

Министерство образования Республики Беларусь

Учреждение образования

**Международный государственный экологический институт
имени А. Д. Сахарова БГУ**

Факультет мониторинга окружающей среды
Кафедра информационных технологий в экологии и медицине

**НАХОЖДЕНИЕ КРАТЧАЙШЕГО ПУТИ МЕЖДУ
ВЕРШИНАМИ ГРАФА**

Курсовой проект студента 2–го курса
Зюсько Михаила Дмитриевича

_____ М. Д. Зюсько

«Допустить к защите»
Зав. кафедрой информационных
технологий в экологии и медицине,
к.б.н., доцент

Научный руководитель
К.ф.–м.н., доцент.
_____ Т. В. Смирнова

_____ В. В. Журавков
« ____ » _____ 2021 г.

Минск 2021

Министерство образования Республики Беларусь
Учреждение образования
Международный государственный экологический институт
имени А. Д. Сахарова БГУ

Факультет мониторинга окружающей среды
Кафедра информационных технологий в экологии и медицине

Пояснительная записка к курсовому проекту
по дисциплине «Исследование операций»
на тему «Нахождение кратчайшего пути между вершинами
графа»

Исполнитель,
студент гр. А91ИСТ2 _____ Зюсько М. Д.

Руководитель,
к.ф.–м.н., доцент _____ Смирнова Т.В.

Минск 2021

Содержание

СОДЕРЖАНИЕ.....	2
ВВЕДЕНИЕ	3
ГЛАВА 1 ОБЩИЕ СВЕДЕНИЯ О ГРАФАХ	5
1.1 ВИДЫ ГРАФОВ И МЕТОДЫ ИХ ЗАДАНИЯ.....	5
1.1.1 <i>Графическое представление</i>	<i>5</i>
1.1.2 <i>Матрица смежности</i>	<i>6</i>
1.1.3 <i>Матрица инцидентности</i>	<i>7</i>
1.1.4 <i>МАРШРУТЫ, ПУТИ, СВЯЗНОСТЬ</i>	<i>7</i>
ГЛАВА 2 МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО. МЕТОДЫ ОПРЕДЕЛЕНИЯ.....	9
ГЛАВА 3 ПОСТАНОВКА ЗАДАЧИ ОПТИМИЗАЦИИ.....	10
ГЛАВА 4 АЛГОРИТМ ОБРАТНОГО УДАЛЕНИЯ.....	11
4.1 ОПИСАНИЕ АЛГОРИТМА.....	11
4.2 ПСЕВДОКОД.....	11
4.3 ПРИМЕР РАБОТЫ	12
4.4 ВРЕМЕННАЯ СЛОЖНОСТЬ.....	13
ГЛАВА 5 ОБОСНОВАНИЕ ВЫБОРА АЛГОРИТМА РЕШЕНИЯ	15
ГЛАВА 6 РЕАЛИЗАЦИЯ АЛГОРИТМА ОБРАТНОГО УДАЛЕНИЯ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ	15
6.1 ШАГ 1: СОЗДАНИЕ HTML-СТРАНИЦЫ С НЕОБХОДИМЫМ ФУНКЦИОНАЛОМ И СТИЛЯМИ	15
6.2 ШАГ 2: ОБЕСПЕЧИТЬ КОМФОРТНОЕ ВЗАИМОДЕЙСТВИЕ ПОЛЬЗОВАТЕЛЯ С САЙТОМ	17
ЗАКЛЮЧЕНИЕ.....	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	21
ПРИЛОЖЕНИЕ Б	29

ВВЕДЕНИЕ

Исследованием операций называется комплексная математическая дисциплина, занимающаяся построением, анализом и применением математических моделей в задаче принятия оптимальных (наилучших в каком-то смысле) решений. Методы исследования операций применяются в самых разных областях: в промышленности, строительстве, торговле, на транспорте, в медицине, планировании научных разработок, военном деле - одним словом, везде, где приходится организовывать какие-то мероприятия, направленные к достижению определенной цели. Очевидно, организовывать их следует так, чтобы они наилучшим образом способствовали достижению поставленной цели, т.е. были максимально эффективными [3].

Несмотря на многообразие задач, классифицируемых как задачи исследования операций, при их решении можно выделить некоторую общую последовательность этапов, через которые проходит любое операционное исследование:

1. Постановка задачи;
2. Идентификация переменных;
3. Построение математической модели;
4. Анализ модели, или решение задачи с помощью выбранного метода;
5. Анализ решения;
6. Проверка адекватности модели;
7. Реализация полученного решения.

В исследовании операций используется разнообразный математический аппарат. Чаще других методов для анализа моделей операций и подготовки решений используются методы математического программирования и комбинаторного анализа.

Математическое программирование — область математики, разрабатывающая теорию и численные методы решения экстремальных задач с

ограничениями, т.е. задач на экстремум функции многих переменных с ограничениями на область изменения этих переменных.

Цель настоящей работы – применить методы исследования операций для построения и анализа модели по конкретной оптимальной задаче.

Глава 1

ОБЩИЕ СВЕДЕНИЯ О ГРАФАХ

Граф – это математическая модель, представленная совокупностью множества вершин и связей между ними [1].

1.1 ВИДЫ ГРАФОВ И МЕТОДЫ ИХ ЗАДАНИЯ

Каждая пара вершин, имеющих связь, называется ребром графа. Ребра, имеющие направления, называются ориентированными. В связи с наличием/отсутствием ориентированных ребер в графе, графы делятся на:

- Ориентированные графы – содержат только ориентированные ребра;
- Неориентированные графы – содержат только неориентированные ребра;
- Смешанные графы – содержат как ориентированные ребра, так и неориентированные.

Ребро и вершина, которой оно принадлежит считаются инцидентными.

Подграф исходного графа – это граф, содержащий некоторое подмножество вершин данного графа и некоторое подмножество инцидентных им рёбер [3].

1.1.1 Графические представление

Обычно граф представляется с помощью подобной диаграммы (рис. 1.1):

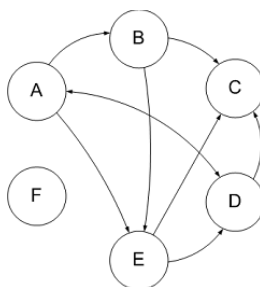


Рисунок 1.1. Ориентированный граф, с 6-ю вершинами

Данная диаграмма наглядно отображает имеющиеся вершины, и связи между ними, однако для решения некоторых задач применяются и другие способы задания графа [4].

1.1.2 Матрица смежности

Матрица смежности графа $G(V,E)$, где V – множество вершин, E – множество ребер данного графа это квадратная матрица A размера n , в которой значение элемента a_{ij} равно числу рёбер из i -й вершины графа в j -ю вершину.

Матрица смежности выглядит следующим образом (рис. 1.2, рис. 1.3):

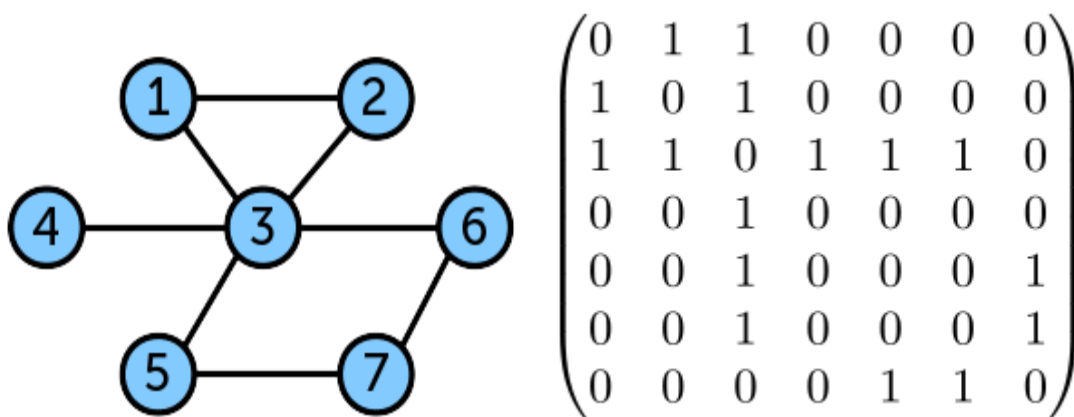


Рис. 1.2. Неориентированный граф

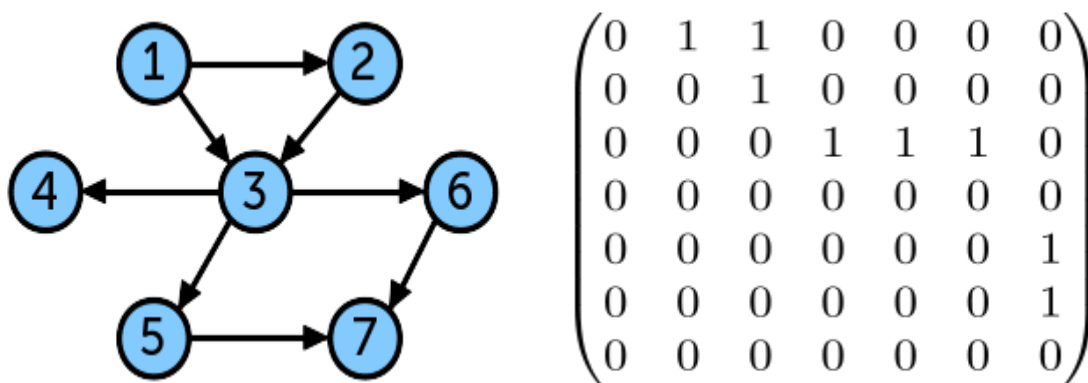


Рис. 1.3. Орграф

1.1.3 Матрица инцидентности

Матрица инцидентности – форма представления графа, в которой указываются связи между инцидентными элементами графа (ребро(дуга) и вершина). Столбцы матрицы соответствуют ребрам, строки – вершинам. Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром (их инцидентность).

Пример матрицы инцидентности (рис. 1.4, рис. 1.5):

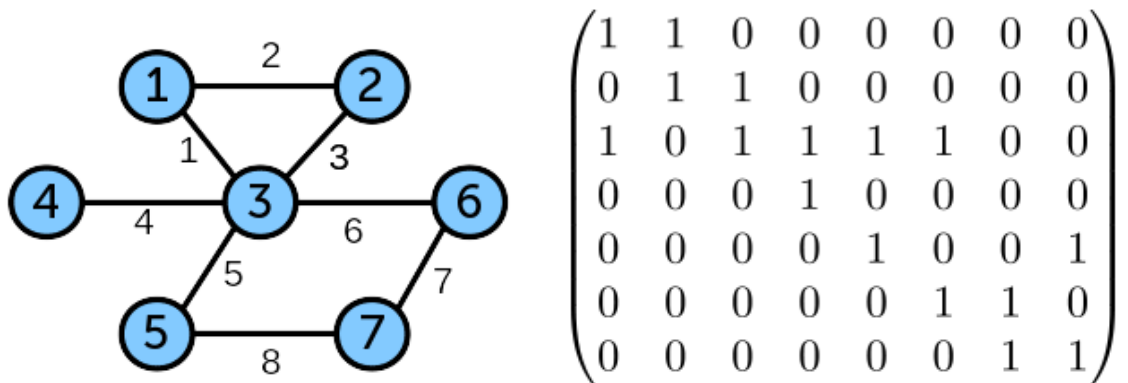


Рис. 1.4. Неориентированный граф

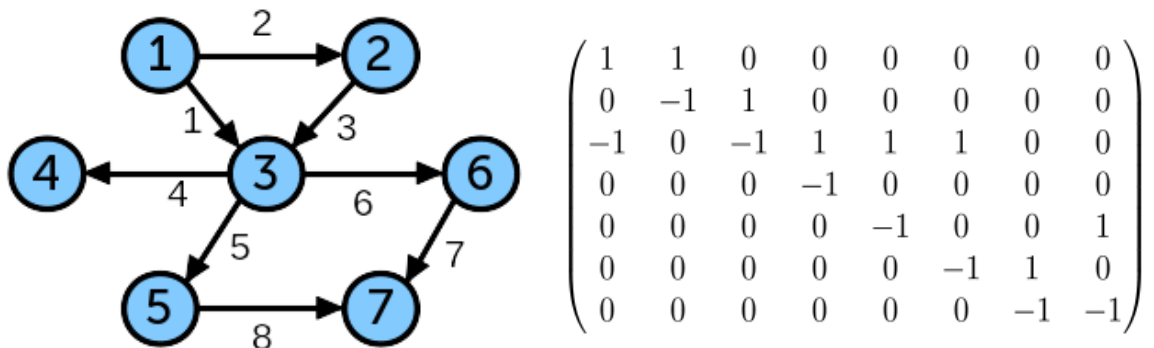


Рис. 1.5. Орграф

Также существуют другие способы, но приведенные являются наиболее емкими и популярными.

1.2 МАРШРУТЫ, ПУТИ, СВЯЗНОСТЬ

Маршрут в графе – это чередующаяся последовательность вершин и рёбер, в которой любые два соседних элемента инцидентны. Если начальная и конечная

вершины маршрута совпадают, то маршрут замкнут, иначе открыт. Маршрут называется цепью, если все его ребра различны, и простой цепью, если также различны и вершины.

Путь – это такая последовательность рёбер в графе, что конец одного ребра является началом другого. Путь можно считать частным случаем маршрута.

Граф называется связным, если любая пара его вершин соединена простой цепью. Пример связного графа (рис. 1,6):

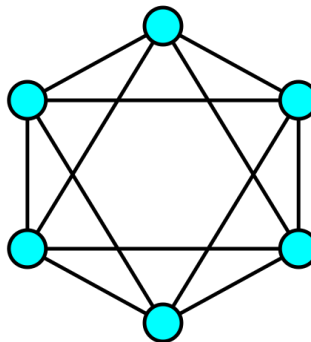


Рис 1.6. Связный граф

Пример несвязного графа (рис. 1.7):

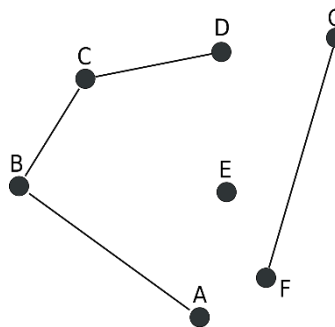


Рис. 1.7. Несвязный граф

В общем случае граф можно описать как 3 множества: X – множество вершин, A – множество дуг, C – множество весов.

$$X = \{x_1, x_2, x_3 \dots x_i\} \quad A = \{a_1, a_2, a_3 \dots a_i\} \quad C = \{c_1, c_2, c_3 \dots c_i\}$$

Глава 2

МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО. МЕТОДЫ ОПРЕДЕЛЕНИЯ

Минимальным остовным деревом (МОД) связного взвешенного графа называется его связный подграф, состоящий из всех вершин исходного дерева и некоторых его ребер, причем сумма весов ребер минимально возможная.

Задача о нахождении минимального остовного дерева часто встречается в подобной постановке: есть n городов, через которые можно проложить маршрут так, чтобы можно было добраться из любого города в любой другой (напрямую или через другие города). Требуется найти такой маршрут, чтобы стоимость проезда была максимальной [6].

Эта задача может быть сформулирована в терминах теории графов как задача о нахождении минимального остовного дерева в графе, вершины которого представляют города, рёбра – это пары городов, между которыми есть маршрут, а вес ребра равен стоимости проезда по соответствующему маршруту.

Существует несколько алгоритмов для нахождения минимального остовного дерева. Некоторые наиболее известные из них перечислены ниже:

- Алгоритм Прима;
- Алгоритм Краскала;
- Алгоритм Борувки.
- Алгоритм обратного удаления

Глава 3

ПОСТАНОВКА ЗАДАЧИ ОПТИМИЗАЦИИ

Требуется спроектировать радиотрансляционную сеть, которая должна обслуживать семь населённых пунктов так, чтобы общая протяженность линий передачи была минимальной. Расстояния между пунктами приведены в таблице (рис. 3.1).

Пункты	a	b	c	d	e	f	g
a		16	13	16	23	32	29
b	16		14	10	20	20	30
c	13	14		4	9	19	21
d	16	10	4		8	16	24
e	23	20	9	8		15	18
f	32	20	19	16	15		33
g	29	35	21	24	18	33	

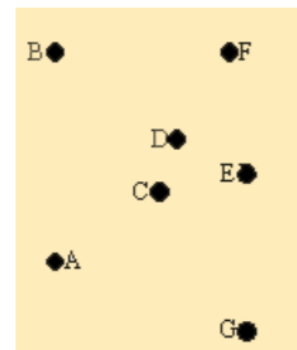


Рис.3.1. Искомая задача

Глава 4

АЛГОРИТМ ОБРАТНОГО УДАЛЕНИЯ

4.1 ОПИСАНИЕ АЛГОРИТМА

Алгоритм обратного удаления – это алгоритм в теории графов, использующийся для получения минимального остовного дерева из данного связного рёберно взвешенного графа. Если граф не связен, этот алгоритм найдёт минимальное остовное дерево для каждой отдельной части графа. Множество этих минимальных остовных деревьев называется минимальным остовным лесом, который содержит все вершины графа.

Алгоритм является жадным алгоритмом, дающим лучшее решение. Алгоритм обратного удаления начинается с исходного графа и удаляет рёбра из него. Алгоритм работает следующим образом [8]:

- Начинаем с графа G , который содержит список рёбер E .
- Проходим через E в порядке убывания веса рёбер.
- Для каждого ребра проверяем, не приводит ли его удаление к несвязному графу.
- Осуществляем удаления, не приводящие к несвязности графа.

4.2 ПСЕВДОКОД

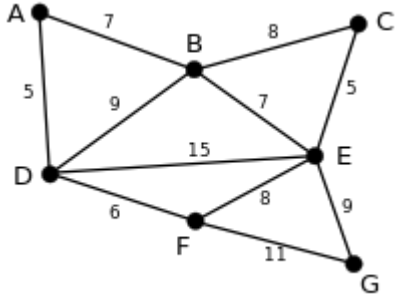
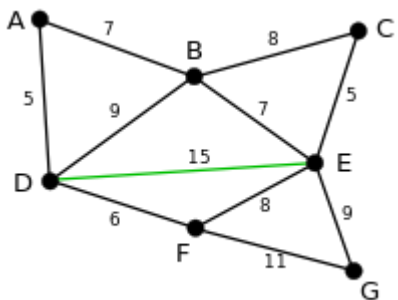
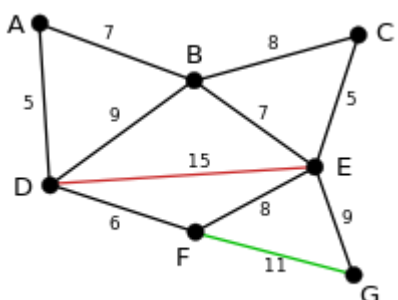
```
1 функция ReverseDelete(рёбра[] E)
2 сортируем E в убывающем порядке
3 Определяем индекс  $i \leftarrow 0$ 
4 пока  $i < \text{размер}(E)$ 
5   Определяем ребро  $\leftarrow E[i]$ 
6   удаляем  $E[i]$ 
7   если граф не связен
8      $E[i] \leftarrow \text{ребро}$ 
```

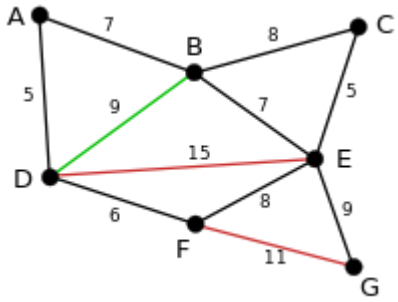
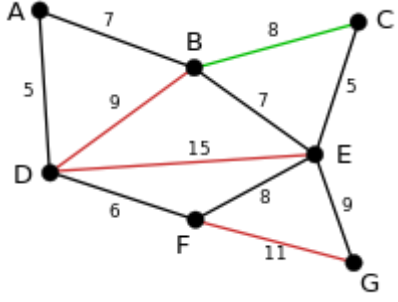
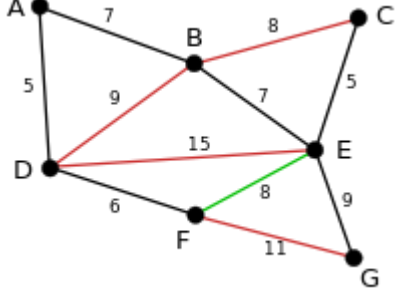
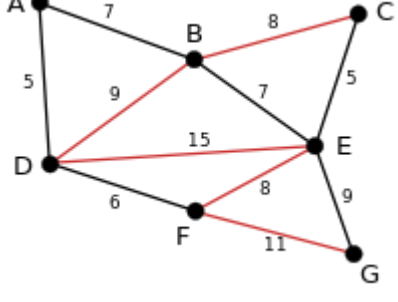
9 $i \leftarrow i + 1$
 10 возвращаем рёбра[] E

4.3 ПРИМЕР РАБОТЫ

В следующем примере зелёные рёбра просматриваются алгоритмом, а красные рёбра алгоритмом удалены [6].

Таблица №1. Пример работы алгоритма

	<p>Это исходный граф. Числа рядом с рёбрами отражают вес рёбер.</p>
	<p>Алгоритм начинает с максимального по весу ребра, в данном случае это ребро DE с весом 15. Поскольку удаление ребра DE не приводит к несвязному графу, ребро удаляется.</p>
	<p>Следующее самое тяжёлое ребро – FG, так что алгоритм проверит, не приведёт ли удаление ребра к несвязности. Поскольку удаление ребра не приводит к несвязности графа, ребро удаляется.</p>

	<p>Следующее самое тяжёлое ребро – BD, так что алгоритм проверит, не приведёт ли удаление ребра к несвязности и удаляет ребро.</p>
	<p>Следующее ребро для проверки – EG, которое удалять нельзя, поскольку это приведёт к отделению вершины G от графа. Следовательно, следующее ребро для удаления – BC.</p>
	<p>Следующее самое тяжёлое ребро – EF, так что алгоритм проверит это ребро и удаляет его.</p>
	<p>Алгоритм просматривает оставшиеся рёбра и не находит пригодных для удаления, таким образом, это финальный граф, который и возвращает алгоритм.</p>

4.4 ВРЕМЕННАЯ СЛОЖНОСТЬ

Можно показать, что алгоритм работает за время $O(E \log V (\log \log V)^3)$, где E – число рёбер, а V – число вершин. Эта граница достигается следующим образом:

- Сортируем рёбра по весу с помощью сортировки сравнения, которая работает за время $O(E \log E)$, которое может быть сокращено до $O(E \log V)$, используя факт, что самое тяжёлое ребро E входит в V^2 .
- Имеется E итераций цикла.
- Операции удаление ребра, проверки связности полученного графа и (если граф несвязен) вставки ребра могут быть сделаны за время $O(\log V (\log \log V)^3)$, на операцию.

Глава 5

ОБОСНОВАНИЕ ВЫБОРА АЛГОРИТМА РЕШЕНИЯ

Выбор данного алгоритма основан на простоте реализации алгоритма и небольшой временной сложности. Все переменные в задаче по своему физическому смыслу должны принимать неотрицательные целые значения.

Глава 6

РЕАЛИЗАЦИЯ АЛГОРИТМА ОБРАТНОГО УДАЛЕНИЯ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ

Для реализации поставленной задачи необходимо решить ряд подзадач [7]:

1. Создание HTML-страницы с необходимым функционалом и стилями;
2. Добавить мобильную версию сайта;
3. Обеспечить комфортное взаимодействие пользователя с сайтом;
4. Написать алгоритм, осуществляющий построение минимального остовного дерева.

6.1 Шаг 1: Создание HTML-страницы с необходимым функционалом и стилями

В результате выполнения имеем готовую HTML-страницу как для десктопная, так и мобильной версии сайта.

“Десктопная” версия сайта (рис. 6.1):

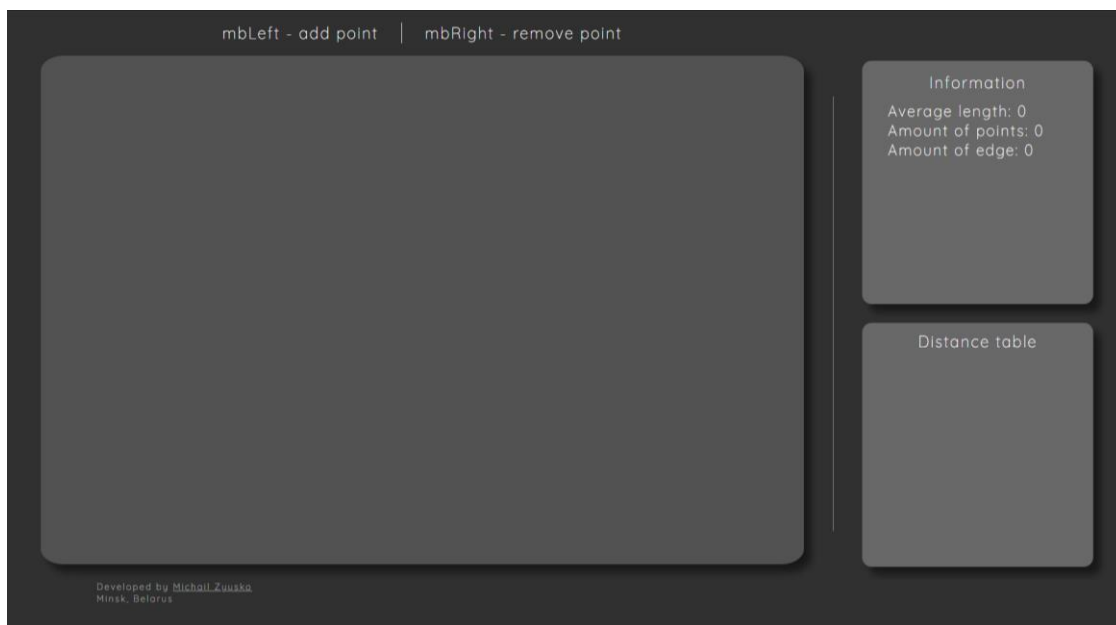


Рис. 6.1. Версия сайта на компьютере

Мобильная версия сайта (рис. 6.2):

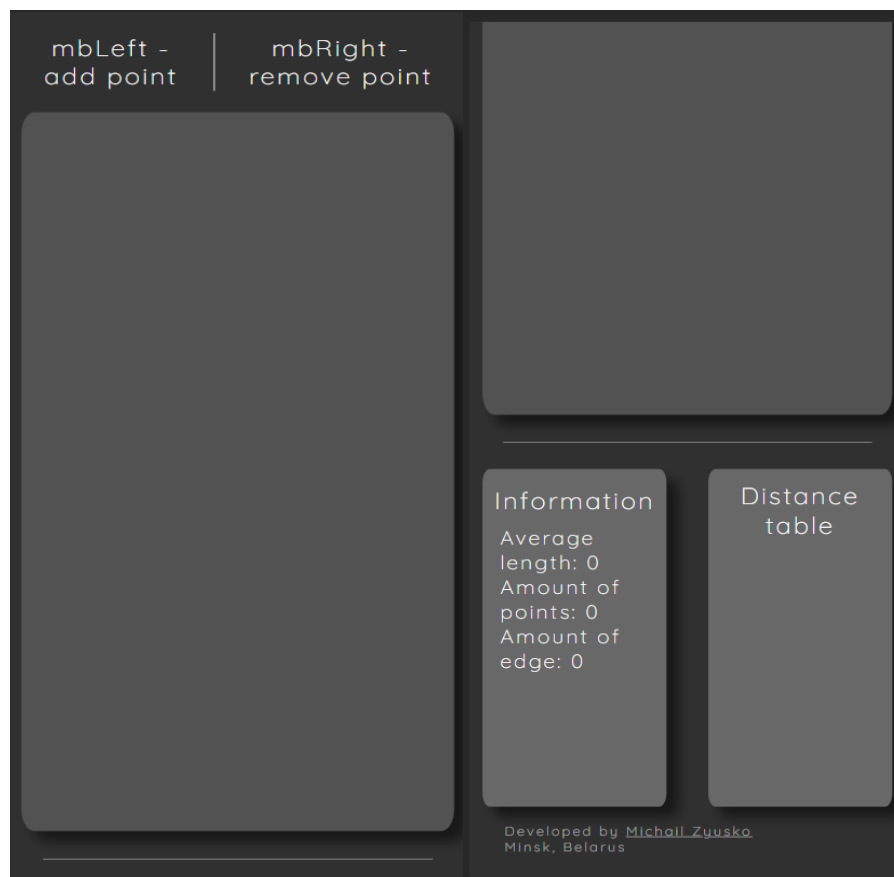


Рис. 6.2. Мобильная версия сайта

Код HTML-страницы можно посмотреть в Приложении А

6.2 Шаг 2: Обеспечить комфортное взаимодействие пользователя с сайтом

Для реализации поставленной задачи воспользуемся возможностями языка JavaScript. Он вносит интерактивность на странице. JS – это по большей части реакция на действия пользователя. В моей программе клик левой клавиши мыши приводит к созданию точки на форме, а клик правой – к удалению. Для этого были написаны функции "AddPoint" и "RemovePoint" и соответственно два события "onclick" и "oncontextmenu". Это позволило создавать точки и удалять их (рис. 6.3).



Рис. 6.3. Расстановка точек

Теперь нужно обеспечить отрисовку линий. Для этого была реализована функция "DrawLine". Она отвечает за создание и добавление на форму линий, который идут от одной точки к другой. Теперь в нашей программе появились линии (рис. 6.4).

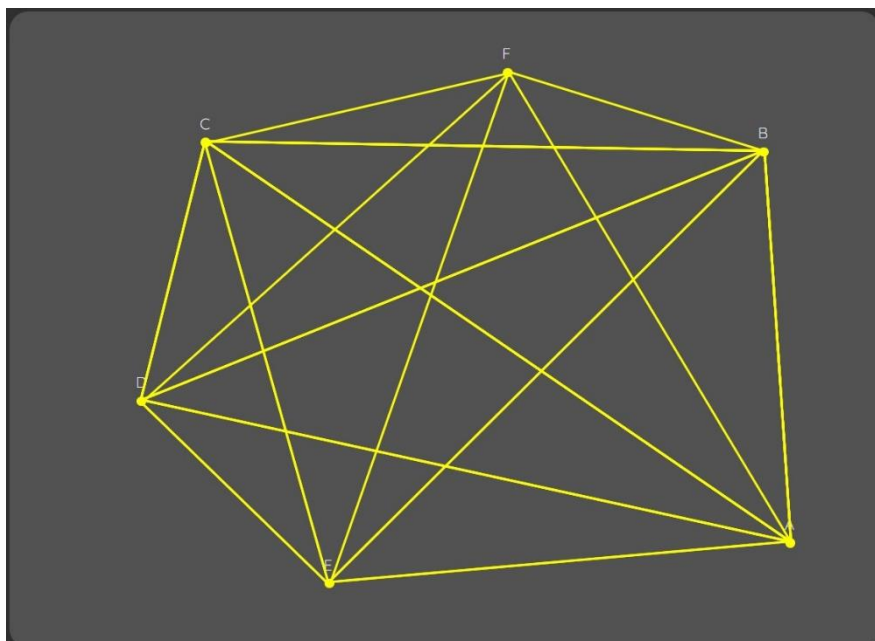
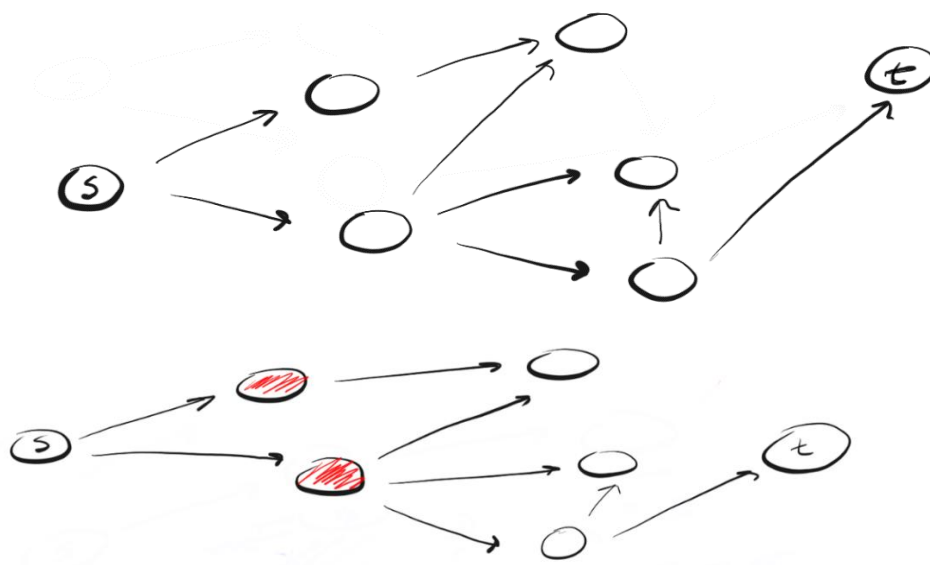


Рис. 6.4. Связный граф

Однако они не строят минимальное остовное дерево. Чтобы это происходило необходимо написать функцию с основным алгоритмом. В файле "Alghoritm.js" написана функция, в которой непосредственно реализуется алгоритм обратного удаления.

Для проверки связности графа используется алгоритм поиска в ширину. Он предполагает движение вперед по одному соседу за раз, затем посещение соседей соседей и так до тех пор, пока не будет достигнута целевая точка.

Графически это можно изобразить так (рис. 6.5):



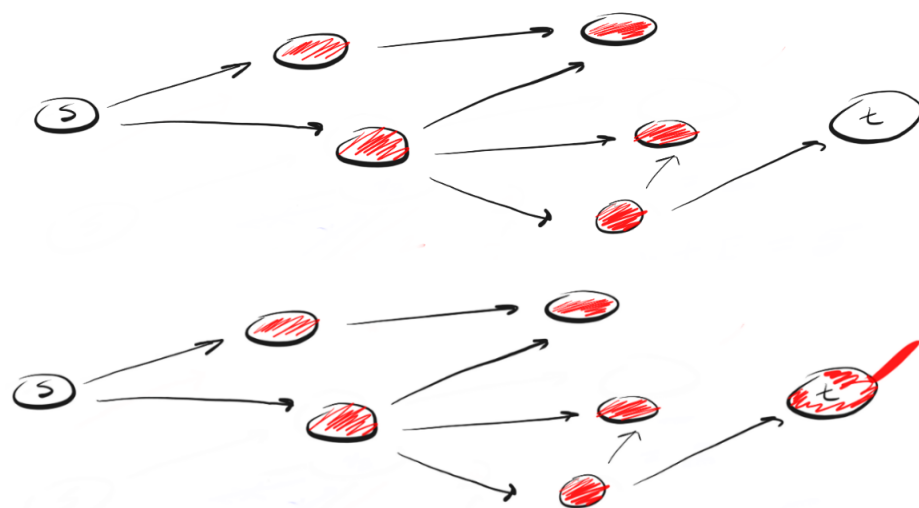


Рис. 6.5. Алгоритм работы поиска в ширину

В результате имеем готовую программу с полностью рабочим функционалом, графическим интерфейсом (рис. 6.6):

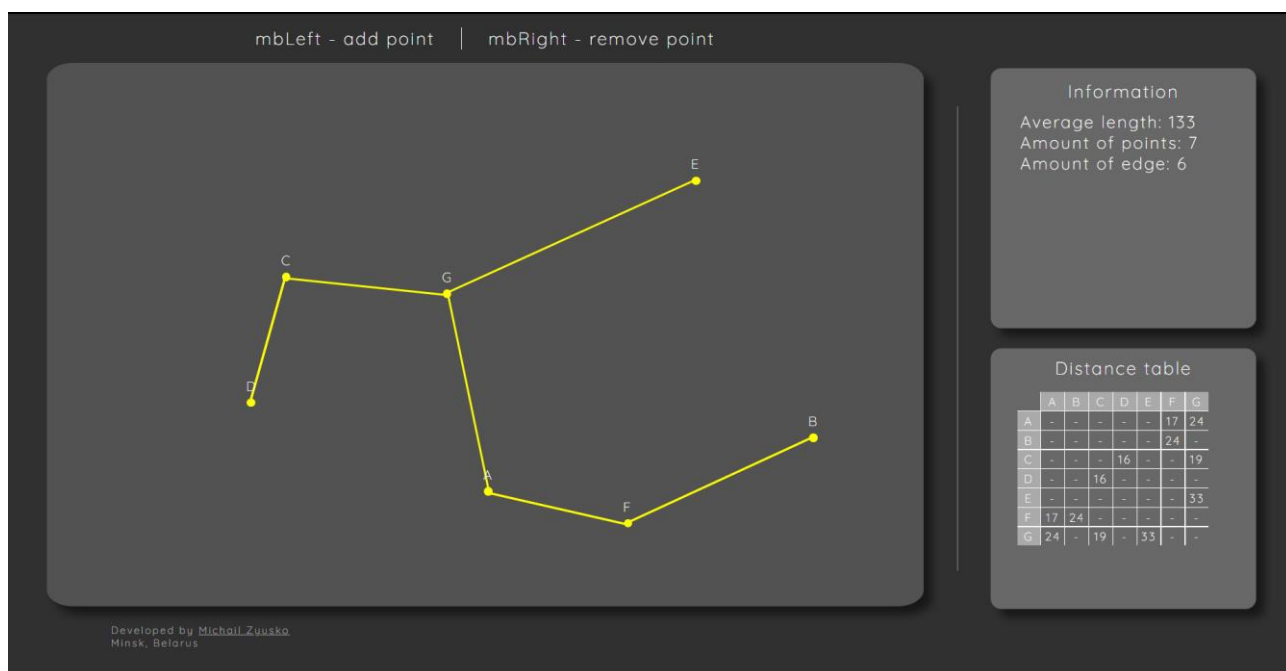


Рис. 6.6. Результат выполнения кода

Листинг с кодом программы приведен в Приложении Б.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы была написана программа, реализующая алгоритм обратного удаления. В результате программа выдает матрицу весов минимального остовного дерева графа, и изображает полученный граф. В ходе реализации алгоритма, были применены теоретические и практические знания полученные за период обучения.

Теория графов применяется во многих областях человеческой деятельности для формализации информации и выявления скрытых закономерностей. Используя алгоритмы построения минимального остовного дерева, можно решать такие логистические задачи как построение минимальной сети дорог, которую необходимо проложить между n городами таким образом, чтобы можно было добраться из любого города в любой другой (напрямую или через другие города).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Таха, Хемди А., Введение в исследование операций, 7-е издание: Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 912 с.: ил. – Парал. тит. англ.
2. Смородинский С.С., Батин Н.В. Оптимизация решений на основе методов и моделей математического программирования - Минск, БГУИР, 2003. – 136 с.
3. Смирнова Т.В. Исследование операций: учебно-методическое пособие: в 2-х ч. – Минск, 2009.
4. Кузнецов О.П., Адельсон-Вельский Г.М. Дискретная математика для инженера. - М.: Энергоатомиздат, 1988.
5. Современный учебник JavaScript [Электронный ресурс]. – Режим доступа: <https://learn.javascript.ru/> – Загл. с экрана.
6. Общедоступная многоязычная универсальная интернет-энциклопедия со свободным контентом [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/> – Загл. с экрана.
7. Кормен Т. Х. / Лейзерсон Ч. И., Ривест Р. Л., Штайн К. Алгоритмы: построение и анализ, 2-е изд. – М.: Вильямс, 2007. – 1296 с.
8. Форум программистов и сисадминов CyberForum.ru [Электронный ресурс]. – Форум начинающих и профессиональных программистов, системных администраторов, администраторов баз данных, компьютерный форум. – Режим доступа: <http://www.cyberforum.ru/> – Загл. с экрана.

Приложение А

index.html

```
<!DOCTYPE html>
<html lang="rus">

<head>
  <meta charset="UTF-16">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="CSS/general.css">
  <link rel="stylesheet" href="CSS/desctop.css">
  <link rel="stylesheet" href="CSS/mobile.css">
  <title>Вариант 49</title>
</head>

<body id="Body">
  <main>
    <div class="LeftColumn">
      <div id="Instruction">
        <span style="margin-right: 3%;">
          mbLeft - add point
        </span>
        <span style="border-left: 1px solid white;"></span>
        <span style="margin-left: 3%;">
          mbRight - remove point
        </span>
      </div>
      <div id="Draw"></div>
    </div>

    <div class="RightColumn">

      <div class="hl"></div>
      <div class="vl"></div>

      <div id="Information">
        <div class="info">
          <div style="margin-top: 5px">Information</div>
          <div style="margin: 10px 0 0 10px; text-align: left;">
            <div class="text" id="Length">Average length: 0</div>
            <div class="text" id="point_count">Amount of points: 0</div>
            <div class="text" id="edge_count">Amount of edge: 0 </div>
          </div>
        </div>
      </div>
    </div>
  </main>
</body>
```

```

        <div class="v11"></div>
        <div class="info_point" id="ShowTable">
            <div>Distance table</div>
            <div id="box"></div>
        </div>
    </div>
</div>
</main>
<footer>
    <div>Developed    by    <a    href="https://github.com/MichailZyusko">Michail
Zyusko</a></div>
    <div>Minsk, Belarus</div>
</footer>
<script type="module" src="./JS/main.js"></script>
</body>

</html>

```

desktop.css

```

@media screen and (min-width: 850px) {
    main {
        height: 90vh;
        margin: 0% 3% 0% 3%;
        display: flex;
        flex-direction: row;
    }
    footer {
        margin: 15px 3% 0% 8%;
        height: 7vh;
        font-size: 12px;
        color: rgb(255, 255, 255, 0.6);
    }
    .LeftColumn {
        flex: 3 2 65%;
    }
    .RightColumn {
        margin-top: 62px;
        flex: 1 1 25%;
        display: flex;
        text-align: center;
        align-items: center;
        flex-direction: row;
        justify-content: space-between;
        font-size: 21px;
    }
    #Instruction {
        display: flex;
        justify-content: center;
    }
}

```



```

    font-size: 21px;
    margin: 2vh 0px 2vh 0px;
    font-family: 'Quicksand', sans-serif;
    letter-spacing: 2px;
}
#Draw {
    transition: all 0.3s ease;
    height: 82vh;
    border-radius: 3%;
    background-color: rgb(97, 97, 97, 0.7);
    box-shadow: 10px 10px 10px rgba(0, 0, 0, 0.5);
}
#box {
    overflow: auto;
    margin: 5% 10% 5% 10%;
    max-width: 34vh;
    max-height: 33vh;
}
.info_point:hover {
    -webkit-transform: scale(1.25);
    -ms-transform: scale(1.25);
    transform: scale(1.25);
}
.info {
    transition: all 0.3s ease;
    background-color: rgb(159, 159, 159, 0.5);
    height: 38vh;
    border-radius: 4%;
    padding-top: 10px;
    box-shadow: 10px 10px 10px rgba(0, 0, 0, 0.5);
}
.info_point {
    transition: all 0.3s ease;
    background-color: rgb(159, 159, 159, 0.5);
    height: 38vh;
    border-radius: 4%;
    margin-top: 3vh;
    padding-top: 10px;
    box-shadow: 10px 10px 10px rgba(0, 0, 0, 0.5);
}
.vl {
    float: left;
    display: block;
    margin-left: 10%;
    margin-right: 10%;
    width: 1px;
    height: 70vh;
    background-color: rgba(255, 255, 255, 0.5);
}
.text {
    font-size: 20px;

```

```

        margin-left: 25px;
        float: left;
    }
}

```

general.css

```

@import url('https://fonts.googleapis.com/css2?family=Questrial&display=swap');
@import
url('https://fonts.googleapis.com/css2?family=Quicksand:wght@300&display=swap');
@import url('https://fonts.googleapis.com/css2?family=Staatliches&display=swap');

```

```

#Point {
    position: absolute;
    background-color: yellow;
    width: 10px;
    height: 10px;
    border-radius: 50%;
    z-index: 2;
}

```

```

#Line {
    background-color: yellow;
    transform-origin: top left;
    position: absolute;
    height: 2px;
    z-index: 1;
}

```

```

#Instruction{
    text-align: center;
}

```

```

#TextLength {
    position: fixed;
    text-align: center;
    padding: 10px 5px 0px 5px;
    background-color: #303030;
    width: 30px;
    height: 30px;
    border-radius: 50%;
    z-index: 3;
    font-size: 16px;
}

```

```

#Table {
    table-layout: fixed;
    font-size: 14px;
    width: 90%;
    height: 90%;
    border-collapse: collapse;
}

```

```

    border-style: hidden;
}

#Table tr, td {
    width: 20px;
    height: 20px;
    border: 1px solid white;
    position: relative;
}

#Table td:hover {
    background: red;
}

#Table td:hover:before {
    background-color: rgba(255, 255, 0, 0.5);
    content: "";
    height: 100%;
    left: -500px;
    position: absolute;
    top: 0;
    width: 500px;
    z-index: -1;
}

#Table td:hover:after {
    background-color: rgba(255, 255, 0, 0.5);
    content: "";
    height: 500px;
    left: 0;
    position: absolute;
    top: -500px;
    width: 100%;
    z-index: -1;
}

.caption {
    position: absolute;
    z-index: 3;
}

body {
    margin: 0px;
    background-color: #303030;
    user-select: none;
    color: white;
}

a {
    color: rgb(255, 255, 255, 0.6);
}

```

```
div, span, a, .caption, .footer {
  font-family: 'Quicksand', sans-serif;
  letter-spacing: 2px;
}
```

```
::-webkit-scrollbar {
  width: 0px;
}
```

mobile.css

```
@media screen and (max-width: 850px) {
  main {
    margin: 0% 3% 0% 3%;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: space-around;
  }
  footer {
    margin: 15px 3% 0% 8%;
    height: 7vh;
    font-size: 12px;
    color: rgb(255, 255, 255, 0.6);
  }
  .LeftColumn {
    width: 100%;
  }
  #Instruction {
    display: flex;
    justify-content: center;
    font-size: 21px;
    margin: 20px 0px 20px 0px;
  }
  #Draw {
    flex: 1 1 100%;
    transition: all 0.3s ease;
    height: 82vh;
    border-radius: 3%;
    background-color: rgb(97, 97, 97, 0.7);
    box-shadow: 10px 10px 10px rgba(0, 0, 0, 0.5);
  }
  .info_point:hover {
    -webkit-transform: scale(1.14);
    -ms-transform: scale(1.14);
    transform: scale(1.14);
  }
  .vl {
    display: none;
  }
}
```

```

.hl {
  margin: 25px 5% 25px 5%;
  width: 90%;
  height: 1px;
  background-color: rgba(255, 255, 255, 0.5);
}
#Information {
  width: 100%;
  display: inline;
  text-align: center;
  flex-direction: row;
  justify-content: space-between;
  font-size: 22px;
}
.info {
  transition: all 0.3s ease;
  float: left;
  width: 45%;
  background-color: rgb(159, 159, 159, 0.5);
  height: 38vh;
  border-radius: 4%;
  padding-top: 10px;
  box-shadow: 10px 10px 10px rgba(0, 0, 0, 0.5);
}
.info_point {
  float: right;
  width: 45%;
  transition: all 0.3s ease;
  background-color: rgb(159, 159, 159, 0.5);
  height: 38vh;
  border-radius: 4%;
  padding-top: 10px;
  box-shadow: 10px 10px 10px rgba(0, 0, 0, 0.5);
}
#box {
  overflow: auto;
  margin: 5% 10% 5% 10%;
  max-width: 34vh;
  max-height: 30vh;
}
.text {
  font-size: 18px;
  margin-left: 5px;
  float: left;
}
}

```

Приложение Б

Main.js

```
import { AddPoint } from './Point/AddPoint.js';
import { RemovePoint } from './Point/RemovePoint.js';

'use strict';

Draw.addEventListener("click", AddPoint); // Добавить точку
Body.addEventListener("contextmenu", RemovePoint); // Удалить точку
```

AddPoint.js

```
import { CreatePoint } from './CreatePoint.js'
import { CreateEdge } from '../Edge/CreateEdge.js'
import { CreateSubGraph } from '../SubGraph/CreateSubGraph.js'

'use strict';

export let Graph = new Array; // Создаем множество вершин
export let Edges = new Array; // Создаем множество ребер

let currentName = 'A'; // Задаем имя первой вершины

export function AddPoint(event) {
  if (currentName !== "[") { // Ограничиваем символами английского алфавита
    currentName = CreatePoint(Graph, event, currentName); // Создаем точку
    CreateEdge(Graph, Edges); // Создаем ребра графа
    CreateSubGraph(Graph, Edges); // Создание минимального островного дерева
  }
}
```

CreatePoint.js

```
'use strict';

export function CreatePoint(Graph, e, currentName) {

  Graph.push({ // Добавляем вершину в граф
    name: currentName,
    visited: false,
    X: e.pageX,
    Y: e.pageY,
  });
}
```

```

    document.getElementById("point_count").textContent = "Amount of points: " +
Graph.length; // меняем кол-во вершин

    let point = document.createElement('div'); // Создаем точку
    point.setAttribute("name", currentName);
    point.id = "Point";
    point.style.left = e.pageX - 5 + 'px';
    point.style.top = e.pageY - 5 + 'px';

    let caption = document.createElement('div'); // Создаем подпись к ней
    caption.textContent = caption.id = currentName;
    caption.className = "caption";
    caption.style.left = e.pageX - 6 + 'px';
    caption.style.top = e.pageY - 30 + 'px';

    document.getElementById('Draw').appendChild(point); // Отображаем элемент
    document.getElementById('Draw').appendChild(caption); // Отображаем элемент

    return String.fromCharCode(currentName.codePointAt(0) + 1); // Изменяем на след
букву
}

```

RemvePoint.js

```

import { Graph, Edges } from './AddPoint.js';
import { CreateSubGraph } from '../SubGraph/CreateSubGraph.js'

'use strict';

export function RemovePoint(e) {

    let point = document.elementFromPoint(e.pageX, e.pageY); // Берем элемент по месту
клика

    if (point.id === "Point") { // Если это точка

        let PointName = point.getAttribute("name"); // Получаем имя точки

        document.getElementById(PointName)?.remove(); // Удаляем подпись к точке

        Graph.find(function (item, index) { // Т.к. удалили удаляем точку, то удалим ее из
графа
            if (item.name === PointName) {
                Graph.splice(index, 1);
                return true;
            }
        });
        document.getElementById("point_count").innerText = "Amount of points: " +
Graph.length; // Изменяем кол-во вершин графа
    }
}

```

```

        for (let i = Edges.length - 1; i > -1; i--) // Удаляем ребра графа, которые были связаны
с этой точкой
            if (Edges[i].name.includes(PointName)) {
                Edges.splice(i, 1);
            }

        for (let line of document.querySelectorAll('.line')) // Удаляем визуальное
отображение ребра
            if (line.getAttribute("name").includes(PointName)) {
                line?.remove();
            }

        point?.remove(); // Удаляем визуальное отображение точки

        CreateSubGraph(Graph, Edges); // Т.к. удалили точку, то исходный граф изменился
    }

    e.preventDefault(); // Чтобы не отображалось контекстное меню
}

```

AddEdge.js

```

'use strict';

export function AddEdge(edge) {

    let x, x1, y, y1; // Узнаем начальное и конечное положение линии

    x = edge.fromX;
    y = edge.fromY;
    x1 = edge.toX;
    y1 = edge.toY;

    let angle = (Math.atan((y1 - y) / (x1 - x)) * 180 / Math.PI); // Считаем угол поворота

    if ((x1 <= x && y1 <= y) || (x1 < x && y1 > y)) {
        angle -= 180;
    }

    let line = document.createElement('div'); // Создаем линию со всеми свойствами
    line.setAttribute("name", edge.name);
    line.setAttribute("length", edge.length);
    line.id = "Line";
    line.className = "line";
    line.style.left = x + 'px';
    line.style.top = y + 'px';
    line.style.width = edge.length + 'px';
    line.style.transform = 'rotate(' + angle + 'deg)';
}

```



```

document.getElementById('Draw').appendChild(line); // Отрисовываем линию
line.addEventListener("mouseover", ShowLength); // Добавляем события
line.addEventListener("mouseout", HideLength); // Добавляем события
}

function ShowLength(event) {
  let TextLength = document.createElement('div'); // Показываем длину
  TextLength.id = "TextLength";
  TextLength.textContent = Math.round(event.currentTarget.getAttribute("length") / 10);
  TextLength.style.left = event.clientX + 10 + "px";
  TextLength.style.top = event.clientY - 30 + "px";
  document.body.appendChild(TextLength);
}

function HideLength() { // Скрываем длину
  document.getElementById("TextLength")?.remove();
}

```

CreateEdge.js

```

'use strict';

export function CreateEdge(Graph, Edges) { // Проходимся по всем вершинам графа
  for (let i = 0; i < Graph.length; i++)
    for (let j = i + 1; j < Graph.length; j++)
      if (!Edges.find(item => item.name === Graph[i].name + Graph[j].name) || Edges.length
== 0) {
        // Если в массиве ребер не нашел совпадения, то добавляет новое ребро
        Edges.push({
          toX: Graph[j].X,
          toY: Graph[j].Y,
          fromX: Graph[i].X,
          fromY: Graph[i].Y,
          name: Graph[i].name + Graph[j].name,
          length: Math.round(Math.sqrt((Graph[j].X - Graph[i].X) ** 2 + (Graph[j].Y -
Graph[i].Y) ** 2)),
        });
      }
}

```

Alghoritm.js

```

import { Graph } from '../Point/AddPoint.js'

'use strict';

```

```

export function MinTreeSearch(Edges, Arr) {

    let NewEdge = Edges.slice();
    NewEdge.sort((a, b) => b.length - a.length); // Сортируем ребра по убыванию

    let result = NewEdge.slice();

    let n = NewEdge.length;
    let m = Graph.length;
    let k = 0;

    while (n > m - 1) {
        let Change = ChangeMatrix(Arr, NewEdge[k], true);

        if (!isGraphConnected(Change, Graph)) { // Если удаление ребра приведет к потере
связности графа
            Change = ChangeMatrix(Arr, NewEdge[k], false);
        }
        else { // Если удаление ребра не приведет к потере связности графа
            result.splice(result.indexOf(NewEdge[k]), 1); // Удаляем ненужно ребро
            n--;
        }
        k++;
    }

    return result; // Получаем минимальное островное дерево
}

function ChangeMatrix(Arr, obj, del) { // Изменяем связный список после удаления и в
случае неправильного удаления

    for (let point of Arr.keys()) {
        if (point == obj.name[0]) {
            let mas = Arr.get(point);
            if (!del) {
                for (let j = 0; j < Graph.length; j++)
                    if (Graph[j].name == obj.name[1]) mas.push(Graph[j]);
            }
            else {
                for (let i = 0; i < mas.length; i++)
                    if (mas[i].name == obj.name[1]) mas.splice(i, 1);
            }
        }
    }

    for (let point of Arr.keys()) {
        if (point == obj.name[1]) {
            let mas = Arr.get(point);
            if (!del) {
                for (let j = 0; j < Graph.length; j++)
                    if (Graph[j].name == obj.name[0]) mas.push(Graph[j]);
            }
        }
    }
}

```

```

    }
    else {
      for (let i = 0; i < mas.length; i++)
        if (mas[i].name == obj.name[0]) mas.splice(i, 1);
    }
  }
}

return Arr;
}

function isGraphConnected(Arr, Graph) { // проверяем граф на связность

  for (let i = 0; i < Arr.size; i++)
    for (let j = 0; j < Arr.size; j++)
      if (i != j && !bfs(Arr, Graph[i], Graph[j]))
        return false;

  return true;
}

function bfs(adj, s, t) { // Алгоритм обхода графа в ширину
  // adj - смежный список
  // s - начальная вершина
  // t - пункт назначения

  Graph.forEach(element => { // Все точки не были посещены
    element.visited = false;
  });

  // инициализируем очередь
  let queue = []
  // добавляем s в очередь
  queue.push(s)
  // помечаем s как посещенную вершину во избежание повторного добавления в
очередь
  s.visited = true
  while (queue.length > 0) {
    // удаляем первый (верхний) элемент из очереди
    let v = queue.shift();
    // abj[v] - соседи v
    for (let neighbor of adj.get(v.name)) {
      // если сосед не посещался
      if (!neighbor.visited) {
        // добавляем его в очередь
        queue.push(neighbor)
        // помечаем вершину как посещенную
        neighbor.visited = true
        // если сосед является пунктом назначения, мы победили
        if (neighbor === t) return true
      }
    }
  }
}

```

```

    }
  }
  // если t не обнаружено, значит пункта назначения достичь невозможно
  return false
}

```

CreateSubGraph.js

```

import { AddEdge } from '../Edge/AddEdge.js';
import { MinTreeSearch } from './Algoritm.js'

'use strict';

export function CreateSubGraph(Graph, Edges) {

  let SubGraph = MinTreeSearch(Edges, CreateMatrix(Graph)); // Создаем минимальное
островное дерево

  document.getElementById("Length").innerText = "Average length: " +
Math.round(SubGraph.reduce((sum, elem) => sum + elem.length, 0) / 10);
  document.getElementById("edge_count").innerText = "Amount of edge: " +
SubGraph.length;

  SubGraph.forEach(element => { // Отрисовываем минимальное островное дерево
    AddEdge(element);
  });

  for (let line of document.querySelectorAll('.line')) { // Удаляем отрисованные линии
    let result = SubGraph.find(function (item) {
      if (item.name == line.getAttribute("name"))
        return true;
    });
    if (!result) line.remove();
  }

  ChangeTable(SubGraph, Graph); // Изменяем таблицу
}

export function CreateMatrix(arr) { // Создаем связный список
  let adj = new Map;

  for (let i = 0; i < arr.length; i++) {
    adj.set(arr[i].name, []); // Добавляем точку
    for (let j = 0; j < arr.length; j++) {
      if (i != j)
        adj.get(arr[i].name).push(arr[j]); // Добавляем связанной с ней точки
    }
  }
}

```

```

    return adj;
}

export function ChangeTable(Arr, Graph) {

    document.getElementById("Table")?.remove(); // Удаляем старую таблицу

    let table = document.createElement("table"); // Создаем таблицу
    table.id = "Table";
    let n = Graph.length + 1;

    for (let i = 0; i < n; i++) {
        let tr = document.createElement("tr"); // Создаем строку
        for (let j = 0; j < n; j++) {
            let td = document.createElement("td"); // Создаем ячейку
            td.style.position = "sticky";
            if (i == 0 && j != 0) { // Фиксируем первую строку и именуем ее
                td.style.top = 0 + "px";
                td.style.backgroundColor = 'rgb(170, 170, 170)';
                td.style.zIndex = 9;
                td.innerHTML = Graph[j - 1].name;
            } else if (j == 0 && i != 0) { // Фиксируем первый столбец и именуем его
                td.style.backgroundColor = 'rgb(170, 170, 170)';
                td.style.left = 0 + "px";
                td.style.zIndex = 9;
                td.innerHTML = Graph[i - 1].name;
            } else if (i > 0 && j > 0) { // Заполняем таблицу значениями
                Arr.forEach(element => {
                    if (i != j && element.name.includes(Graph[i - 1].name) &&
                        element.name.includes(Graph[j - 1].name)) {
                        td.innerHTML = Math.round(element.length / 10);
                    }
                });
            }

            if (td.innerHTML == "") { // Отсутствует прямая дорога
                td.innerHTML = "-";
            }

            if (j == 0 && i == 0) { // Первая ячейка
                td.style.top = 0 + "px";
                td.style.left = 0 + "px";
                td.style.backgroundColor = 'rgb(104,104,104)';
                td.style.zIndex = 10;
                td.innerHTML = "";
            }
            tr.append(td); // Добавляем ячейку к строке
        }
        table.append(tr); // Добавляем строку к таблице
    }

    document.getElementById("box").appendChild(table); // Отрисовываем таблицу
}

```