КУРСОВАЯ РАБОТА (ПРОЕКТ) ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, канд. техн. наук |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ | | | | | |
| РЕАЛИЗАЦИЯ АЛГОРИТМА ПРИМА ДЛЯ ПОИСКА МОД СВЯЗАННОГО ВЗВЕШЕННОГО НЕОРИЕНТИРОВАННОГО ГРАФА НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++ | | | | | |
| по дисциплине: Дискретная математика | | | | | |
|  | | | | | |
|  | | | | | |
| РАБОТУ ВЫПОЛНИЛ  СТУДЕНТ ГР. № | P3122 |  |  |  | К. Е. Кузьмичева | |
|  |  |  | подпись, дата |  | инициалы, фамилия | |

Оглавление

[Введение 3](#_Toc514525004)

[1 Анализ Задачи 4](#_Toc514525005)

[**1.1** **Основные определения** 4](#_Toc514525006)

[**1.2** **Реализация алгоритма Прима** 4](#_Toc514525007)

[**1.3** **Вывод информации для пользователя** 6](#_Toc514525008)

[2 Проектирование программного инструмента 6](#_Toc514525009)

[**2.1** **Структура программы** 6](#_Toc514525012)

[**2.2** **Псевдокод Алгоритма Прима** 8](#_Toc514525012)

[3 Реализация программного инструмента 9](#_Toc514525013)

[4 Тестирование 10](#_Toc514525015)

[**4.1** **Тест №1** 10](#_Toc514525016)

[**4.2** **Тест№2** 10](#_Toc514525017)

[**4.3** **Тест№3** 11](#_Toc514525018)

[**4.4** **Тест№4** 12](#_Toc514525018)

[Заключение 14](#_Toc514525019)

Список использованных источников [15](#_Toc514525020)

[Приложение 16](#_Toc514525020)

# **Введение**

При проектировании железных дорог, линий электропередачи и других линий коммуникации возникает проблема построения сети с минимальными затратами. В теории графов такая задача успешно решается путем построения минимального остовного дерева неориентированного графа. Данная задача имеет несколько методов решения. Один из них – алгоритм Прима. Суть этого метода заключается в последовательном добавлении к остову минимального, «безопасного» ребра (ребра, которое не образует цикла). В данной работе представлена программа, базирующаяся на алгоритме Прима, которая вычисляет минимальное остовное дерево неориентированного графа и делает визуализацию графа.

# **Анализ Задачи**

## **Основные определения**

Остовное дерево (англ. spanning tree) графа — ациклический связный подграф данного связного неориентированного графа.

Минимальное остовное дерево (англ. minimum spanning tree) графа — это его ациклический связный подграф, в который входят все его вершины, обладающий минимальным суммарным весом ребер.

Алгоритм Прима — алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

## **Реализация алгоритма Прима**

Алгоритм Прима дает точное решение задачи построения минимального остовного дерева (МОД) T∗(X, U∗) связного взвешенного n, m - графа G(X, U). Исходный граф задается в виде матрицы весов ||W||n×n. Алгоритм строит согласно теореме Прима цепочку из n - поддеревьев МОД T′→T′′→⋯→T∗, где T′ – граф, состоящий из одной произвольно выбранной вершины графа G(X, U), например, из вершины x1.

Наращивание МОД T∗(X, U∗) в алгоритме Прима происходит последовательно. Алгоритм состоит из следующих 2-х этапов:

1. Построение поддерева T′.
2. Построение цепочки поддеревьев МОД.

Введем следующие обозначения:

g – индекс вершины графа G(X,U), выбранной для построения очередного поддерева в цепочке;

S – суммарная стоимость ребер очередного дерева в цепочке.

На этапе построения поддерева T′ выполняются следующие действия:

g:=1

S:=0

На этапе построения цепочки поддеревьев МОД в цикле от i=1 до (n−1) строится множество ребер МОД – U∗={u1∗, u2∗, ⋯, u(n−1)∗}. Тело цикла включает в себя следующие действия:

В матрице W вычеркиваем g-столбец и выделяем g-строку.

В матрице W просматриваем все выделенные строки сверху вниз и слева направо и выбираем минимальный элемент wij (если таких элементов несколько, то выбираем первый по счету элемент).

Строим очередное поддерево МОД в цепочке путем добавления новой вершины xj и ребра (xi, xj).

g:=j

S:=S+wij

После завершения цикла МОД T∗(X, U∗) построено, S – суммарный вес его ребер.

**На первом этапе** вначале полагается, что W0={ν}, вершина ν получает метку 0 – номера фронта волны. Очередной фронт волны Wi, (i>0) строится из образов вершин предыдущего фронта волны Wi−1 за исключением тех вершин, которые ранее были размечены алгоритмом; все вершины фронта волны Wi получают метку i. Фронт волны Wl, в который входит вершина конца маршрута – p, определяет длину минимального маршрута – l.

**На втором этапе**искомый маршрут μ∗=(ν, ⋯, p) строится по фронтам волны W0, W1, ⋯, Wl следующим образом. Рассмотрим последний фронт волны Wl и положим, что i=l, μi∗=p. Далее рассмотрим предыдущий фронт волны Wi−1 и найдем множество вершин A, претендующих на включение в маршрут – A=Wi−1∩Γ−1(μi∗), где Γ−1(μi∗) – множество прообразов вершины, выбранной в маршрут μ∗ на i-шаге. Если множество A содержит не одну, а несколько вершин, то выбор конкретной вершины в маршрут μ∗ осуществляется по правилу выбора. Например, выбирается первая по счету вершина, или выбирается последняя по счету вершина или вершина выбирается случайным образом. Установим i=i−1 и определим выбранную из множества A вершину как μi∗. Рассмотрим следующий Wi−1 фронт волны и т.д. до тех пор, пока не будет включена в маршрут μ∗ вершина ν. После завершения этого этапа будет построен минимальный маршрут μ∗=(ν, ⋯, p)длиной l.

## **Вывод информации для пользователя**

Для вывода итоговых и промежуточных результатов будет использоваться формат векторной графики SVG. Отображения вершин графа будет осуществляться по заранее подготовленным сеткам, выбранным в зависимости от количества вершин.

# **Проектирование программного инструмента**

## **Структура программы**

* Модуль отрисовки графа

Coords – класс точек центра вершин графов

* x
* y

Graph – класс графа

* r – радиус вершин
* capacity – количество вершин
* vertexCoord[] – координаты центров вершин

symbol[] – имена вершин графа

* Алгоритмический модуль

matrix[][] - целочисленный массив для хранения матрицы весов исходного графа. Значения matrix[i][i] всегда равны 0. При отсутствии ребра между вершинами i и j будем также обнулять matrix[i][j].

lightrows[] - логический массив, указывающий на номера вершин графа, включенных в остовное дерево.

min – целочисленная переменная для поиска минимального по весу ребра.

v1, v2 – целочисленные переменные для хранения номеров смежных вершин.

SVGLightLines – строковая переменная для записи кода svg.

allLinesIsNull() – функция, возвращающая логическое true, если все ячейки matrix[][] нулевые (т.е. для построения остовного дерева были задействованы все вершины исходного графа), иначе false.

calc() – главная функция, которая будет запускаться после заполнения матрицы весов. Результатом ее работы будет .html файл с пошаговым отображением состояния остовного дерева исходного графа.

Первая вершина в матрице весов всегда будет корневой для искомого остовного дерева. Поэтому первым шагом будем выделять первую строку (lightrows[0] := true) и вычеркивать первый столбец (matrix[i][0] := 0) матрицы весов.

Затем циклично, пока не будут включены в остовное дерево все вершины исходного графа, будем выполнять следующие действия:

* искать ненулевой min в выделенных строках (при равных весах потенциальных ребер для добавления в остовное дерево приоритет устанавливается слева направо и сверху вниз в соответствии с матрицей весов);
* дополнить SVGLightLines кодом отрисовки нового ребра веса min, инцидентного вершинам v1 и v2;
* выделить новую строку и вычеркнуть новый столбец.

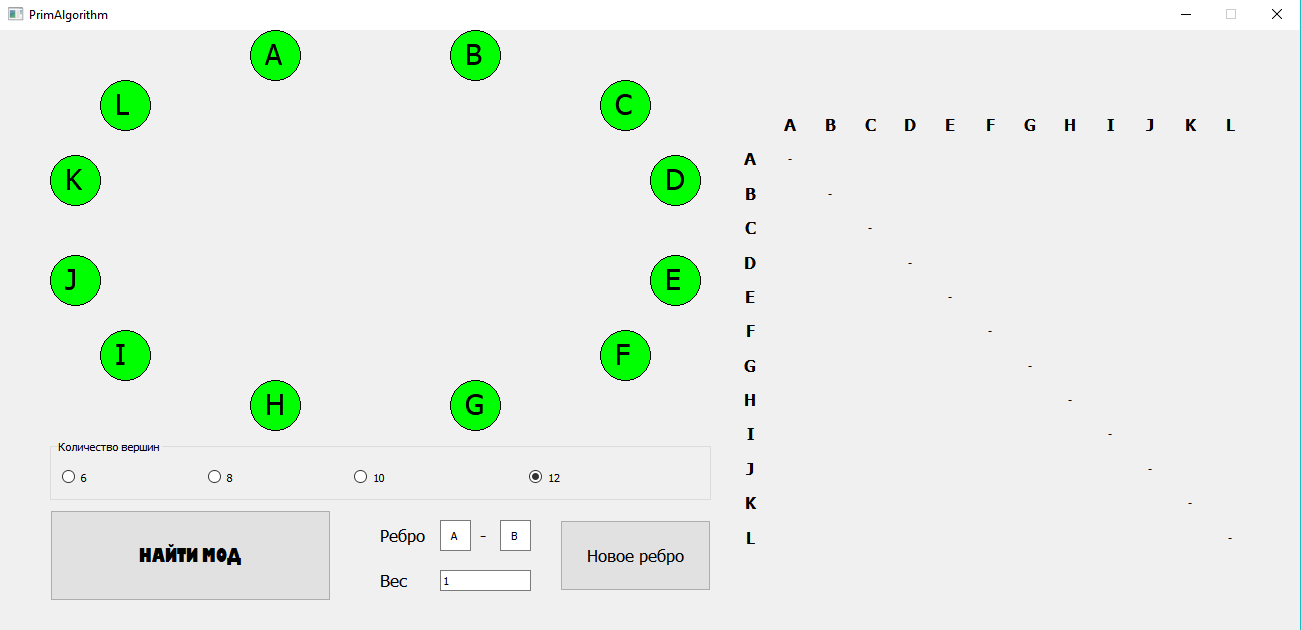


Рисунок 1 – Окно программы

## **Псевдокод Алгоритма Прима**

1. Пользователь выбирает количество вершин [6, 8, 10, 12] и задает ребра графа, сразу определяя их вес, пока граф не станет связным;
2. Пользователь нажимает кнопку «НАЙТИ МОД», тем самым запуская выполнение алгоритма;
3. lightrows[1..11] := false;
4. lightrows[0] := true; //выделяем первую строку
5. matrix[0..11][0] := 0; //вычеркиваем первый столбец
6. while(!allLinesIsNull()) //пока не все ячейки матрицы равны нулю
   1. min := 0;
   2. for (i := 0..11) //присвоим min первое ненулевое значение матрицы
      1. for (j := 0..11)
         1. if (matrix[i][j] != 0 and lightrows[i]) min := matrix[i][j];
      2. if (min != 0) break;
   3. for (i := 11..0) //найдем минимальный вес среди выделенных строк
      1. for (j := 11..0)
         1. if (matrix[i][j] != 0 and lightrows[i] and matrix[i][j] <= min)
            1. min = temp[i][j];
            2. v1 = i;
            3. v2 = j;
   4. SVGLightLines += svg код отрисовки графа с выделением ребер остовногодерева, включая новое, инцидентное вершинам v1 и v2;
   5. lightrows[v2] = true; //выделяем новую строку
   6. matrix[0..11][v2] = 0; //вычеркиваем новый столбец столбец
7. Создание .html файла;
8. Открытие файла;
9. Добавление в файл html-тегов;
10. Добавление в файл SVGLightLines;
11. Добавление в файл закрывающих html-тегов;
12. Закрытие файла.

# **Реализация программного инструмента**

Для визуализации взаимодействия классов в программе была представлена следующая диаграмма (рис. 2)

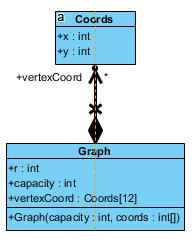


Рисунок 2 – UML-диаграмма классов

В проектировании данной диаграммы был использован инструмент Visual Paradigm 15.0.

# **Тестирование**

Были проведены 4 теста для 6, 8, 10 и 12 вершин соответственно. Использовались как полные, так и неполные графы с весами ребер от -100 до 100. В качестве демонстрации результата приведены начальная матрица весов и результат работы программы в виде svg-изображения исходного графа с выделенными ребрами его минимального остовного дерева.

## **Тест №1**

Зададим полный 6-вершинный граф и получим его остовное дерево (см. рис. 3 – рис. 4)

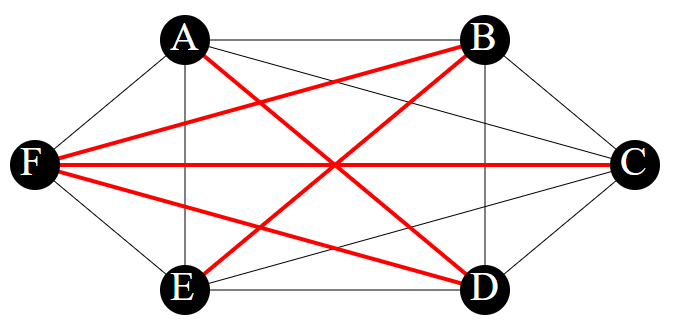


Рисунок 4 – Результат работы программы. Красными линиями обозначены ребра остовного дерева для 6-вершинного графа.

Рисунок 3 – Матрица весов тестового 6-вершинного графа

Поэтапная прорисовка содержится в приложении (рис.11 – рис.15)

## **Тест№2**

Зададим неполный 8-вершинный граф и получим его остовное дерево (см. рис. 5 – рис. 6)

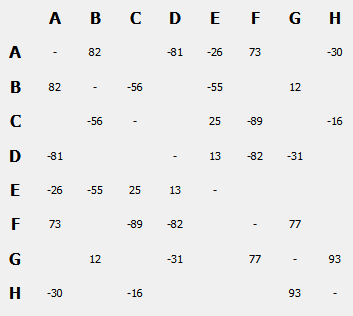
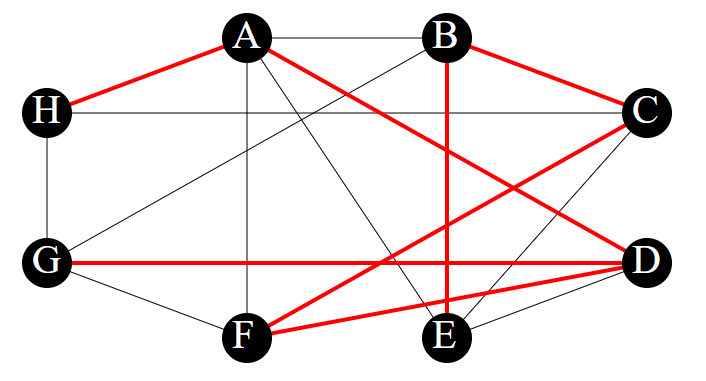
Поэтапное построение прорисовка содержится в приложении (рис.16 – рис.22)

Рисунок 6 – Результат работы программы. Красными линиями обозначены ребра остовного дерева для 8-вершинного графа

Рисунок 5 – Матрица весов тестового 8-вершинного графа

## **Тест№3**

Зададим неполный 10-вершинный граф и получим его остовное дерево (см. рис. 7 – рис. 8)

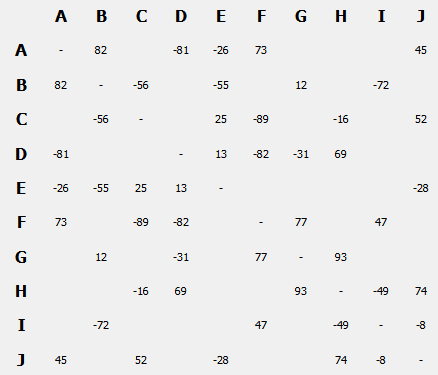


Рисунок 7 – Матрица весов тестового 10-вершинного графа

## **Тест№4**

Зададим неполный 12-вершинный граф и получим его остовное дерево (см. рис. 9 – рис. 10)

Рисунок 8 – Результат работы программы. Красными линиями обозначены ребра остовного дерева для 10-вершинного графа

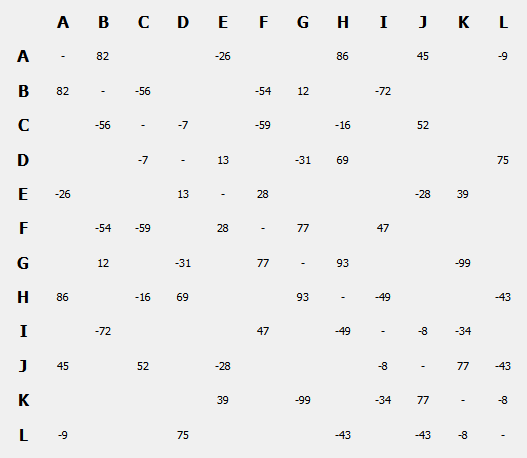


Рисунок 9 – Матрица весов тестового 12-вершинного графа

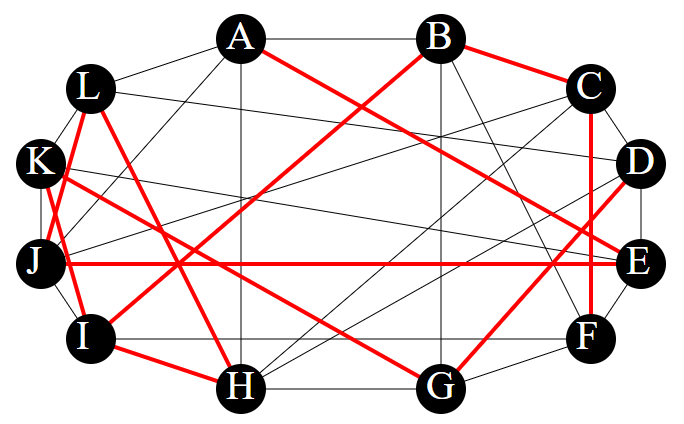
****

Рисунок 10 – Результат работы программы. Красными линиями обозначены ребра остовного дерева для 12-вершинного графа

# **Заключение**

В ходе проделанной работы была написана программа, реализующая алгоритм Прима. В результате программа создает .html файл с svg-изображениями остовного дерева исходного графа на каждом шаге алгоритма. Реализация производилась на языке программирование C++ с использованием фреймворка Qt.

# **Список использованных источников**

1. Eisenberg, J. SVG Essentials / J. Eisenberg, A. Bellamy. — Köln: O'Reilly Media, 2014. — 366 с.
2. Касьянов, В.Н. Графы в программировании: обработка, визуализация и применение / В.Н. Касьянов, В.А. Евстигнеев. — Спб.: БХВ-Петербург, 2003. — 1104 с.: ил.
3. Кормен, Т. Х. Алгоритмы: построение и анализ / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. — М.: Вильямс, 2005. — 1296 с.
4. Кристофидес, Н. Теория графов. Алгоритмический подход / Н. Кристофидес. — М.: ДМК Пресс, 2003. — 356 с.: ил.
5. Седжвик, Р. Фундаментальные алгоритмы на С++. Алгоритмы на графах: Пер. с англ. / Р. Седжвик. — СПб.: ООО «ДиаСофтЮП», 2002. — 496 с.
6. Шлее, М. Qt 5.3. Профессиональное программирование на C++ / М. Шлее. — СПб.: БХВ-Петербург, 2015. — 928 с.: ил.

# **Приложение**

Рисунок 11 – Шаг 1 алгоритма для 6-вершинного графа

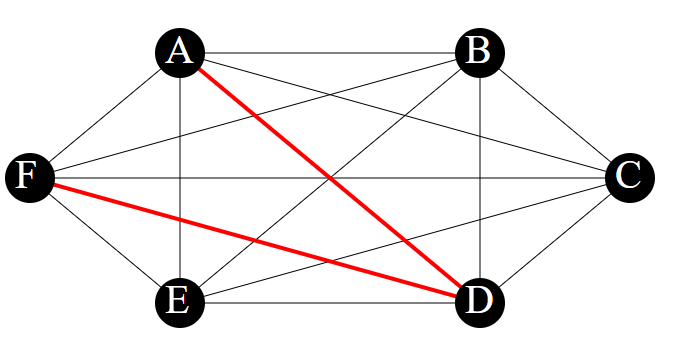


Рисунок 12 – Шаг 2 алгоритма для 6-вершинного графа

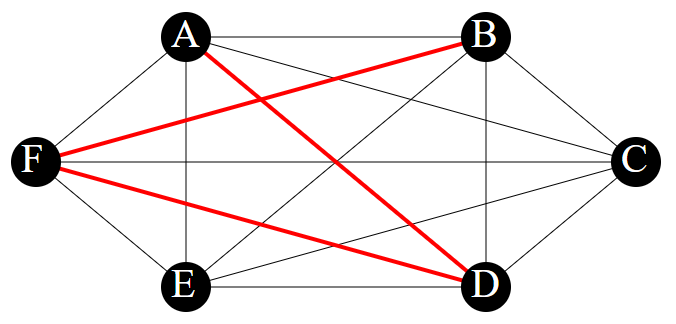


Рисунок 13 – Шаг 3 алгоритма для 6-вершинного графа

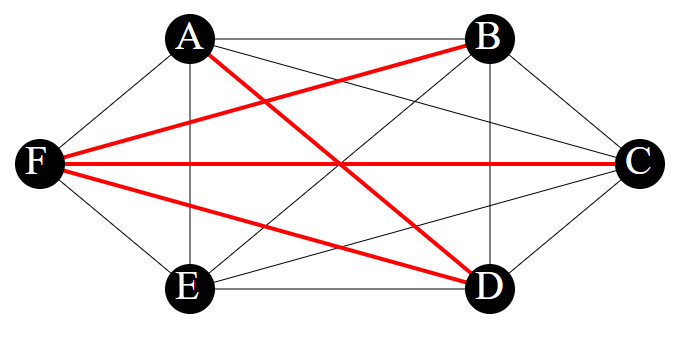


Рисунок 14 – Шаг 4 алгоритма для 6-вершинного графа

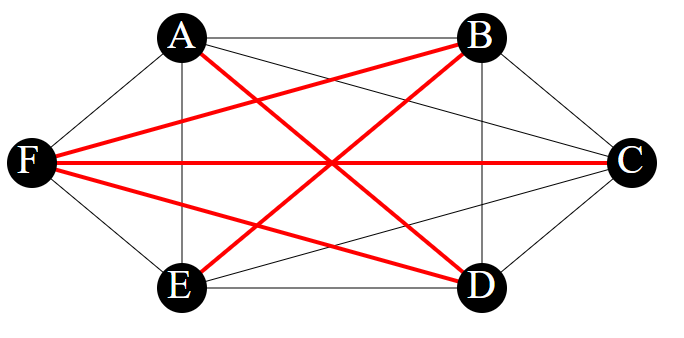


Рисунок 15 – Шаг 5 алгоритма для 6-вершинного графа. Конечный результат

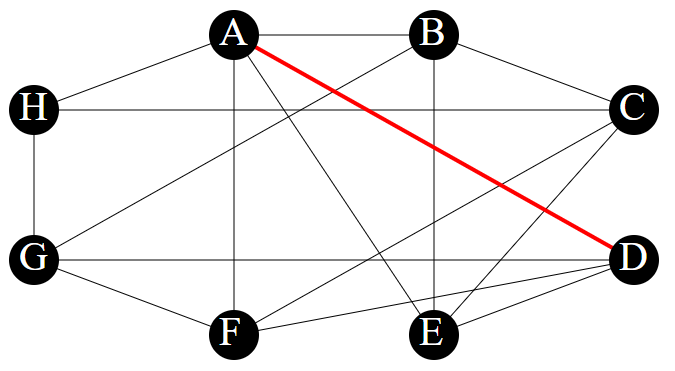


Рисунок 16 – Шаг 1 алгоритма для 8-вершинного графа

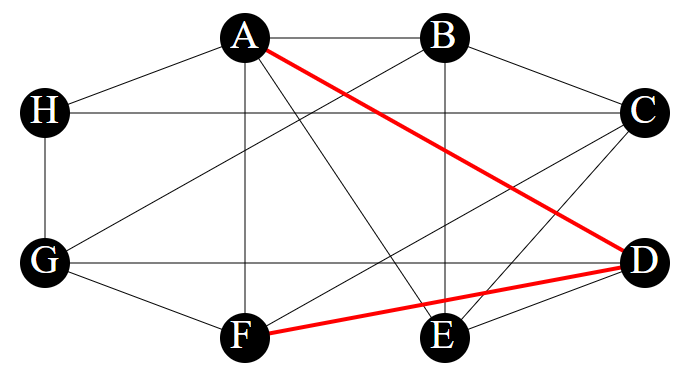


Рисунок 17 – Шаг 2 алгоритма для 8-вершинного графа

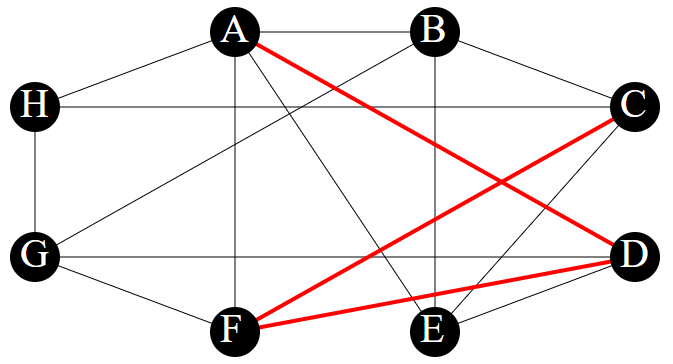


Рисунок 18 – Шаг 3 алгоритма для 8-вершинного графа

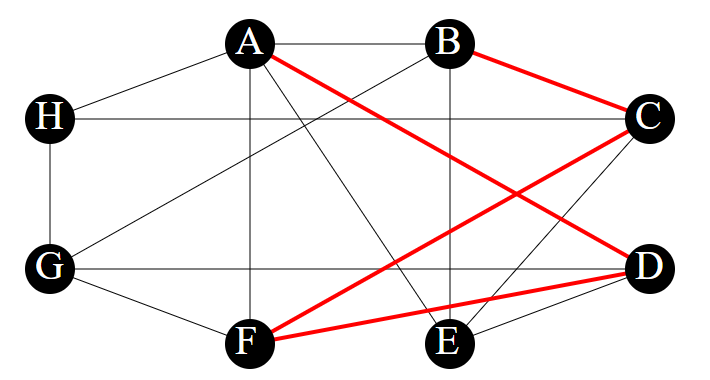


Рисунок 19 – Шаг 4 алгоритма для 8-вершинного графа

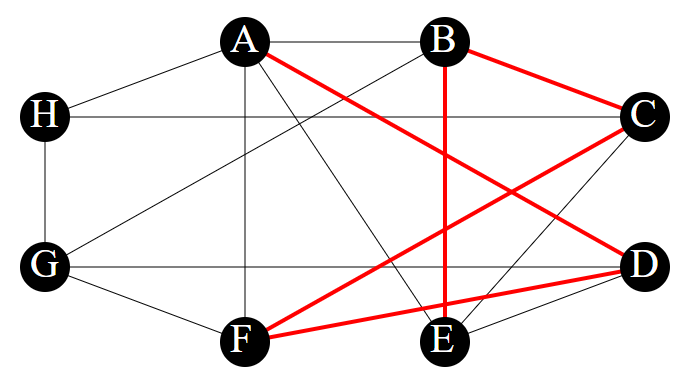


Рисунок 20 – Шаг 5 алгоритма для 8-вершинного графа

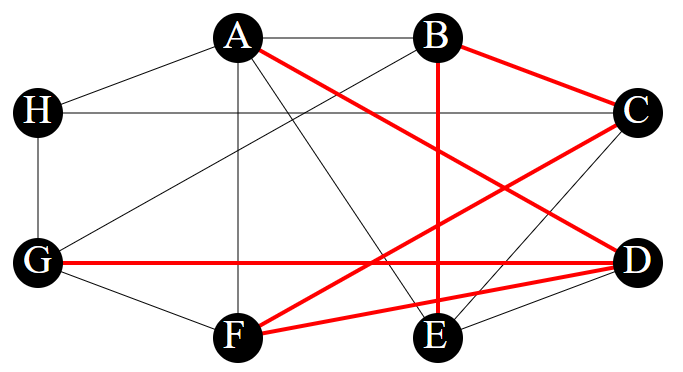


Рисунок 21 – Шаг 6 алгоритма для 8-вершинного графа

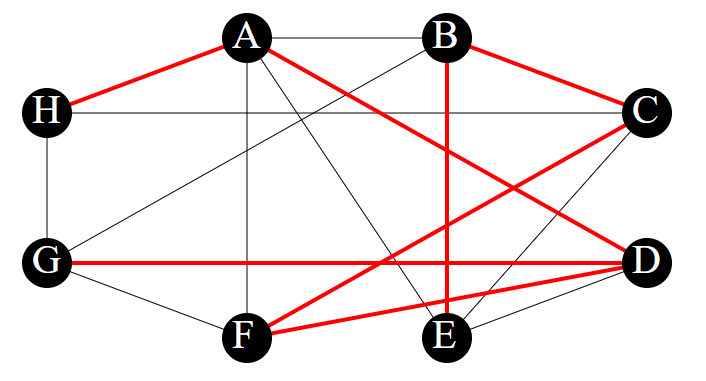


Рисунок 22 – Шаг 7 алгоритма для 8-вершинного графа. Конечный результат