Алгоритм Прима

это алгоритм поиска минимального остовного дерева связного взвешенного неориентированного графа. Был открыт Робертом Примом в 1957 году.

Граф - совокупность узлов (вершин) и связывающих их ребер без дополнительных ограничений на них.

Дерево – это частный случай графа.

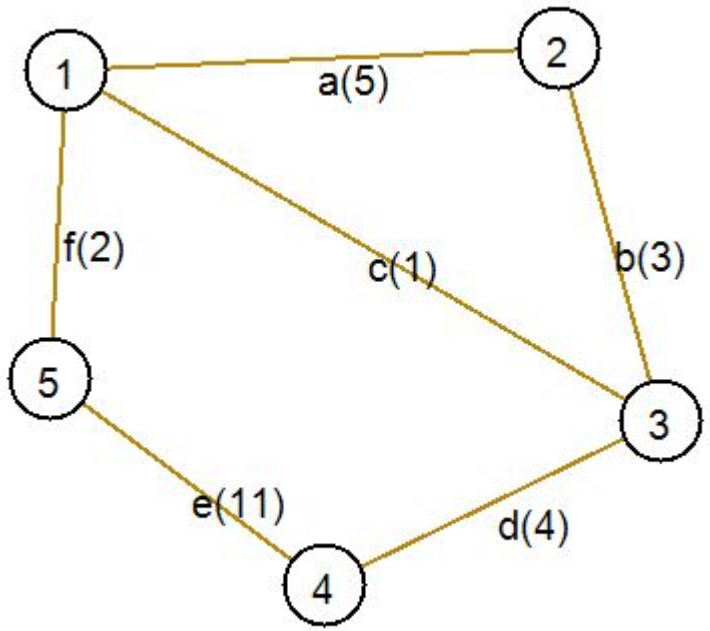
Вес ребра – числовое значение, поставленное в соответствие данному ребру графа. Вес ребра можно интерпретировать как длину ребра.

Взвешенный граф – граф, для каждого ребра которого задан вес.

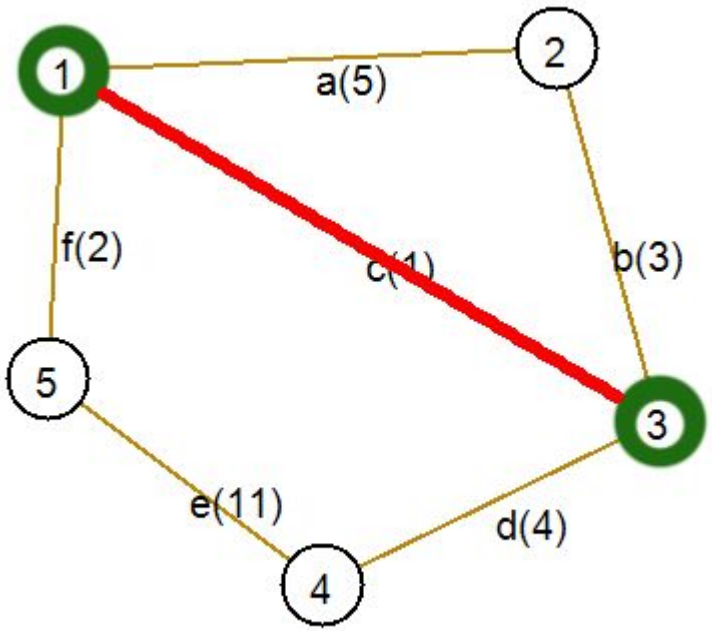
Связный граф – это такой граф, в котором из любой его вершины можно попасть в любую другую вершину.

Остовное дерево – это связный подграф без циклов данного связного неориентированного графа, в который входят все его вершины.

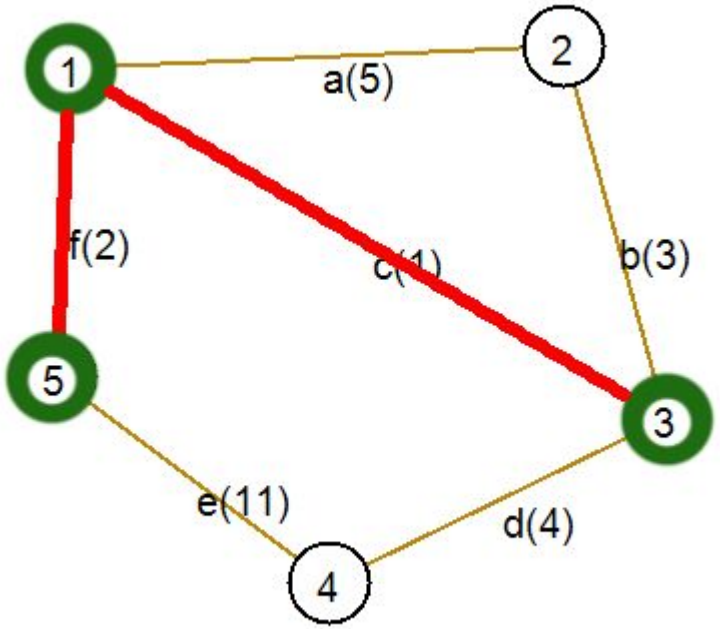
Минимальное остовное дерево – это остовное дерево данного графа имеющее минимальный возможный вес. Под весом остовного дерева понимается сумма весов всех ребер, входящих в него.



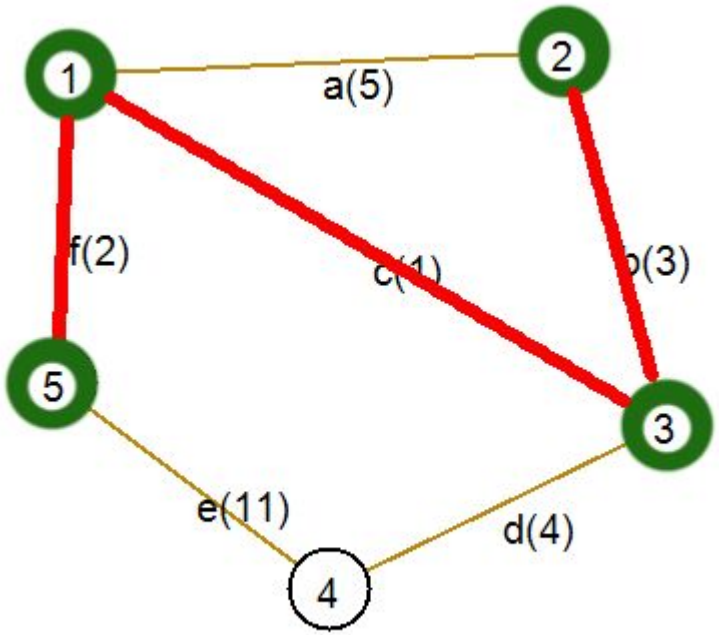
Пусть дан граф, как на рисунке ниже. В скобках указаны веса ребер.



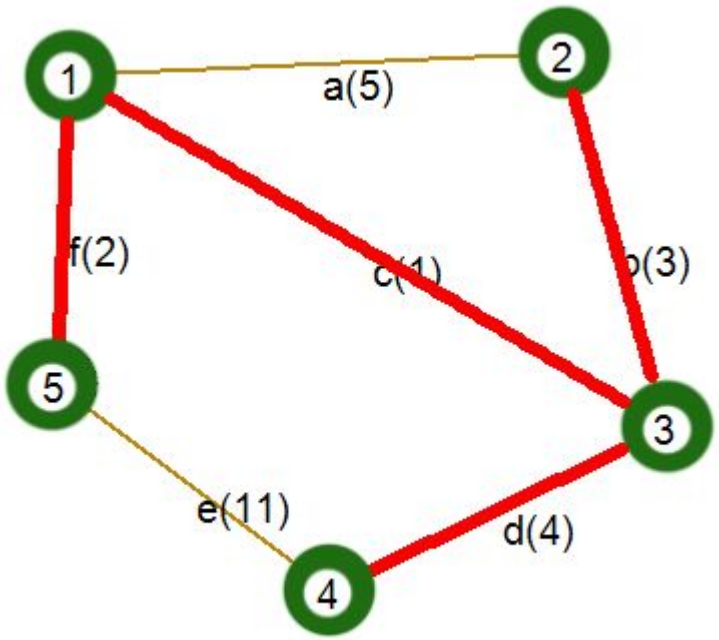
Шаг 1. Выберем произвольную вершину. Пусть это будет вершина номер **3**. Ей инцидентны ребра (с неиспользованными вершинами): b(3), c(1), d(4). Ребро с наименьшим весом – **c**. Включим его и инцидентную ему вершину в дерево.



Шаг 2. Вершинам **1** и **3** инцидентны ребра (с неиспользованными вершинами): a(5), b(3), d(4), f(2). Ребро с наименьшим весом – **f**. Включим его и инцидентную ему вершину в дерево.



Шаг 3. Вершинам **1**, **3** и **5** инцидентны ребра (с неиспользованными вершинами): a(5), b(3), d(4), e(11). Ребро с наименьшим весом – **b**. Включим его и инцидентную ему вершину в дерево.



Шаг 4. Вершинам **1**, **2**, **3** и **5** инцидентны ребра (с неиспользованными вершинами): d(4) и e(11). Ребро с наименьшим весом – **d**. Включим его и инцидентную ему вершину в дерево.

Все вершины графа включены в дерево. Работа алгоритма завершена.

```
class Edge
{
    public int v1, v2;

    public int weight;

    public Edge(int v1, int v2, int weight)
    {
        this.v1 = v1;
        this.v2 = v2;
        this.weight = weight;
    }
}
```

Пусть ребро графа представлено экземпляром класса **Edge**, где
v1 и **v2** – номера вершин (нумерация вершин начинается с нуля), инцидентных ребру,
weight - вес ребра.

```
//все рёбра графа
List<Edge> E;
//неиспользованные ребра
List<Edge> notUsedE = new List<Edge>(E);
//использованные вершины
List<int> usedV = new List<int>();
//неиспользованные вершины
List<int> notUsedV = new List<int>();
for (int i = 0; i < numberV; i++)
    notUsedV.Add(i);

//выбираем случайную начальную вершину
Random rand = new Random();
usedV.Add(rand.Next(0, numberV));
notUsedV.RemoveAt(usedV[0]);
```

В начале инициализируются списки с данными: ребра, не включенные в дерево, вершины, включенные в дерево, и вершины, не включенные в дерево

Затем выбирается случайная начальная вершина, с которой начнется построение минимального остовного дерева

```
while (notUsedV.Count > 0)
  {
     int minE = -1; //номер наименьшего ребра
     //поиск наименьшего ребра
     for (int i = 0; i < notUsedE.Count; i++)
     {
        if ((usedV.IndexOf(notUsedE[i].v1) != -1) &&
     (notUsedV.IndexOf(notUsedE[i].v2) != -1) ||
           (usedV.IndexOf(notUsedE[i].v2) != -1) &&
(notUsedV.IndexOf(notUsedE[i].v1) != -1))
        {
           if (minE != -1)
           {
              if (notUsedE[i].weight <notUsedE[minE].weight)
                 minE = i;
           }
           else
              minE = i;
        }
     }
..............................................................................
   }
```

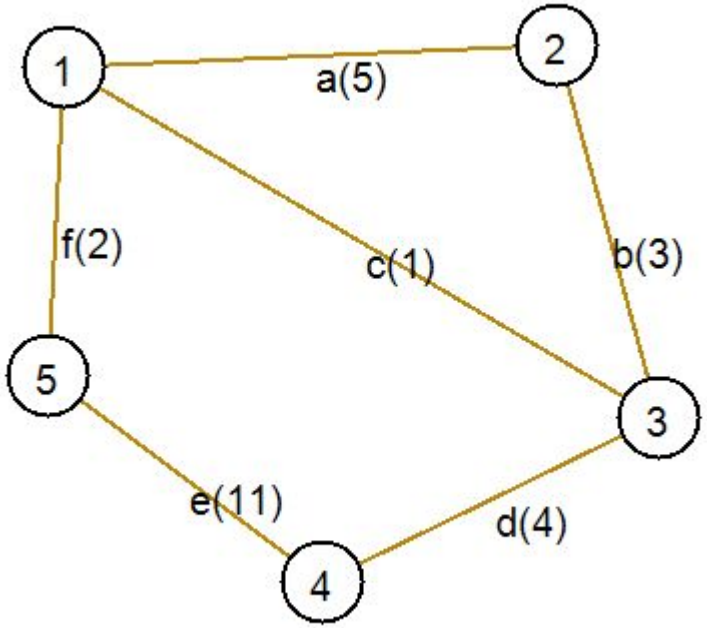
Цикл **while** будет продолжаться до тех пор, пока все вершины графа не будут включены в дерево.

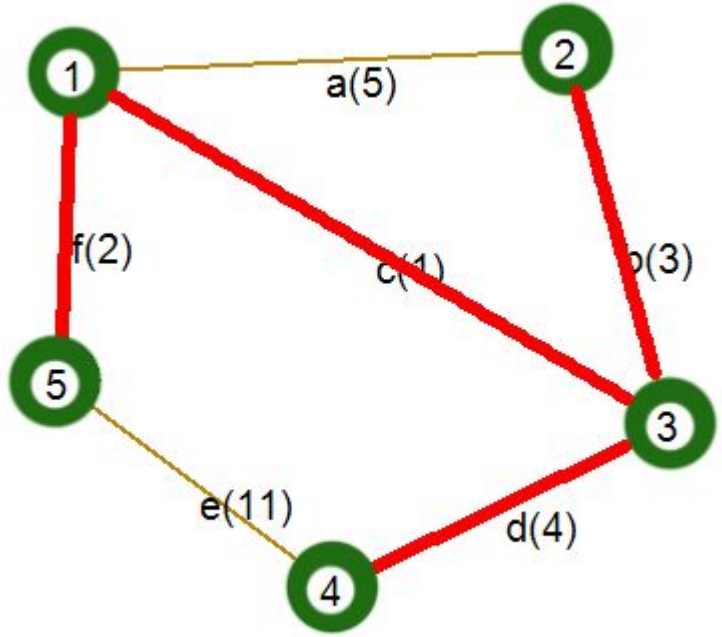
На каждой итерации цикла выполняется следующее:

1. Производится поиск ребра с наименьшим весом, один конец которого – это вершина, входящая в дерево, а другой – нет.

```
    {
……….
//заносим новую вершину в список использованных и
удаляем ее из списка неиспользованных
        if (usedV.IndexOf(notUsedE[minE].v1) != -1)
        {
            usedV.Add(notUsedE[minE].v2);
            notUsedV.Remove(notUsedE[minE].v2);
        }
        else
        {
            usedV.Add(notUsedE[minE].v1);
            notUsedV.Remove(notUsedE[minE].v1);
        }
        //заносим новое ребро в дерево и удаляем его из
списка неиспользованных
        MST.Add(notUsedE[minE]);
        notUsedE.RemoveAt(minE);
    }
```

2. Вершина, инцидентная найденному ребру, заносится в список использованных и удаляется из списка неиспользованных.
3. Найденное ребро заносится в список ребер, составляющих дерево, и удаляется из списка неиспользованных ребер.






```
List<Edge> Edges = new List<Edge>();
Edges.Add(new Edge(0, 1, 5));
Edges.Add(new Edge(1, 2, 3));
Edges.Add(new Edge(2, 3, 4));
Edges.Add(new Edge(3, 4, 11));
Edges.Add(new Edge(0, 4, 2));
Edges.Add(new Edge(0, 2, 1));
```