

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Зав. кафедрой \_\_ПОВТиАС\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Долгов В.В.\_

(подпись) (И.О.Ф.)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020г.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе по дисциплине Объектно-ориентированное программирование

на тему: Приложение для работы с графами

Автор проекта (работы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Омельченко С.Е.

подпись

Направление/специальность, профиль/специализация:

020303 Математическое обеспечение и администрирование информационных систем

Математическое обеспечение и администрирование информационных систем

Обозначение курсового проекта (работы) КР.410000.000 Группа ВМО32

Руководитель проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент, Габрельян Б.В. подпись (должность, И.О.Ф.)

Проект (работа) защищен (а)

дата оценка подпись

Ростов-на-Дону

2020



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Зав. кафедрой \_\_ПОВТиАС\_\_

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_Долгов В.В.\_

(подпись) (И.О.Ф.)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020г.

**ЗАДАНИЕ**

к курсовой работе по дисциплине Объектно-ориентированное программирование

Студент Омельченко С.Е. Код КР.410000.000 Группа ВМО32

Тема Приложение для работы с графами

Срок представления проекта (работы) к защите «\_\_» \_\_\_\_\_\_\_\_\_ 2020 г.

Исходные данные для курсового проекта (работы)

задание на выполнение курсовой работы Джеффри Рихтер - CLR VIA C# https://www.youtube.com/channel/UCBuMFSRHiSJPrxSu9RFunBg https://www.youtube.com/user/admshwan

Содержание пояснительной записки

ВВЕДЕНИЕ:

В разделе «Введение» рассматривается история возникновения графов, их актуальность в современном мире и области применения.

Наименование и содержание разделов:

1. «Постановка задачи». В разделе задаются требования к конечному результату, содержится описание структуры программы для работы с графами. Описываются возможности пользователя.
2. «Программное конструирование». В разделе рассматриваются инструменты разработки, описывается рабочий функционал компонентов программы. Описываются классы вершины, ребра и т.д., а также методы их взаимодействия.
3. «Инструкция по эксплуатации». В разделе описываются требования к пользователю для корректной работоспособности программного средства.
4. «Тестирование». В разделе проверена работоспособность программного средства. Приведен пример его работы.

ЗАКЛЮЧЕНИЕ:

В курсовой работе, согласно заданию, в рамках объектно-ориентированной парадигмы программирования, были реализованы все задачи, создано рабочее приложение.

Перечень графического материала:

1. Рисунок 1. Изначальный интерфейс пользователя
2. Рисунок 2. Пример вывода информации о графе, часть 1.
3. Рисунок 3. Пример вывода информации о графе, часть 2.
4. Рисунок 4. Пример работы алгоритма поиска в ширину.
5. Рисунок 5. Пример работы алгоритма поиска в глубину.
6. Рисунок 6. Пример работы алгоритма Прима.
7. Рисунок 7. Взвешенный ориентированный граф.
8. Рисунок 8 . Невзвешенный ориентированный граф.

Руководитель проекта (работы)

подпись, дата И.О.Ф.

Задание принял к исполнению

подпись, дата И.О.Ф

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

1 Постановка задачи 6

2 Программное конструирование 7

3 Инструкция по эксплуатации 14

4 Тестирование 16

ЗАКЛЮЧЕНИЕ 19

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20

ПРИЛОЖЕНИЕ А UML-диаграмма классов 21

ПРИЛОЖЕНИЕ B Фрагмент исходного кода приложения 22

**ВВЕДЕНИЕ**

Родоначальником теории графов считается Леонард Эйлер. В 1736 году в одном из своих писем он формулирует и предлагает решение задачи о семи кёнигсбергских мостах, ставшей впоследствии одной из классических задач теории графов[1]. Толчок к развитию теория графов получила на рубеже XIX и XX столетий, когда резко возросло число работ в области топологии и комбинаторики, с которыми её связывают самые тесные узы родства. Как отдельная математическая дисциплина теория графов была впервые представлена в работе венгерского математика Кёнинга в 30-е годы XX столетия.

В последнее время графы и связанные с ними методы исследований пронизывают на разных уровнях едва ли не всю современную математику. Графы используются в теории планирования и управления, теории расписаний, социологии, математической лингвистике, экономике, биологии, медицине. Как более жизненный пример можно взять использование графов в геоинформационных системах. Существующие или вновь проектируемые дома, сооружения, кварталы и т. п. рассматриваются как вершины, а соединяющие их дороги, инженерные сети, линии электропередачи и т. п. — как рёбра. Применение различных вычислений, производимых на таком графе, позволяет, например, найти кратчайший объездной путь или ближайший продуктовый магазин, спланировать оптимальный маршрут[2].

Теория графов быстро развивается и находит всё новые области применения.

В рамках данного курсового проекта поставлена задача реализовать программный стенд для работы с графами.

**1 Постановка задачи**

Разработать и реализовать программный стенд, удовлетворяющий парадигмам ООП, позволяющий вводить граф, отслеживать информацию о нём в реальном времени, а также применять к нему алгоритмы обработки.

Программное средство должно поддерживать активное взаимодействие с пользователем посредством:

1. функции сохранения информации о графе в файл;
2. страницы с графическим интерфейсом, содержащей:
   1. поле ввод графа посредством добавления вершин и рёбер при помощи кликов мыши;
   2. страницу, отображающую состояние графа в реальном времени, реализующей:
      1. настройку ориентированности графа;

2.2.2. настройку взевешенности графа;

2.2.3. отображение основной информации о графе

2.3. страницы действий с графом, реализующей:

2.3.1. удаление графа

2.3.2 очистку графа

2.3.3 очистку полей вывод информации

2.3.4 алгоритмы для работы с графом

3. страницы, содержащей руководство пользователя.

**2 Программное конструирование**

Для решения поставленной задачи необходимо соблюдать парадигмы ООП.Объектно-ориентированное программирование (ООП) – методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса, а классы образуют иерархию на принципах наследуемости[3].С появлением ООП, открылась возможность работы с программами большего объема, что очень удобно для промышленного программирования и создания различных приложений.

Класс – это структурный тип, используемый для описания некоторого множества объектов предметной области, имеющих общие свойства и поведение. Конкретные переменные типа данных «класс» называются экземплярами класса или объектами.

В качестве основного инструмента разработки приложения для работы с графами был выбран язык программирования C# с графической подсистемой, входящей в состав .NET Framework, WPF.

Язык программирования C# - современный мультипарадигмальный язык программирования с большим количество встроенных библиотек для решения поставленных задач любой степени сложности. Для расширения функциональности обладает системой управления пакетами Nuget, привычным С подобным синтаксисом.

Встроенная графическая подсистема WPF позволяет создавать клиентские приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем. Благодаря встроенному визуальному конструктору приложений данная подсистема позволяет быстро создавать визуальную составляющую программы.

Средой разработки программного средства был выбран продукт компании Microsoft - Microsoft Visual Studio. Данное средство включает в себя [интегрированную среду разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) программного обеспечения c  [редактор исходного кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0) с поддержкой технологии [IntelliSense](https://ru.wikipedia.org/wiki/IntelliSense) и возможностью простейшего [рефакторинга кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3). Так же оно включает в себя и ряд других инструментальных средств, которые позволяют разрабатывать как [консольные](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) [приложения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), так и игры и приложения с [графическим интерфейсом](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F), в том числе с поддержкой технологии [WPF](https://ru.wikipedia.org/wiki/Windows_Forms).

Все описанные преимущества делают разработку приложения c помощью C# с его графической подсистемой WPF для визуализации моделирования растительных объектов удобной, быстрой и практичной.

Для реализации поставленной задачи были использованы[4]:

1. Класс Vertex – представляющий из себя вершину. Данный класс включает в себя следующие методы:
2. конструкторы public Vertex(Point center, int id, Canvas canvas) и public Vertex(string name, int id, Point center, double r, Canvas canvas) для инициализации объекта класса;
3. public bool Equality(Vertex to) для сравнения объектов класса Vertex;
4. public void Draw() для отрисовки вершины;
5. public override string ToString() для вывода информации о вершине;
6. public string GetName(), public int GetID(), public Point GetCenter(), public double GetR(), public Brush GetFillColor(), public Brush GetStrokeColor()для получения имени, ID, координаты центра, радиуса, цвета фона и контура вершины соответственно;
7. public void SetName(string name), public void SetPoint(Point center), public void SetR(double r), public void SetFillColor(byte red, byte green, byte blue) и public void SetStrokeColor(byte red, byte green, byte blue) для изменения имени, центра вершины, радиуса, цвета фона и контура вершины соответственно.
8. Класс Edge – представляющий из себя ребро. Данный класс включает в себя следующие методы:
9. Конструктор public Edge(Vertex from, Vertex to, int id, Canvas canvas, int weight = 1) для инициализации объекта класса;
10. public void Draw(bool orient, bool weighted) для управления отрисовкой ребра. В зависимости от входных параметров и свойств ребра вызывает одну из следующих функций:
    1. private void DrawArrowEdge(bool weighted) для отрисовки ребра в ориентированном графе;
    2. private void DrawLineEdge(bool weighted) для отрисовки ребра в неориентированном графе;
    3. private void DrawReturnArrowEdge для отрисовки петли в ориентированном графе;
    4. private void DrawReturnLineEdge для отрисовки петли в неориентированном графе;
11. public bool Equality(Edge to) для сравнения объектов класса Edge;
12. public override string ToString() для вывода информации о ребре;
13. public Vertex GetFrom(), public Vertex GetTo(), public int GetWeight(), public int GetID(), public Canvas GetCanvas() для получения вершины, из которой идёт ребро, вершины, в которую приходит ребро, веса, ID и места отрисовки соответственно;
14. public void SetWeight(int w), public void SetColor(Brush br) для установки веса и цвета ребра соответственно.
15. Класс MainGraph – представляющий из себя граф. Данный класс включает в себя следующие методы:
16. public Vertex SearchVertex(int ID) для поиска вершины в списке вершин;
17. public void AddVertex(Point click) для добавления вершины в список вершин;
18. public void RemoveVertex(Point click) для удаления вершины из списка вершин;
19. public void AddEdge(Vertex from, Vertex to, int weight = 1) для добавления ребра в список рёбер;
20. public void RemoveEdge(Edge edge) для удаления ребра из списка рёбер;
21. public void RemoveRepeatingEdge(bool orient) для удаления рёбер при переходе из ориентированного графа в неориентированный;
22. public int[,] GetMatrixAdjacency(bool orient) для получения матрицы смежности;
23. public int[,] GetMatrixIncidence(bool orient) для получения матрицы инцидентности;
24. public int[,] GetMatrixWeight(bool orient, bool weighted) для получения матрицы весов;
25. public Dictionary<int, List<(int, int)>> GetListAdjacency(bool orient, bool weighted) для получения списка смежных рёбер;
26. public string GetListEdges() для получения списка рёбер;
27. public Dictionary<int, List<(int,bool)>> GetPowVertex() для получения степеней вершин;
28. public Vertex PushOnVertex(Point click) для проверки нажатия на вершину;
29. public double Distance(double x0, double y0, double x1, double y1) для просчёта расстояния между двумя точками;
30. public Edge PushOnEdge(Point click) для проверки нажатия на ребро;
31. public Edge SearchEdge(Edge sEdge, bool orient) для проверки вхождения ребра в список рёбер;
32. public void PrintMatrixs() для вывода информации о графе;
33. public void DrawGraph() для отрисовки графа;
34. public List<Vertex> GetVertexes() для получения списка вершин графа;
35. public List<Edge> GetEdge() для получения списка рёбер графа.
36. Класс WorkWithGraph содержащий методы работы с графом:
37. private bool SearchElement(Queue<(int, int)> q, int el) для поиска ребра в очереди рёбер ;
38. private bool SearchElement(Stack<(int, int)> s, int el) для поиска ребра в стэке рёбер;
39. private int SearchPosition(List<(int from, int to)> lInput, int index) для поиска позиции вставки ребра для алгоритма DFS;;
40. public List<(int, int)> BFS(Dictionary<int, List<(int, int)>> listAdj, int stID) - реализации алгоритма поиска в ширину
41. public List<(int,int)> DFS(Dictionary<int, List<(int, int)>> listAdj, int stID) – реализация поиска в глубину;
42. public List<(int, int, int)> AlgorithmPrima(List<Vertex> Vertexes, List<Edge> Edges, bool orient) – реализация алгоритма Прима;
43. private List<Edge> SortedEdge(List<Edge> Edges) для сортировки рёбер по возрастанию их веса;
44. private List<List<Vertex>> UnionList(List<List<Vertex>> lSet, Vertex v1, Vertex v2) для объединения подсписков рёбер в списке списков рёбер;
45. private bool DifferentSets(List<List<Vertex>> lSet, Vertex v1, Vertex v2) для проверки принадлежности вершин разным спискам вершин;
46. public List<(int from, int to, int weight)> AlgorithmKruskal(List<Vertex> vertexes, List<Edge> nedges) – реализация алгоритма Краскала.
47. Класс PrintMatrix для вывода информации в форматированном виде. Содержит следующие методы:
48. public string printMatrix(Dictionary<int, Dictionary<int, int>> matrix) для печати матриц из словаря словарей
49. public string printMatrix(int[,] matrix) для форматированной печати матрицы
50. public string printListAdjacency(Dictionary<int, List<(int, int)>> listAdjacency) для печати списка смежности
51. public string printList(List<int> lVisited) для печати посещённых во время работы алгоритмов вершин
52. public string printList(List<(int from, int to)> lVisited) для печати путей при посещении вершин
53. public string printList(List<(int, int, int)> lVisited) для печати путей с весом при посещении вершин
54. public string printPow(Dictionary<int, List<(int, bool)>> dInput, bool orient) для печати степеней вершин в графе

**3 Инструкция по эксплуатации**

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ Для добавления вершины пользователю необходимо нажать левой кнопкой мыши по свободному от вершин месту, в противном случае вершина добавлена не будет.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ Индексы вершинам присваиваются динамически, в зависимости от количество ранее созданных вершин. Нумерация начинается с 1.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ Для удаления вершины пользователю необходимо снова нажать левой кнопкой мыши, но уже по ранее добавленной вершине.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ Для добавления ребра необходимо:

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎- во-первых, правой кнопкой мыши выбрать стартовую вершину, из которой начнётся построение будущего ребра. Если нажатие будет произведено по вершине, то её контур изменит свой цвет, в противном случае ничего не произойдёт;

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎- во-вторых, правой кнопкой мыши выбрать финальную вершину, в которую будет приходить ребро.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ При этом, если нажатие было произведено не по вершине, то ранее выбранная стартовая вершина сбросится, иначе, в зависимости от того, равны или нет стартовая и конечная вершина, существует 2 варианта рёбер:

* вершины разные – ребро;
* вершина одна - петля.

Не зависимо от попадания\промаха стартовая вершина изменит цвет своего контура на прежний, что визуально определит конец операции построения ребра.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ Для удаления обычного ребра в не взвешенном графе необходимо нажать левой кнопкой мыши на него.

Если же граф взвешенный, то нужно нажать левой кнопкой мыши на кружок, в котором отображается вес ребра. В открывшемся окне нужно нажать на кнопку "удалить ребро".

Для удаления же петли, необходимо нажать дважды правой кнопкой по вершине, которой принадлежит эта петля.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ Так же если граф взвешенный и возникла необходимость изменить вес ребра, то сделать это можно кликнув по кружку, в котором отображается его вес, левой кнопкой, после чего в открывшемся окне ввести в поле нужный вес, после чего сохранить.

Если нажатие было произведено нечаянно, то просто закройте окно, вес ребра при этом останется прежним.

При этом у петли нельзя изменить вес. Он по умолчанию равен 1.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ В зависимости от настроек графа (ориентированный, взвешенный) и отношения стартовой и конечной вершины (разные, одна и та же) возможные различные вариации вида рёбер.

Рассмотрим на примере случая, когда стартовая вершина не является конечной:

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ 1) Граф неориентированный и не взвешенный.

Самый простой случай: ребро представляет из себя прямую, соединяющую 2 вершины.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ 2) Граф ориентированный и не взвешенный.

Теперь же на линии, соединяющей 2 ребра, появляется небольшая стрелка, указывающая направление ребра.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ 3) Граф неориентированный и взвешенный.

На линии, соединяющей вершины, появляется окружность, отображающая в себе вес ребра.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ 4) Граф ориентированный и взвешенный.

Самой насыщенный вид рёбер. Они имеют в своём составе как стрелки, указывающие направление ребра, так и окружность, отображающую его вес.

‌‌‍‍‎‌‌‍‍‎‌‌‍‍‎ В случае, если начальная и конечная вершина одна и та же, то линии превращаются в окружность, сохраняя остальные свойства, кроме отображения веса, который по умолчанию равен 1.

**4 Тестирование**

При запуске программы откроется интерфейс пользователя (Рисунок 1)

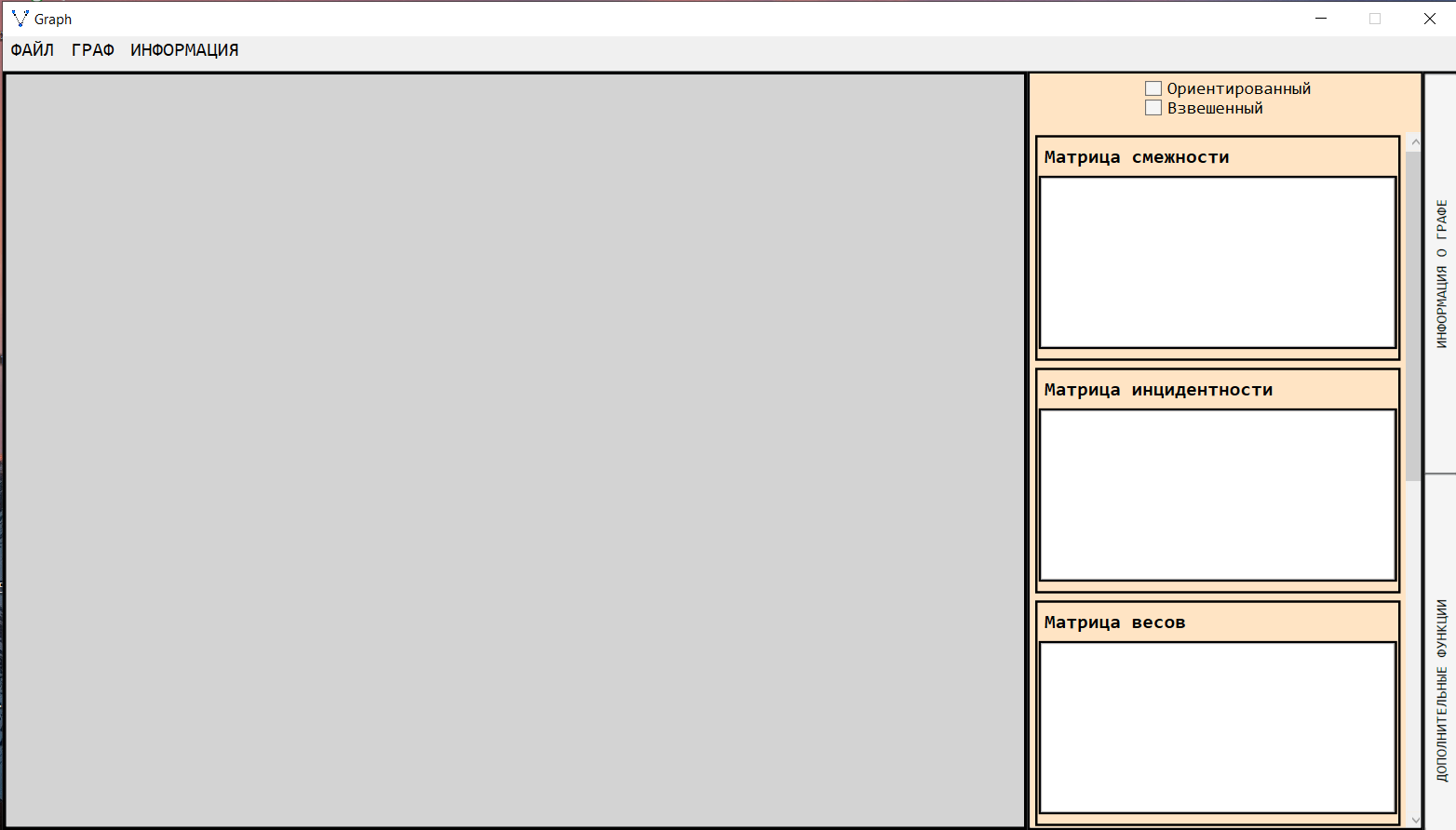
****

Рисунок 1 – Изначальный интерфейс пользователя.

Начиная вводить необходимый граф, пользователь может наблюдать в динамике изменение его состояния (Рисунок 2, Рисунок 3).

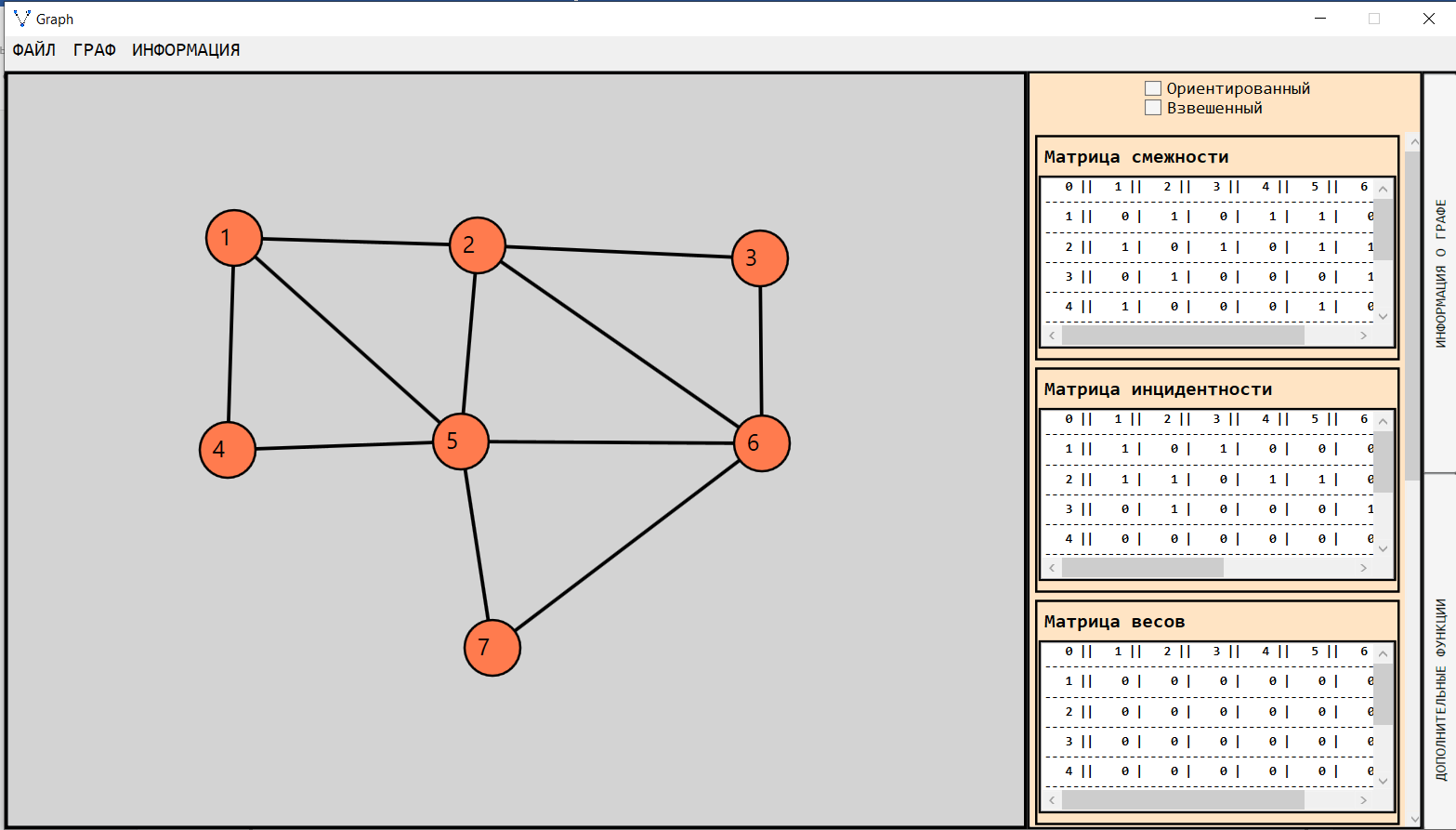
****

Рисунок 2 – Пример вывода информации о графе, часть 1.

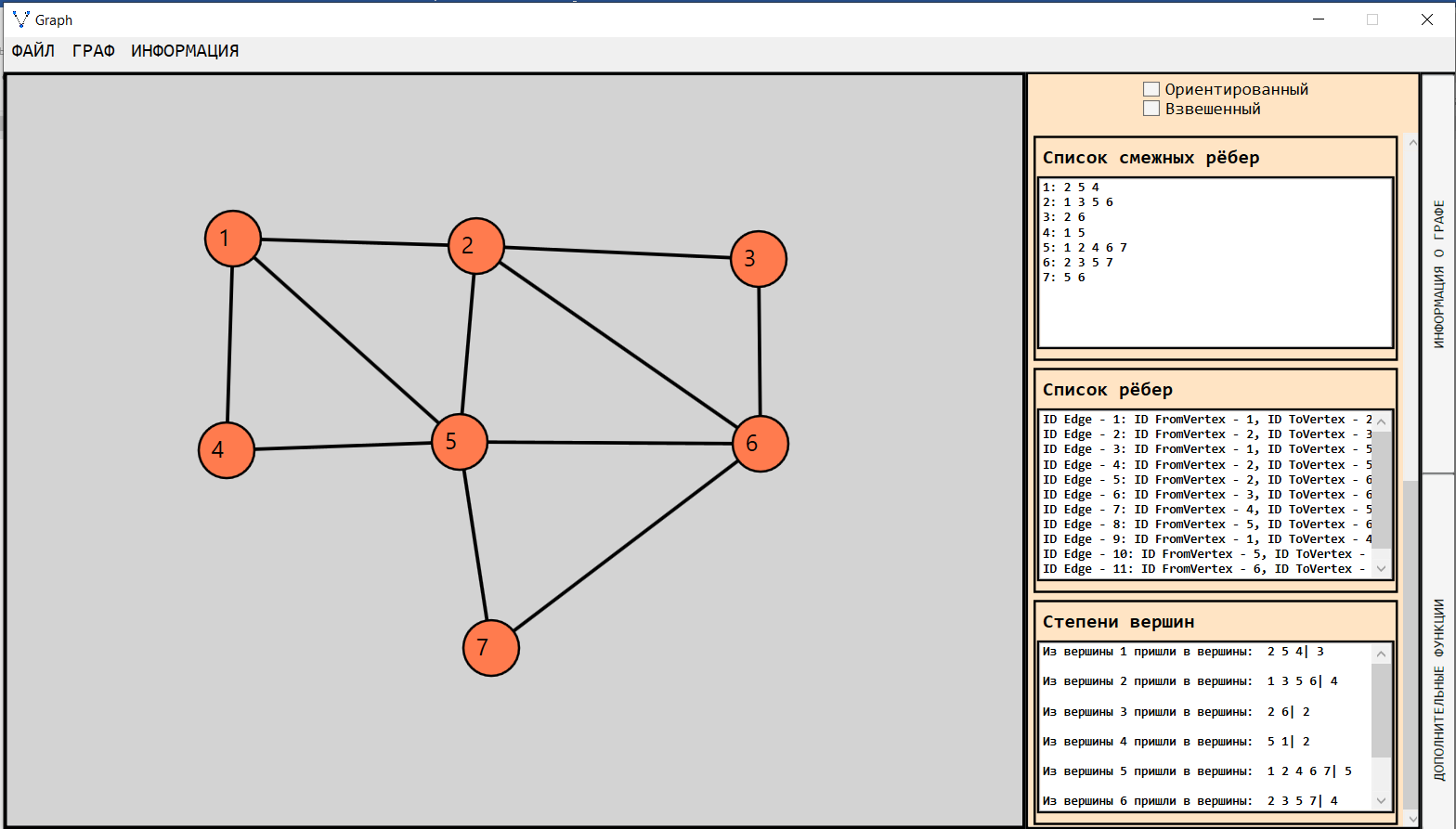
****

Рисунок 3 – Пример вывода информации о графе, часть 2.

Введя необходимый граф, пользователь может применить к нему один из представленных в программе алгоритмов (Рисунок 4, Рисунок 5, Рисунок 6).

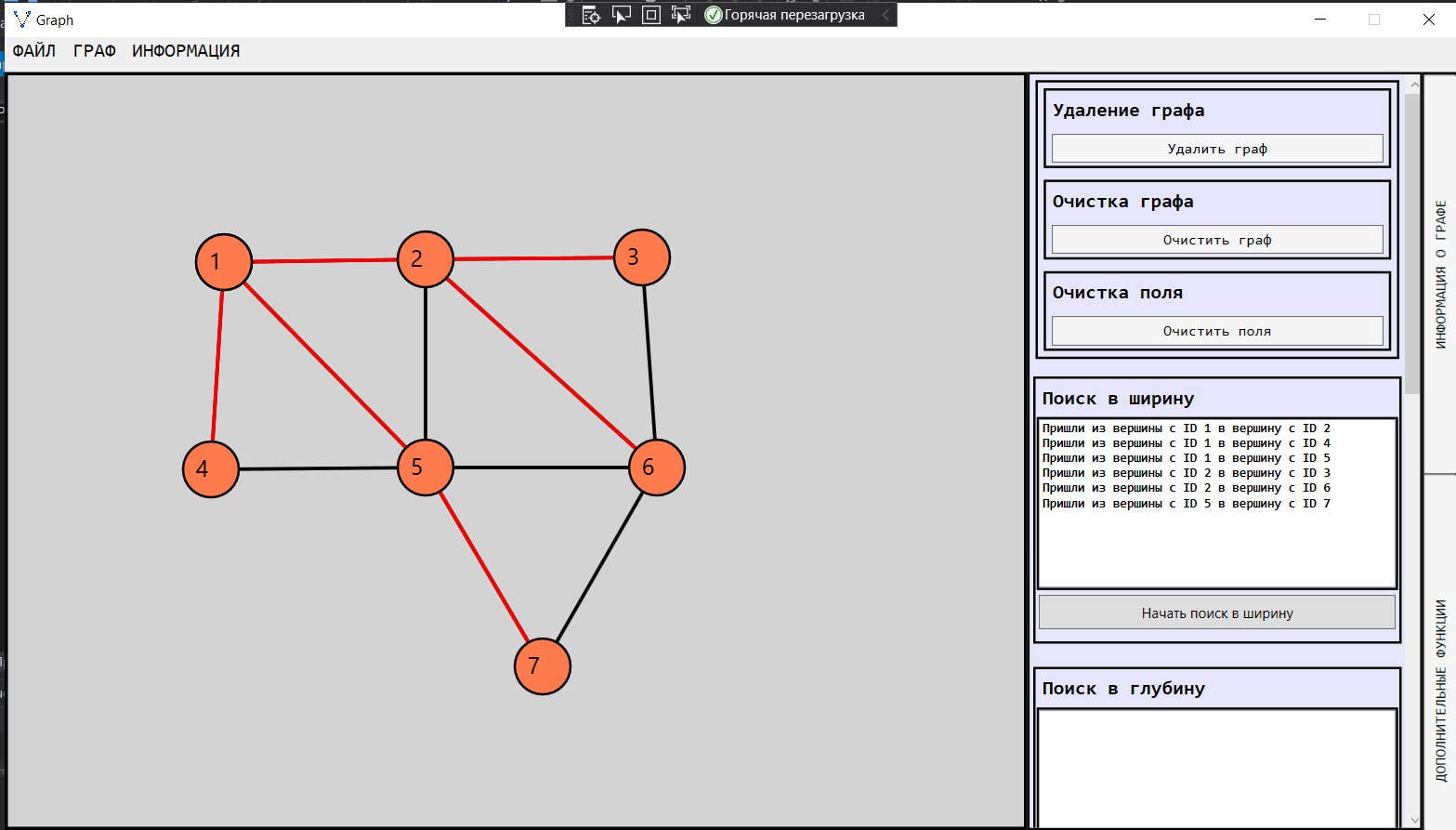
****

Рисунок 4 – Пример работы алгоритма поиска в ширину.

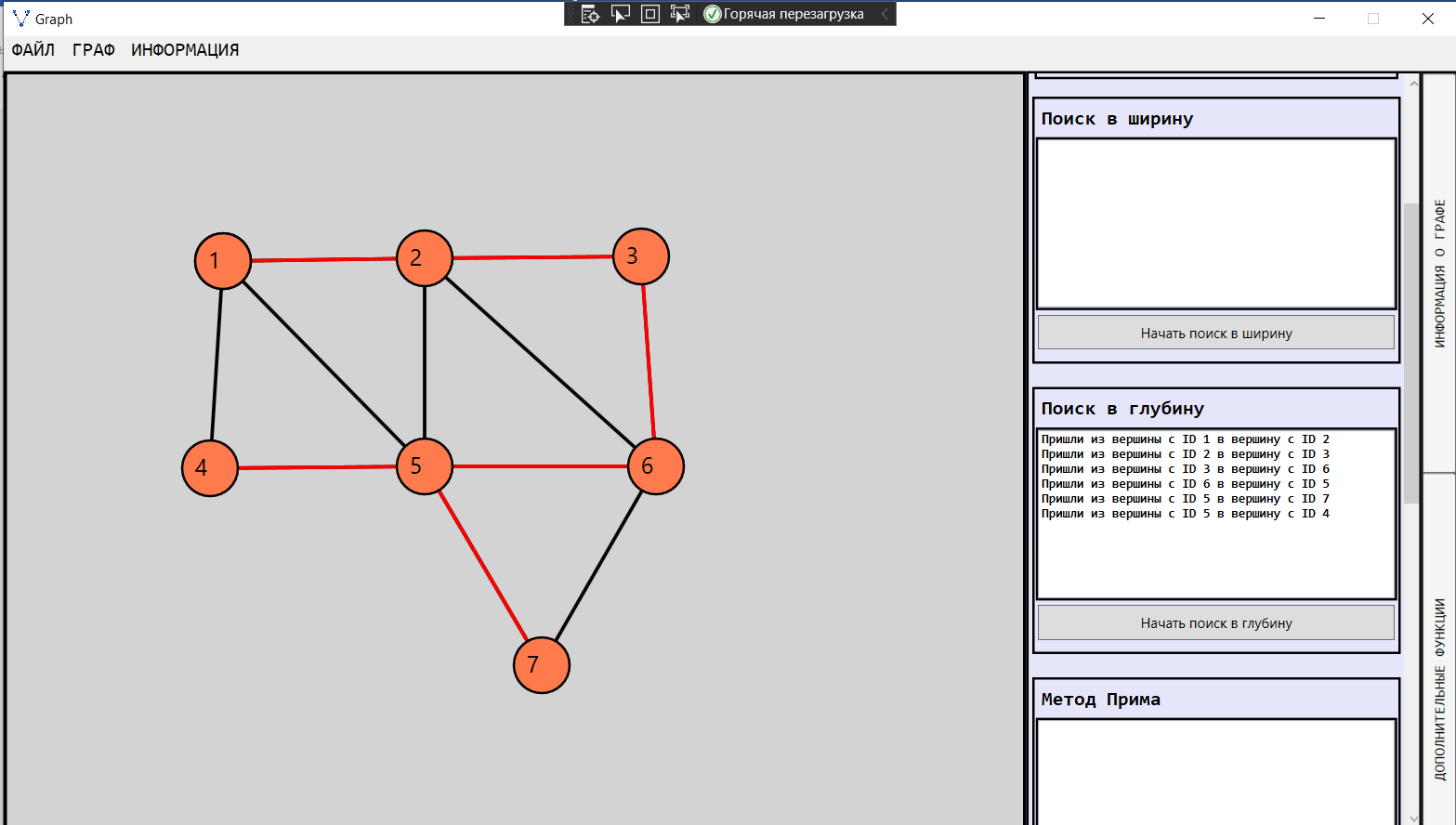
****

Рисунок 5 – Пример работы алгоритма поиска в глубину.

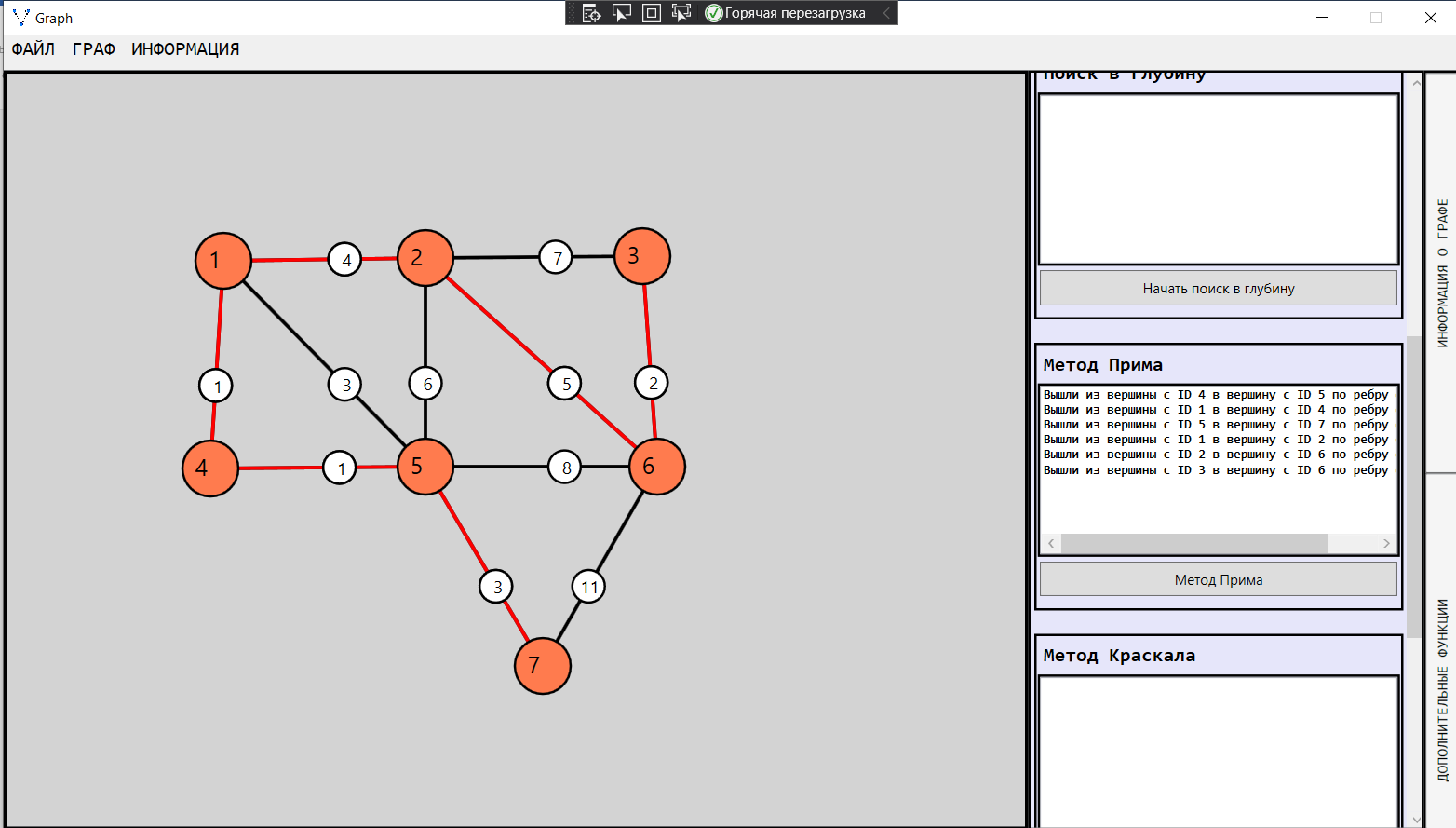
****

Рисунок 6 – Пример работы алгоритма Прима.

Так же примеры графа при разных его настройках (Рисунок 7, Рисунок 8).

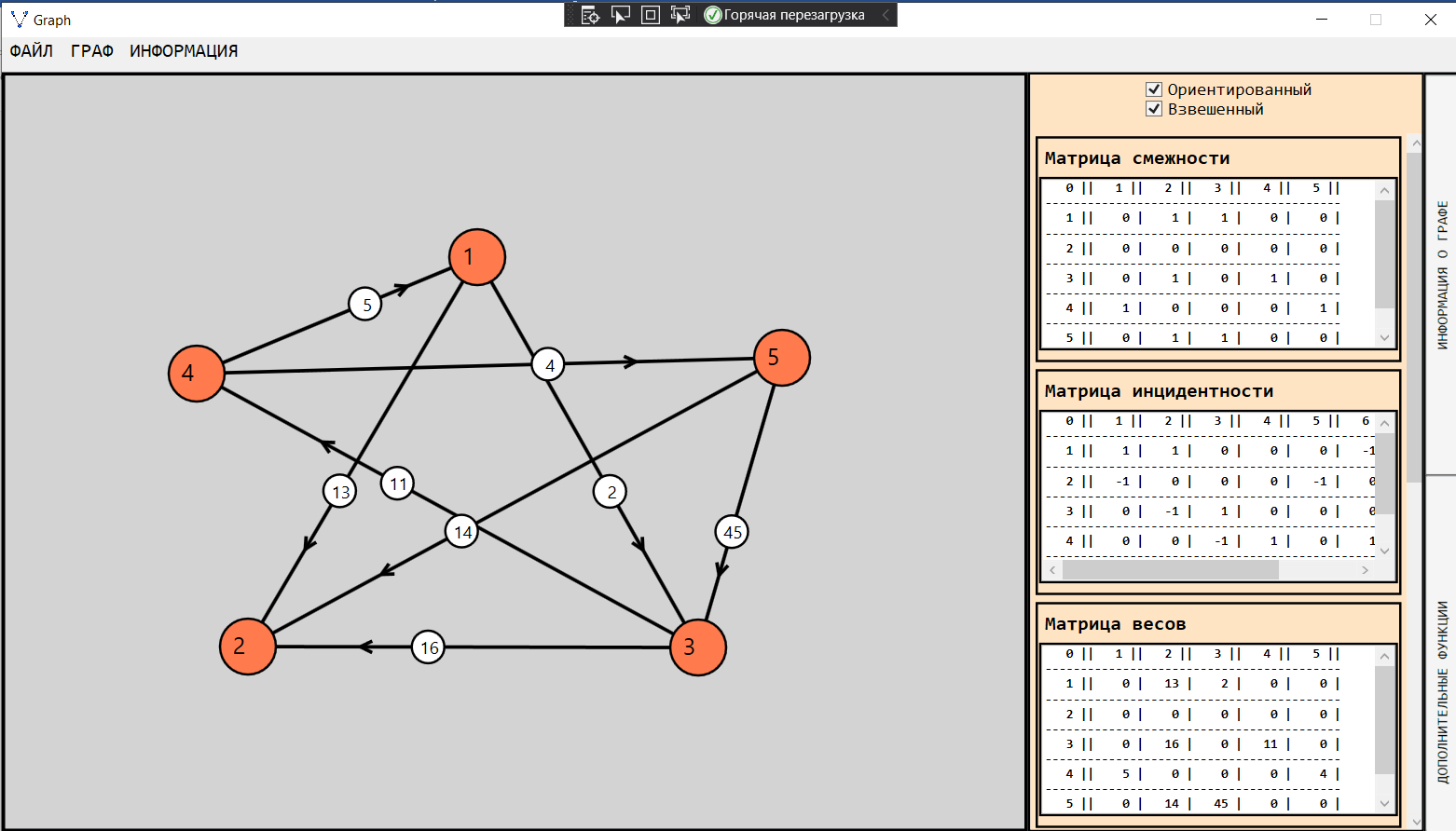


Рисунок 7 – Взвешенный ориентированный граф.

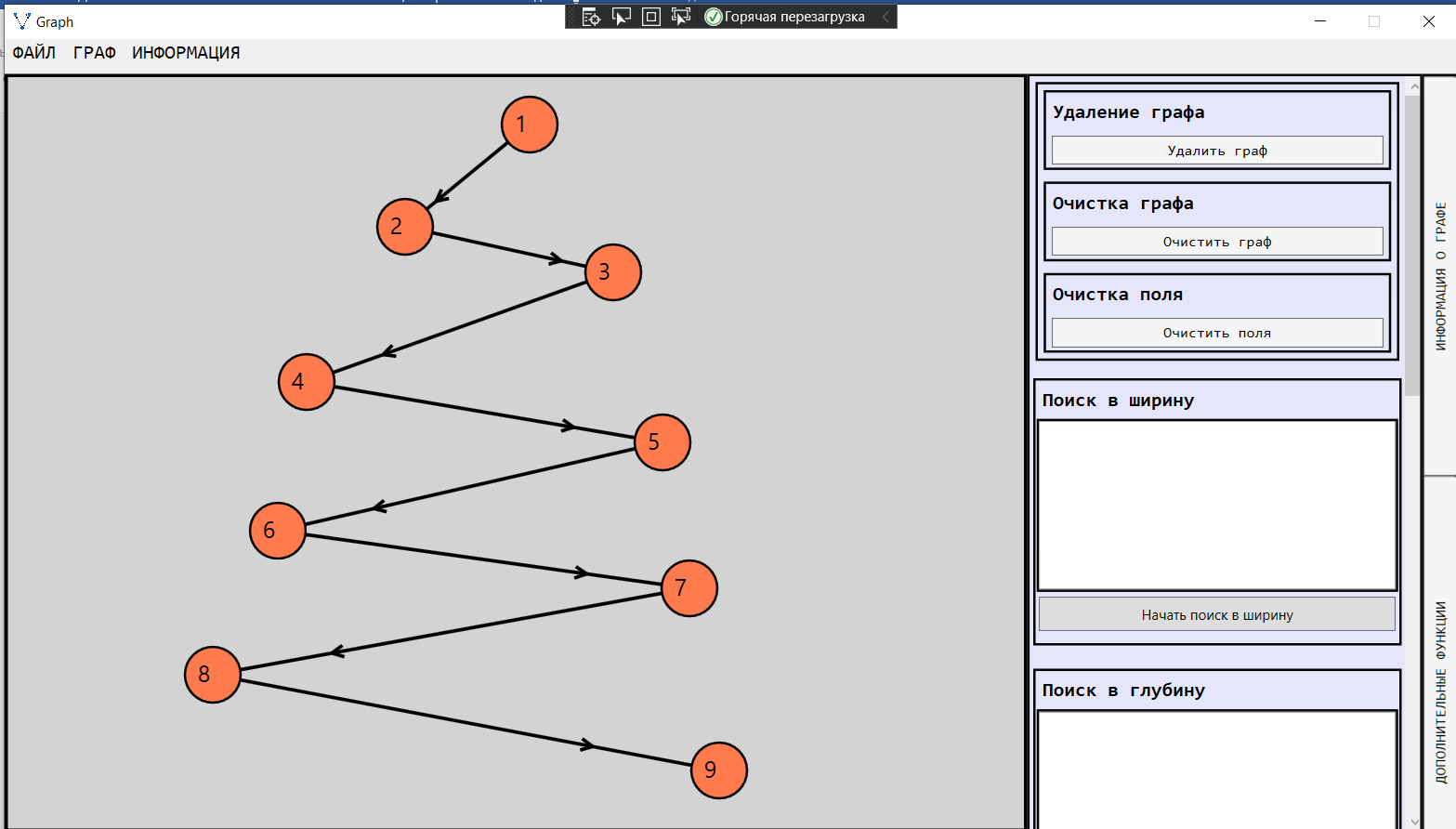
****

Рисунок 8 – Невзвешенный ориентированный граф.

**ЗАКЛЮЧЕНИЕ**

Результатом проделанной работы является программный стенд, позволяющий работать с графами.

В ходе выполнения данной работы на основе различных источников были повышены знания в области ООП и программирования на С#.

Все поставленные ранее задачи по курсовому проекту были успешно выполнены.

Написанная программа соответствует заявленным требованиям задания на курсовую работу и успешно прошла тестирование.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Электронный ресурс: https://ru.wikipedia.org/wiki/Теория\_графов

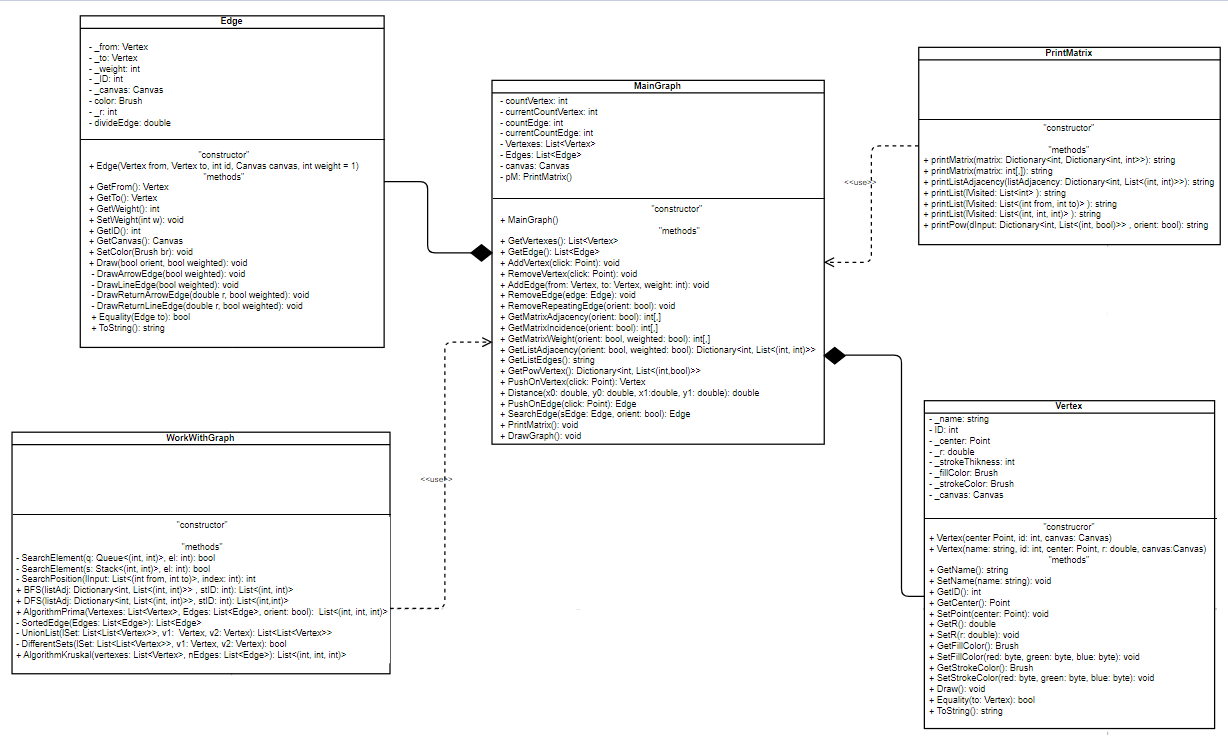
2. Электронный ресурс: https://school-science.ru/4/7/33615

3. Иан Грэхем. Объектно-ориентированные методы. Принципы и практика = Object-Oriented Methods: Principles & Practice. — 3-е изд. — М.: «Вильямс», 2004. — С. 880. — ISBN 0-201-61913-X

4. Электронный ресурс: https://www.youtube.com/user/admshwan

**ПРИЛОЖЕНИЕ А**

**UML-диаграмма классов**



**ПРИЛОЖЕНИЕ B**

**Фрагмент исходного кода приложения**

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

namespace Graph

{

public class MainGraph

{

//индексы для ID

private int countVertex = 0;

private int currentCountVertex = 0;

private int countEdge = 0;

private int currentCountEdge = 0;

//спики для вершин и рёбер

private List<Vertex> Vertexes = new List<Vertex>();

private List<Edge> Edges = new List<Edge>();

public List<Vertex> GetVertexes() { return Vertexes; }

public List<Edge> GetEdge() { return Edges; }

public Vertex SearchVertex(int ID)

{

foreach(var vert in Vertexes)

{

if (vert.GetID() == ID)

return vert;

}

return null;

}

private Canvas canvas;

PrintMatrix pM = new PrintMatrix();

PageInformationAboutGraph pInfo;

public MainGraph(Canvas c, PageInformationAboutGraph pageInfo)

{

canvas = c;

pInfo = pageInfo;

}

#region Add/Remove

#region Add/Remove Vertex

public void AddVertex(Point click)

{

if (PushOnVertex(click) == null)

{

if (currentCountVertex == 0)

countVertex = 0;

++currentCountVertex;

Vertex vertex = new Vertex(click, ++countVertex, canvas);

Vertexes.Add(vertex);

vertex.Draw();

printMatrixs();

}

}

public void RemoveVertex(Point click)

{

Vertex tVertex = PushOnVertex(click);

if (tVertex != null)

{

--currentCountVertex;

if (tVertex.GetID() == countVertex)

--countVertex;

Vertexes.Remove(tVertex);

List<Edge> tlEdge = new List<Edge>();

foreach (var edge in Edges)

{

if (!edge.GetFrom().Equality(tVertex) && !edge.GetTo().Equality(tVertex))

tlEdge.Add(edge);

}

Edges = tlEdge;

DrawGraph();

printMatrixs();

}

}

#endregion

#region Add/Remove Edge

public void AddEdge(Vertex from, Vertex to, int weight = 1)

{

if (currentCountEdge == 0)

countEdge = 0;

++currentCountEdge;

Edge tEdge = new Edge(from, to, ++countEdge, canvas, weight);

Edges.Add(tEdge);

DrawGraph();

printMatrixs();

}

public void RemoveEdge(Edge edge)

{

--currentCountEdge;

if (edge.GetID() == countEdge)

--countEdge;

Edges.Remove(edge);

DrawGraph();

printMatrixs();

}

//удаление рёбер при неориентированном графе из ориентированного

public void RemoveRepeatingEdge(bool orient)

{

if (!orient)

{

var newEdges = new List<Edge>();

bool flag = false;

foreach (var e1 in Edges)

{

flag = false;

foreach (Edge e2 in newEdges)

{

if (e1.GetFrom().Equality(e2.GetTo()) && e1.GetTo().Equality(e2.GetFrom()))

flag = true;

}

if (!flag)

newEdges.Add(e1);

}

Edges = newEdges;

}

}

#endregion

#endregion

#region Matrix

//матрица смежности - вершины вершины

public int[,] GetMatrixAdjacency(bool orient)

{

var matrix = new int[Vertexes.Count + 1, Vertexes.Count + 1];

for (int i = 0; i < Vertexes.Count + 1; i++)

{

for (int j = 0; j < Vertexes.Count + 1; j++)

{

matrix[i, j] = 0;

}

}

for (int i = 0; i < Vertexes.Count; i++)

{

matrix[0, i + 1] = Vertexes[i].GetID();

}

for (int i = 0; i < Vertexes.Count; i++)

{

matrix[i + 1, 0] = Vertexes[i].GetID();

}

foreach (var edge in Edges)

{

var row = Vertexes.IndexOf(edge.GetFrom()) + 1;

var column = Vertexes.IndexOf(edge.GetTo()) + 1;

matrix[row, column] = 1;

if (!orient)

matrix[column, row] = 1;

}

return matrix;

}

//матриц инциденстности - ребра вершины

public int[,] GetMatrixIncidence(bool orient)

{

var matrix = new int[Vertexes.Count + 1, Edges.Count + 1];

for (int i = 0; i < Vertexes.Count + 1; i++)

{

for (int j = 0; j < Edges.Count + 1; j++)

{

matrix[i, j] = 0;

}

}

for (int i = 0; i < Edges.Count; i++)

{

matrix[0, i + 1] = Edges[i].GetID();

}

for (int i = 0; i < Vertexes.Count; i++)

{

matrix[i + 1, 0] = Vertexes[i].GetID();

}

//заполнение путями

for (int i = 0; i < Edges.Count; i++)

{

var column = i + 1;

var rowFrom = Vertexes.IndexOf(Edges[i].GetFrom()) + 1;

var rowTo = Vertexes.IndexOf(Edges[i].GetTo()) + 1;

matrix[rowFrom, column] = 1;

if (orient)

matrix[rowTo, column] = -1;

else

matrix[rowTo, column] = 1;

}

return matrix;

}

//матриц весов - ребра вершины

public int[,] GetMatrixWeight(bool orient, bool weighted)

{

var matrix = new int[Vertexes.Count + 1, Vertexes.Count + 1];

//заполнение 0

for (int i = 0; i < Vertexes.Count + 1; i++)

{

for (int j = 0; j < Vertexes.Count + 1; j++)

{

matrix[i, j] = 0;

}

}

for (int i = 0; i < Vertexes.Count; i++)

{

matrix[0, i + 1] = Vertexes[i].GetID();

}

for (int i = 0; i < Vertexes.Count; i++)

{

matrix[i + 1, 0] = Vertexes[i].GetID();

}

//заполнение весами

if (!weighted)

return matrix;

else

{

foreach (var edge in Edges)

{

var row = Vertexes.IndexOf(edge.GetFrom()) + 1;

var column = Vertexes.IndexOf(edge.GetTo()) + 1;

matrix[row, column] = edge.GetWeight();

if (!orient)

matrix[column, row] = edge.GetWeight();

}

return matrix;

}

}

//список смежности

public Dictionary<int, List<(int, int)>> GetListAdjacency(bool orient, bool weighted)

{

var listAdjancy = new Dictionary<int, List<(int, int)>>();

foreach(var vert in Vertexes)

listAdjancy.Add(vert.GetID(), new List<(int, int)>());

foreach (var vert in Vertexes)

{

foreach (var edge in Edges)

{

if (edge.GetFrom().Equality(vert))

{

int weight = -1;

if (weighted)

weight = edge.GetWeight();

listAdjancy[vert.GetID()].Add((edge.GetTo().GetID(), weight));

if(!orient)

listAdjancy[edge.GetTo().GetID()].Add((vert.GetID(), weight));

}

}

}

return listAdjancy;

}

//список рёбер

public string GetListEdges()

{

string output = "";

foreach (var edge in Edges)

output += edge.ToString() + "\n";

return output;

}

//степени вершин

public Dictionary<int, List<(int,bool)>> GetPowVertex()

{

bool orient = (bool)pInfo.btnOrient.IsChecked;

var output = new Dictionary<int, List<(int, bool)>>();

foreach(var vertex in Vertexes)

{

output.Add(vertex.GetID(), new List<(int, bool)>());

foreach(var edge in Edges)

{

if (!orient)

{

if (edge.GetFrom().Equality(vertex))

output[vertex.GetID()].Add((edge.GetTo().GetID(), true));

if (edge.GetTo().Equality(vertex))

output[vertex.GetID()].Add((edge.GetFrom().GetID(), true));

}

else

{

if (edge.GetFrom().GetID() == vertex.GetID())

output[vertex.GetID()].Add((edge.GetTo().GetID(), true));

if (edge.GetTo().GetID() == vertex.GetID())

output[vertex.GetID()].Add((edge.GetFrom().GetID(), false));

}

}

}

return output;

}

#endregion

#region Нажатие на объект

// нажатие на верщину

public Vertex PushOnVertex(Point click)

{

double minlength = 9999;

Vertex tVertex = null;

foreach (var vertex in Vertexes)

{

double length = Math.Sqrt(Math.Pow(vertex.GetCenter().X - click.X, 2) + Math.Pow(vertex.GetCenter().Y - click.Y, 2));

if ((length <= vertex.GetR()) && (length < minlength))

{

minlength = length;

tVertex = vertex;

}

}

return tVertex;

}

// нажатие на ребро

public double Distance(double x0, double y0, double x1, double y1)

{

return Math.Sqrt(Math.Pow(x1 - x0, 2) + Math.Pow(y1 - y0, 2));

}

public Edge PushOnEdge(Point click)

{

double x = click.X;

double y = click.Y;

foreach(var edge in Edges)

{

double x1 = edge.GetFrom().GetCenter().X;

double y1 = edge.GetFrom().GetCenter().Y;

double x2 = edge.GetTo().GetCenter().X;

double y2 = edge.GetTo().GetCenter().Y;

double length = Distance(x1, y1, x2, y2);

double tOne = Distance(x1, y1, x, y);

double tTwo = Distance(x, y, x2, y2);

if ((bool)pInfo.btnWeight.IsChecked)

{

if ((tOne >= 1.5 \* tTwo - 30) && (tOne <= 1.5 \* tTwo + 30) && (tOne + tTwo <= length + 1.5))

return edge;

}

else

{

if (tOne + tTwo <= length + 0.1)

return edge;

}

}

return null;

}

#endregion

//проверка ребра на вхождение

public Edge searchEdge(Edge sEdge, bool orient)

{

if (orient)

{

foreach (var edge in Edges)

{

if (edge.Equality(sEdge))

return edge;

}

return null;

}

else

{

foreach (var edge in Edges)

{

var nEdge = new Edge(edge.GetTo(), edge.GetFrom(), -1, edge.GetCanvas());

if (edge.Equality(sEdge) || nEdge.Equality(sEdge))

return edge;

}

return null;

}

}

//печать выходных данных

public void printMatrixs()

{

bool orient = (bool)pInfo.btnOrient.IsChecked;

bool weight = (bool)pInfo.btnWeight.IsChecked;

pInfo.tbAdjacencyMatrix.Text = pM.printMatrix(GetMatrixAdjacency(orient));

pInfo.tbIncidentMatrix.Text = pM.printMatrix(GetMatrixIncidence(orient));

pInfo.tbLibraMatrix.Text = pM.printMatrix(GetMatrixWeight(orient, weight));

pInfo.tbListAdjacency.Text = pM.printListAdjacency(GetListAdjacency(orient, weight));

pInfo.tbListEdges.Text = GetListEdges();

pInfo.tbPowVertex.Text = pM.printPow(GetPowVertex(), orient);

}

//отрисовка графа

public void DrawGraph()

{

canvas.Children.Clear();

foreach (var vertex in Vertexes)

vertex.Draw();

foreach (var edge in Edges)

edge.Draw((bool)pInfo.btnOrient.IsChecked, (bool)pInfo.btnWeight.IsChecked);

}

}

}