

федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Санкт-Петербургский государственный технологический институт  
(технический университет)»  
СПбГТИ(ТУ)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | | |
| Направление подготовки | | 09.03.03 | Прикладная информатика в химии | | | |
| Направленность (профиль) | |  | САПР | | | |
|  | |  |  | | | |
| Факультет | |  | Информационных технологий и управления | | | |
| Кафедра | |  | Систем автоматизированного  проектирования и управления | | | |
| Учебная дисциплина | |  | Методы поддержки принятия решений | | | |
|  |  | | |  |  |  |

Курс \_\_\_3\_\_\_ Группа 475

**Лабораторная работа №2**

**Динамическое программирование**

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Андрианова К.И.\_\_

(подпись, дата) (инициалы, фамилия)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Овчинников Р.С.

(подпись, дата) (инициалы, фамилия)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пекер В.А.

(подпись, дата) (инициалы, фамилия)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уланов В.Н.

(подпись, дата) (инициалы, фамилия)

Санкт-Петербург

2019

# Цель работы

Целью данной работы является создание программы, которая осуществляет решение динамического программирования с помощью метода Беллмана. Предлагается практическое решение задачи динамического программирования для закрепления теоретических знаний. Для удобства использования программы разработан эргономичный интерфейс.

**План выполнения**

1. Знакомство с динамическим программированием
2. Знакомство с методом Беллмана
3. Блок-схема программы
4. Разработка интерфейса
5. Тестирование
6. Анализ тестирования
7. Вывод

**Ход выполнения работы**

1. Знакомство с динамическим программированием

*Динамическое программирование* — метод решения задачи путём её разбиения на несколько одинаковых подзадач, рекуррентно связанных между собой. Самым простым примером будут [числа Фибоначчи](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B0_%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8) — чтобы вычислить некоторое число в этой последовательности, нам нужно сперва вычислить третье число, сложив первые два, затем четвёртое таким же образом на основе второго и третьего, и так далее.

Методы решения динамического программирования

1) Метод прямого хода

2) Метод обратного хода

Процесс нахождения решения разбивается на 2 стадии условную и безусловную оптимизацию.

На этапе условной оптимизации для каждого шага определяется условное оптимальное управление и условный оптимальный выигрыш, на всех шагах начиная с данного и до последнего включительно, в методе обратной прогонки, или с 1го до данного включительно в методе прямой прогонки.

На этапе безусловной оптимизации для каждого шага находится безусловное оптимальное управление и max значение W. Безусловная оптимизация в методе обратной прогонки идет от 1го шага к последнему.

В методе прямой прогонки наоборот условная от 1 к последнему, безусловная от последнего к первому.

3) Метод обратной прогонки

При нахождении управления на каждом шаге нельзя исходить из интересов шага в отдельности. Необходимо исходить из всего процесса в целом.

Принцип оптимальности Беллмана:

Какого бы ни было состояние системы к очередному шагу, управление на этом шаге нужно выбирать так, чтобы сумма выигрыша на этом шаге и оптимальна на всех последующих шагах была max.

2.Знакомство с методом Беллмана

Метод динамического программирования состоит в том, что оптимальное управление строится постепенно. На каждом шаге оптимизируется управление только этого шага. Вместе с тем на каждом шаге управление выбирается с учетом последствий, так как управление, оптимизирующее целевую функцию только для данного шага, может привести к неоптимальному эффекту всего процесса. Управление на каждом шаге должно быть оптимальным с точки зрения процесса в целом.

Каково бы ни было начальное состояние системы перед очередным шагом, управление на этом этапе выбирается так, чтобы выигрыш на данном шаге плюс оптимальный выигрыш на всех последующих шагах был максимальным.

Так, если система в начале *k*-го шага находится в состоянии  и мы выбираем произвольное управление , то она придет в новое состоя-

Ние , и последующие управления  должны выбираться оптимальными относительно состояния .

Последнее означает, что при этих управлениях максимизируется величина , т. е. показатель эффективности на последующих до конца процесса шагах . Обозначим через .

Выбрав оптимальное управление на оставшихся  шагах, получим величину , которая зависит только от , т. е.

.

Назовем величину  **условным максимумом**. Если мы теперь выберем на *k*-м шаге некоторое произвольное управление , то система придет в состояние  (рис. 1). Согласно принципу оптимальности, необходимо выбирать управление  так, чтобы оно в совокупности с оптимальным управлением на последующих шагах (начиная с -го) приводило бы к общему показателю эффективности на  шагах, начиная с *k*-го и до конца. Это положение в аналитической форме можно записать в виде следующего соотношения:



,

получившего название **основного функционального уравнения** динамического программирования, или **основного рекуррентного уравнения Беллмана**.

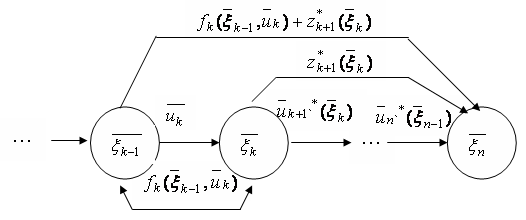


Рисунок 1 – Граф-схема уравнения Беллмана

Из уравнения может быть получена функция , если известна функция . Аналогично можно получить, если найдена  и т. д., пока не будет определена величина , представляющая по определению максимальное значение показателя эффективности процесса в целом:

.

Решая уравнение для определения условного максимума показателя эффективности за  шагов, начиная с *k*-го, мы определяем соответствующее оптимальное управление , при котором этот максимум достигается. Это управление также зависит от ; будем обозначать его через  и называть **условным оптимальным управлением на *k*-м шаге**. Основное значение уравнения в котором реализована идея динамического программирования, заключается в том, что решение исходной задачи определения максимума функции (16.3) *n* переменных  сводится к решению последовательности *n* задач, задаваемых соотношениями (16.4), каждое из которых является задачей максимизации функции одной переменной .

В результате последовательного решения *n* частных задач на условный максимум определяют две последовательности функций: – условные максимумы и соответствующие им  – условные оптимальные управления. Указанные последовательности функций в дискретных задачах получают в табличной форме, а в непрерывных моделях – аналитически. После выполнения первого этапа (условной оптимизациии) приступают ко второму этапу – безусловной оптимизации.

Если начальное состояние  задано , то непосредственно определяют максимум целевой функции

,

а затем – искомое безусловное оптимальное управление по цепочке

.     (16.5)

Если задано множество  начальных состояний , то дополнительно решают еще одну задачу на максимум

,

откуда находят , а затем по цепочке– безусловное оптимальное управление.

В рассмотренных рекуррентных соотношениях предписывают начинать вычисления с последнего этапа и затем передвигаться назад до этапа 1. Такой метод вычислений известен как **алгоритм обратной прогонки**. Если расчеты осуществляются в естественном порядке следования этапов, то такой метод вычислений известен как **алгоритм прямой прогонки**.

Приведем рекуррентные соотношения для этого случая. Уравнения состояний для прямого хода удобно записывать в виде



Введем в рассмотрение условные максимумы показателя эффективности за

*k* шагов, от 1-го до *k*-го включительно, – величину . Повторив приведенные рассуждения, придем к следующей системе уравнений Беллмана:

;

.

В результате решения этих уравнений получим последовательности

.

Далее определим безусловное оптимальное управление по цепочке

.

**Описание алгоритма**

Мы считаем, что граф не содержит цикла отрицательного веса. Случай наличия отрицательного цикла будет рассмотрен ниже в отдельном разделе.

Заведём массив расстояний d[0 \ldots n-1], который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: d[v] = 0, а все остальные элементы d[] равны бесконечности \infty.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию (relax, ослабление) вдоль каждого ребра (a,b) стоимости c. Релаксация вдоль ребра — это попытка улучшить значение d[b] значением d[a] + c. Фактически это значит, что мы пытаемся улучшить ответ для вершины b, пользуясь ребром (a,b) и текущим ответом для вершины a.

Утверждается, что достаточно n-1 фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе (повторимся, мы считаем, что циклы отрицательного веса отсутствуют). Для недостижимых вершин расстояние d[] останется равным бесконечности \infty.

**Блок-схема алгоритма**

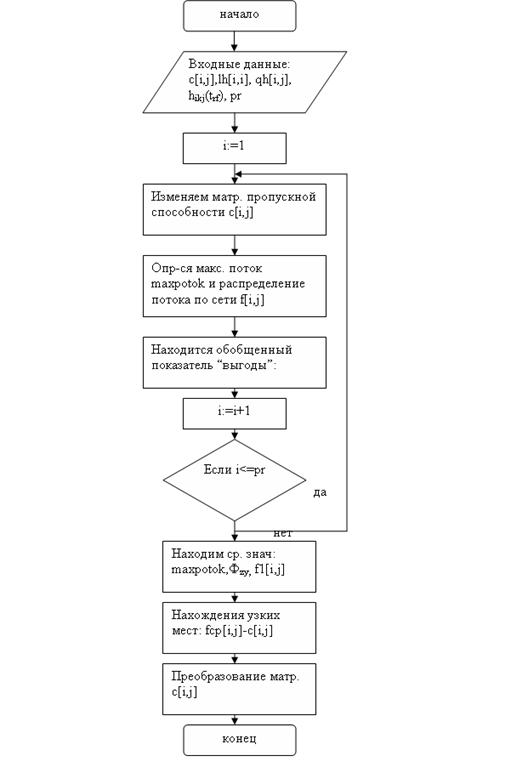
****

Рисунок 2- Блок-схема метода Беллмана

**Выбор инструментария**

Программа написана в Visual Studio 2017 на языке C#.

**Интерфейс**

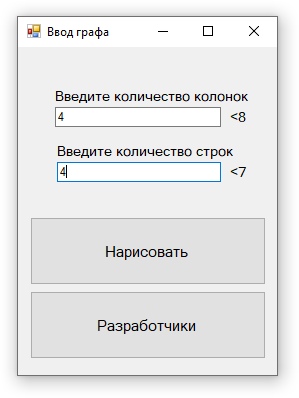
****

Рисунок 3- Интерфейс программы

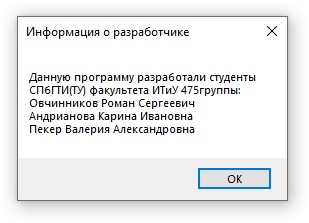
****

Рисунок 4- Информация о разработчиках

**Блок-схемы программы**

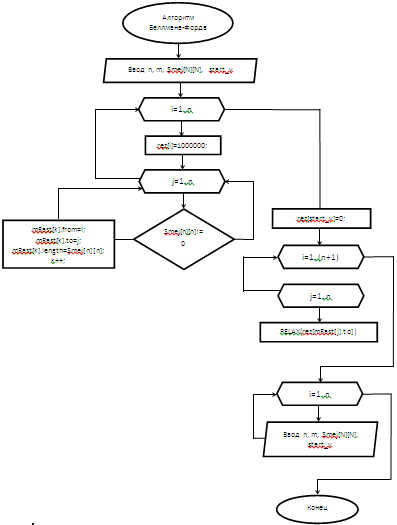


Рисунок 5 – Блок-схема программы

**Тестирование**

Сначала мы заполняем вводим количество столбцов и строк (рис.6). Затем на экран выводится граф с построенной таблицей путей (рис.7). Далее заполняем в таблице заполняем расстояние от одной вершины к другой. После нажимаем кнопку старта и на экран выводится оптимальный маршрут (рис.8). Также в итоговом графе пишется расстояние от истока к вершине.

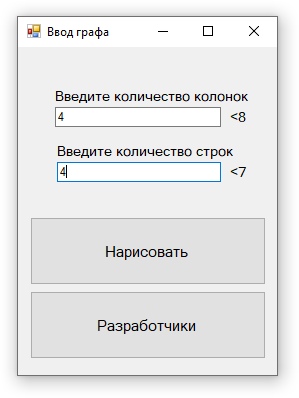


Рисунок 6-Ввод количества столбцов и строк

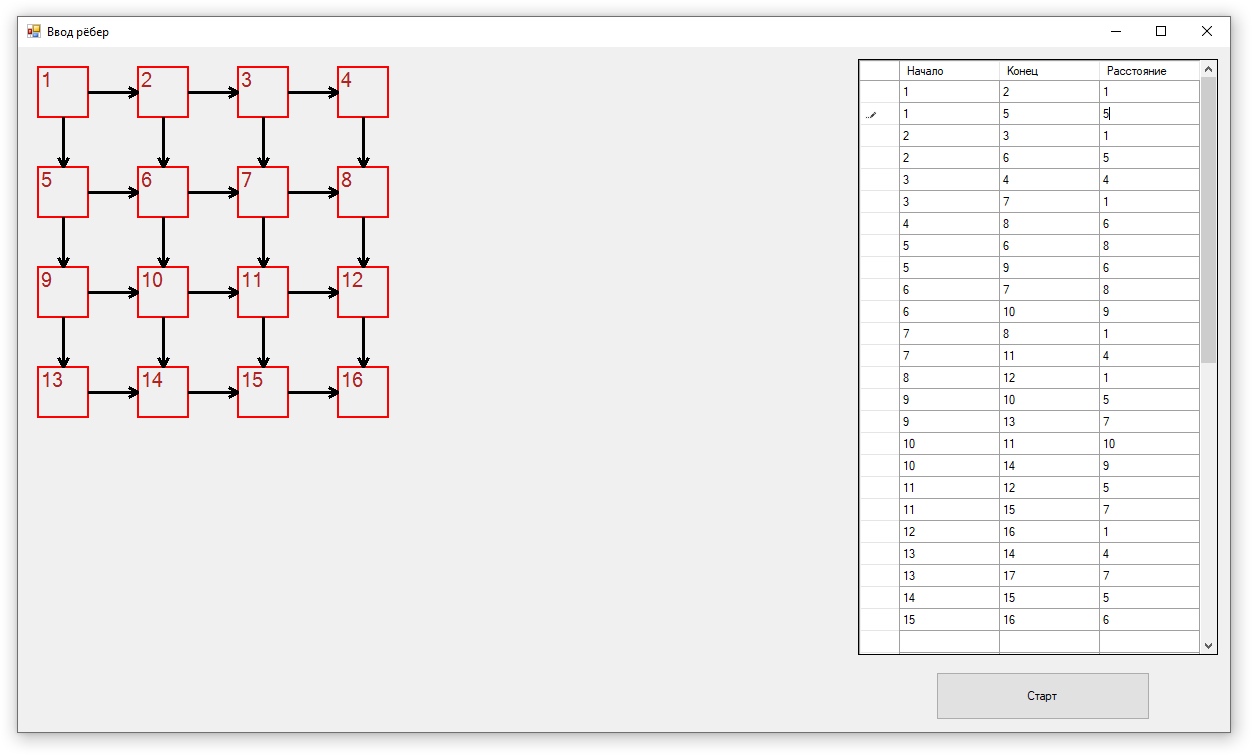


Рисунок 7-Граф с построенной таблицей весов

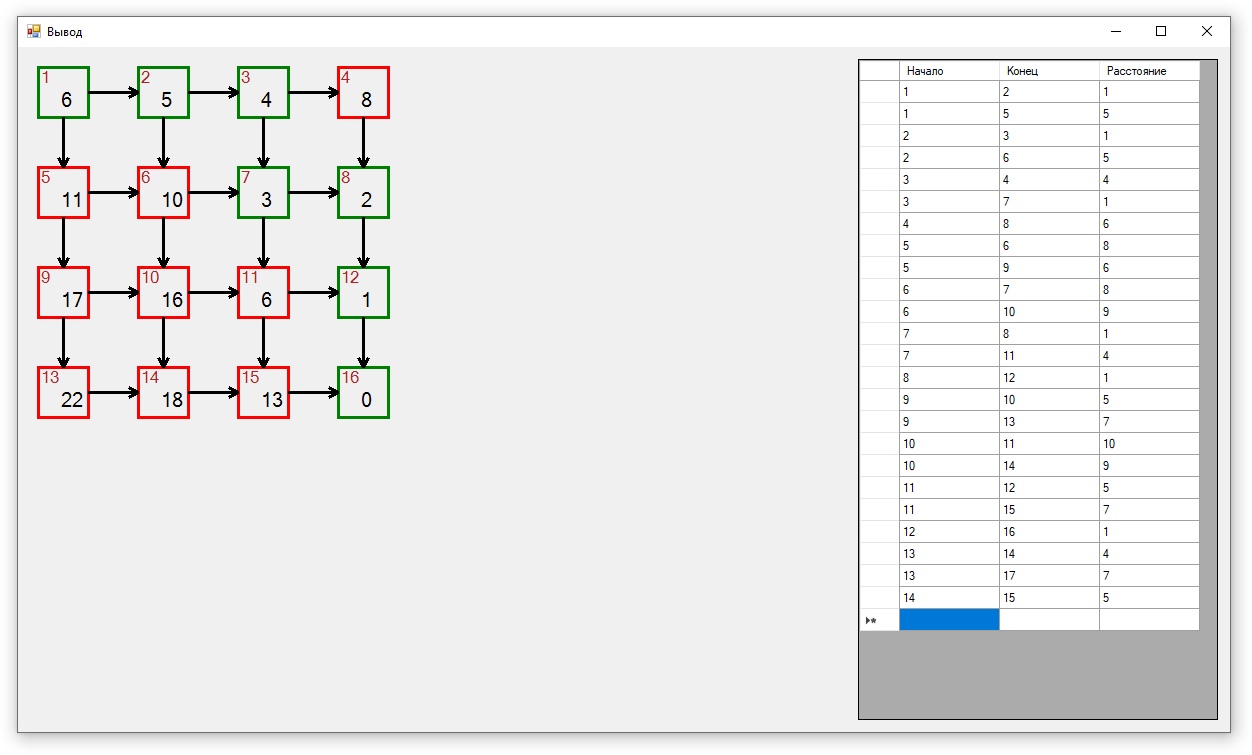


Рисунок 8- Граф с оптимальным маршрутом

Далее мы проводим проверку алгоритма в случае если в поле ввели не число, а букву (рис.9).

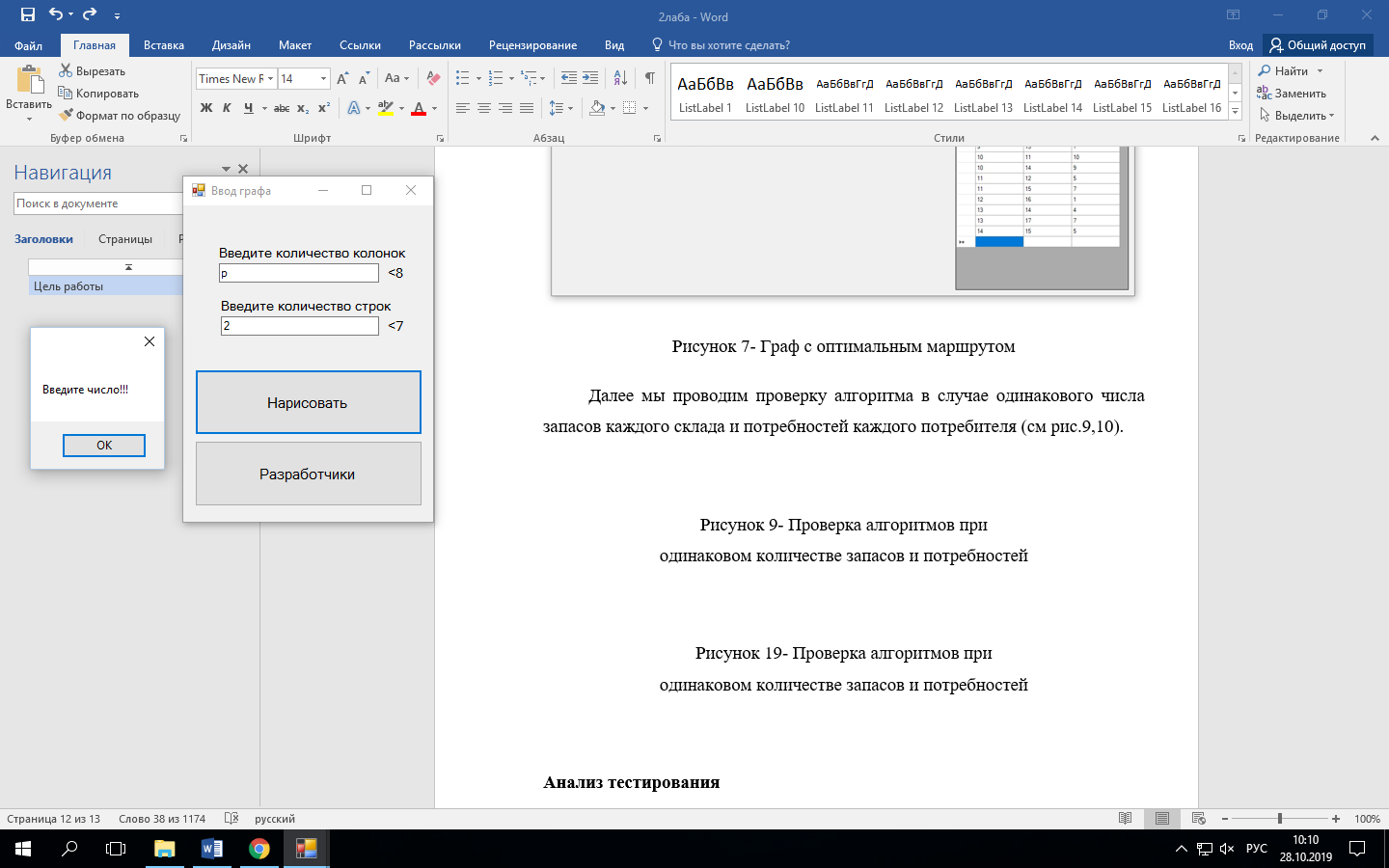
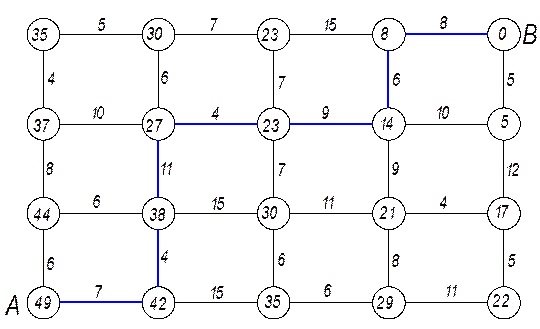


Рисунок 9- Проверка корректности ввода

Далее мы проводим проверку работы нашей программы на основе метода Беллмана, и сравниваем с результатом работы по данному методу из книги (рис.10 и рис.11). Для этого мы построили точно такой же граф, и убедились, что наша программа работает корректно, так как результаты совпали.

 Рисунок 10- Результат работы метода из книги

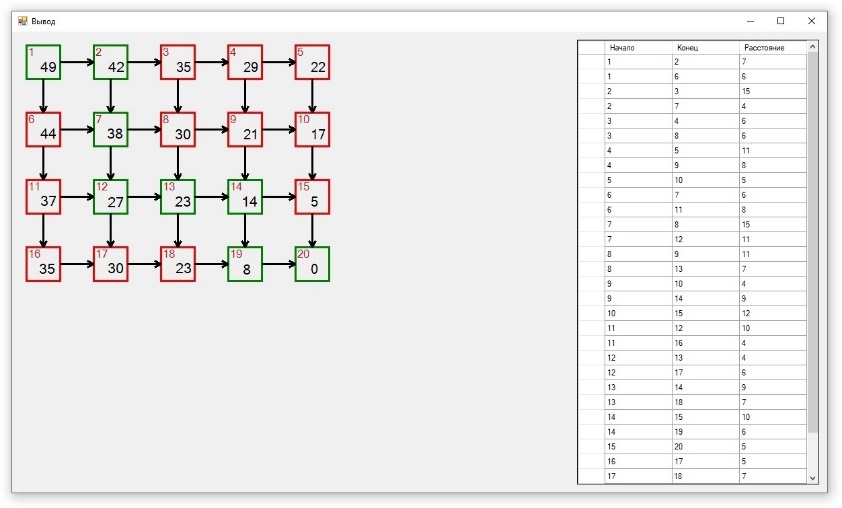


Рисунок 11- Проверка корректности работы программы с методом Беллмана из книги

**Анализ тестирования**

В ходе тестирования мы убедились, что написанная нами программа работает. Сравнивая, работу и результат нашей программы и результат из книги, мы убедились, что наша программа работает исправно. Интерфейс предложенный нами эргономичен и удобен для использования.

**Код программы (алгоритм)**

static class Program

{

sealed public class Node

{

public List<Link> Links { get; set; }

public Node()

{

Links = new List<Link>();

}

}

sealed public class Link

{

public double Distance { get; set; }

public string Node { get; set; }

}

sealed public class Description

{

public bool Visited { get; set; }

public double Distance { get; set; }

public Description()

{

Visited = false;

Distance = double.PositiveInfinity;

}

}

sealed public class Graph

{

private Dictionary<string, Node> nodes;

public Graph()

{

nodes = new Dictionary<string, Node>();

}

public void AddNode(string name)

{

nodes.Add(name, new Node());

}

public void AddLinkToNode(string start, string end, double distance, bool isOriented)

{

nodes[start].Links.Add(new Link { Node = end, Distance = distance });

if (!isOriented)

nodes[end].Links.Add(new Link { Node = start, Distance = distance });

}

public double FindShortestDistance(string start, string end)

{

// Алгоритм.

Dictionary<string, Description> info = new Dictionary<string, Description>(this.nodes.Count);

foreach (string current in this.nodes.Keys)

info.Add(current, new Description());

info[start].Distance = 0;

// Пока все вершины непосещенные.

while (!info.Select(x => x.Value.Visited).Aggregate((x, y) => x & y))

{

bool yes = true;

// Находим непосещенную вершину с минимальной меткой.

string current = info.Where(x => !x.Value.Visited && x.Value.Distance == info.Where(y => !y.Value.Visited).Min(y => y.Value.Distance)).First().Key;

// Находим все непосещенные соседние вершины для текущей вершины.

List<Link> neighbors = nodes[current].Links.Where(x => !info[x.Node].Visited).ToList();

// Рассматриваем новую длину пути для каждой соседней вершины.

foreach (Link link in neighbors)

{

double distance = info[current].Distance + link.Distance;

if (info[link.Node].Distance > distance)

{

info[link.Node].Distance = distance;

}

}

path.Add(current);

// Отмечаем текущую вершину как посещенная.

info[current].Visited = true;

}

return info[end].Distance;

}

}

**Вывод**

В ходе лабораторной работы мы изучили динамическое программирование и методы его реализации. Одним из методов является метод Беллмана. Написали программу с помощью данного метода, который находит кратчайший путь от истока к стоку. Используемый метод является эффективным.