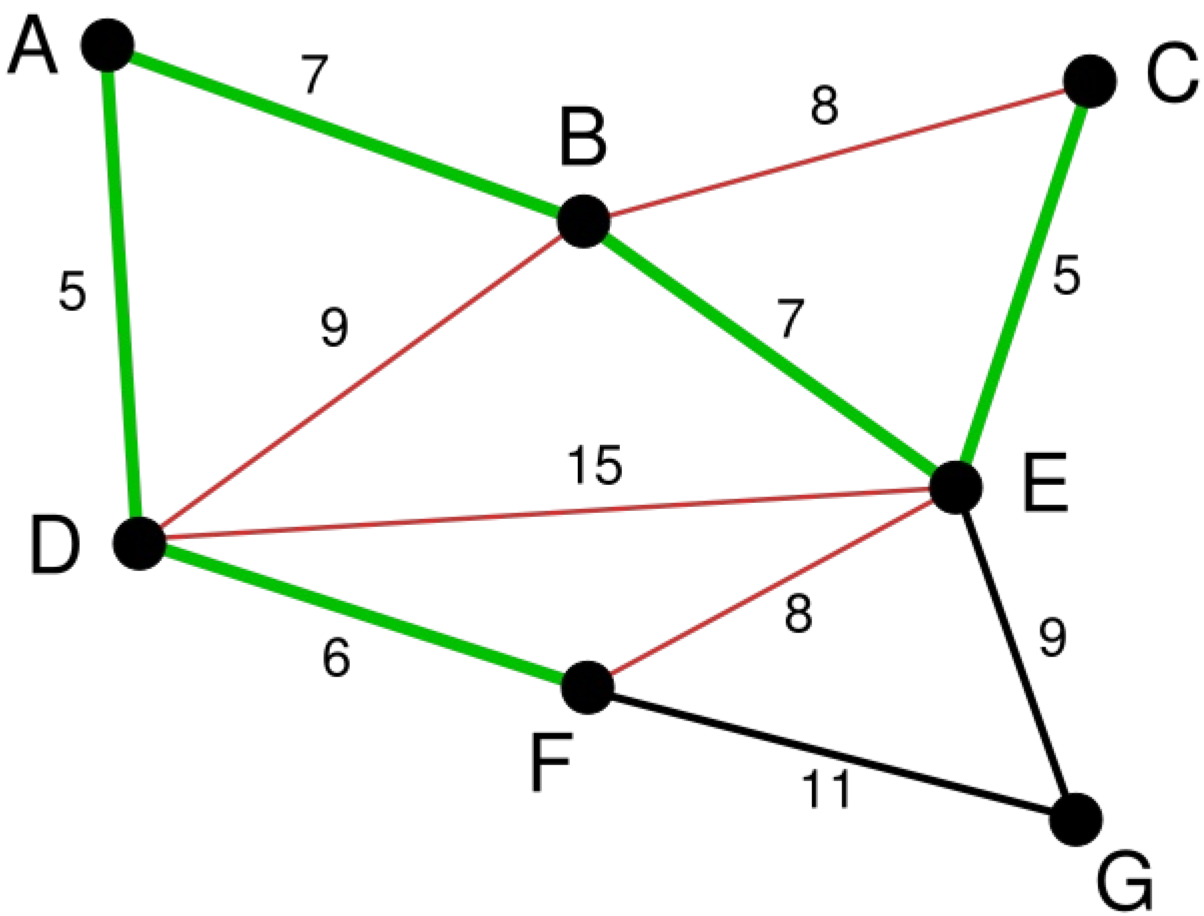
Kruskal’s Algorithm 4



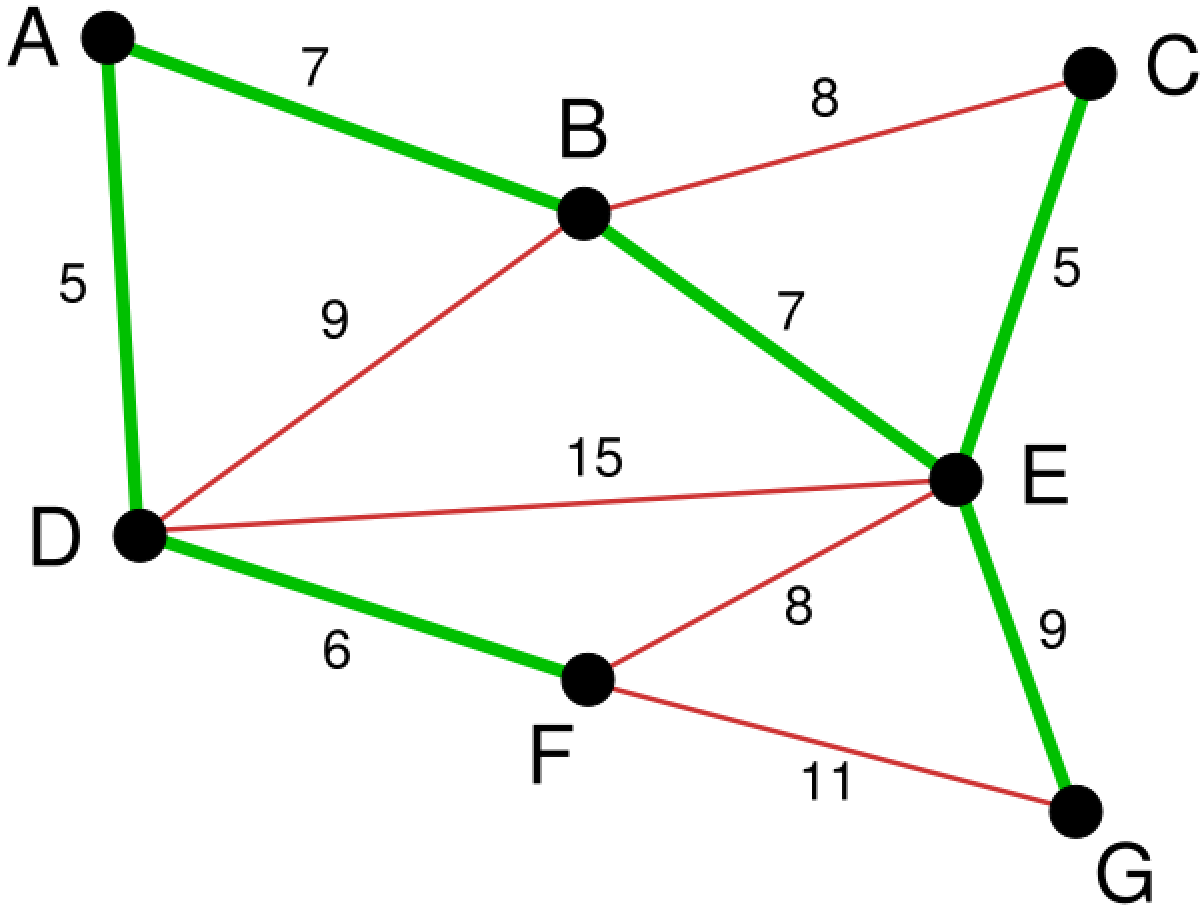
Kruskal’s Algorithm 5



Kruskal’s Algorithm 6



Kruskal's Algorithm 7



Kruskal's Algorithm

```
public class Algorithms
{
    public static Graph KruskalsAlgorithm(Graph g)
    {
        int n = g.NumberOfVertices;

        Graph result = new GraphAsLists(n);
        for (int v = 0; v < n; ++v)
            result.AddVertex(v);

        PriorityQueue queue =
            new BinaryHeap(g.NumberOfEdges);
        foreach (Edge e in g.Edges)
        {
            int weight = (int)e.Weight;
            queue.Enqueue(new Association(weight, e));
        }

        Partition partition = new PartitionAsForest(n);
        while (!queue.IsEmpty && partition.Count > 1)
        {
            Association assoc = (Association)queue.DequeueMin();
            Edge e = (Edge)assoc.Value;
            int n0 = e.V0.Number;
            int n1 = e.V1.Number;
            Set s = partition.Find(n0);
            Set t = partition.Find(n1);
            if (s != t)
            {
                partition.Join(s, t);
                result.AddEdge(n0, n1);
            }
        }
        return result;
    }
}
```

Prim’s & Kruskal’s Demos

- Prim’s:
  - http://www.unf.edu/~wkloster/foundations/PrimApplet/PrimApplet.htm
- Kruskal’s:
  - http://students.ceid.upatras.gr/~papagel/project/kruskal.htm
  - http://www.unf.edu/~wkloster/foundations/KruskalApplet/KruskalApplet.htm

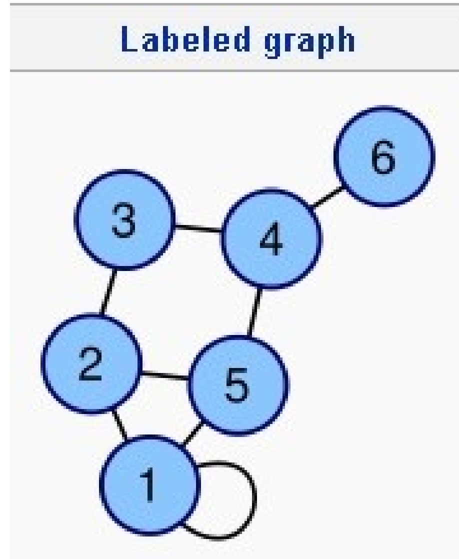
Prim’s & Kruskal’s Running Time

{1,1}, {1,2}, {1,5}

{2,3}, {2,5}

{3,4}

{4,5}, {4,6}



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

	Input is Adjacency List	Input is Adjacency Matrix
Prim’s	$O(V+E(\log E))$	$O(V^2+E(\log E))$
Kruskal’s	$O(V+E(\log E)+E(\log V))$	$O(V^2+E(\log E)+E(\log V))$

Sources

Ikeda, Kenji. http://www-b2.is.tokushima-u.ac.jp/~ikeda/

Panagiotis, Papaioannou.http://students.ceid.upatras.gr/~papagel/

B. R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in C#.*

Cormen et al. *Introduction to Algorithms.*

Wikipedia