Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Дисциплина «Алгоритмы и структуры данных»
Лабораторная работа №3

Вариант 19

Выполнил:
Съестов Дмитрий Вячеславович
Группа P3217

Преподаватель:
Зинчик Александр Адольфович

Санкт-Петербург
2018

Задание

Написать программу, реализующую алгоритм А и алгоритм В для проведения экспериментов, в которых можно выбирать:

- число n вершин и число m ребер графа,
- натуральные числа q и r, являющиеся соответственно нижней и верхней границей для весов ребер графа.

Выходом данной программы должно быть время работы T_A алгоритма А и время работы T_B алгоритма Б в секундах.

Алгоритмы

А: алгоритм Борувки

Б: алгоритм Краскала

Данные для эксперимента

$n = 10^4$

а) $m = 10^5, \dots, 10^7$ с шагом 10^5

б) $m = 10^3, \dots, 10^5$ с шагом 10^3

$r = 10^6$

Нарисовать графики функций $T_A(m)$ и $T_B(m)$ для обоих случаев.

Граф

```
internal partial struct Graph
{
    private AdjacencyMatrix matrix;
    public int VertexCount { get; private set; }
    public int EdgeCount { get; private set; }


    private Graph(int vertices, int edges)
    {
        matrix = new AdjacencyMatrix(vertices);
        VertexCount = vertices;
        EdgeCount = edges;
    }

    public static Graph GenerateTree(int vertices, int maxWeight)
    {
        var graph = new Graph(vertices, vertices - 1);

        int[] vertexSequence = Enumerable.Range(0, vertices)
                                         .OrderBy(n => RNG.Random())
                                         .ToArray();

        for (int i = 1; i < vertices; i++)
        {
            int a = vertexSequence[i - 1];
            int b = vertexSequence[i];
            int weight = RNG.Random(1, maxWeight);
            graph.Connect(a, b, weight);
        }
```

```
    return graph;
}

public static Graph Generate(int vertices, int edges, int maxWeight)
{
    Debug.Assert(edges >= vertices - 1);
    Debug.Assert(edges <= vertices * (vertices - 1) / 2);

    var graph = GenerateTree(vertices, maxWeight);
    edges = edges - graph.EdgeCount;

    for (int i = 0; i < edges; i++)
    {
        int weight = RNG.Random(1, maxWeight);

        int vertexA, vertexB;
        do
        {
            vertexA = RNG.Random(0, vertices - 1);
            vertexB = RNG.Random(0, vertices - 1);
        } while (vertexA == vertexB || graph.HasEdge(vertexA, vertexB));

        graph.Connect(vertexA, vertexB, weight);
    }

    return graph;
}

public Edge[] Edges
{
    get {
        var edges = new Edge[EdgeCount];
        int k = 0;
        for (int i = 0; i < VertexCount - 1; i++)
        {
            for (int j = i + 1; j < VertexCount; j++)
            {
                if (!HasEdge(i, j)) continue;

                edges[k] = new Edge(i, j, matrix[i, j]);
                k++;
            }
        }

        return edges;
    }
}

public bool HasEdge(int a, int b)
{
    return matrix[a, b] > 0;
}

private bool IsConnected(int i)
{
    return matrix.IsConnected(i);
}
```

```
    private void Connect(int a, int b, int weight)
    {
        Debug.Assert(a != b);
        Debug.Assert(weight > 0);

        EdgeCount++;
        matrix[a, b] = weight;
    }
}
```

Алгоритм Борувки

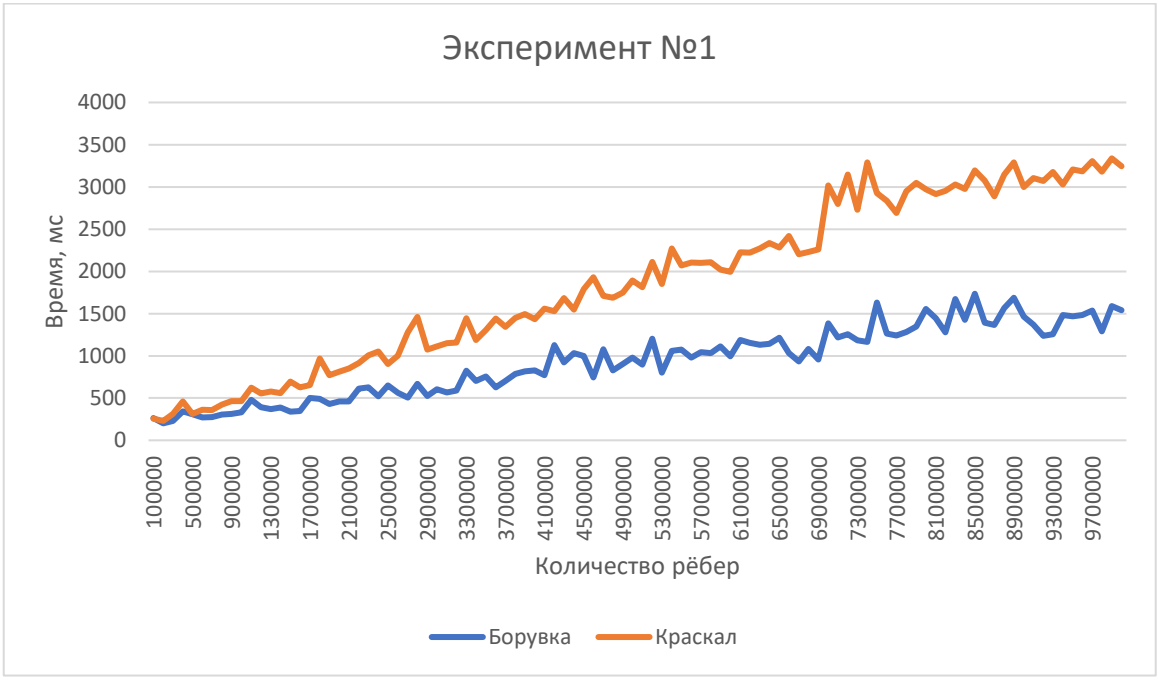
```
public static Edge[] Boruvka(Graph graph)
{
    var mst = new Edge[graph.VertexCount - 1];
    var set = new DisjointSet(graph.VertexCount);
    var cheapest = new int[graph.VertexCount];

    for (int v = 0; v < graph.VertexCount; v++)
        set.MakeSet(v);

    var edges = graph.Edges;

    int treeCount = graph.VertexCount;
    int mstIndex = 0;
    while (treeCount > 1)
    {
        for (int v = 0; v < graph.VertexCount; v++)
            cheapest[v] = -1;

        for (int e = 0; e < edges.Length; e++)
        {
            int i = set.Find(edges[e].U);
            int j = set.Find(edges[e].V);
            if (i == j) continue;

            if (cheapest[i] == -1 || edges[cheapest[i]].Weight > edges[e].Weight)
                cheapest[i] = e;

            if (cheapest[j] == -1 || edges[cheapest[j]].Weight > edges[e].Weight)
                cheapest[j] = e;
        }

        for (int v = 0; v < graph.VertexCount; v++)
        {
            if (graph.IsConnected(v) && cheapest[v] != -1)
            {
                int i = set.Find(edges[cheapest[v]].U);
                int j = set.Find(edges[cheapest[v]].V);
                if (i == j) continue;

                mst[mstIndex] = edges[cheapest[v]];
                set.Union(i, j);
                mstIndex++;
                treeCount--;
            }
        }
    }

    return mst;
}
```

Алгоритм Краскала

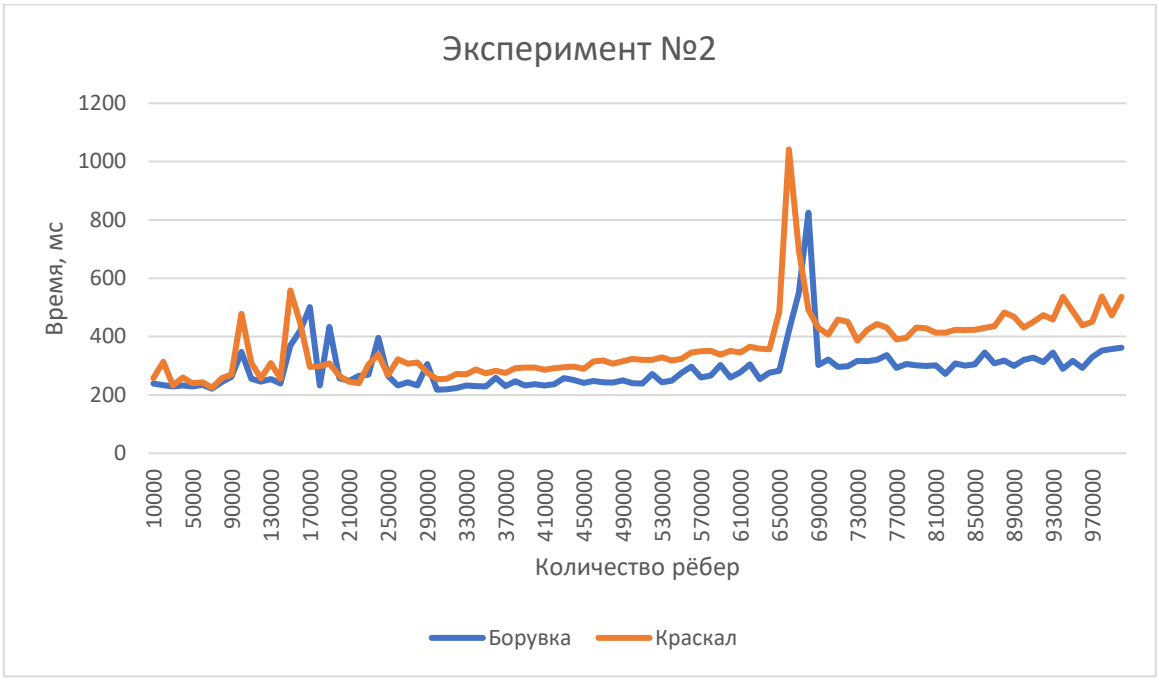
```
public static Edge[] Kruskal(Graph graph)
{
    var mst = new Edge[graph.VertexCount - 1];
    var set = new DisjointSet(graph.VertexCount);

    for (int v = 0; v < graph.VertexCount; v++)
        set.MakeSet(v);

    var edges = graph.Edges;
    var comparer = new EdgeComparer();
    Array.Sort(edges, comparer);

    int mstIndex = 0;
    for (int i = 0; i < edges.Length; i++)
    {
        if (set.Find(edges[i].U) == set.Find(edges[i].V)) continue;

        mst[mstIndex] = edges[i];
        set.Union(edges[i].U, edges[i].V);
        mstIndex++;
    }

    return mst;
}
```

Эксперименты

```
Генерация графов:  02:31
Алгоритм Борувки:  01:37
Алгоритм Краскала: 03:08
```



```
Генерация графов:  00:40
Алгоритм Борувки:  00:29
Алгоритм Краскала: 00:35
```



Вывод: алгоритм Борувки имеет сложность O(m · log n), а алгоритм Краскала - O((m + n) · log n), поскольку последний требует дополнительные затраты на сортировку массива рёбер. Таким образом, алгоритм Краскала приемлем только для относительно небольших графов.