МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ОТЧЁТ**

**ПО ИНДИВИДУАЛЬНОЙ РАБОТЕ №1**

**ДИСЦИПЛИНА: «КОНСТРУИРОВАНИЕ АЛГОРИМОВ И СТРУКТУР ДАННЫХ»**

Работу выполнил\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Л.Е.Лозовик

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Компьютерные науки

Преподаватель

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Климец

Краснодар

2020

Оглавление

[Математическая постановка задачи 3](#_Toc41835708)

[Описание алгоритмов решения 5](#_Toc41835709)

[Техническое описание программного продукта 7](#_Toc41835710)

[UML диаграмма программы 11](#_Toc41835711)

[Инструкция по эксплуатации 12](#_Toc41835712)

[Графы для тестирования 14](#_Toc41835713)

# Математическая постановка задачи

Программный продукт предназначен для решения следующих задач:

1. *Дан смешанный граф. Дано натуральное число n. Найти количество путей длины n.*

Смешаннымназывают граф, в котором имеются рёбра хотя бы двух из упомянутых трёх разновидностей (звенья, дуги, петли).

1. *Дан неориентированный граф. Построить произвольное максимальное независимое множество вершин графа.*

Графы, в которых все рёбра являются [звеньями](https://function-x.ru/graphs5.html) (порядок двух концов ребра графа не существенен), называются неориентированными.

Множество вершин графа называется независимым, если никакие две вершины этого множества не соединены ребром.

1. *Дан ориентированный слабосвязный граф. Построить топологическую сортировку вершин этого графа.*

Графы, в которых все рёбра являются дугами (порядок двух концов ребра графа существенен), называются ориентированными графами или орграфами.

Орграф называется слабо связным, или слабым, если для любых двух различных вершин графа существует по крайней мере один маршрут, соединяющий их.

Топологическая сортировка (Topological sort) — один из основных алгоритмов на графах, который применяется для решения множества более сложных задач. Задача топологической сортировки графа состоит в следующем: указать такой линейный порядок на его вершинах, чтобы любое ребро вело от вершины с меньшим номером к вершине с большим номером. Очевидно, что если в графе есть циклы, то такого порядка не существует.

1. *Дан произвольный неориентированный граф, проверить, будет ли он деревом.*

Дерево – это граф, между любыми двумя вершинами которого существует ровно один путь.

1. *Реализовать алгоритм Беллмана – Форда нахождения кратчайшего пути. Описать возможности его применения.*

За время O(|V|\*|E|) алгоритм находит кратчайшие пути от одной вершины графа до всех остальных. В отличие от алгоритма Дейкстры, алгоритм Беллмана — Форда допускает рёбра с отрицательным весом.

# Описание алгоритмов решения

*Задача №1*: выполняется рекурсивный обход по всем смежным вершинам создавая путь. Когда путь достигает заданной длинны n счетчик количества инкрементируется. *\*При создании пути его ребра записываются в стек и за счет этого наличие в пути двух одинаковых ребер исключается.*

*Задача №2*: на первом этапе выбирается 1 вершина графа и заносится в стек. На втором этапе цикл проходит по остальным вершинам и добавляет в стек те, что не смежны ни с одной вершиной в стеке. После обработки всех вершин длинна получившегося стека сравнивается с результирующим и если результирующий стек оказался меньше, то он присваивает значения получившегося. Далее снова переходим к 1 этапу, но выбираем следующую вершину и так пока все вершины не будут перебраны.

*Задача №3*: поиск в глубину или обход в глубину (англ. Depth-first search, сокращенно DFS) — один из методов обхода графа. Алгоритм поиска описывается следующим образом: для каждой не пройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них. Запускаем обход в глубину, и когда вершина обработана, заносим ее в стек. По окончании обхода в глубину вершины достаются из стека. Новые номера присваиваются в порядке вытаскивания из стека. Во время обхода в глубину используется 3 цвета. Изначально все вершины белые. Когда вершина обнаружена, красим ее в серый цвет. Когда просмотрен список всех смежных с ней вершин, красим ее в черный цвет.

*Задача №4*: рекурсивно обходим граф добавляя пройденный вершины в стек. Если при обходе встречается вершина, которая уже есть в стеке, то этот граф не является деревом.

*Задача №5*: ниже приведены подробно расписанные шаги.  
Входные данные: Граф и начальная вершина src.  
Выходные данные: Кратчайшее расстояние до всех вершин от src. Если попадается цикл отрицательного веса, то самые короткие расстояния не вычисляются, выводится сообщение о наличии такого цикла.

1. На этом шаге инициализируются расстояния от исходной вершины до всех остальных вершин, как бесконечные, а расстояние до самого src принимается равным 0. Создается массив dist[] размера |V| со всеми значениями равными бесконечности, за исключением элемента dist[src], где src — исходная вершина.
2. Вторым шагом вычисляются самые короткие расстояния. Следующие шаги нужно выполнять |V|-1 раз, где |V| — число вершин в данном графе.

* Произведите следующее действие для каждого ребра u-v:  
  Если dist[v] > dist[u] + вес ребра uv, то обновите dist[v]  
  dist [v] = dist [u] + вес ребра uv

1. На этом шаге сообщается, присутствует ли в графе цикл отрицательного веса. Для каждого ребра u-v необходимо выполнить следующее:

* Если dist[v] > dist[u] + вес ребра uv, то в графе присутствует цикл отрицательного веса.

Идея шага 3 заключается в том, что шаг 2 гарантирует кратчайшее расстояние, если граф не содержит цикла отрицательного веса. Если мы снова переберем все ребра и получим более короткий путь для любой из вершин, это будет сигналом присутствия цикла отрицательного веса.

# Техническое описание программного продукта

Программа содержит 4 основных класса: класс Graph, Vertex, Edge и класс формы Form1.

Класс Graph содержит переменную имени, списки вершин и ребер, а также стеки вершин и ребер. Класс имеет 2 конструктора: первый инициализирует все списки по умолчанию и задает пустое имя, второй конструктор идентичен первому, но в нем можно указать имя графа.

Методы класса Graph:

void setupGraph(int[,] matrSm) – Инициализирует граф. Принимает на вход матрицу смежности вершин и заполняет списки вершин и ребер.

1. List<Vertex> getVertexList() и List<Edge> getEdgeList() – возвращают списки вершин и ребер соответственно.
2. void setName(string newName) – Устанавливает имя графа.
3. Vertex addVertex() – добавляет новую вершину в граф
4. Vertex addVertex(string newName) – добавляет новую вершину в граф и задает её имя.
5. Edge addEdge(string Type) – добавляет новое ребро в граф, задавая ему тип.
6. Edge addEdge(Vertex transmitter, Vertex receiver, int Weight, string Type) – добавляет новое ребро в граф, задавая ему вершины transmitter и receiver, а также вес и тип.
7. bool checkForRecCountWays(Vertex now, Vertex next) – вспомогательная функция для recCountWays. Проверяет начилие ребра в стеке ребер.
8. int recCountWays(Vertex now, int wayLengthNow) – функция поиска количества путей длинны N(Задача №1).
9. int countWays(Vertex start, int wayLength)– оболочка для recCountWays, которая принимает параметрами начальную вершину и длину пути и возвращает количество путей длинны N.
10. int countWays(int wayLength)– оболочка для recCountWays, которая принимает параметром длину пути и возвращает количество путей длинны N.
11. bool checkForMaximum\_independent\_set\_of\_vertexes(Vertex v List<Vertex> verList) – вспомогательная функция для List<Vertex> maximum\_independent\_set\_of\_vertexes(). Проверяет нет ли в переданном списке вершин смежных с данной.
12. List<Vertex> maximum\_independent\_set\_of\_vertexes() – функция, которая ищет максимальное независимое множество вершин(Задача №2).
13. int topology\_sort\_rec(Vertex now) – функция которая делает топологическую сортировку графа.
14. int topology\_sort() – оболочка для topology\_sort\_rec, которая сортирует список вершин графа.
15. bool recIsTree(Vertex now, Vertex pred) – функция, которая проверяет граф на соответствие дереву.
16. bool isTree() – оболочка для recIsTree.
17. List<int> BFShortWay(int start) – функция, которая реализует алгоритм Беллмана – Форда нахождения кратчайшего пути. На вход поступает стартовая вершина, а на выходе список расстояний от стартовой вершин до остальных.

Класс Edge содержит вершины Transmitter и Receiver, переменные тип, имя, id и вес. Класс имеет 2 конструктора: первый принимает id, вершину transmitter и receiver и задает тип “none”, второй конструктор идентичен, но в нем можно указать тип ребра.

Методы класса Edge:

1. int GetWeight() – возвращает вес ребра.
2. void changeName(string newName) и void changeType(string newType) – изменяют имя и тип ребра соответственно.
3. void SetTransmitter(Vertex newValue) и void SetReciever(Vertex newValue) – устанавливают transmitter и receiver соответственно.
4. Vertex GetTransmitter() и Vertex GetReciever() – возвращает transmitter и receiver соответственно.
5. int GetTransmitterId() и int GetRecieverId() – возвращает transmitter id и receiver id соответственно.
6. void EdgeSetup(Vertex value1, Vertex value2) – устанавливает transmitter и receiver.

Класс Vertex содержит список смежных вершин, переменные имя и id.

Есть 2 конструктора: первый в качестве аргумента принимает id, ставит пустое имя и инициализирует список смежных вершин. Второй конструктор принимает 2 аргумента – id и name, он работает аналогично первому, но еще устанавливает имя вершины.

Методы класса Vertex:

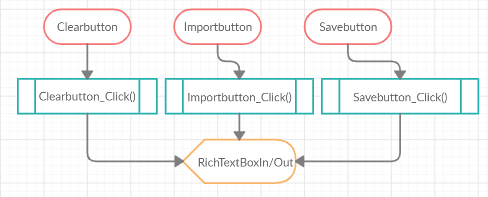
1. void addAdjacentVertex(ref Vertex newVertex) – добавляет вершину в список смежных вершин.
2. List<Vertex> getAdjacentVertex() – возвращает список смежных вершин.
3. void setVertexName(string newName) – устанавливает имя вершины.
4. string getVertexName() – возвращает имя вершины.
5. int getVertexId() - возвращает id вершины.
6. void setVertexId(int newId) – изменяет id вершины.

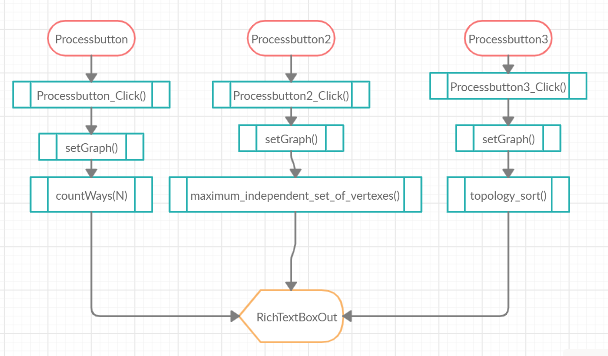
Форма Form1 содержит контейнер TabControl, два RichTextBox для ввода графа и вывода результата, кнопка очистки полей, кнопки импотра графа из файла и кнопка сохранения результата. TabControl содержит 5 вкладок, каждая вкладка содержит кнопку, которая вызывает функцию для решения задачи соответствующей названию вкладки.

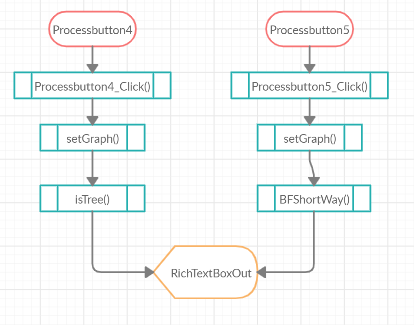
Методы класса Vertex:

1. void Importbutton\_Click(object sender, EventArgs e) – вызывает диалоговое окно для импорта графа из файла.
2. void Savebutton\_Cick(object sender, EventArgs e) - вызывает диалоговое окно для сохранения результата выполнения в файл.
3. void Clearbutton\_Click(object sender, EventArgs e) – очищает окно ввода и вывода.
4. Graph setGraph() – преобразует входной граф из текста в матрицу смежности и инициализирует ей объект графа. В результате функция возвращает объект графа.
5. void Processbutton\_Click(object sender, EventArgs e) – решает задачу №1 и выводит результат в окно вывода.
6. void Processbutton2\_Click(object sender, EventArgs e) – решает задачу №2 и выводит результат в окно вывода.
7. void Processbutton3\_Click(object sender, EventArgs e) – решает задачу №3 и выводит результат в окно вывода.
8. void Processbutton4\_Click(object sender, EventArgs e) – решает задачу №4 и выводит результат в окно вывода.
9. void Processbutton5\_Click(object sender, EventArgs e) – решает задачу №5 и выводит результат в окно вывода.

# UML диаграмма программы







# Инструкция по эксплуатации

Граф вводится в левое окно в виде матрицы смежности вершин, в которой веса вершин разделяются пробелами. Если формат матрицы неверен, то программа выдаст ошибку. Результат выводится в правом окне.

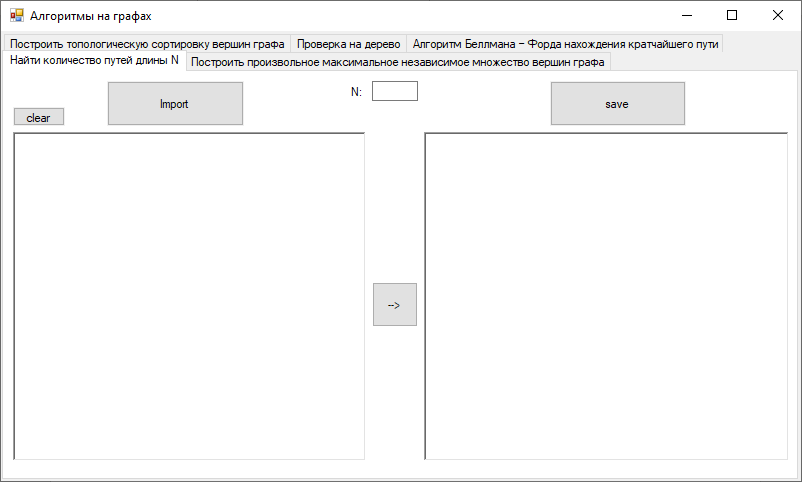


Рисунок 1 Начальное окно

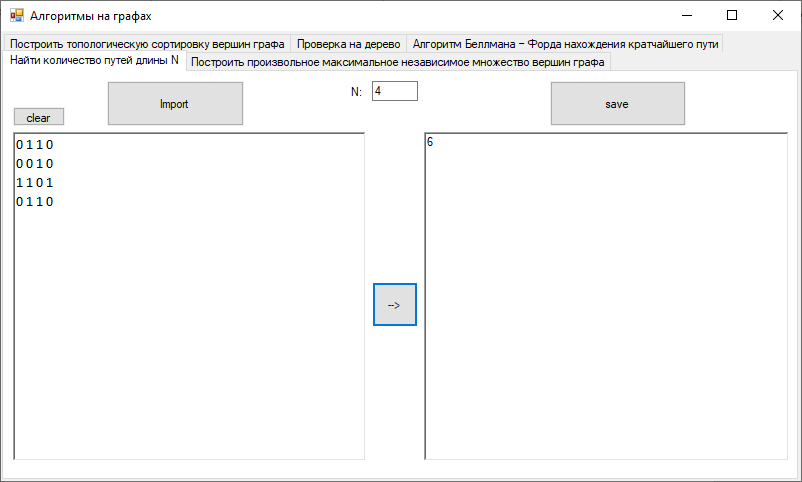


Рисунок 2 Решение задачи №1

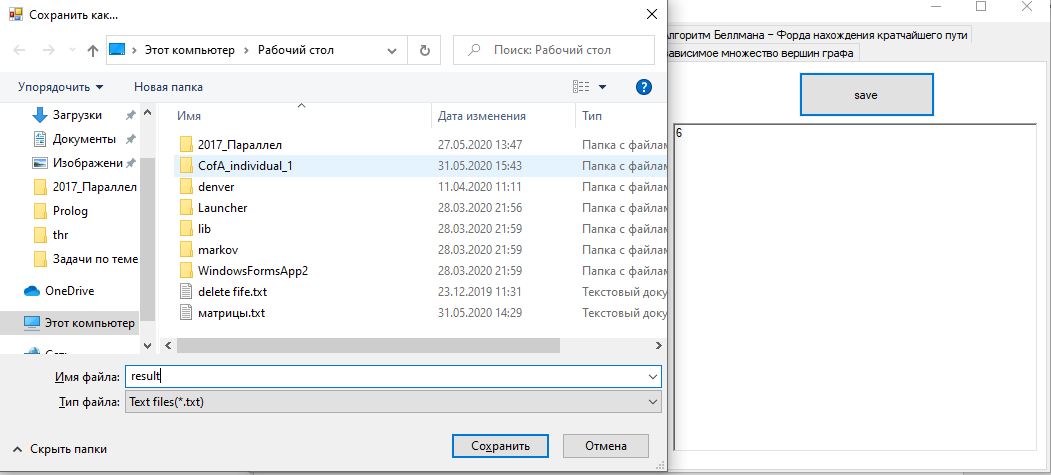


Рисунок 3 Сохранение результата

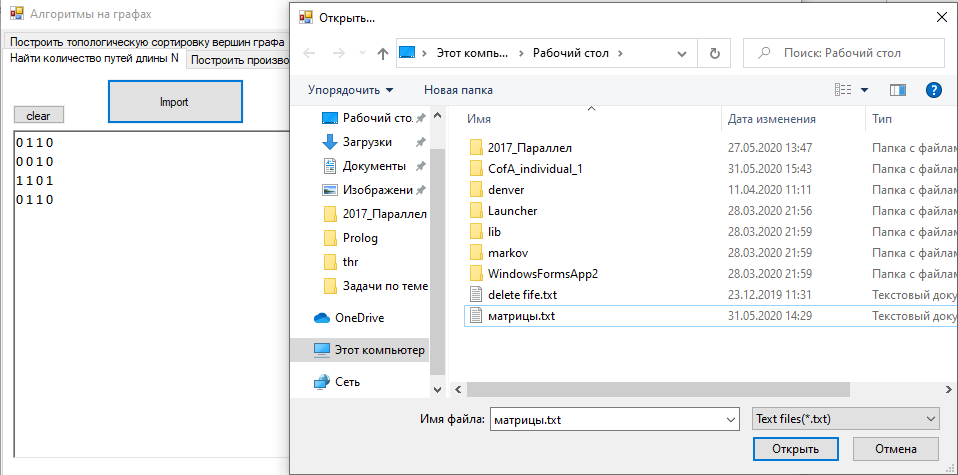
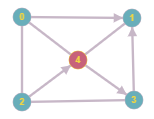


Рисунок 4 Импорт матрицы

# Графы для тестирования

Задача №1

0 1 1 0 1

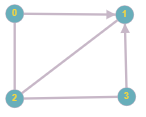
0 0 0 0 1

1 0 0 1 1

0 1 1 0 0

1 1 0 1 0

Результат: при N=2 Количество путей = 20

0 1 1 0

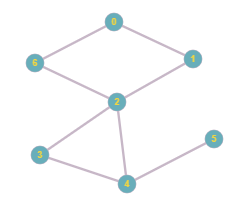
0 0 1 0

1 1 0 1

0 1 1 0

Результат: при N=2 Количество путей = 10

Задача №2



0 1 0 0 0 0 1

1 0 1 0 0 0 0

0 1 0 1 1 0 1

0 0 1 0 1 0 0

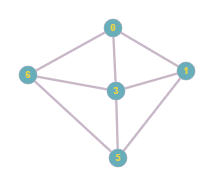
0 0 1 1 0 1 0

0 0 0 0 1 0 0

1 0 1 0 0 0 0

Результат: Все вершины: 0 1 2 3 4 5 6

Независимое множество вершин: 1 3 5 6

0 1 1 0 1

1 0 1 1 0

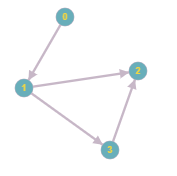
1 1 0 1 1

0 1 1 0 1

1 0 1 1 0

Результат: Все вершины: 0 1 2 3 4

Независимое множество вершин: 0 3

Задача №3

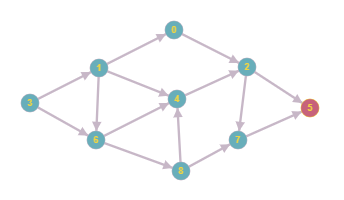
0 0 0 1

0 0 0 0

0 1 0 0

0 1 1 0

Результат: Отсортированный граф: 0 3 2 1



0 0 1 0 0 0 0 0 0

1 0 0 0 1 0 1 0 0

0 0 0 0 0 1 0 1 0

0 1 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

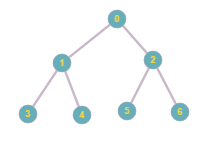
0 0 0 0 1 0 0 0 1

0 0 0 0 0 1 0 0 0

0 0 0 0 1 0 0 1 0

Результат: Отсортированный граф: 0 2 7 5 4 5 6 7 8

Задача №4

0 1 1 0 0 0 0

1 0 0 1 1 0 0

1 0 0 0 0 1 1

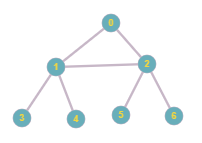
0 1 0 0 0 0 0

0 1 0 0 0 0 0

0 0 1 0 0 0 0

0 0 1 0 0 0 0

Результат: Дерево - да

0 1 1 0 0 0 0

1 0 1 1 1 0 0

1 1 0 0 0 1 1

0 1 0 0 0 0 0

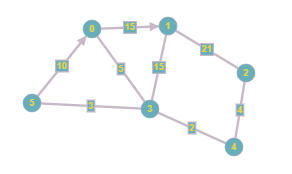
0 1 0 0 0 0 0

0 0 1 0 0 0 0

0 0 1 0 0 0 0

Результат: Дерево - нет

Задача №5

0 15 0 5 0 0

0 0 21 15 0 0

0 21 0 0 4 0

5 15 0 0 2 3

0 0 4 2 0 0

10 0 0 3 0 0

Результат: начальная вершина=0

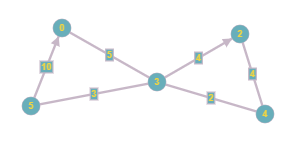
0->1=15

0->2=11

0->3=5

0->4=7

0->5=8

0 0 5 0 0

0 0 0 4 0

5 4 0 2 3

0 4 2 0 0

10 0 3 0 0

Результат: начальная вершина=0

0->1=9

0->2=5

0->3=7

0->4=8