

Федеральное государственное автономное образовательное учреждение  
высшего образования «Пермский национальный исследовательский  
политехнический университет»

ОТЧЁТ

ПО ТВОРЧЕСКОЙ РАБОТЕ ЧАСТЬ 2

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Задача коммивояжера»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-36

Акбашева Софья Руслановна

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

## РЕФЕРАТ

КОММИВОЯЖЕР, МЕТОД ВЕТВЕЙ И ГРАНИЦ, ГРАФ, КЛАСС, SFML, РЕДУКЦИЯ

Объектом исследования является решение задачи коммивояжера.

Предмет исследования – метод ветвей и границ

Целью работы является получение навыков работы с алгоритмом коммивояжера в графах.

В результате проведенного исследования было разработано оконное приложение, которое позволяет редактировать граф и находить оптимальный путь через все вершины, используя метод ветвей и границ.

## ОГЛАВЛЕНИЕ

РЕФЕРАТ .....	2
ВВЕДЕНИЕ .....	4
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ .....	5
2 ОПИСАНИЕ ИСПОЛЬЗУЕМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ....	8
3 РАЗРАБОТКА ПРОГРАММЫ .....	10
Анализ задачи.....	10
4 ТЕСТИРОВАНИЕ ПРОГРАММЫ.....	15
ЗАКЛЮЧЕНИЕ .....	17
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	18
ПРИЛОЖЕНИЯ.....	19
ПРИЛОЖЕНИЕ А.....	19
Рисунок А.1 – uml диаграмма класса TextBox .....	19
Рисунок А.2 – uml диаграмма класса Button .....	19
Рисунок А.2 – uml диаграмма класса Graph .....	20
ПРИЛОЖЕНИЕ Б.....	21
Листинг Б.1 – Файл Graphs.h.....	21
Листинг Б.2 – Файл other functions.h.....	36
Листинг Б.3 – Файл textbox.hpp.....	43
Листинг Б.4 – Файл textbox.cpp.....	44
Листинг Б.5 – Файл sfml_button.hpp.....	47
Листинг Б.6 – Файл sfml_button.cpp.....	48
Листинг Б.7 – Файл RectButton.cpp.....	49
Листинг Б.8 – Файл main.cpp.....	51

## ВВЕДЕНИЕ

Целью данной лабораторной работы является изучение алгоритма решения задачи коммивояжера и использование этого алгоритма в графах

Постановка задачи:

Имеется  $N$  городов. Выезжая из исходного города  $A_1$ , коммивояжер должен побывать во всех городах по одному разу и вернуться в город  $A_1$ . Задача заключается в определении последовательности объезда городов, при которой коммивояжеру требуется минимизировать некоторый критерий эффективности: стоимость проезда, время пути, суммарное расстояние.

Для расчета затрат существует матрица условий, содержащая затраты на переход из каждого города в каждый, при этом считается, что можно перейти из любого города в любой, кроме того же самого. Целью решения является нахождения маршрута, удовлетворяющего всем условиям и при этом имеющего минимальную сумму затрат.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) Анализ предметной области.
- 2) Описание используемого программного обеспечения.
- 3) Разработка программы.
- 4) Тестирование программы на корректность работы.

## 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Задача коммивояжера – далее ЗК – (коммивояжёр – бродячий торговец) заключается в отыскании самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и т. п.) и соответствующие матрицы расстояний, стоимости и т. п. Как правило указывается, что маршрут должен проходить через каждый город только один раз, в таком случае выбор осуществляется среди гамильтоновых циклов. Гамильтоновым циклом в графе называют простой цикл, содержащий все вершины графа ровно по одному разу. Существует масса разновидностей обобщённой постановки задачи, в частности геометрическая задача коммивояжёра (когда матрица расстояний отражает расстояния между точками на плоскости), треугольная задача коммивояжёра (когда на матрице стоимостей выполняется неравенство треугольника), симметричная и асимметричная задачи коммивояжёра. Простейшие методы решения задачи коммивояжёра: полный лексический перебор, жадные алгоритмы (метод ближайшего соседа, метод включения ближайшего города, метод самого дешёвого включения), метод минимального основного дерева. На практике применяются различные модификации более эффективных методов: метод ветвей и границ и метод генетических алгоритмов, а так же алгоритм муравьиной колонии [1].

В данной лабораторной работе рассматривается метод ветвей и границ. Для решения задачи коммивояжера методом ветвей и границ необходимо выполнить следующий алгоритм (последовательность действий):

1. Построение матрицы с исходными данными — в таблицу заносятся расстояния ( $C_{ij}$ ) между городами (в ячейки типа А-А, В-В и т. д. ставится символ М — условно бесконечно большое число); при этом строкам соответствуют города отбытия, а столбцам города прибытия;

2. Нахождение минимумов по строкам — в каждой строке определяется минимальное число.

3. Редукция строк — из значений ячеек каждой строки вычитаем соответствующий минимум ( $C_{ij} = C_{ij} - d_i$ ).

4. Нахождение минимумов по столбцам — в каждом столбце определяется минимальное число.

5. Редукция столбцов — из значений ячеек каждого столбца вычитаем соответствующий минимум ( $C_{ij} = C_{ij} - d_j$ ).

6. Нахождение корневой нижней границы — вычисляем нижнюю границу (минимально возможную на текущем этапе длину маршрута) в стартовой (корневой) точке решения, как сумму найденных ранее минимумов ( $H_0 = \sum d_i + \sum d_j$ ) и начинаем построение графа решения с внесения в него корневой вершины.

7. Вычисление оценок нулевых клеток — считаем оценки ( $r_{ij}$ ) для каждой ячейки с нулями, как сумму минимумов по строке и столбцу, в которых располагается нулевая клетка, не учитывая при этом саму нулевую клетку.

8. Выбор нулевой клетки с максимальной оценкой — ищем среди нулевых клеток обладающую наибольшей оценкой, получаем пару ветвей (вариантов) решения задачи: с включением в маршрут отрезка пути относящегося к выбранной ячейке и без включения.

9. Редукция матрицы — вычеркиваем относящиеся к выбранной клетке строку и столбец, а также заменяем значение ячейки соответствующей обратному пути на  $M$ .

10. Вычисление нижней границы первой ветви (включающей отрезок пути) — вновь находим минимумы по строкам, проводим редукцию строк, находим минимумы по столбцам, проводим редукцию столбцов, после чего вычисляем локальную нижнюю границу, как сумму предыдущей локальной нижней границы и минимумов ( $H_k = H_{k-1} + \sum d_i + \sum d_j$ ), и добавляем вершину в граф.

11. Вычисление нижней границы второй ветви (не включающей отрезок пути) — считаем локальную нижнюю границу, как сумму

предыдущей локальной нижней границы и оценки выбранной ранее нулевой клетки ( $H_k^* = H_{k-1} + p_{ij}$ ), и добавляем вершину в граф.

12. Выбор ветви с минимальным значением нижней границы — среди еще не ветвившихся вершин выбираем обладающую минимальным значением локальной нижней границы.

13. Если полный маршрут еще не найден, продолжаем решение, если найден — переходим к пункту 10 — если маршрут еще не найден, то ход дальнейшего решения зависит от выбранной ветви: (а) первая ветвь — переходим к пункту 7, (б) вторая ветвь — в клетку с максимальной оценкой ставим  $M$  и переходим к пункту 2, (в) другая ветвь — возвращаемся к соответствующим ей этапу решения и таблице данных [2].

14. Построение полного маршрута.

Применение задачи коммивояжера на практике довольно обширно: ее можно использовать для поиска кратчайшего пути при гастролях эстрадной группы или обеспечения наименьшего времени выполнения производственного цикла.

Таким образом, данная лабораторная работа имеет практическую значимость и может быть использована для получения навыков работы с графами.

## 2 ОПИСАНИЕ ИСПОЛЬЗУЕМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В работе был использован язык программирования C++. C++ — это язык программирования, который был разработан в 80-х годах прошлого века как расширение языка C. Этот язык отличается от Си тем, что имеет больший набор возможностей, включая объектно-ориентированное программирование и шаблоны.

C++ используется для создания программного обеспечения разного рода: от игр до операционных систем. Этот язык также широко применяется в интенсивной обработке данных и научных расчетах.

C++ предоставляет разработчикам мощный и гибкий инструмент для создания программного обеспечения. Он позволяет писать эффективный и быстрый код, что делает его одним из наиболее популярных языков программирования в мире [3].

Для визуализации графа и интерфейса была использована библиотека SFML.

SFML (Simple and Fast Multimedia Library — простая и быстрая мультимедийная библиотека) — свободная кроссплатформенная мультимедийная библиотека.

Написана на C++, но доступна также для C, C#, .Net, D, Java, Python, Ruby, OCaml, Go и Rust. Представляет собой объектно-ориентированный аналог SDL. SFML содержит ряд модулей для простого программирования игр и мультимедиа приложений.

В качестве пояснения к проделанной работе был записан видеоролик. Для захвата экрана был использован OBS. OBS (Open Broadcaster Software) — это бесплатное и открытое программное обеспечение для записи видео и трансляции в реальном времени. Оно позволяет транслировать игры, вебинары, стримы и другой контент. Пользователи могут записывать свой экран, вебкамеру, игры и другие источники видео. OBS поддерживает все основные платформы, включая Windows, macOS и Linux. Для монтажа видеоролика был использован CapCut — бесплатный видеоредактор с



большим разнообразием фильтров, переходов, стикеров, шрифтов и наложений.

### 3 РАЗРАБОТКА ПРОГРАММЫ

Анализ данной задачи позволит нам оценить эффективность и корректность реализации алгоритмов работы с графами, а также выявить возможные улучшения и оптимизации. В ходе анализа мы рассмотрим основные этапы работы с графами, включая решение задачи коммивояжера.

#### Анализ задачи

- 1) Класс `Graph` описывает графы. Класс содержит вектор вершин, матрицу смежности, вектор координат вершин и словарь, где ключом является имя вершины, а значением ее координаты. В классе описаны методы редактирования графа (добавления графа и ребра, удаление графа и ребра), функции отрисовки графа.
- 2) В методе `Draw` происходит отрисовка графа в окне, переданном через параметр `window`. Код определяет размеры окна, центрирует граф и устанавливает радиус и угол для расположения вершин. Затем он проходит по списку вершин и рисует каждую из них, используя словарь, который содержит имя вершины как ключ, и координаты графа как значение. После этого код рисует все рёбра графа и расстояния между вершинами, используя матрицу смежности `adjMatrix`.
- 3) Функция `beam_length` вычисляет длину отрезка между двумя точками, используя теорему Пифагора и функцию `sqrt` для извлечения квадратного корня.
- 4) Функция `point_on_the_node_boundary` определяет координаты точки на границе узла, исходя из координат двух других точек и расстояния от одной из них. Используются функции `atan2` для определения угла и `sqrt` для извлечения квадратного корня.
- 5) `triangleArea` вычисляет площадь треугольника, заданного тремя точками. Используется формула площади треугольника через координаты его вершин.

6) `sideLength` вычисляет длину стороны треугольника, заданного двумя точками. Используется формула длины отрезка через координаты его концов.

7) `find_angle` ищет угол треугольника по координатам его вершин. Сначала вычисляется площадь треугольника, затем длины его сторон, и после этого используется формула для нахождения угла между двумя сторонами через площадь и длины сторон.

8) `string_to_int` преобразует строку в целое число. Используется потоковый ввод для преобразования строки в число.

9) `string_to_int_bool` проверяет, можно ли преобразовать строку в целое число. Используется тот же подход, что и в `string_to_int`, но возвращается булево значение, указывающее на успешность преобразования.

10) Метод `the_traveling_salesman_task` решает задачу коммивояжера. Для решения задачи коммивояжера составляется матрица условий, которая содержит расстояния между городами. Считается, что можно перейти из любого города в любой, кроме того же самого. Так как пути из города A в город A по условию нет, обозначим его -1. После построения изначальной матрицы условий проводим редукцию строк и столбцов. Редукция строк: находим минимальное расстояние в строке, вычитаем его из всех расстояний в строке, кроме -1. Редукция столбцов: находим минимальное расстояние в столбце, вычитаем его из всех расстояний в столбце, кроме -1. Оценка нулей: каждый 0 в таблице оцениваем: считаем сумму минимального элемента в строке и в столбце и запоминаем результат. Чистка карты: выбираем 0 с наибольшей оценкой, если таких несколько, выбираем любой, вычеркиваем строку и столбец, в котором находится этот 0 (вычеркивание — замена элементов строки и столбца на -1). Координаты вычеркнутого столбца и строки запоминаем (заменяем элемент на -2), назовем их ребрами. Повторяем редукцию столбцов, строк и чистку карты, пока последний переход не станет очевидным (сведем матрицу к размерам 2x2). Полученные ребра выстраиваем в последовательность и получаем наименьший путь, из исходной матрицы

берем расстояния, двигаясь по проложенному пути, получаем оптимальное расстояние всего пути.

11) Функция `clear_the_graph` очищает текущий граф. Функция очищает матрицу смежности, вектора содержащие вершины и координаты, а также словарь, содержащий элементы по принципу имя вершины – координаты.

12) Для получения значения узла, который надо добавить или удалить, необходимо текстовое поле, куда пользователь будет вводить данные. Класс `TextBox` в этом коде представляет собой текстовый редактор, который может быть использован в приложениях, созданных с помощью библиотеки SFML. Он предоставляет различные методы для настройки размера, положения, текста и других параметров. Также класс имеет внутренние структуры для рисования рамки и мигающего курсора. Методы класса включают:

- `draw`: для отрисовки текстового поля и его содержимого.
- `handleEvent`: для обработки событий ввода, таких как нажатия клавиш.
- `getCurrentText`: для получения текущего текста в текстовом поле.

Кроме того, класс `TextBox` содержит вложенный класс `Text`, который управляет отображением текста внутри текстового поля. Этот класс предоставляет методы для установки текста, его позиции и размера.

13) Класс `Button` представляет собой абстрактный класс, который определяет базовые методы для всех типов кнопок. Подклассы должны реализовать эти методы для конкретной реализации кнопки.

14) Класс `RectButton` наследуется от `Button` и реализует конкретные методы для прямоугольной кнопки. Он также содержит дополнительные члены данных, такие как `sf::RectangleShape button`, которые используются для рисования самой кнопки. Конструктор `RectButton` принимает параметры для определения размеров и положения кнопки. Деструктор не делает ничего особенного, так как он пустой. Метод `getButtonStatus` обрабатывает события

мышь и обновляет состояние кнопки (isHover, isPressed). Метод draw отвечает за отрисовку кнопки на экране. Метод setButtonLabel устанавливает надпись на кнопке.

15) Функция enter\_the\_data создает окно с заголовком "Ведите..." и содержит кнопку "Продолжить". Когда пользователь нажимает кнопку, программа закрывается. В окне также есть текстовое поле, где пользователь может ввести информацию. После закрытия окна программа возвращает введенную информацию с форматом string. Функции enter\_the\_three\_data и enter\_the\_two\_data работают аналогично и используются для ввода трех и двух данных соответственно.

16) Функция error\_or\_success\_message создает окно с заголовком title и отображает сообщение message в центре окна. Она также добавляет кнопку "Ok" внизу окна. При нажатии на кнопку "Ok" окно закрывается. Пользовательский ввод обрабатывается через событие мыши.

17) Для использования какого-либо шрифта в интерфейсе необходимо загрузить в папку с проектом файл с расширением .ttf, в котором хранится необходимый шрифт.

18) В главной функции main реализуется меню, с помощью которого пользователь взаимодействует с графом. В меню доступны функции: решение задачи коммивояжера, создание случайного графа, очистка графа, передвижение вершины, добавление и удаление узлов и ребер, переименование вершин, завершение работы.

Для данной лабораторной работы требуется uml диаграмма, которая позволит наглядно визуализировать методы и поля классов, а также взаимодействие классов между собой.

Класс TextBox (Приложение А.1) реализует текстовое поле, которое позволяет ввести в него информацию и извлечь ее, с помощью реализованных методов.

Класс Button (Приложение А.2) реализует кнопки. С помощью полей и методов класса реализовано: отслеживание наведения на кнопку курсора

мышь, нажатие на кнопку. Также кнопке можно задать необходимый цвет и отобразить на ней заданный текст. Класс `RectButton` является дочерним классом для `Button` и реализует прямоугольные кнопки.

Класс `Graph` (Приложение А.3) реализует графы. В полях класса находятся: вектор вершин, вектор координат вершин, матрица смежности, словарь (ключ – имя вершины, значение – координаты вершины). Методы класса реализуют: добавление и удаление вершины и ребра, получение количества вершин и индекс переданной вершины, а также отрисовку графа.

Листинг программы указан в Приложении Б. В работе также были использованы файлы с расширением `.ttf`.

`TrueType`— формат компьютерных шрифтов, разработанный корпорацией Apple в конце 1980-х годов. Шрифты в данном формате используются во многих современных операционных системах. Файлы с такими шрифтами имеют расширение имени `«ttf»`.

#### 4 ТЕСТИРОВАНИЕ ПРОГРАММЫ

Для тестирования написанной программы необходимо выполнить следующие шаги:

- 1) Запустить программу.
- 2) Провести необходимые операции с графом, с помощью пользовательского интерфейса.

Пояснение пользовательского интерфейса (Приложение В.1):

- Кнопка «Создать граф случайно» позволяет создать случайный граф с заданными количеством вершин (Приложение В.2.1 и Приложение В.2.2). Если граф уже создан, то предварительно очищается и создается новый граф.
- Кнопка «Граф для демонстрации» позволяет создать граф, который дан в 22 варианте лабораторной работы «Графы» (Приложение В.3).
- Кнопка «Задача Коммивояжера» решает задачу коммивояжера – находит оптимальный (кратчайший) путь для обхода всех вершин (каждая вершина посещена только 1 раз, путь заканчивается в начальной вершине). Начальная вершина запрашивается у пользователя (Приложение В.4.1). Результат алгоритма выводится пользователю в виде пути (Приложение В.4.2).
- Кнопка «Очистить граф» очищает текущий граф.
- Кнопка «Сдвинуть вершину» позволяет передвинуть вершины в произвольном порядке (Приложение В.5). Для прекращения передвижения вершин необходимо навестись на кнопку курсором мыши повторно.
- Кнопка «Добавить вершину в граф» добавляет вершину в граф. Предварительно Программа просит пользователя указать кликом мыши координаты новой вершины.
- Кнопка «Удалить вершину» удаляет вершину из графа. Предварительно программа запрашивает у пользователя имя вершины,

которую необходимо удалить. В случае, если введенной вершины не существует программы выведет сообщение об ошибке.

- Кнопка «Переименовать вершину» позволяет сменить имя существующей вершине. Предварительно программа запрашивает у пользователя: какую вершину переименовать (Приложение В.6.1). Для выбора вершины пользователь должен кликнуть на вершину мышью. После, у вершины поменяется имя (Приложение В.6.2).

- Кнопка «Добавить (Изменить) ребро» позволяет добавить новое ребро или изменить расстояние уже существующего ребра. Программа запрашивает у пользователя начальную и конечную вершину, а также расстояние между ними (Приложение В.7).

- Кнопка «Удалить ребро из графа» позволяет удалить ребро из графа.

- Кнопка «Выход» позволяет завершить работу программы.

Пояснение визуализации графа

Длина ребра графа отображена на самом ребре. Задача коммивояжера допускает различие длин ребер из вершины А в Б и из Б в А. Именно поэтому на каждом двунаправленном ребре отображено два расстояния (Приложение В.8). Например: длина ребра из вершины «6» в вершину «4» равна 10, а длина из вершины «4» в вершину «6» равна 31.



## ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были рассмотрены основные операции с графами, такие как создание и удаление вершины, создание и удаление ребра, добавление и удаление вершины, добавление и удаление ребра.

В ходе работы я применила знания о работе с классами и интерфейсами и изучила алгоритм решения задачи коммивояжера, а также научилась использовать его в графе. По ходу работы был разработан граф, операции с которым выполняются посредством работы с интерфейсом, разработанным с помощью средств SFML.

Был реализован алгоритм Дейкстры, позволяющий найти оптимальный путь. Также, были разработаны функции редактирования графа: перемещение существующих вершин, переименование вершин. В коде были реализованы особые классы, которые реализуют кнопки и текстовые боксы для упрощения реализации интерфейса. По итогу работы был реализован граф, с меню, которое позволяет управлять им.

В дальнейшем, полученные навыки могут быть применены в различных задачах программирования, связанных с обработкой данных.

В GitHub представлен полный код программы (Приложение Г.1).

Ссылка: [https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/creative%20work/traveling\\_salesman](https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/creative%20work/traveling_salesman)

На YouTube представлено видео, которое поясняет функционал программы.

Ссылка: [https://youtu.be/vfr6gA\\_pARs](https://youtu.be/vfr6gA_pARs)

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) StudFiles URL: <https://studfile.net/preview/1102862/> (дата обращения: 04.05.2024).
- 2) Решение задачи о коммивояжере // StudFiles URL: <https://studfile.net/preview/1102862/> (дата обращения: 04.05.2024).
- 3) Подбельский В.В. Язык Си++: учеб. пособие. — 5-е изд. / В.В. Подбельский. - Москва : Финансы и статистика, 2022. - 560 с. - ISBN 978-5-00184-082-4. - URL: <https://ibooks.ru/bookshelf/388183/reading> (дата обращения: 03.05.2024). - Текст: электронный.

# ПРИЛОЖЕНИЯ

## ПРИЛОЖЕНИЕ А

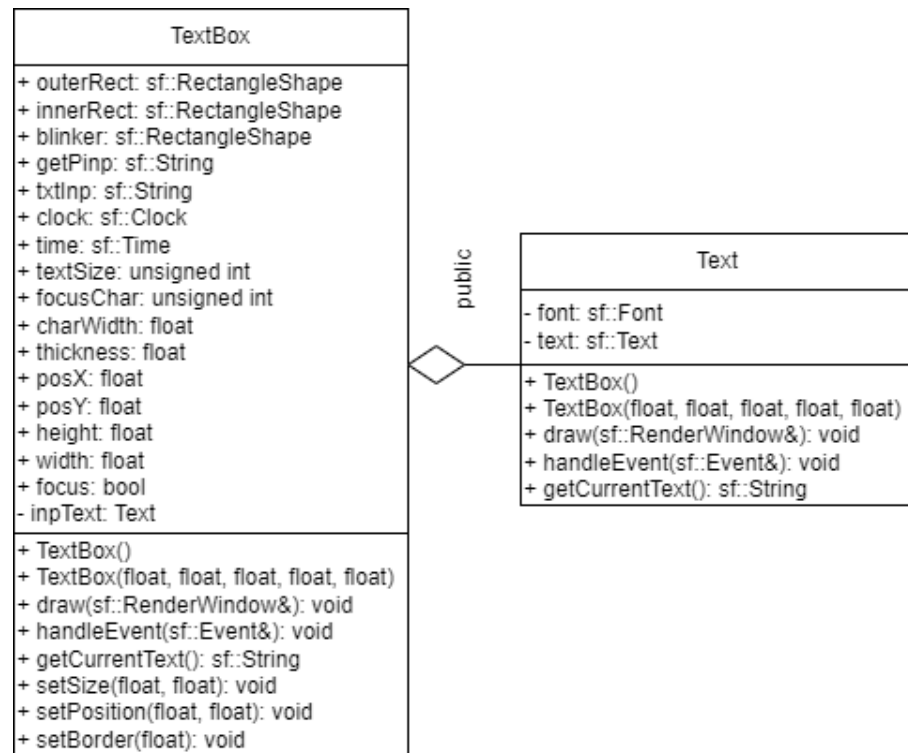


Рисунок А.1 – uml диаграмма класса TextBox

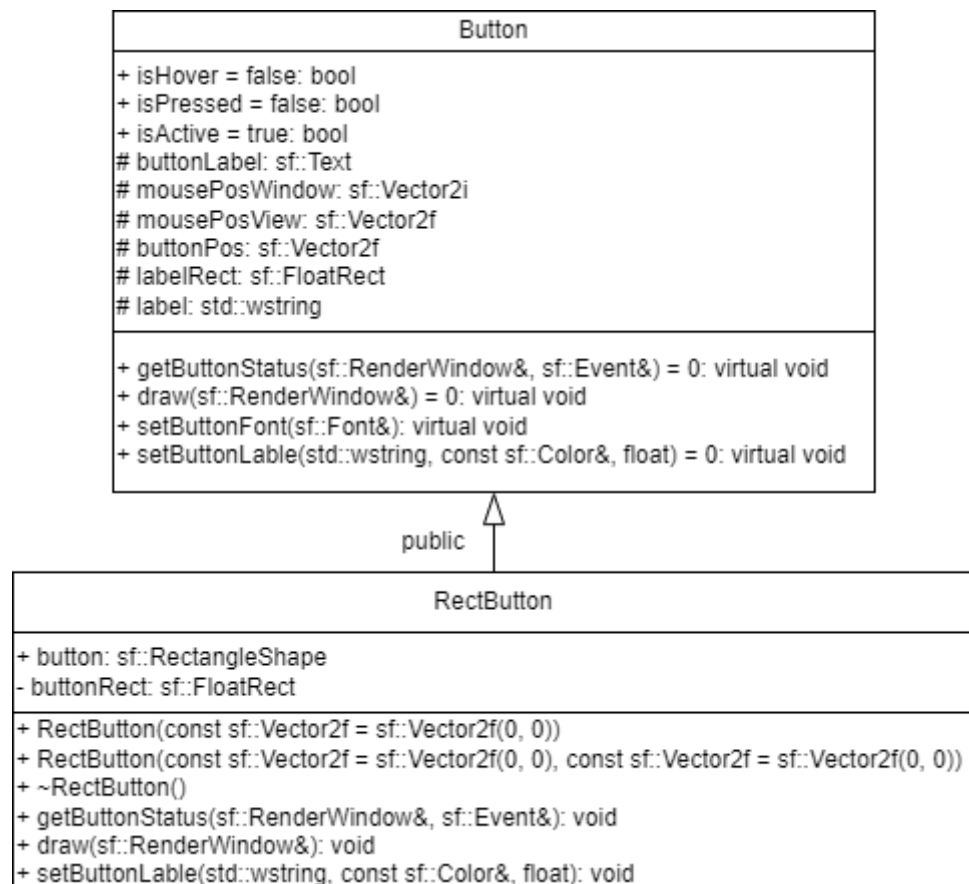


Рисунок А.2 – uml диаграмма класса Button

Graph
- Positions_vert: std::map<T, sf::Vector2f> - coord_of_the_vert: std::vector<sf::Vector2f> - vertex_list: std::vector<T> - adjMatrix: std::vector<std::vector<int>>
+ Graph(const int& size = 0) + ~Graph() {} + is_empty(): bool + insert_vertex(const T& vert, sf::Vector2f coords): void + erase_vertex(const T& vert): void + change_the_vertex_position(T vertex_0, sf::Vector2f new_coords): void + rename_a_vertex(int index, T new_name): void + get_vert_T(const int& index): T + is_it_possible_to_solve_the_traveling_salesman_problem(): bool + get_vert_pos(const T& data): int + get_amount_verts(): int + get_weight(const T& vert_1, const T& vert_2): int + get_neighbors(const T& data): std::vector<T> + insert_edge_orient(const T& vert_1, const T& vert_2, int weight = 1): void + erase_edge_orient(const T& vert_1, const T& vert_2): void + get_amount_edge_orient(): int + clear_the_graph(): void + this_is_node(sf::Vector2f cur_coord): int + Draw(sf::RenderWindow& window): void + Draw_node(sf::RenderWindow& window, std::map < T, sf::Vector2f>& positions, T vertex): void + Draw_edge(sf::RenderWindow& window, std::map < T, sf::Vector2f>& positions, T vertex_1, T vertex_2): void + Draw_distance(sf::RenderWindow& window, std::map < T, sf::Vector2f>& positions, T vertex_1, T vertex_2): void + the_traveling_salesman_task(T& vertex_start): std::vector<T> + DFS(T& start_verts, std::vector<bool>& visited_verts, std::vector<T>& vect): std::vector<T> + path_length(std::vector<T> vect_1): int

Рисунок А.3 – uml диаграмма класса Graph

## ПРИЛОЖЕНИЕ Б

### Листинг Б.1 – Файл Graphs.h

```
#pragma once
#include "other_functions.h"
#include <SFML/Graphics.hpp>
#include <vector>
#include <queue>
#include <iomanip>
#include <set>

template <class T>
class Graph {
private:
    std::map<T, sf::Vector2f> Positions_vert;
    std::vector<sf::Vector2f> coord_of_the_vert; //координаты вершин
    std::vector<T> vertex_list; //вектор вершин
    std::vector<std::vector<int>> adjMatrix; //матрица смежности
public:
    Graph(const int& size = 0); //конструктор с размером графа
    ~Graph() {} //деструктор
    bool is_empty(); //граф пуст?
    void insert_vertex(const T& vert, sf::Vector2f coords); //вставка вершины
    void erase_vertex(const T& vert); //удаление вершины

    void change_the_vertex_position(T vertex_0, sf::Vector2f
new_coords); //изменение позиции вершины
    void rename_a_vertex(int index, T new_name); //переименование вершины

    T get_vert_T(const int& index);
    bool is_it_possible_to_solve_the_traveling_salesman_problem(); //проверяю
можно ли запустить коммивояжера

    int get_vert_pos(const T& data); //ИНДЕКС вершины с переданными данными
    int get_amount_verts(); //количество существующих вершин
    int get_weight(const T& vert_1, const T& vert_2); //вес пути между вершинами
    std::vector<T> get_neighbors(const T& data); //вектор соседей элемента с
переданными данными
    void insert_edge_orient(const T& vert_1, const T& vert_2, int weight = 1);
//вставка ребра между двумя узлами - ОРИЕНТИРОВАННЫЙ граф
    void erase_edge_orient(const T& vert_1, const T& vert_2); //удаление ребра
между двумя узлами
    int get_amount_edge_orient(); //количество ребер - ОРИЕНТИРОВАННЫЙ граф

    void clear_the_graph(); //очищаю граф
    int this_is_node(sf::Vector2f cur_coord); //это корды вершины?

    void Draw(sf::RenderWindow& window); //отрисовываю граф полностью
    void Draw_node(sf::RenderWindow& window, std::map < T, sf::Vector2f>&
positions, T vertex); //отрисовываю вершину
    void Draw_edge(sf::RenderWindow& window, std::map < T, sf::Vector2f>&
positions, T vertex_1, T vertex_2); //отрисовываю ребро
    void Draw_distance(sf::RenderWindow& window, std::map < T, sf::Vector2f>&
positions, T vertex_1, T vertex_2); //отрисовываю расстояние

    std::vector<T> the_traveling_salesman_task(T& vertex_start); //решение
задачи коммивояжера
    std::vector<T> DFS(T& start_verts, std::vector<bool>& visited_verts,
std::vector<T>& vect); //обход графа в ГЛУБИНУ
    int path_length(std::vector<T> vect_1); //длина пути
};

template <class T>
int Graph<T>::path_length(std::vector<T> vect_1) { //общая длина маршрута
    int the_sum_of_the_lengths = 0;
    for (int i = 0; i < this->get_amount_verts(); i++) {
```

```

        the_sum_of_the_lengths += this-
>adjMatrix[get_vert_pos(vect_1[i])][get_vert_pos(vect_1[i + 1])];
    }
    return the_sum_of_the_lengths;
}

template <class T>
std::vector<T> Graph<T>::the_traveling_salesman_task(T& vertex_start) { //решение
задачи коммивояжера
    int memory_i_1, memory_j_1, memory_i_2 = -1, memory_j_2 = -1,
num_of_positive_paths;
    int min_path, min_i_path, min_j_path, max_zero;
    bool flag_status = false; //найден ли нормальный путь
    int quantity_of_nodes = this->get_amount_verts();
    std::vector<std::vector<int>> matrix_of_conditions = copyNestedVector(this-
>adjMatrix);
    std::vector<T> vect_of_paths;
    for (int i = 0; i < quantity_of_nodes; i++) { //заполняю матрицу смежности
        for (int j = 0; j < quantity_of_nodes; j++) {
            if (matrix_of_conditions[i][j] == 0) { //если нет пути
                matrix_of_conditions[i][j] = -1; //заполняю, т.к. нет пути
из текущего города 1 в текущий город 2
            }
        }
    }
    for (int v = 0; v < quantity_of_nodes; v++) { //прохожу по всем городам
        // необходим один элемент в столбце или строке, который будет
единственным возможным путем
        flag_status = false; //найден искомый элемент?
        for (int i = 0; (i < quantity_of_nodes) && (flag_status == false);
i++) { //пока не пройду все элементы или не найден необходимый элемент
            num_of_positive_paths = 0; //количество положительных путей
            for (int j = 0; j < quantity_of_nodes && num_of_positive_paths
< 2; j++) { //прохожу по столбцам
                if (matrix_of_conditions[i][j] >= 0) { //если есть
неотмеченный путь между этими узлами
                    memory_i_1 = i; //запоминаю строку
                    memory_j_1 = j; //запоминаю столбец
                    num_of_positive_paths++; //увеличиваю количество
положительных путей
                }
            }
            if (num_of_positive_paths == 1) { //если есть только одно
положительное значение в столбце
                flag_status = true; //найден только 1 путь
            }
        }
        for (int j = 0; (j < quantity_of_nodes) && (flag_status == false);
j++) { //пока не пройду все элементы или не найден необходимый элемент
            num_of_positive_paths = 0; //количество положительных путей
            for (int i = 0; i < quantity_of_nodes && num_of_positive_paths
< 2; i++) { //прохожу по строкам
                if (matrix_of_conditions[i][j] >= 0) { //если есть
неотмеченный путь между этими узлами
                    memory_i_1 = i; //запоминаю строку
                    memory_j_1 = j; //запоминаю столбец
                    num_of_positive_paths++; //увеличиваю количество
положительных путей
                }
            }
            if (num_of_positive_paths == 1) { //если есть только одно
положительное значение в строке
                flag_status = true; //найден только 1 путь
            }
        }
    }
    if (flag_status == true) { //найден только один нормальный путь

```

```

        makebase(memory_i_1, memory_j_1, matrix_of_conditions,
quantity_of_nodes); //пересобираю матрицу смежности с учетом найденного пути,
помечаю элемент как базовый
    }
    else { //если не найден ни одного нормального пути
        for (int i = 0; i < quantity_of_nodes; i++) { //иду по строкам
матрицы - редукция строк
            min_path = 100000; //недостижимый максимум
            for (int j = 0; j < quantity_of_nodes; j++) { //ищу
минимум в строке
                if (matrix_of_conditions[i][j] >= 0 &&
matrix_of_conditions[i][j] < min_path) { //если меньше текущего минимума (нули не
учитываю)
                    min_path =
matrix_of_conditions[i][j]; //обновляю минимум
                }
            }
            for (int j = 0; j < quantity_of_nodes; j++) { //иду по
строке
                if (matrix_of_conditions[i][j] >= 0) { //если
положительное значение, т.е. существует неотмеченный путь
                    matrix_of_conditions[i][j] -= min_path; //из
каждого нормального значения в строке вычитаю найденный минимум
                }
            }
        }
        for (int j = 0; j < quantity_of_nodes; j++) { //иду по столбцам
матрицы - редукция столбцов
            min_path = 100000; //недостижимый максимум
            for (int i = 0; i < quantity_of_nodes; i++) { //ищу
минимум в столбце
                if (matrix_of_conditions[i][j] >= 0 &&
matrix_of_conditions[i][j] < min_path) { //если меньше текущего минимума (нули не
учитываю)
                    min_path =
matrix_of_conditions[i][j]; //обновляю минимум
                }
            }
            for (int i = 0; i < quantity_of_nodes; i++) { //иду по
столбцу
                if (matrix_of_conditions[i][j] >= 0) { //если
положительное значение, т.е. существует неотмеченный путь
                    matrix_of_conditions[i][j] -= min_path; //из
каждого нормального значения в столбце вычитаю найденный минимум
                }
            }
        }
        //проверка нулевых значений и поиск базовых значений
        max_zero = -1; //недостижимый минимум
        for (int i = 0; i < quantity_of_nodes; i++) { //иду по строкам
матрицы
            for (int j = 0; j < quantity_of_nodes; j++) { //иду по
столбцам матрицы
                if (matrix_of_conditions[i][j] == 0) { //если
найден зануленный элемент
                    min_i_path = 100000; //недостижимый максимум
строки
                    min_j_path = 100000; //недостижимый максимум
столбца
                    for (int i_2 = 0; i_2 < quantity_of_nodes;
i_2++) { //оцениваю нули строк
                        if (matrix_of_conditions[i_2][j] >= 0
&& i_2 != i && matrix_of_conditions[i_2][j] < min_i_path) { //если есть путь,
элемент не равен текущему, меньше текущего минимума
                            min_i_path =
matrix_of_conditions[i_2][j]; //текущий минимум обновлен
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    for (int j_2 = 0; j_2 < quantity_of_nodes;
j_2++) { //оцениваю нули столбцов
        if (matrix_of_conditions[i][j_2] >= 0
&& j_2 != j && matrix_of_conditions[i][j_2] < min_j_path) { //если есть путь,
элемент не равен текущему, меньше текущего минимума
            min_j_path =
matrix_of_conditions[i][j_2]; //текущий минимум обновлен
        }
        if (min_i_path + min_j_path > max_zero) {
            max_zero = min_i_path +
min_j_path; //обновляю максимум - сумма индексов минимальных элементов в текущих
строке и столбце
            memory_i_2 = i; //запоминаю строку
            memory_j_2 = j; //запоминаю столбец
        }
    }
}
if (memory_i_2 == -1 || memory_j_2 == -1) {
    return vect_of_paths;
}
makebase(memory_i_2, memory_j_2, matrix_of_conditions,
quantity_of_nodes); //пересобираю матрицу смежности с учетом найденного пути,
помечаю элемент как базовый
}
memory_i_1 = this->get_vert_pos(vertex_start); //вектор пути
vect_of_paths.push_back(this->vertex_list[memory_i_1]); //добавляю
первоначальный город
for (int c = 0; c < quantity_of_nodes; c++) { //проход по всем городам
    bool flag_1 = true;
    for (int j = 0; j < quantity_of_nodes && flag_1; j++) { //поиск
следующего города, который является базовым значением
        if (matrix_of_conditions[memory_i_1][j] == -2) { //является ли
текущий город базовым значением
            vect_of_paths.push_back(this->vertex_list[j]);
            memory_i_1 = j; //обновление значения переменной
            flag_1 = false; //выход из цикла поиска следующего города.
        }
    }
}
return vect_of_paths;
}

template <class T>
int Graph<T>::this_is_node(sf::Vector2f cur_coord) { //это корды вершины?
    int x_cur = cur_coord.x;
    int y_cur = cur_coord.y;
    for (int i = 0; i < this->vertex_list.size(); i++) {
        if (this->coord_of_the_vert[i].x + 22 > cur_coord.x && this-
>coord_of_the_vert[i].x - 22 < cur_coord.x &&
            this->coord_of_the_vert[i].y + 22 > cur_coord.y && this-
>coord_of_the_vert[i].y - 22 < cur_coord.y) {

            return i; //индекс вершины
        }
    }
    return -1;
}

template <class T>
void Graph<T>::clear_the_graph() { //очищаю граф
    this->adjMatrix = std::vector<std::vector<T>>(0, std::vector<T>(0));
    this->Positions_vert = std::map<T, sf::Vector2f>();
    this->vertex_list = std::vector<T>();
}

```



```

        this->coord_of_the_vert = std::vector<sf::Vector2f>();
    }

template <class T>
std::vector<T> Graph<T>::DFS(T& start_verts, std::vector<bool>& visited_verts,
std::vector<T>& vect) { //обход графа в глубину
    vect.push_back(start_verts); //добавляю текущую вершину в вектор
    visited_verts[this->get_vert_pos(start_verts)] = true; //отмечаю, что
    вершина посещена
    std::vector<T> neighbors = this->get_neighbors(start_verts); //соседи данной
    вершины
    for (int i = 0; i < neighbors.size(); ++i) { //прохожу по всем соседям
        if (!visited_verts[this->get_vert_pos(neighbors[i])]) { //если узел еще
        не посещен
            this->DFS(neighbors[i], visited_verts, vect); //посещаю узел
        }
    }
    return vect;
}

template <class T>
bool Graph<T>::is_it_possible_to_solve_the_traveling_salesman_problem()
{ //проверяю можно ли запустить коммивояжера
    bool flag = true;
    for (int i = 0; i < this->get_amount_verts() && flag; i++) { //возможно ли
    из каждой вершины попасть в каждую?
        std::vector<T> vect;
        std::vector<bool> visited_verts( this->get_amount_verts() , false );
        if (!(DFS(this->vertex_list[i], visited_verts, vect).size() == this-
        >get_amount_verts())) {
            flag = false;
        }
    }
    return flag;
}

template <class T>
T Graph<T>::get_vert_T(const int& index) {
    return this->vertex_list[index];
}

template <class T>
void Graph<T>::rename_a_vertex(int index, T new_name) { //смена имени для вершины
    T old_name = this->vertex_list[index]; //старое имя
    sf::Vector2f static_coords = this->Positions_vert[old_name]; //запоминаю
    корды
    this->Positions_vert.erase(old_name); //удаляю старое имя
    this->Positions_vert[new_name] = static_coords; //старые корды по новому
    имени
    this->vertex_list[index] = new_name; //меняю имя в списке вершин
}

template <class T>
void Graph<T>::change_the_vertex_position(T vertex_0, sf::Vector2f new_coords)
{ //изменяю позицию вершины
    this->Positions_vert[vertex_0] = new_coords; //присваиваю новые координаты
    this->coord_of_the_vert[this->get_vert_pos(vertex_0)] = new_coords;
}

template <class T>
Graph<T>::Graph(const int& size) { //конструктор с размером графа
    this->adjMatrix = std::vector<std::vector<T>>(size,
std::vector<T>(size)); //устанавливаю матрицу смежности
    for (int i = 0; i < size; i++) { //иду по строкам
        for (int j = 0; j < size; j++) { //иду по столбцам
            this->adjMatrix[i][j] = 0;
        }
    }
}

```

```

    }
}

template <class T>
bool Graph<T>::is_empty() { //граф пуст?
    return this->vertex_list.size() == 0;
}

template <class T>
void Graph<T>::insert_vertex(const T& data, sf::Vector2f coords) { //вставка
    вершины
    this->vertex_list.push_back(data); //добавляю новый узел в вектор всех узлов
    std::vector<int> tmp_1(vertex_list.size(), 0); //вектор с 0 для добавленного
узла
    this->adjMatrix.push_back(tmp_1); //добавляю в матрицу новую строку
    this->coord_of_the_vert.push_back(coords);
    this->Positions_vert[data] = coords;
    for (int i = 0; i < vertex_list.size() - 1; i++) {
        this->adjMatrix[i].push_back(0); //добавляю новый столбец для нового
узла
    }
}

template <class T>
void Graph<T>::erase_vertex(const T& data) { //удаление вершины
    int index_vert = this->get_vert_pos(data);
    if (index_vert != -1) { //если такая вершина существует
        this->vertex_list.erase(this->vertex_list.begin() +
index_vert); //удаляю вершину из вектора узлов
        this->coord_of_the_vert.erase(this->coord_of_the_vert.begin() +
index_vert); //удаляю вершину из вектора координат

        this->adjMatrix[index_vert].erase(this-
>adjMatrix[index_vert].begin(), this->adjMatrix[index_vert].end());
        this->adjMatrix.erase(this->adjMatrix.begin() + index_vert);
        this->Positions_vert.erase(data);
        for (int i = 0; i < vertex_list.size(); i++) {
            this->adjMatrix[i].erase(this->adjMatrix[i].begin() +
index_vert);
        }
    }
}

template <class T>
int Graph<T>::get_vert_pos(const T& data) { //ИНДЕКС вершины с переданными данными
    for (int i = 0; i < this->vertex_list.size(); i++) { //иду по всем вершинам
        if (this->vertex_list[i] == data) { //если вершина найдена
            return i; //возвращаю индекс
        }
    }
    return -1; //если такой вершины нет
}

template <class T>
int Graph<T>::get_amount_verts() { //количество существующих вершин
    return this->vertex_list.size(); //размер вектора вершин
}

template <class T>
int Graph<T>::get_weight(const T& vert_1, const T& vert_2) { //вес пути между
вершинами
    if (this->is_empty()) { //если нет вершин
        return 0;
    }
    int position_1 = this->get_vert_pos(vert_1); //индекс узла
    int position_2 = this->get_vert_pos(vert_2); //индекс узла
    if (position_1 == -1 || position_2 == -1) { //если к-л узла нет

```

```

        return 0;
    }
    return this->adjMatrix[position_1][position_2];
}

template <class T>
std::vector<T> Graph<T>::get_neighbors(const T& data) { //вектор соседей элемента с
переданными данными, только те соседи куда можно перейти
    std::vector<T> nbrs_list; //вектор соседей
    int pos = this->get_vert_pos(data); //индекс узла в матрице смежности
    if (pos != -1) { //если узел существует
        for (int i = 0; i < this->vertex_list.size(); i++) { //прохожу по всем
узлам
            if (this->adjMatrix[pos][i] != 0) { //если есть путь между
необходимым узлом и k-л другим
                nbrs_list.push_back(this->vertex_list[i]);
            }
        }
    }
    return nbrs_list;
}

template <class T>
void Graph<T>::insert_edge_orient(const T& vert_1, const T& vert_2, int weight) {
//вставка ребра между двумя узлами
    if (this->get_vert_pos(vert_1) == -1 || this->get_vert_pos(vert_2) == -1)
    { //если вершин не существует
        return;
    }
    else {
        int position_1 = this->get_vert_pos(vert_1); //индекс узла
        int position_2 = this->get_vert_pos(vert_2); //индекс узла

        this->adjMatrix[position_1][position_2] = weight;
    }
}

template <class T>
void Graph<T>::erase_edge_orient(const T& vert_1, const T& vert_2) {
    if (this->get_vert_pos(vert_1) == -1 || this->get_vert_pos(vert_2) == -1)
    { //если вершин не существует
        return;
    }
    else {
        int position_1 = this->get_vert_pos(vert_1); //индекс узла
        int position_2 = this->get_vert_pos(vert_2); //индекс узла
        this->adjMatrix[position_1][position_2] = 0;
    }
}

template <class T>
int Graph<T>::get_amount_edge_orient() { //количество ребер - ОРИЕНТИРОВАННЫЙ граф
    int amount = 0;
    if (!this->is_empty()) {
        for (int i = 0; i < this->vertex_list.size(); i++) { //иду по строкам
            for (int j = 0; j < this->vertex_list.size(); j++) { //иду по
столбцам
                if (this->adjMatrix[i][j] != 0) {
                    if (!(this->adjMatrix[i][j] != 0 && this-
>adjMatrix[j][i] != 0 && i > j)) {
                        amount++;
                    }
                }
            }
        }
    }
}

```

```

        return amount;
    }

template <class T>
void Graph<T>::Draw_distance(sf::RenderWindow& window, std::map < T,
sf::Vector2f>& positions, T vertex_1, T vertex_2) {
    sf::Vector2f positions_1 = positions[vertex_1];
    sf::Vector2f positions_2 = positions[vertex_2];

    positions_1 = point_on_the_node_boundary(positions_2, positions_1, 22);
    positions_2 = point_on_the_node_boundary(positions_1, positions_2, 22);
    sf::Vector2f positions_new = point_on_the_node_boundary(positions_1,
positions_2, sideLength(positions_1, positions_2) / 4);

    sf::Text text_1;
    sf::Font font;
    font.loadFromFile("ofont.ru_Desyatiy.ttf");//загружаю шрифт
    text_1.setFont(font);

    text_1.setString(std::to_string(this->adjMatrix[this-
>get_vert_pos(vertex_1)][this->get_vert_pos(vertex_2)]));//настраиваю текст
    text_1.setFillColor(text_color);
    text_1.setCharacterSize(25);

    sf::FloatRect textRect = text_1.getLocalBounds();//центрирую текст
    text_1.setOrigin(textRect.left + textRect.width / 2.0f, textRect.top +
textRect.height / 2.0f);
    text_1.setPosition(positions_new);

    window.draw(text_1);
}

template <class T>
void Graph<T>::Draw(sf::RenderWindow& window) {
    sf::Vector2u size_window = window.getSize();
    unsigned int width = size_window.x;
    unsigned int height = size_window.y;

    unsigned int zero_x = width / 2 + 170;//условный центр графа по x
    unsigned int zero_y = height / 2;//условный центр графа по y

    int default_radius = 80 + this->vertex_list.size() * 15;
    float default_angle = 360 / this->vertex_list.size();
    for (int i = 0; i < this->vertex_list.size(); i++) {//иду по всем вершинам
        Draw_node(window, this->Positions_vert, this->vertex_list[i]);
    }

    for (int i = 0; i < this->adjMatrix.size(); i++) {//прохожу по матрице
смежности
        for (int j = 0; j < this->adjMatrix.size(); j++) {//прохожу по
матрице смежности
            if (this->adjMatrix[i][j] != 0) {
                Draw_edge(window, this->Positions_vert, vertex_list[i],
vertex_list[j]);//рисую стрелку
            }
        }
    }

    for (int i = 0; i < this->adjMatrix.size(); i++) {//прохожу по матрице
смежности
        for (int j = 0; j < this->adjMatrix.size(); j++) {//прохожу по
матрице смежности
            if (this->adjMatrix[i][j] != 0) {
                Draw_distance(window, this->Positions_vert,
vertex_list[i], vertex_list[j]);//пишу расстояние
            }
        }
    }
}

```

```

}

template <class T>
void Graph<T>::Draw_node(sf::RenderWindow& window, std::map < T, sf::Vector2f>&
positions, T vertex) {
    sf::Color node_color(222, 232, 201); //вершина
    sf::Vector2f position = positions[vertex]; //позиция узла
    int radiys = 20;
    sf::CircleShape circle_1(radiys); //генерирую круг
    circle_1.setFillColor(node_color); //цвет внутри круга
    circle_1.setOutlineColor(text_color); //цвет снаружи круга
    circle_1.setOutlineThickness(2); //толщина внешнего контура
    circle_1.setPosition(position.x - radiys, position.y - radiys); //позиция

    sf::Text text_1;
    sf::Font font;
    font.loadFromFile("ofont.ru_Expressway.ttf"); //загружаю шрифт
    text_1.setFont(font);

    text_1.setString(std::to_string(vertex)); //настраиваю текст
    text_1.setFillColor(text_color);
    text_1.setCharacterSize(radiys);

    sf::FloatRect textRect = text_1.getLocalBounds(); //центрирую текст
    text_1.setOrigin(textRect.left + textRect.width / 2.0f, textRect.top +
textRect.height / 2.0f);
    text_1.setPosition(sf::Vector2f(position.x, position.y));

    window.draw(circle_1); //рисую круг
    window.draw(text_1); //рисую текст
}

template <class T>
void Graph<T>::Draw_edge(sf::RenderWindow& window, std::map < T, sf::Vector2f>&
positions, T vertex_1, T vertex_2) { //рисую ребро

    sf::Color arrow_color(115, 135, 100); //стрелка

    sf::Vector2f positions_1 = positions[vertex_1];
    sf::Vector2f positions_2 = positions[vertex_2];

    positions_1 = point_on_the_node_boundary(positions_2, positions_1, 22);
    positions_2 = point_on_the_node_boundary(positions_1, positions_2, 22);

    sf::VertexArray triangleStrip(sf::TriangleStrip, 4);
    if ((positions_1.x < positions_2.x && positions_1.y < positions_2.y) ||
        (positions_1.x > positions_2.x && positions_1.y > positions_2.y)) {

        triangleStrip[0].position = sf::Vector2f(positions_1.x + 1,
positions_1.y - 1);
        triangleStrip[1].position = sf::Vector2f(positions_1.x - 1,
positions_1.y + 1);
        triangleStrip[2].position = sf::Vector2f(positions_2.x + 1,
positions_2.y - 1);
        triangleStrip[3].position = sf::Vector2f(positions_2.x - 1,
positions_2.y + 1);
    }
    else {
        triangleStrip[0].position = sf::Vector2f(positions_1.x - 1,
positions_1.y - 1);
        triangleStrip[1].position = sf::Vector2f(positions_1.x + 1,
positions_1.y + 1);
        triangleStrip[2].position = sf::Vector2f(positions_2.x - 1,
positions_2.y - 1);
        triangleStrip[3].position = sf::Vector2f(positions_2.x + 1,
positions_2.y + 1);
    }
}

```

```

triangleStrip[0].color = arrow_color;
triangleStrip[1].color = arrow_color;
triangleStrip[2].color = arrow_color;
triangleStrip[3].color = arrow_color;

sf::VertexArray myTriangles(sf::Triangles, 3); //сама стрелка
double arrow_angle = find_angle(sf::Vector2f(positions_2.x, positions_2.y),
sf::Vector2f(positions_1.x, positions_1.y), sf::Vector2f(positions_2.x,
positions_2.y + 20));
if (positions_1.x > positions_2.x) {
    arrow_angle *= -1;
}

sf::Vector2f point_2 =
calculating_node_coordinates(sf::Vector2f(positions_2.x - 10, positions_2.y + 20),
positions_2, arrow_angle);
sf::Vector2f point_3 =
calculating_node_coordinates(sf::Vector2f(positions_2.x + 10, positions_2.y + 20),
positions_2, arrow_angle);

myTriangles[0].position = sf::Vector2f(positions_2.x, positions_2.y);
myTriangles[1].position = point_2;
myTriangles[2].position = point_3;

myTriangles[0].color = arrow_color;
myTriangles[1].color = arrow_color;
myTriangles[2].color = arrow_color;

window.draw(myTriangles);
window.draw(triangleStrip);
}

```

```

template <class T>
void add_a_vertex_completely(Graph<T>& Graf_1, sf::Vector2f position) { //добавляю
вершину
    std::string str_vertex = enter_the_data(L"Введите название вершины, которую
хотите добавить (int)");
    if (string_to_int_bool(str_vertex)) {
        int vert_int = string_to_int(str_vertex); //вершина
        if (Graf_1.get_vert_pos(vert_int) == -1) {
            Graf_1.insert_vertex(vert_int, position);
            error_or_success_message(L"Вершина добавлена", L"Успех");
        }
        else {
            error_or_success_message(L"Такая вершина уже есть", L"Ошибка");
        }
    }
    else {
        error_or_success_message(L"Это не число", L"Ай-ай-ай");
    }
}

```

```

template <class T>
void delete_a_vertex_completely(Graph<T>& Graf_1) { //удаляю вершину
    std::string str_vertex = enter_the_data(L"Введите имя вершины, которую
хотите удалить (int)");
    if (string_to_int_bool(str_vertex)) {
        int vert_int = string_to_int(str_vertex); //вершина
        if (Graf_1.get_vert_pos(vert_int) != -1) {
            Graf_1.erase_vertex(vert_int);
            error_or_success_message(L"Вершина удалена", L"Успех");
        }
        else {
            error_or_success_message(L"Такой вершины нет", L"Ошибка");
        }
    }
}

```

```

    }
    else {
        error_or_success_message(L"Такой вершины нет!", L"Ай-ай-ай");
    }
}

template <class T>
void add_an_edge_completely(Graph<T>& Graf_1) { //добавляю ребро
    if (!Graf_1.is_empty()) {
        std::string vertex_1, vertex_2, content;
        enter_the_three_data(L"Добавить/Изменить ребро...", L"Введите первую
вершину", L"Введите вторую вершину", L"Введите расстояние между вершинами",
vertex_1, vertex_2, content);
        if (string_to_int_bool(vertex_1) && string_to_int_bool(vertex_2))
            { //вершины-числа?
                if (string_to_int_bool(content) && vertex_1 != vertex_2)
                    { //расстояние - число?
                        int content_int = string_to_int(content);
                        if (content_int > 0) { //расстояние положительное?
                            int vertex_1_int = string_to_int(vertex_1);
                            int vertex_2_int = string_to_int(vertex_2);
                            if (Graf_1.get_vert_pos(vertex_1_int) != -1 &&
Graf_1.get_vert_pos(vertex_2_int) != -1) { //вершины есть в графе?
                                Graf_1.insert_edge_orient(vertex_1_int,
vertex_2_int, content_int);
                                error_or_success_message(L"Ребро добавлено",
L"Успех");
                            }
                        }
                        else {
                            error_or_success_message(L"Одной из вершин
(или обеих) не существует", L"Ошибка");
                        }
                    }
                else {
                    error_or_success_message(L"Расстояние между
вершинами не корректно", L"Ошибка");
                }
            }
        else {
            error_or_success_message(L"Расстояние между вершинами не
корректно", L"Ай-ай-ай");
        }
    }
    else {
        error_or_success_message(L"Одной из вершин (или обеих) не
существует", L"Ай-ай-ай");
    }
}
else {
    error_or_success_message(L"Граф пуст!", L"Ошибка");
}
}

template <class T>
void delete_an_edge_completely(Graph<T>& Graf_1) { //удаление ребра
    std::string vertex_1, vertex_2;
    enter_the_two_data(L"Удалить ребро...", L"Введите первую вершину",
L"Введите вторую вершину", vertex_1, vertex_2);
    if (string_to_int_bool(vertex_1) && string_to_int_bool(vertex_2))
        { //вершины-числа?
            int vertex_1_int = string_to_int(vertex_1);
            int vertex_2_int = string_to_int(vertex_2);
            if (Graf_1.get_vert_pos(vertex_1_int) != -1 &&
Graf_1.get_vert_pos(vertex_2_int) != -1) { //вершины есть в графе?
                Graf_1.erase_edge_orient(vertex_1_int, vertex_2_int);
                error_or_success_message(L"Ребро удалено", L"Успех");
            }
        }
    }
}

```

```

    }
    else {
        error_or_success_message(L"Одной из вершин (или обеих) не
существует", L"Ошибка");
    }
}
else {
    error_or_success_message(L"Одной из вершин (или обеих) не
существует", L"Ай-ай-ай");
}
}

template <class T>
void a_random_graph(Graph<T>& Graf_1, sf::RenderWindow& window) { //создаю
случайный граф
    std::string number_of_vertices = enter_the_data(L"Сколько вершин у
графа?");
    if (string_to_int_bool(number_of_vertices)) {
        int number = string_to_int(number_of_vertices);
        if (number > 0) {
            Graf_1.clear_the_graph();

            sf::Vector2u size_window = window.getSize();
            unsigned int width = size_window.x;
            unsigned int height = size_window.y;

            unsigned int zero_x = width / 2 + 170; //условный центр графа по
x
            unsigned int zero_y = height / 2; //условный центр графа по y

            int default_radius = 100 + number * 15;
            float default_angle = 360 / number;

            std::vector< sf::Vector2f> def_pos_vert;
            for (int i = 0; i < number; i++) { //расчитываю позиции вершин

                def_pos_vert.push_back(calculating_node_coordinates(sf::Vector2f(zero_x,
zero_y - default_radius), sf::Vector2f(zero_x, zero_y), default_angle * i));
            }
            for (int i = 0; i < number; i++) { //добавляю вершины
                Graf_1.insert_vertex(i + 1, def_pos_vert[i]);
            }
            for (int i = 1; i <= number; i++) { //добавляю ребра
                for (int j = 1; j <= number; j++) {
                    if (i != j && rand()%2 == 0) {
                        Graf_1.insert_edge_orient(i, j,
a_random_number());
                    }
                }
            }
        }
    }
    else {
        error_or_success_message(L"Число не корректно", L"Ошибка");
    }
}
else {
    error_or_success_message(L"Это не число", L"Ай-ай-ай");
}
}

template <class T>
void THE_SAME_GRAPH(Graph<T>& Graf_1, sf::RenderWindow& window) { //граф для
демонстрации из методички (последний вариант)
    Graf_1.clear_the_graph(); //очищаю текущий граф
    int number = 6;

    sf::Vector2u size_window = window.getSize();

```



```

    unsigned int width = size_window.x;
    unsigned int height = size_window.y;

    unsigned int zero_x = width / 2 + 170; //условный центр графа по x
    unsigned int zero_y = height / 2; //условный центр графа по y

    int default_radius = 100 + number * 15;
    float default_angle = 360 / number;

    std::vector< sf::Vector2f> def_pos_vert;
    for (int i = 0; i < number; i++) {

        def_pos_vert.push_back(calculating_node_coordinates(sf::Vector2f(zero_x,
zero_y - default_radius), sf::Vector2f(zero_x, zero_y), default_angle * i));
    }
    for (int i = 0; i < number; i++) { //добавляю вершины
        Graf_1.insert_vertex(i + 1, def_pos_vert[i]); //создаю вершины
    }

    Graf_1.insert_edge_orient(1, 3, 13); //создаю все ребра
    Graf_1.insert_edge_orient(1, 4, 15);
    Graf_1.insert_edge_orient(2, 4, 20);
    Graf_1.insert_edge_orient(2, 1, 28);
    Graf_1.insert_edge_orient(3, 5, 30);
    Graf_1.insert_edge_orient(4, 6, 31);
    Graf_1.insert_edge_orient(4, 5, 39);
    Graf_1.insert_edge_orient(5, 4, 39);
    Graf_1.insert_edge_orient(5, 2, 21);
    Graf_1.insert_edge_orient(6, 1, 18);
}

template <class T>
void traveling_salesman_is_completely(Graph<T>& Graf_1) { //запуск алгоритма
коммивояжера
    if (Graf_1.is_it_possible_to_solve_the_traveling_salesman_problem() &&
!Graf_1.is_empty()) {
        std::string start_vert_string = enter_the_data(L"С какой вершины
начать?");
        if (string_to_int_bool(start_vert_string)) {
            int start_vert_int = string_to_int(start_vert_string);
            if (Graf_1.get_vert_pos(start_vert_int) != -1) {
                std::vector<T> vect_of_paths =
Graf_1.the_traveling_salesman_task(start_vert_int);
                if (vect_of_paths.size() != 0) {
                    if (!has_Duplicates_vector(vect_of_paths)) {
                        std::string all_str =
std::to_string(vect_of_paths[0]);
                        for (int i = 1; i <=
Graf_1.get_amount_verts(); i++) { //иду по всем обходам
                            all_str = all_str + " -> " +
std::to_string(vect_of_paths[i]);
                        }
                        sf::RenderWindow window(sf::VideoMode(650 +
20 * vect_of_paths.size(), 280), L"Задача Коммивояжера");

                        sf::Font font;

                        font.loadFromFile("ofont.ru_Expressway.ttf"); //загружаю шрифт

                        sf::Text mess;
                        mess.setFont(font);
                        mess.setString(L"Оптимальный путь, начиная с
вершины " + std::to_wstring(vect_of_paths[0]));
                        mess.setFillColor(text_color);
                        mess.setCharacterSize(40);
                        mess.setPosition(30, 10);

```

```

        sf::Text way;
        way.setFont(font);
        way.setString(all_str);
        way.setFillColor(text_color);
        way.setCharacterSize(39);
        way.setPosition(30, 80);

        sf::Text all_len;
        all_len.setFont(font);
        all_len.setString(L"Длина пути: " +
std::to_wstring(Graf_1.path_length(vect_of_paths)));
        all_len.setFillColor(text_color);
        all_len.setCharacterSize(39);
        all_len.setPosition(30, 130);

        RectButton button1(sf::Vector2f(150, 60),
sf::Vector2f(window.getSize().x / 2 - 75, window.getSize().y - 90));
        button1.setButtonFont(font);
        button1.setButtonLabel(L"Ok", text_color,
30);

        while (window.isOpen()) {
            sf::Vector2i mousePoz =
sf::Mouse::getPosition(window); //позиция мыши в окне
            sf::Event event;
            button1.getButtonStatus(window,
event);

            while (window.pollEvent(event))
            {
                if (event.type ==
sf::Event::Closed)
                    window.close();

                if (event.type ==
sf::Event::MouseButtonPressed) {
                    if (event.key.code ==
sf::Mouse::Left) {
                        if
(button1.isPressed) {
                            window.close();
                        }
                    }
                }
            }
            window.clear(background_color);
            button1.draw(window);
            window.draw(mess);
            window.draw(way);
            window.draw(all_len);
            window.display();
        }
        else {
            error_or_success_message(L"Алгоритм
запутался, не ругайте его ~(>_<.)~", L"Ой(");
        }
        else {
            error_or_success_message(L"Для текущей вершины
нельзя решить задачу коммивояжера", L"Ошибка");
        }
        else {
            error_or_success_message(L"Такой вершины нет",
L"Ошибка");
        }
    }
}

```

```

        else {
            error_or_success_message(L"Это не число", L"Ай-ай-ай");
        }
    }
    else {
        error_or_success_message(L"Для текущего графа нельзя решить задачу коммивояжера", L"Ошибка");
    }
}

template <class T>
void rename_the_vertex_completely(Graph<T>& Graf_1, sf::Vector2f coords) {
    int is_it_coord_vert = Graf_1.this_is_node(coords); //индекс вершины
    if (is_it_coord_vert != -1) { //если переданы координаты вершины
        std::string new_name = enter_the_data(L"Введите новое имя (int)");
        if (string_to_int_bool(new_name)) {
            int new_vert_name_int = string_to_int(new_name);
            if (new_vert_name_int > 0 && new_vert_name_int < 10000) {
                if (Graf_1.get_vert_pos(new_vert_name_int) == -1) { //если
такой вершины нет
                    Graf_1.rename_a_vertex(is_it_coord_vert,
new_vert_name_int);
                }
            }
            else {
                error_or_success_message(L"Такая вершина уже
есть", L"Ай-ай-ай");
            }
        }
        else {
            error_or_success_message(L"Некорректное значение", L"Ай-
ай-ай");
        }
    }
    else {
        error_or_success_message(L"Это не число", L"Ай-ай-ай");
    }
}
else {
    error_or_success_message(L"Это не вершина", L"Атата");
}
}
}

```

## Листинг Б.2 – Файл other functions.h

```
#pragma once
#include <vector>
#include <iostream>
#include <sstream>
#include <cmath>
#include <string>
#include <utility>
#include "sfml_button.hpp"
#include "textbox.hpp"
#define PI 3.14159265

sf::Color background_color(247, 247, 232); //фон
sf::Color button_color(199, 207, 183); //кнопка
sf::Color button_press_color(157, 173, 127); //кнопка нажата
sf::Color text_color(37, 64, 65); //текст

sf::Vector2f calculating_node_coordinates(sf::Vector2f first_point, sf::Vector2f
second_point, float angle) { //вычисляю координаты 3 точки
    sf::Vector2f third_point;
    float angle_in_radians = angle * (PI / 180);
    third_point.x = std::cos(angle_in_radians) * (first_point.x -
second_point.x) - std::sin(angle_in_radians) * (first_point.y - second_point.y) +
second_point.x;
    third_point.y = std::sin(angle_in_radians) * (first_point.x -
second_point.x) + std::cos(angle_in_radians) * (first_point.y - second_point.y) +
second_point.y;
    return third_point;
}

double beam_length(sf::Vector2f first_point, sf::Vector2f second_point) { //длина
вектора
    double dx = second_point.x - first_point.x;
    double dy = second_point.y - first_point.y;
    double distSq = dx * dx + dy * dy; // Вычисляем квадрат расстояния между
точками
    double length = sqrt(distSq); // Извлекаем квадратный корень, чтобы получить
длину луча
    return length;
}

sf::Vector2f point_on_the_node_boundary(sf::Vector2f first_point, sf::Vector2f
second_point, double minus) { //координаты до точки
    double distance = beam_length(first_point, second_point) - minus;
    double alpha = atan2(second_point.y - first_point.y, second_point.x -
first_point.x); // Находим угол альфа между лучом и осью X
    double x = first_point.x + distance * cos(alpha);
    double y = first_point.y + distance * sin(alpha);
    return sf::Vector2f(x, y);
}

double sideLength(sf::Vector2f pos_1, sf::Vector2f pos_2) { // Функция для
вычисления длины стороны треугольника
    return sqrt(pow(pos_2.x - pos_1.x, 2) + pow(pos_2.y - pos_1.y, 2));
}

double find_angle(sf::Vector2f pos_1, sf::Vector2f pos_2, sf::Vector2f pos_3)
{ //ищу угол по трем координатам треугольника
    double ab = sideLength(pos_1, pos_2); // Вычисление длин сторон треугольника
    double ac = sideLength(pos_1, pos_3);
    double bc = sideLength(pos_2, pos_3);

    double angle = acos((ab * ab + ac * ac - bc * bc) / (2 * ab * ac)); //
Вычисление угла между сторонами АВ и АС
    return angle * 180 / PI;
}
```

```

}

int string_to_int(const std::string& s) { //из строки делаю int
    std::istringstream i(s);
    int x;
    if (!(i >> x))
        return -1;
    return x;
}

bool string_to_int_bool(const std::string& s) { //проверяю можно ли сделать int из
string
    std::istringstream i(s);
    int x;
    if (!(i >> x))
        return false;
    return true;
}

std::string enter_the_data(std::wstring mes) { //получение данных от пользователя
    sf::RenderWindow window(sf::VideoMode(700, 350), L"Ведите...",
sf::Style::Titlebar | sf::Style::Close);

    sf::Font font;
    font.loadFromFile("ofont.ru_Expressway.ttf");//загружаю шрифт

    RectButton button_1(sf::Vector2f(340, 40), sf::Vector2f(325, 240));
    button_1.setButtonFont(font);
    button_1.setButtonLabel(L"Продолжить", text_color, 30);

    sf::Text text_mes;
    text_mes.setFont(font);
    text_mes.setString(mes);
    text_mes.setFillColor(text_color);
    text_mes.setCharacterSize(30);
    text_mes.setPosition(30, 70);

    sdx::TextBox::Text text("", 124, 220); //Текстовый бокс
    text.setSize(20);
    sdx::TextBox textBox(440, 32, 130, 160, 2);

    while (window.isOpen()) {
        sf::Vector2i mousePoz = sf::Mouse::getPosition(window); //ПОЗИЦИЯ МЫШИ В
окне
        sf::Event event;
        button_1.getButtonStatus(window, event);
        while (window.pollEvent(event)) {
            textBox.handleEvent(event);

            if (event.type == sf::Event::Closed) {
                window.close();
            }
            if (event.type == sf::Event::MouseButtonPressed) {
                if (event.key.code == sf::Mouse::Left) {
                    if (button_1.isPressed) {
                        window.close();
                    }
                }
            }
        }

        window.clear(background_color);
        button_1.draw(window);
        textBox.draw(window);
        window.draw(text_mes);
        window.display();
    }
}

```

```

        return textBox.getCurrentText(); //извлекаю введенный текст
    }

void enter_the_three_data(std::wstring title, std::wstring mes_1, std::wstring
mes_2, std::wstring mes_3, std::string& content_1, std::string& content_2,
std::string& content_3) { //получение данных от пользователя
    sf::RenderWindow window(sf::VideoMode(700, 470), title, sf::Style::Titlebar |
sf::Style::Close);

    sf::Font font;
    font.loadFromFile("ofont.ru_Expressway.ttf"); //загружаю шрифт

    RectButton button_1(sf::Vector2f(260, 40), sf::Vector2f(window.getSize().x / 2
- 130, window.getSize().y - 80));
    button_1.setButtonFont(font);
    button_1.setButtonLabel(L"Продолжить", text_color, 30);

    int def_pos_y = 20;

    sf::Text text_mes_1;
    text_mes_1.setFont(font);
    text_mes_1.setString(mes_1);
    text_mes_1.setFillColor(text_color);
    text_mes_1.setCharacterSize(30);
    text_mes_1.setPosition(30, def_pos_y);

    sdx::TextBox::Text text_1("", 124, 220); //Текстовый бокс
    text_1.setSize(20);
    sdx::TextBox textBox_1(440, 32, 130, def_pos_y += 60, 2);

    sf::Text text_mes_2;
    text_mes_2.setFont(font);
    text_mes_2.setString(mes_2);
    text_mes_2.setFillColor(text_color);
    text_mes_2.setCharacterSize(30);
    text_mes_2.setPosition(30, def_pos_y += 60);

    sdx::TextBox::Text text_2("", 130, 210); //Текстовый бокс
    text_2.setSize(20);
    sdx::TextBox textBox_2(440, 32, 130, def_pos_y += 60, 2);

    sf::Text text_mes_3;
    text_mes_3.setFont(font);
    text_mes_3.setString(mes_3);
    text_mes_3.setFillColor(text_color);
    text_mes_3.setCharacterSize(30);
    text_mes_3.setPosition(30, def_pos_y += 60);

    sdx::TextBox::Text text_3("", 130, 210); //Текстовый бокс
    text_3.setSize(20);
    sdx::TextBox textBox_3(440, 32, 130, def_pos_y += 60, 2);

    while (window.isOpen()) {
        sf::Vector2i mousePoz = sf::Mouse::getPosition(window); //позиция мыши в
окне

        sf::Event event;
        button_1.getButtonStatus(window, event);
        while (window.pollEvent(event)) {
            textBox_1.handleEvent(event);
            textBox_2.handleEvent(event);
            textBox_3.handleEvent(event);

            if (event.type == sf::Event::Closed) {
                window.close();
            }
            if (event.type == sf::Event::MouseButtonPressed) {
                if (event.key.code == sf::Mouse::Left) {

```

```

        if (button_1.isPressed) {
            window.close();
        }
    }
}
window.clear(background_color);

button_1.draw(window);
textBox_1.draw(window);
textBox_2.draw(window);
textBox_3.draw(window);
window.draw(text_mes_1);
window.draw(text_mes_2);
window.draw(text_mes_3);

window.display();
}
content_1 = textBox_1.getCurrentText();//извлекаю введенный текст
content_2 = textBox_2.getCurrentText();//извлекаю введенный текст
content_3 = textBox_3.getCurrentText();//извлекаю введенный текст
}

void enter_the_two_data(std::wstring title, std::wstring mes_1, std::wstring
mes_2, std::string& content_1, std::string& content_2) { //получение данных от
ПОЛЬЗОВАТЕЛЯ
    sf::RenderWindow window(sf::VideoMode(700, 370), title, sf::Style::Titlebar |
sf::Style::Close);

    sf::Font font;
    font.loadFromFile("ofont.ru_Expressway.ttf");//загружаю шрифт

    RectButton button_1(sf::Vector2f(260, 40), sf::Vector2f(window.getSize().x / 2
- 130, window.getSize().y - 80));
    button_1.setButtonFont(font);
    button_1.setButtonLabel(L"Продолжить", text_color, 30);

    int def_pos_y = 20;

    sf::Text text_mes_1;
    text_mes_1.setFont(font);
    text_mes_1.setString(mes_1);
    text_mes_1.setFillColor(text_color);
    text_mes_1.setCharacterSize(30);
    text_mes_1.setPosition(30, def_pos_y);

    sdx::TextBox::Text text_1("", 124, 220); //Текстовый бокс
    text_1.setSize(20);
    sdx::TextBox textBox_1(440, 32, 130, def_pos_y += 60, 2);

    sf::Text text_mes_2;
    text_mes_2.setFont(font);
    text_mes_2.setString(mes_2);
    text_mes_2.setFillColor(text_color);
    text_mes_2.setCharacterSize(30);
    text_mes_2.setPosition(30, def_pos_y += 60);

    sdx::TextBox::Text text_2("", 130, 210); //Текстовый бокс
    text_2.setSize(20);
    sdx::TextBox textBox_2(440, 32, 130, def_pos_y += 60, 2);

    while (window.isOpen()) {
        sf::Vector2i mousePoz = sf::Mouse::getPosition(window); //позиция мыши в
окне
        sf::Event event;
        button_1.getButtonStatus(window, event);
        while (window.pollEvent(event)) {

```

```

        textBox_1.handleEvent(event);
        textBox_2.handleEvent(event);

        if (event.type == sf::Event::Closed) {
            window.close();
        }
        if (event.type == sf::Event::MouseButtonPressed) {
            if (event.key.code == sf::Mouse::Left) {
                if (button_1.isPressed) {
                    window.close();
                }
            }
        }
        window.clear(background_color);

        button_1.draw(window);
        textBox_1.draw(window);
        textBox_2.draw(window);
        window.draw(text_mes_1);
        window.draw(text_mes_2);

        window.display();
    }
    content_1 = textBox_1.getCurrentText(); //извлекаю введенный текст
    content_2 = textBox_2.getCurrentText(); //извлекаю введенный текст
}

void error_or_success_message(std::wstring message, std::wstring title)
{ //сообщение о выполнении операции
    sf::RenderWindow window(sf::VideoMode(message.size() * 20, 250), title);

    sf::Font font;
    font.loadFromFile("ofont.ru_Expressway.ttf"); //загружаю шрифт

    sf::Text mes;
    mes.setFont(font);
    mes.setString(message);
    mes.setFillColor(sf::Color::Black);
    mes.setCharacterSize(40);
    mes.setPosition(30, 45);

    RectButton button1(sf::Vector2f(150, 60), sf::Vector2f(window.getSize().x / 2
- 75, 140)); //Вертикальная печать дерева
    button1.setButtonFont(font);
    button1.setButtonLabel(L"Ok", sf::Color::Black, 30);

    while (window.isOpen())
    {
        sf::Vector2i mousePoz = sf::Mouse::getPosition(window); //ПОЗИЦИЯ МЫШИ В
окне
        sf::Event event;
        button1.getButtonStatus(window, event);

        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
            if (event.type == sf::Event::MouseButtonPressed) {
                if (event.key.code == sf::Mouse::Left) {
                    if (button1.isPressed) {
                        window.close();
                    }
                }
            }
        }
        window.clear(background_color);
    }
}

```



```

        button1.draw(window);
        window.draw(mes);

        window.display();
    }
}

int a_random_number() { //случайное число от 1 до 100
    return rand() % 100 + 1;
}

template<typename T>
std::vector<std::vector<T>> copyNestedVector(const std::vector<std::vector<T>>&
nested) {
    std::vector<std::vector<T>> result;
    for (const auto& inner : nested) {
        std::vector<T> copiedInner;
        copiedInner.reserve(inner.size()); // Предварительное выделение памяти
        for (const auto& value : inner) {
            copiedInner.push_back(value); // Копирование каждого элемента
        }
        result.push_back(copiedInner); // Добавление скопированного внутреннего
вектора в результат
    }
    return result;
}

void makebase(int i_current, int j_current, std::vector<std::vector<int>>&
matrix_of_conditions, int quantity_of_nodes) { //обновления матрицы смежности при
нахождении базового значения
    int i, j;
    for (i = 0; i < quantity_of_nodes; i++) { //проход по всем столбцам
        if (matrix_of_conditions[i][j_current] >= 0) { //если есть путь -
положительное значение
            matrix_of_conditions[i][j_current] = -1; //блокирую путь через этот
столбец
        }
    }
    for (j = 0; j < quantity_of_nodes; j++) { //проход по всем строкам
        if (matrix_of_conditions[i_current][j] >= 0) { //если есть путь -
положительное значение
            matrix_of_conditions[i_current][j] = -1; //блокирую путь через эту
строку
        }
    }
    matrix_of_conditions[i_current][j_current] = -2; //элемент отмечен как часть
решения
    if (matrix_of_conditions[j_current][i_current] >= 0) { //если есть
неотмеченный обратный путь
        matrix_of_conditions[j_current][i_current] = -1; //если обратный путь
положителен, он также блокируется
    }
}

bool has_Duplicates_vector(std::vector<int> vec) { //проверяю наличие дубликатов не
учитывая первый элемент
    if (vec.size() > 1) {
        std::sort(vec.begin() + 1, vec.end()); // Сортировка вектора
        for (int i = 2; i < vec.size(); ++i) { // Проверка наличия дубликатов
            if (vec[i] == vec[i - 1]) {
                return true; // Найден дубликат
            }
        }
    }
    return false; // Дубликаты отсутствуют
}

```



## Листинг Б.3 – Файл textbox.hpp

```
#include<SFML/Graphics.hpp>
#ifndef SDX_TEXTBOX
#define SDX_TEXTBOX

namespace sdx {
    class TextBox {
    private:
        sf::RectangleShape outerRect;
        sf::RectangleShape innerRect;
        sf::RectangleShape blinker;
        sf::String getPinp;
        sf::String txtInp;
        sf::Clock clock;
        sf::Time time;

        unsigned int textSize;
        unsigned int focusChar;
        float charWidth;
        float thickness;
        float posX;
        float posY;
        float height;
        float width;
        bool focus;
    public:
        class Text {
        private:
            sf::Font font;
            sf::Text text;
        public:
            Text(sf::String, float, float); //конструктор, первый параметр -
            //текстовая строка, второй - позиция x, третий - позиция y
            sf::Text get(); //возвращает класс sf::Text, который можно отрисовать
            void setText(sf::String); //установка текста
            void setPosition(float, float); //позиция текста
            void setSize(unsigned int); //размер текста
        };
        TextBox(); //конструктор
        TextBox(float, float, float, float, float); //первые два параметра задают
        //размер, вторые два - положение, а последний - толщину
        void draw(sf::RenderWindow&); //отрисовка
        void handleEvent(sf::Event&); //обрабатывает ввод текста
        sf::String getCurrentText(); //получаю то, что в текстовом поле
    public:
        void setSize(float, float); //размер окна обновления - первый параметр
        //для x, второй для y
        void setPosition(float, float); //ставлю позицию (x,y)
        void setBorder(float); //ставлю толщину границы
    private:
        Text inpText;
    };
}
#endif
```

## Листинг Б.4 – Файл textbox.cpp

```
#include "textbox.hpp"

namespace sdx {
    sf::Color textbar_color(226, 243, 234); //фон текстбара
    TextBox::Text::Text(sf::String string, float x, float y) {
        font.loadFromFile("monospace.ttf");
        text.setFont(font);
        text.setString(string);
        text.setFillColor(sf::Color::Black);
        text.setCharacterSize(18);
        text.setPosition(sf::Vector2f(x, y));
        text.setLineSpacing(0);
        text.setOutlineThickness(0);
    }

    sf::Text TextBox::Text::get() { return text; }
    void TextBox::Text::setText(sf::String string) { text.setString(string); }
    void TextBox::Text::setPosition(float x, float y) {
        text.setPosition(sf::Vector2f(x, y));
    }
    void TextBox::Text::setSize(unsigned int x) { text.setCharacterSize(x); }

    TextBox::TextBox() : inpText("", 6, 5) {
        outerRect.setSize(sf::Vector2f(460, 32));
        innerRect.setSize(sf::Vector2f(456, 28));
        outerRect.setPosition(sf::Vector2f(0, 0));
        innerRect.setPosition(sf::Vector2f(2, 2));
        outerRect.setFillColor(sf::Color::Black);
        innerRect.setFillColor(textbar_color);

        blinker.setSize(sf::Vector2f(1, 26));
        blinker.setPosition(sf::Vector2f(4, 3));
        blinker.setFillColor(sf::Color::Black);

        time = sf::Time::Zero;
        textSize = 18;
        getPinp = "";
        txtInp = "";
        thickness = 2;
        posX = 0;
        posY = 0;
        height = 32;
        width = 460;
        focusChar = 0;
        focus = false;
        charWidth = 0;
    }

    TextBox::TextBox(float x1, float x2, float y1, float y2, float z) :
    inpText("", y1 + z + 2, y2 + z - 1) {
        outerRect.setSize(sf::Vector2f(x1, x2));
        innerRect.setSize(sf::Vector2f(x1 - 2 * z, x2 - 2 * z));
        outerRect.setPosition(sf::Vector2f(y1, y2));
        innerRect.setPosition(sf::Vector2f(y1 + z, y2 + z));
        outerRect.setFillColor(sf::Color::Black);
        innerRect.setFillColor(textbar_color);

        blinker.setSize(sf::Vector2f(1, x2 - 2 * z - 2));
        blinker.setPosition(sf::Vector2f(y1 + z + 2, y2 + z + 1));
        blinker.setFillColor(sf::Color::Black);

        time = sf::Time::Zero;
        textSize = (unsigned int)(x2 - 4 - 2 * z);
        getPinp = "";
        txtInp = "";
        thickness = z;
        posX = y1;
    }
}
```

```

        posY = y2;
        height = x2;
        width = x1;
        focusChar = 0;
        focus = false;
        charWidth = 0;
        inpText.setSize(textSize);
    }

    void TextBox::setSize(float x, float y) {
        height = y;
        width = x;
        textSize = (unsigned int)(y - 4 - 2 * thickness);
        outerRect.setSize(sf::Vector2f(x, y));
        innerRect.setSize(sf::Vector2f(x - 2 * thickness, y - 2 * thickness));
        blinker.setSize(sf::Vector2f(1, y - 2 * thickness - 2));
        inpText.setSize(textSize);
        inpText.setPosition(posX + thickness + 2, posY + thickness - 1);
    }

    void TextBox::setPosition(float x, float y) {
        posX = x;
        posY = y;
        outerRect.setPosition(sf::Vector2f(x, y));
        innerRect.setPosition(sf::Vector2f(x + thickness, y + thickness));
        blinker.setPosition(sf::Vector2f(x + thickness + 2, y + thickness + 1));
        inpText.setPosition(x + thickness + 2, y + thickness - 1);
    }

    void TextBox::setBorder(float x) {
        thickness = x;
        textSize = (unsigned int)(height - 4 - 2 * x);
        innerRect.setSize(sf::Vector2f(width - 2 * x, height - 2 * x));
        inpText.setSize(textSize);
        setPosition(posX, posY);
    }

    sf::String TextBox::getCurrentText() { return getPinp; }

    void TextBox::handleEvent(sf::Event& event) {
        if (event.type == sf::Event::TextEntered && focus) {
            if ((inpText.get().findCharacterPos(focusChar).x + 1.2 * textSize) <
                (width + posX) && 31 < int(event.text.unicode) && 256 > int(event.text.unicode)) {
                if (focusChar == getPinp.getSize()) getPinp += event.text.unicode;
                else {
                    getPinp = getPinp.substring(0, focusChar) + event.text.unicode
                        + getPinp.substring(focusChar, getPinp.getSize() - focusChar);
                }
                focusChar++;
            }
        }
        if (event.type == sf::Event::KeyPressed && focus) {
            if (event.key.code == sf::Keyboard::BackSpace) {
                if (focusChar != 0) {
                    getPinp.erase(focusChar - 1, 1);
                    if (focusChar > 0) focusChar--;
                }
            }
            if (event.key.code == sf::Keyboard::Delete) {
                if (focusChar != getPinp.getSize()) {
                    getPinp.erase(focusChar);
                }
            }
            else if (event.key.code == sf::Keyboard::Left) {
                if (focusChar > 0) { focusChar--; }
            }
            else if (event.key.code == sf::Keyboard::Right) {

```

```

        if (focusChar < getPinp.getSize()) focusChar++;
    }
}
if (event.type == sf::Event::MouseButtonPressed) {
    if (event.mouseButton.button == sf::Mouse::Left) {
        if (getPinp.getSize() > 0) {
            if (charWidth == 0) charWidth =
inpText.get().findCharacterPos(1).x - inpText.get().findCharacterPos(0).x;
            unsigned int temp = (unsigned int)((event.mouseButton.x -
posX) / charWidth);
            if (temp > getPinp.getSize()) focusChar = getPinp.getSize();
            else focusChar = temp;
        }
        if (event.mouseButton.x > posX && event.mouseButton.x < posX +
width && event.mouseButton.y > posY && event.mouseButton.y < posY + height) focus =
true;
        else focus = false;
    }
}
}

void TextBox::draw(sf::RenderWindow& window) {
    time += clock.restart();
    if (focus) {
        if (time.asSeconds() > 1) {
            time = sf::Time::Zero;
            blinker.setFillColor(sf::Color::Black);
        }
        else if (time.asSeconds() > 0.5) blinker.setFillColor(textbar_color);
    }
    else {
        blinker.setFillColor(textbar_color);
        if (time.asSeconds() > 300) time = sf::Time::Zero;
    }
    if (focusChar == 0) blinker.setPosition(posX + thickness + 2, posY +
thickness + 1);
    else
blinker.setPosition(sf::Vector2f(inpText.get().findCharacterPos(focusChar).x, posY
+ thickness + 1));
    inpText.setText(getPinp);
    window.draw(outerRect);
    window.draw(innerRect);
    window.draw(blinker);
    window.draw(inpText.get());
}
}

```

## Листинг Б.5 – Файл sfml\_button.hpp

```
#pragma once

#ifndef BUTTON_HPP_INCLUDED
#define BUTTON_HPP_INCLUDED
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>

const sf::Color defaultHovered = sf::Color(199, 207, 183); //цвет кнопки
const sf::Color defaultPressed = sf::Color(157, 173, 127); //цвет кнопки если она
нажата

class Button{
public:
    virtual void getButtonStatus(sf::RenderWindow&, sf::Event&) = 0; //статус
кнопки
    virtual void draw(sf::RenderWindow&) = 0; //отображение кнопки
    virtual void setButtonFont(sf::Font&); //шрифт текста на кнопке
    virtual void setButtonLabel(std::wstring, const sf::Color&, float) =
0; //установка надписи на кнопке

    bool isHover = false; //курсор наведен?
    bool isPressed = false; //нажата или нет
    bool isActive = true; //состояние кнопки

protected:
    sf::Text buttonLabel; //буквы на кнопке
    sf::Vector2i mousePosWindow; //позиция мыши
    sf::Vector2f mousePosView;
    sf::Vector2f buttonPos; //позиция кнопки
    sf::FloatRect labelRect;
    std::wstring label; //надпись
};

class RectButton : public Button{ //подкласс прямоугольных кнопок
public:
    RectButton(const sf::Vector2f = sf::Vector2f(0, 0)); //конструкторы
    RectButton(const sf::Vector2f = sf::Vector2f(0, 0), const sf::Vector2f =
sf::Vector2f(0, 0));
    ~RectButton(); //деструктор

    void getButtonStatus(sf::RenderWindow&, sf::Event&); //статус кнопки (нажата/не
нажата)
    void draw(sf::RenderWindow&); //отображение кнопки
    void setButtonLabel(std::wstring, const sf::Color&, float); //отображение
надписи
    sf::RectangleShape button;

private:
    sf::FloatRect buttonRect;
};
#endif
```

## Листинг Б.6 – Файл sfml\_button.cpp

```
#include "sfml_button.hpp"

void Button::setButtonFont(sf::Font& font) { //ставлю шрифт
    buttonLabel.setFont(font);
}
```



## Листинг Б.7 – Файл RectButton.cpp

```
#include "SFML_button.hpp"

RectButton::RectButton(const sf::Vector2f size) { //конструктор
    this->button.setSize(size);
    this->buttonRect = this->button.getLocalBounds();
}

RectButton::RectButton(const sf::Vector2f size, const sf::Vector2f
position) { //конструктор
    this->button.setSize(size);
    this->button.setPosition(position);
    this->buttonPos = position;
    this->buttonRect = this->button.getLocalBounds();
}

RectButton::~RectButton() {} //Деструктор

void RectButton::getButtonStatus(sf::RenderWindow& window, sf::Event&
event) //статус
{
    this->mousePosWindow = sf::Mouse::getPosition(window); //позиция мыши в окне
    this->mousePosView = window.mapPixelToCoords(this->mousePosWindow);

    this->isHover = false; //курсор наведен?
    this->isPressed = false; //нажато?

    if (isActive) { //если кнопка активна
        if (button.getGlobalBounds().contains(this->mousePosView)) { //курсор
наведен
            this->isHover = true;
        }
        if (button.getGlobalBounds().contains(this->mousePosView)) { //после 1
нажатия кнопки
            this->isPressed = true; //состояние меняется на активное - кнопка
нажата
        }

        if (isHover) { //курсор наведен
            button.setFillColor(defaultPressed); //меняю цвет при наведении
        }
        else button.setFillColor(defaultHovered); //обычный цвет

        if (isPressed) { //кнопка нажата
            button.setFillColor(defaultPressed); //возвращаю истинный цвет
        }
    }
    else { //обычный цвет
        button.setFillColor(defaultPressed);
    }
}

void RectButton::draw(sf::RenderWindow& window) { //отображаю кнопку в окне
    window.draw(button); //кнопка
    window.draw(buttonLabel); //надпись
}

void RectButton::setButtonLable(std::wstring label, const sf::Color& color, float
charSize) { //устанавливаю надпись
    this->buttonLabel.setString(label); //надпись
    this->buttonLabel.setCharacterSize(charSize); //размер символов
    this->buttonLabel.setFillColor(color); //устанавливаю цвет
    this->label = label; //присваиваю надпись

    this->labelRect = this->buttonLabel.getLocalBounds();
}
```

```
        this->buttonLabel.setOrigin(this->labelRect.width / 2.0f, this->labelRect.height / 2.0f); //ставлю координаты

        this->buttonLabel.setPosition(this->buttonPos.x + (this->buttonRect.width / 2.0f), //ставлю координаты
            this->buttonPos.y + (this->buttonRect.height / 4.0f) + 7);
    }
```

## Листинг Б.8 – Файл main.cpp

```
#include <SFML/Graphics.hpp>
#include "Graph.h"
#include "other_functions.h"

int main() {
    system("chcp 1251 > Null");
    srand(time(0));
    sf::Font jackInput;

    sf::RenderWindow window(sf::VideoMode(1300, 700), "Graph - menu");//главное
    окно
    jackInput.loadFromFile("ofont.ru_Nikoleta.ttf");
    int tmp_size_y = window.getSize().y / 2 - 310;

    Graph<int> Graf_2;//создаю граф

    int tmp_1 = 1;

    sf::Text explanation;
    explanation.setFont(jackInput);
    explanation.setString(L"");
    explanation.setFillColor(text_color);
    explanation.setCharacterSize(40);
    explanation.setPosition(380, 20);

    sf::Text menu;
    menu.setFont(jackInput);
    menu.setString(L"Меню");
    menu.setFillColor(text_color);
    menu.setCharacterSize(40);
    menu.setPosition(150, tmp_size_y);

    RectButton button_2(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
    button_2.setButtonFont(jackInput);
    button_2.setButtonLabel(L"Задача Коммивояжера", text_color, 30);

    RectButton button_1(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
    button_1.setButtonFont(jackInput);
    button_1.setButtonLabel(L"Создать граф случайно", text_color, 30);

    RectButton button_10(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
    button_10.setButtonFont(jackInput);
    button_10.setButtonLabel(L"Граф для демонстрации", text_color, 30);

    RectButton button_8(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
    button_8.setButtonFont(jackInput);
    button_8.setButtonLabel(L"Очистить граф", text_color, 30);

    RectButton button_3(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
    button_3.setButtonFont(jackInput);
    button_3.setButtonLabel(L"Сдвинуть вершину", text_color, 30);

    RectButton button_4(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
    button_4.setButtonFont(jackInput);
    button_4.setButtonLabel(L"Добавить вершину в граф", text_color, 30);

    RectButton button_5(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
    button_5.setButtonFont(jackInput);
```

```

button_5.setButtonLabel(L"Удалить вершину из графа", text_color, 30);

RectButton button_9(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
button_9.setButtonFont(jackInput);
button_9.setButtonLabel(L"Переименовать вершину", text_color, 30);

RectButton button_6(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
button_6.setButtonFont(jackInput);
button_6.setButtonLabel(L"Добавить (Изменить) ребро", text_color, 30);

RectButton button_7(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
50));
button_7.setButtonFont(jackInput);
button_7.setButtonLabel(L"Удалить ребро из графа", text_color, 30);

RectButton button_exit(sf::Vector2f(360, 40), sf::Vector2f(20, tmp_size_y +=
80));
button_exit.setButtonFont(jackInput);
button_exit.setButtonLabel(L"Выход", text_color, 30);

bool flag_add_vert = false; //надо добавить вершину
bool flag_move_vert = false; //надо сдвинуть вершину
bool flag_motion_tracking = false; //надо отслеживать движения мыши
bool flag_return_the_normal_but = false; //при нажатии на кнопку передвижение
вершин прекращается
bool flag_rename = false; //надо переименовать вершину

int is_it_coord_vert = -1; //индекс вершины
int vert_cont_T = -1; //имя вершины

while (window.isOpen()) {
    sf::Vector2i mousePoz = sf::Mouse::getPosition(window); //позиция мыши в
окне

    sf::Event event;
    button_1.getButtonStatus(window, event);
    button_2.getButtonStatus(window, event);
    button_3.getButtonStatus(window, event);
    button_4.getButtonStatus(window, event);
    button_5.getButtonStatus(window, event);
    button_6.getButtonStatus(window, event);
    button_7.getButtonStatus(window, event);
    button_8.getButtonStatus(window, event);
    button_9.getButtonStatus(window, event);
    button_10.getButtonStatus(window, event);
    button_exit.getButtonStatus(window, event);

    while (window.pollEvent(event)) {
        if (event.type == sf::Event::Closed) //закреть
            window.close();

        if (flag_add_vert) { //если надо добавить вершину
            if (event.type == sf::Event::MouseButtonPressed) { //если нажата
кнопка
                if (event.key.code == sf::Mouse::Left) { //если нажата левая
кнопка
                    flag_add_vert = false;
                    add_a_vertex_completely(Graf_2, sf::Vector2f(mousePoz.x,
mousePoz.y)); //передаю граф и текущую позицию мыши
                    explanation.setString(L"");
                }
            }
        }
        else if (flag_move_vert) { //если надо передвинуть вершину
            if (event.type == sf::Event::MouseButtonPressed) { //если мышь
нажата

```

```

        if (event.key.code == sf::Mouse::Left) { //если нажата левая
кнопка
            flag_return_the_normal_but = true;
            is_it_coord_vert =
Graf_2.this_is_node(sf::Vector2f(mousePoz.x, mousePoz.y)); //индекс вершины
            if (is_it_coord_vert != -1) { //если нажата именно вершина
                flag_motion_tracking = true; //надо отслеживать
движение
            }
        }
        if (event.type == sf::Event::MouseMove && flag_motion_tracking)
        { //если мышь движется и надо отследить движение
            int vert_cont_T = Graf_2.get_vert_T(is_it_coord_vert); //имя
вершины
            Graf_2.change_the_vertex_position(vert_cont_T,
sf::Vector2f(mousePoz.x, mousePoz.y)); //меняю расположение вершины
        }
        if (event.type == sf::Event::MouseButtonReleased &&
flag_motion_tracking) { //мышь отжата
            if (event.key.code == sf::Mouse::Left) { //отжата левая кнопка
                flag_motion_tracking = false; //движение мыши не надо
отслеживать
            }
        }
        if (button_3.isPressed && flag_return_the_normal_but) { //перестать
двигать вершину
            flag_move_vert = false;
            explanation.setString(L"");
            button_3.setButtonLabel(L"Сдвинуть вершину", text_color, 30);
            flag_return_the_normal_but = false;
        }
    }
    else if (flag_rename) { //переименовать вершину
        if (event.type == sf::Event::MouseButtonPressed) { //если мышь
нажата
            if (event.key.code == sf::Mouse::Left) { //если нажата левая
кнопка
                rename_the_vertex_completely(Graf_2,
sf::Vector2f(mousePoz.x, mousePoz.y));
                flag_rename = false;
                explanation.setString(L"");
            }
        }
    }
    else {
        if (event.type == sf::Event::MouseButtonPressed) {
            if (event.key.code == sf::Mouse::Left) {
                if (button_exit.isPressed) {
                    window.close();
                }
                else if (button_1.isPressed) { //случайный граф
                    a_random_graph(Graf_2, window); //создаю случайный граф
                }
                else if (button_2.isPressed) { //коммивояжер
                    traveling_salesman_is_completely(Graf_2);
                }
                else if (button_3.isPressed) { //сдвинуть вершину
                    flag_move_vert = true; //надо передвинуть вершину
                    explanation.setString(L"Передвиньте
вершину"); //пояснение
                    button_3.setButtonLabel(L"Перестать двигать",
text_color, 30);
                }
            }
            else if (button_4.isPressed) { //Добавить вершину в граф
                flag_add_vert = true;
            }
        }
    }
}

```

```

        explanation.setString(L"Куда добавить вершину?");
    }
    else if (button_5.isPressed) { //удалить вершину
        delete_a_vertex_completely(Graf_2);
    }
    else if (button_6.isPressed) { //добавить ребро
        add_an_edge_completely(Graf_2);
    }
    else if (button_7.isPressed) { //удалить ребро
        delete_an_edge_completely(Graf_2);
    }
    else if (button_8.isPressed) { //очистить граф
        Graf_2.clear_the_graph();
    }
    else if (button_9.isPressed) { //переименовать вершину
        flag_rename = true;
        explanation.setString(L"Выберете вершину");
    }
    else if (button_10.isPressed) { //граф из методички
        THE_SAME_GRAPH(Graf_2, window);
    }
    }
    }
}

window.clear(background_color);
if (!Graf_2.is_empty()) { //если граф не пустой
    Graf_2.Draw(window);
}
window.draw(menu);
window.draw(explanation);
button_1.draw(window);
button_2.draw(window);
button_3.draw(window);
button_4.draw(window);
button_5.draw(window);
button_6.draw(window);
button_7.draw(window);
button_8.draw(window);
button_9.draw(window);
button_10.draw(window);
button_exit.draw(window);

window.display();
}
}

```

## ПРИЛОЖЕНИЕ В

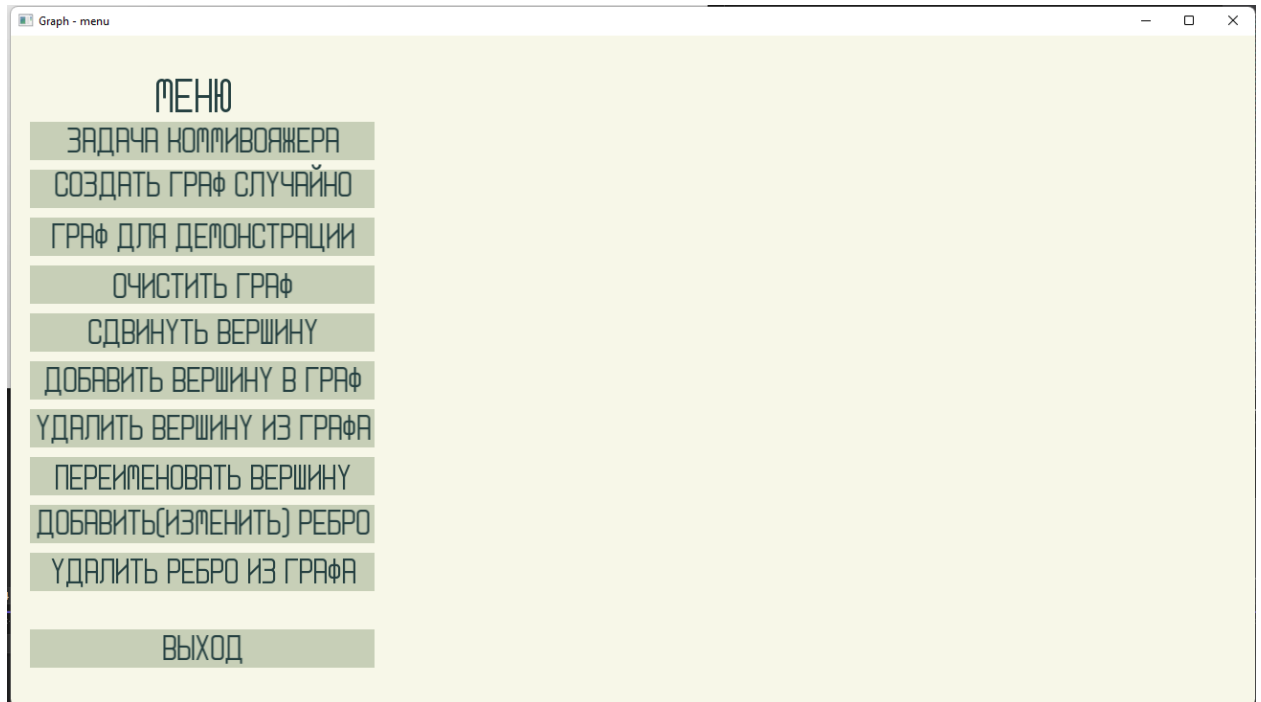


Рисунок В.1 – пользовательский интерфейс

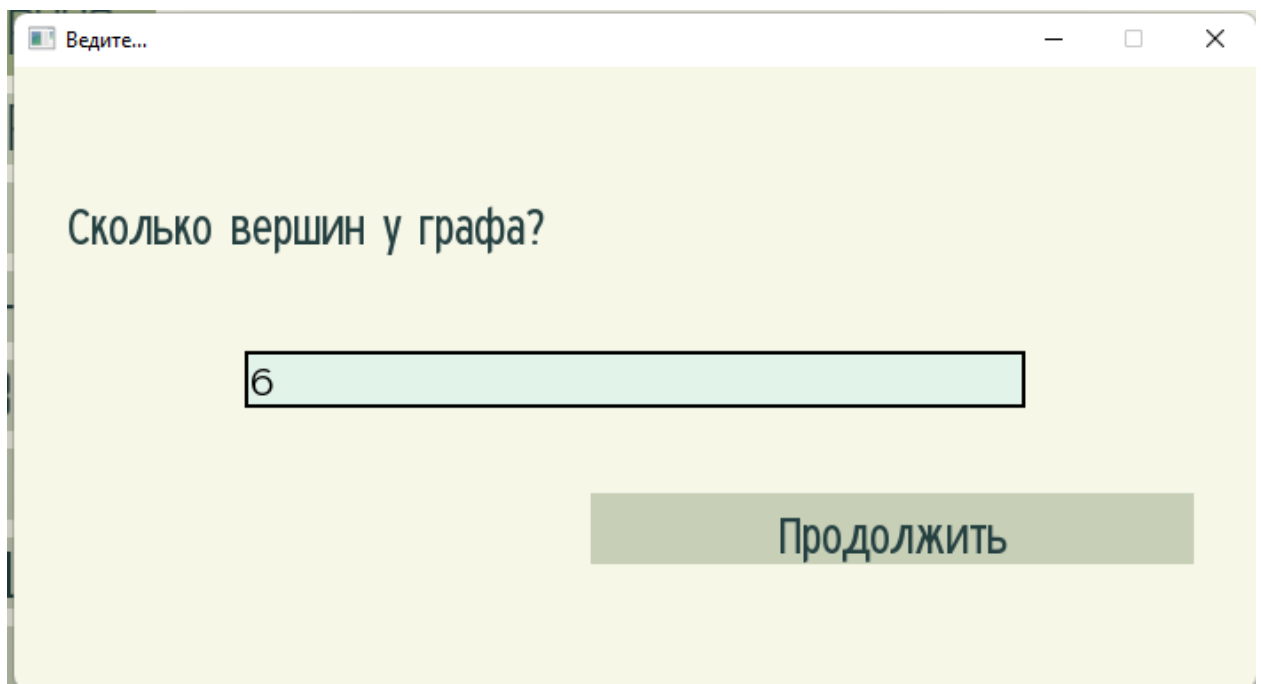


Рисунок В.2.1 – количество вершин у случайного графа

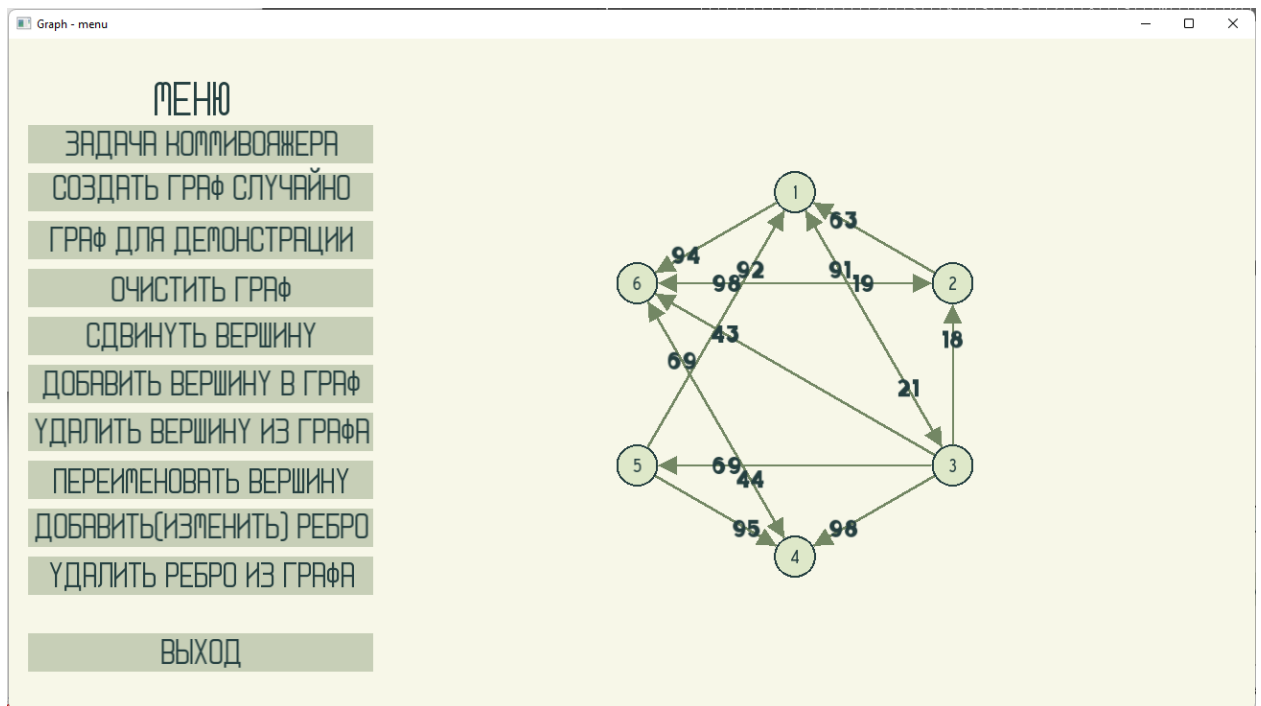


Рисунок В.2.2 – случайный граф

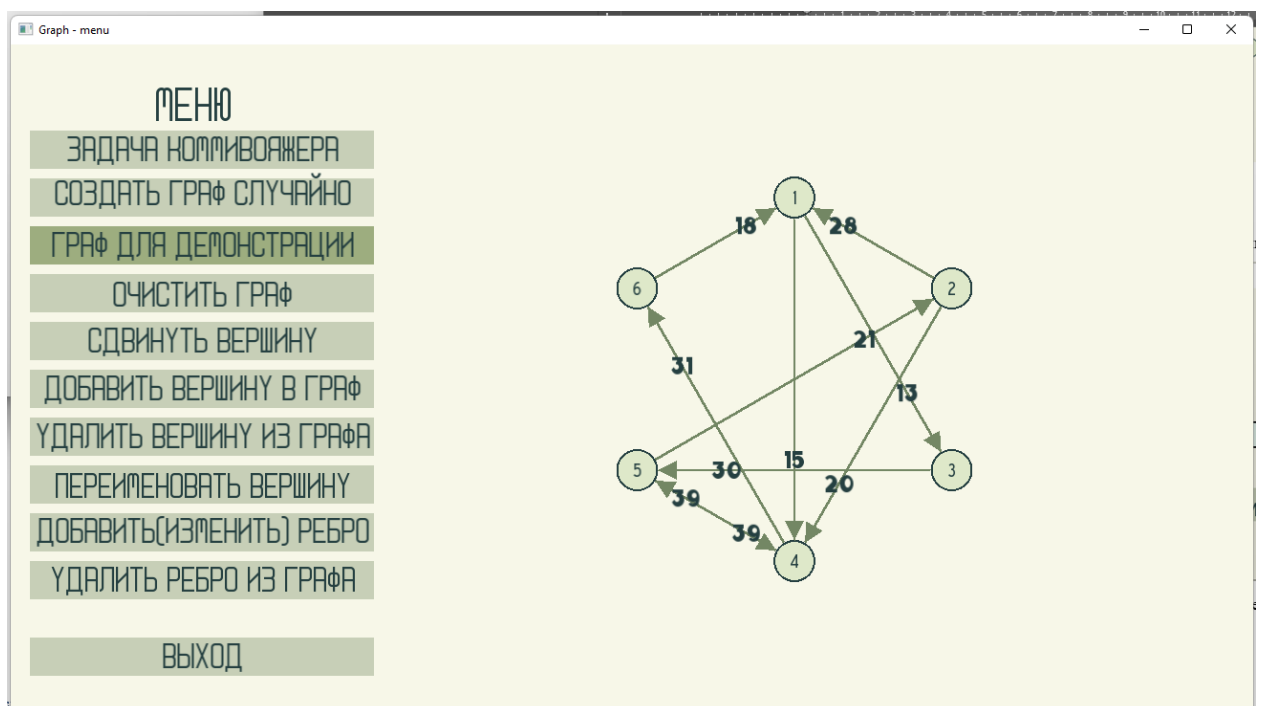


Рисунок В.3 – граф для демонстрации



Ведите...

С какой вершины начать?

5

Продолжить

Рисунок В.4.1 – запрос начальной вершины для решения задачи коммивояжера

Задача Коммивояжера

Оптимальный путь, начиная с вершины 5

5 -> 2 -> 4 -> 6 -> 1 -> 3 -> 5

Длина пути: 133

Ok

Рисунок В.4.2 – решение задачи коммивояжера

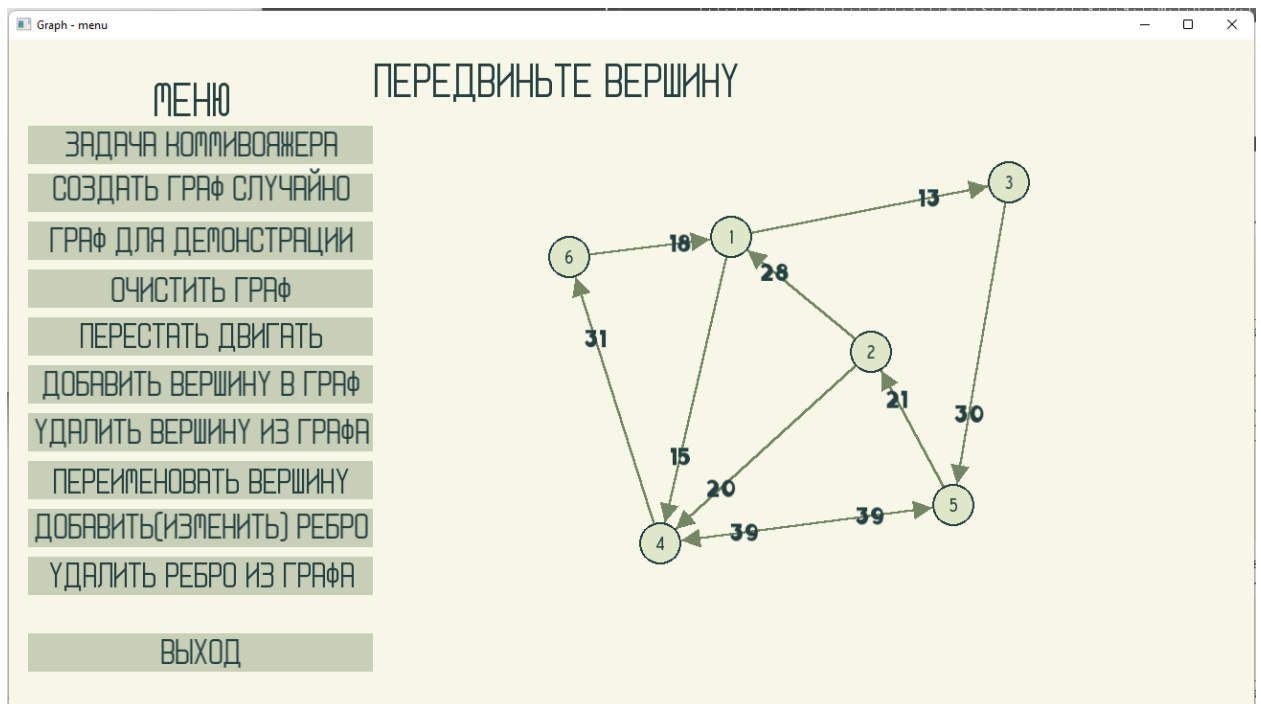


Рисунок В.5 – передвижение вершин

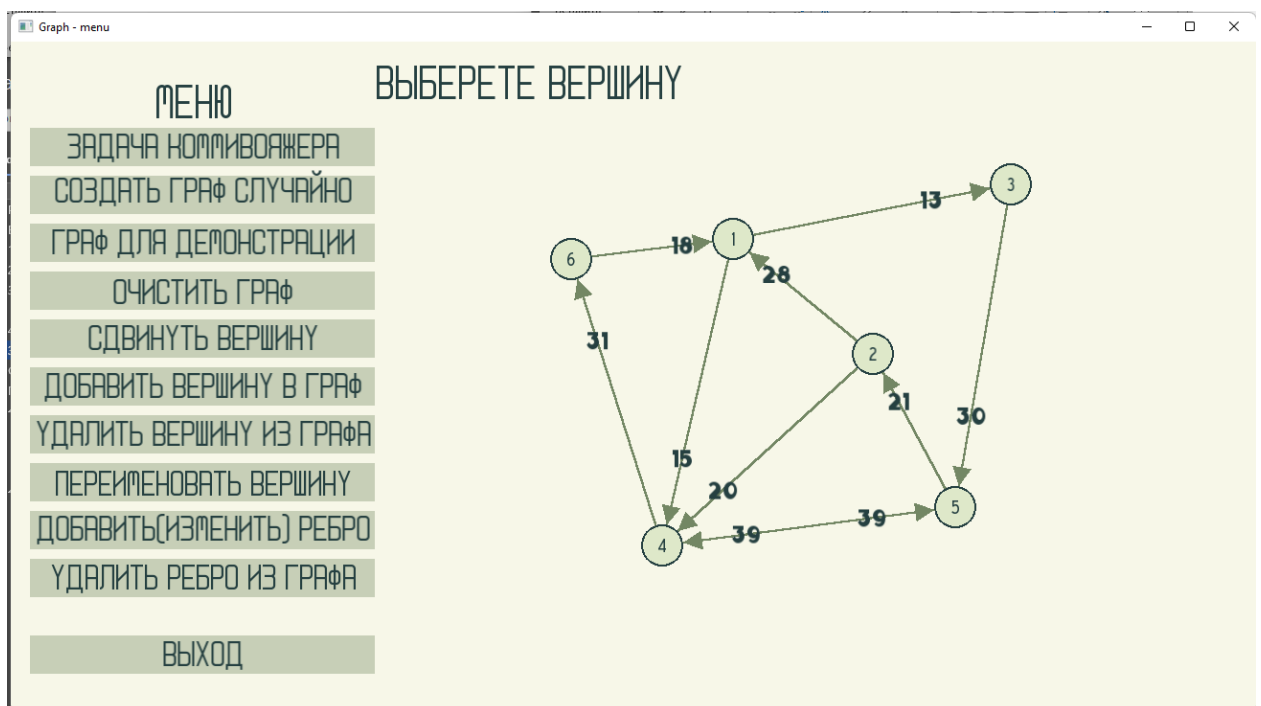


Рисунок В.6.1 – запрос имени вершины, которую необходимо переименовать

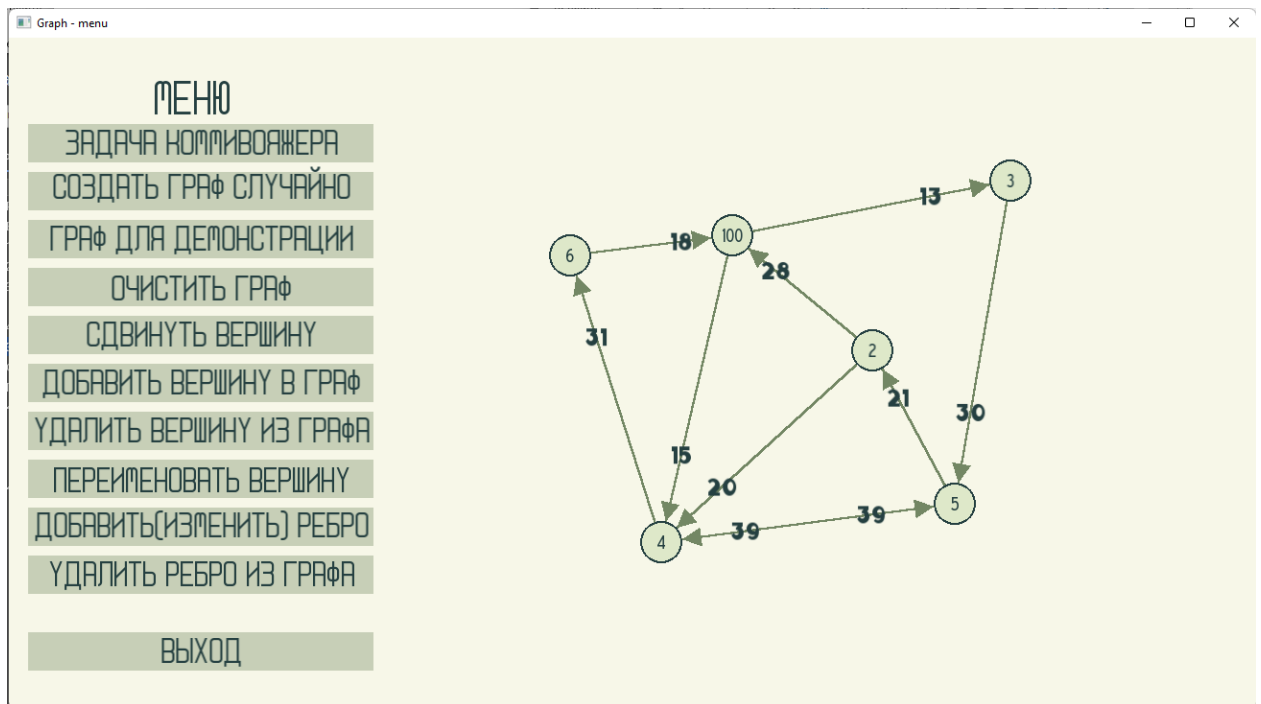


Рисунок В.6.2 – вершина переименована

The screenshot shows a window titled "Добавить/Изменить ребро..." with three input fields and a button. The labels and values are:

- Введите первую вершину: 6
- Введите вторую вершину: 4
- Введите расстояние между вершинами: 10
- Button: Продолжить

Рисунок В.7 – добавление или изменение ребра

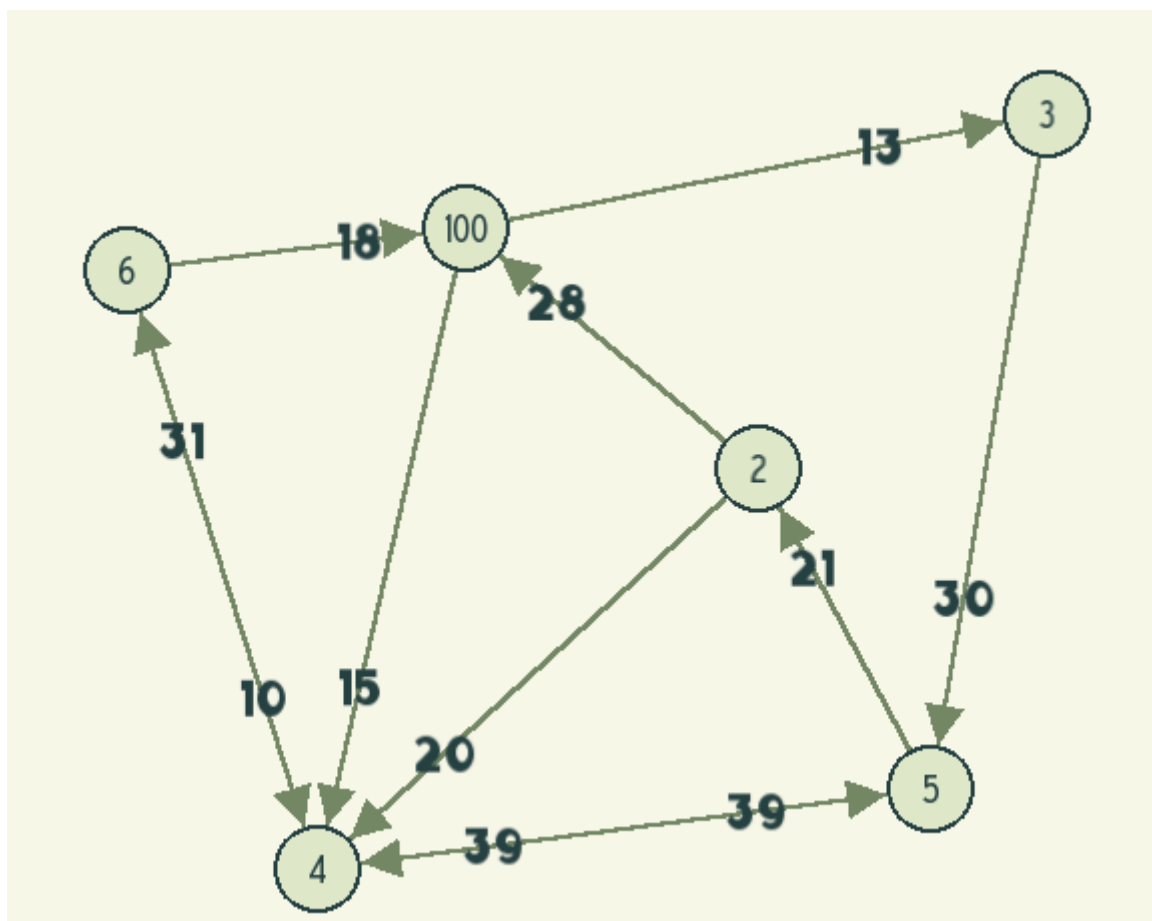


Рисунок В.8 – визуализация графа

## ПРИЛОЖЕНИЕ Г

[Laboratory-works-for-the-2-semester](#) / [creative work](#) / [traveling\\_salesman](#) / 








 <b>SonyAkb</b> Обновление uml-Graph.drawio.png	
Name	
	..
	traveling_salesman
	README
	uml-Graph.drawio.png
	uml-button.drawio.png
	uml-textBox.drawio.png

Рисунок Г.1 – результат работы в GitHub