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

ОТЧЕТ

по учебной практике

Тема: Реализация алгоритма Прима построения МОД на языке Java с визуализацией

Студентка гр. 2382 _____ Семененко А.С.

Студент гр. 2300 _____ Жохов К.С.

Студент гр. 2300 _____ Рогожин К.Д.

Руководитель _____ Шестопалов Р.П.

Санкт-Петербург
2024

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студентка Семененко А.С. группы 2382

Студент Жохов К.С. группы 2300

Студент Рогожин К.Д. группы 2300

Тема практики: Командная разработка визуализатора алгоритма Прима построения МОД с графическим интерфейсом.

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: алгоритм Прима построения МОД.

Сроки прохождения практики: 26.06.2024 – 09.07.2024

Дата сдачи отчета: 08.07.2024

Дата защиты отчета: 08.07.2024

Студентка	_____	Семененко А.С.
Студент	_____	Жохов К.С.
Студент	_____	Рогожин К.Д.
Руководитель	_____	Шестопалов Р.П.

АННОТАЦИЯ

Целью проекта является развитие навыков программирования на языке Java с последующей реализацией алгоритма Прима построения минимального остовного дерева в связном взвешенном (положительные целочисленные веса) неориентированном графе. Для удобного использования приложения разработан графический интерфейс.

SUMMARY

The aim of the project is to develop programming skills in the Java language with the subsequent implementation of the algorithm for constructing a minimum spanning tree in a connected weighted (positive integer weights) undirected graph. A graphical interface has been developed for convenient use of the application.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель: разработать программу на ЯП Java, реализующую алгоритм Прима построения МОД (для взвешенного (положительные целочисленные веса) связного неориентированного графа), содержащую графический интерфейс.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Продумать логику считывания, хранения графа.
2. Проверить пригодность заданного пользователем графа для реализуемого алгоритма.
3. Реализовать графический интерфейс для удобства использования программы.
4. Проверить работоспособность программы.

Теоретическое описание алгоритма Прима [2]: на вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость. Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

Псевдокод:

```
// Граф G = (E, V)
// Построение дерева T = (E_T, V_T)
```

// E_T – рёбра, принадлежащие T

// V_T – вершины, принадлежащие T

// Начинаем с произвольной вершины s

$\boldsymbol{V_T} \leftarrow \{s\}$ // множество вершин, принадлежащих T

// Пока существуют вершины, не принадлежащие дереву T

while ($\exists v \in V \mid v \notin V_T$) do

    // Из ребер, соединяющих вершины $\boldsymbol{T}$ с вершинами графа $\boldsymbol{G}$,

    // выбираем ребро минимального веса

    $\boldsymbol{e} \leftarrow \min \{w_{i,j} \mid (v_i, v_j) \in \boldsymbol{E} \mid v_i \in V_T, v_j \notin V_T\}$

    // Включаем его в дерево вместе с его инцидентной

    // вершиной, не входящей в $\boldsymbol{T}$

    $\boldsymbol{E_T} \leftarrow \boldsymbol{E_T} \cup \{e\}$

    $\boldsymbol{V_T} \leftarrow \boldsymbol{V_T} \cup \{v_j\}$

end while

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1 Требования к вводу исходных данных

Пользователю предлагается 3 способа задания графа (нажав на кнопку со знаком вопроса, можно прочесть инструкцию по использованию приложения):

1) Drawing Input - задание графа с помощью курсора мыши.

2) Matrix Input - задание графа с помощью матрицы смежности.

3) File Input - задание графа с помощью матрицы смежности, расположенной в файле.

1) Drawing Input

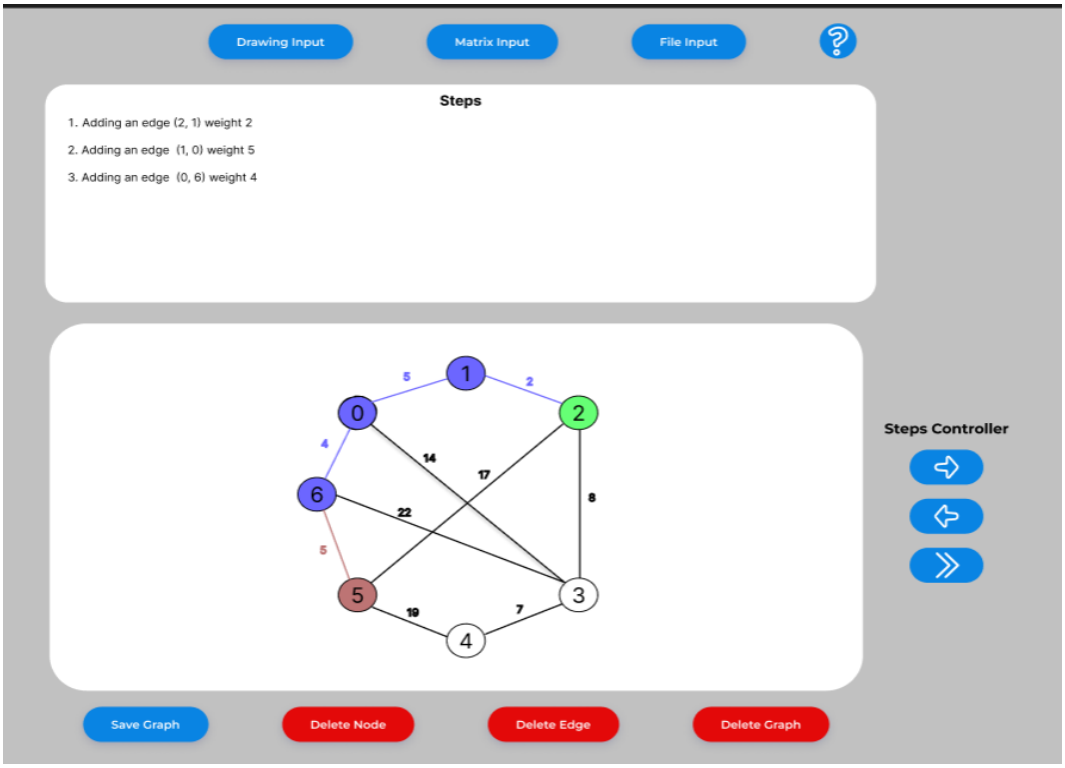

Рисунок 1 – Прототип Drawing Input

Пользователь задаёт граф, используя мышь (ЛКМ - добавить вершину, ПКМ - выбрать вершину инцидентную ребру). При добавлении ребра пользователю будет предложено задать его вес. Также есть возможность удалить граф, удалить отдельную вершину с инцидентными ей рёбрами или

удалить ребро с помощью кнопок Delete Graph, Delete Node и Delete Edge соответственно. Можно перемещать вершину графа, зажав её ЛКМ. При двойном нажатии ПКМ на ребро можно изменить его вес.

2) Matrix Input:

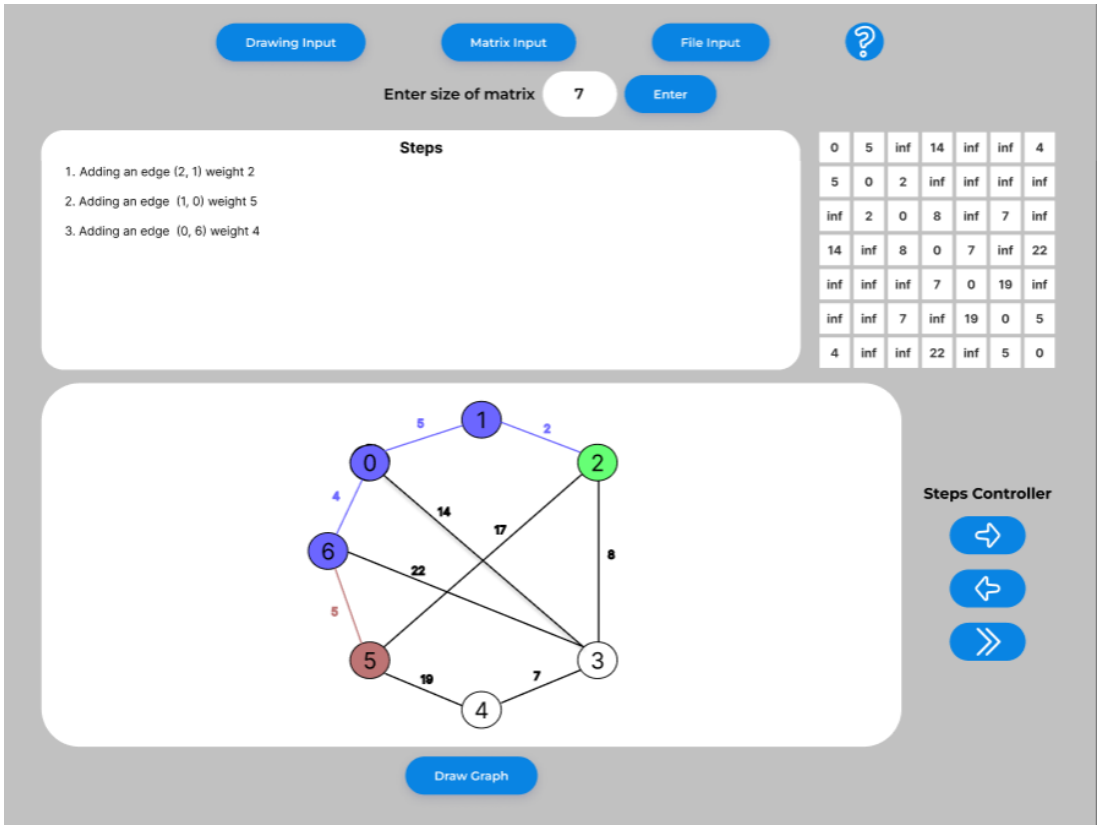

Рисунок 2 – Прототип Matrix Input

Пользователь задаёт размер матрицы смежности, затем заполняет её в соответствии с требованиями: должна быть симметричной, содержать неотрицательные веса рёбер.

3) File Input:

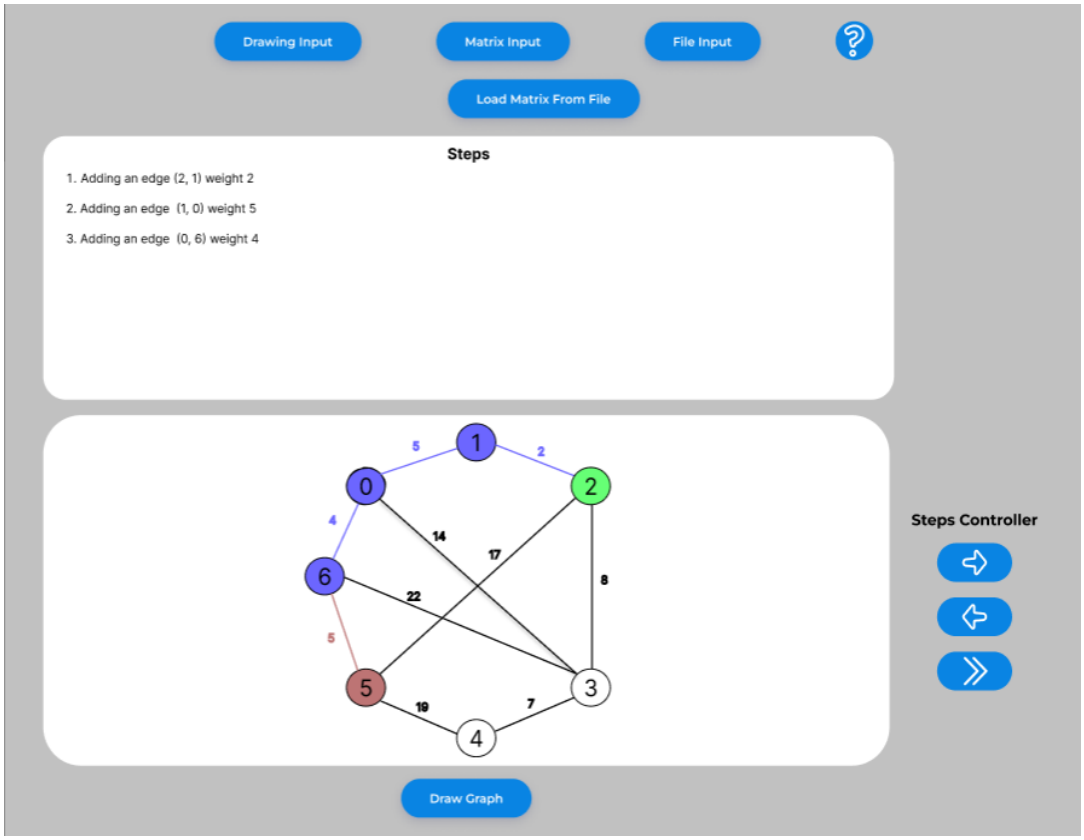

Рисунок 3 – Прототип File Input

После нажатия клавиши File Input программа переключается в режим работы с файлом, далее с помощью кнопки Load Matrix From File запускается файловый менеджер, с помощью которого выбирается файл, который должен содержать матрицу смежности, отформатированную в соответствии с требованиями: веса в матрице разделены пробелами, строки матрицы разделены переносом строки.

1.1.2 Требования к визуализации

После того, как пользователь задал граф пригодный для работы алгоритма Прима и нажал кнопку «Save Graph» (в режиме Drawing Input) или «Draw Graph» (в режимах Matrix Input, File Input), начинается процесс отображения работы алгоритма.

1) В верхнем окошке выводится информация о последнем шаге алгоритма (список выбираемых рёбер на последнем выполненном шаге, выбранное ребро,

список всех выбранных рёбер, включая последний шаг). На последнем шаге алгоритма выводится суммарный вес построенного минимального остовного дерева.

2) В режиме Matrix Input в правом верхнем углу отображается заданная пользователем матрица смежности.

3) В нижнем окошке отображается заданный пользователем граф. Для наглядного представления процесс работы алгоритма сопровождается выделением цветом вершин и рёбер, принадлежащих минимальному остовному дереву.

4) С помощью клавиш со стрелками выполняется пошаговая отрисовка алгоритма Прима, для быстрого перехода к конечному результату используется двойная стрелка. Во время пошаговой работы алгоритма выбранные на предыдущих шагах вершины и ребра окрашиваются для наглядного представления, ребро и вершина, выбранные на текущем шаге, также выделяются цветом.

1.2. Уточнение требований после сдачи прототипа.

Добавить возможность перемещения вершин с помощью мыши.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

01.07.24 - сдача вводного задания, согласование спецификации, демонстрация эскиза графического интерфейса.

03.07.24 - сдача прототипа (приложение, демонстрирующее интерфейс и частично реализующее основные функции).

05.07.24 - сдача бета-версии приложения (реализует работу алгоритма с визуализацией).

08.07.24 – релиз (протестированное и отлаженное приложение с ликвидированными недочётами, возникшими по итогам защиты бета-версии), сдача отчёта по проекту.

2.2. Распределение ролей в бригаде

Семененко Анна - разработка интерфейса приложения.

Жохов Кирилл - реализация алгоритма, отчётная документация.

Рогожин Константин - разработка эскиза интерфейса, тестирование.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Код программы представлен в репозитории бригады [1].

Для создания графических элементов и управления пользовательским интерфейсом в программе применялась библиотека Java Swing (Oracle, 2022) [3].

Класс *Edge*

Используется для представления ребер в графе, содержит три основных поля:

int src – начальный узел.

int dest – конечный узел.

int weight – вес ребра.

Класс *PrimAlgorithm*

Реализует пошаговый алгоритм Прима для построения минимального остовного дерева (МОД) из графа, представленного матрицей смежности.

Конструктор *PrimAlgorithm(int[][] matrix, int size)* - принимает матрицу смежности и количество вершин графа. Эти данные используются для инициализации внутреннего состояния объекта *PrimAlgorithm*.

Поля класса:

matrix – двумерный массив int, представляющий матрицу смежности исходного графа.

size – целочисленная переменная, хранящая количество вершин в графе.

Поле *Map<Edge, String> map* – ассоциативный массив, где ключ — ребро МОД, выбранное на текущем шаге, а значение — информация о текущем шаге.

Метод *primMST()* - *о*сновной метод класса, реализующий пошаговый алгоритм Прима. Метод возвращает *Map<Edge, String>*, где каждое ребро МОД ассоциировано с информацией о шаге, на котором было выбрано.

Класс *MatrixValidation*

Предназначен для валидации матриц смежности, которые используются для представления графов. Этот класс содержит методы проверки матрицы на наличие только положительных значений, отсутствие петель, симметричность и связность графа, чтобы убедиться, что она соответствует ожидаемым требованиям для корректного представления графа.

Поля класса:

Поле *matrix* – двумерный массив int, представляющий матрицу смежности исходного графа.

Методы класса:

checkMatrix(int[][] matrix) – главный метод, который вызывает другие методы для проверки различных условий матрицы. В зависимости от результатов проверок, метод возвращает код ошибки или 0, если все проверки пройдены успешно.

checkValues(int[][] matrix) – *п*роверяет, что все значения в матрице являются положительными числами и что граф не имеет петель.

checkSymmetry(int[][] matrix) – проверяет, является ли матрица симметричной относительно главной диагонали.

checkConnectivity(int[][] matrix) – проверяет, является ли граф связным.

Класс *GraphPanel*

Наследует класс *JPanel* [3] и представляет собой панель для рисования графа на основе матрицы смежности. Этот класс позволяет отображать граф в виде набора вершин, расположенных вокруг центральной точки, и ребер, соединяющих эти вершины.

Конструктор *GraphPanel(int[][] matrix, int vertexCount)* – принимает матрицу смежности и количество вершин в графе. На основе этих данных инициализируется внутреннее состояние объекта *GraphPanel*.

Поля класса:

matrix – двумерный массив int, представляющий матрицу смежности исходного графа.

vertexCount – целочисленная переменная, хранящая количество вершин в графе.

points – массив Point, содержит координаты вершин графа.

Методы класса:

paintComponent(Graphics g) - переопределенный метод *paintComponent* отвечает за отрисовку графа на панели. В этом методе рассчитываются координаты вершин на основе количества вершин и размеров панели, затем рисуются вершины и ребра графа.

drawEdge(Graphics g, int i, int j, Color oldColor, Color newColor) – используется для отрисовки ребра между двумя вершинами с возможностью изменения цвета ребра и вершин. Этот метод полезен для динамического изменения вида графа во время выполнения программы.

Класс *GraphDrawingPanel*

Представляет собой графическую панель, которая используется для визуализации графа путем добавления и удаления вершин и ребер на панели, а также применять алгоритм Прима для поиска МОД. В классе реализованы функции для добавления вершин и ребер, удаления вершин и ребер, изменения веса ребра, построения минимального остовного дерева (МОД) по алгоритму Прима и анимации этого процесс. В классе также реализованы обработчики нажатия кнопок для сохранения графа, удаления вершин и ребер, а также обработчики для движения мыши и кликов мыши.

Конструктор *GraphDrawingPanel()* – инициализирует панель, устанавливает макет и добавляет компоненты для рисования графа, управления процессом вычисления МОД и других операций, таких как сохранение графа, удаление узлов и ребер.

Поля класса:

ArrayList<Point> points – массив вершин графа.

ArrayList<Edge> edges – массив ребер графа.

Point tempRightPoint – переменная для хранения временно выбранной вершины.

tempRightPointIndex – целочисленная переменная, хранит индекс временно выбранной вершины.

Point *draggedPoint* – хранит вершину, которая перетаскивается мышью.

draggedPointIndex – целочисленная переменная, хранит индекс перетаскиваемой вершины.

Edge tempEdge – переменная, которая временно хранит выбранное ребро.

graph – двумерный массив int, использован для хранения структуры графа.

RoundedTextArea outputArea – панель для вывода результатов выполнения алгоритма Прима.

RoundedPanel drawingPanel – панель для рисования графа.

PADDING – статическая константа, определяет отступы от краев панели.

currentStep – целочисленная переменная, отслеживает текущий шаг алгоритма Прима.

List<Edge> edgesMST – список содержащий ребра минимального остовного дерева (МОД).

List<String> messages – список содержащий сообщения о каждом ребре МОД.

Методы:

isWithinDrawingArea(int x, int y) – проверяет, находится ли указанная точка внутри допустимого диапазона рисования.

findPointIndex(Point point) – ищет индекс точки среди списка точек.

isPointOnEdge(Point point) – проверяет, находится ли точка на ребре графа.

StepForward() – переход к следующему шагу алгоритма Прима, отображая соответствующее ребро и сообщение.

StepBack() – возврат к предыдущему шагу алгоритма Прима, удаляя последнее отображенное ребро.

drawEdge(int i, int j, Color oldColor, Color newColor) – метод для рисования ребра между двумя вершинами графа (i и j) с возможностью изменения цвета ребра и вершин. Также метод обновляет отображение номеров вершин и перерисовывает панель.

SaveGraph() – сохраняет текущее состояние графа в матрицу смежности и выполняет алгоритм Прима для построения минимального остовного дерева (МОД). После выполнения алгоритма, метод обновляет списки ребер МОД и сообщений, связанных с ними.

calculateMST() – используется для немедленного отображения МОД на основе ранее сохраненных данных. Перебирает через список ребер МОД и вызывает *drawEdge()* для каждого ребра, обновляя отображение графа и вывод сообщений.

deleteGraph() – удаляет весь граф, очищая списки вершин и ребер, сбрасывая временные переменные и обновляя отображение.

deleteNode() – удаляет выбранную вершину из графа, удаляя ее из списка вершин и всех связанных с ней ребер.

deleteEdge() – *у*даляет выбранное ребро из списка ребер графа.

Класс *AdjacencyMatrixPanel*

Предоставляет интерфейс пользователя для ввода матрицы смежности, отображения графа и выполнения алгоритма Прима для построения минимального остовного дерева (МОД). Он наследует класс *JPanel* [3] и содержит панель для ввода матрицы, панель для отображения графа, а также панель сообщений о шагах алгоритма Прима.

Конструктор *AdjacencyMatrixPanel()* – *и*нициализирует панель, устанавливает её макет и добавляет необходимые компоненты для ввода размера матрицы, отображения графа и управления процессом вычисления МОД.

Поля класса:

RoundedTextField[][] matrixFields – двумерный массив текстовых полей, через которые пользователь заполняет матрицу смежности.

RoundedTextArea outputArea – текстовое поле для вывода результатов выполнения алгоритма Прима.

RoundedPanel matrixPanel – панель для матрицы смежности.

RoundedPanel graphPanel – панель для отображения графа.

List<Edge> edges – массив для хранения ребер графа.

List<String > messages – массив для хранения сообщений о каждом добавленном ребре.

currentStep – целочисленная переменная для отслеживания текущего шага алгоритма Прима.

Методы класса:

createMatrix(int size) – создает матрицу заданного размера, заполняя её полями ввода *RoundedTextField*. Устанавливает слушатели для документов этих полей, чтобы реагировать на изменения.

*addListenerToMatrixCells() – д*обавляет слушателей к документам всех полей ввода матрицы, чтобы отслеживать изменения и обновлять граф при необходимости.

*calculateMST() – в*ыполняет алгоритм Прима для построения МОД, если граф уже был нарисован. Выводит результаты в *outputArea*.

*drawGraph() – р*исует граф на основе введенной матрицы смежности. Проверяет валидность матрицы и выполняет алгоритм Прима для построения МОД.

*StepForward() – п*ереходит к следующему шагу алгоритма Прима, отображая соответствующее ребро и сообщение.

*StepBack() – в*озвращается к предыдущему шагу алгоритма Прима, удаляя последнее отображенное ребро.

Класс *AdjMatrixFromFilePanel*

Предоставляет интерфейс пользователя для работы с матрицей смежности, полученной из файла, и визуализации графа. Он наследует класс *JPanel* [3] и содержит поля для матрицы смежности, панели для отображения графа, панели для кнопок управления, а также списки для хранения ребер минимального остовного дерева (МОД) и сообщений о шагах алгоритма Прима.

Конструктор *AdjMatrixFromFilePanel ()* – *и*нициализирует панель, устанавливает её макет и добавляет необходимые компоненты для ввода размера матрицы, отображения графа и управления процессом вычисления МОД.

Поля класса:

matrix – двумерный массив int, представляющий матрицу смежности графа.

RoundedTextArea outputArea – поле для вывода результатов выполнения алгоритма Прима.

RoundedPanel graphPanel – поле для отображения графа.

size - целочисленная переменная, хранящая размер матрицы.

GraphPanel graphDraw – поле для отображения графа.

List<Edge> edges – список содержащий ребра минимального остовного дерева (МОД).

List<String> messages – *список* содержащий сообщения о каждом ребре МОД.

currentStep – целочисленная переменная, отслеживающая текущий шаг алгоритма Прима.

Методы класса:

loadMatrixFromFile() – метод для загрузки матрицы смежности из файла. Использует *JFileChooser* для выбора файла, считывает данные из файла и заполняет матрицу смежности.

calculateMST() – *в*ыполняет алгоритм Прима для построения МОД, если граф уже был нарисован. Отображает результаты в *outputArea*.

*drawGraph() – р*исует граф на основе введенной матрицы смежности. Проверяет валидность матрицы и выполняет алгоритм Прима для построения МОД.

*StepForward() – п*ереход к следующему шагу алгоритма Прима, отображая соответствующее ребро и сообщение.

*StepBack() – в*озвращение к предыдущему шагу алгоритма Прима, удаление последнего отображенного ребра.

Класс *PrimGUI*

Представляет собой графический интерфейс пользователя для алгоритма Прима. Он наследует класс *JFrame* и содержит основную панель *mainPanel*, которая является контейнером для других панелей, таких как *matrixPanel*, *drawingPanel* и *filePanel*. *CardLayout* используется для управления отображением этих панелей. Класс также содержит кнопки для переключения между панелями и обработчики событий для этих кнопок.

Поля класса:

CardLayout cardLayout – объект, используется для переключения между разными панелями в одном контейнере.

JPanel mainPanel – основная панель, которая содержит различные панели для ввода данных и отображения результатов.

matrixPanel, drawingPanel, filePanel – специализированные панели для ввода матрицы смежности, рисования графа и чтения матрицы из файла соответственно.

JPanel buttonPanel – панель, содержащая кнопки для переключения между различными режимами ввода.

Конструктор *PrimGUI()* – инициализирует окно, устанавливает его свойства, такие как заголовок, размер, поведение при закрытии и расположение. Также создаются и настраиваются панели для ввода данных и отображения результатов, а также кнопки для переключения между этими панелями.

Методы класса:

showHelpWindow() – отображает окно с помощью, содержащее инструкции по использованию приложения.

main(String[] args) - *с*татический метод *main* запускает GUI в потоке обработки событий *Swing [3]*, создавая экземпляр *PrimGUI* и делая его видимым.

4. ТЕСТИРОВАНИЕ

1) Тестирование расстановки вершин в Drawing Input.

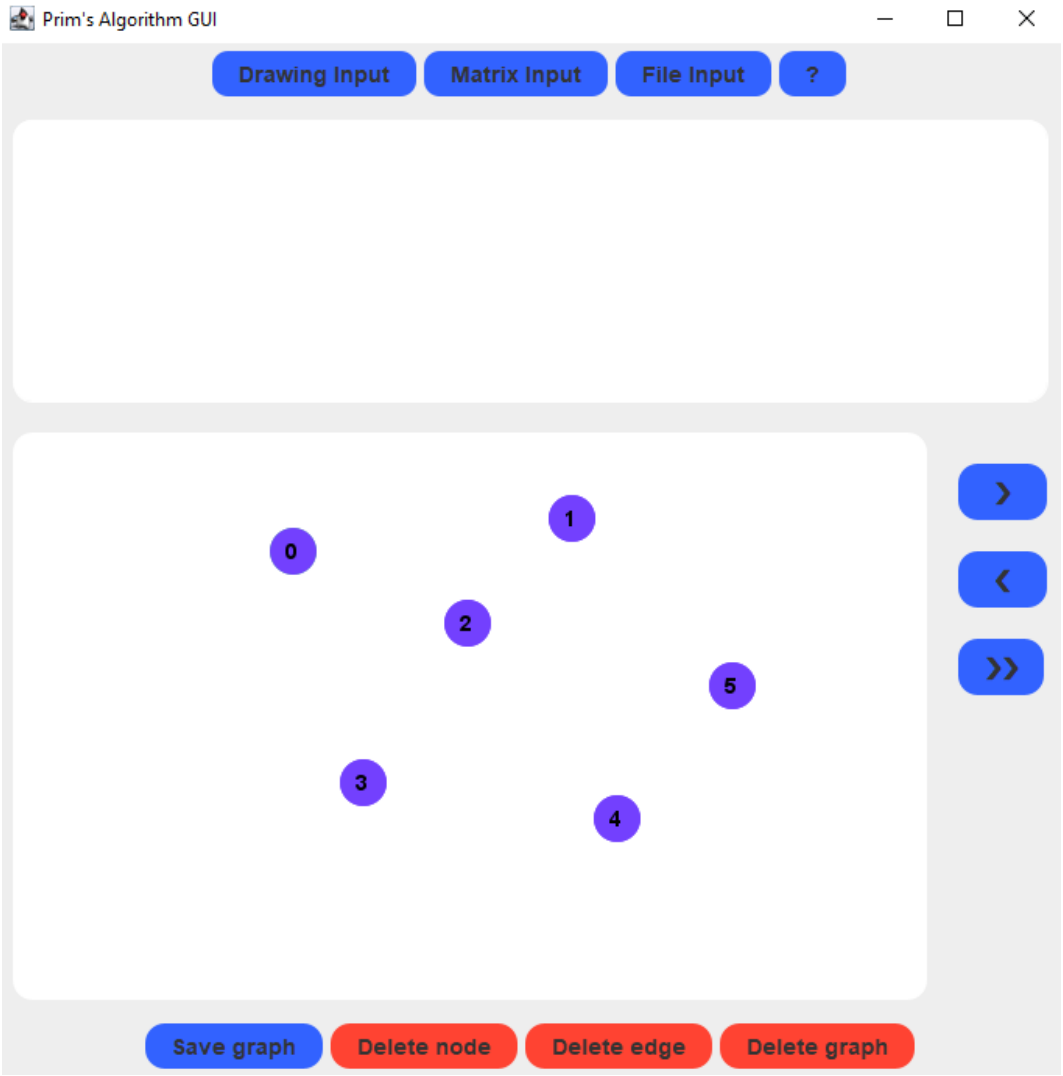

Рисунок 4 – Проверка корректности расстановки вершин

2) Тестирование смены индексации вершин при удалении промужуточной (3 вершины).

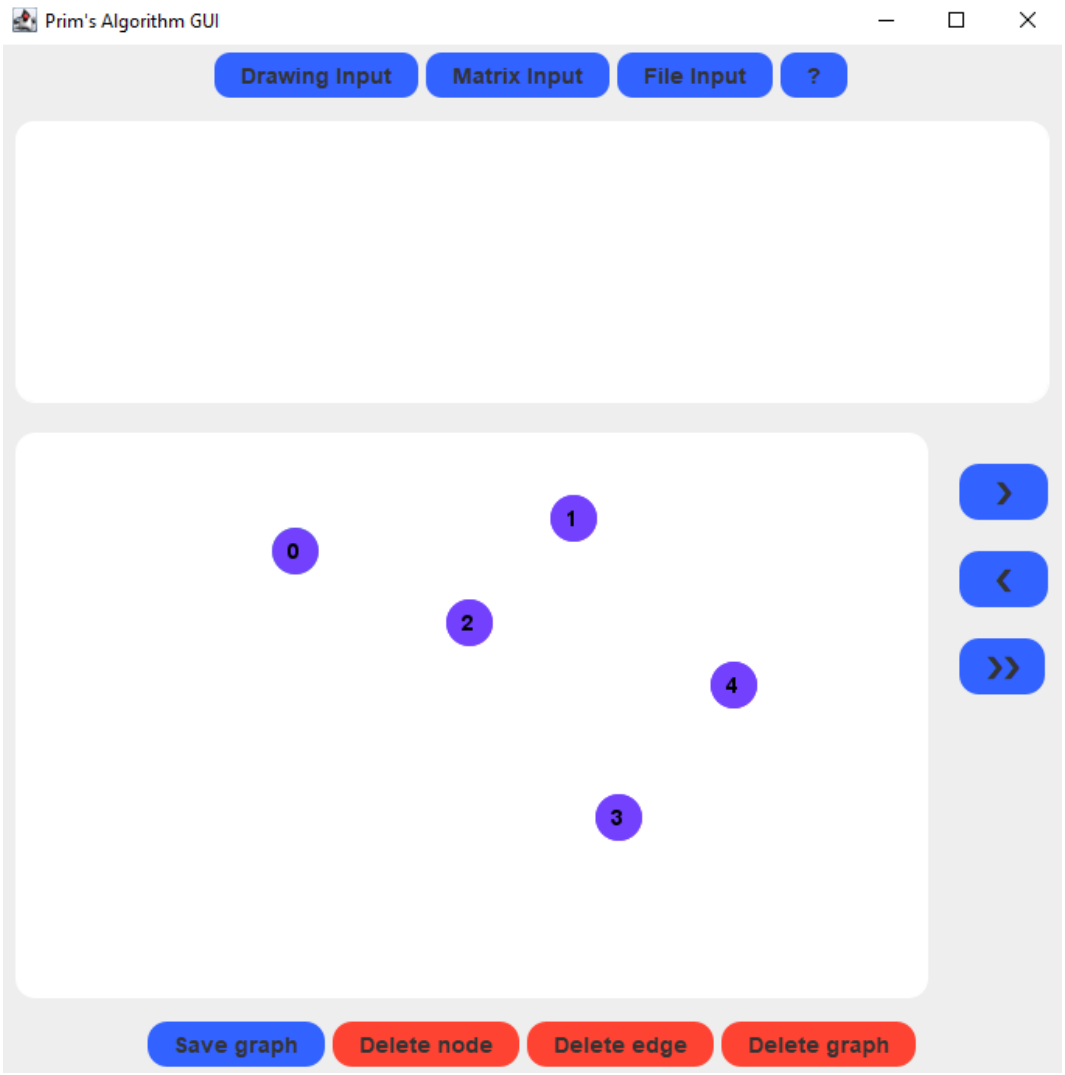

Рисунок 5 – Удаление промежуточной вершины

3) Тестирование корректной работы индексации при добавление новой вершины после того, как было выполненено удаление.

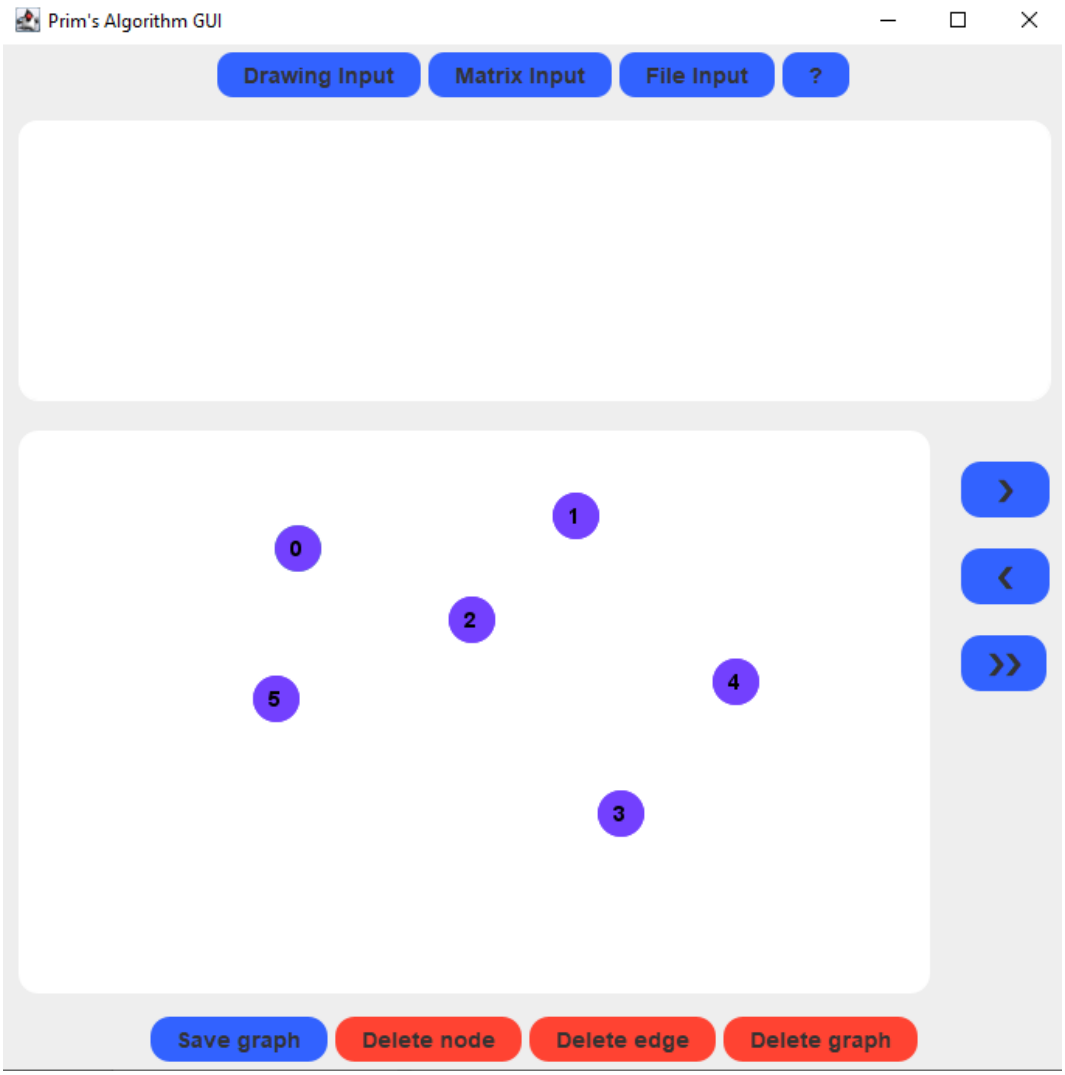

Рисунок 6 - Корректная работа индексации

4) Тестирование задания рёбер и их весов. Проверка на возможность создания петель.

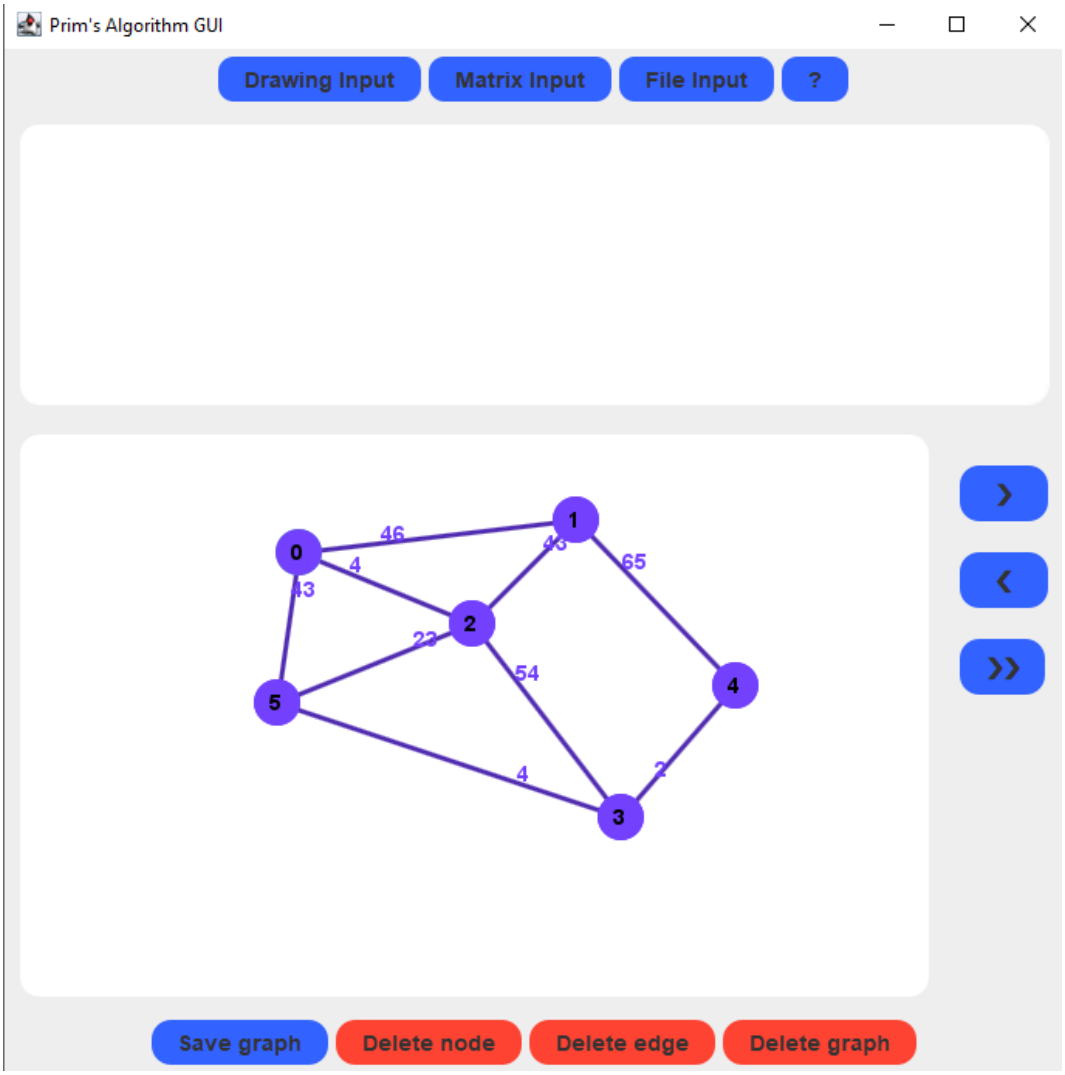

Рисунок 7 – Проверка работы расстановки рёбер

5) Тестирование удаления ребра графа.

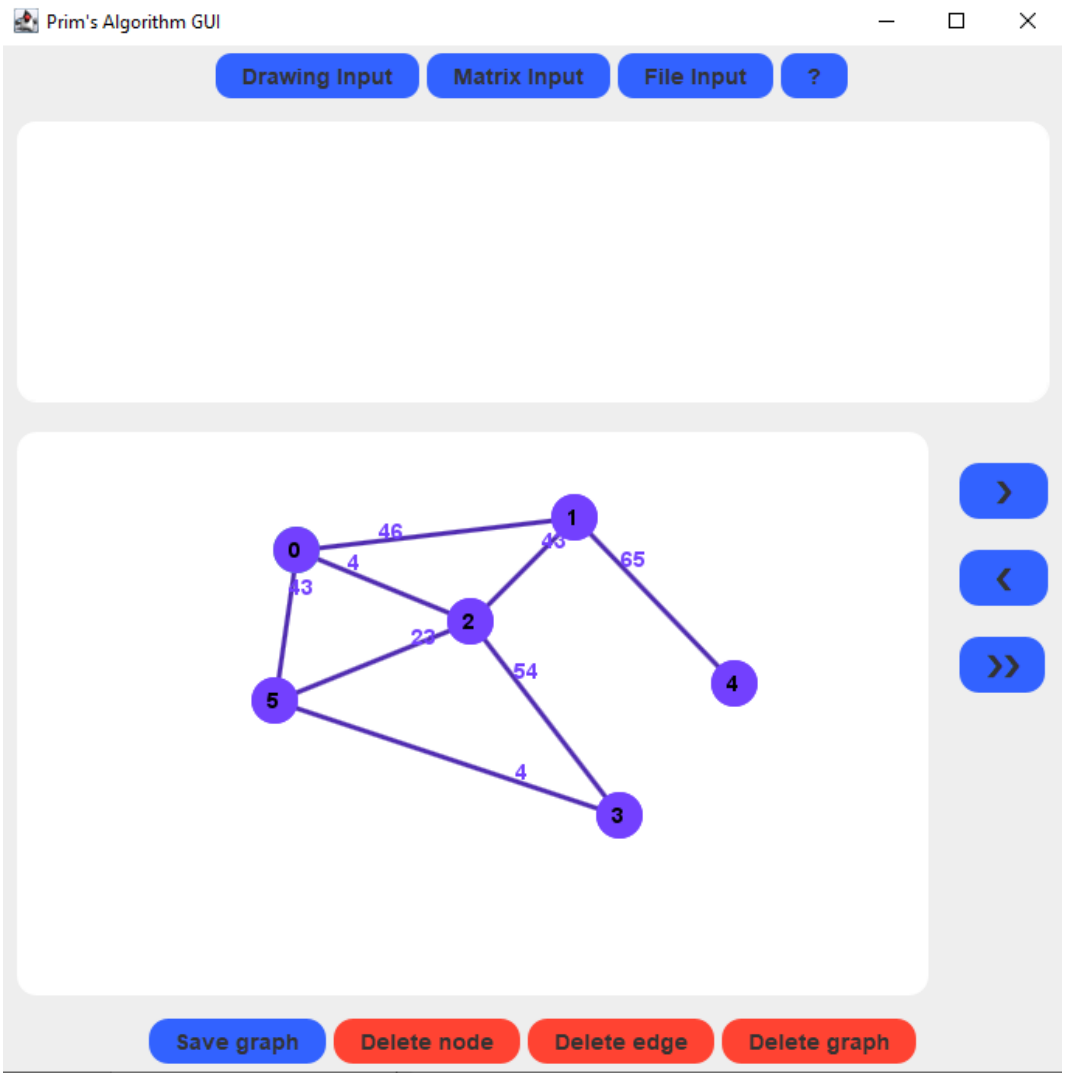

Рисунок 8 – Удаление ребра

6) Тестирование пошагового выполнения алгоритма.

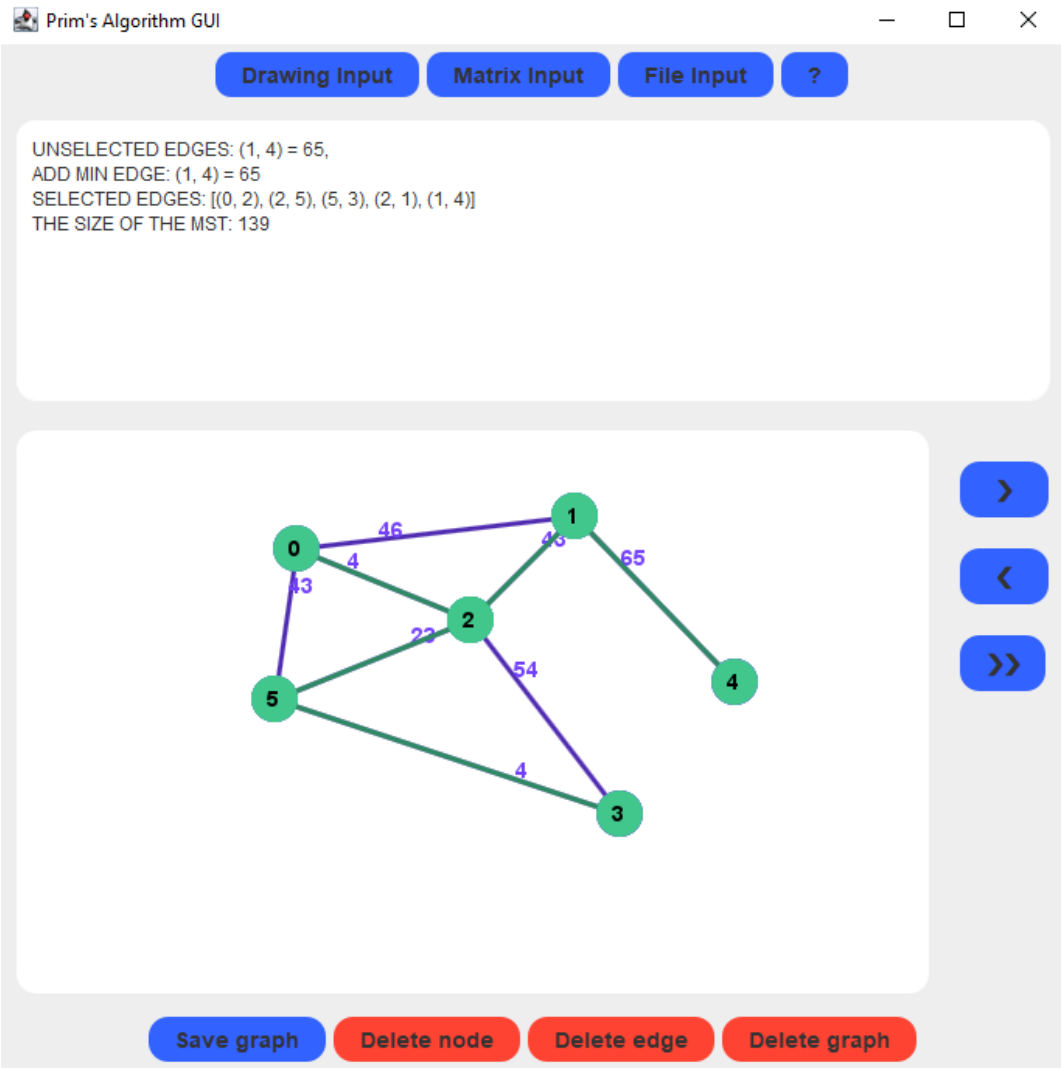

Рисунок 9 – Построение МОД

7) Тестирование на несвязном графе. Для графа была добавлена вершина для создания несвязного графа. Получение пользователем сообщения об ошибке.

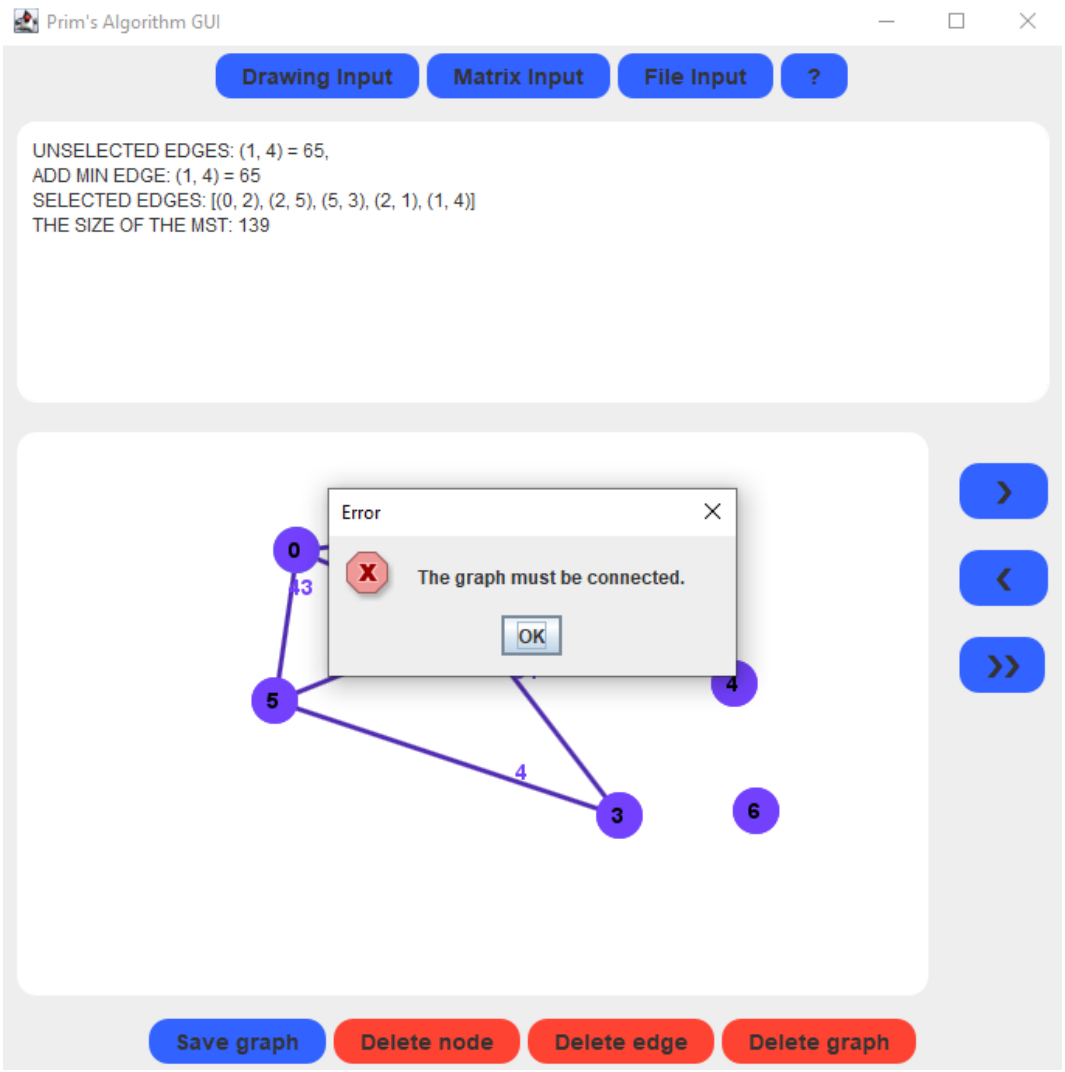

Рисунок 10 – Проверка на связность

8) Тестирование корректности работы программы с обновленным графом после того, как алгоритм отработал для старой версии графа.

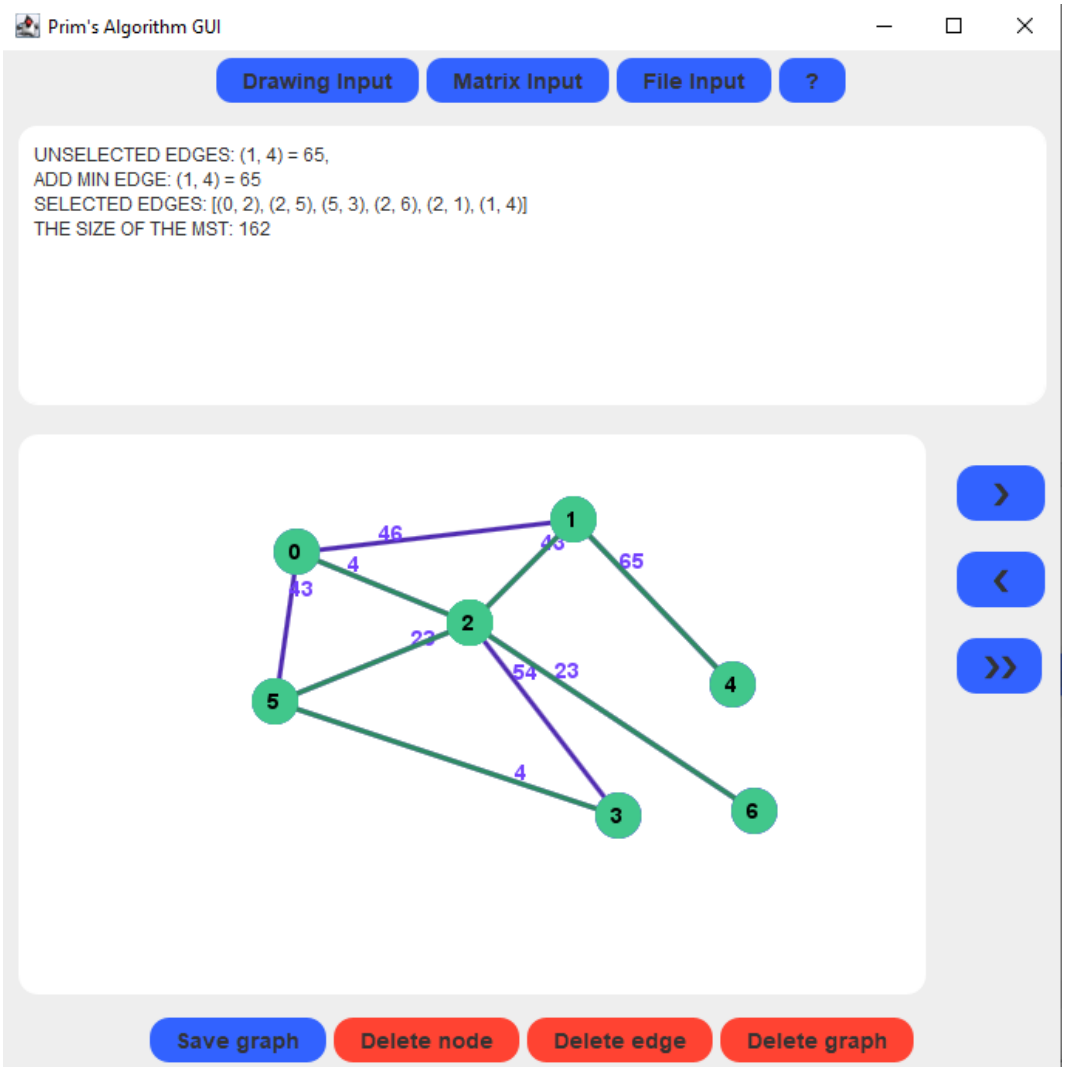

Рисунок 11 – Работа алгоритма после обновления графа

9) Тестирование работы пошагового возврата в одно из состояний отрисовки МОД.

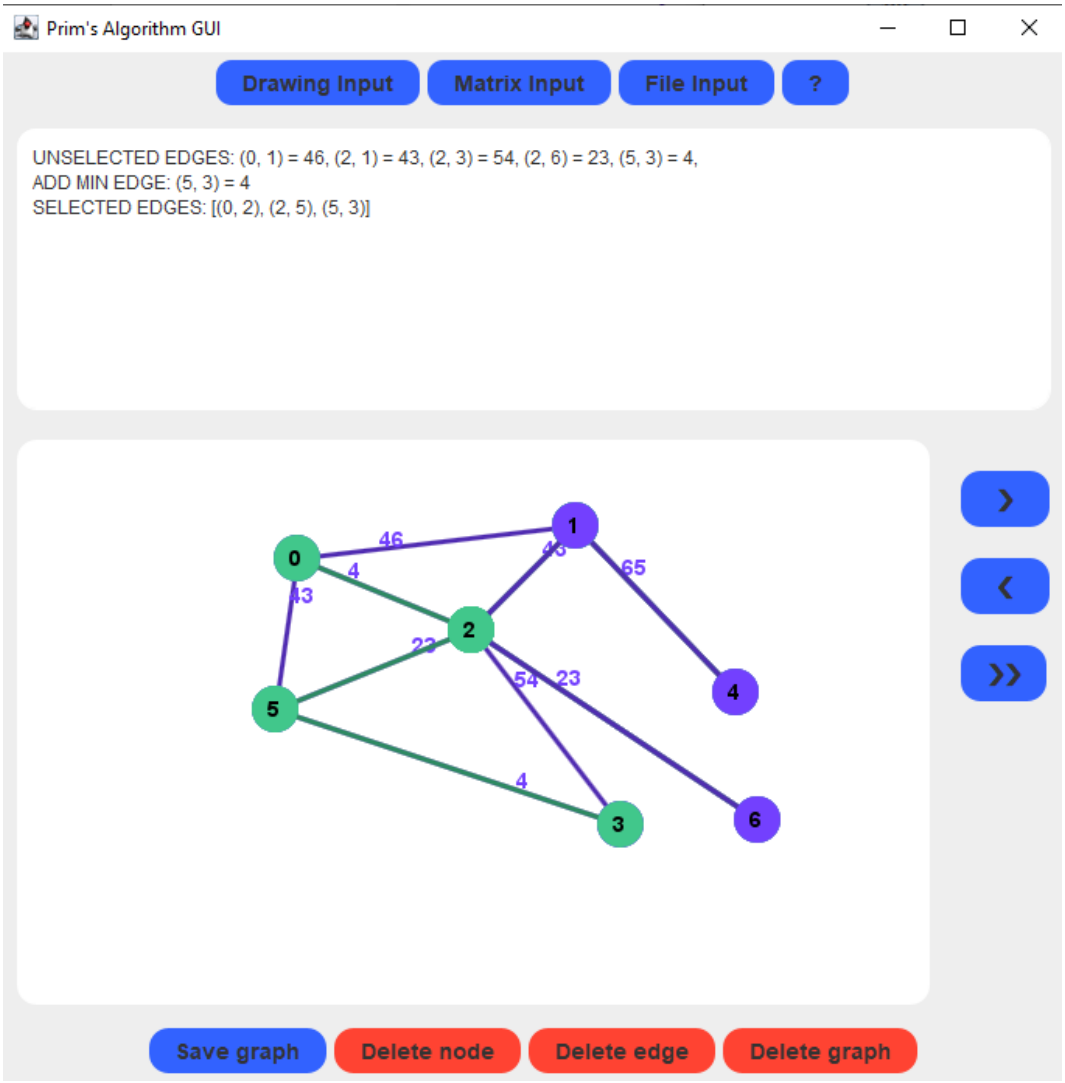

Рисунок 12 – Пошаговый возврат в промежуточное состояние

10) При нажатии на кнопку «?» появляется инструкция по использованию приложения (рисунок 13).

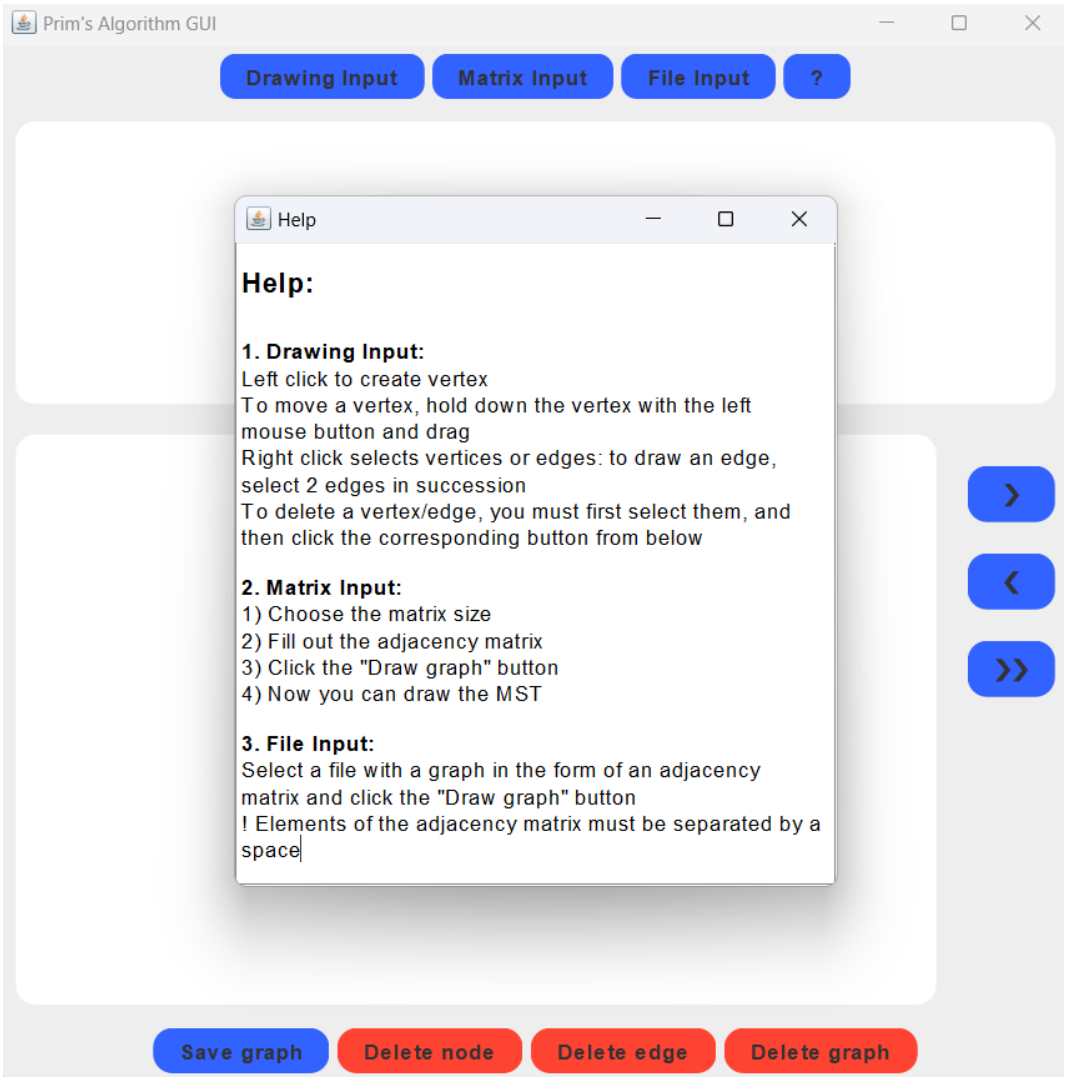

Рисунок 13 - Инструкция по использованию приложения

11) При попытке задать матрицу смежности для несвязного графа пользователь получит оповещение о некорректности задаваемого им графа. Эта же ситуация обрабатывается при задании графа из файла (Рисунок 14).

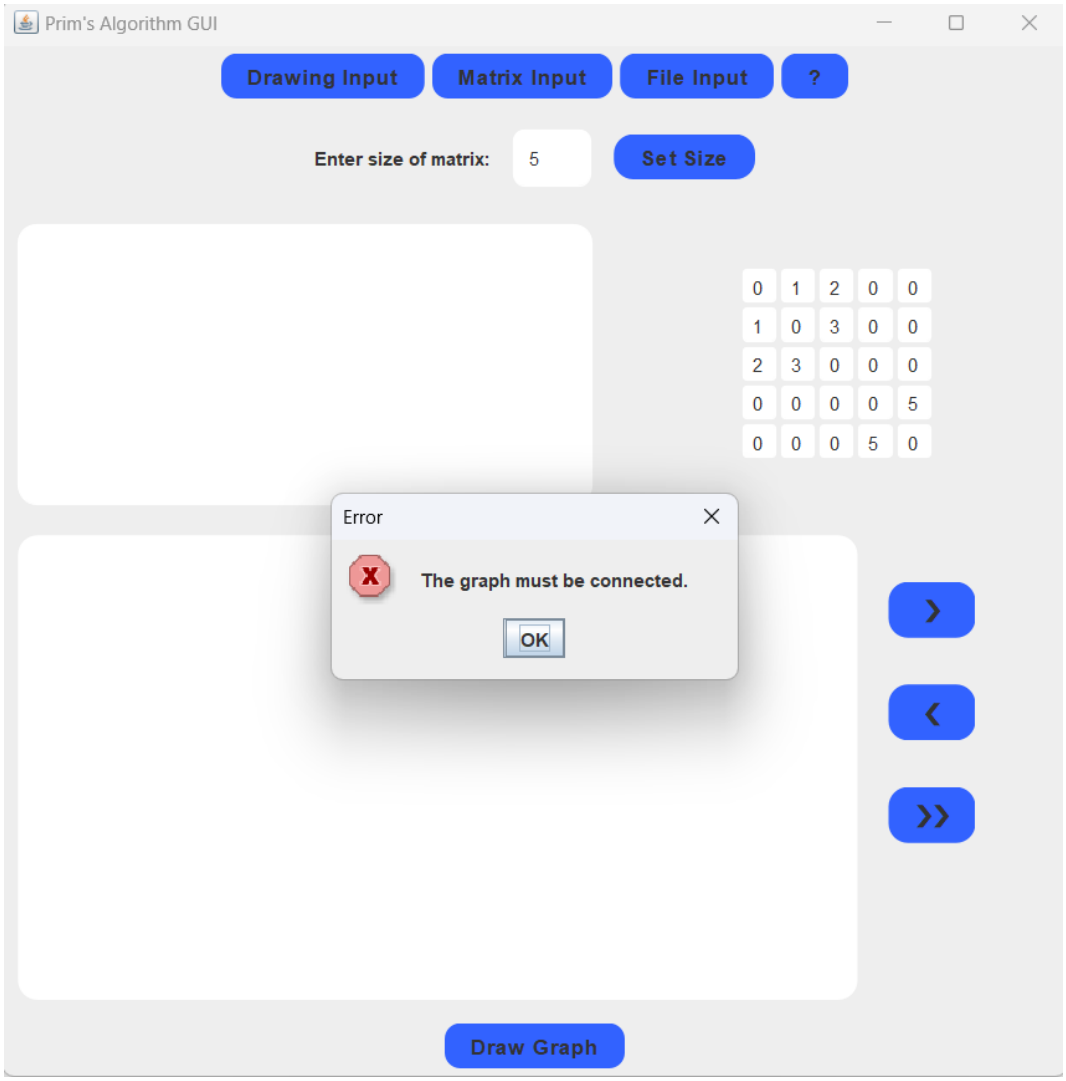

Рисунок 14 - Попытка задать несвязный граф

12) Если в матрице содержатся некорректные значения (например, отрицательные числа) пользователь будет оповещён об этом (Рисунок 15).

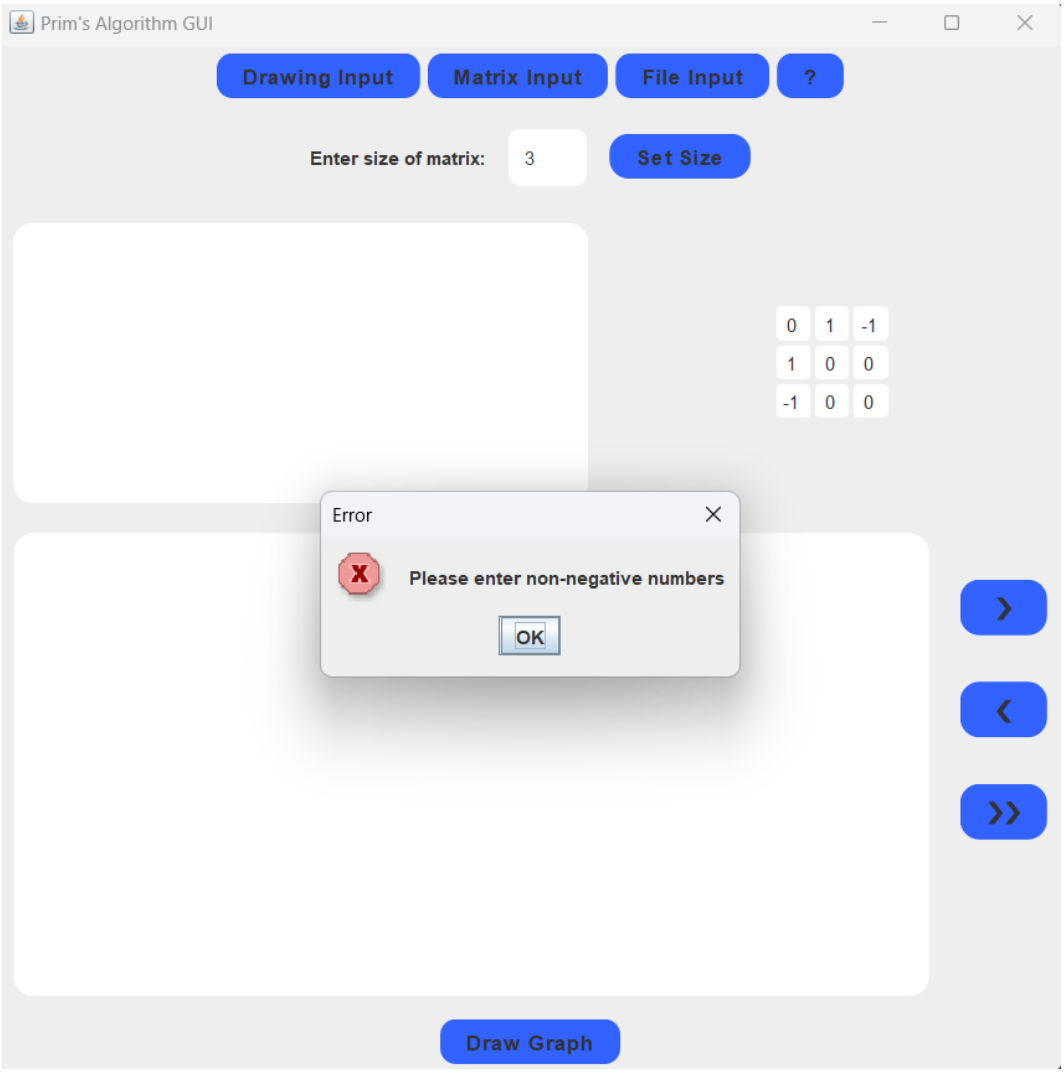

Рисунок 15 – Попытка ввести некорректные значения в матрицу смежности

13) При задании матрицы смежности в режиме File Input, если матрица задана корректно (строки матрицы разделены переносом строки, значения в строке разделены пробелами) и соблюдены все требования к матрице смежности для работы алгоритма, граф будет успешно нарисован (Рисунок 16).

Содержимое файла test_matrix.txt:

0 33 31 45 7 43 43

33 0 2 34 27 44 43

31 2 0 34 25 43 42

45 34 34 0 38 16 15

7 27 25 38 0 37 37

43 44 43 16 37 0 1

43 43 42 15 37 1 0

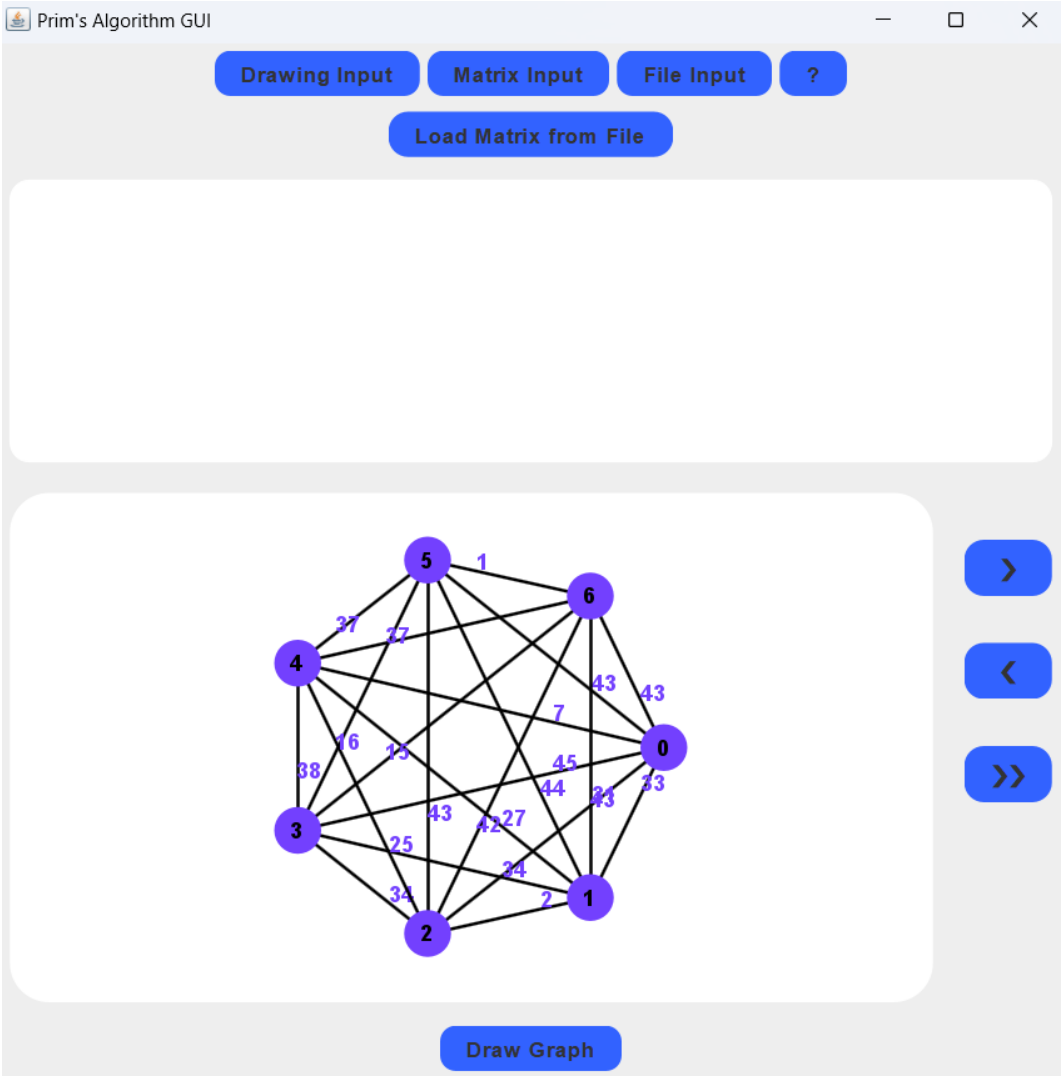

Рисунок 16 - Задание матрицы смежности из файла test_matrix.txt

14) Если все требования к матрице смежности соблюдены, алгоритм работает корректно (Рисунок 17).

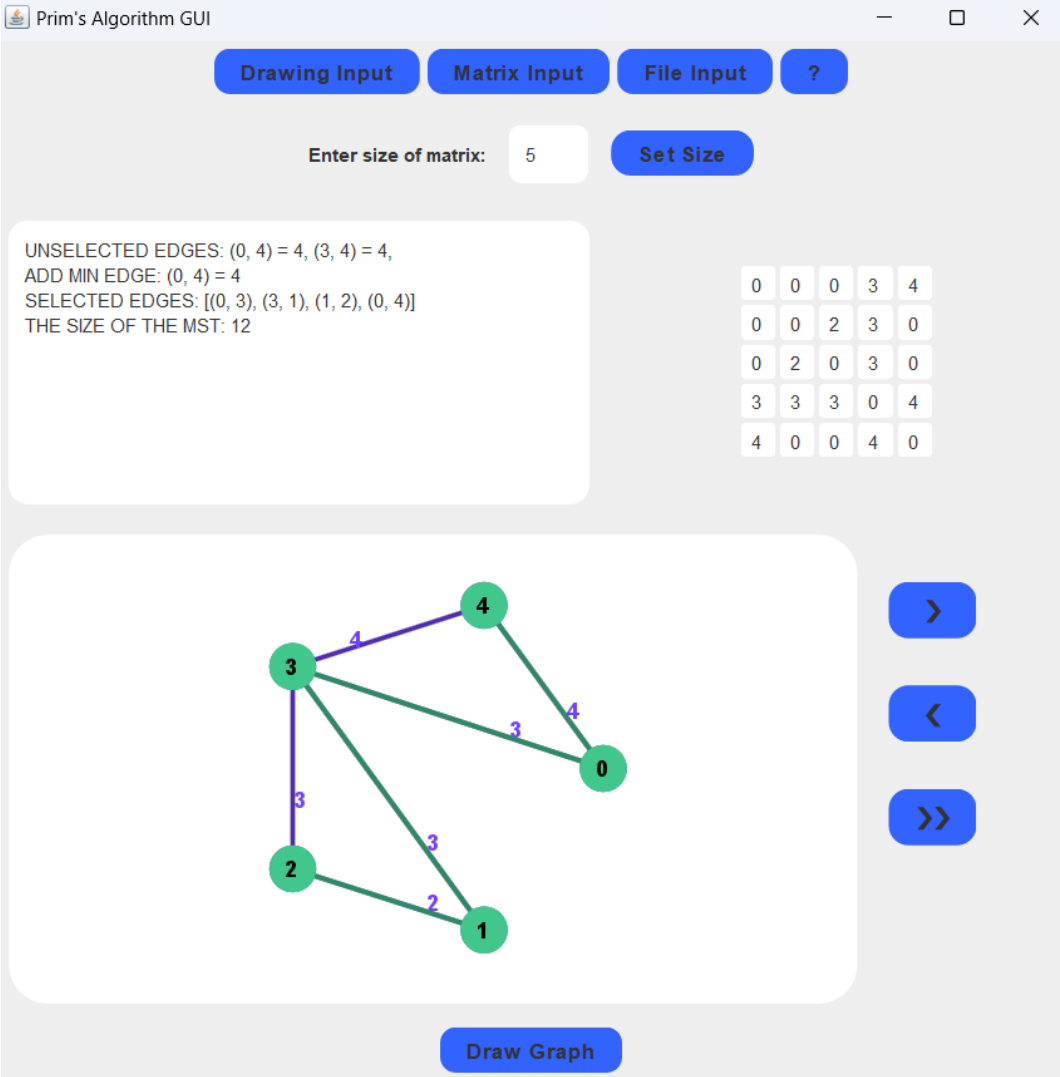

Рисунок 17 - Корректная работа алгоритма

ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта были получены базовые навыки программирования на языке Java. В соответствии со спецификацией успешно разработано приложение с графическим интерфейсом, реализующее алгоритм Прима построения минимального остовного дерева для взвешенного неориентированного графа. Проведён анализ поведения программы при различных тестовых сценариях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий бригады // GitHub URL: https://github.com/Son-of-Henry-Ford/PrimGUI (дата обращения: 07.07.2024).

2. Дискретная математика и информатика / Рыбин С.В. СПб.: ЭБС Лань, 2022. 000 с. 374 (дата обращения 02.07.2024).

3. Документация к библиотеки Swing // Package javax.swing URL: https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F/javax/swing/package-summary.html (дата обращения: 02.07.2024).