**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**
**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Заместитель декана по учебно-методической работе факультета компьютерных наук
Доцент департамента больших данных и информационного поиска

_____ И.Ю. Самоненко
«_7_» _____мая_____ 2022 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия»
профессор департамента программной инженерии,
канд. техн. наук

_____ В.В.Шилов
«_7_» _____мая_____ 2022 г.

| | |
|---|---|
| *Подп. и дата* | |
| *Инв. № дубл.* | |
| *Взам. инв. №* | |
| *Подп. и дата* | |
| *Инв. № подл* | RU.17701729.10.03-01 |

# Приложение для визуализации алгоритма сжатия цветков

## Пояснительная записка

## ЛИСТ УТВЕРЖДЕНИЯ

## RU.17701729. 10.03-01 81 01-1

Исполнитель: студентка группы БПИ-206

_____ / Куликова Т. Д. /
«_7_»_____мая___2022 г.

**2022**

УТВЕРЖДЕН
RU.17701729.10.03-01 81 01-1-ЛУ

# Приложение для визуализации алгоритма сжатия цветков

## Пояснительная записка

## RU.17701729. 10.03-01 81 01-1

## Листов 33

| Инв. № подл | Подп. и дата | Взам. инв. № | Инв. № дубл. | Подп. и дата |
|---|---|---|---|---|
| RU.17701729.10.03-01 81 01-1 | | | | |

**2022**

# СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ ... 3
1.1. Наименование разработки ... 3
1.2. Основание для разработки ... 3
1.3. Краткая характеристика и область применения ... 3
2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ... 4
2.1. Функциональное назначение ... 4
2.2. Эксплуатационное назначение ... 4
3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ... 5
3.1. Постановка задачи на разработку программы ... 5
3.2 Описание алгоритма и функционирования программы ... 6
3.2.1 Рисование графа ... 6
3.2.2 Генерация случайного графа ... 8
3.2.3 Описание алгоритма сжатия цветков ... 8
3.3 Организация входных данных ... 18
3.3.1 Описание организации входных данных ... 18
3.3.2 Обоснование организации входных данных ... 18
3.4 Организация выходных данных ... 19
3.4.1 Описание организации выходных данных ... 19
3.4.2 Обоснование организации выходных данных ... 19
3.5 Описание и обоснование выбора и состава технических и программных средств ... 19
3.5.1. Описание технических и программных средств ... 19
3.5.2. Обоснование выбора технических и программных средств ... 19
4 ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ ... 21
4.1 Предполагаемая потребность ... 21
4.2 Ориентировочная экономическая эффективность ... 21
4.3 Экономические преимущества разработки по сравнению с аналогами ... 21
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ... 23
ПРИЛОЖЕНИЕ 1 ... 24
ПРИЛОЖЕНИЕ 2 ... 25
ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ ... 33

# 1. ВВЕДЕНИЕ

## 1.1. Наименование разработки

**Наименование разработки на русском языке**: «Приложение для визуализации алгоритма сжатия цветков».

**Наименование разработки на английском языке:** «Application for visualization of Blossom algorithm»

## 1.2. Основание для разработки

**Документ, на основании которого ведется разработка:**

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденная академическим руководителем программы тема курсового проекта «Приложение для визуализации алгоритма сжатия цветков»

## 1.3. Краткая характеристика и область применения

В процессе реализации того или иного алгоритма или структуры данных, а также решения различных задач по данной тематике часто появляется необходимость наглядно представить принципы работы алгоритмов и особенности структур данных. “Приложение для визуализации алгоритма сжатия цветков” – это web-приложение, которое облегчит пользователям понимание и запоминание шагов алгоритма сжатия цветков.

Алгоритм сжатия цветков используется для нахождения максимального паросочетания в неориентированном невзвешенном графе за полиномиальное время.

Данное приложение будет эксплуатироваться следующими категориями пользователей:

1) Студенты университетов, изучающие курсы по алгоритмам и структурам данных
2) Преподаватели, ведущие занятия на курсах, связанных с изучением алгоритмов и структур данных
3) Разработчики, которым необходимо реализовать алгоритм сжатия цветков, перед этим наглядно изучив принципы его работы
4) Исследователи в области математики и компьютерных наук

# 2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

## 2.1.Функциональное назначение

Функциональным назначением приложения является наглядное изображение каждого шага алгоритма сжатия цветков для поиска максимального паросочетания в неориентированном невзвешенном графе.

## 2.2.Эксплуатационное назначение

Разработанное web-приложение позволит пользователю облегчить изучение алгоритма сжатия цветков. Также с помощью данного приложения пользователь сможет находить паросочетания в графах для решения тех или иных прикладных задач, тестировать свою собственную реализацию алгоритма сжатия цветков.

# 3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

## 3.1.Постановка задачи на разработку программы

Данное web-приложение должно содержать только клиентскую часть.

Приложение должно обладать следующим функционалом:

1) Должна присутствовать возможность самостоятельно нарисовать граф.

   1.1) Должна предоставляться возможность добавить вершину графа.

   1.2) Должна быть реализована возможность выделить вершину или ребро путем нажатия на нее курсором.

   1.3) Должна предоставляться возможность добавления ребра, соединяющего две добавленные ранее вершины.

   1.4) Должна быть реализована возможность удаления вершины или ребра.

   1.5) Должна предоставляться возможность очистить поле для рисования графа.

2) Должна быть предусмотрена возможность создать случайный неориентированный невзвешенный граф, имеющий хотя бы две вершины, соединенные ребром.

3) После нажатия на кнопку запуска визуализации должны быть изображены последовательные шаги работы алгоритма.

   3.1) На каждой итерации поиска увеличивающего пути пользователь должен иметь возможность выбирать просматриваемую вершину и просматриваемое ребро.

   3.2) После нахождения очередного увеличивающего пути необходимо подсвечивать данный путь контрастным цветом.

   3.3) На каждом шаге алгоритма пользователь должен видеть текущее найденное паросочетание графа.

3.4) После определения увеличивающего пути должно быть изображено изменение паросочетания.

3.5) На каждом шаге алгоритма пользователю должен быть виден текст, описывающий суть этого шага.

3.6) Также на каждом шаге алгоритма должно отображаться состояние “леса” ребер и вершин графа.

3.7) При нахождении цветка (цикла нечетной длины) пользователь должен видеть сжатие графа.

3.8) Возвращение графа в первоначальное состояние (до сжатия цветков) также должно быть продемонстрировано в визуализации.

### 3.2 Описание алгоритма и функционирования программы

После открытия главной страницы приложения пользователь увидит на экране три панели:
“Blossom graph” – панель, в которой будет отображаться граф, к которому будет применен алгоритм.
“Forest” – панель, в которой будет отображаться состояние леса ребер и вершин графа. Назначение данного объекта и его роль в понимании алгоритма будет описано далее.
“Algorithm description” – панель, в которой можно будет увидеть описание результатов каждого шага алгоритма и действий пользователя.
В самом начале в панели “Blossom graph” будет изображен граф “по умолчанию”. Здесь пользователю предоставляется три возможности: нарисовать свой граф, нажав на кнопку “Create graph”, сгенерировать случайный граф, нажав на кнопку “Generate graph”, начать алгоритм, нажав на кнопку “Start blossom algorithm”.

#### 3.2.1 Рисование графа

Визуализация графа осуществляется с помощью средств библиотеки “Cytoscape”.
После нажатия на кнопку “Create graph” появится всплывающее окно с холстом и вспомогательными кнопками для рисования. Пользователь сможет осуществлять следующие действия:

1) **Добавить вершину графа:**

Добавление вершины может быть реализовано посредством двойного клика по холсту. При этом с использованием API библиотеки обрабатывается событие “dblclick”, берется позиция клика относительно холста, и в это место добавляется узел графа (с помощью функции add, в которую передается информация об узле графа в формате YAML). При этом в метод add передается id узла, который всегда на один больше, чем количество уже добавленных узлов. Еще одним параметром узла является label, в котором отображается порядковый номер добавленной вершины, данный номер всегда равен id узла. Также в функцию add передается позиция (относительно холста), в которой было поймано событие и в которую необходимо добавить узел.

2) **Удалить вершину графа:**

Удалить вершину можно посредством выделения вершины (с помощью одного клика на вершину) и нажатия на кнопку “Delete”. При этом обрабатывается событие “select”, после осуществления которого запоминается id выбранного узла, а после обработки события нажатия на кнопку “Delete”, удаляется последний выбранный узел с помощью функции remove.

3) **Добавить ребро графа:**

Чтобы добавить ребро, необходимо нажать клавишу ctrl и выбрать сразу два узла графа посредством одного клика на каждый узел (выбрать два узла одновременно возможно, если долго держать клавишу ctrl нажатой). При этом нажатие двух узлов одновременно отслеживается и запоминаются id обеих вершин. Далее с помощью функции add в граф добавляется ребро, в функцию add передаются id начала ребра (source) и конца (target).

4) **Удалить ребро графа:**

Удалить ребро графа можно, выбрав ребро путем одного клика по нему и нажав на кнопку “Delete”. При этом запоминается id последнего выбранного ребра, и при обработке события нажатия на кнопку “Delete” данное ребро удаляется с помощью функции remove.

5) **Очистить холст:**

Чтобы очистить холст для рисования графа, пользователю нужно нажать на кнопку “Clear”. При этом с помощью функции remove из графа удаляются все узлы и ребра.

6) **Отменить рисование графа и закрыть окно:**

   Для этого нужно нажать на кнопку “Cancel”, при этом ранее нарисованный граф не будет перенесен в основное окно.

7) **Завершить рисование графа и закрыть окно:**

   Для этого необходимо нажать на кнопку “OK”, при этом окно для рисования графа закроется, а нарисованный граф перенесется в основное окно в панель “Blossom graph”**.** Это реализовано путем полного копирования экземпляра графа cytoscape из всплывающего окна в экземпляр графа cytoscape из окна “Blossom graph”

### 3.2.2 Генерация случайного графа

После нажатия на кнопку “Generate graph” в окне “Blossom graph” появится случайный граф. Алгоритм генерации графа следующий:

1) Сначала генерируется количество вершин графа. Это количество может принимать значения от 2 до 10, следовательно, количество генерируется по формуле **max (2; Math.random() * 10 + 1).**
2) Далее в граф добавляются вершины со значениями label от 1 до максимального количества вершин. По умолчанию в граф уже добавлены первые две вершины и ребро между ними.
3) Для каждой вершины генерируется случайная позиция на панели: для генерации случайной координаты x берется ширина панели и умножается на случайное вещественное число от 0 до 1 (Math.random()), аналогично находится случайная координата y, только для подсчета берется высота панели.
4) Теперь заметим, что между любыми двумя вершинами ребро может как существовать, так и не существовать. Пусть вероятность того, что между двумя отдельно взятыми вершинами есть ребро, равна 0.5. Тогда для каждой пары созданных вершин будем генерировать вещественное число от 0 до 1 (с помощью Math.random()). Если полученное число больше 0.5, добавим ребро, соединяющее данную пару вершин, в противном случае не будем ничего добавлять.
5) Полученный граф отображается в панели “Blossom graph”.

### 3.2.3 Описание алгоритма сжатия цветков

Алгоритм сжатия цветков [1-4] применяется для поиска максимального паросочетания в произвольном неориентированном невзвешенном графе за время O(n^3). Паросочетанием в графе называется набор ребер этого графа, такой, что никакие два ребра не инцидентны одной вершине.

Основная идея алгоритма заключается в поиске так называемых увеличивающих путей (augmenting paths). Увеличивающим путем называется путь, в котором ребра, входящие в текущее найденное паросочетание, чередуются с ребрами, не входящими в это паросочетание, причем в начале и в конце такого пути должны находиться вершины, не входящие в паросочетание. Заметим, что в таком пути всегда нечетное количество ребер, причем ребер, не входящих в паросочетание, больше. Это значит, что в таком пути можно увеличить количество ребер, входящих в паросочетание, на 1, путем добавления в паросочетание ребер, не входивших в него до этого, и исключения из него ребер, входивших в него до этого.

Пример увеличения количества ребер в паросочетании с помощью увеличивающего пути изображен на рисунках 1-2:

*Рисунок 1-2 – примеры увеличивающих путей и увеличения количества ребер, находящихся в паросочетании, с помощью них*

Условие нахождения максимального паросочетания описывается леммой Бержа, которая говорит, что, если в графе нельзя найти увеличивающий путь относительно текущего паросочетания M, то данное паросочетание M максимально.

Следовательно, можно кратко описать общую схему шагов алгоритма:

Пусть имеется граф G и паросочетание M.

1) Пытаемся найти увеличивающий путь
2) Если удалось найти, то:
   2.1 – Увеличиваем текущее паросочетание с помощью найденного увеличивающего пути
   2.2 - Ищем новый увеличивающий путь в том же графе, но уже с новым паросочетанием
3) иначе:

   Максимальное паросочетание найдено, заканчиваем алгоритм.

## Нахождение увеличивающего пути

1) Все ребра и вершины, на определенном этапе входящие в паросочетание, помечаются в приложении **красным цветом**.
2) Все вершины, которые не входят в паросочетание, в приложении помечаются **синим цветом** (см. рис. 3).

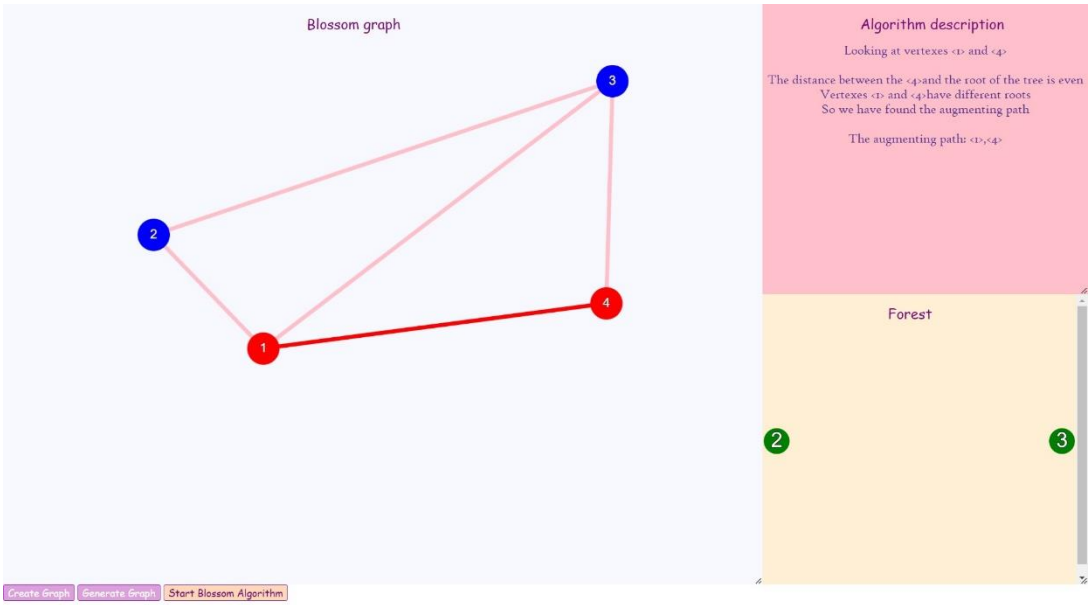


*Рисунок 3 – вершины и ребра, входящие в паросочетание, выделены красным цветом, вершины, не входящие в паросочетание, выделены синим цветом.*

3) Для каждой вершины, не входящей в паросочетание, создаем дерево из одной этой вершины и помещаем это дерево в лес (панель “Forest”), данная вершина будет корнем дерева (см. рис. 4).

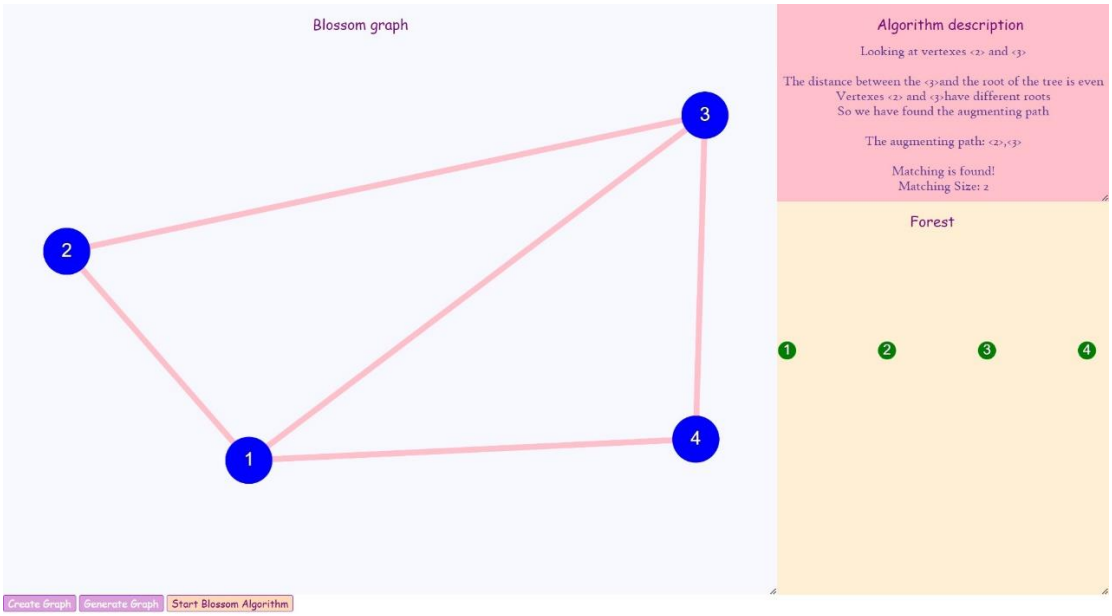

*Рисунок 4 – вершины 1, 2, 3, 4 – не входят в паросочетание, и на панели “Forest” мы видим 4 дерева из одной вершины, данные вершины будут корнями своих деревьев.*

4) Теперь надо выбрать просматриваемую вершину. Просматриваемой вершиной может быть вершина, не входящая в паросочетание, то есть, вершина, помеченная синим цветом. В приложении эти вершины будут кликабельными, и пользователь сможет выбрать одну из них. После этого все инцидентные данной вершине ребра подсветятся синим цветом (см. рис. 5).

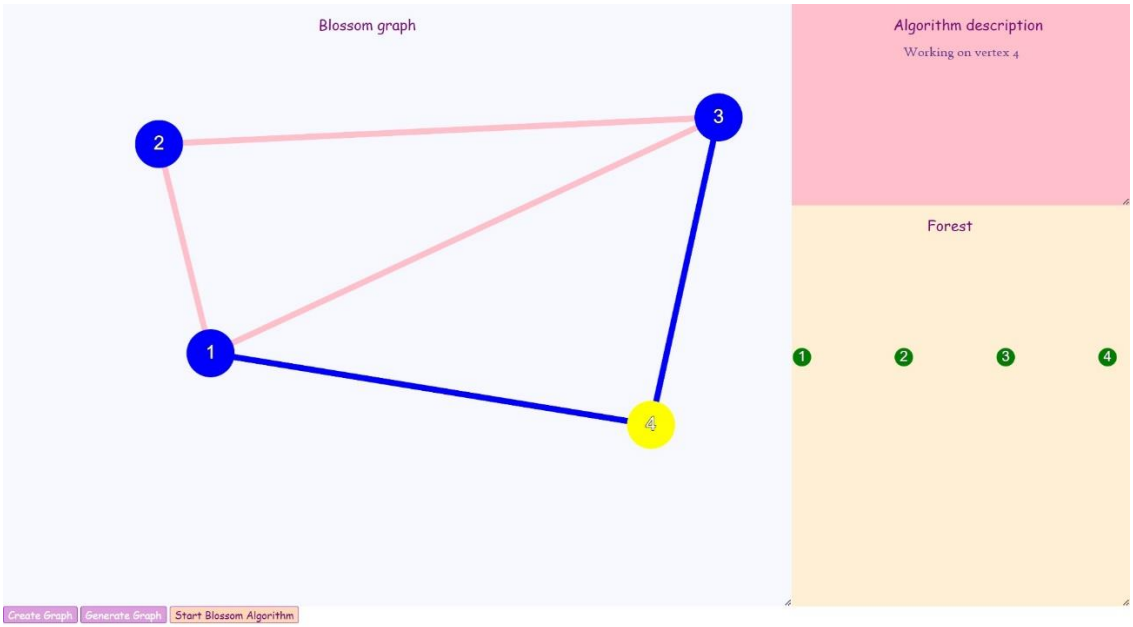


*Рисунок 5 – вершина 4 была выбрана пользователем, она подсвечивается желтым цветом, инцидентные ей ребра (4-3 и 4-1) подсвечены синим цветом.*

5) Теперь необходимо рассмотреть все инцидентные выбранной вершине ребра, просматривать их можно в разном порядке, поэтому пользователю предлагается выбрать ребро среди подсвеченных синим цветом ребер. Пусть сейчас мы рассматриваем некоторое ребро e, инцидентное выбранной вершине v.

Возможны 4 случая:

**1 случай**

Другая вершина (назовем ее w), инцидентная ребру e, не расположена в лесу (в приложении лес отображается в панели “Forest”), значит, она принадлежит паросочетанию. Рассмотрим ребро из паросочетания, инцидентное вершине w, найдем вторую вершину, принадлежащую данному ребру. Назовем ее x. Вершина v уже находится в лесу. Добавим туда ребро e, вершину w, ребро из паросочетания, инцидентное ребру w, вершину x (то есть, подвесим к

вершине v пару следующих друг за другом ребер: ребро, не принадлежащее паросочетанию, и ребро, принадлежащее паросочетанию). При этом вершина x станет синей, ее можно будет выбрать на следующем шаге, просматриваемая вершина v и ребро e в приложении подсветятся зеленым (см. рис. 6).

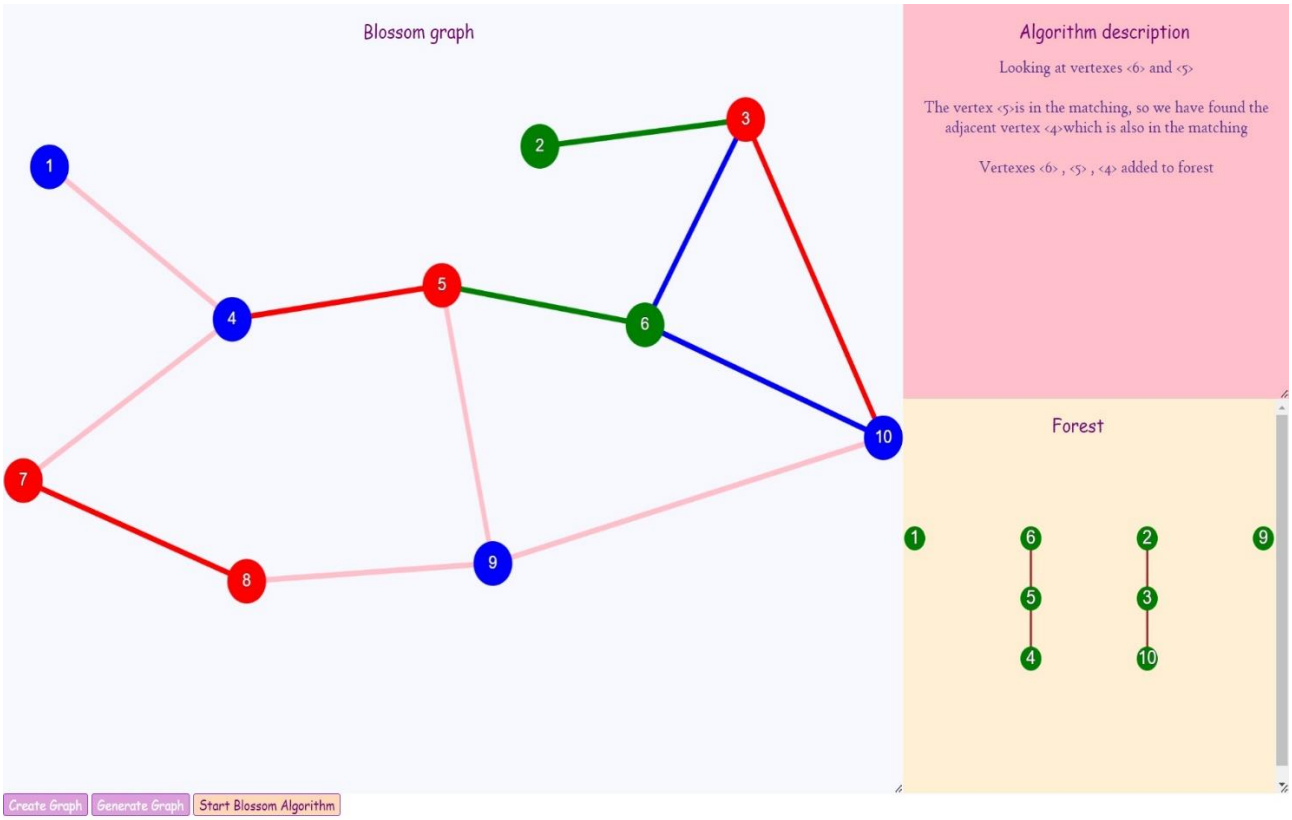


*Рисунок 6 – здесь рассмотрим вершину 6 и ребро 6-5, 5 принадлежит паросочетанию, следовательно, нужно добавить в лес ребра 6-5 и 5-4, справа в панели Forest видим дерево 6-5-4, аналогично можно сказать и о вершинах 2-3-10.*

**2 случай**

Вершина w находится в лесу, и расстояние от нее до корня дерева, в котором она находится (то есть, длина кратчайшего пути от данной вершины до корня), нечетно. В этом случае мы ничего не делаем, просматриваемое ребро помечается как просмотренное. В приложении данное ребро примет первоначальный цвет (светло-розовый), а просматриваемая вершина не поменяет цвет.

Рассмотрим рисунок 7. Пусть для рассмотрения мы выбрали ребро 6-3. Посмотрим на панель Forest. Вершина 3 уже есть в лесу. Она находится в дереве, в котором корнем является вершина 2. Расстояние от вершины 3 до вершины 2 равно 1, и оно нечетно. Ребро 3-2 не лежит в паросочетании, ребро 6-3, так как расстояние от 6 до корня четно, тоже не лежит в паросочетании, следовательно, путь 6-3-2 не является увеличивающим, и ничего делать не нужно. Результат этого шага показан на рисунке 8.

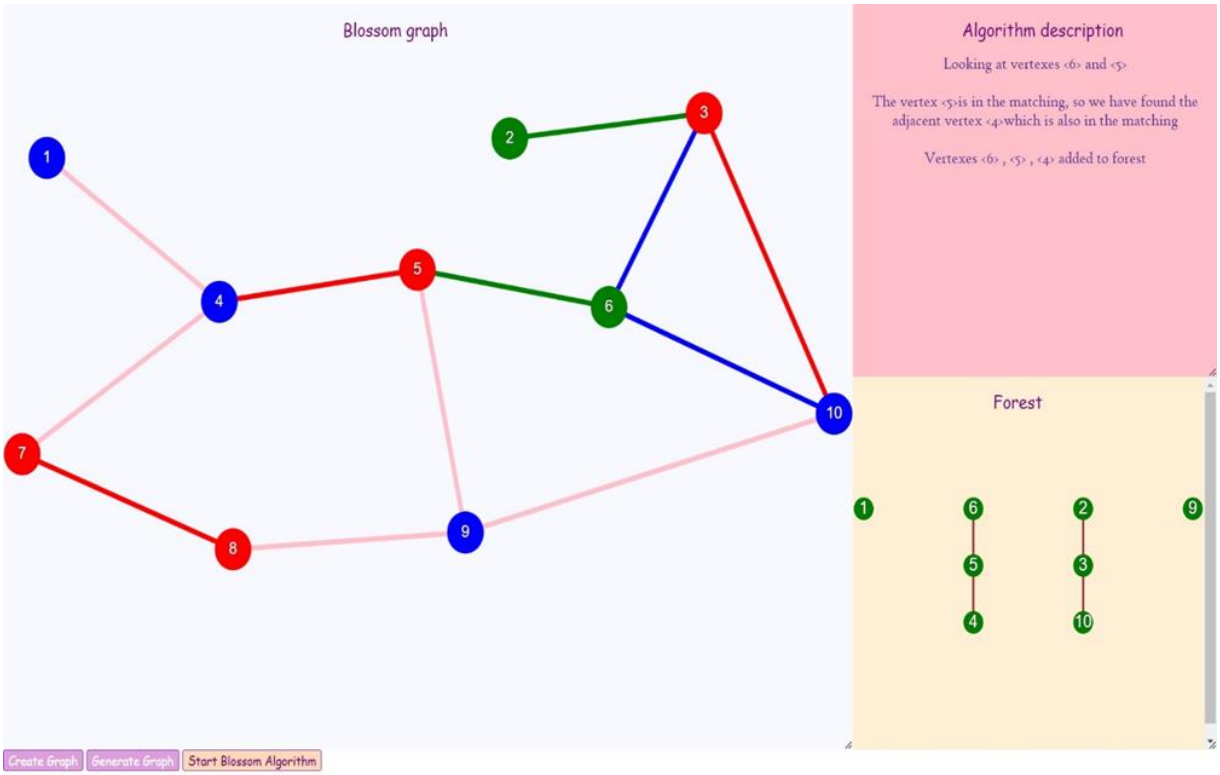


*Рисунок 7*

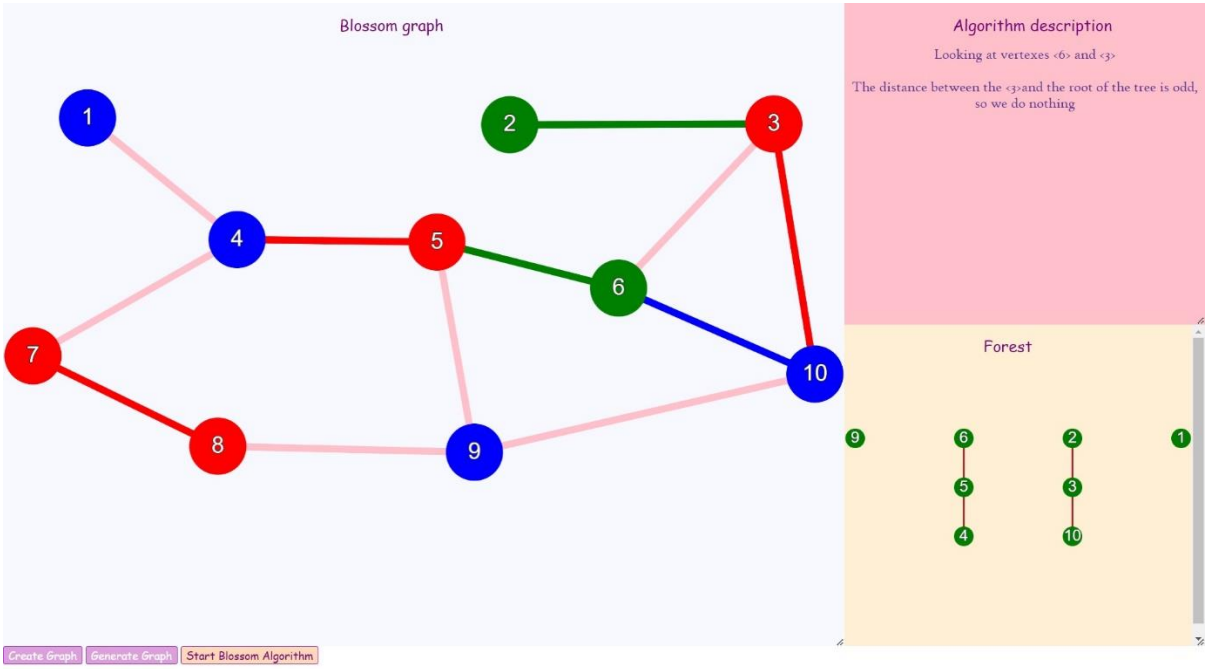


*Рисунок 8*

**Случай 3**

Вершина w находится в лесу, расстояние от нее до корня дерева, в котором она находится, четно. Вершины v и w находятся в разных деревьях. Данная ситуация означает, что мы нашли увеличивающий путь. Увеличиваем паросочетание на 1 (см. рис. 9-10).

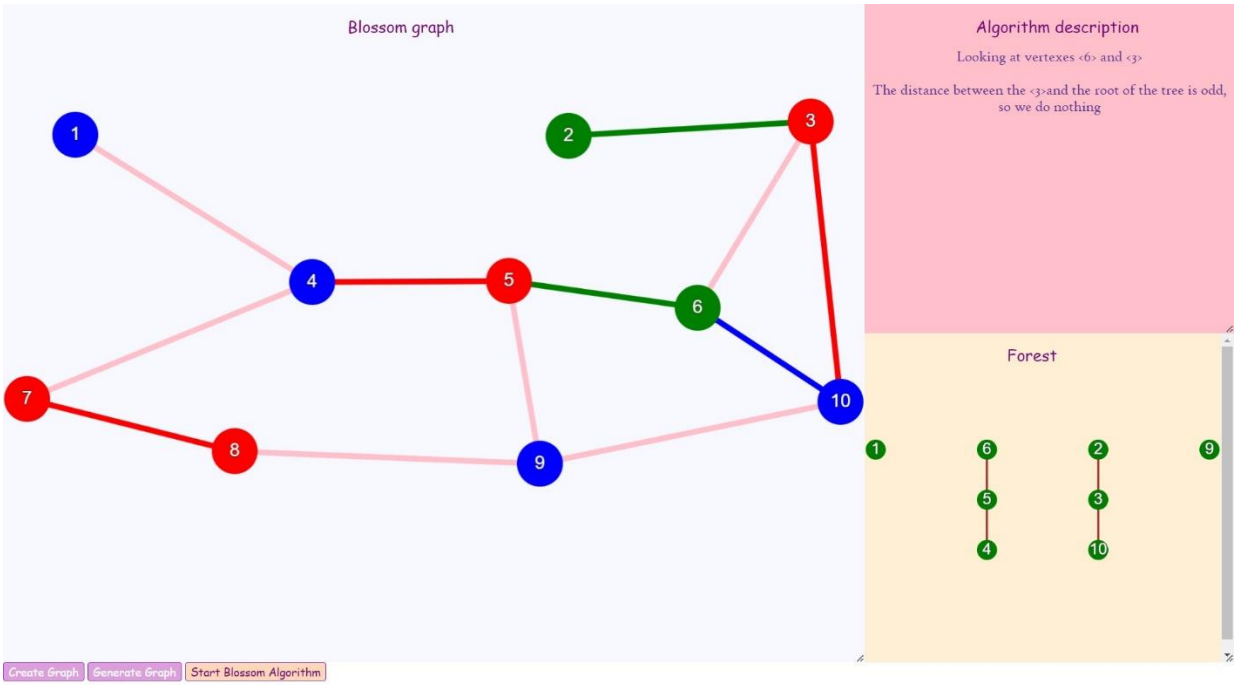


*Рисунок 9 – выбираем ребро 6-10, 10 находится в лесу, расстояние до корня равно 2, 6 и 10 в разных деревьях, следовательно, мы нашли увеличивающий путь 6-10-3-2 (идем от корня дерева, в котором находится первая вершина, до первой вершины, затем переходим ко второй вершине и идем до корня дерева, в котором находится вторая вершина).*

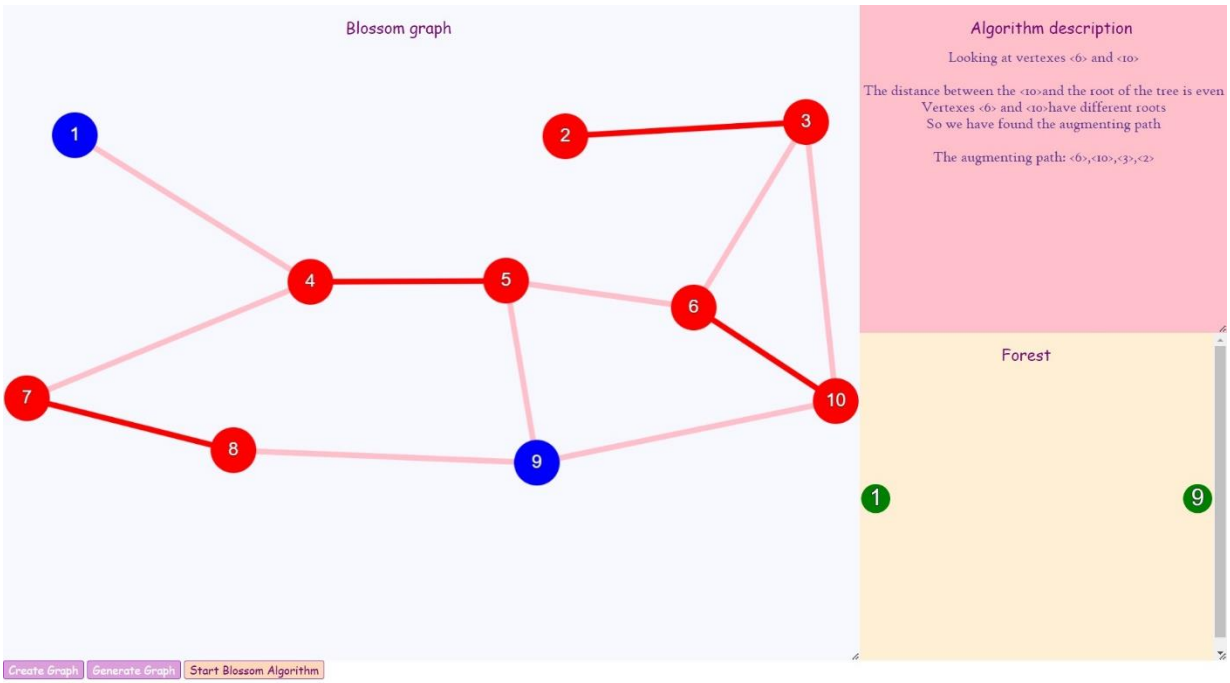


*Рисунок 10 – увеличиваем паросочетание на 1, ребра 6-10 и 3-2 помещаются в паросочетание, ребро 3-10 исключается из паросочетания.*

Теперь необходимо проверить, является ли найденное паросочетание максимальным. Для этого граф разбивается на компоненты связности, и в каждой компоненте ищутся вершины, не принадлежащие паросочетанию. Если хотя бы в одной из компонент найдено как минимум две вершины, не принадлежащие паросочетанию, алгоритм продолжается. Если в процессе выполнения алгоритма возникает ситуация, при которой заканчиваются просматриваемые вершины, алгоритм заканчивается, а найденное к тому моменту паросочетание признается максимальным.

**4 случай**

Вершина w находится в лесу, расстояние от нее до корня дерева, в котором она находится, четно. Вершины v и w находятся в одном дереве. Данная ситуация означает, что мы нашли в нашем графе цикл нечетной длины. Сожмем данный цикл в одну вершину и продолжим искать увеличивающий путь в сжатом графе, при этом возможны дальнейшие сжатия циклов нечетной длины в уже сжатом графе. Сразу после нахождения увеличивающего пути в таком графе все цветки, которые до этого были сжаты, разжимаются, и граф возвращается к первоначальному состоянию (см. рис. 11-16).

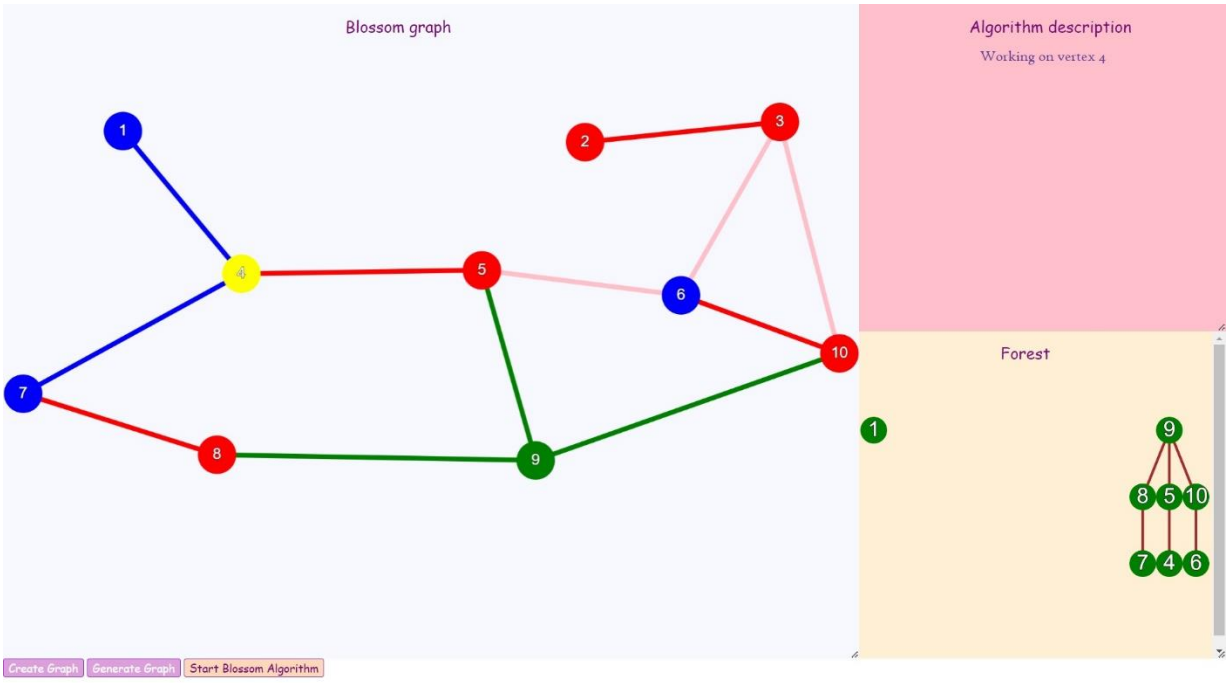


*Рисунок 11 – рассмотрим вершину 4 и ребро 4-7, вершина 7 находится в лесу, расстояние от нее до корня равно 2, и оно четно, вершины 4 и 7 находятся в одном дереве. Мы нашли цветок 9-5-4-7-8. Вершина 9 не находится в паросочетании и является корнем цветка.*

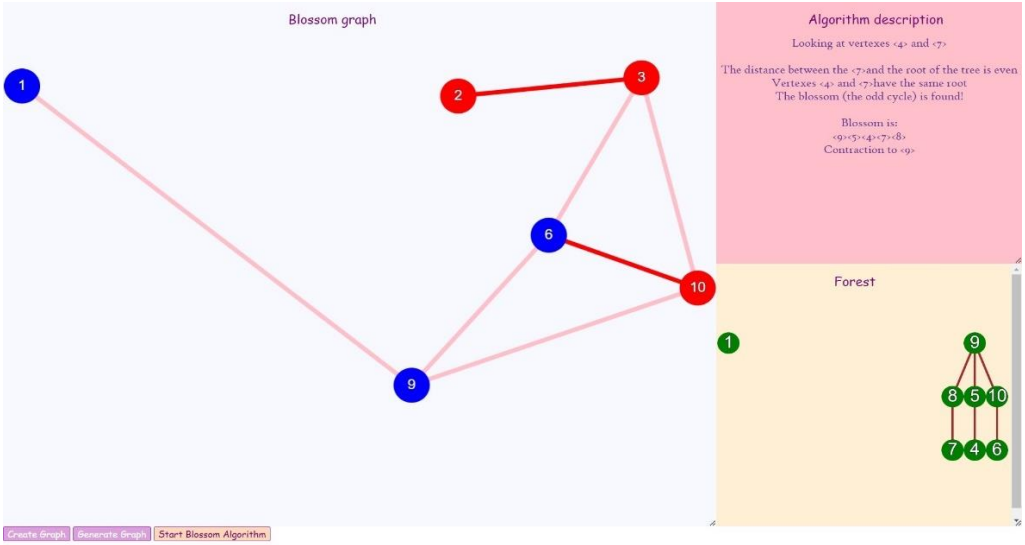


*Рисунок 12 – цветок 9-5-4-7-8 сжат в вершину 9.*

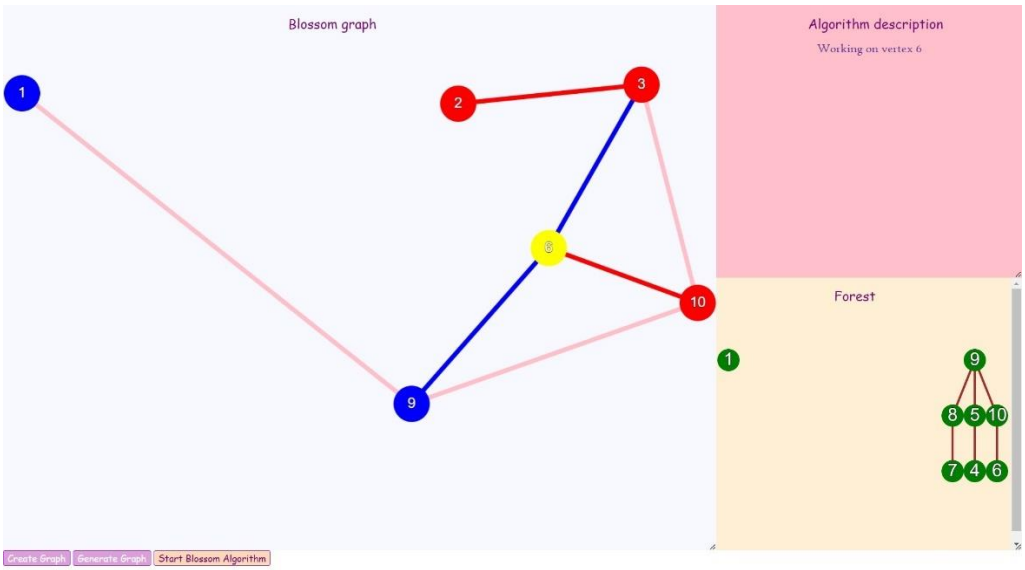


*Рисунок 13 – рассмотрим вершину 6 и ребро 6-9, вершина 9 находится в лесу, ее расстояние до корня равно 0, оно четно, и вершины 6 и 9 находятся в одном дереве, следовательно, мы снова нашли цветок 9-10-6 с корнем 9.*

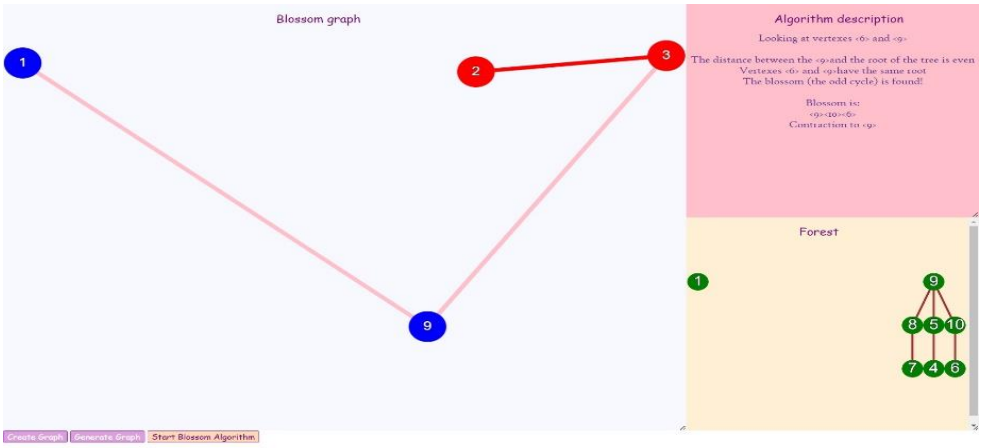


*Рисунок 14 – цветок 9-10-6 сжат в вершину 9*

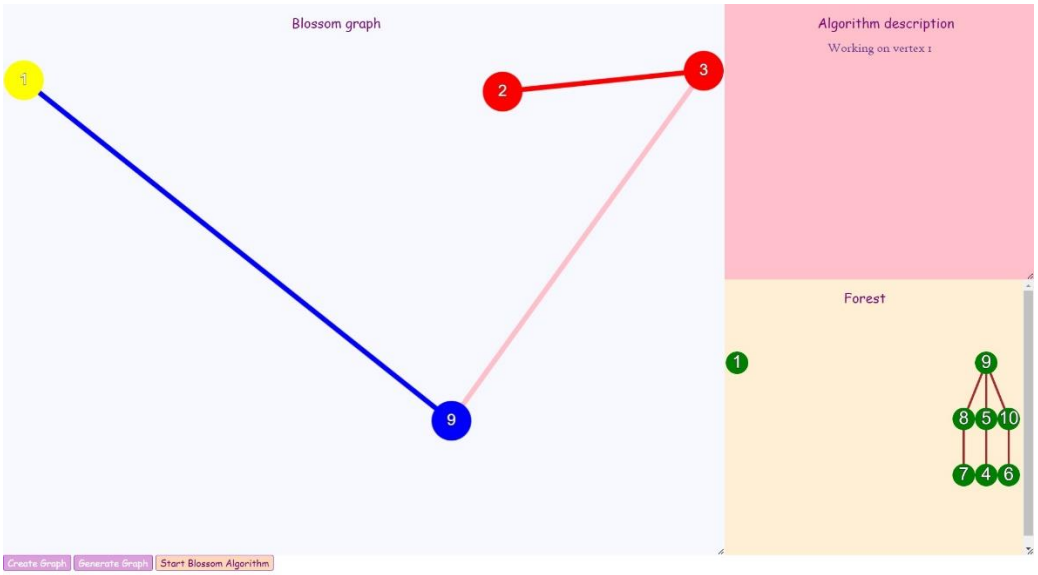


*Рисунок 15 – рассматриваем вершину 1 и ребро 1-9 – находим увеличивающий путь 1-9, теперь необходимо разжать все цветки, которые были сжаты до этого.*

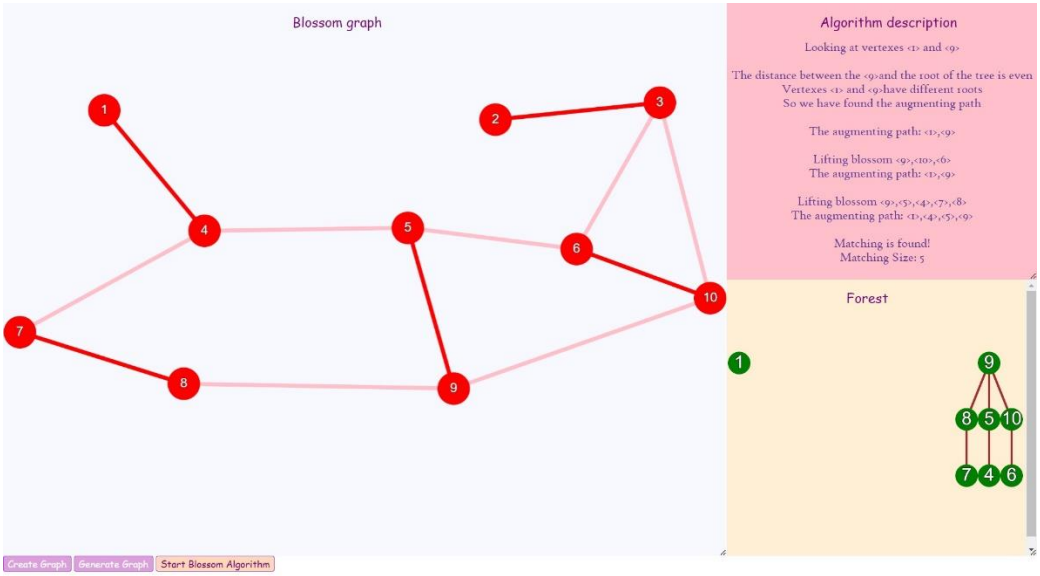


*Рисунок 16 – все цветки разжаты, граф вернулся к первоначальному состоянию, максимальное паросочетание найдено, так как в графе больше нет увеличивающих путей*

## Алгоритм сжатия цветка

1) Сначала находятся все узлы, принадлежащие цветку. От вершины v идем вверх кратчайшим путем по ветке дерева до корня, кладем вершины в массив, потом этот массив переворачиваем и идем от корня до вершины w вниз по ветке дерева, кладем эти вершины в массив – получаем массив вершин цветка.
2) Удаляются все ребра, образующие найденный нечетный цикл.
3) Для каждой вершины цветка удаляются все инцидентные ей ребра

4) Каждая не принадлежащая цветку вершина, которая до этого была связана ребром с какой-то вершиной цветка, связывается ребром с корнем цветка.

**Алгоритм разжатия цветка**

1) Если цветок не входит в состав найденного увеличивающего пути, то просто делаем все его вершины и ребра видимыми, удаляем лишние ребра, не входящие в предыдущие состояния графа (до сжатия цветка).
2) Если цветок входит в увеличивающий путь, то находим индекс вершины, которая была сжата, в увеличивающем пути.
3) Далее ищем индекс вершины, принадлежащей цветку и связанной со следующей (или предыдущей в случае конца увеличивающего пути) вершиной увеличивающего пути.
4) В зависимости от четности индексов, найденных во 2 и 3 пунктах, решаем, как обходить цветок (длина обхода всегда должна быть четной). Если цветок находится в начале или в конце увеличивающего пути, то нужно обходить цветок от корня до вершины, инцидентной исходящему из цветка ребру в направлении пути четной длины. Если цветок в середине увеличивающего пути, то нужно идти от вершины, инцидентной входящему в цветок ребру до вершины, инцидентной исходящему из цветка ребру, в направлении пути четной длины.
5) Таким образом находим полный увеличивающий путь и увеличиваем паросочетание.

## 3.3 Организация входных данных

### 3.3.1 Описание организации входных данных

1) На вход алгоритма подается произвольный неориентированный невзвешенный граф.
2) Граф должен иметь хотя бы две вершины, соединенные ребром.
3) При генерации случайного графа количество вершин графа не превышает 10.
4) Граф, который будет подаваться на вход алгоритма, можно получить двумя способами: нарисовав граф в разделе “Create graph” или сгенерировать случайный граф, нажав на кнопку “Generate graph”.

### 3.3.2 Обоснование организации входных данных

1) Вид графа, который подается на вход алгоритма, обусловлен особенностями и целями выполнения алгоритма.
2) Условие о двух вершинах, соединенных ребром, связано с тем, что нет смысла искать паросочетание в графе, в котором меньше двух вершин или нет ребер.
3) Ограничение на количество вершин при генерации графа связано с удобством

восприятия визуальной информации. Приложение имеет образовательные цели, поэтому понятность и компактность визуализации играют большую роль.

4) Рисование графа – это наиболее понятный и удобный способ задать граф для пользователя. Генерация графа также удобна для пользователя, так как может быть произведена по одному клику на кнопку.

## 3.4 Организация выходных данных

### 3.4.1 Описание организации выходных данных

1) Программа корректно изображает поэтапную визуализацию работы алгоритма сжатия цветков.
2) После окончания работы алгоритма ребра и вершины графа, принадлежащие найденному максимальному паросочетанию, выделены красным цветом.
3) Во время визуализации алгоритма каждый его шаг сопровождается текстовым комментарием, описывающим, что на этом шагу происходит.
4) На каждой итерации алгоритма отображается состояние “леса” ребер и вершин.
5) После нахождения максимального паросочетания выводится количество ребер, входящих в него (размер паросочетания).

### 3.4.2 Обоснование организации выходных данных

Организация выходных данных направлена на максимальное облегчение понимания пользователем того, что происходит на каждом шаге алгоритма сжатия цветков.

## 3.5 Описание и обоснование выбора и состава технических и программных средств

### 3.5.1. Описание технических и программных средств

1) Для эксплуатации приложения требуется наличие веб-браузера, запущенного на компьютере, а также доступ к сети Интернет.
2) Для визуализации графа используется библиотека “cytoscape” [6].
3) Алгоритм и его визуализация реализованы на языке JavaScript (см. Приложения 1, 2).

### 3.5.2. Обоснование выбора технических и программных средств

1) Приложение может эксплуатироваться только на компьютере, так как на других устройствах у пользователя могут возникнуть трудности с рисованием графа (неудобно делать двойной клик) и с самим алгоритмом (неудобно выбирать вершину и ребро), более того, тяжело воспринимать визуализацию алгоритма на экранах маленьких размеров.

2) Выбор библиотеки Cytoscape обусловлен удобством API и полнотой предоставленных в библиотеке программных средств: есть возможность выбирать вершины и ребра, реализованы все необходимые события, можно изменять цвета ребер и вершин, есть возможность создать анимацию, есть возможность добавлять вершины и ребра и удалять их по id, есть возможность сделать ребро или вершину невидимыми (без удаления), можно получить элемент графа по его id.

# 4 ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

## 4.1 Предполагаемая потребность

Необходимость в данном программном продукте имеют несколько категорий пользователей:

1) Преподаватели ВУЗов, ведущие курсы, связанные с алгоритмами и структурами данных.
2) Студенты ВУЗов, изучающие алгоритмы и структуры данных.
3) Разработчики, которым требуется реализовать этот алгоритм для решения прикладных задач.
4) Исследователи в области математики и компьютерных наук.

## 4.2 Ориентировочная экономическая эффективность

Программа позволит быстро и эффективно изучить принципы работы алгоритма, что поможет значительно сэкономить время, требующееся для понимания алгоритма и его реализации.

## 4.3 Экономические преимущества разработки по сравнению с аналогами

**Сравнение конкурентов**

*Таблица 1. Анализ конкурентов.*

| | Приложение для визуализации алгоритма сжатия цветков | Algorithm Visualizer | VisuAlgo | Algorithm Wiki | Vamonos | Data structure visualizations |
|---|---|---|---|---|---|---|
| Есть визуализация алгоритма сжатия цветков | + | - | + | - | - | - |
| Пояснения к каждому шагу алгоритма | + | + | + | + | + | + |

| | | | | | | |
|---|---|---|---|---|---|---|
| Возможность самостоятельно задать граф | + | - | + | - | + | - |
| Возможность сгенерировать граф | + | - | - | + | - | + |
| Интерактивность (возможность пользователя самому выполнять алгоритм или влиять на его выполнение) | + | - | - | - | - | - |
| Изображение “леса” вершин и ребер графа (только для визуализации алгоритма сжатия цветков) | + | - | - | - | - | - |

Таким образом, среди конкурентов визуализацию алгоритма сжатия цветков имеет только VisuAlgo, однако в VisuAlgo нет изображения “леса” вершин и ребер графа. Более того, ни один из аналогов не имеет возможности изучать алгоритм интерактивно (во всех аналогах можно только запустить алгоритм и просто смотреть на его выполнение) (см. Таблицу 1).

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Amy Shoemaker and Sagar Ware -Edmond's Blossom Algorithm //[Электронный ресурс]: Stanford University – Режим доступа: https://web.stanford.edu/~rezab/classes/cme323/S16/projects_reports/shoemaker_vare.pdf .свободный. (дата обращения: 06.05.2022).

2) Алгоритм сжатия цветков // [Электронный ресурс]: Википедия – Режим доступа: https://ru.wikipedia.org/wiki/. свободный. (дата обращения: 03.02.2022).

3) Richard L. Apodaca, Edmonds' Blossom Algorithm Part 1: Cast of Characters // [Электронный ресурс]: Depth-first – Режим доступа: https://depth-first.com/articles/2020/09/28/edmonds-blossom-algorithm-part-1-cast-of-characters/. свободный. (дата обращения: 03.02.2022).

4) Алгоритм Эдмондса нахождения наибольшего паросочетания в произвольных графах // [Электронный ресурс]: Maximal – Режим доступа: https://e-maxx.ru/algo/matching_edmonds свободный. (дата обращения: 07.05.2022).

5) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

6) Graph theory (network) library for visualization and analysis // [Электронный ресурс]: Cytoscape.js – Режим доступа: https://js.cytoscape.org/. свободный. (дата обращения: 06.05.2022).

# ПРИЛОЖЕНИЕ 1

## ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ СКРИПТОВ JAVASCRIPT

*Таблица 1.*

| **Скрипт** | **Назначение** |
|---|---|
| algorithm | Содержит методы, связанные с реализацией шагов алгоритма и обработкой вершин и ребер, выбранных пользователем |
| graph | Содержит класс Graph и его методы |
| auxiliaryFunctions | Содержит вспомогательные функции, применяемые для осуществления шагов алгоритма |
| edge | Содержит класс edge и его методы |
| lifting | Разжатие цветка |
| vertex | Содержит класс vertex и его методы |
| drawGraph | Рисование графа в основной панели “Blossom graph” |
| initialization | Привязывание обработчиков событий к кнопкам, генерация графа, рисование графа и перенос его в основную панель |
| forest | Рисование леса ребер и вершин |
| makeVisual | Методы для визуализации алгоритма |

# ПРИЛОЖЕНИЕ 2

## ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ МЕТОДОВ, КЛАССОВ И ПОЛЕЙ

## algorithm

*Таблица 1.*

| Поля | |
|---|---|
| **Имя** | **Назначение** |
| matching | паросочетание |
| clickedVertex | последняя выбранная вершина |
| forestRoots | мэп, в котором хранятся вершины, находящиеся в лесу, и им в соответствие приведены корни деревьев, в которых они находятся |
| forestParents | пары родитель-ребенок из леса |
| forestChild | Пары ребенок-родитель из леса |
| forestHeights | Мэп, который хранит пары вершина – глубина, на которой она находится в дереве |
| vertexesToCheck | Вершины, которые можно просматривать на данном этапе алгоритма |
| augPath | Последний найденный увеличивающий путь |
| matchingConditions | Состояния паросочетания после каждого сжатия цветка |
| blossoms | Все цветки, которые были сжаты |
| graphConditions | Состояния графа после каждого сжатия цветка |
| contractedVertexes | Все сжатые вершины |
| graph | Граф, к которому будет применяться алгоритм |
| adjacentEdges | Инцидентные ребра для текущей выбранной вершины |
| contrMatching | Сжатое паросочетание |
| notContractedGraph | Состояние графа до последнего сжатия цветка |

*Таблица 2.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| findAugmentingPath | graph, matching, blossomVertexes, cy | Инициализирует увеличивающий путь, очищает лес, находит все вершины, которые могут быть обработаны на данном этапе |
| processingExposedVertexes | blossomVertexes, cy | Окрашивает вершины, не принадлежащие паросочетанию, добавляет их в лес |
| findAugPathWithBlossom | contractedGraph, contractedMatching, blossomVertexes, cy | Окрашивает вершины, не принадлежащие паросочетанию, после сжатия цветка |
| vertexProicessing | cy | Обрабатывает вершину, выбранную пользователем, находит все инцидентные ей ребра, не лежащие в паросочетании, и окрашивает их. |
| edgeProcessing | cy, source, target | Обрабатывает ребро, выбранное пользователем, и запускает шаг алгоритма |
| checkingSelectedEdge | w, cy | Исходя из того, какая вершина и какое ребро были выбраны пользователем, определяет, к какому случаю относится данная ситуация и запускает соответствующий метод |
| secondVertexInMatching | w, cy | Реализует первый случай (добавление двух ребер в лес) |
| foundAugmentingPath | cy, w | Реализует третий случай (нахождение увеличивающего пути) |
| initializeOrDrawAugPath | wasBlossomed, cy | Если граф до этого был разжат, заново инициализирует все поля, иначе изображает изменение паросочетания в соответствии с найденным увеличивающим путем. |
| endOrGoNext | graphToCheck, cy | Проверяет, можно ли еще найти в графе увеличивающий путь, если да, то продолжает алгоритм, если нет, то возвращает граф в изначальное состояние и выводит информацию о паросочетании |
| lifting | i, cy | Разжимает цветок с определенным индексом в массиве blossoms. |
| foundNothing | cy, w | Реализует второй случай (когда не нужно ничего делать) |
| blossomFound | w, cy | Реализует четвертый случай (нахождение цветка) |
| endingAlgorithm | selectingEdge, cy | Проверяет, можно ли закончить алгоритм, и если да, то увеличивает паросочетание выводит финальную информацию о паросочетании |
| fullLifting | cy | Разжимает все цветки, которые до этого были сжаты |
| addToForest | v, w, x | Добавляет вершины v, w, x в лес |
| blossomConstruction | cy, matching, forestRoots, forestParents, forestChild, forestHeights | Формирование цветка, который необходимо сжать |
| contraction | blossomVertexes, cy | Сжатие цветка и текущего паросочетания |
| contractBlossom | graph, blossom, cy | Сжатие цветка |

| | | |
|---|---|---|
| initializeFields | | Инициализация полей |
| checkAndStart | cy | Проверка введенного графа и стар алгоритма |
| selectingVertexEvent | cy | Обработка события выбора вершины |
| selectingEdgeEvent | cy | Обработка события выбора ребра |
| addEdgesAndVertexesToGraph | cy | Инициализация поля graph нарисованным на панели графом (cy) |
| getGraphFromCanvas | cy | Инициализация поля graph, нахождение его компонентов связности, привязка основных событий, старт алгоритма |

## auxiliaryFunctions

*Таблица 3.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| finalOutput | matching | Выводит размер паросочетания и активирует заблокированные до этого кнопки |
| correctingNodesAndEdges | cy | Делает все узлы и ребра некликабельными |
| checkingComponent | component, matching, graph | Проверяет компонент связности графа на возможность нахождения нового увеличивающего пути |
| findAnotherVertexInMatching | edges, vertex | Получает на вход вершину, которая находится в паросочетании и возвращает парную ей вершину относительно паросочетания. |
| containsEdgeWithVertex | path, vertex | Проверяет, есть ли в увеличивающем пути данная вершина |
| findOutgoingIndex | vertex, blossom, graph | Возвращает индекс вершины в цветке, инцидентной заданной вершине, следующей за цветком |
| findIngoingIndex | vertex, blossom, graph | Возвращает индекс вершины в цветке, инцидентной заданной вершине, идущей перед цветком |
| inverseAugPath | augPath, graph, matching | Инвертирует увеличивающий путь (увеличивает размер паросочетания на 1) |
| returnAugPath | forestParents, v, w | Возвращает увеличивающий путь от вершины v до вершины w |
| contractMatching | contracted, matching, blossom, contractedVertex | Сжатие паросочетания |
| getExposedVertexes | graph, blossomVertexes, matching, | Получение всех вершин текущего графа, не лежащих в паросочетании |

**graph – содержит класс Graph, описывающий граф, к которому применяется алгоритм**

*Таблица 4.*

| Поля | |
|---|---|
| **Имя** | **Назначение** |
| nodeMap | Хранит пары значение-вершина |
| adjacencyList | Для каждой вершины хранит множество инцидентных ей ребер |
| edgeSet | Совокупность всех ребер графа |
| contractedVertex | Последняя сжатая вершина |
| components | Компоненты связности графа |
| used | Массив, хранящий информацию о том, была ли вершина пройдена во время dfs |

*Таблица 5.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| dfs | v | Обход графа в глубину, начиная от вершины v |
| findComponents | | Нахождение компонент связности графа |
| size | | Возвращает количество вершин в графе |
| addEdgeByTwoVertexes | first_vertex, second_vertex | Добавляет в граф ребро с двумя данными вершинами |
| addEdge | edge | Добавляет заданное ребро в граф |
| addVertex | value | Добавляет в граф вершину с данным значением |
| getVertex | value | Возвращает вершину графа по ее значению |
| getAllEdges | | Возвращает все ребра графа |
| getAllVertexes | | Возвращает все вершины графа |
| getAdjacentEdges | v, matching | Возвращает все инцидентные вершине ребра, не входящие в паросочетание |
| getAdjacentVertexes | vertex | Возвращает все вершины, которые связаны с данной вершиной инцидентными ей ребрами |

| | | |
|---|---|---|
| getContractedVertex | | Возвращает текущую сжатую вершину |
| getEdge | first_vertex, second_vertex | Возвращает ребро графа по двум вершинам |
| removeEdge | first_vertex, second_vertex | Удаляет ребро графа по двум вершинам. |
| addContractedVertex | vertex | Присваивает новое значение текущей сжатой вершине |
| toString | | Возвращает строковое представление списка вершин графа |

**edge – содержит класс Edge, описывающий ребро графа**

*Таблица 6.*

| Поля | |
|---|---|
| **Имя** | **Назначение** |
| firstVertex | Первая вершина ребра |
| secondVertex | Вторая вершина ребра |

*Таблица 7.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| getAnotherEnd | first_vertex | Возвращает вторую вершину ребра |
| toString | | Возвращает строковое представление ребра |

**lifting**

*Таблица 8.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| liftPathWithBlossom | cy, augPath, blossom, notContractedGraph, contractedGraph | Разжимает цветок |

**vertex – содержит класс vertex, описывающий вершину графа**

*Таблица 9.*

| Поля | |
|---|---|
| **Имя** | **Назначение** |
| value | Значение, порядковый номер вершины |
| visible | Поле, которое определяет, является ли в данный момент вершина видимой. |

*Таблица 10.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| toString | | Возвращает строковое представление вершины |

**drawGraph**

*Таблица 11.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| main | | Создает лес и рисует граф по умолчанию в основной панели |
| makeFirstGraph | | Рисует граф по умолчанию в панели “Blossom graph” |

**Initialization**

*Таблица 12.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| init | | Инициализирует обработчики событий нажатия на кнопки в основном окне |
| graphGenerator | | Генерирует случайный граф |
| initial | | Инициализирует граф во всплывающем окне и обработчики событий кнопок всплывающего окна |

**Forest**

*Таблица 13.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| makeForest | | Инициализирует и вставляет в панель Forest пустой лес ребер и вершин |
| getForest | | Возвращает текущий лес ребер и вершин |
| initial | | Инициализирует граф во всплывающем окне и обработчики событий кнопок всплывающего окна |

**makeVisual**

*Таблица 14.*

| Методы | | |
|---|---|---|
| **Имя** | **Аргументы** | **Назначение** |
| colorAdjacentEdges | cy, adjacentEdges, matching | Раскрашивает все инцидентные данной вершине ребра, не входящие в паросочетание |
| finalColoring | cy, matching | Раскрашивает все вершины и ребра, входящие в паросочетание, красным цветом. |
| unColorEdge | selectedEdge, cy | Возвращает заданному ребру его первоначальный цвет |
| uncolorAdjacentEdges | cy, matching | Возвращает всем ребрам, которые не в паросочетании, их первоначальный цвет |
| drawAugmentingPath | augPath, cy | Изображает изменение увеличивающего пути с помощью анимации |
| colorExposedVertexes | vertexesToCheck, cy | Окрашивает все вершины, не входящие в паросочетание, в синий цвет |
| drawAddingToForest | v, w, x, cy | Добавляет вершины v, w, x в лес на панели “Forest” |
| drawRemovingEdge | firstVertex, secondVertex, cy | Скрывает заданное ребро |
| drawAddingEdge | firstVertex, secondVertex, cy | Добавляет ребро к нарисованному графу |

| | | |
|---|---|---|
| drawShowingEdge | firstVertex, secondVertex, cy | Показывает скрытое до этого ребро |
| drawAddingEdgeToForest | forest, firstVertex, secondVertex, forestRoots, forestParents, forestChild | Добавляет новое ребро на панель “Forest”, при этом рассчитывает позицию вершин ребра исходя из того, что нельзя нарушать древовидную структуру |
| drawAddingNodeToforest | Forest, vertex, size, number | Добавляет вершину в лес, при этом рассчитывая ее позицию на панели, исходя из количества уже добавленных вершин. |
| hideBlossomVertexes | blossomVertexes, cy, contractedVertex | Скрывает все вершины, принадлежащие цветку, кроме той, в которую цветок сжимается |

## ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

| Лист регистрации изменений | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Заменных | Новых | Аннулированных | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |