**Спецификация программы:**

Входные данные: граф, представленный списком ребер.

Программа выполняет функцию визуализации алгоритма поиска наибольшего паросочетания в двудольном графе (если граф не двудольный, программа выводит соответствующее сообщение).

**Программа выполняет следующую последовательность действий:**

1) Считывание входных данных;

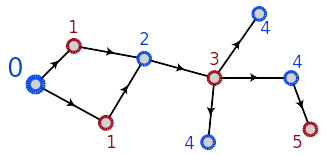
2) Проверка графа на двудольность;

3) Если проверка пройдена, выполняется визуализация алгоритма, иначе — вывод сообщения об ошибке;

4) Вывод результатов работы алгоритма.

**Алгоритм работы программы.**

1) **Проверка на двудольность:**

В каждой компоненте связности выбрать любую вершину и помечать оставшиеся вершины во время обхода графа в ширину поочередно, как четные и нечетные. Если при этом не возникает конфликта, то все нечетные вершины относятся к одной доле, а все четные — к другой.

2) **Поиск наибольшего паросочетания:**

Для поиска наибольшего паросочетания используем алгоритм Куна.

Сначала возьмём пустое паросочетание, а потом — пока в графе удаётся найти увеличивающую цепь, — будем выполнять чередование паросочетания вдоль этой цепи, и повторять процесс поиска увеличивающей цепи. Как только такую цепь найти не удалось — процесс останавливаем, — текущее паросочетание и есть максимальное.

Поиск увеличивающей цепи осуществляется с помощью специального обхода в глубину или ширину (обычно в целях простоты реализации используют именно обход в глубину). Изначально обход в глубину стоит в текущей ненасыщенной вершине *v* первой доли. Просматриваем все рёбра из этой вершины, пусть текущее ребро — это ребро *(v,to).* Если вершина *to* ещё не насыщена паросочетанием, то, значит, мы смогли найти увеличивающую цепь: она состоит из единственного ребра *(v,to)*; в таком случае просто включаем это ребро в паросочетание и прекращаем поиск увеличивающей цепи из вершины *v*. Иначе, — если *to* уже насыщена каким-то ребром *(p,to)* то попытаемся пройти вдоль этого ребра: тем самым мы попробуем найти увеличивающую цепь, проходящую через рёбра *(v,to), (to,p).* Для этого просто перейдём в нашем обходе в вершину *p* — теперь мы уже пробуем найти увеличивающую цепь из этой вершины.

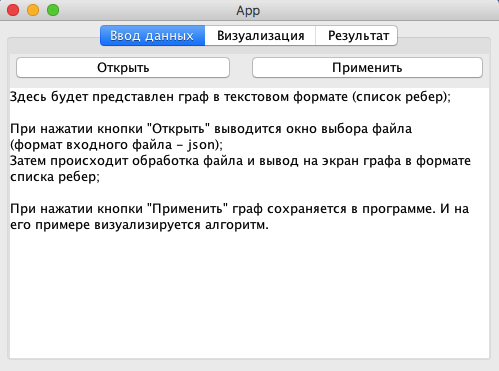
**Время работы**

Итак, алгоритм Куна можно представить как серию из *n* запусков обхода в глубину/ширину на всём графе. Следовательно, всего этот алгоритм исполняется за время *O(nm)*, что в худшем случае есть *O(n3)*.

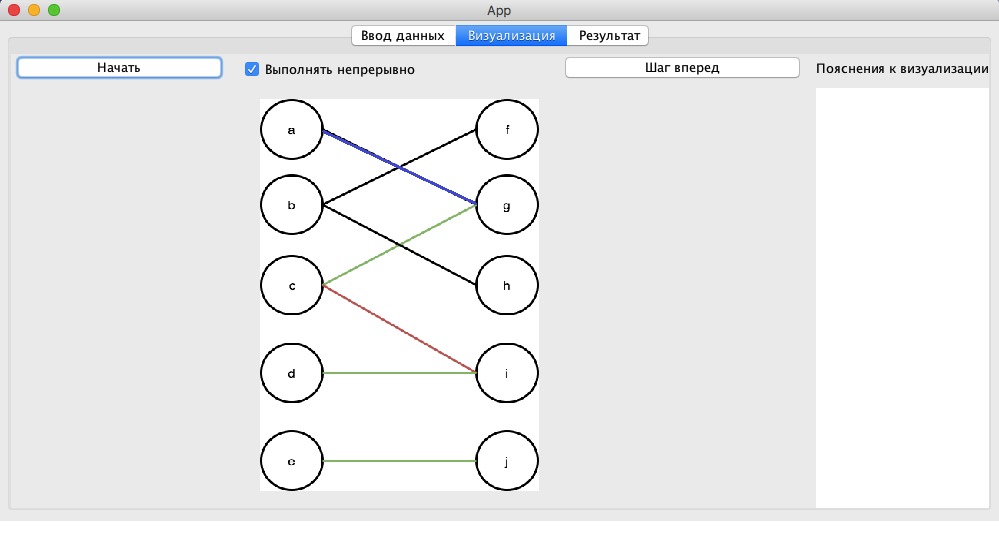
**Эскизы интерфейса.**

Приложение делится на 3 вкладки:

1) Ввод данных и вывод графа на экран



2) Визуализация алгоритма



После проверки на двудольность, доли графа выводятся на экран в два столбца и отрисовываются ребра между ними.

Во время визуализации ребра графа помечаются разными цветами:

1. Черные – еще не рассмотренные ребра
2. Красные – рассмотренные ребра, не вошедшие в паросочетание
3. Зеленые – рассмотренные ребра, вошедшие в паросочетание
4. Синее – текущее рассматриваемое ребро

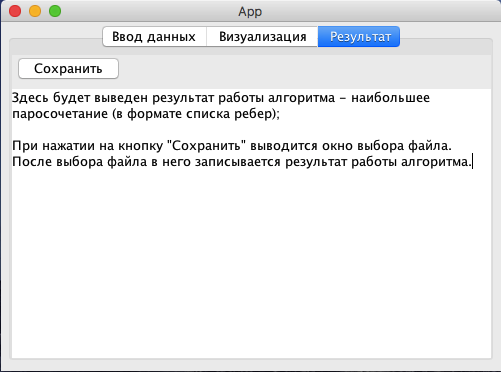
При нажатии на кнопку "Начать" будет отрисован исходный граф. Если выбрана опция "Выполнять непрерывно", будет итерации будут выполняться автоматически с выводом соответствующих пояснений.

При нажатии на кнопку "Вперед" будет выполнена следующая итерация алгоритма с перерисовкой графа и выводом соответствующих пояснений.

Выводимые пояснения к визуализации:

1. Текущее рассматриваемое ребро
2. Добавлено (удалено) ли рассматриваемое ребро в паросочетание или нет
3. Текущее паросочетание

3) Вывод наибольшего паросочетания и сохранение результатов



**План работы**

23.06.2017 - Подготовка спецификации программы. Распределение ролей. Представление прототипа интерфейса.

26.06.2017 - Демонстрация интерфейса. Реализация ввода данных, чтения из файла. Код алгоритма.

28.06.2017 – Визуализация работы алгоритма (подсветка текущего ребра). Реализация вывода данных, запись в выходной файл. Тестирование.

30.06.2017 – Полная подсветка ребер. Добавление пояснений к визуализации. Исправление ошибок. Выпуск продукта. Написание отчета.

**Распределение ролей**

Допира В. Е. - ответственный за итерацию (реализация ввода/вывода из файла и тестирование).

Лянгузов А. А. - разработчик (разработка интерфейса и визуализация алгоритма).

Ярошенко М. А. - разработчик (реализация алгоритма на языка Java).