МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт–Петербургский государственный университет  
аэрокосмического приборостроения»

ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ОТЧЕТ О ПРАКТИКЕ   
ЗАЩИЩЕН С ОЦЕНКОЙ г

РУКОВОДИТЕЛИ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| преподаватели |  | 24.05.2022 |  | И.Г.Бартасевич,  У.С.Опалева |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |
|  |  |  |  |  |

|  |
| --- |
| ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ |
| В СОСТАВЕ ПРОФЕССИОНАЛЬНОГО МОДУЛЯ  ПМ 02 Осуществление интеграции программных модулей |
|  |
|  |
|  |

ОТЧЕТ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | С921 |  | 01.06.2022 |  | А.А.Чиновникв |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на прохождение учебной практики обучающегося по специальности

09.02.07 «Программирование в компьютерных системах»

*код и наименование специальности*

1. Фамилия, имя, отчество обучающегося: Чиновников Андрей Андреевич
2. Группа: С921 Период практики: с «25» мая 2022г. по «07» июня 2022г.
3. Тема задания: получение первичных профессиональных умений и навыков, начального опыта практической деятельности, овладение необходимыми компетенциями по профессиональному модулю:

ПМ.02, осуществление интеграции программных модулей

1. Вопросы, подлежащие изучению:

* разработка структуры проекта, построение диаграмм UML;
* решение задачи математического моделирования;
* подключение системы контроля версий Git;
* выполнение интеграции программных модулей;
* написание модульных тестов.

1. Выполнение комплексных работ по созданию проекта на основе фреймворка Bottle, выполнению интеграции шаблонов, обработчиков форм и файлов данных, использованию методов для получения кода на языке Python с заданной функциональностью и степенью качества, автоматизации тестирования посредством платформы unittest.
2. Содержание отчетной документации:
   1. Отчёт, включающий в себя:

* титульный лист;
* индивидуальное задание;
* материалы о выполнении индивидуального задания;
* список использованных источников.
  1. Аттестационный лист.

1. Срок представления отчета заместителю декана по учебно–производственной работе: «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_\_ г.

Руководители практики от факультета СПО:

У.С. Опалева

преподаватель 11.05.2022 И. Г. Бартасевич

должность, ученая степень, звание подпись, дата инициалы, фамилия

Обучающийся

11.05.2022 А.А.Чиновников

дата подпись инициалы, фамилия

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc104243292)

[1. Техническое задание 5](#_Toc104243293)

[4. Разработка логики сайта 13](#_Toc104243294)

[5. Веб-страница по решению индивидуального задания 14](#_Toc104243295)

[6. Дополнительное задание 18](#_Toc104243296)

[7. Заключение 20](#_Toc104243297)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21](#_Toc104243298)

[ПРИЛОЖЕНИЕ А 22](#_Toc104243299)

[Структура файла README 22](#_Toc104243300)

[ПРИЛОЖЕНИЕ Б 23](#_Toc104243301)

[Структура проекта, отслеживание Git-репозитория в Visual Studio, отслеживание Git-репозитория на GitHub 23](#_Toc104243302)

[ПРИЛОЖЕНИЕ В 27](#_Toc104243303)

[Листинги программного кода и приложения 27](#_Toc104243304)

[ПРИЛОЖЕНИЕ Г 52](#_Toc104243305)

[Листинги юнит-тестов 52](#_Toc104243306)

### ВВЕДЕНИЕ

Задачей учебной практики по интеграции программных модулей по МДК 02.02 Инструментальные средства разработки программного обеспечения и МДК 02.03 Математическое моделирование является закреплением теоретического материала междисциплинарного курса МДК 02.02 и МДК 02.03, а также получение практических навыков по созданию и использованию юнит-тестов и сайта.

Для реализации поставленной задачи необходимо разработать сайт используются возможности PyCharm с поддержкой фреймворка Bottle.

### Техническое задание

* 1. **Введение**

Данное техническое задание распространяется на разработку сайта по решению задач математического моделирования, а именно задач на поиск минимального остовного дерева графа посредством различных алгоритмов:

Целью работы учебной практики является разработка сайта по решению задач математического моделирования, а именно задач на поиск минимального остовного дерева графа посредством алгоритма Краскала.

Вариант 3. Требуется найти кратчайший остов данного графа с помощью алгоритма Краскала.

* 1. **Основания для разработки**

Задание на разработку в рамках прохождения учебной практики по ПМ02 «Осуществление интеграции программных модулей».

* 1. **Назначение разработки**

Основным назначением является пользовательское решение таких задач как поиск минимального остова при помощи алгоритма Крускала (Краскала).

* 1. **Требования к программе и программному изделию**
     1. **Требования к графическому дизайну сайта**

При разработке сайта должны быть использованы любые цветовые решения.

Основные разделы сайта должны быть доступны с первой страницы.

На первой странице не должно быть большого объема текстовой информации. В дизайне сайта не должны присутствовать:

* мелькающие баннеры,
* много сливающегося текста,
* тёмные и агрессивные цветовые сочетания и графические решения.

**1.4.2. Требования к функциональным характеристикам**

**1.4.2.1. Классы пользователей**

Гость – пользователь, обладающий следующими правами:

* Разделы сайта со статической информацией – просмотр.
* Разделы с нахождением минимального остовного дерева графа алгоритмом Краскала - ввод данных и просмотр результата.
* Разделы сайта с кнопками перехода – переход на другие страницы.
* Разделы с выполнением функций в соответствии с ТЗ.

**1.4.2.2. Требования к функциональной части**

Необходимо решить задачу по поиску кратчайшего остовного дерева алгоритмом Краскала. Реализовать проверку корректности введенных пользователем данных, при некорректном вводе указать, какая конкретно ошибка была допущена. Если данные были введены корректно, отобразить решение поставленной задачи. В результате пользователь получает подробное решение с объяснением работы алгоритма.

На форме для заполнения доступны следующие значения: количество вершин и ребер графа, таблица, в которой указываются две вершины, соединенные ребром, и вес данного соединения. При реализации полей ввода учесть, что у определенного количества вершин имеется максимально возможное число ребер.

**1.4.2.3. Требования к представлению сайта**

Сайт должен состоять из главной страницы с заголовком, согласно определённой тематике, кратким её описанием и ссылками (в виде текста, картинок или кнопок) на соответствующие методы решения задачи; страниц с реализацией методов решения задачи, где пользователь может ввести исходные данные для расчета задачи, страниц с выводом результатов решения, страницы «Об авторах» с фотографиями разработчиков, ФИО и кратким пояснением о личном вкладе в проект приложения. На странице для нахождения соответствующей задачи должен быть заголовок с наименованием метода; текст с объяснением его сути с сопровождением примера; формы с элементами управления для ввода исходных данных.

* + 1. **Требования к программному обеспечению**

Требования к части разработчиков:

- PyCharm Community Edition 2021.3 или IDE Microsoft Visual Studio Community 2019 Version 16.8.4 и выше.

Требования к клиентской части:

Любой из перечисленных ниже браузеров (указана минимальная версия):

-Internet Explorer 6,

-Mozilla 1.6 (Firefox 1.0),

-Opera 9,

-Microsoft Edge,

-Chrome.

**1.4.4. Требования к техническому обеспечению**

Клиентская часть:

-Расширение экрана 1920x 1080 (минимальное),

-Оперативная память 256 Мб (рекомендуется от 512 Мб).

**1.5. Требования к эргономике и технической эстетике**

Сайт должен быть оптимизирован для просмотра при разрешении 1920x 1080 (минимальное) без пустых (белых) полей для основных типов разрешения. Требования к дружественности интерфейса приложения включают удобство навигации по сайту (желательно, чтобы на программно генерируемой странице с выводом результата решения была ссылка для возврата на главную или предыдущую страницу); контрастность текста по отношению к фону и его читабельность (в том числе за счёт выбора гарнитуры, размера шрифта); использование высококачественной графики и уместной цветовой схемы; достаточную структурированность содержимого, представление в удобном для восприятия виде – блоков, списков, таблиц и т.д.

* 1. **Стадии и этапы разработки**

-Проработать сценарий сайта,

-Произвести выбор языков программирования,

-Проработать функциональность сайта,

-Написать программный код сайта,

-Проработать дизайн сайта,

-Провести тестирование функциональных возможностей сайта,

-Представить и защитить учебную практику.

1. **Алгоритм**

Алгоритм Крускала предназначен для решения задачи о построении минимального остовного дерева во взвешенном неориентированном графе. Последовательная сложность алгоритма O(mlnn).

Пусть задан связный неориентированный граф G=(V,E) с весами рёбер f(e). Предполагается, что веса всех рёбер различны (если это не так, то можно упорядочить рёбра сначала по весу, а потом по номеру).

Алгоритм Крускала основан на следующих двух свойствах задачи:

Минимальное ребро графа. Если e∗ – единственное ребро графа с минимальным весом, то оно принадлежит минимальному остовному дереву.

Схлопывание фрагментов. Пусть F – фрагмент минимального остовного дерева графа G, а граф G′ получен из G склеиванием вершин, принадлежащих F. Тогда объединение F и минимального остовного дерева графа G′ даёт минимальное остовное дерево исходного графа G.

В начале работы алгоритма каждая вершина графа G является отдельным фрагментом. На каждом шаге из рёбер, ещё не рассмотренных на предыдущих шагах, выбирается ребро с минимальным весом. Если оно соединяет два различных фрагмента, то оно добавляется в минимальное остовное дерево, а фрагменты склеиваются. В противном случае это ребро отбрасывается.

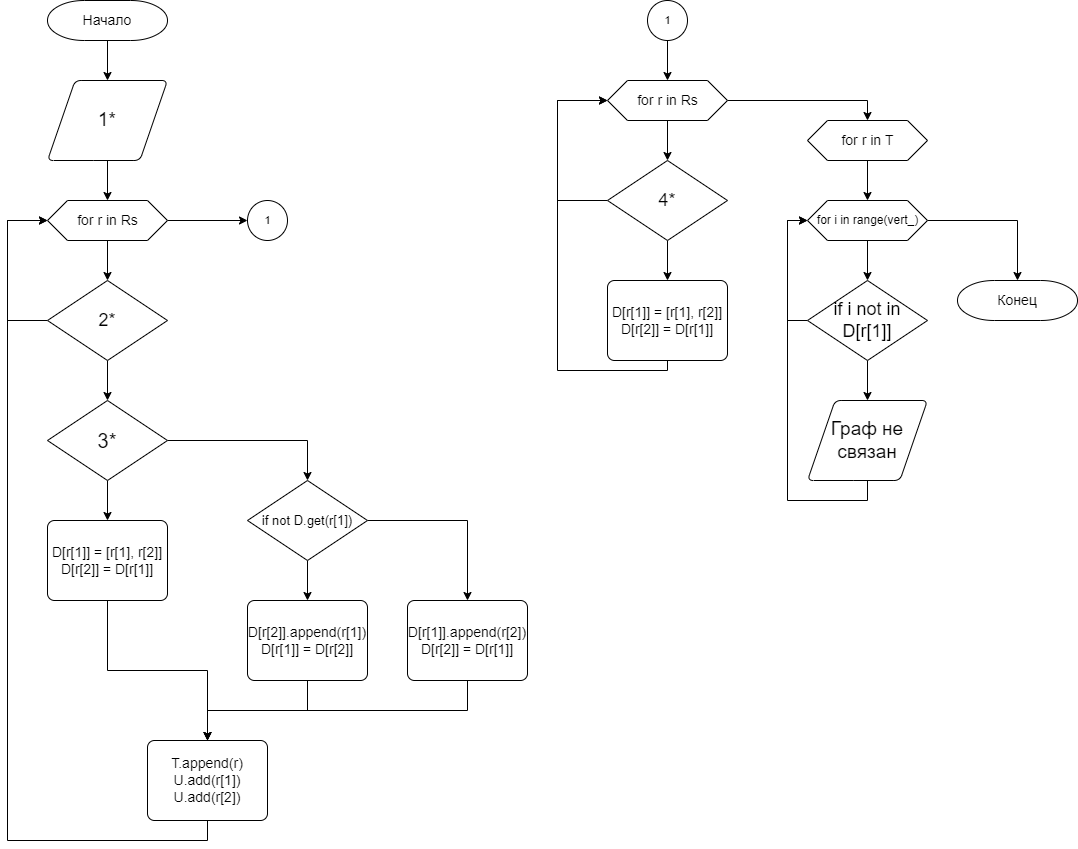


Рисунок 1. Блок-схема алгоритма Крускала.

1. **UML – диаграммы UseCase и компонентов приложения**

Диаграмма Use Case в UML — диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели программы, позволяющей описать систему на концептуальном уровне.

Диаграммы компонентов UML используются для моделирования больших систем в более мелкие подсистемы, которыми легко управлять.

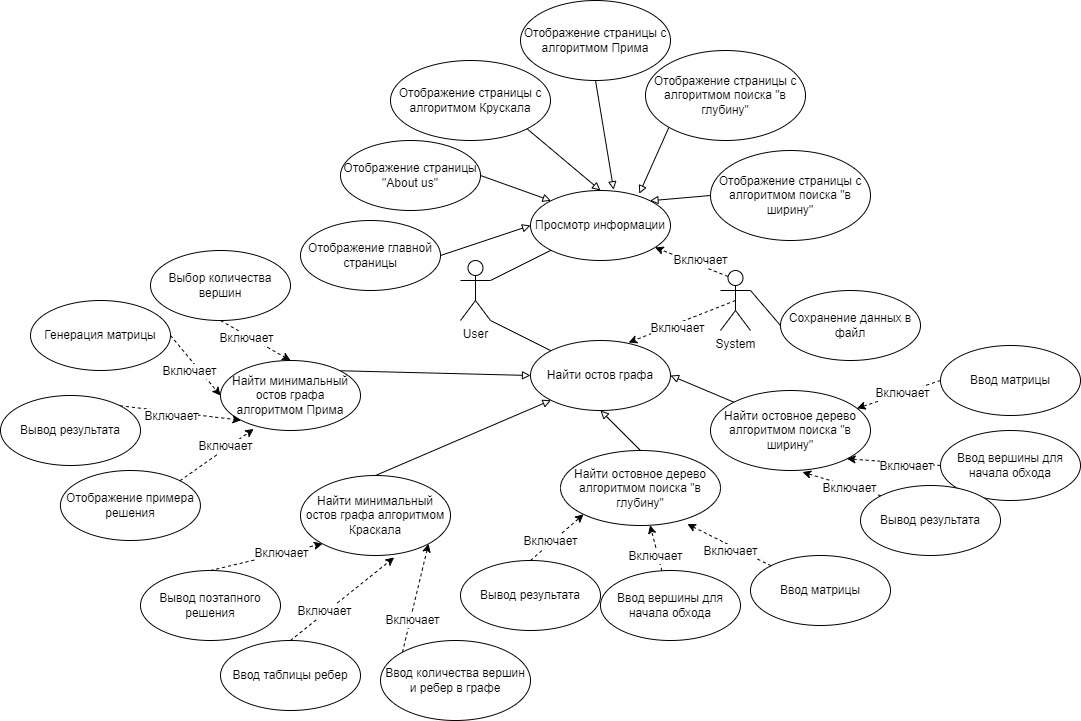
На рисунке 2 представлена диграмма UML Use Case, а на рисунке 3 диаграмма UML компонентов приложения.

Рисунок 2. UML - диаграмма UseCase.

### 

3. UML - диаграмма компонентов

### Разработка логики сайта

Разрабатываемое приложение создавалась в PyCharm Community Edition 2021.3 и выше и носит название BottleWebProject\_C921\_1\_RRC. В структуру проекта входят:  
- текстовый файл README с кратким описанием используемого ПО, решаемых задач, особенностей запуска, названиями файлов, содержащих входные и выходные данные проекта, который исключен из отслеживаемых Git файлов. В приложение А находятся скриншоты файла README;  
- веб-страницы: главная – index.tpl, о разработчиках проекта – about.tpl, веб-страниц с методами решения – breadth – first search; depth – first search; The Prim`s algorithm; Kruskal`s algorithm. В приложении Б на рисунках ПБ 1 изображена структура проекта из Обозревателя решений. На рисунках ПБ 2 изображены истории коммитов в Visual Studio. На рисунках ПБ 3 – ПБ 4 изображены файлы из репозитория на Github.

### Веб-страница по решению индивидуального задания

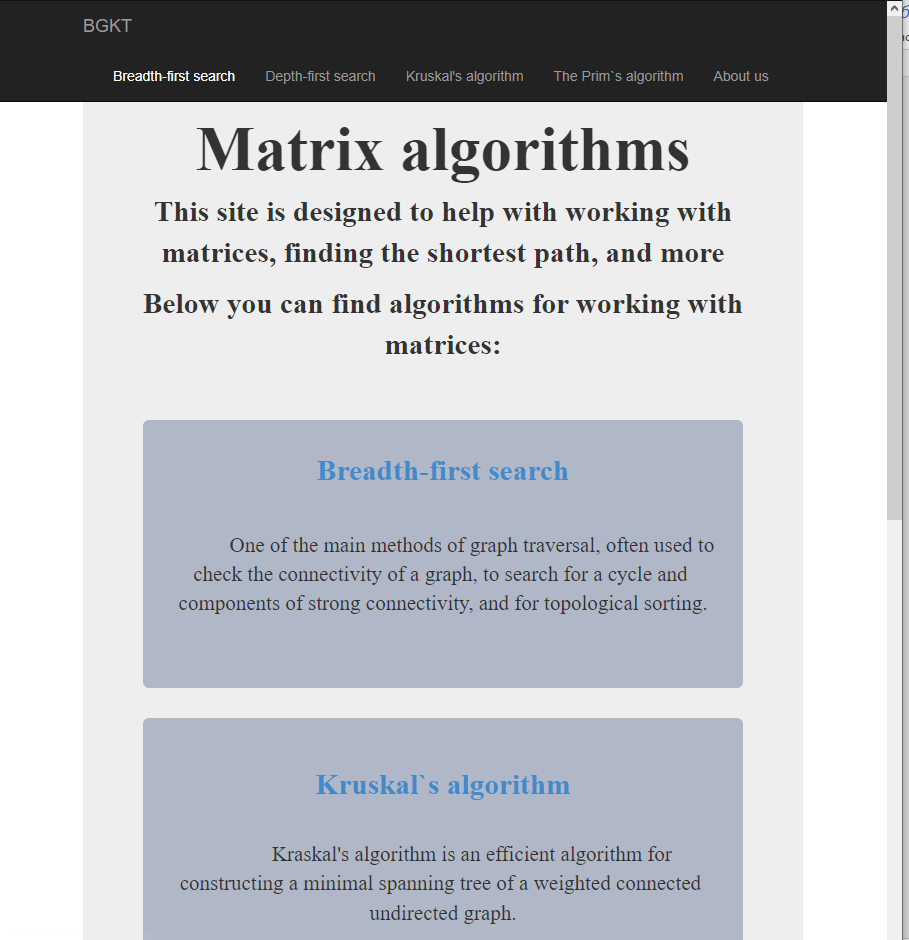


Рисунок 4. Главная страница сайта (1 часть).

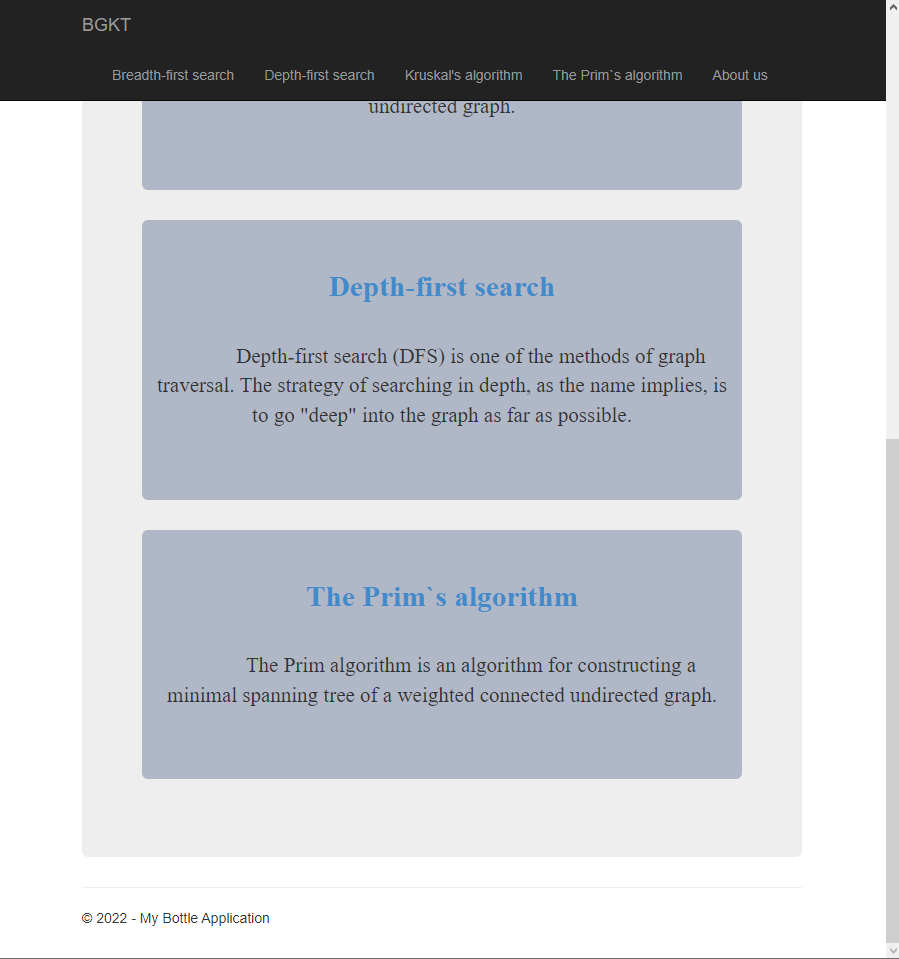


Рисунок 5. Главная страница сайта (2 часть).

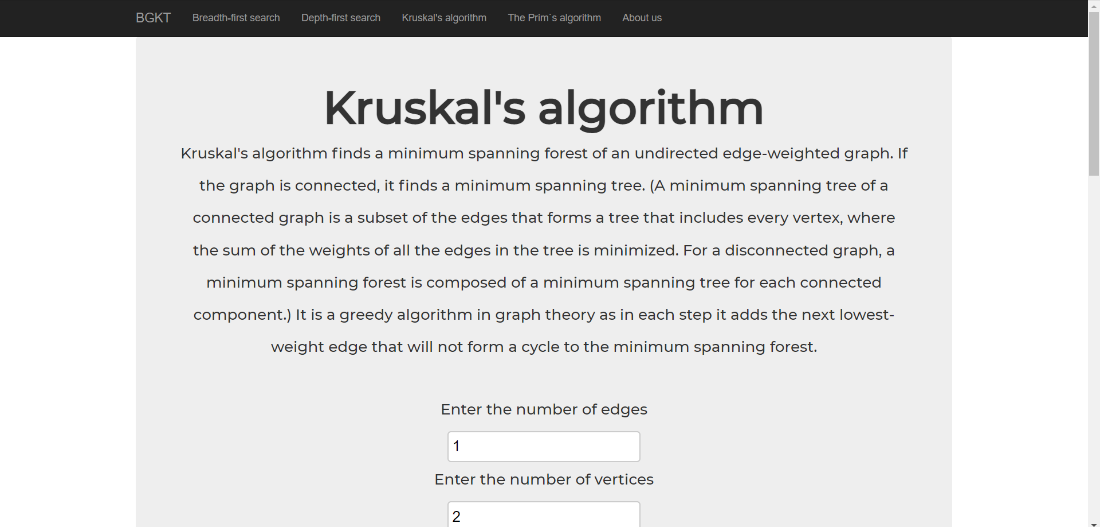


Рисунок 6. Страница с алгоритмом Крускала (1 часть).

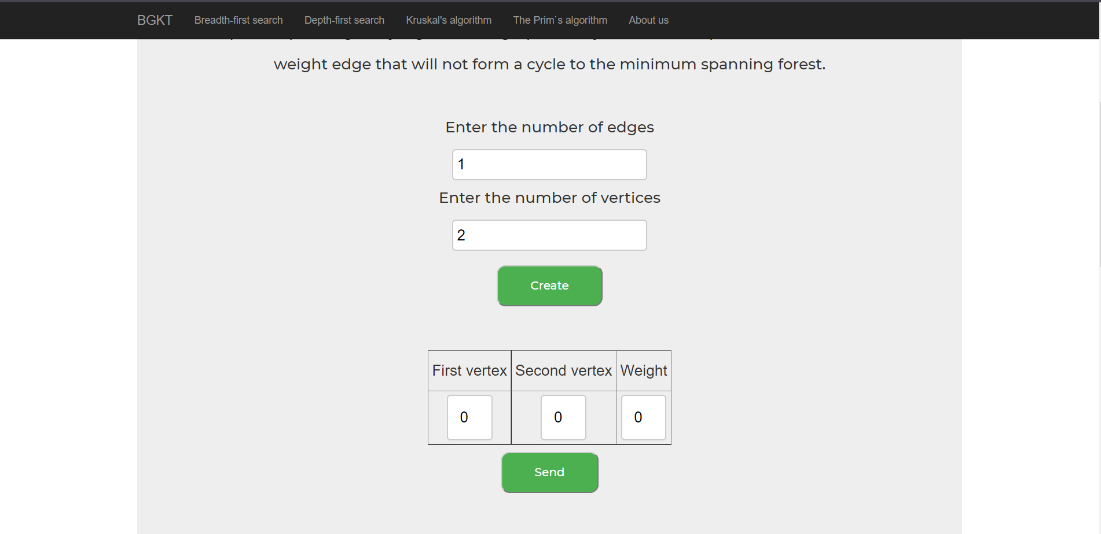


Рисунок 7. Страница с алгоритмом Крускала (2 часть)

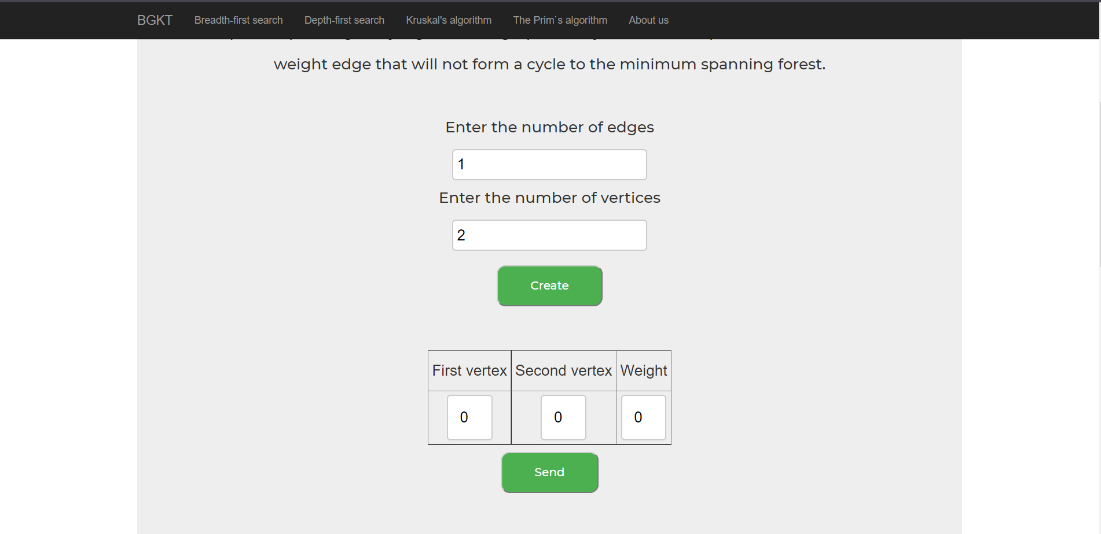


Рисунок 8. Страница с алгоритмом Крускала (3 часть).

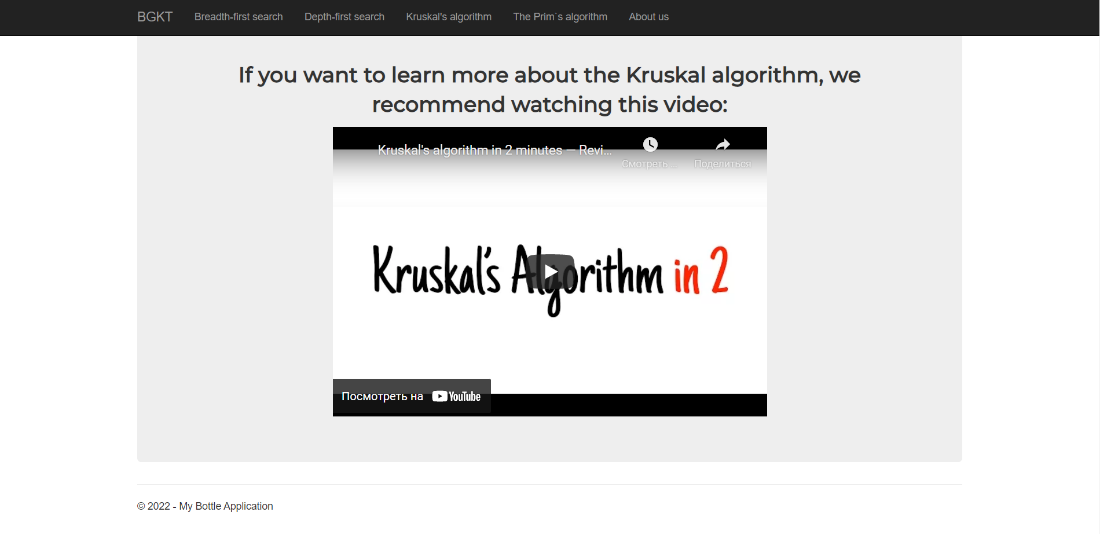


Рисунок 9. Страница с алгоритмом Крускала (4 часть).

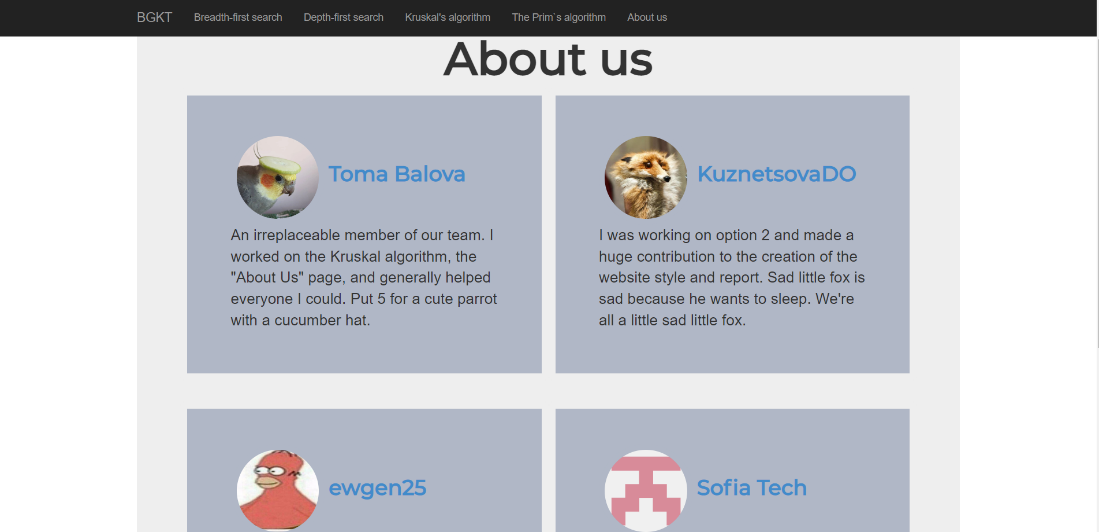


Рисунок 10. Страница с информацией о разработчиках.

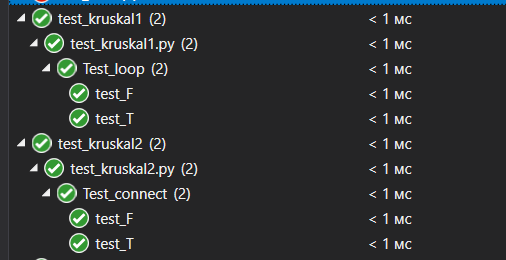


Рисунок 11. Результат юнит-тестов

### Дополнительное задание

В качестве дополнительного задания предлагается сохранить разработанную страницу с методом решения в формате html. Затем выполнить проверку валидности html-кода на сервисе https://validator.w3.org/#validate\_by\_upload.

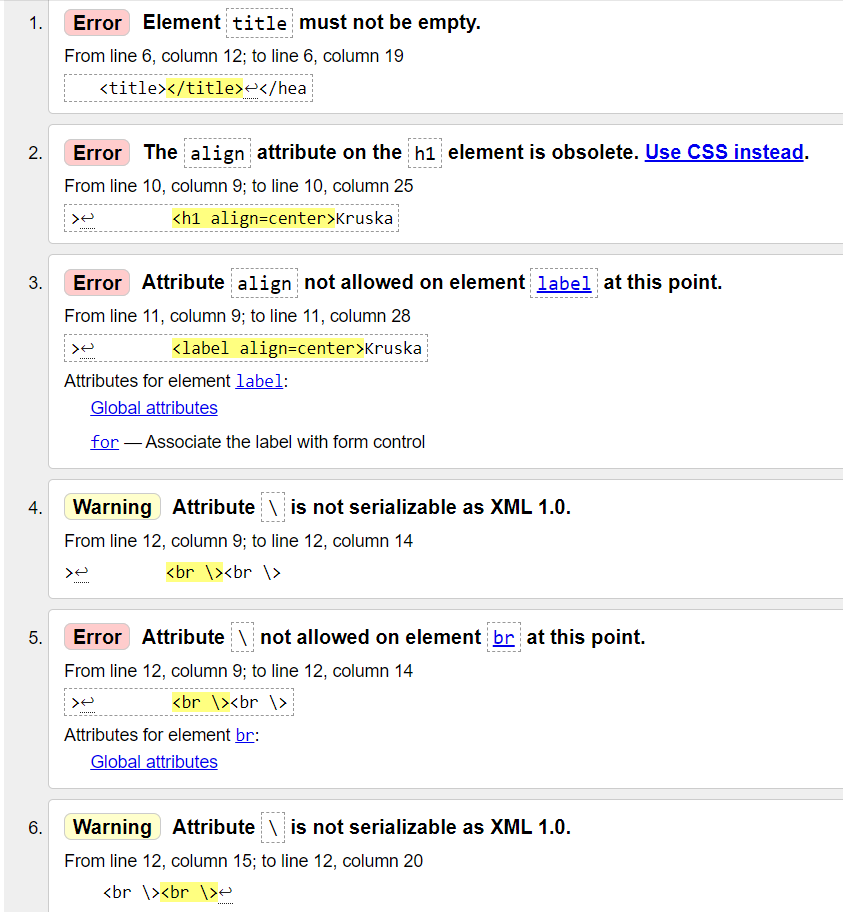


Рисунок 12. Результат валидации (1 часть).

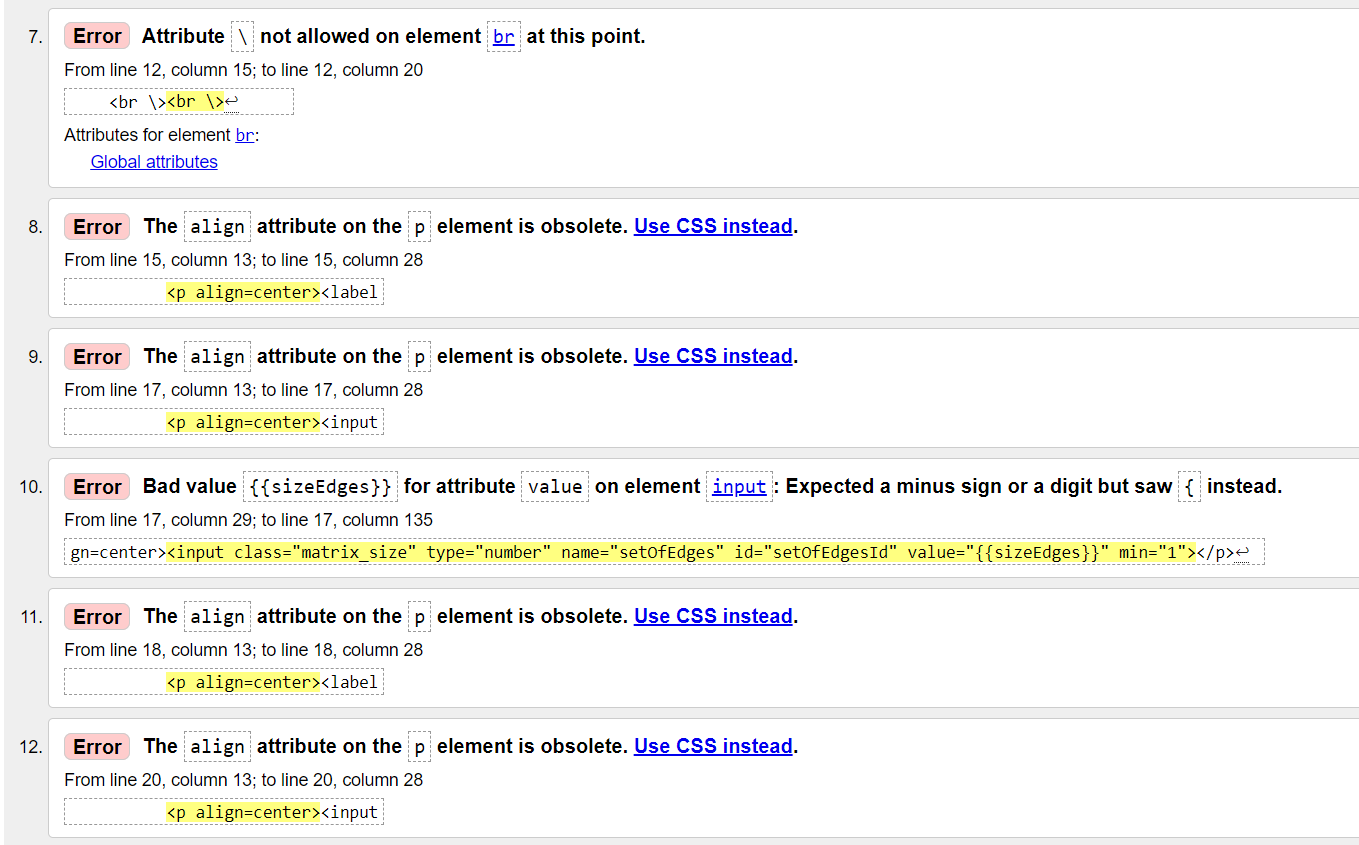


Рисунок 13. Результат валидации (2 часть).

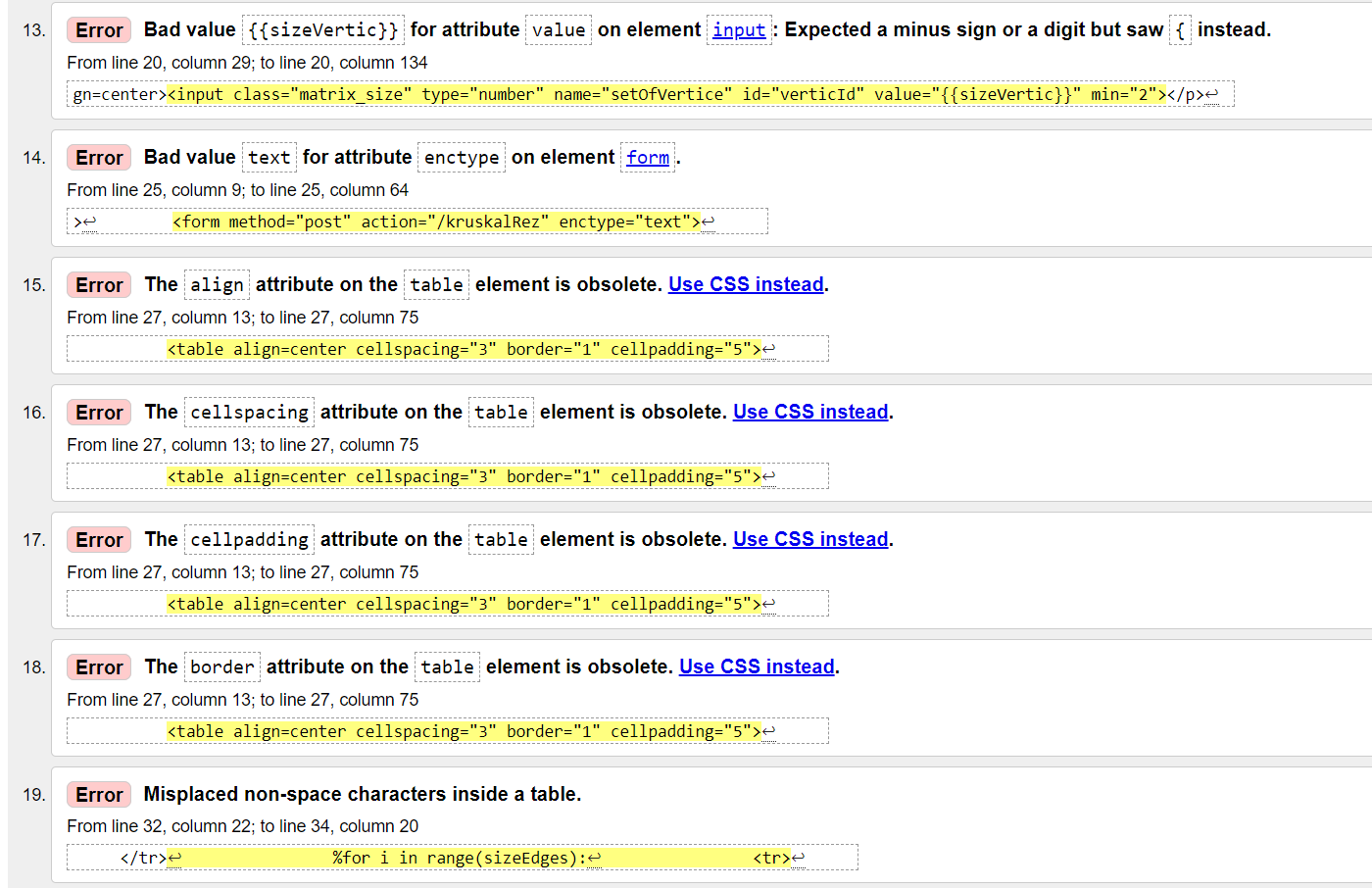


Рисунок 14. Результат валидации (3 часть).

### Заключение

В ходе учебной практики по МДК 02.02 Инструментальные средства разработки программного обеспечения и МДК 02.03 Математическое моделирование был создан веб-проект для решения математической задачи по поиску минимального остовного дерева графа алгоритмом Краскала.

А также получены практические навыки по работе с WebBottle project и оформления сайта при помощи HTML. И также изучена работа с template для отображения результатов. В том числе была проведена работа с unittest.

# 

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначение условные графические
2. ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению.
3. ГОСТ 19.402-78 ЕСПД. Описание программы
4. Справочник HTML – URL: https://htmlbase.ru/ (дата обращения 14-05-2022);
5. Работа с датой и временем в Python – URL: <https://python-scripts.com/datetime-time-python> (дата обращения 20-05-2022);
6. Template : элемент шаблона контента – URL: <https://developer.mozilla.org/ru/docs/Web/HTML/Element/template> (дата обращения 18-05-2022);

### ПРИЛОЖЕНИЕ А

### Структура файла README

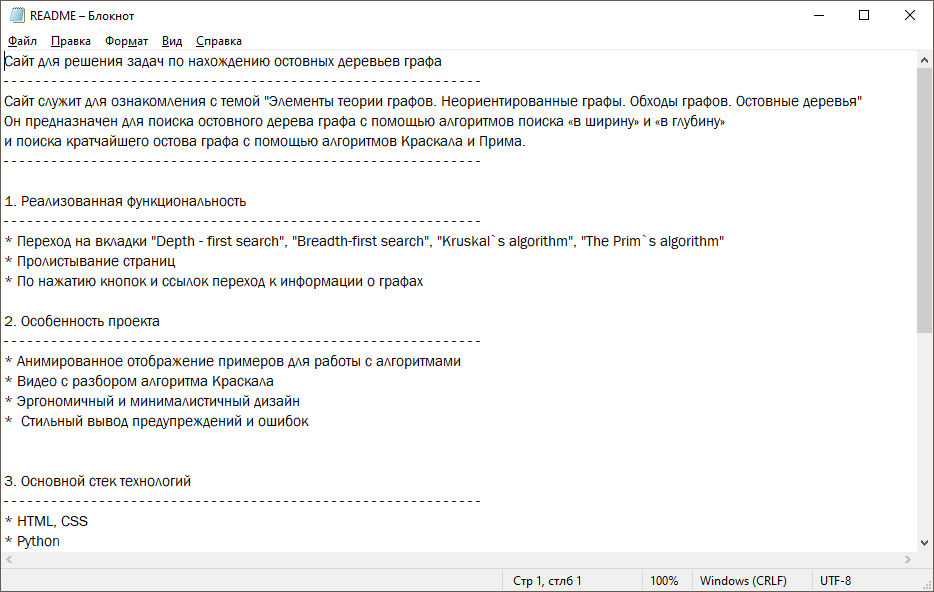


Рисунок ПА 1. Содержимое файла README (1 часть).

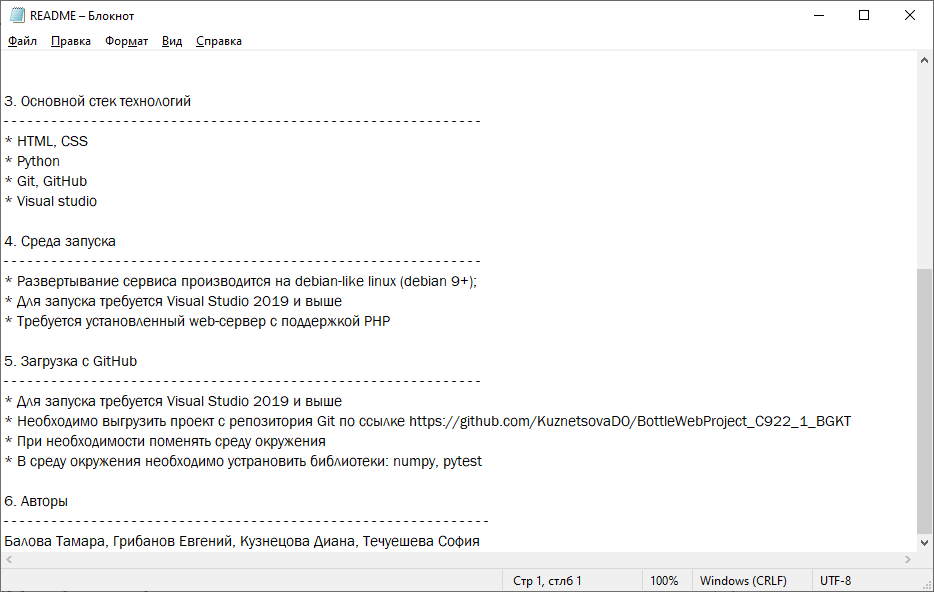


Рисунок ПА 2. Содержимое файла README (2 часть).

### ПРИЛОЖЕНИЕ Б

### Структура проекта, отслеживание Git-репозитория в Visual Studio, отслеживание Git-репозитория на GitHub

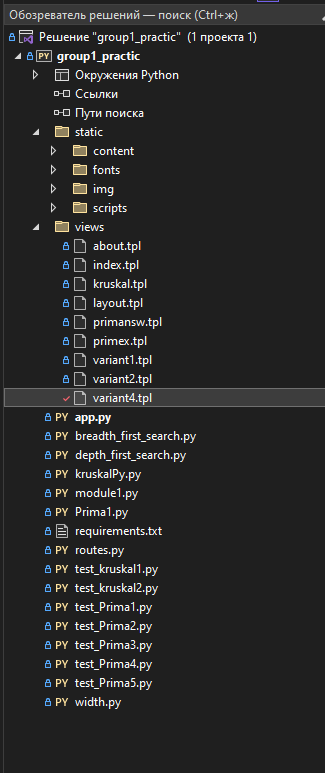


Рисунок ПБ 1. Структура проекта в обозревателе решений

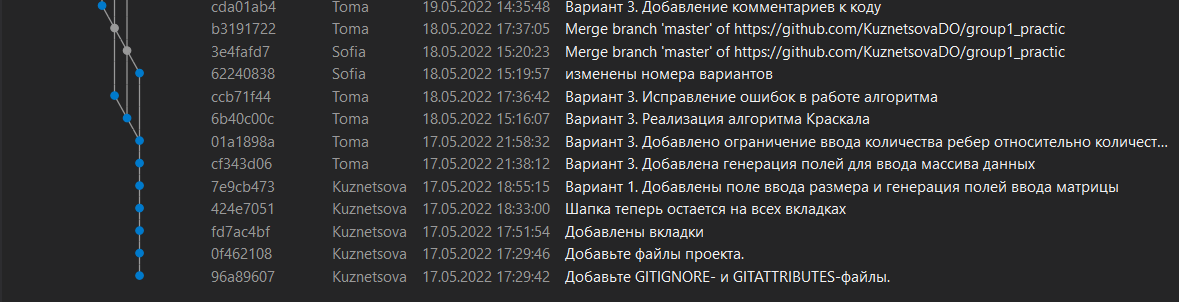


Рисунок ПБ 2. Фрагмент журнала коммитов (1 часть).

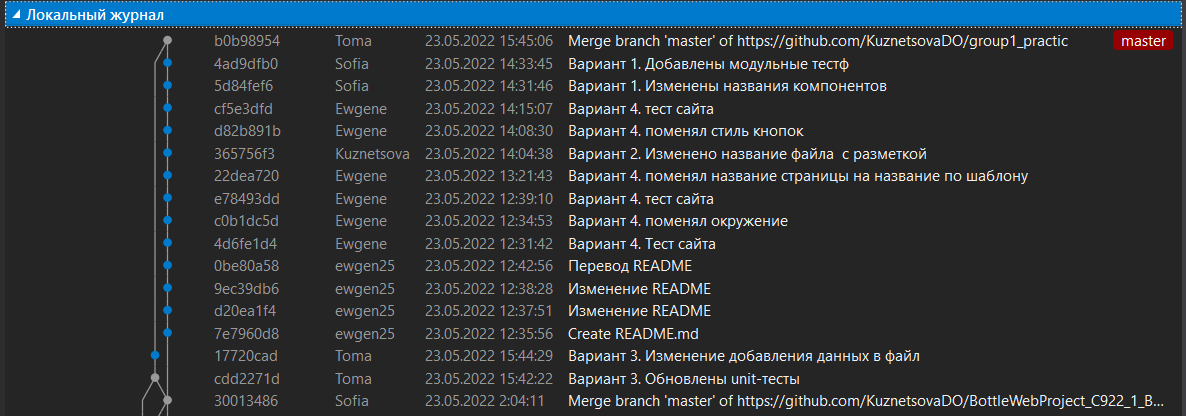


Рисунок ПБ 3. Фрагмент журнала коммитов (2 часть).

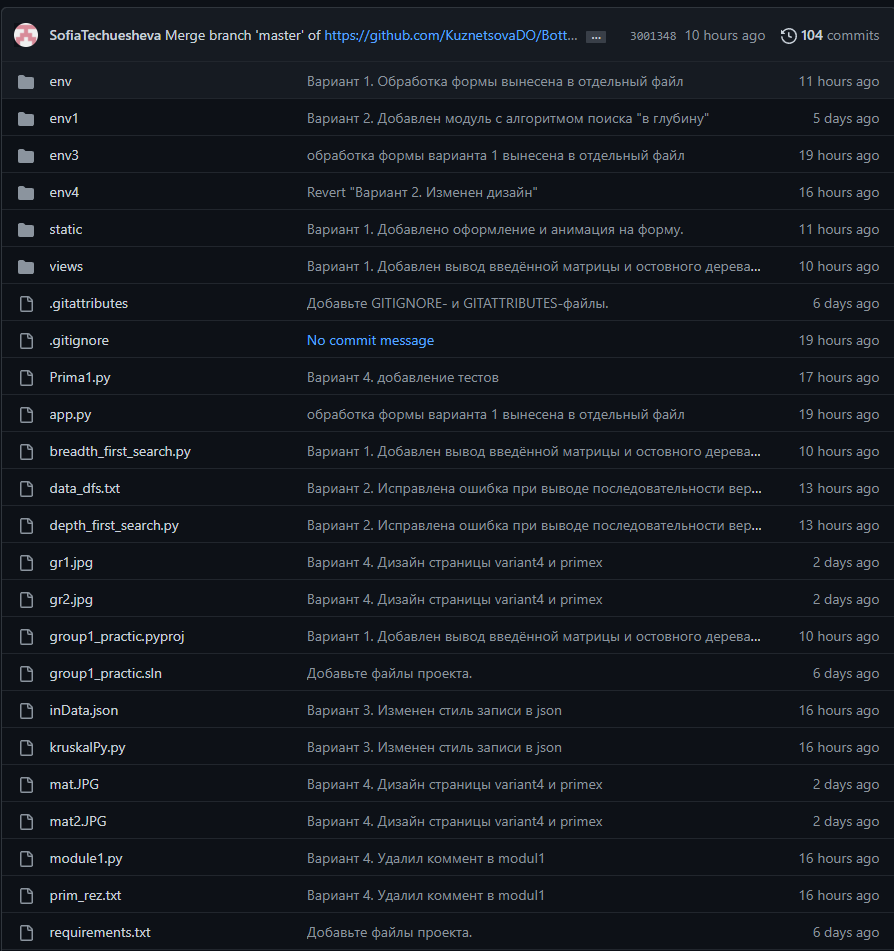


Рисунок ПБ 4. Файла в репозитории (1 часть)

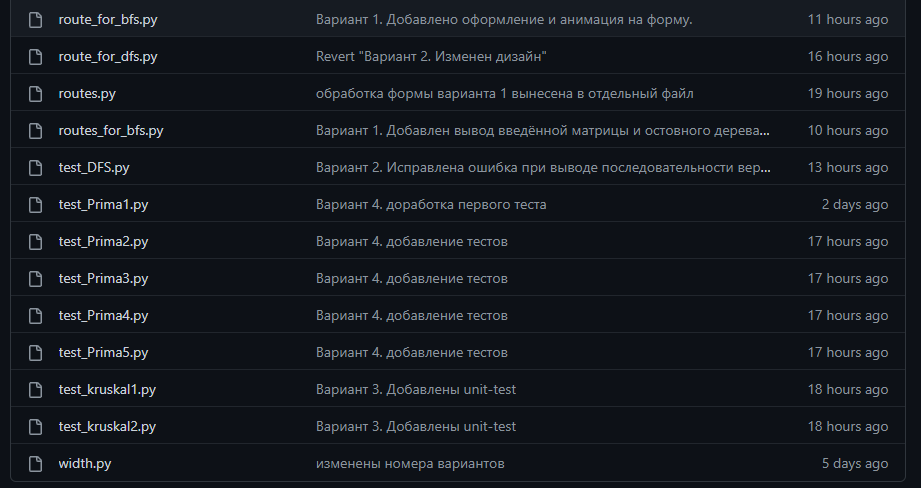


Рисунок ПБ 5. Файлы в репозитории (2 часть)

### ПРИЛОЖЕНИЕ В

# Листинги программного кода и приложения

Листинг главной страницы (index.tpl):

% rebase('layout.tpl', title='Home Page', year=year)

<head>

<link href="static/content/style.css" rel="stylesheet" type="text/css">

</head>

<div class="jumbotron" align=center>

<h1 align=center>Matrix algorithms</h1>

<p class="hhh">This site is designed to help with working with matrices, finding the shortest path, and more</p>

<p class="hhh">Below you can find algorithms for working with matrices:</p>

<br>

<div class="jum" align=center>

<div class="row">

<div class="row-md-4">

<a href="/variant1" style="text-decoration: none;">

<p class="hhh">Breadth-first search</p>

</a>

<p class="hh2">

One of the main methods of graph traversal, often used to check the connectivity of a graph, to search for a cycle and components of strong connectivity, and for topological sorting.

</p>

</div>

</div>

</div>

div class="jum" align=center>

<div class="row">

<div class="row-md-4">

<a href="/kruskal" style="text-decoration: none;">

<h2 class="hhh">Kruskal`s algorithm</h2>

</a>

<p class="hh2">

Kraskal's algorithm is an efficient algorithm for constructing a minimal spanning tree of a weighted connected undirected graph.

</p>

</div>

</div>

</div

<div class="jum" align=center>

<div class="row">

<div class="row-md-4">

<a href="/variant2" style="text-decoration: none;">

<h2 class="hhh">Depth-first search</h2>

</a>

<p class="hh2">

Depth-first search (DFS) is one of the methods of graph traversal. The strategy of searching in depth, as the name implies, is to go "deep" into the graph as far as possible.

</p>

</div>

</div>

</div>

<div class="jum" align=center>

<div class="row">

<div class="row-md-4">

<a href="/prim\_algorithm" style="text-decoration: none;">

<h2 class="hhh">The Prim`s algorithm</h2>

</a>

<p class="hh2">

The Prim algorithm is an algorithm for constructing a minimal spanning tree of a weighted connected undirected graph.

</p>

</div>

</div>

</div>

</div>

Листинг индивидуальной страница:

<!DOCTYPE html>

% rebase('layout.tpl', title=title, year=year, sizeEdges=sizeEdges, sizeVertic=sizeVertic)

<link href="/static/content/style.css" rel="stylesheet"

<div class="jumbotron">

<body>

<!-- заголовок и описание алгоритма -->

<h1 align = center>Kruskal's algorithm</h1>

<label align = center>Kruskal's algorithm finds a minimum spanning forest of an undirected edge-weighted graph. If the graph is connected, it finds a minimum spanning tree. (A minimum spanning tree of a connected graph is a subset of the edges that forms a tree that includes every vertex, where the sum of the weights of all the edges in the tree is minimized. For a disconnected graph, a minimum spanning forest is composed of a minimum spanning tree for each connected component.) It is a greedy algorithm in graph theory as in each step it adds the next lowest-weight edge that will not form a cycle to the minimum spanning forest.</label>

<br \><br \>

<!-- форма для ввода вершин и ребер -->

<form >

<p align = center><label>Enter the number of edges</label></p>

<!-- ввод количества ребер -->

<p align = center><input class="matrix\_size" type="number" name="setOfEdges" id="setOfEdgesId" value = "{{sizeEdges}}" min="1"></p>

<p align = center><label>Enter the number of vertices</label></p>

<!-- ввод количества вершин -->

<p align = center><input class="matrix\_size" type="number" name="setOfVertice" id="verticId" value = "{{sizeVertic}}" min="2"></p>

<!-- кнопка для отправки формы -->

<p><button style="margin: 10px 0px 5%" class="button button1 hh2" type="submit">Create</button></p>

</form>

<!-- форма для ввода значений (начальной вершины, конечной вершины и веса ребра между ними) -->

<form method="post" action="/kruskalRez" enctype="text" >

<!-- цикл для создания необходимого количества полей ввода -->

<table align = center cellspacing="3" border="1" cellpadding="5">

<tr>

<td>First vertex</td>

<td>Second vertex</td>

<td>Weight</td>

</tr>

%for i in range(sizeEdges):

<tr>

<td align = center><input class = "cell" type="number" name="startingVertex" value = "0" min = "1" max = "{{sizeVertic}}" required></td>

<td align = center><input class = "cell" type="number" name="finalVertex" value = "0" min = "1" max = "{{sizeVertic}}" required></td>

<td align = center><input class = "cell" type="number" name="ribWeight" value = "0" min = "1" required>

<br name="row0"></td>

</tr>

%end

</table>

<!-- скрытый элемент для передачи в kruskalPy количества вершин -->

<input hidden type="number" name="el1" size="3" value = "{{sizeVertic}}" min = "1" required>

<!-- кнопка для отправки формы -->

<p><button style="margin: 10px 0px 5%" class="button button1 hh2" type="submit">Send</button></p>

</form>

<!-- форма с инструкцией к калькулятору -->

<form align = center>

<p class="hhh">How does it work?</p>

<p>1. Specify the number of edges and vertices of your graph and click on the "create" button. Note that a graph with a certain number of vertices has a maximum number of edges. If you drive too many edges, the calculator will automatically replace it with the maximum possible.</p>

<p>2. Fill in the table, avoiding repetition of vertices, loops, and make sure that the connections between all vertices are specified. Otherwise, the calculator will indicate an error without giving a solution.</p>

<p>3. Check the entered data, making sure that they are filled in correctly, and click on the "send" button.</p>

<p>4. Study the received answer. To return to the form to fill out, click the button at the very bottom of the form with answers or select the calculator you are interested in in the header of the site.</p>

</form>

<br \><br \>

<!-- форма с информацией о методе -->

<form align = center>

<p class="hhh">If you want to learn more about the Kruskal algorithm, we recommend watching this video:</p>

<!-- видео с объяснением алгоритма Краскала -->

<iframe src="https://www.youtube.com/embed/71UQH7Pr9kU?rel=0&amp;fs=1&amp;wmode=transparent" width="600" height="400" allow="autoplay; fullscreen; encrypted-media" allowfullscreen="true" frameborder="0" scrolling="no" title="JoomlaWorks AllVideos Player"></iframe>

</form>

</body>

</div>

</html>

Файл для решения задачи

from bottle import route, view, request, template

from array import \*

from datetime import datetime

import json

def isLoop(start\_, final\_):

size\_ = len(final\_)

for i in range(size\_):

if (int(start\_[i]) == int(final\_[i])):

return True

return False

def isConnectV(start\_, final\_, vert\_):

for i in range(vert\_):

i += 1

if not(str(i) in start\_) and not(str(i) in final\_):

return True

return False

@route('/kruskalRez', method='post')

def kruskal\_yu():

# получение данных с формы

weight\_ = request.forms.getall('ribWeight')

start\_ = request.forms.getall('startingVertex')

final\_ = request.forms.getall('finalVertex')

size\_ = len(weight\_)

vert\_ = request.forms.get('el1', type=int)

errors = []

er = 0

# проверка на наличие несвязанных вершин в графе

if isConnectV(start\_, final\_, vert\_):

errors.insert(er, "ne svyazan")

er += 1

# проверка на наличие петель в графе

if isLoop(start\_, final\_):

errors.insert(er, "petlya")

er += 1

# проверка на наличие повторяющихся ребер

for i in range(size\_):

for j in range(size\_):

duo11 = start\_[i] + final\_[i]

duo12 = final\_[i] + start\_[i]

duo21 = start\_[j] + final\_[j]

duo22 = final\_[j] + start\_[j]

if i != j:

if duo11 == duo21 or duo11 == duo22 or duo12 == duo21 or duo12 == duo22:

errors.insert(er, "povtor")

er += 1

R = [[]]

# заполнение массива

for i in range(size\_):

R.insert(i, [int(weight\_[i]), int(start\_[i]), int(final\_[i])])

del(R[size\_])

Rs = sorted(R, key=lambda x: x[0], reverse=False) # сортировка массива по возрастанию веса

U = set() # список соединенных вершин

D = {} # словарь списка изолированных групп вершин

T = [] # список ребер остова

proc = "Next, we will create a list of isolated vertex groups to avoid the appearance of cycles. "

process = []

p = 0

for r in Rs:

if r[1] not in U or r[2] not in U: # проверка для исключения циклов в остове

if r[1] not in U and r[2] not in U: # если обе вершины не соединены, то

D[r[1]] = [r[1], r[2]] # формируем в словаре ключ с номерами вершин

D[r[2]] = D[r[1]] # и связываем их с одним и тем же списком вершин

proc = "Vertices " + str(r[1]) + " and " + str(r[2]) + " meet for the first time, forming a group of isolated vertices without creating a cycle. Therefore, they are definitely part of the spanning tree. "

process.insert(p, proc)

p += 1

else: # иначе

if not D.get(r[1]): # если в словаре нет первой вершины, то

D[r[2]].append(r[1]) # добавляем в список первую вершину

D[r[1]] = D[r[2]] # и добавляем ключ с номером первой вершины

proc = "The vertex " + str(r[2]) + " has already been encountered before, but the vertex " + str(r[1]) + " occurs for the first time and does not form a cycle. Therefore, the edge between these two vertices is part of the spanning tree. "

process.insert(p, proc)

p += 1

else:

D[r[1]].append(r[2]) # иначе, все то же самое делаем со второй вершиной

D[r[2]] = D[r[1]]

proc = "The vertex " + str(r[1]) + " has already been encountered before, but the vertex " + str(r[2]) + " occurs for the first time and does not form a cycle. Therefore, the edge between these two vertices is part of the spanning tree. "

process.insert(p, proc)

p += 1

T.append(r) # добавляем ребро в остов

U.add(r[1]) # добавляем вершины в множество U

U.add(r[2])

else:

proc = "The vertices " + str(r[1]) + " and " + str(r[2]) + " have already met before, so the edge connecting these vertices is not of interest to us at the moment. "

process.insert(p, proc)

p += 1

proc = "Now that we have groups of isolated vertices, we need to connect them together. To do this, once again consider the sorted table with the edges of the graph and find the edge with the lowest weight connecting the isolated groups. "

process.insert(p, proc)

p += 1

for r in Rs: # проходим по ребрам второй раз и объединяем разрозненные группы вершин

if r[2] not in D[r[1]]: # если вершины принадлежат разным группам, то объединяем

T.append(r) # добавляем ребро в остов

gr1 = D[r[1]] # сохраняем значение до объединения

D[r[1]] += D[r[2]] # объединем списки двух групп вершин

D[r[2]] += gr1

proc = "Consider vertices " + str(r[1]) + " and " + str(r[2]) + " connected by an edge. These vertices belong to different isolated groups, therefore this edge will be part of the spanning tree. "

process.insert(p, proc)

p += 1

# проверка на связанность изолированных групп графа

for r in T:

for i in range(vert\_):

i += 1

if i not in D[r[1]]:

errors.insert(er, "ne svyazan")

er += 1

# запись данных в файл

if er == 0:

jsonIn = "Time " + str(datetime.now()) + "; Input " + str(R) + "; Output " + str(T)

with open('inData.json', 'a') as file:

json.dump(jsonIn, file)

file.write('\n')

else:

jsonIn = "Time " + str(datetime.now()) + "; Input " + str(R) + "; Output errors"

with open('inData.json', 'a') as file:

json.dump(jsonIn, file)

file.write('\n')

return template('kruskalRez',

title='Kruskal',

year=datetime.now().year,

T=T,

R=R,

U=U,

Rs=Rs,

D=D,

errors = errors,

process=process)

Файл для вывода результата

<!DOCTYPE html>

% rebase('layout.tpl', title=title, year=year, T=T, R=R, U=U, Rs=Rs, D=D, errors=errors, process=process)

<link href="/static/content/style.css" rel="stylesheet">

<div class="jumbotron">

<body>

<!-- форма с выводом результата по методу Крускала -->

<form align = center>

<!-- проверка на наличие ошибок в введенных данных -->

% if len(errors) != 0:

<p class="hhh">Error</p>

<!-- вывод информации об ошибкая -->

<p>When entering data, you made the following mistakes:</p>

<!-- ошибка не связанного графа -->

% if "ne svyazan" in errors:

<p>The graph is not connected. Make sure that the data is correct. There is no minimum backbone in an unrelated graph.</p>

% end

<!-- ошибка наличия петель -->

% if "petlya" in errors:

<p>There is a loop in the graph. Make sure that all the data is entered correctly. Loops cannot be part of a minimal spanning tree, so enter the data ignoring all loops in the graph.</p>

% end

<!-- ошибка наличия повторов при вводе ребер -->

% if "povtor" in errors:

<p>In the introduced graph there is a repetition of the same edge. Check the correctness of the entered data and try again.</p>

% end

% end

<!-- вывод результата при правильном вводе данных -->

% if len(errors) == 0:

<p class="hhh">Solution by Kruslak 's algorithm</p>

<p> <label>Source data (make sure all data has been entered correctly): </label></p>

<!-- вывод данных, которые ввел пользователь -->

<p><table align = center cellspacing="3" border="1" cellpadding="5">

<tr>

<td>weight</td>

<td>sart</td>

<td>final</td>

</tr>

% for row in R:

<tr>

% for value in row:

<td>{{ value }}</td>

% end

</tr>

% end

</table></p>

<!-- описание первого этапа алгоритма -->

<p> <label>The first step to find the minimum spanning tree is to sort the entered data in ascending order of the edge weight.</label></p>

<!-- вывод результата сортировки графа -->

<p><table align = center cellspacing="3" border="1" cellpadding="5">

<tr>

<td>weight</td>

<td>sart</td>

<td>final</td>

</tr>

% for row in Rs:

<tr>

% for value in row:

<td>{{ value }}</td>

% end

</tr>

% end

</table></p>

<!-- вывод поэтапного пояснения действия алгоритма -->

% i = 0

% for pr in process:

<p><label>{{pr}}</label></p>

% if "part of the spanning tree" in pr:

% i += 1

<table align = center cellspacing="3" border="1" cellpadding="5">

<tr>

<td>weight</td>

<td>sart</td>

<td>final</td>

</tr>

% for j in range(i):

<tr>

% for value in T[j]:

<td>{{ value }}</td>

% end

</tr>

% end

</table></p>

%end

% end

<br \><br \>

<!-- вывод итогового ответа -->

<p class="hhh">The result of the search for the minimum spanning tree by the Kruskal method: </p>

<p><table align = center cellspacing="3" border="1" cellpadding="5">

<tr>

<td>weight</td>

<td>sart</td>

<td>final</td>

</tr>

% for row in T:

<tr>

% for value in row:

<td>{{ value }}</td>

% end

</tr>

% end

</table></p>

% end

</form>

<br \>

<form action="/kruskal">

<p><button style="margin: 10px 0px 5%" class="button button1 hhh" type="submit">Back</button></p>

</body>

</div>

</html>

### ПРИЛОЖЕНИЕ Г

# Листинги юнит-тестов

Файл test\_kruskal1.py для проверки:

import unittest

import kruskalPy

class Test\_loop(unittest.TestCase):

# тест с данными без петель

def test\_F(self):

first = [["1","2","3","8","1"],["2","1","2","4","1","5"], ["2", "3", "4"], ["1", "2", "4", "5", "2"], ["4", "3", "2", "3"]]

second = [["4","4","1","3","7"],["3","4","4","6","5","2"], ["1", "1", "2"], ["5", "3", "1", "3", "5"], ["1", "1", "1", "4"]]

for i in range(len(first)):

self.assertFalse(kruskalPy.isLoop(first[i], second[i]))

# тест с данными с петлей

def test\_T(self):

first = [["1","2","3","8","1"],["2","1","2","4","1","5"], ["1", "3", "3"], ["1", "5", "2", "5", "3"], ["1", "2", "3", "4"]]

second = [["1","4","1","3","7"],["3","4","4","4","5","2"], ["2", "4", "3"], ["2", "2", "2", "4", "3"], ["1", "2", "3", "4"]]

for i in range(len(first)):

self.assertTrue(kruskalPy.isLoop(first[i], second[i]))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Файл test\_kruskal2.py для проверки

import unittest

import kruskalPy

# тест на наличие несвязанных вершин

class Test\_connect(unittest.TestCase):

# тест с данными без несвязанных вершин

def test\_F(self):

vert = [5, 6, 4, 5, 4]

first = [["1","2","3","5","1"],["2","1","2","4","1","5"], ["2", "3", "4"], ["1", "2", "4", "5", "2"], ["4", "3", "2", "3"]]

second = [["4","4","1","3","5"],["3","4","4","6","5","2"], ["1", "2", "3"], ["2", "3", "5", "1", "4"], ["1", "1", "3", "4"]]

for i in range(len(first)):

self.assertFalse(kruskalPy.isConnectV(first[i], second[i], vert[i]))

# тест с данными с наличием несвязанных вершин

def test\_T(self):

vert = [5, 6, 4, 6, 4]

first = [["1","2","3","4","1"],["2","1","2","4","1","5"], ["2", "3", "4"], ["1", "2", "4", "5", "2"], ["4", "3", "2"]]

second = [["4","4","1","3","2"],["3","4","4","3","5","2"], ["3", "4", "2"], ["5", "1", "3", "2", "3"], ["3", "2", "4"]]

for i in range(len(first)):

self.assertTrue(kruskalPy.isConnectV(first[i], second[i], vert[i]))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()