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

Кафедра информатики и прикладной математики

Дисциплина «Алгоритмы и структуры данных»
Лабораторная работа №2

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 Дейкстры, реализованный на основе 7–кучи

Б: алгоритм Форда-Беллмана;

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
 {
     public const int INFINITY = 0x3f3f3f3f;

     public int[,] Matrix;
     public int VertexCount => Matrix.GetLength(dimension: 0);

     private int[] dist;
     private short[] prev;

     private Graph(int[,] matrix)
     {
         Matrix = matrix;
         dist = new int[matrix.GetLength(0)];
         prev = new short[matrix.GetLength(0)];
     }

     public static Graph Generate(int vertices, int edges, int maxWeight)
     {
         var matrix = new int[vertices, vertices];
         var graph = new Graph(matrix);

         for (int i = 0; i < vertices; i++)
         {
             graph.SetConnectionStatus(i, false);
         }

         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.Matrix[vertexA, vertexB] = weight;
            graph.SetConnectionStatus(vertexA, true);
        }

        return graph;
    }

    public IEnumerable<int> NeighborsOf(int source)
    {
        if (IsDisconnected(source)) yield break;

        for (int i = 0; i < VertexCount; i++)
        {
            if (i == source) continue;

            int weight = Matrix[source, i];
            if (weight > 0) yield return i;
        }
    }

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

    public bool IsDisconnected(int i)
    {
        return Matrix[i, i] == 0;
    }

    private void SetConnectionStatus(int i, bool status)
    {
        Matrix[i, i] = status ? 1 : 0;
    }
}
```

Алгоритм Дейкстры

```
public static PathResult Dijkstra(Graph graph, int source)
{
    int[]   dist = graph.dist;
    short[] prev = graph.prev;

    for (short i = 0; i < graph.VertexCount; i++)
    {
        dist[i] = INFINITY;
        prev[i] = -1;
    }

    var heap = new DHeap(graph.VertexCount);
    heap.Key[source] = 0;
    heap.Heapify();
```

```
    while (!heap.IsEmpty)
    {
        heap.DeleteMin();
        short i = heap.Name0;
        dist[i] = heap.Key0;

        foreach (int j in graph.NeighborsOf(i))
        {
            int weight = graph.Matrix[i, j];
            if (dist[j] >= INFINITY)
            {
                int jq = heap.Index[j];
                if (heap.Key[jq] > dist[i] + weight)
                {
                    heap.Key[jq] = dist[i] + weight;
                    heap.SiftUp(jq);
                    prev[j] = i;
                }
            }
        }
    }

    return Tuple.Create(dist, prev);
}
```

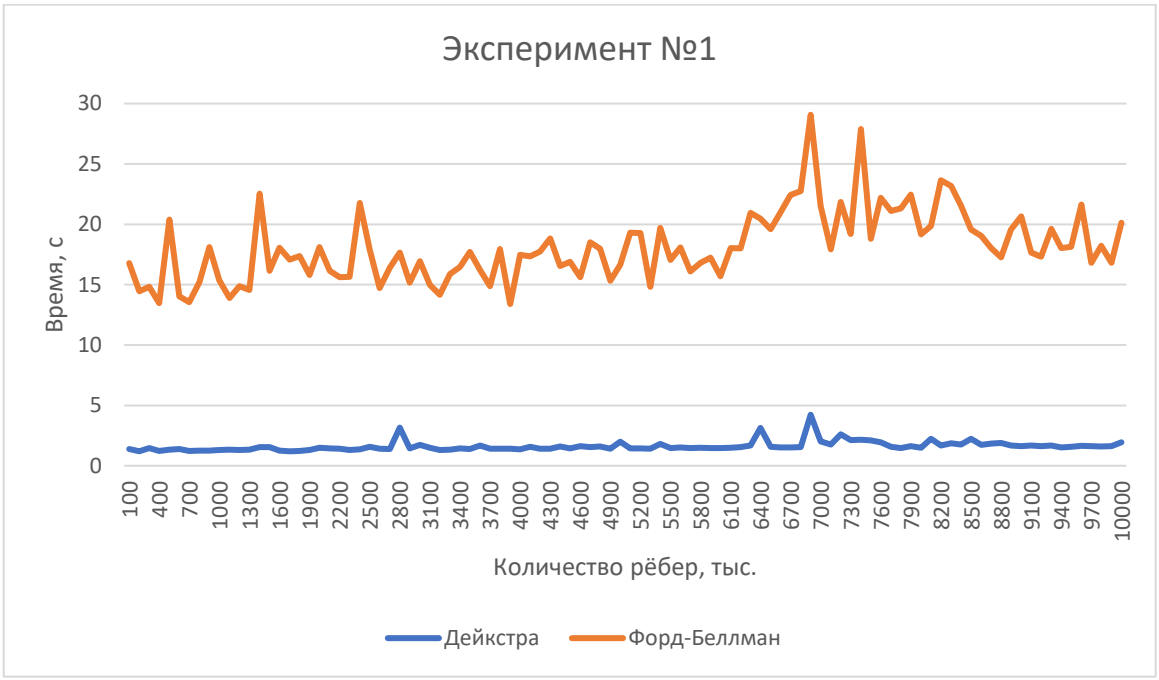
Алгоритм Форда-Беллмана

```
public static PathResult FordBellman(Graph graph, int source)
{
    int[] dist = graph.dist;
    short[] prev = graph.prev;

    for (short i = 0; i < graph.VertexCount; i++)
    {
        dist[i] = INFINITY;
        prev[i] = -1;
    }

    dist[source] = 0;

    bool shouldRetry;
    do
    {
        shouldRetry = false;
        for (short i = 0; i < graph.VertexCount; i++)
        {
            foreach (int j in graph.NeighborsOf(i))
            {
                int weight = graph.Matrix[i, j];

                if (j != source && dist[j] > dist[i] + weight)
                {
                    dist[j] = dist[i] + weight;
                    prev[j] = i;
                    shouldRetry = true;
                }
            }
        }
    } while (shouldRetry);
```

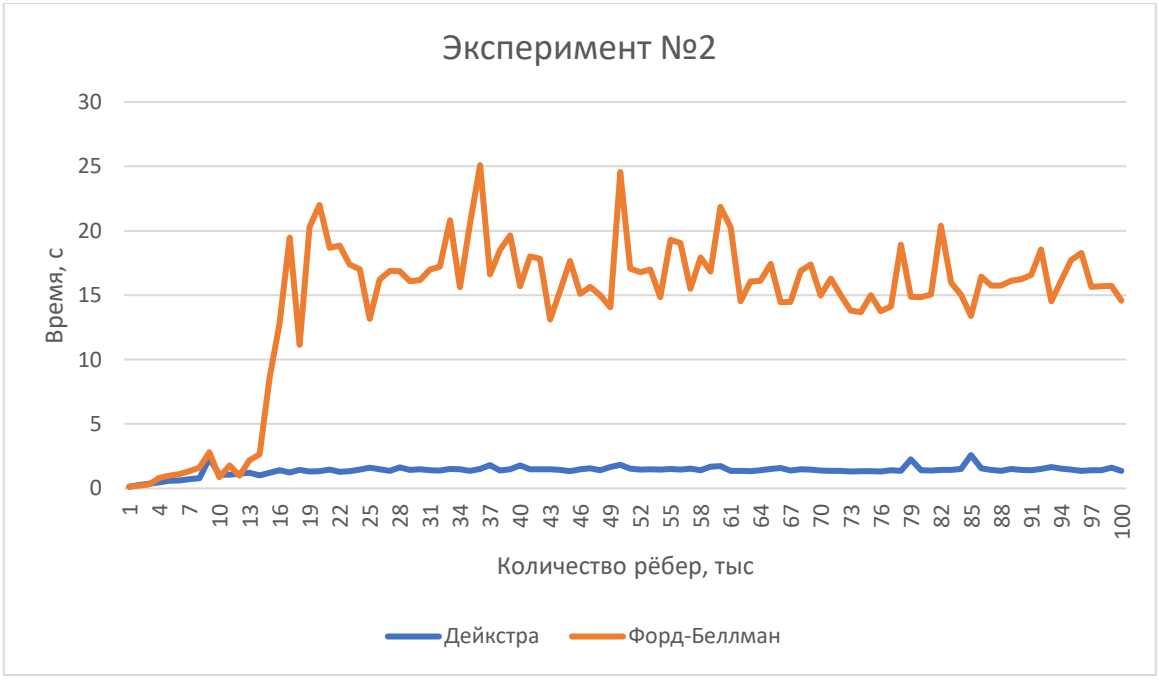
```
    return Tuple.Create(dist, prev);
}
```

Результат экспериментов

```
Генерация графов:  02:15
Алгоритм Дейкстры:  02:41
Алгоритм Форда-Беллмана: 30:09
```




```
Генерация графов:  00:08
Алгоритм Дейкстры:  02:18
Алгоритм Форда-Беллмана: 23:38
```



Вывод: в данном случае алгоритм Дейкстры оказался гораздо более эффективным, так как, во-первых, он был ускорен с использованием 7-кучи, а во-вторых, его время работы в худшем случае O(M * log N) против O(M * N) у алгоритма Форда-Беллмана.

Преимущество алгоритма Форда-Беллмана в том, что он поддерживает рёбра с отрицательным весом.