МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Алгоритмы на графах. Вариант: 2

Студент гр. 0302 _____ Савенко Н.С

Преподаватель _____ Тутуева А.В.

Санкт-Петербург

2022

Постановка задачи

Дан список возможных авиарейсов в текстовом файле в формате:

Город отправления 1;Город прибытия 1;цена прямого перелета 1;цена обратного перелета 1

Город отправления 2;Город прибытия 2;цена перелета 2;цена обратного перелета 1

найти наиболее эффективный по стоимости перелет из города ***i*** в город ***j***.

Описание реализуемых алгоритмов

В решении используется алгоритм Беллмана-Форда

Дан ориентированный или неориентированный граф со взвешенными рёбрами. Длиной пути назовём сумму весов рёбер, входящих в этот путь.

Требуется найти кратчайшие пути от выделенной вершины до всех вершин графа.

Заметим, что кратчайших путей может не существовать. Так, в графе, содержащем цикл с отрицательным суммарным весом, существует сколь угодно короткий путь от одной вершины этого цикла до другой (каждый обход цикла уменьшает длину пути). Цикл, сумма весов рёбер которого отрицательна, называется *отрицательным циклом*.

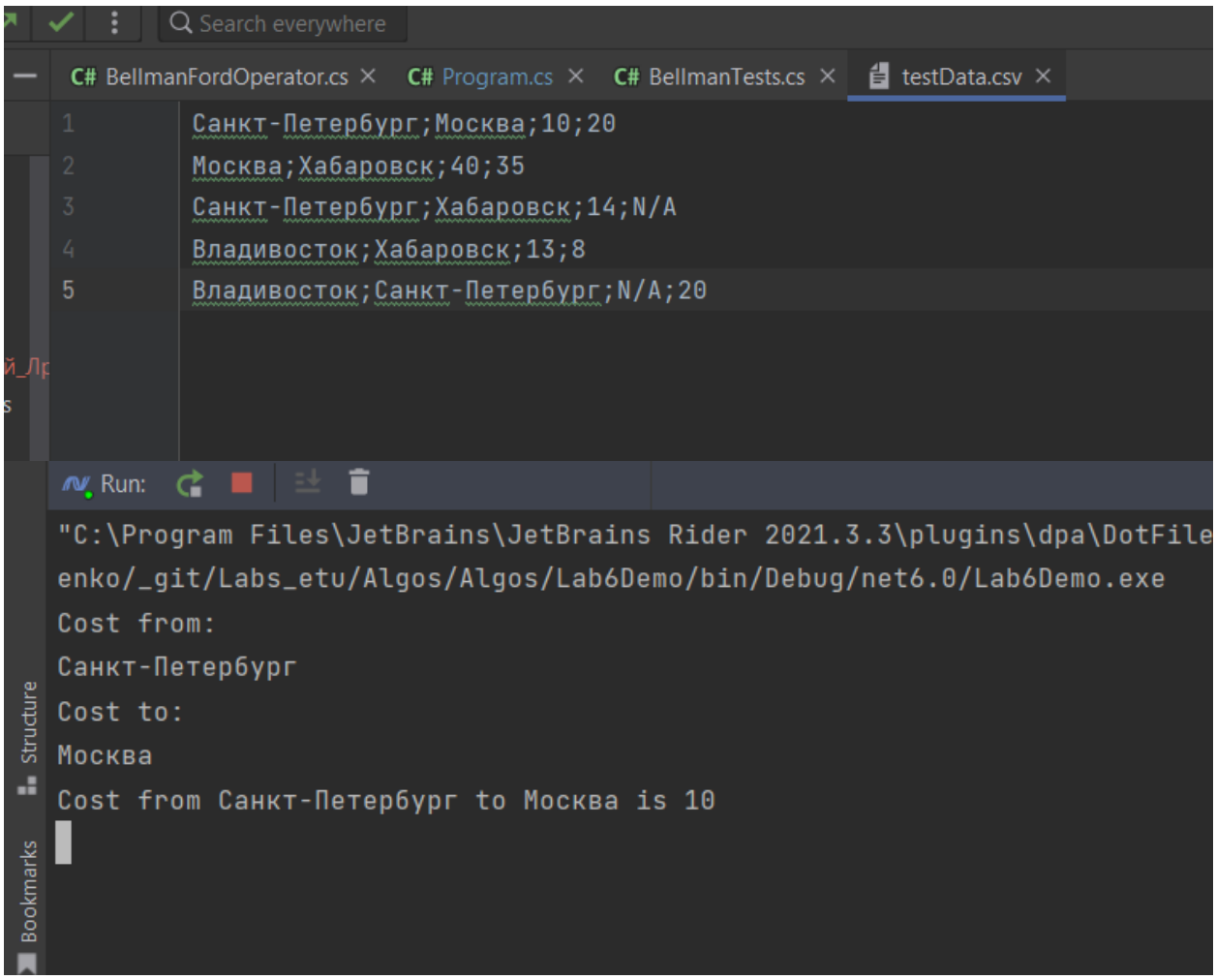
Оценка временной сложности

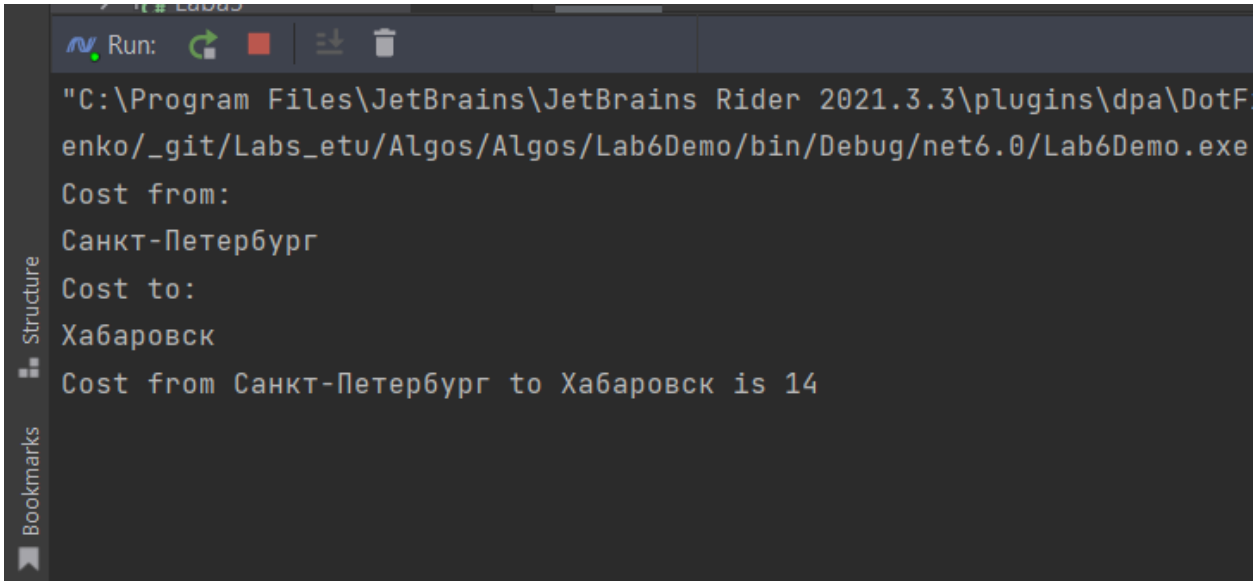
1. CalculateCost
   O(VE)

Описание Unit тестов

В unit тестах поводится проверка методов чтения из файла, парсинга ребер и расчета стоимости перелета.

Примеры работы





Листинг

```
namespace Lab6;

public class BellmanFordOperator
{
```

```
    public IList<Link> Links { get; init; }

    public BellmanFordOperator()
    {
        Links = new List<Link>();
    }

    public BellmanFordOperator(string filePath) : this()
    {
        LoadFromFile(filePath);
    }

    public void ParseLines(string[] lines)
    {
        foreach (var line in lines)
        {
            ParseLinks(line);
        }
    }

    public bool LoadFromFile(string filePath)
    {
        try
        {
            string[] lines = File.ReadAllLines(filePath);
            ParseLines(lines);
            return true;
        }
        catch (Exception e)
        {
            return false;
        }
    }

    public void ParseLinks(string source)
    {
        var fields = source.Split(';');
        if (int.TryParse(fields[2], out var depCost))
        {
            AddLink(new Link(fields[0], fields[1], depCost));
        }
        if (int.TryParse(fields[3], out var destCost))
        {
            AddLink(new Link(fields[1], fields[0], destCost));
        }
    }

    public void AddLink(Link link)
    {
        Links.Add(link);
    }

    public int CalculateCost(string from, string to)
    {
        var vertices = GetVertices();
```

```
        var dist = vertices.ToDictionary(verticle => verticle, verticle =>
int.MaxValue);
        dist[from] = 0;

        for (var i = 0; i < vertices.Count - 1; i++)
        {
            foreach (var link in Links)
            {
                if (dist[link.from] != int.MaxValue && dist[link.to] >
dist[link.from] + link.cost)
                {
                    dist[link.to] = dist[link.from] + link.cost;
                }
            }
        }

        return dist[to];
    }

    public List<string> GetVertices()
    {
        var verticles = new List<string>();

        foreach (var link in Links)
        {
            if (!verticles.Contains(link.from))
            {
                verticles.Add(link.from);
            }
            if (!verticles.Contains(link.to))
            {
                verticles.Add(link.to);
            }
        }

        return verticles;
    }
}

namespace Lab6;

public record Link(string from, string to, int cost);
```