```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PrimAndKruskal
{
    class Program
    {
        public class Edge
        {
            public int first;
            public int second;
            public int weight;

            public Edge(int first, int second, int weight)
            {
                this.first = first;
                this.second = second;
                this.weight = weight;
            }
        }
        public class Graph
        {
            public List<Edge> edges;
            public int vertexCount;
            public Graph(IEnumerable<Edge> edges)
            {
                this.edges = edges.ToList();
                vertexCount = edges.GroupBy(e => e.first).Count() + 1;
            }
            public Dictionary<int, List<Edge>> ToDictionary()
            {
                Dictionary<int, List<Edge>> result = new Dictionary<int, List<Edge>>();
                foreach (var edge in edges)
                {
                    AddInDictionary(result, edge.first, edge);
                    AddInDictionary(result, edge.second, edge);
                }
                return result;
            }
            private void AddInDictionary(Dictionary<int, List<Edge>> result, int vertex, Edge
edge)
            {
                if (result.ContainsKey(vertex))
                {
                    result[vertex].Add(edge);
                }
                else
                {
                    result[vertex] = new List<Edge>() { edge };
    }
}
public void Kruskal(Graph graph)
{
    List<Edge> result = new List<Edge>();

    List<Edge> orderedEdgeList = graph.edges.OrderBy(g => g.weight).ToList();

    int[] visited = new int[graph.vertexCount];
    for (int i = 0; i < visited.Length; i++)
    {
        visited[i] = i;
    }

    for (int i = 0; i < orderedEdgeList.Count; i++)
    {
        int firstVertex = orderedEdgeList[i].first;
        int secondVertex = orderedEdgeList[i].second;

        if (visited[firstVertex] != visited[secondVertex])
        {
            result.Add(orderedEdgeList[i]);

            int max = Math.Max(visited[firstVertex], visited[secondVertex]);

            int firstVertexMark = visited[firstVertex];
            int secondVertexMark = visited[secondVertex];

            for (int j = 0; j < visited.Length; j++)
            {
                if (visited[j] == firstVertexMark || visited[j] == secondVertexMark)
                {
                    visited[j] = max;
                }
            }
        }
    }
}
public void Prim(Graph graph)
{
    List<Edge> result = new List<Edge>();
    bool[] visited = new bool[graph.vertexCount];
    Dictionary<int, List<Edge>> dictionaryGraph = graph.ToDictionary();
    List<Edge> adjacentEdges = new List<Edge>();
    Edge currentEdge = graph.edges.OrderBy(g => g.weight).First();
    while (visited.Contains(false))
    {
        dictionaryGraph[currentEdge.first].Remove(currentEdge);
        dictionaryGraph[currentEdge.second].Remove(currentEdge);
        adjacentEdges.Remove(currentEdge);
        adjacentEdges.Union(dictionaryGraph[currentEdge.first]);
        adjacentEdges.Union(dictionaryGraph[currentEdge.second]);
```

```
                    result.Add(currentEdge);
                    visited[currentEdge.first] = true;
                    visited[currentEdge.second] = true;
                    List<Edge> currentEdgeList = adjacentEdges.OrderBy(e => e.weight).ToList();
                    IEnumerator<Edge> enumerator = currentEdgeList.GetEnumerator();
                    while (enumerator.MoveNext() && visited[currentEdge.first] && visited
[currentEdge.second])
                    {
                        currentEdge = enumerator.Current;
                    }
                }
            }
        }
    }
}
```