

МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
«МОиПЭВМ»

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Разработка алгоритмов и программ решения
олимпиадных задач спортивного
программирования**

Студент

Дунаев 18.06.20 Дунаев Владислав Юрьевич
(подпись, дата) (ФИО полностью)

Руководитель

Балашова 18.06.20 Балашова И.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

(подпись, дата) Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись) Макарычев П.П.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись) Попова Н.А.
(фамилия, инициалы)

Пенза, 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
«МОиПЭВМ»

«Утверждаю»
Заведующий кафедрой
П.П. Макарычев
«__» _____ 2020 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Дунаев Владислав Юрьевич гр. 16ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Разработка алгоритмов и программ решения олимпиадных задач
спортивного программирования

Тема утверждена приказом ПГУ № 1488/о от "30" декабря 2019

2. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Балашова И.Ю.

3. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: программное средство реализует основные алгоритмы решения олимпиадных
задач спортивного программирования

Основные функции:

1) Решение задач на подотрезках массива

2) Решение задач на графах

3) Решения задач на обработку строк

Технология разработки: ООП

Язык программирования: C++

Инструментальные средства разработки: CLion

Среда исполнения: Windows MinGW, Linux GNU G++

4. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) План разработки и оценка бюджета

3) Проектирование программных средств

4) Реализация программных средств

5) Тестирование и анализ качества программного кода

6) Рефакторинг программных средств

7) Оформление пояснительный записки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	10	13.04.2020	<i>Вит</i>
2	План разработки и оценка бюджета	5	21.04.2020	<i>Вит</i>
3	Проектирование программных средств	30	25.04.2020	<i>Вит</i>
4	Реализация программных средств	25	16.05.2020	<i>Вит</i>
5	Тестирование и анализ качества программного кода	20	30.05.2020	<i>Вит</i>
6	Оформление пояснительный записки	10	06.06.2020	<i>Вит</i>

Дата выдачи задания 30.12.2019

Руководитель бакалаврской работы *Вит* 30.12.2020
(подпись, дата)

Задание к исполнению принял студент *Денис* 30.12.2020
(подпись, дата)

Работу к защите допустить
Декан факультета
(подпись, дата) _____

Реферат

Пояснительная записка содержит 86 листов, 23 рисунка, 6 таблиц, 15 использованных источников, 4 приложения.

АЛГОРИТМЫ, СТРУКТУРЫ ДАННЫХ, СПОРТИВНОЕ ПРОГРАММИРОВАНИЕ, ОЛИМПИАДЫ ПО ПРОГРАММИРОВАНИЮ, ДЕРЕВЬЯ, ГРАФЫ, СУФФИКСНЫЙ АВТОМАТ, C++.

Объект исследования – программы и алгоритмы решения олимпиадных задач спортивного программирования.

Цель выпускной квалификационной работы – разработка алгоритмов и программ решения олимпиадных задач спортивного программирования

Технология разработки – ООП.

Среда исполнения – Linux, Windows.

Разработка алгоритмов проводилась на языке программирования C++ в среде CLion в операционной системе Linux и Windows. Программа собиралась при помощи компилятора MinGW. Тестирование проводилось при помощи библиотеки модульного тестирования Catch.

					ПГУ 09.03.04 – 08БР161.11 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Разработка алгоритмов и программ решения олимпиадных задач спортивного программирования. Пояснительная записка.	Лит.	Лист	Листов
Разраб.	Дунаев В.Ю.	Дунаев	18.06.20					
Пров.	Балашова И.Ю.	И.Ю.	18.06.20				4	86
Н. контр.	Такташкин Д.В.					Группа 16ВП1		
Утверд.								

Содержание

Введение.....	7
1 Анализ предметной области	8
1.1 Алгоритмы решения задач спортивного программирования.....	8
1.2 Классификация алгоритмов решения задач спортивного программирования	9
1.3 Сравнительный анализ программных продуктов-аналогов	11
1.3.1 Стандартная библиотека шаблонов C++	11
1.3.2 Библиотека алгоритмов <code>std::algo</code>	13
1.3.3 Competitive Programming Library	13
1.4 Постановка задачи на разработку программных средств для решения задач спортивного программирования	14
1.5 Анализ требований к программным средствам	15
1.5.1 Анализ функциональных требований	15
1.5.2 Анализ нефункциональных требований	20
1.6 Технологии и средства разработки	20
2 Проектирование алгоритмов и программ решения олимпиадных задач спортивного программирования	21
2.1 Проектирование алгоритмов и структур данных для решения задач на подотрезках массива	21
2.2 Проектирование дерева отрезков	22
2.3 Проектирование алгоритмов для решения задач на графах	25
2.4 Проектирование суффиксного автомата	28
3 Реализация и тестирование алгоритмов и программ решения олимпиадных задач спортивного программирования.....	32
3.1 Реализация программных средств.....	32
3.2 Тестирование программных средств	34

	6
3.2.1 Функциональное тестирование.....	34
3.2.2 Структурное тестирование.....	37
3.2.3 Модульное тестирование	39
3.3 Анализ качества кода.....	40
3.4 Планирование разработки и оценка бюджета.....	43
Заключение	46
Список использованных источников	47
Приложение А Глоссарий	49
Приложение Б Листинг программы	52
Приложение В Результаты тестирования	73
Приложение Г Исходный код модульного тестирования.....	78

Введение

Спортивное (или олимпиадное) программирование – интеллектуальное соревнование по решению различных задач на ЭВМ, для решения которых необходимо придумать и применить какой-либо алгоритм или программу на одном из языков программирования [1].

Спортивное программирование – это перспективный интеллектуальный вид спорта. В настоящее время задачи по спортивному программированию набирают все большую популярность. Решение таких олимпиадных задач позволяет расширять знания по математике, алгоритмам, языкам программирования, а также совершенствоваться в написании эффективных программ в условиях ограниченного времени. Рост популярности олимпиадного программирования положительно влияет на другие сферы жизнедеятельности человека. Алгоритмы, которые применяются для решения задач олимпиадного программирования, применяются при разработке сложных программ [2].

Особенностью алгоритмов спортивного программирования является их эффективность. Для решения задачи необходимо не только разработать правильную программу, но и создать ее таким образом, чтобы время работы программы и используемая память укладывались в ограничения. Алгоритмы, которые были разработаны для решения олимпиадных задач, часто могут быть использованы в промышленном производстве при создании высоконагруженных систем.

Целью выпускной квалификационной работы является разработка алгоритмов и программ решения олимпиадных задач спортивного программирования. Разработка включает в себя анализ требования, этапы проектирования, реализации и тестирования программных средств.

Алгоритмы, разработанные в рамках выпускной квалификационной работы, можно будет применять для решения задач в различных онлайн соревнованиях или использовать при разработке программ для обработки больших объемов данных.

1 Анализ предметной области

1.1 Алгоритмы решения задач спортивного программирования

Алгоритм – конечная совокупность точно заданных правил решения произвольного класса задач или набор инструкций, описывающих порядок действий исполнителя для решения некоторой задачи [3].

Под спортивным или олимпиадным программированием, как правило, понимают интеллектуальное состязание, требующее от участников демонстрации знаний и навыков в области алгоритмизации и программирования.

На соревнованиях по спортивному программированию участникам предлагают решить ряд задач. Обычно задачи по спортивному программированию несут формальный характер: условия задач даны максимально кратко и понятно, есть точные ограничения на время работы программы, используемую память, входные и выходные данные. Решить задачу по спортивному программированию значит разработать и реализовать такой алгоритм, который будет выдавать верный результат при любых возможных входных данных, при этом не превышая ограничения по времени и памяти. Каждую задачу можно решить множеством различных путей. Многие решения являются простыми, но они не укладываются в ограничения по времени или по памяти. Основная сложность решения задач спортивного программирования заключается в разработке эффективного алгоритма.

Одним из самых важных показателей эффективности алгоритма является его сложность или асимптотика. Вычислительная сложность (асимптотика алгоритма) – понятие в информатике и теории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных [4]. Как правило, для успешного решения задачи программисту необходимо придумать алгоритм, асимптотика которого близка к наилучшей для этой задачи.

1.2 Классификация алгоритмов решения задач спортивного программирования

Задачи по спортивному программированию довольно разнообразны и для успешного их решения необходимо знать множество алгоритмов. В зависимости от темы задачи алгоритмы можно классифицировать следующим образом [4, 5]:

- алгоритмы теории чисел;
- графовые алгоритмы;
- алгоритмы вычислительной геометрии;
- алгоритмы обработки строк;
- алгоритмы теории игр;
- алгоритмы динамического программирования;
- алгоритмы для работы со структурами данных;
- комбинаторные алгоритмы.

Алгоритмы теории чисел основаны на дискретной математике. Такие алгоритмы необходимы для решения следующих задач: факторизация числа, проверка числа на простоту, вычисления наибольшего общего делителя, задачи на модульную арифметику и др.

Графовые алгоритмы основаны на теории графов. Такие алгоритмы позволяют решать задачи на нахождение кратчайших путей, поиск минимального остова графа, поиск мостов и точек сочленения. Отдельно в этой категории стоит отметить задачи на деревья и задачи для нахождения потоков.

Алгоритмы вычислительной геометрии необходимы для решения различных геометрических задач. Особенностью геометрических алгоритмов спортивного программирования является их дискретность: большинство задач требуют высокую точность вычислений и поэтому большинство расчетов необходимо проводить в целых числах. К алгоритмам вычислительной геометрии относят вычисление площади фигур, проверка фигур на пересечение или принадлежность, поиск центра тяжести фигуры и др.

Алгоритмы обработки строк, как правило, решают задачи, связанные с поиском подстрок в строке. Стоит отметить, что часто под строками в олимпиадных задачах понимают не только последовательности символов, но и последовательности любых объектов. Многие из алгоритмов обработки строк основаны на теории конечных автоматов и суффиксных структурах данных. Также, стоит отметить алгоритмы, основанные на хешировании, которые позволяют эффективно решать задачи с высокой вероятностью.

Алгоритмы теории игр необходимы для решения задач, в которых необходимо обыграть противника в математическую игру. Такие алгоритмы основаны на функции Шпрага-Гранди и методе ретроспективного анализа. К задачам теории игр относят задачу об игре Ним, игры на графах.

Алгоритмы динамического программирования позволяют решать оптимизационные задачи путем разбиения их на более простые подзадачи. Такие алгоритмы встречаются практически во всех соревнованиях спортивного программирования. Алгоритмы динамического программирования позволяют решать следующие задачи: задача о рюкзаке, задача о размене монет, задача коммивояжера и др.

Алгоритмы для работы со структурами данных позволяют создавать структуры данных и проводить над ними различные вычисления. Для решения некоторых задач спортивного программирования необходимо реализовать различные структуры данных. Некоторые из структур, такие как массивы, списки, красно-черные деревья, хеш-таблицы, куча – уже реализованы во многих языках программирования. Часто, функционала готовых структур данных недостаточно и для решения задачи необходимо реализовать собственную структуру данных. К наиболее популярным структурам данных, которые встречаются в олимпиадном программировании, относят дерево Фенвика, дерево отрезков, декартовое дерево, Link-Cut деревья.

Комбинаторные алгоритмы основаны на дискретной математике. Такие алгоритмы часто необходимы чтобы подсчитать количество различных решения той или иной задачи. Для этого используются различные комбинаторные формулы и методы, такие как число сочетаний и комбинаций,

принцип включений-исключений, известные числовые последовательность (например, числа Каталана, биномиальные коэффициенты, числа Фибоначчи).

Данную классификацию нельзя назвать полной. Многие задачи спортивного программирования требуют разработки нового алгоритма, который сложно отнести к какому-либо разделу классификации. Кроме того, на соревнованиях по олимпиадному программированию, достаточно редко встречаются задачи, решение которых полностью основано на известном алгоритме. Обычно, участникам необходимо разработать собственный алгоритм, на базе уже известных алгоритмов. Часто, требуется совместить алгоритмы из разных категорий, например, применить комбинаторный алгоритм в динамическом программировании или при помощи теории чисел свести задачу к графам.

1.3 Сравнительный анализ программных продуктов-аналогов

Алгоритмы решения задач спортивного программирования становятся все более популярными. Многие алгоритмы и структуры данных, которые были придуманы для решения задач олимпиадных задач, успешно используются в других областях разработки. В тоже время, спортивные программисты часто используют общеизвестные простые алгоритмы. Чтобы произвести сравнительный анализ продуктов-аналогов необходимо рассмотреть как алгоритмы, которые чаще всего встречаются в спортивном программировании, так и простые алгоритмы, которые реализованы в стандартных библиотеках языков программирования.

1.3.1 Стандартная библиотека шаблонов C++

Самым популярным языком программирования для решения олимпиадных задач является C++. Это связано с двумя факторами. Во-первых программы, созданные на C++, наиболее эффективны. Программы работают быстрее и потребляют меньше памяти. Во-вторых, C++ содержит мощную библиотеку алгоритмов и структур данных – стандартную библиотеку шаблонов.

Библиотека стандартных шаблонов (STL) (англ. Standard Template Library) – набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++ [6]. С точки зрения разработки алгоритмов для решения задач спортивного программирования, данная библиотека обладает следующими преимуществами и недостатками:

Преимущества:

- алгоритмы уже реализованы и для использования алгоритма необходимо только подключить соответствующий заголовочный файл;
- алгоритмы и структуры данных имеют единый интерфейс взаимодействия, что позволяет быстро изучить библиотеку;
- в библиотеке уже реализованы наиболее часто встречающиеся простые алгоритмы.
- библиотека разрабатывалась и продолжает разрабатываться большой группой профессионалов;
- библиотека содержит следующие структуры данных: динамический массив, односвязный список, двусвязный список, красно-черное дерево, хеш-таблицы, куча и др.

Недостатки:

- алгоритмов, представленных в библиотеке, недостаточно для решения задач спортивного программирования;
- нет возможности модификации алгоритмов;
- реализация алгоритмов скрыта;
- документация не предоставляет подробного описания сложных алгоритмов и структур данных;
- реализация алгоритмов зависит от компилятора языка и его версии.

Таким образом можно сделать вывод, что стандартная библиотека шаблонов C++ не содержит всех необходимых олимпиадных алгоритмов, но является отличной базой для разработки алгоритмов решения задач спортивного программирования.

1.3.2 Библиотека алгоритмов `emaxx algo`

`Emaxx algo` – это Web-ресурс на котором представлены алгоритмы решения задач спортивного программирования [5]. Данный ресурс является отечественное разработкой. Его создал Максим Иванов – студент Саратовского Государственного Университета, серебряный медалист финала ICPC 2011 года.

Преимущества:

- алгоритмы ориентированы на решение задач спортивного программирования;
- все алгоритмы классифицированы и распределены по разделам;
- алгоритмы содержат описание их работы, оценку временной сложности и математическое доказательство их корректности;
- реализация алгоритмов на C++.

Недостатки:

- алгоритмы представлены на Web-ресурсе и для их использования алгоритмов необходимо скопировать исходный код с сайта;
- ресурс давно не обновлялся и часть алгоритмов устарела;
- некоторые алгоритмы представлены не полностью, а лишь фрагментами кода.

`Emaxx algo` представляет из себя отличный ресурс для изучения и знакомства с алгоритмами решения задач спортивного программирования. Использовать `Emaxx algo` как готовую библиотеку олимпиадных алгоритмов нельзя: многие алгоритмы необходимо дорабатывать или переписывать.

1.3.3 Competitive Programming Library

Competitive Programming Library – библиотека алгоритмов решения задач спортивного программирования на C++ [7]. Библиотека разрабатывается спортивным программистом из Египта Ахмадом Эльзагиром.

Преимущества:

- алгоритмы собраны в виде библиотеки C++. Библиотеку можно подключить к проекту и использовать необходимые алгоритмы;

- библиотека находится в разработке и постоянно расширяется;
- структуры данных реализованы согласно парадигме ООП, что упрощает понимание код и программное взаимодействие;
- реализованные алгоритмы содержат описание интерфейсов взаимодействия с ними.

Недостатки:

- библиотека находится в разработке и содержит только небольшую часть популярных алгоритмов решения задач спортивного программирования;
- библиотека разрабатывается одним человеком и не покрыта тестами;
- нет оценки сложности алгоритмов и математического доказательства их корректности;
- интерфейс взаимодействия с алгоритмами не предоставляет возможности их модификации.

Можно сделать вывод, что библиотека Competitive Programming Library предоставляет часть алгоритмов решения задач спортивного программирования в удобной форме. Разработка нуждается в расширении.

1.4 Постановка задачи на разработку программных средств для решения задач спортивного программирования

В результате анализа предметной области было принято решение разработать библиотеку для решения олимпиадных задач спортивного программирования.

Для решения задач олимпиадного программирования применяется большое количество различных алгоритмов и структур данных. Реализовать все существующие алгоритмы невозможно. Поэтому было решено реализовать наиболее популярные, универсальные и часто-встречающиеся алгоритмы и структуры данных. В рамках данной работы будут реализованы следующие алгоритмы:

- алгоритмы для решения задач на подотрезках массива;
- алгоритмы для решения задач на графах;

- алгоритмы обработки строк.

Разрабатываемые алгоритмы должны быть эффективными и универсальными. Также, должен быть реализован удобный программный интерфейс взаимодействия с алгоритмами.

1.5 Анализ требований к программным средствам

Согласно поставленной задаче, в рамках данной работы необходимо разработать алгоритмы для решения задач на подотрезках массива, для решения задач на графах и для обработки строк.

Задачи на подотрезках массива заключаются в следующем: есть последовательность элементов $a_1, a_2, \dots, a_n$, необходимо большое количество раз находить значение

$$f(a_l, a_{l+1}, \dots, a_{r-1}, a_r), 1 \leq l \leq r \leq n,$$

где f – некоторая ассоциативная функция (например, сумма или максимум).

Для решения задач будут реализованы такие структуры данных как, дерево отрезков, разреженная таблица, дерево Фенвика, декартовое дерево.

Задачи на графах достаточно разнообразны и для их решения существует множество различных алгоритмов. В рамках данной работы будут реализованы наиболее популярные олимпиадные алгоритмы: поиск компонент связности, проверка графа на ацикличность, поиск мостов и точек сочленения, поиск кратчайших путей на графе.

Для реализации алгоритмов обработки строк будет использоваться суффиксный автомат. Данная структура данных позволяет решать практически любую задачу, связанную с обработкой строк. Кроме того, существует алгоритм построения суффиксного автомата с линейной вычислительной сложностью от длины строки.

1.5.1 Анализ функциональных требований

Согласно поставленным задачам, были выделены следующие функциональные требования:

- решить задачу на подотрезках массива;

- построить структуру данных;
- получить значение функции на отрезке массива;
- решить задачу на графах;
- построить граф;
- использовать необходимый алгоритм;
- решить задачу на обработку строк;
- построить суффиксный автомат;
- использовать суффиксный автомат.

На рисунке 1 представлена диаграмма вариантов использования.

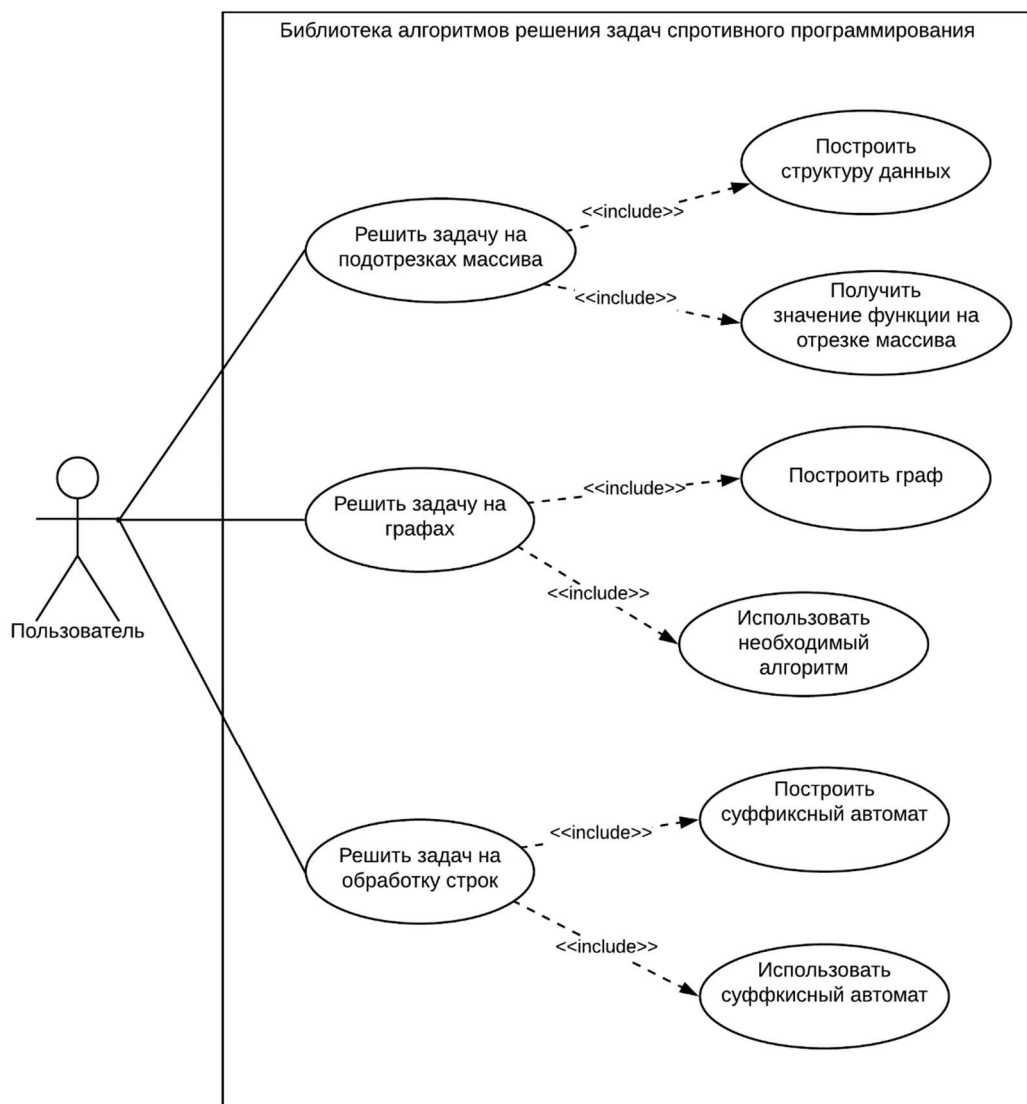


Рисунок 1 – Диаграмма вариантов использования

На диаграмме вариантов использования представлены функциональные требования к разрабатываемой системе. Спецификация прецедента «Решить задачу на подотрезках массива» представлена на рисунке 2.

Прецедент: Решить задачу на подотрезках массива
ID: 1
Краткое описание: Система многократно вычисляет значение ассоциативной функции на выбранных пользователем отрезках массива
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь выбирает подходящую структуру данных 2. include «Построить структуру данных» 3. Пользователь выбирает отрезки массива, на которых необходимо вычислить функцию 4. include «Получить значение функции на отрезке массива»
Постусловия: Нет
Альтернативные потоки: Нет

Рисунок 2 – Спецификация прецедента «Решить задачу на подотрезках массива»

На рисунке 3 представлена спецификация прецедента «Построить структуру данных».

Прецедент: Построить структуру данных
ID: 2
Краткое описание: Система строит структуру данных для решения задачи на подотрезках массива
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Выбрана структура данных
Основной поток: 1. Пользователь передает исходный массив 2. Система по заданному массиву формирует структуру данных 3. Система возвращает структуру данных
Постусловия: Структура данных передана пользователю
Альтернативные потоки: Нет

Рисунок 3 – Спецификация прецедента «Построить структуру данных»

На рисунке 4 представлена спецификация прецедента «Получить значение функции на отрезке массива».

Прецедент: Получить значение функции на отрезке массива
ID: 3
Краткое описание: Система вычисляет значение ассоциативной функции на заданном отрезке массиве с помощью построенной структуры данных
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Построена структура данных
Основной поток: 1. Пользователь передает левую и правую границу подотрезка массива 2. Система вычисляет значение функции в заданном отрезке массива 3. Система возвращает результат
Постусловия: Пользователь получил результат
Альтернативные потоки: Переданные границы запроса были некорректны

Рисунок 4 – Спецификация прецедента «Получить значение функции на отрезке массива»

На рисунке 5 представлена спецификация прецедента «Решить задачу на графах».

Прецедент: Решить задачу на графах
ID: 4
Краткое описание: Система выполняет необходимые вычисления для решения задачи на графах
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь определяет какой граф необходимо построить 2. include «Построить граф» 3. Пользователь выбирает алгоритм, который необходимо применить для решения задачи 4. include «Использовать необходимый алгоритм»
Постусловия: Нет
Альтернативные потоки: Нет

Рисунок 5 – Спецификация прецедента «Решить задачу на графах»

На рисунке 6 представлена спецификация прецедента «Решить задачу на обработку строк».

Прецедент: Решить задачу на обработку строк
ID: 5
Краткое описание: Система решает задачу, связанную со строками
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь получает строку, для которой нужно решить задачу 2. include «Построить суффиксный автомат» 3. Пользователь получает суффиксный автомат 4. include «Использовать суффиксный автомат»
Постусловия: Нет
Альтернативные потоки: Нет

Рисунок 6 – Спецификация прецедента «Решить задачу на обработку строк»

На рисунке 7 представлена спецификация прецедента «Построить суффиксный автомат».

Прецедент: Построить суффиксный автомат
ID: 6
Краткое описание: Система строит суффиксный автомат по заданной строке
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь передает исходную строку 2. Система по заданной строке строит суффиксный автомат 3. Система возвращает суффиксный автомат в виде класса
Постусловия: Суффиксный автомат передан пользователю
Альтернативные потоки: Нет

Рисунок 7 – Спецификация прецедента «Построить суффиксный автомат»

В результате анализа функциональных требований были выделены варианты использования программы. Для основных вариантов использования была приведена спецификация, которая позволяет понять последовательность действий, которые выполняет система при использовании функций пользователем [8].

1.5.2 Анализ нефункциональных требований

Согласно поставленным задачам, разрабатываемые алгоритмы должны быть максимально эффективными. Исходя из этого требования для разработки был выбран язык C++. Программы, созданные на компилируемом языке программирования C++, работают значительно быстрее программ на транслируемых языках (Java, C#) или скриптовых (Python). Кроме того, большинство задач спортивного программирования решаются на C++, а значит востребованность разрабатываемой библиотеки будет выше.

Разрабатываемая библиотека алгоритмов должна функционировать под управлением операционной системы Windows и Linux.

Создаваемые алгоритмы должны быть универсальными и легко модифицируемыми. Для этого будет использована парадигма ООП. Чтобы алгоритмы были универсальными необходимо использовать шаблоны C++.

1.6 Технологии и средства разработки

Для разработки алгоритмов и программ используется язык программирования C++ 17. Для сборки программы в операционной системе Linux (Ubuntu 18.04) используется компилятор GNU G++, а для сборки в ОС Windows 10 – MinGW. Такой выбор компиляторов позволит использовать программу в обеих операционных системах без изменения исходного кода.

В качестве среды разработки было решено использовать CLion, так как она включает в себя все средства для работы с выбранными технологиями. Данная среда разработки предоставляет возможности подсветки синтаксиса, анализа кода и автоматического рефакторинга для C++ 17 стандарта. Кроме того, CLion является кроссплатформенной средой разработки, которая работает, как под управлением операционной системы Linux, так и Windows.

2 Проектирование алгоритмов и программ решения олимпиадных задач спортивного программирования

2.1 Проектирование алгоритмов и структур данных для решения задач на подотрезках массива

Задачи на подотрезках массива являются очень популярными в спортивном программировании. Решение таких задач на небольших массивах не вызывает сложностей: достаточно вычислить значение некоторой функции от каждого элемента массива, входящего в подотрезок. Данная задача становится значительно более сложной, когда количество элементов в массиве велико и необходимо множество раз находить значение функции на подотрезке массива.

Для решения задач на подотрезках массива будут спроектированы следующие алгоритмы и структуры данных: дерево отрезков, декартовое дерево, дерево Фенвика, разреженная таблица, алгоритм корневой декомпозиции. Чтобы предоставить пользователю удобный интерфейс для решения задач на подотрезках массива была спроектирована следующая диаграмма классов (рисунок 8).

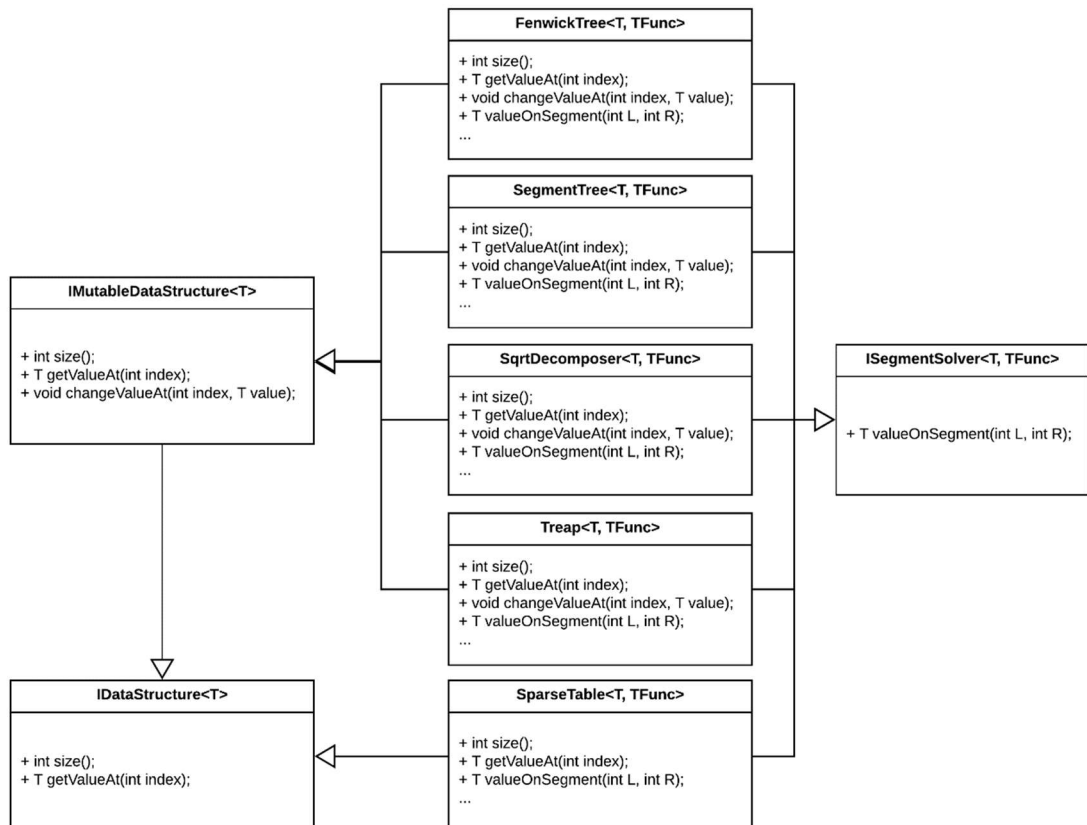


Рисунок 8 – Диаграмма классов для решения задач на подотрезках массива

Классы, представленные на диаграмме, соответствуют вышеописанным структурам данных. За счет наследования от интерфейсов, каждый класс имеет общие методы для нахождения функции на отрезке массива, но реализации этих методов будет различна.

2.2 Проектирование дерева отрезков

Дерево отрезков – это дерево, листья которого являются элементами исходного массива, остальные вершины хранят результат функции от своих двух потомков [9]. Дерево отрезков одна из наиболее эффективных структур для решения олимпиадных задач на отрезках массива. Дерево отрезков использует линейной количество памяти, находит значение функции на подотрезке массива за логарифмическое время и при этом имеет достаточно короткую и понятную реализацию. На рисунке 9 приведен пример дерева отрезков, построенном на массиве $[3, 4, 2, 6, 1]$ для функции минимума [9].

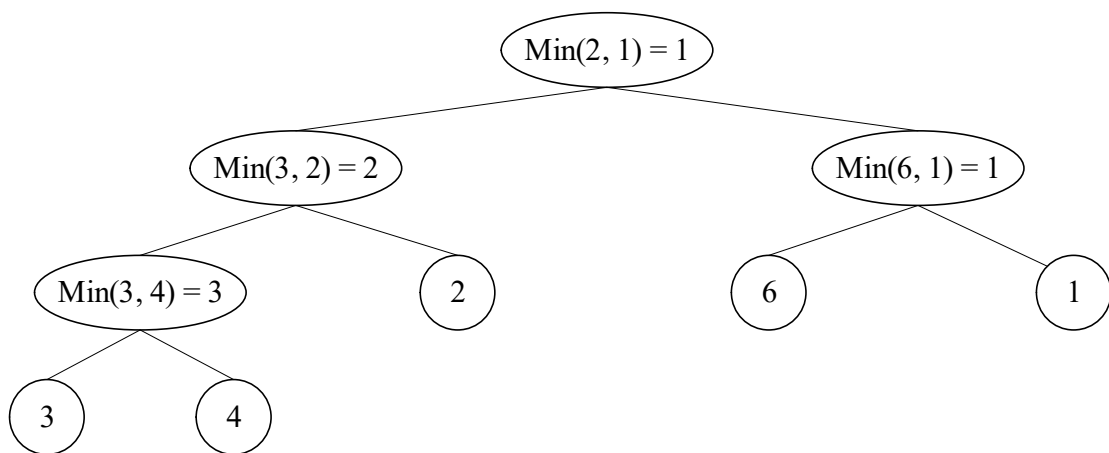


Рисунок 9 – Графическое представление дерева отрезков

Для построения дерева отрезков необходимо $O(n)$ памяти и времени, где n – размер исходного массива.

Для хранения структуры можно использовать одномерный массив размером $2 \cdot n - 1$. Элементы с индексами от n до $2 \cdot n - 1$ являются листьями дерева и равны данным из исходного массива. Остальные элементы дерева отрезков принимают значения функции своих сыновей. Узел дерева отрезков с номером i хранит значение ассоциативной функции f от своих детей с

номера $2 \cdot i$ и $2 \cdot i + 1$. Такой подход к описанию структуры позволит сэкономить память (не нужно хранить указатели на детей) и увеличить эффективность алгоритма (значительно уменьшается количество прыжков по оперативной памяти). Таким образом для построения дерева отрезков необходимо ровно один раз проинициализировать значение каждого узла. Количество узлов равно $2 \cdot n - 1$. Значит, алгоритм для построения дерева отрезков имеет линейную вычислительную сложность от размера массива. На рисунке 10 отображена диаграмма активности, на которой представлен алгоритм построения дерева отрезков.

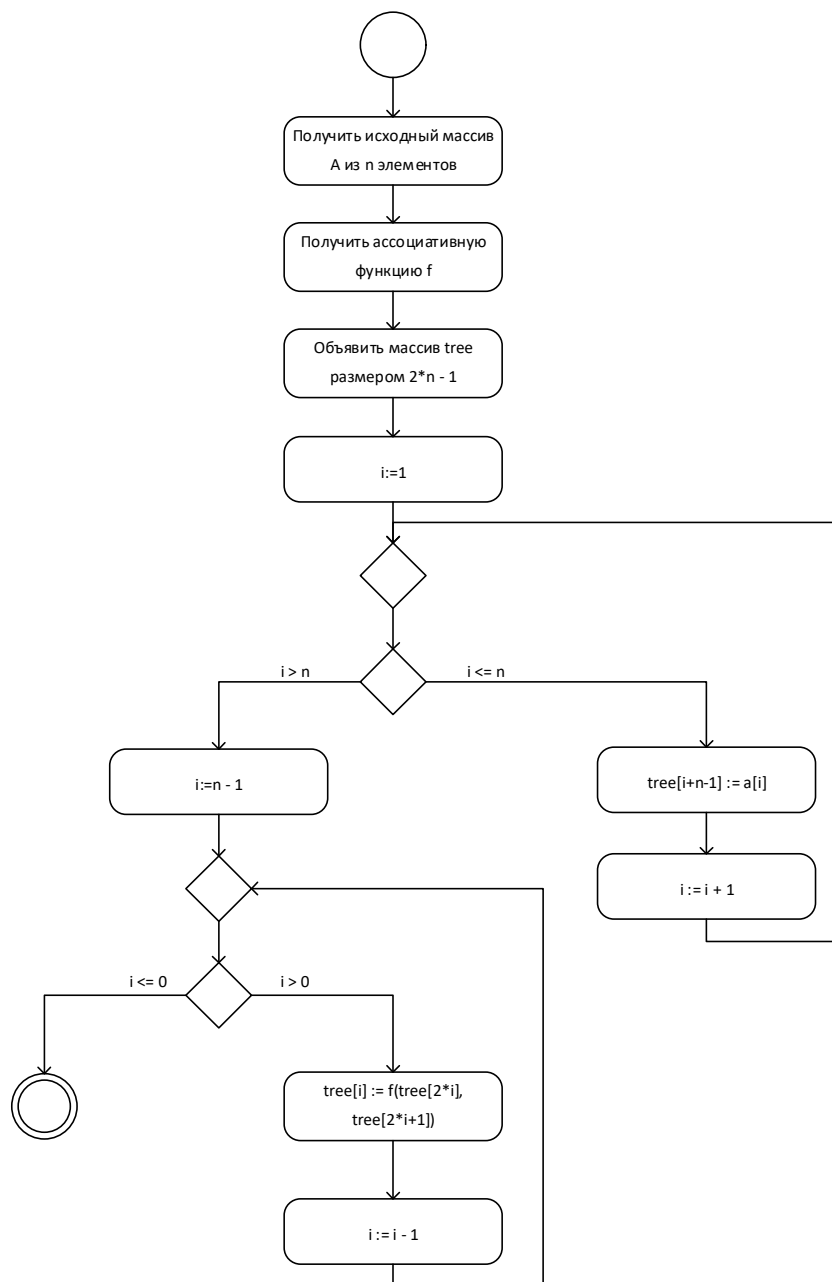


Рисунок 10 – Диаграмма активности для построения дерева отрезков

Дерево отрезков позволяет вычислять значение ассоциативной функции на любом отрезке массива за $O(\log(n))$, где n – размер исходного массива. Любой отрезок массива можно представить в виде некоторого числа узлов дерева отрезков. На рисунке 11 отображена диаграмма активности, на которой представлен алгоритм вычисления значения функции на отрезке.

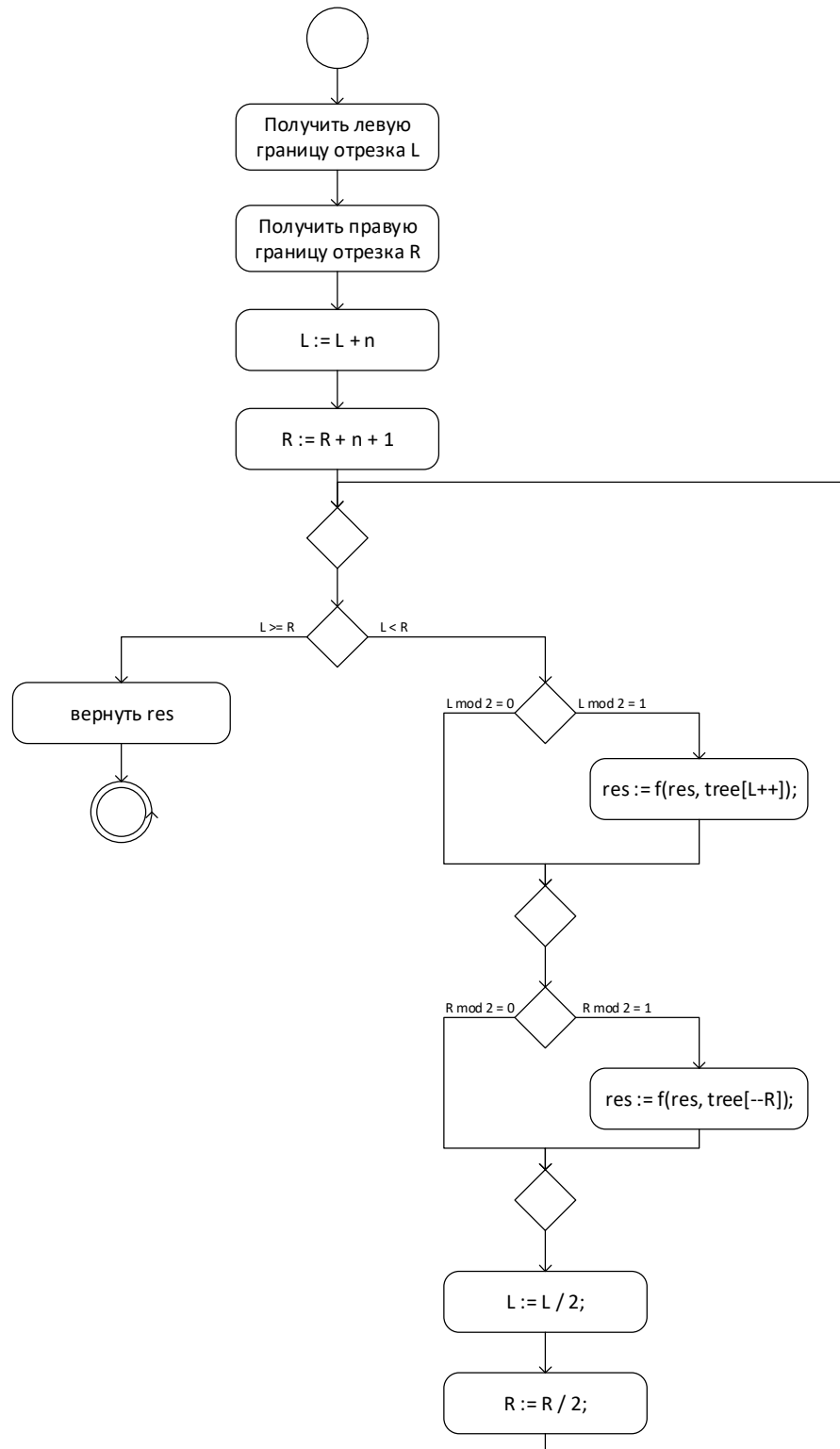


Рисунок 11 – Диаграмма активности для вычисления значения функции на подотрезке массива

Из диаграммы активности на рисунке 11 видно, что операция нахождения функции на отрезке массива циклична. На каждой итерации цикла происходит деление текущей левой и правой границы отрезка на 2. Из этого следует, что алгоритм имеет вычислительную сложность $O(\log(n))$.

Также, стоит отметить, что дерево отрезков позволяет реализовывать за логарифмическое время изменение элемента исходного массива. Для этого необходимо найти узел и пересчитать значения узлов на пути до корня.

2.3 Проектирование алгоритмов для решения задач на графах

Графовые задачи очень популярны в спортивном программировании. В некоторых задачах граф описан явным образом, в других необходимо преобразовать входные данные, чтобы получить граф.

Граф $G(V, E)$ – множество вершин V и множество пар вершин – ребер E [4]. Графы бывают ориентированные и неориентированные, в зависимости от того является ли пары вершин в ребрах упорядоченным или нет.

Также графы бывают взвешенными, если каждому ребру задан некоторый параметр w – вес ребра.

Создавать для каждого типа графа отдельные классы нецелесообразно. Невзвешенный граф можно представить, как взвешенный граф, в котором вес любого ребра равен 1. Неориентированный граф можно представить как ориентированный, в котором для каждого ребра (v, u) есть обратное ребро (u, v)

На рисунке 12 представлена диаграмма классов для решения олимпиадных задач на графах. Основным классом является класс `Graph`, в котором содержатся методы для формирования графа и работы с ним. Класс `IEdge` описывает интерфейс для ребер. Класс `Edge` представляет из себя невзвешенное ребро – при вызове метода `getWeight` всегда возвращается 1. Класс `WeightedEdge` представляет из себя взвешенное ребро. Класс является шаблонным, то есть пользователь может сам определить тип данных для веса. Такое решения является наиболее универсальным, и позволит не создавать

множество одинаковых классов для ребер, которые отличаются только типом хранимых данных.

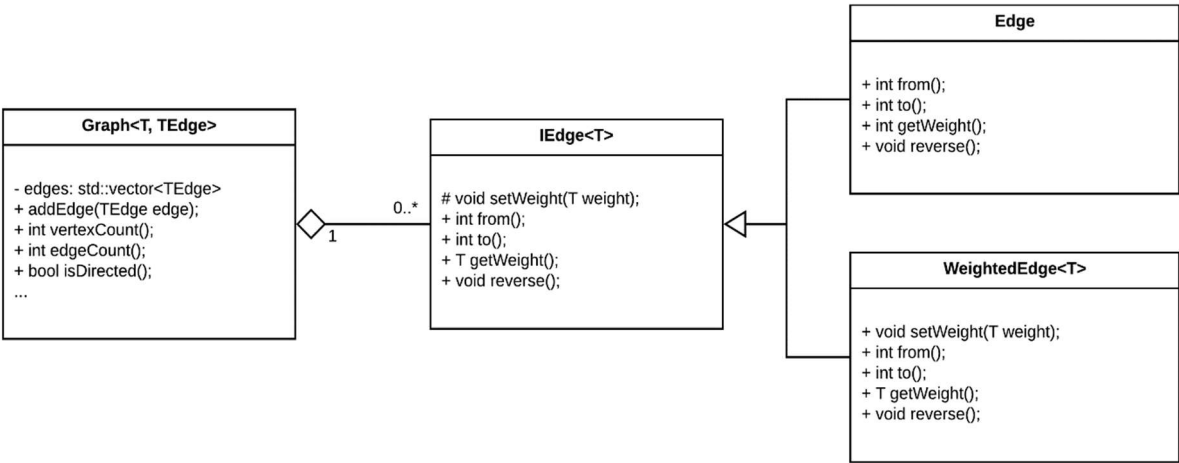


Рисунок 12 – Диаграмма классов для решения задач на графах

Класс Graph реализует все необходимые методы для построения и работы с графом (рисунок 13).

Graph<TEdge, TWeight>
<div>- m_edges: std::vector<TEdge> - m_incidentEdges: vector<std::vector<std::size_t>> - m_isDirected: bool = false</div>
<div>+ Graph(std::size_t t_vertexCount = 0, bool t_isDirected = false) + Graph(const Graph<TEdge, TWeight> &other) + Graph(Graph<TEdge, TWeight> &&other) + Graph<TEdge, TWeight> transposedGraph() + std::size_t vertexCount() + std::size_t edgeCount() + std::vector<TEdge> edges() + bool isDirected() + void addEdge(const TEdge &t_edge) + std::vector<std::vector<std::size_t>> connectivityComponents() + std::vector<std::size_t> topologicalSort() + bool isAcyclic() + std::vector<std::vector<std::size_t>> strongConnectivityComponents() + std::vector<TEdge> bridges() + std::vector<std::size_t> cutVertexes() + std::pair<std::vector<TWeight>, std::vector<std::size_t>> Dijkstra(std::size_t t_from) + std::pair<std::vector<TWeight>, std::vector<std::size_t>> FordBellman(std::size_t t_from, const TWeight &t_minusInf) - void topologicalSortDfs(std::size_t t_from, mark_list &t_was, std::vector<std::size_t> &t_order) - void findBridge(std::size_t t_from, int t_parentEdge, mark_list &was, std::vector<std::size_t> &t_tin, std::vector<std::size_t> &t_fup, std::size_t &mark_list &t_is_bridge) - void findCutVertexes(std::size_t t_from, int t_parentEdge, mark_list &was, std::vector<std::size_t> &t_tin, std::vector<std::size_t> &t_fup, std::size_t &timer, Graph::mark_list &isCutVertex) - bool checkCycleDfs(std::size_t t_from, int t_parentEdge, mark_list &t_colors)</div>

Рисунок 13 – Класс Graph

Атрибуты класса Graph:

- `m_edges` – список ребер;
- `m_incidentEdges` – список из списков номеров ребер инцидентных вершине;
- `m_isDirected` – булевское значение является ли граф ориентированным.

Методы класса Graph:

- `Graph(...)` – конструкторы класса;
- `transposedGraph` – возвращает транспонированный граф;
- `vertexCount` – возвращает количество вершин в графе;
- `edgeCount` – возвращает количество ребер в графе;
- `edges` – возвращает список ребер в графе;
- `isDirected` – возвращает является ли граф ориентированным;
- `addEdge` – добавляет новое ребро в граф;
- `connectivityComponents` – возвращает список компонент связности в графе, каждая компонента представлена в виде списка вершин;
- `topologicalSort` – возвращает список вершин в топологическом порядке;
- `isAcyclic` – возвращает булевское значение является ли граф ациклическим;
- `strongConnectivityComponents` – возвращает список компонент сильной связности, каждая компонента представлена в виде списка вершин;
- `bridges` – возвращает список ребер, которые являются мостами в графе;
- `cutVertex` – возвращает список вершин, которые являются точками сочленения графа;
- `Dijkstra` – возвращает пару значений из списка расстояний до каждой вершины из заданной и списка предшественников вершин в дереве кратчайших расстояний;

- `FordBellman` – возвращает пару значений из списка расстояний до каждой вершины из заданной и списка предшествующих вершин в дереве кратчайших расстояний;
- `topologicalSort` – рекурсивный поиск в глубину для топологической сортировки вершин;
- `findBridges` – рекурсивный поиск в глубину для нахождения мостов в графе;
- `findCutVertexes` – рекурсивный поиск в глубину для нахождения точек сочленения в графе;
- `checkCycleDfs` – рекурсивный поиск в глубину для проверки графа на ацикличность.

Методы класса `Graph` реализуют большинство алгоритмов для решения олимпиадных задач спортивного программирования. Класс предоставляет удобный программный интерфейс для его использования и позволит быстро и эффективно решать задачи на графах.

2.4 Проектирование суффиксного автомата

Одним из наиболее универсальных способов решения задач на обработку строк является суффиксный автомат.

Суффиксный автомат – это минимальный детерминированный конечный автомат, который принимает все суффиксы строки и только их. Суффикс строки – это строка, которая состоит из некоторого числа последних символов исходной строки.

Детерминированный конечный автомат – это набор из пяти элементов

$$\langle \Sigma, Q, s \in Q, T \subset Q, \delta: Q \times \Sigma \rightarrow Q \rangle,$$

где Σ – алфавит;

Q – множество состояний;

s – начальное (стартовое состояние);

T – множество допускающих состояний;

δ – функция переходов.

Минимальный детерминированный конечный автомат — это такой детерминированный конечный автомат, который содержит минимальное количество состояний среди всех возможных автоматов [10].

Суффиксный автомат — это мощная структура данных, которая позволяет решать множество строковых задач. Например, поиск строки в строке (проверка вхождения, позиция первого вхождения, позиции всех вхождений), количество различных подстрок исходной строки, лексикографическая k -ая подстрока, наименьший циклический сдвиг строки, поиск кратчайшей строки, не входящей в данную, наидлиннейшая общая строка двух подстроки и др.

Алгоритм построения суффиксного автомата имеет линейную вычислительную сложность относительно длины строки [10]. За построение автомата отвечает метод assign. Работа метода assign основана на итеративном добавлении символов в конец суффиксного автомата при помощи метода addCharacter. Алгоритм работы метода assign представлен на диаграмме активностей на рисунке 14, метода addCharacter на рисунке 15.

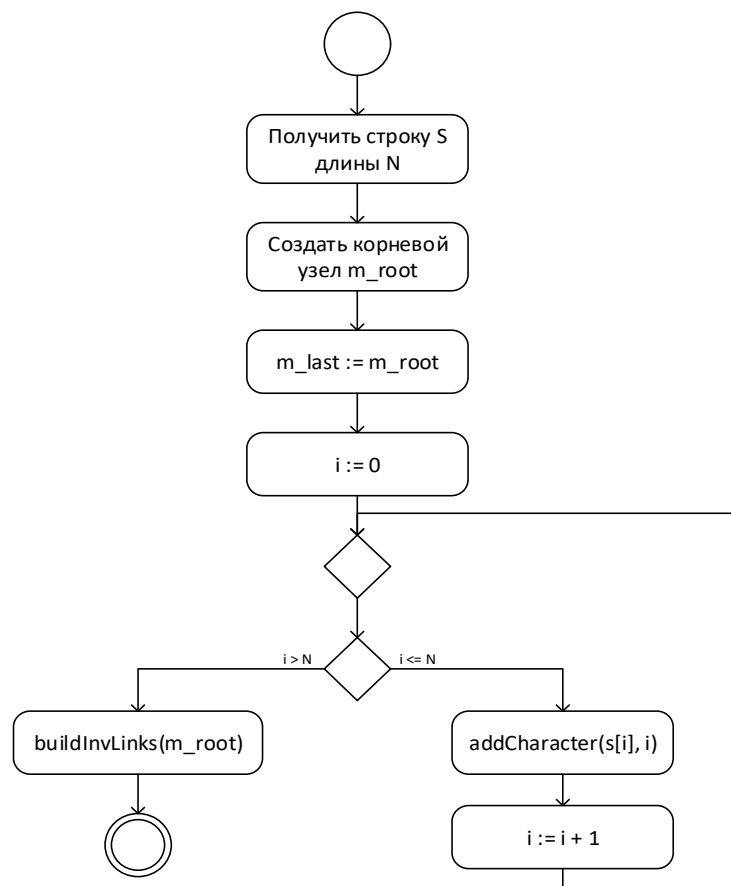


Рисунок 14 – Диаграмма активности алгоритма работы метода assign

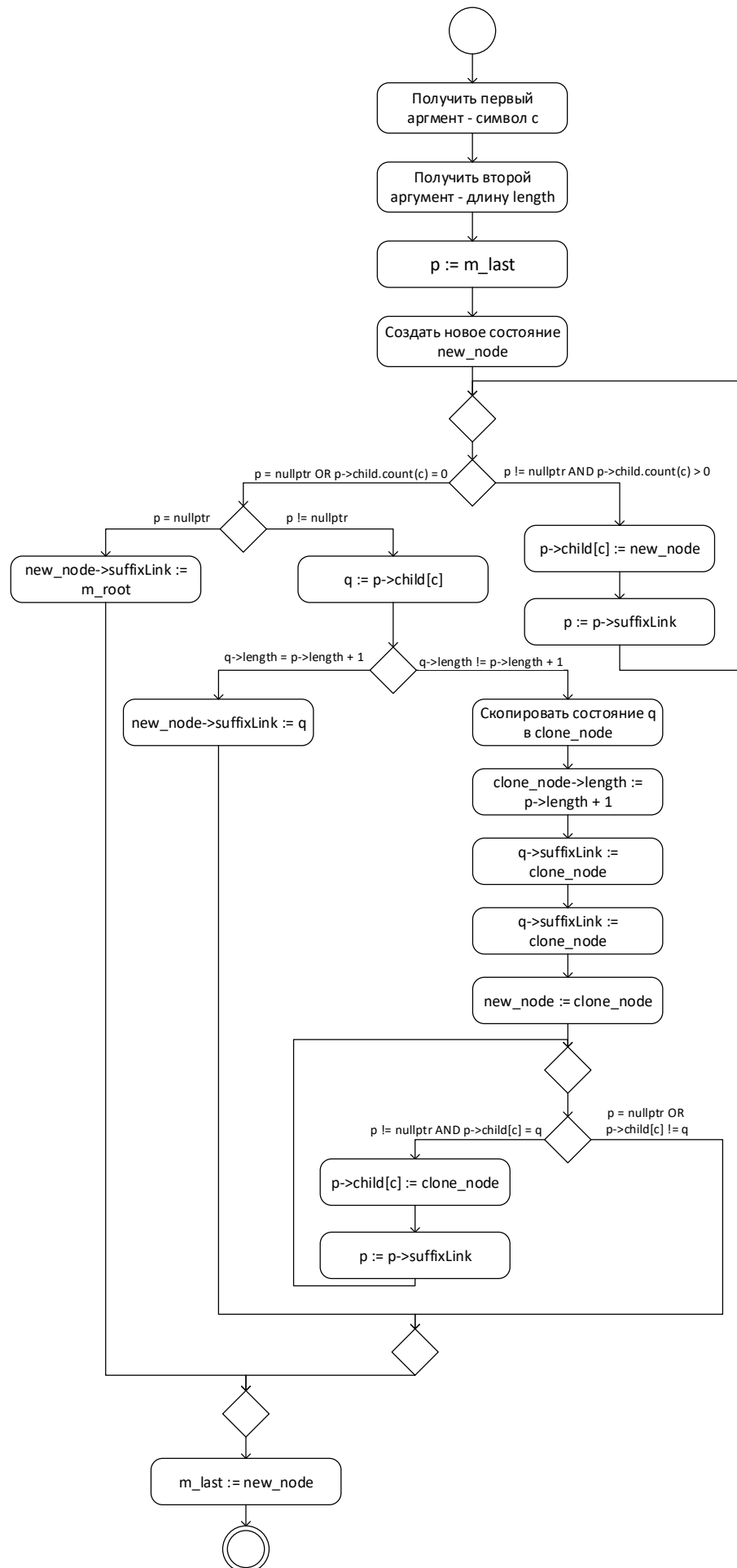


Рисунок 15 – Диаграмма активности алгоритма работы метода addCharacter

На рисунке 16 представлена диаграмма классов суффиксного автомата.

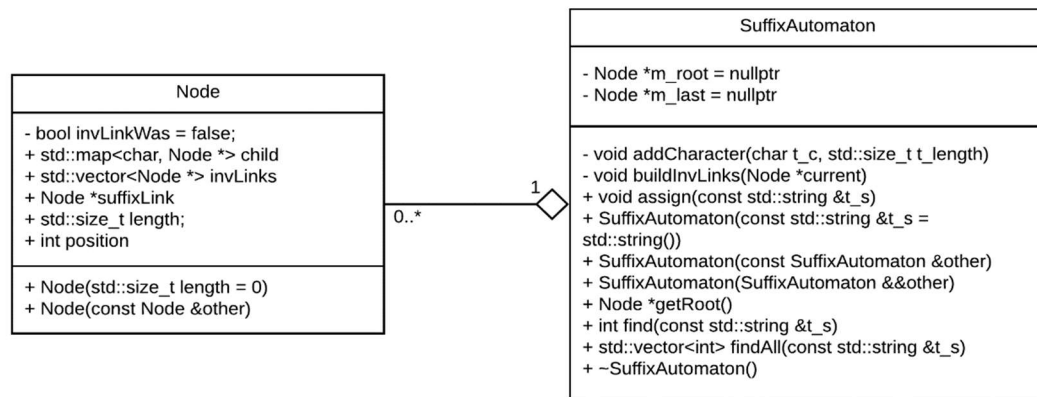


Рисунок 16 – Диаграмма классов суффиксного автомата

На диаграмме представлены два класса. Класс **Node** реализует состояние суффиксного автомата. Класс **SuffixAutomaton** реализует суффиксный автомат.

Атрибуты класса **SuffixAutomaton**:

- `m_root` – корневой узел суффиксного автомата, который является стартовым состоянием в ДКА;

- `m_last` – последний добавленный узел в суффиксный автомат.

Методы класса **SuffixAutomaton**:

- `addCharacter` – основной метод для построения суффиксного автомата, добавляет новый символ в суффиксный автомат;

- `buildInvLinks` – метод для построения обратных суффиксных ссылок, используется при построения суффиксного автомата;

- `assign` – метод для инициализации суффиксного автомата строкой;

- `SuffixAutomaton(...)` – конструкторы суффиксного автомата;

- `getRoot` – метод для получения начального состояния;

- `find` – метод для поиска первого вхождения строки в исходной;

- `findAll` – метод для поиска всех вхождений строки в исходную;

- `~SuffixAutomaton()` – деструктор класса.

Таким образом были спроектированы программные средства для работы с суффиксным автоматом. Полученные программные средства можно использовать для решения большинства задач связанных с обработкой строк.

3 Реализация и тестирование алгоритмов и программ решения олимпиадных задач спортивного программирования

3.1 Реализация программных средств

Алгоритмы и программы решения олимпиадных задач спортивного программирования были реализованы на языке C++ в среде разработки CLion. Исходный код программы представлен в приложении Б. Диаграмма компонентов программы представлена на рисунке 17.

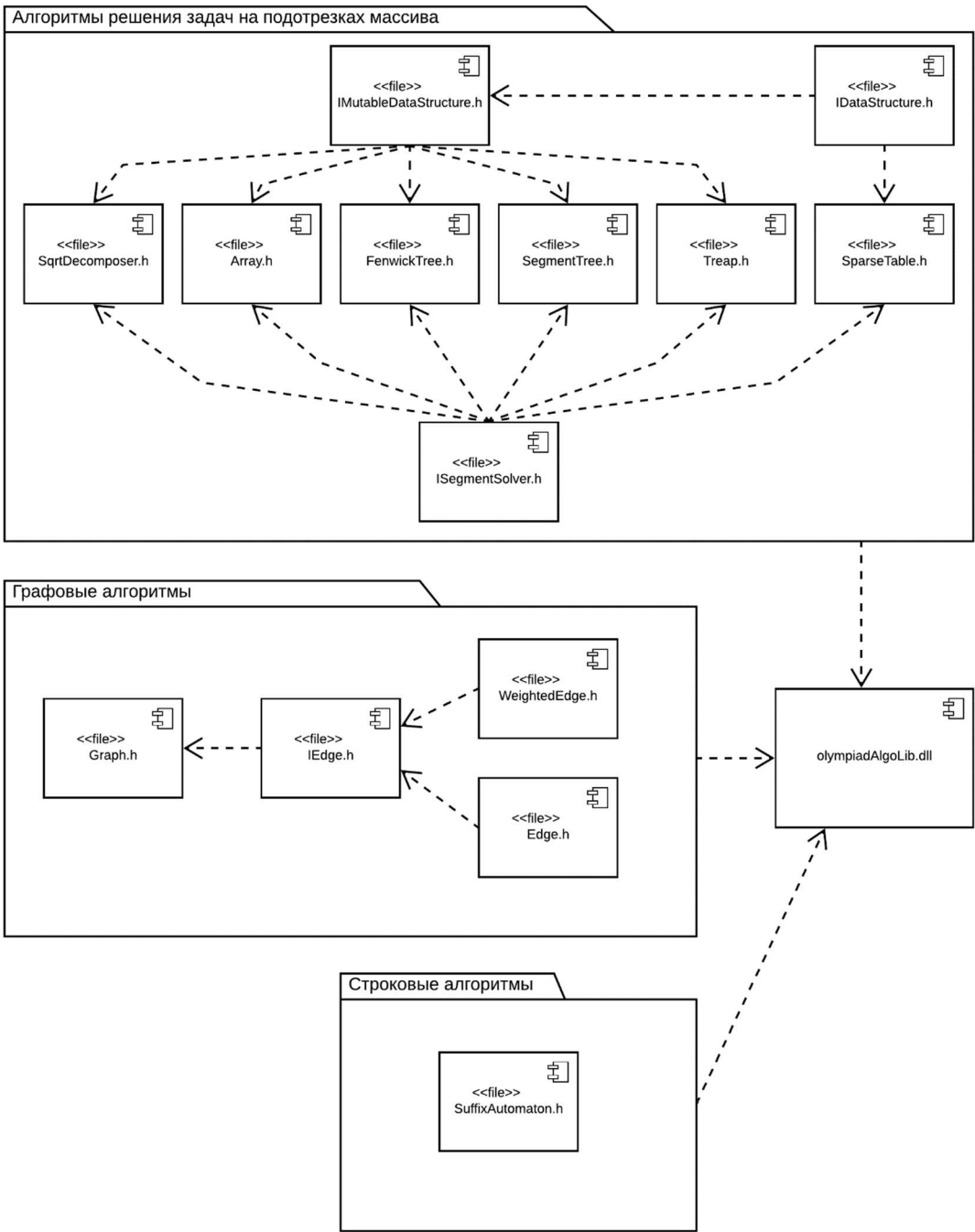


Рисунок 17 – Диаграмма компонентов

Описание компонентов проекта представлено в таблице 1.

Таблица 1 – Описание компонентов проекта

Название	Описание
olympiadAlgoLib.dll	Скомпилированный файл динамической библиотеки.
IDataStructure.h	Исходный код интерфейса структуры данных.
ImmutableDataStructure.h	Исходный код интерфейса изменяемой структуры данных.
ISegmentSolver.h	Исходный код интерфейса для решения задач на подотрезках массива.
SqrtDecomposer.h	Исходный код с реализацией корневой декомпозиции.
Array.h	Исходный код с реализацией массива.
FenwickTree.h	Исходный код с реализацией дерева Фенвика.
SegmentTree.h	Исходный код с реализацией дерева отрезков.
Treap.h	Исходный код с реализацией декартового дерева.
SparseTable.h	Исходный код с реализацией разреженной таблицы.
IEdge.h	Исходный код интерфейса ребра графа.
Edge.h	Исходный код с реализацией невзвешенного ребра в графе.
WeightedEdge.h	Исходный код с реализацией взвешенного ребра в графе.
Graph.h	Исходный код с реализацией графовых алгоритмов.
SuffixAutomaton.h	Исходный код суффиксного автомата.

На рисунке 18 представлена диаграмма развертывания.

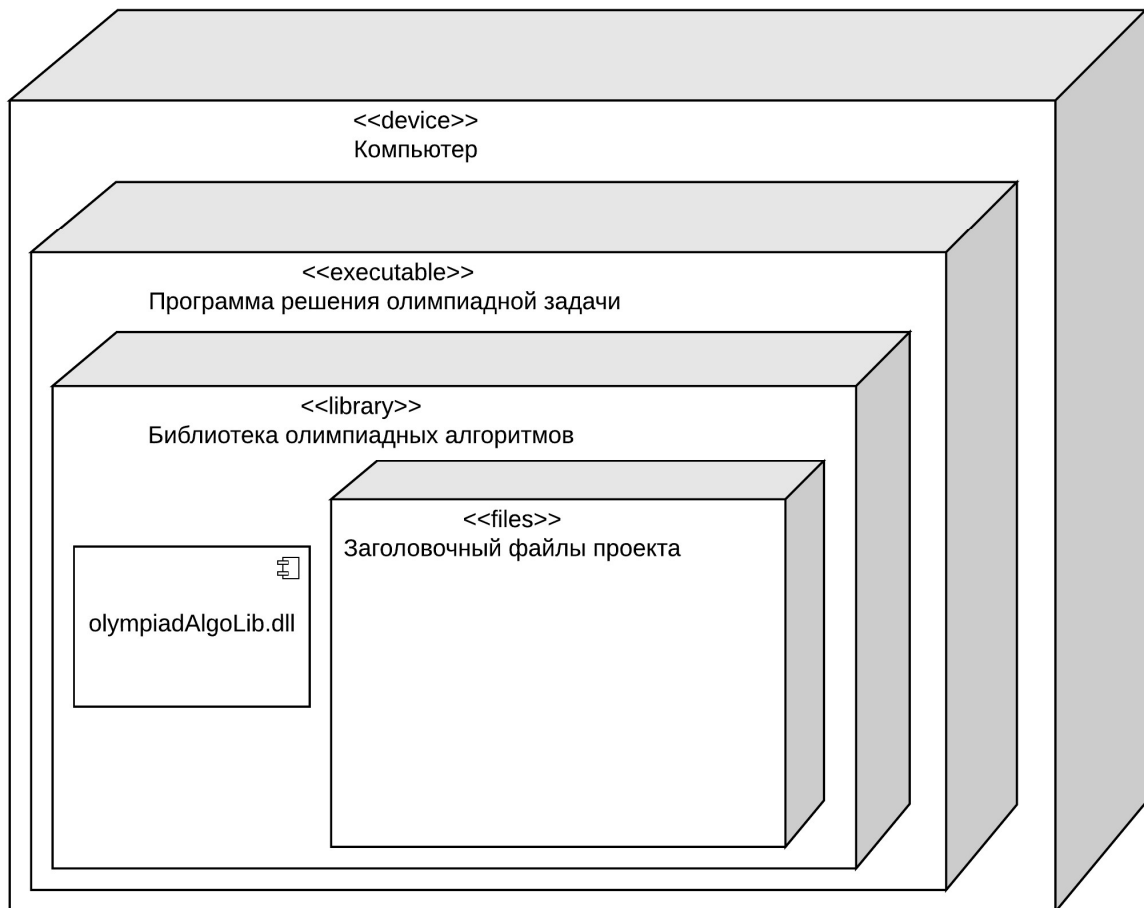


Рисунок 18 – Диаграмма развертывания

На диаграмме представлено развертывание программы с использованием реализованной библиотеки олимпиадных алгоритмов.

3.2 Тестирование программных средств

3.2.1 Функциональное тестирование

Для проверки корректности работы разработанного программного обеспечения будет использовано функциональное тестирование.

Для тестирования нахождения функции на отрезке массива было применен метод эквивалентного разбиения [11]. Отрезок массива задается двумя параметрами – левой и правой границей.

Для левой и правой границы отрезка были выделены следующие ограничения:

- границы отрезка меньше размера массива;

- границы отрезка больше нуля;
- левая граница не превосходит правую границу отрезка.

Для проведения тестирования методом эквивалентного разбиения были составлены классы эквивалентности (таблица 2).

Таблица 2 – Классы эквивалентности

Входные данные	Класс эквивалентности	Полученный результат
$0 \leq L \leq R < n$	Корректный подотрезок массива.	Получена функция на отрезке массива
$0 \leq R < L < n$	Правая граница отрезка меньше левой.	Получено программное исключение с сообщением о том, что правая не может быть меньше левой.
$L < 0$	Левая граница меньше нуля.	Получено программное исключение с сообщением о том, что одна из границ подотрезка не принадлежит массиву.
$L \geq n$	Левая граница больше или равна размеру массива.	Получено программное исключение с сообщением о том, что одна из границ подотрезка не принадлежит массиву.
$R < 0$	Правая граница меньше нуля.	Получено программное исключение с сообщением о том, что одна из границ подотрезка не принадлежит массиву.
$R \geq n$	Правая граница больше или равна размеру массива.	Получено программное исключение с сообщением о том, что одна из границ подотрезка не принадлежит массиву.

В таблице используются следующие обозначения: L – левая граница подотрезка, R – правая граница подотрезка, n – размер массива.

Для проверки правильности работы разработанных алгоритмов было проведено функциональное тестирование в соответствии с выделенными вариантами использования [11]. Результаты тестирования представлены в таблице 3.

Таблица 3 – Тестирование в соответствии с вариантами использования

Номер теста	Вариант использования	Описание теста	Результат тестирования
1	Решить задачу на подотрезках массива	Пользователь решает задачу: дан массив [4,3,7,0,8], необходимо найти минимум на всех возможных отрезках массива	Тест пройден. Минимум на всех отрезках массива найден верно (рисунок В.1)
2	Построить структуру данных	Пользователь строит дерево отрезков для массива [89,12,17,4,9]	Тест пройден. Дерево отрезков успешно построено (рисунок В.2)
3	Получить значение функции на отрезке массива	Пользователь находит максимум в массиве [89,12,17,4,9] на подотрезке с 2 по 4 элемент	Тест пройден. Максимум найден верно (рисунок В.2)
4	Решить задачу на графах	Пользователю необходимо проверить является ли ориентированный граф на 4 вершинах с ребрами {(1,2), (2,4), (1,3), (3,4)} ациклическим	Тест пройден. Программа верно определила, что граф является ациклическим (рисунок В.3)
5	Построить граф	Пользователь строит неориентированный граф на 5 вершинах с ребрами {(1,2), (2,3), (3,1), (3,5), (5,4), (4,3)}	Тест пройден. Граф успешно построен (рисунок В.4)
6	Использовать необходимый алгоритм	После построения графа пользователь выбирает алгоритм для поиска точек сочленения.	Тест пройден. Точки сочленения успешно найдены (рисунок В.4)
7	Решить задачу на обработку строк	Пользователь решает задачу: в строке «abacaba» необходимо найти позицию первого вхождения строки «bac»	Тест пройден. Позиция строки была найдена (рисунок В.5)
8	Построить суффиксный автомат	Пользователь строит суффиксный автомат для строки «ababb»	Тест пройден. Суффиксный автомат был построен (рисунок В.6)
9	Использовать суффиксный автомат	После построения суффиксного автомата пользователь находит позиции всех вхождений строки «ab»	Тест пройден. Все вхождения строки были успешно найдены (рисунок В.6)

Функциональное тестирование и тестирование методом разбиения границ показало правильность работы разработанных алгоритмов и программ решения олимпиадных задач спортивного программирования для всех выделенных вариантов использования.

3.2.2 Структурное тестирование

Структурное тестирование, за счет изучения логики работы программы, позволяет детально проверить правильность работы функции. Достаточно часто происходят случаи, когда функциональные тесты не покрывают полностью все варианты работы программы. Например, такое может произойти, когда функция принимает небольшое количество параметров, но при этом имеет сложную логику [11].

Для проверки корректности работы было решено структурно протестировать алгоритм нахождения функции на подотрезке массива при помощи дерева отрезков (листинг 1).

Листинг 1 – Исходный код алгоритма нахождения функции на подотрезке массива

```

1 t_left_index += this->m_size; //левая граница отрезка
2 t_right_index += this->m_size + 1; //правая граница отрезка
3 while ((t_left_index & 1u) == 0 && (t_right_index & 1u) == 0) {
4     t_left_index >>= 1u;
5     t_right_index >>= 1u;
6 }
7 auto odd_left = (t_left_index & 1u) == 1;
8 auto odd_right = (t_right_index & 1u) == 1;
9 auto result = odd_left ? this->m_nodes[t_left_index++] : this->m_nodes[--t_right_index];
10 if (odd_left && odd_right) {
11     result = this->m_function(result, this->m_nodes[--t_right_index]);
12 }
13 t_left_index >>= 1u;
14 t_right_index >>= 1u;
15 while (t_left_index < t_right_index) {
16     if ((t_left_index & 1u) == 1) {
17         result = this->m_function(result, this->m_nodes[t_left_index++]);
18     }
19     if ((t_right_index & 1u) == 1) {
20         result = this->m_function(result, this->m_nodes[--t_right_index]);
21     }
22     t_left_index >>= 1u;
23     t_right_index >>= 1u;
24 }
25 return result;

```

В листинге все строки кода пронумерованы. На основании этой нумерации были составлены маршруты для структурного тестирования (таблица 4).

Таблица 4 – Список маршрутов для структурного тестирования

Маршрут	Входные данные	Ожидаемый результат	Полученный результат
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25	Левая граница отрезка: 0 Правая граница отрезка: 7 Исходный массив: [2, 4, 1, 0, 6, 5, 9, 3, 8, 7] Дерево отрезков построено для нахождения максимума	Функция вернет 9, так как число 9 является максимумом на советующем отрезке	Функция вернула 9
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 25	Левая граница отрезка: 0 Правая граница отрезка: 3 Исходный массив: [4, 3, 2, 1, 6, 8] Дерево отрезков построено для нахождения максимума	Функция вернет 4, так как число 4 является максимумом на советующем отрезке	Функция вернула 4
1, 2, 3, 7, 8, 9, 10, 11, 12 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25	Левая граница отрезка: 1 Правая граница отрезка: 4 Исходный массив: [4, 3, 2, 1, 6, 8] Дерево отрезков построено для нахождения максимума	Функция вернет 6, так как число 6 является максимумом на советующем отрезке	Функция вернула 6
1, 2, 3, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25	Левая граница отрезка: 1 Правая граница отрезка: 3 Исходный массив: [4, 3, 2, 1, 6, 8] Дерево отрезков построено для нахождения максимума	Функция вернет 3, так как число 3 является максимумом на советующем отрезке	Функция вернула 3
1, 2, 3, 4, 5 7, 8, 9, 10, 11, 12 13, 14, 15, 25	Левая граница отрезка: 2 Правая граница отрезка: 3 Исходный массив: [4, 3, 2, 1, 6, 8] Дерево отрезков построено для нахождения максимума	Функция вернет 2, так как число 2 является максимумом на советующем отрезке	Функция вернула 2
1, 2, 3, 7, 8, 9, 10, 11, 12 13, 14, 15, 25	Левая граница отрезка: 3 Правая граница отрезка: 4 Исходный массив: [4, 3, 2, 1, 6, 8] Дерево отрезков построено для нахождения максимума	Функция вернет 6, так как число 6 является максимумом на советующем отрезке	Функция вернула 6

При помощи структурного тестирования были протестированы все возможные маршруты в алгоритме нахождения функции на подотрезке массива при помощи дерева отрезков. Все тесты были успешно пройдены, что говорит о том, что работа данного алгоритма полностью корректна.

3.2.3 Модульное тестирование

Модульное тестирование позволяет проверить корректность работы отдельных модулей программы. При помощи модульного тестирования можно в автоматическом режиме проверить корректно ли работают все функции программы, а также убедиться в том, что изменение кода не привело к ошибкам [11].

Для разработанной программы было решено провести модульное тестирование при помощи библиотеки Catch [12]. У данной библиотеки модульного тестирования есть следующие преимущества перед другими решениями:

- библиотека подключается путем добавления одного файла;
- подробная документация;
- тестирование производится на языке C++;
- открытый исходный код;
- свободная лицензия на использование;
- интеграция со средой разработки CLion.

Модульные тесты в Catch представлены в виде секций. Внутри секции тест выполняется независимо от остальных секций. Секции объединяются в тестовые случаи [12].

Исходный код модульного тестирования представлен в приложении Г. В таблице 5 представлено описание разработанных модульных тестов.

Таблица 5 – Описание модульных тестов

Название тестового случая	Описание	Модульные тесты
Directed graph testing	Тестовый случай для тестирования алгоритмов на ориентированном графе	Создание графа Добавление ребра в граф Выделение компонент связности Топологическая сортировка Проверка графа на ацикличность Выделение компонент сильной связности Поиск кратчайших путей алгоритмом Дейкстры Поиск кратчайших путей алгоритмом Форда-Беллмана

Продолжение таблицы 5

Название тестового случая	Описание	Модульные тесты
Undirected graph testing	Тестовый случай для тестирования алгоритмов на неориентированном графе	Создание графа Выделение компонент связности Проверка графа на ацикличность Поиск мостов Поиск точек сочленения
Shortest path in graph	Тестовый случай для тестирования алгоритмов нахождения кратчайших путей	Нахождение кратчайших путей в графе «бамбук» Нахождение кратчайших путей в графе «звезда» Нахождение кратчайших путей в ориентированном графе Нахождение кратчайших путей в неориентированном графе Нахождение кратчайших путей в взвешенном графе с ребрами отрицательного веса
Suffix automaton	Тестовый случай для тестирования суффиксного автомата	Создание суффиксного автомата Поиск вхождений в строку Проверка конструктора копирования Поиск вхождений в строку, состоящую из большого количества одинаковых символов
Range query	Тестовый случай для тестирования алгоритмов и структур данных для нахождения функций на подотрезке массива	Тестирование массива Тестирование разреженной таблицы Тестирование дерева отрезков Тестирование дерева Фенвика Тестирование корневой декомпозиции Тестирование декартового дерева Тестирование структур данных на случайных тестах без изменений

Всего было разработано 33 модульных теста. Все тесты были успешно пройдены (рисунок В.7). Таким образом, модульное тестирование показало корректность разработанного программного обеспечения.

3.3 Анализ качества кода

Для анализа качества кода был вычислен комплексный показатель качества кода. Этот показатель разработан специалистами из Carnegie Mellon Software Engineering Institute. Комплексный показатель качества кода может принимать значения от 0 до 100 и показывает относительную сложность поддержки кода. Чем больше значение этой метрики, тем легче поддерживать код [13].

Для вычисления комплексного показателя качества кода необходимо подсчитать количество операторов и операндов.

В таблице 6 представлены список операторов и операндов для алгоритма нахождения функции на подотрезке массива (смотри листинг 1).

Таблица 6 – Данные для расчета комплексного показателя качества кода

Операторы		Операнды	
Оператор	Количество	Операнд	Количество
+=	2	t_left_index	10
->	10	t_right_index	11
;	15	m_size	2
+	1	odd_left	3
while	2	odd_right	2
&	4	result	8
==	6	m_nodes	5
&&	2		
{}	5		
=	6		
? :	1		
[]	5		
++	2		
--	3		
if	3		
>>=	6		
<	1		
return	1		

Для вычисления комплексного показателя качества кода необходимо подсчитать следующие переменные:

- η_1 – число уникальных операторов;
- η_2 – число уникальных операндов;

- N_1 – общее число операторов;
- N_2 – общее число операндов;
- CC – структурная сложность программы;
- LoC – количество строк кода в программе.

Основываясь на таблице 6, были подсчитаны необходимые параметры:

- $\eta_1 = 18$;
- $\eta_2 = 7$;
- $N_1 = 75$;
- $N_2 = 41$.

Для вычисления словаря программы необходимо использовать следующую формулу:

$$\eta = \eta_1 + \eta_2.$$

Длина программы вычисляется при помощи данной формулы:

$$N = N_1 + N_2.$$

Таким образом, получаем что словарь программы $\eta = 25$, а длина программы $N = 116$. Используя данные значения, можно подсчитать вычислительную сложность программы при помощи следующей формулы:

$$V = N \cdot \log_2 \eta.$$

После подстановки словаря и длины программы получим, что вычислительная сложность программы $V = 538,69$.

Основываясь на листинге 1, были вычислены показатели:

- структурной сложности программы $CC = 5$;
- количества строк кода $LoC = 25$.

Для вычисления комплексного показателя сложности кода необходимо использовать следующую формулу [14]:

$$MI = \max(0, (171 - 5,2 \cdot \ln(V) - 0,23 \cdot CC - 16,2 \cdot \ln(LoC)) \cdot 100/171).$$

При помощи данной формулы был вычислен комплексный показатель сложности кода $MI = 49,7$. Показатель является достаточно высоким, что говорит о хорошем качестве кода. Понимание и поддержка разработанного алгоритма не должны вызывать проблем.

3.4 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета была выполнена при помощи программы для управления проектами Microsoft Project [14]. Проект был начат 20 апреля 2020 года. При планировании разработки было выделено 23 задачи. Список задач и их описание представлено на рисунке 19.

	1	Название задачи	Длительность	Начало	Окончание	Предш	Названия ресурсов
1		Начало проекта	0 дней	Пн 20.04.20	Пн 20.04.20		
2		Анализ требований и постановка задачи	4 дней	Пн 20.04.20	Чт 23.04.20	1	
3		Анализ предметной области	1 день	Пн 20.04.20	Пн 20.04.20		Аналитик[30%];Программист
4		Сравнение программ-аналогов	2 дней	Пн 20.04.20	Вт 21.04.20		Аналитик[70%]
5		Постановка задачи	1 день	Ср 22.04.20	Ср 22.04.20	3;4	Аналитик
6		Анализ функциональных требований	1 день	Чт 23.04.20	Чт 23.04.20	5	Аналитик;Программист
7		Анализ требования и поставка	0 дней	Чт 23.04.20	Чт 23.04.20	6	
8		Проектирование программных средств	6 дней	Пт 24.04.20	Пт 01.05.20	2	
9		Проектирование алгоритмов для решения задач на подотрезках массива	4 дней	Пт 24.04.20	Ср 29.04.20		Аналитик
10		Проектирование алгоритмов для решения задач на графах	4 дней	Пт 24.04.20	Ср 29.04.20		Программист
11		Проектирование алгоритмов для решения строковых задач	2 дней	Чт 30.04.20	Пт 01.05.20	9;10	Аналитик
12		Проектирование программных сред	0 дней	Пт 01.05.20	Пт 01.05.20	11	
13		Реализация	10 дней	Пн 04.05.20	Пт 15.05.20	8	
14		Реализация алгоритмов для решения задач на подотрезках массива	5 дней	Пн 04.05.20	Пт 08.05.20		Программист
15		Реализация алгоритмов для решения задач на графах	5 дней	Пн 11.05.20	Пт 15.05.20	14	Программист[80%]
16		Реализация алгоритмов для решения строковых задач	2 дней	Пн 11.05.20	Вт 12.05.20	14	Программист[20%]
17		Реализация завершена	0 дней	Пт 15.05.20	Пт 15.05.20	15;16	
18		Тестирование	4 дней	Пн 18.05.20	Чт 21.05.20	13	
19		Функциональное тестирование	2 дней	Пн 18.05.20	Вт 19.05.20		Тестировщик
20		Модульное тестирование	2 дней	Ср 20.05.20	Чт 21.05.20	19	Тестировщик
21		Тестирование завершено	0 дней	Чт 21.05.20	Чт 21.05.20	20	
22		Составление и оформление документации	3 дней	Пт 22.05.20	Вт 26.05.20	18	Аналитик;Программист
23		Конец проекта	0 дней	Вт 26.05.20	Вт 26.05.20	22	

Рисунок 19 – Список задач проекта

Для реализации проекта было выделено три трудовых ресурса: аналитик, программист и тестировщик. Описание ресурсов представлено на рисунке 20.

	1	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на исполн.	Начисление	Базовый календарь
1		Аналитик	Трудовой		А		100%	170,00 Р/ч	340,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
2		Программист	Трудовой		П		100%	160,00 Р/ч	320,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
3		Тестировщик	Трудовой		Т		100%	140,00 Р/ч	280,00 Р/ч	0,00 Р	Пропорциональное	Стандартный

Рисунок 20 – Список ресурсов проекта

В результате выделения задач и необходимых ресурсов проекта была построена диаграмма Ганта (рисунок 21). На диаграмме Ганта представлены задачи продолжительность и взаимосвязи между задачами в проекте. Также, на диаграмме отображены даты начала и конца каждой задачи и назначенные ресурсы для выполнения задачи.

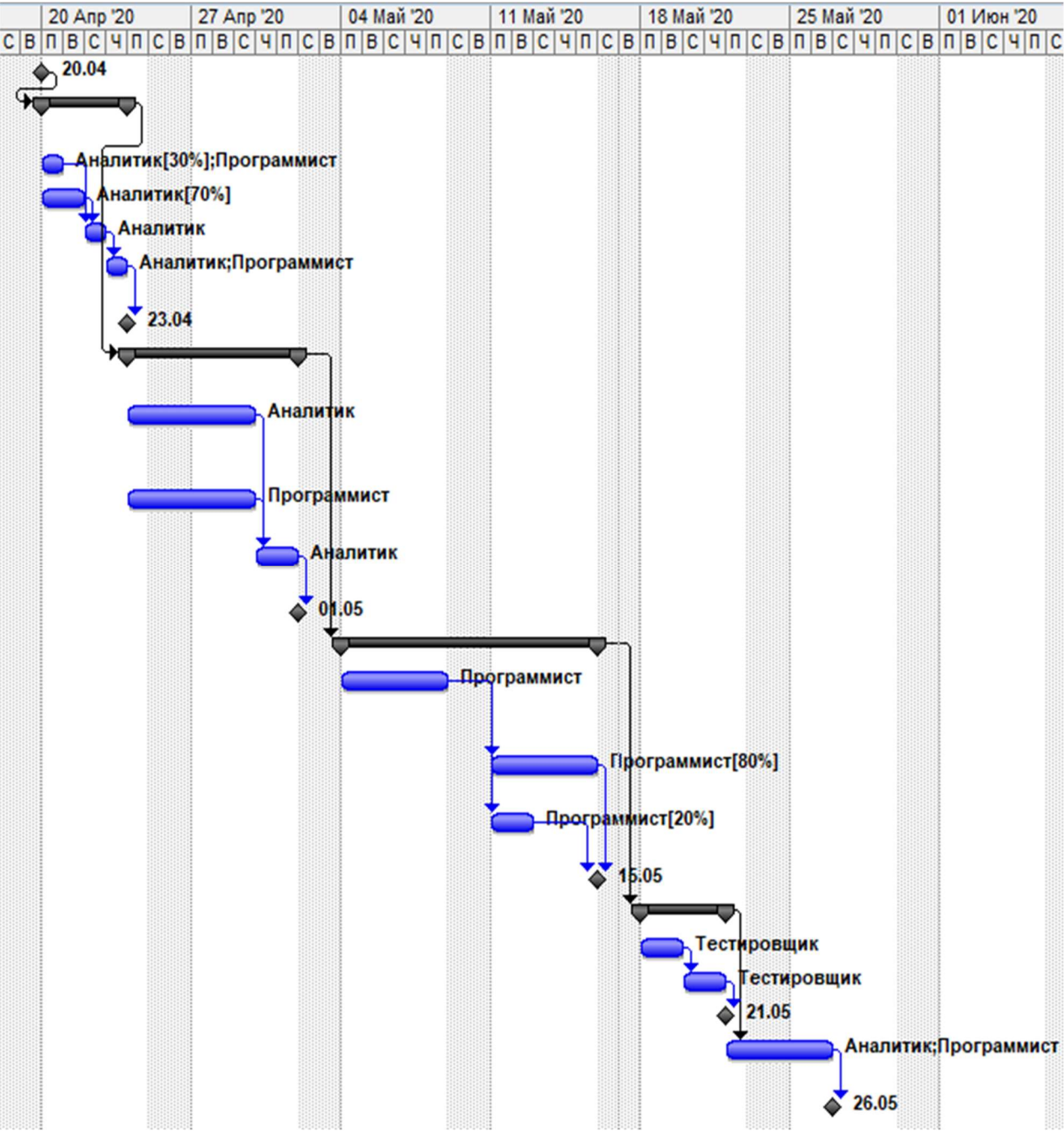


Рисунок 21 – Диаграмма Ганта

На рисунке 22, 23 представлены гистограмма загруженности ресурсов и гистограмма затрат на задачи соответственно.

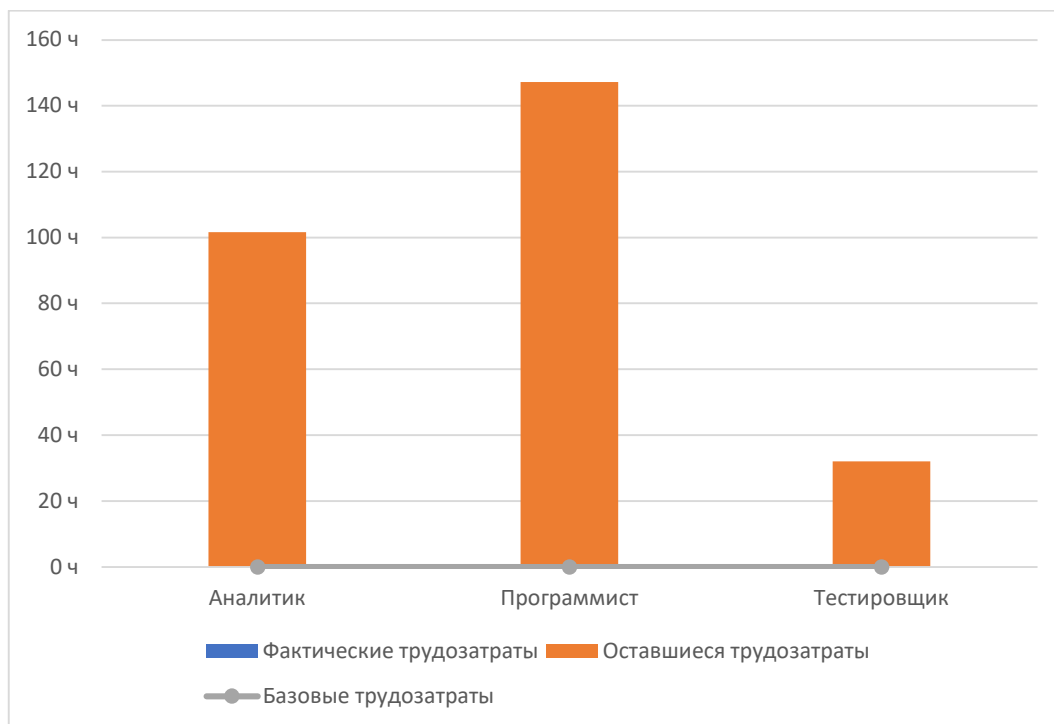


Рисунок 22 – Гистограмма загрузки ресурсов

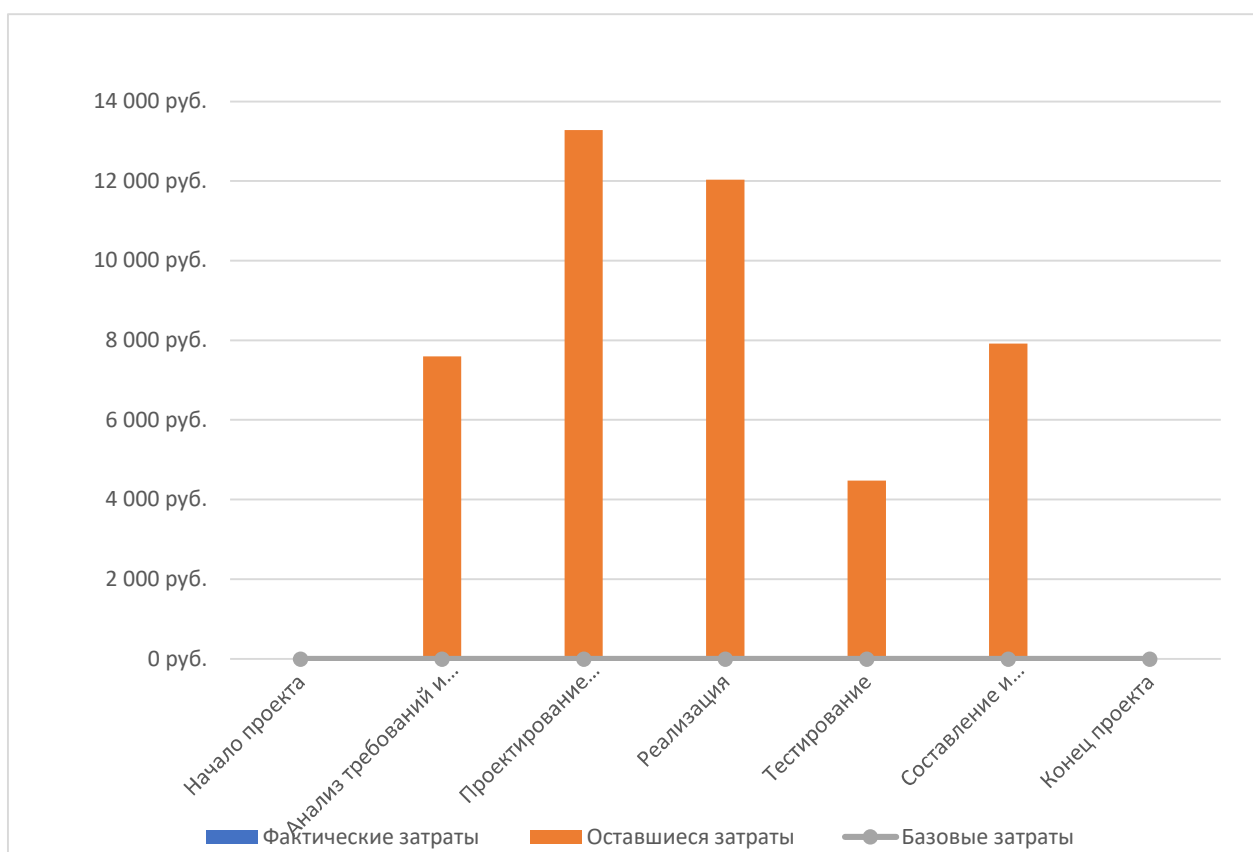


Рисунок 23 – Гистограмма затрат на задачи

Таким образом разработка проекта составит 27 полных рабочих дней с 20 апреля 2020 года по 26 мая 2020 года. Трудозатраты составят 280 человеко-часов. Оценка бюджета проекта составляет 45304 рубля.

Заключение

В ходе выполнения выпускной квалификационной работы разработана библиотека алгоритмом и программ решения олимпиадных задач спортивного программирования.

Проведен анализ алгоритмов решения задач спортивного программирования, рассмотрены задачи спортивного программирования и охарактеризованы алгоритмы их решения. Алгоритмы решения задач спортивного программирования классифицированы. На основе сравнения имеющихся аналогов и требований к заданию сформулирована задача на разработку программного средства, а также определены функциональные и нефункциональные требования к системе.

Для решения поставленной задачи спроектированы и реализованы алгоритмы и структуры данных для решения олимпиадных задач на подотрезках массива, на графах и задач на обработку строк. Необходимые структуры данных представлены в виде классов.

Разработка программных средств проводилась в среде разработки CLion на языке программирования C++. Разработанная программа функционирует, как под управлением операционной системы Linux (Ubuntu 18.04), так и под управлением Windows 10.

Разработанное программное обеспечение протестировано. Выполнено функциональное тестирование, структурное тестирование и модульное тестирование при помощи библиотеки Catch. Все тесты пройдены успешно и в результате тестирования не было выявлено ошибок.

В ходе выполнения выпускной квалификационной работы проведено планирование и оценка бюджета разработки при помощи программы для управления проектами Microsoft Project. Затраты на проект составили 45304 рубля. Время разработки составило 27 рабочих дней.

Таким образом в результате выполнения выпускной квалификационной работы разработаны алгоритмы и программы решения олимпиадных задач спортивного программирования, которые удовлетворяют всем требованиям технического задания.

Список использованных источников

1. Олимпиады по программированию // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Олимпиады_по_программированию (дата обращения: 05.06.2020).
2. Арлазаров В. Спортивное программирование: метод. пособие по подготовке к олимпиадам школьников. 7–11-й классы / Арлазаров. В, Мамай И. – М.: Изд. Дом МИСиС, 2017. – 28 с.
3. Алгоритм // Википедия [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Алгоритм> (дата обращения: 05.06.2020).
4. Лааксонен А. Олимпиадное программирование / Лааксонен А. – М.: ДМК Пресс, 2018. – 300 с.
5. Иванов. М. Библиотека алгоритмов спортивного программирования // Emaxx [Электронный ресурс]. URL: <https://e-maxx.ru/> (дата обращения: 05.06.2020).
6. Кройтер М. Основные понятия стандартной библиотеки C++ // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/post/434986/> (дата обращения: 05.06.2020).
7. Эльзагир А. Competitive Programming Library // Гитхаб [Электронный ресурс]. URL: <https://github.com/AhmadElsagheer/Competitive-programming-library> (дата обращения: 05.06.2020).
8. Арлоу Д. UML 2 и Унифицированный процесс, 2-е издание. Практический объектно-ориентированный анализ и проектирование. / Арлоу Д., Нейштадт А. – СПб: Символ Плюс, 2007. – 624 с.
9. Дунаев, В. Ю. Решение задач на отрезках массива с использованием дерева отрезков / В. Ю. Дунаев, Л. В. Гурьянов // Новые информационные технологии и системы: сб. науч. ст. XVI Междунар. науч.-техн. конф. (г. Пенза, 27 – 29 ноября 2019 г.). – Пенза: ПГУ, 2019. – с. 198 – 201.

10. Суффиксный автомат // Викиконспекты [Электронный ресурс]. URL: https://neerc.ifmo.ru/wiki/index.php?title=Суффиксный_автомат (дата обращения: 05.06.2020).
11. Майерс Г. Искусство тестирования программ / Майерс Г. – М.: Финансы и статистика, 1982. – 176 с.
12. Документация по библиотеке Catch // Гитхаб [Электронный ресурс]. URL: <https://github.com/catchorg/Catch2/blob/master/docs/tutorial.md> (дата обращения 05.06.2020).
13. Белоусов П. Средство оценки качества кода в MS Visual Studio // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/post/111524/> (дата обращения 05.06.2020).
14. Куперштейн В. Microsoft Project 2013 в управлении проектами. / Куперштейн В. – СПб.: БХВ-Петербург, 2014. – 432 с.

Приложение А
ГЛОССАРИЙ
(справочное)

Алгоритм – конечная совокупность точно заданных правил решения произвольного класса задач или набор инструкций, описывающих порядок действий исполнителя для решения некоторой задачи.

Ациклический граф – граф без циклов.

Вес ребра – значение, поставленное в соответствие данному ребру взвешенного графа.

Высота дерева – количество узлов на самом длинном пути от листа до корня.

Вычислительная сложность (асимптотика алгоритма) – понятие в информатике и теории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных.

Декартовое дерево – это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу.

Дерево – связанный ациклический граф.

Дерево отрезков – это дерево, листья которого являются элементами исходного массива, остальные вершины хранят результат функции от своих двух потомков.

Дерево Фенвика – двоичное индексированное дерево, позволяющее быстро изменять значения в массиве и находить некоторые функции от элементов массива.

ДКА – детерминированный конечный автомат.

Компонента связности графа – некоторое подмножество вершин графа, такое, что для любых двух вершин из этого множества существует путь из одной в другую, и не существует пути из вершины этого множества в вершину не из этого множества.

Компонента сильной связности графа, слой – максимальное множество вершин ориентированного графа, такое, что для любых двух вершин из этого множества существует путь как из первой во вторую, так и из второй в первую.

Корень дерева – узел дерева с нулевой степенью захода, т.е. такой узел, из которых не выходит ни одно ребро.

Корневая декомпозиция – структура данных, основанная на разбивании исходных данных на блоки размером квадратичного корня от размера входных данных.

Листья дерева – узлы дерева с нулевой степенью исхода, т.е. такие узлы, из которых не выходит ни одно ребро.

Мост в графе – ребро графа, удаление которого увеличит количество компонент связности в графе.

Подстрока – непустая связанная часть строки.

Разрезанная таблица – структура данных, позволяющая отвечать на запросы минимума на отрезке за константное время.

Суффикс строки – подстрока, оканчивающаяся на последний символ исходной строки.

Точка сочленения в графе – вершина графа, удаление которой увеличит количество компонент связности в графе.

Транспонированный граф – граф, полученный путем смены ориентации всех дуг графа.

Хеш-таблица – это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

C++ – компилируемый, статически типизированный язык программирования общего назначения.

CLion – интегрированная среда разработки на C++.

ICPC – международная студенческая олимпиада по программированию.

STL – библиотека стандартных шаблонов: набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.

Приложение Б
ЛИСТИНГ ПРОГРАММЫ
(рекомендуемое)

Листинг Б.1 – Файл Array.h

```
#ifndef COMPETITIVE_ALGORITHMS_ARRAY_H
#define COMPETITIVE_ALGORITHMS_ARRAY_H

#include <functional>
#include <vector>
#include "ImmutableDataStructure.h"
#include "ISegmentSolver.h"

namespace cpa {

    template<class TType, class TFunction = std::function<TType(const TType &, const
TType &>>>

        class Array :
            public ImmutableDataStructure<TType>,
            public ISegmentSolver<TType, TFunction> {
        private:
            std::vector<TType> m_data{};
        public:
            explicit Array(const TFunction &t_function = nullptr);

            template<class TIter>
            Array(TIter t_begin, TIter t_end, const TFunction &t_function = nullptr);

            Array(const Array<TType, TFunction> &other);

            Array(Array<TType, TFunction> &&other) noexcept;

            const TType &getValueAt(std::size_t t_index) const override;

            void changeValueAt(std::size_t t_index, const TType &t_value) override;

            TType valueOnSegment(std::size_t t_left_index, std::size_t t_right_index)
const override;
        };
    }

#endif //COMPETITIVE_ALGORITHMS_ARRAY_H
```

Листинг Б.2 – Файл Edge.h

```
#ifndef COMPETITIVE_ALGORITHMS_EDGE_H
#define COMPETITIVE_ALGORITHMS_EDGE_H

#include "IEdge.h"

namespace cpa {

    class Edge : public IEdge<int> {
    protected:
        void setWeight(const int &t_weight) override {};
    public:
        explicit Edge(std::size_t t_from = 0, std::size_t t_to = 0);

        Edge(const Edge &other);

        [[nodiscard]] int getWeight() const noexcept override;
    };

    Edge::Edge(std::size_t t_from, std::size_t t_to) {
        this->m_from = t_from;
    }
```

```

        this->m_to = t_to;
    }

    Edge::Edge(const Edge &other) : IEdge(other) {
        this->m_from = other.m_from;
        this->m_to = other.m_to;
    }

    int Edge::getWeight() const noexcept {
        return 1;
    }
}

#endif //COMPETITIVE_ALGORITHMS_EDGE_H

```

Листинг Б.3 – Файл FenwickTree.h

```

#ifndef COMPETITIVE_ALGORITHMS_FENWICKTREE_H
#define COMPETITIVE_ALGORITHMS_FENWICKTREE_H

#include "ImmutableDataStructure.h"
#include "ISegmentSolver.h"

namespace cpa {

    template<class TType, class TFunction = std::function<TType(const TType &, const
TType &>>>
    class FenwickTree :
        public ImmutableDataStructure<TType>,
        public ISegmentSolver<TType, TFunction> {
    private:
        std::vector<TType> m_tree;
        std::vector<TType> m_counter_tree;

        template<class TIter>
        void build(TIter t_begin, TIter t_end);

        [[nodiscard]] int f(unsigned int x) const noexcept;

        [[nodiscard]] int g(unsigned int x) const noexcept;

    public:
        const TType &getValueAt(std::size_t t_index) const override;

        void changeValueAt(std::size_t t_index, const TType &t_value) override;

        TType valueOnSegment(std::size_t t_left_index, std::size_t t_right_index)
const override;

        explicit FenwickTree(const TFunction &t_function = nullptr);

        template<class TIter>
        FenwickTree(TIter t_begin, TIter t_end, const TFunction &t_function =
nullptr);

        FenwickTree(const FenwickTree<TType, TFunction> &other);

        FenwickTree(FenwickTree<TType, TFunction> &&other) noexcept;

    };
}

```

```
#endif //COMPETITIVE_ALGORITHMS_FENWICKTREE_H
```

Листинг Б.4 – Файл Graph.h

```
#ifndef COMPETITIVE_ALGORITHMS_GRAPH_H
#define COMPETITIVE_ALGORITHMS_GRAPH_H

#include <cstdlib>
#include <vector>
#include <queue>
#include <set>
#include <numeric>
#include "IEdge.h"

namespace cpa {

    template<class TEdge, class TWeight = int>
    class Graph {
    private:
        typedef std::vector<std::int8_t> mark_list;
        std::vector<TEdge> m_edges;
        std::vector<std::vector<std::size_t>> m_incidentEdges;
        bool m_isDirected = false;

        void topologicalSortDfs(std::size_t t_from, mark_list &t_was,
            std::vector<std::size_t> &t_order) const;

        void findBridge(std::size_t t_from, int t_parentEdge, mark_list &was,
            std::vector<std::size_t> &t_tin,
                std::vector<std::size_t> &t_fup, std::size_t &timer,
                mark_list &t_is_bridge) const;

        void
            findCutVertexes(std::size_t t_from, int t_parentEdge, mark_list &was,
                std::vector<std::size_t> &t_tin,
                std::vector<std::size_t> &t_fup, std::size_t &timer,
                Graph::mark_list &isCutVertex) const;

        bool checkCycleDfs(std::size_t t_from, int t_parentEdge, mark_list &t_colors)
            const;

    public:
        explicit Graph(std::size_t t_vertexCount = 0, bool t_isDirected = false);

        Graph(const Graph<TEdge, TWeight> &other);

        Graph(Graph<TEdge, TWeight> &&other) noexcept;

        Graph<TEdge, TWeight> transposedGraph() const noexcept;

        [[nodiscard]] std::size_t vertexCount() const noexcept;

        [[nodiscard]] std::size_t edgeCount() const noexcept;

        std::vector<TEdge> edges() const;

        [[nodiscard]] bool isDirected() const noexcept;

        void addEdge(const TEdge &t_edge);

        [[nodiscard]] std::vector<std::vector<std::size_t>> connectivityComponents()
            const;

        [[nodiscard]] std::vector<std::size_t> topologicalSort() const;
```

```

[[nodiscard]] bool isAcyclic() const;

[[nodiscard]] std::vector<std::vector<std::size_t>>
strongConnectivityComponents() const;

[[nodiscard]] std::vector<TEdge> bridges() const;

[[nodiscard]] std::vector<std::size_t> cutVertexes() const;

std::pair<std::vector<TWeight>, std::vector<std::size_t>>
Dijkstra(std::size_t t_from) const;

std::pair<std::vector<TWeight>, std::vector<std::size_t>>
FordBellman(std::size_t t_from, const TWeight &t_minusInf) const;

};

template<class TEdge, class TWeight>
Graph<TEdge, TWeight>::Graph(std::size_t t_vertexCount, bool t_isDirected) {
    this->m_incidentEdges.assign(t_vertexCount, {});
    this->m_isDirected = t_isDirected;
}

template<class TEdge, class TWeight>
Graph<TEdge, TWeight>::Graph(const Graph<TEdge, TWeight> &other) {
    this->m_edges = other.m_edges;
    this->m_incidentEdges = other.m_incidentEdges;
}

template<class TEdge, class TWeight>
Graph<TEdge, TWeight>::Graph(Graph<TEdge, TWeight> &&other) noexcept {
    this->m_edges = std::move(other.m_edges);
    this->m_incidentEdges = std::move(other.m_incidentEdges);
}

template<class TEdge, class TWeight>
Graph<TEdge, TWeight> Graph<TEdge, TWeight>::transposedGraph() const noexcept {
    Graph<TEdge, TWeight> transposedGraph(*this);
    transposedGraph.m_incidentEdges.assign(transposedGraph.vertexCount(), {});
    for (auto i = 0u; i < transposedGraph.m_edges.size(); ++i) {
        auto &e = transposedGraph.m_edges[i];
        e.reverse();
        transposedGraph.m_incidentEdges[e.from()].push_back(i);
    }
    return transposedGraph;
}

template<class TEdge, class TWeight>
std::size_t Graph<TEdge, TWeight>::vertexCount() const noexcept {
    return m_incidentEdges.size();
}

template<class TEdge, class TWeight>
std::size_t Graph<TEdge, TWeight>::edgeCount() const noexcept {
    if (m_isDirected) {
        return m_edges.size();
    }
    return m_edges.size() / 2;
}

template<class TEdge, class TWeight>
bool Graph<TEdge, TWeight>::isDirected() const noexcept {

```

```

        return this->m_isDirected;
    }

template<class TEdge, class TWeight>
void Graph<TEdge, TWeight>::addEdge(const TEdge &t_edge) {
    this->m_incidentEdges[t_edge.from()].push_back(this->m_edges.size());
    this->m_edges.push_back(t_edge);
    if (!m_isDirected) {
        auto rEdge(t_edge);
        rEdge.reverse();
        this->m_incidentEdges[rEdge.from()].push_back(this->m_edges.size());
        this->m_edges.push_back(rEdge);
    }
}

template<class TEdge, class TWeight>
std::vector<std::vector<std::size_t>> Graph<TEdge,
TWeight>::connectivityComponents() const {
    if (this->m_isDirected) {
        Graph<TEdge, TWeight> g(this->vertexCount(), false);
        for (auto &e : this->edges()) {
            g.addEdge(e);
        }
        return g.connectivityComponents();
    }
    std::vector<std::vector<std::size_t>> comp;
    mark_list was(this->vertexCount());
    for (std::size_t startV = 0; startV < was.size(); ++startV) {
        if (!was[startV]) {
            std::queue<std::size_t> q;
            q.push(startV);
            was[startV] = true;
            comp.emplace_back();
            while (!q.empty()) {
                auto v = q.front();
                q.pop();
                comp.back().push_back(v);
                for (auto &i : this->m_incidentEdges[v]) {
                    auto u = this->m_edges[i].to();
                    if (!was[u]) {
                        q.push(u);
                        was[u] = true;
                    }
                }
            }
        }
    }
    return comp;
}

template<class TEdge, class TWeight>
std::vector<std::size_t> Graph<TEdge, TWeight>::topologicalSort() const {
    std::vector<std::size_t> order;
    mark_list was(this->vertexCount());
    for (std::size_t v = 0; v < was.size(); ++v) {
        if (!was[v]) {
            topologicalSortDfs(v, was, order);
        }
    }
    std::reverse(order.begin(), order.end());
    return order;
}

```

```

template<class TEdge, class TWeight>
void
Graph<TEdge, TWeight>::topologicalSortDfs(std::size_t t_from, mark_list &t_was,
                                         std::vector<std::size_t> &t_order)
const {
    t_was[t_from] = true;
    for (auto &i : this->m_incidentEdges[t_from]) {
        auto to = this->m_edges[i].to();
        if (!t_was[to]) {
            topologicalSortDfs(to, t_was, t_order);
        }
    }
    t_order.push_back(t_from);
}

template<class TEdge, class TWeight>
std::vector<std::vector<std::size_t>> Graph<TEdge,
TWeight>::strongConnectivityComponents() const {
    auto rGraph = this->transposedGraph();
    std::vector<std::vector<std::size_t>> comps;
    mark_list was(this->vertexCount());
    for (auto v : this->topologicalSort()) {
        if (!was[v]) {
            comps.emplace_back();
            rGraph.topologicalSortDfs(v, was, comps.back());
        }
    }
    return comps;
}

template<class TEdge, class TWeight>
std::vector<TEdge> Graph<TEdge, TWeight>::bridges() const {
    mark_list was(this->vertexCount());
    std::vector<std::size_t> tin(this->vertexCount()), fup(this->vertexCount());
    mark_list is_bridge(this->edgeCount() * 2);
    std::size_t timer = 0;
    for (std::size_t v = 0; v < was.size(); ++v) {
        if (!was[v]) {
            findBridge(v, -1, was, tin, fup, timer, is_bridge);
        }
    }

    std::vector<TEdge> bridgesList;
    for (std::size_t i = 0; i < is_bridge.size(); ++i) {
        if (is_bridge[i]) {
            bridgesList.push_back(this->m_edges[i]);
        }
    }
    return bridgesList;
}

template<class TEdge, class TWeight>
void Graph<TEdge, TWeight>::findBridge(std::size_t t_from, int t_parentEdge,
Graph::mark_list &was,
                                         std::vector<std::size_t> &t_tin,
std::vector<std::size_t> &t_fup,
                                         size_t &timer,
                                         mark_list &t_is_bridge) const {
    was[t_from] = true;
    t_tin[t_from] = t_fup[t_from] = timer++;
    for (auto &i : this->m_incidentEdges[t_from]) {
        if ((i ^ 1u) == t_parentEdge) {

```

```

        continue;
    }
    auto to = m_edges[i].to();
    if (was[to]) {
        t_fup[t_from] = std::min(t_fup[t_from], t_tin[to]);
    } else {
        findBridge(to, i, was, t_tin, t_fup, timer, t_is_bridge);
        t_fup[t_from] = std::min(t_fup[t_from], t_fup[to]);
        if (t_fup[to] > t_tin[t_from]) {
            t_is_bridge[i / 2 * 2] = true;
        }
    }
}
}
}

```

```

template<class TEdge, class TWeight>
std::vector<std::size_t> Graph<TEdge, TWeight>::cutVertexes() const {
    mark_list was(this->vertexCount());
    std::vector<std::size_t> tin(this->vertexCount()), fup(this->vertexCount());
    std::size_t timer = 0;
    mark_list isCutVertex(this->vertexCount());
    for (std::size_t v = 0; v < was.size(); ++v) {
        if (!was[v]) {
            findCutVertexes(v, -1, was, tin, fup, timer, isCutVertex);
        }
    }
    std::vector<std::size_t> cutVertexesList;
    for (std::size_t v = 0; v < isCutVertex.size(); ++v) {
        if (isCutVertex[v]) {
            cutVertexesList.push_back(v);
        }
    }
    return cutVertexesList;
}

```

```

template<class TEdge, class TWeight>
void Graph<TEdge, TWeight>::findCutVertexes(std::size_t t_from, int t_parentEdge,
Graph::mark_list &was,
std::vector<std::size_t> &t_tin,
std::vector<std::size_t> &t_fup,
size_t &timer, Graph::mark_list
&isCutVertex) const {
    was[t_from] = true;
    t_tin[t_from] = t_fup[t_from] = timer++;
    std::size_t dfsCount = 0;
    for (auto &i : this->m_incidentEdges[t_from]) {
        if (i == t_parentEdge) {
            continue;
        }
        auto to = this->m_edges[i].to();
        if (was[to]) {
            t_fup[t_from] = std::min(t_fup[t_from], t_tin[to]);
        } else {
            dfsCount++;
            findCutVertexes(to, i, was, t_tin, t_fup, timer, isCutVertex);
            t_fup[t_from] = std::min(t_fup[t_from], t_fup[to]);
            if (t_parentEdge != -1 && t_fup[to] >= t_tin[t_from]) {
                isCutVertex[to] = true;
            }
        }
    }
    if (t_parentEdge == -1 && dfsCount > 1) {
        isCutVertex[t_from] = true;
    }
}

```

```

    }
}

template<class TEdge, class TWeight>
std::pair<std::vector<TWeight>, std::vector<std::size_t>>
Graph<TEdge, TWeight>::Dijkstra(std::size_t t_from) const {
    std::size_t n = this->vertexCount();
    std::vector<TWeight> dist(n, TWeight());
    std::vector<std::size_t> parent(n);
    std::iota(parent.begin(), parent.end(), 0);
    mark_list initialDist(n);
    std::set<std::pair<TWeight, std::size_t>> q;
    initialDist[t_from] = 1;
    q.emplace(dist[t_from], t_from);
    while (!q.empty()) {
        auto [d, v] = *q.begin();
        q.erase(q.begin());
        for (auto &i : this->m_incidentEdges[v]) {
            auto u = this->m_edges[i].to();
            auto nd = d + this->m_edges[i].getWeight();
            if (!initialDist[u]) {
                initialDist[u] = true;
                parent[u] = v;
                dist[u] = nd;
                q.emplace(nd, u);
            } else if (dist[u] > nd) {
                q.erase({dist[u], u});
                parent[u] = v;
                dist[u] = nd;
                q.emplace(nd, u);
            }
        }
    }
    return {dist, parent};
}

template<class TEdge, class TWeight>
bool Graph<TEdge, TWeight>::isAcyclic() const {
    mark_list colors(this->vertexCount());
    for (std::size_t v = 0; v < colors.size(); ++v) {
        if (colors[v] == 0) {
            if (checkCycleDfs(v, -1, colors)) {
                return false;
            }
        }
    }
    return true;
}

template<class TEdge, class TWeight>
bool Graph<TEdge, TWeight>::checkCycleDfs(std::size_t t_from, int t_parentEdge,
Graph::mark_list &t_colors) const {
    t_colors[t_from]++;
    for (auto &i : this->m_incidentEdges[t_from]) {
        if (!m_isDirected && ((i ^ 1) == t_parentEdge)) {
            continue;
        }
        auto u = this->m_edges[i].to();
        if (t_colors[u] == 0) {
            if (checkCycleDfs(u, i, t_colors)) {
                return true;
            }
        } else if (t_colors[u] == 1) {
            return true;
        }
    }
}

```

```

    }
}
t_colors[t_from]++;
return false;
}

template<class TEdge, class TWeight>
std::pair<std::vector<TWeight>, std::vector<std::size_t>>
Graph<TEdge, TWeight>::FordBellman(std::size_t t_from, const TWeight &t_minusInf)
const {
    std::size_t n = this->vertexCount();
    std::vector<TWeight> dist(n, TWeight());
    std::vector<std::size_t> parent(n);
    std::iota(parent.begin(), parent.end(), 0);
    mark_list initialDist(n);
    initialDist[t_from] = 1;

    for (std::size_t i = 1; i < n; ++i) {
        bool condensate = false;
        for (auto &e : this->m_edges) {
            if (!initialDist[e.from()]) {
                continue;
            }
            auto u = e.to();
            auto nd = dist[e.from()] + e.getWeight();
            if (!initialDist[u] || dist[u] > nd) {
                dist[u] = std::max(t_minusInf, nd);
                initialDist[u] = 1;
                parent[u] = e.from();
                condensate = true;
            }
        }
        if (!condensate) {
            break;
        }
    }

    mark_list was(n);
    for (auto &e : this->m_edges) {
        if (!initialDist[e.from()]) {
            continue;
        }
        auto nd = dist[e.from()] + e.getWeight();
        if (dist[e.to()] > nd) {
            dist[e.to()] = t_minusInf;
            parent[e.to()] = e.to();
            if (!was[e.to()]) {
                std::queue<std::size_t> q;
                q.push(e.to());
                was[e.to()] = true;
                while (!q.empty()) {
                    auto v = q.front();
                    q.pop();
                    for (auto &i : this->m_incidentEdges[v]) {
                        auto u = this->m_edges[i].to();
                        if (!was[u]) {
                            dist[u] = t_minusInf;
                            parent[u] = u;
                            was[u] = true;
                            q.push(u);
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    return {dist, parent};
}

template<class TEdge, class TWeight>
std::vector<TEdge> Graph<TEdge, TWeight>::edges() const {
    return this->m_edges;
}

}

#endif //COMPETITIVE_ALGORITHMS_GRAPH_H

```

Листинг Б.5 – Файл IDataStructure.h

```

#ifndef COMPETITIVE_ALGORITHMS_IDATASTRUCTURE_H
#define COMPETITIVE_ALGORITHMS_IDATASTRUCTURE_H

#include <cstdlib>

namespace cpa {

    template<class TType>
    class IDataStructure {
    protected:
        std::size_t m_size{};
    public:
        [[nodiscard]]
        std::size_t size() const noexcept;

        virtual const TType &getValueAt(std::size_t t_index) const = 0;
    };

    template<class TType>
    std::size_t IDataStructure<TType>::size() const noexcept {
        return this->m_size;
    }

}

#endif //COMPETITIVE_ALGORITHMS_IDATASTRUCTURE_H

```

Листинг Б.6 – Файл IEdge.h

```

#ifndef COMPETITIVE_ALGORITHMS_IEDGE_H
#define COMPETITIVE_ALGORITHMS_IEDGE_H

#include <cstdlib>
#include <algorithm>

namespace cpa {

    template<class TWeight>
    class IEdge {
    protected:
        std::size_t m_from{};
        std::size_t m_to{};
        TWeight m_weight;

        virtual void setWeight(const TWeight &t_weight);
    };
}

```

```

public:
    IEdge() = default;
    IEdge(const IEdge &other);

    [[nodiscard]] std::size_t from() const noexcept;

    [[nodiscard]] std::size_t to() const noexcept;

    [[nodiscard]] virtual TWeight getWeight() const noexcept;

    void reverse() noexcept;
};

}

#endif //COMPETITIVE_ALGORITHMS_IEDGE_H

```

Листинг Б.7 – Файл IMutableDataStructure.h

```

#ifndef COMPETITIVE_ALGORITHMS_IMUTABLEDATASTRUCTURE_H
#define COMPETITIVE_ALGORITHMS_IMUTABLEDATASTRUCTURE_H

#include "IDataStructure.h"

namespace cpa {

    template<class TType>
    class IMutableDataStructure : public IDataStructure<TType> {
    public:
        virtual void changeValueAt(std::size_t t_index, const TType& t_value) = 0;
    };

}

#endif //COMPETITIVE_ALGORITHMS_IMUTABLEDATASTRUCTURE_H

```

Листинг Б.8 – Файл ISegmentSolver.h

```

#ifndef COMPETITIVE_ALGORITHMS_ISEGMENTSOLVER_H
#define COMPETITIVE_ALGORITHMS_ISEGMENTSOLVER_H

#include <functional>
#include <vector>

namespace cpa {

    template<class TType, class TFunction = std::function<TType(const TType&, const
TType&>>>
    class ISegmentSolver {
    protected:
        TFunction m_function;
    public:
        virtual TType valueOnSegment(std::size_t t_left_index, std::size_t
t_right_index) const = 0;
    };

}

#endif //COMPETITIVE_ALGORITHMS_ISEGMENTSOLVER_H

```

Листинг Б.9 – Файл SegmentTree.h

```

#ifndef COMPETITIVE_ALGORITHMS_SEGMENTTREE_H
#define COMPETITIVE_ALGORITHMS_SEGMENTTREE_H

#include "ImmutableDataStructure.h"
#include "ISegmentSolver.h"

namespace cpa {

    template<class TType, class TFunction = std::function<TType(const TType &, const
TType &>>>
    class SegmentTree :
        public ImmutableDataStructure<TType>,
        public ISegmentSolver<TType, TFunction> {
    private:
        std::vector<TType> m_nodes;

        template<class TIter>
        void build(TIter t_begin, TIter t_end);

    public:
        explicit SegmentTree(const TFunction &t_function = nullptr);

        template<class TIter>
        SegmentTree(TIter t_begin, TIter t_end, const TFunction &t_function =
nullptr);

        SegmentTree(const SegmentTree<TType, TFunction> &other);

        SegmentTree(SegmentTree<TType, TFunction> &&other) noexcept;

        const TType &getValueAt(std::size_t t_index) const override;

        void changeValueAt(std::size_t t_index, const TType &t_value) override;

        TType valueOnSegment(std::size_t t_left_index, std::size_t t_right_index)
const override;
    };

    template<class TType, class TFunction>
    SegmentTree<TType, TFunction>::SegmentTree(const TFunction &t_function) {
        this->m_function = t_function;
    }

    template<class TType, class TFunction>
    template<class TIter>
    SegmentTree<TType, TFunction>::SegmentTree(TIter t_begin, TIter t_end, const
TFunction &t_function) {
        this->m_function = t_function;
        build(t_begin, t_end);
    }

    template<class TType, class TFunction>
    SegmentTree<TType, TFunction>::SegmentTree(const SegmentTree<TType, TFunction>
&other) {
        this->m_function = other.m_function;
        this->m_size = other.m_size;
        this->m_nodes = other.m_nodes;
    }

    template<class TType, class TFunction>
    SegmentTree<TType, TFunction>::SegmentTree(SegmentTree<TType, TFunction> &&other)
noexcept {

```

```

        this->m_function = std::move(other.m_function);
        this->m_size = std::move(other.m_size);
        this->m_nodes = std::move(other.m_nodes);
    }

    template<class TType, class TFunction>
    template<class TIter>
    void SegmentTree<TType, TFunction>::build(TIter t_begin, TIter t_end) {
        this->m_size = std::distance(t_begin, t_end);
        if (this->m_size == 0) {
            return;
        }
        this->m_nodes.resize(this->m_size * 2);
        std::copy(t_begin, t_end, this->m_nodes.begin() + this->m_size);
        for (auto i = this->m_size - 1; i > 0; --i) {
            this->m_nodes[i] = this->m_function(this->m_nodes[i << 1u], this->m_nodes[(i << 1u) | 1u]);
        }
    }

    template<class TType, class TFunction>
    const TType &SegmentTree<TType, TFunction>::getValueAt(std::size_t t_index) const
    {
        return this->m_nodes[this->m_size + t_index];
    }

    template<class TType, class TFunction>
    void SegmentTree<TType, TFunction>::changeValueAt(std::size_t t_index, const
    TType &t_value) {
        t_index += this->m_size;
        for (this->m_nodes[t_index] = t_value; t_index > 1; t_index >>= 1u) {
            this->m_nodes[t_index >> 1u] = this->m_function(this->m_nodes[t_index],
            this->m_nodes[t_index ^ 1u]);
        }
    }

    template<class TType, class TFunction>
    TType SegmentTree<TType, TFunction>::valueOnSegment(std::size_t t_left_index,
    std::size_t t_right_index) const {
        t_left_index += this->m_size;
        t_right_index += this->m_size + 1;
        while ((t_left_index & 1u) == 0 && (t_right_index & 1u) == 0) {
            t_left_index >>= 1u;
            t_right_index >>= 1u;
        }
        auto odd_left = (t_left_index & 1u) == 1;
        auto odd_right = (t_right_index & 1u) == 1;
        auto result = odd_left ? this->m_nodes[t_left_index++] : this->m_nodes[--
        t_right_index];
        if (odd_left && odd_right) {
            result = this->m_function(result, this->m_nodes[--t_right_index]);
        }
        t_left_index >>= 1u;
        t_right_index >>= 1u;
        while (t_left_index < t_right_index) {
            if ((t_left_index & 1u) == 1) {
                result = this->m_function(result, this->m_nodes[t_left_index++]);
            }
            if ((t_right_index & 1u) == 1) {
                result = this->m_function(result, this->m_nodes[--t_right_index]);
            }
            t_left_index >>= 1u;
            t_right_index >>= 1u;
        }
    }

```

```

        return result;
    }

}

#endif //COMPETITIVE_ALGORITHMS_SEGMENTTREE_H

```

Листинг Б.10 – Файл SparseTable.h

```

#ifndef COMPETITIVE_ALGORITHMS_SPARSETABLE_H
#define COMPETITIVE_ALGORITHMS_SPARSETABLE_H

#include <vector>
#include "ISegmentSolver.h"
#include "IDataStructure.h"

namespace cpa {

    template<class TType, class TFunction = std::function<TType(const TType &, const
TType &)>>
    class SparseTable :
        public IDataStructure<TType>,
        public ISegmentSolver<TType, TFunction> {
    private:
        std::vector<std::size_t> m_log;
        std::vector<std::vector<TType>> m_st;

        void calculateLog();

        template<class TIter>
        void calculateSparseTable(TIter t_begin, TIter t_end);

    public:
        explicit SparseTable(const TFunction &t_function = nullptr);

        template<class TIter>
        SparseTable(TIter t_begin, TIter t_end, const TFunction &t_function =
nullptr);

        SparseTable(const SparseTable<TType, TFunction> &other);

        SparseTable(SparseTable<TType, TFunction> &&other) noexcept;

        const TType &getValueAt(std::size_t t_index) const override;

        TType valueOnSegment(std::size_t t_left_index, std::size_t t_right_index)
const override;
    };
}

#endif //COMPETITIVE_ALGORITHMS_SPARSETABLE_H

```

Листинг Б.11 – Файл SqrtDecomposer.h

```

#ifndef COMPETITIVE_ALGORITHMS_SQRTDECOMPOSER_H
#define COMPETITIVE_ALGORITHMS_SQRTDECOMPOSER_H

#include <cmath>
#include "ISegmentSolver.h"
#include "IMutableDataStructure.h"

namespace cpa {

```

```

    template<class TType, class TFunction = std::function<TType(const TType &, const
TType &)>>
    class SqrtDecomposer :
        public IMutableDataStructure<TType>,
        public ISegmentSolver<TType, TFunction> {
    private:
        std::vector<TType> m_data;
        std::vector<TType> m_block_data;
        std::size_t m_block_size;

        template<class TIter>
        void build(TIter t_begin, TIter t_end);

    public:
        explicit SqrtDecomposer(const TFunction &t_function = nullptr);

        template<class TIter>
        SqrtDecomposer(TIter t_begin, TIter t_end, const TFunction &t_function =
nullptr);

        SqrtDecomposer(const SqrtDecomposer<TType, TFunction> &other);

        SqrtDecomposer(SqrtDecomposer<TType, TFunction> &&other) noexcept;

        const TType &getValueAt(std::size_t t_index) const override;

        void changeValueAt(std::size_t t_index, const TType &t_value) override;

        TType valueOnSegment(std::size_t t_left_index, std::size_t t_right_index)
const override;
    };
}

#endif //COMPETITIVE_ALGORITHMS_SQRTDECOMPOSER_H

```

Листинг Б.12 – Файл SuffixAutomaton.h

```

#ifndef COMPETITIVE_ALGORITHMS_SUFFIXAUTOMATON_H
#define COMPETITIVE_ALGORITHMS_SUFFIXAUTOMATON_H

#include <map>
#include <vector>
#include <queue>

namespace cpa {

    class SuffixAutomaton {
    public:
        struct Node {
            friend class SuffixAutomaton;
        private:
            bool invLinkWas = false;
        public:
            std::map<char, Node *> child;
            std::vector<Node *> invLinks;
            Node *suffixLink;
            std::size_t length;
            int position;

            explicit Node(std::size_t length = 0);

            Node(const Node &other) = default;
        };
};

```

```

void assign(const std::string &t_s);

explicit SuffixAutomaton(const std::string &t_s = std::string());

SuffixAutomaton(const SuffixAutomaton &other);

SuffixAutomaton(SuffixAutomaton &&other) noexcept;

[[nodiscard]] Node *getRoot() const noexcept;

[[nodiscard]] int find(const std::string &t_s) const;

[[nodiscard]] std::vector<int> findAll(const std::string &t_s) const;

~SuffixAutomaton();

private:
    Node *m_root{};
    Node *m_last{};

    void addCharacter(char t_c, std::size_t t_length);

    void buildInvLinks(Node *current);
};

void SuffixAutomaton::addCharacter(char t_c, std::size_t t_length) {
    auto new_node = new Node(t_length);
    auto p = m_last;
    while (p != nullptr && p->child.count(t_c) == 0) {
        p->child[t_c] = new_node;
        p = p->suffixLink;
    }
    if (p == nullptr) {
        new_node->suffixLink = m_root;
    } else {
        auto q = p->child[t_c];
        if (q->length == p->length + 1) {
            new_node->suffixLink = q;
        } else {
            auto clone_node = new Node(*q);
            clone_node->length = p->length + 1;
            q->suffixLink = clone_node;
            new_node->suffixLink = clone_node;
            while (p != nullptr && p->child[t_c] == q) {
                p->child[t_c] = clone_node;
                p = p->suffixLink;
            }
        }
    }
    m_last = new_node;
}

void SuffixAutomaton::buildInvLinks(Node *current) {
    current->invLinkWas = true;
    if (current->suffixLink != nullptr) {
        current->suffixLink->invLinks.push_back(current);
    }
    for (auto &p : current->child) {
        if (!p.second->invLinkWas){
            buildInvLinks(p.second);
        }
    }
}

```

```

void SuffixAutomaton::assign(const std::string &t_s) {
    m_root = new Node();
    m_last = m_root;
    for (std::size_t i = 0; i < t_s.length(); ++i) {
        addCharacter(t_s[i], i + 1);
    }
    buildInvLinks(m_root);
}

SuffixAutomaton::SuffixAutomaton(const std::string &t_s) {
    assign(t_s);
}

SuffixAutomaton::SuffixAutomaton(const SuffixAutomaton &other) {
    std::string s;
    auto cur = other.m_root;
    while (!cur->child.empty()) {
        char nxt;
        for (auto &p : cur->child) {
            if (p.second->position == cur->position + 1) {
                nxt = p.first;
                break;
            }
        }
        s.push_back(nxt);
        cur = cur->child[nxt];
    }
    assign(s);
}

SuffixAutomaton::SuffixAutomaton(SuffixAutomaton &&other) noexcept {
    this->m_root = other.m_root;
    this->m_last = other.m_last;
    other.m_root = nullptr;
    other.m_last = nullptr;
}

int SuffixAutomaton::find(const std::string &t_s) const {
    auto cur = this->m_root;
    for (auto c : t_s) {
        if (cur->child.count(c) == 0) {
            return -1;
        }
        cur = cur->child[c];
    }
    return static_cast<int>(cur->position + 1 - t_s.length());
}

std::vector<int> SuffixAutomaton::findAll(const std::string &t_s) const {
    auto cur = this->m_root;
    for (auto c : t_s) {
        if (cur->child.count(c) == 0) {
            return {};
        }
        cur = cur->child[c];
    }
    std::vector<int> pos;
    std::queue<Node *> q;
    q.push(cur);
    while (!q.empty()) {
        auto v = q.front();
        q.pop();
        if (v->position + 1 == v->length) {

```

```

        pos.push_back(static_cast<int>(v->position + 1 - t_s.length()));
    }
    for (auto &u : v->invLinks) {
        q.push(u);
    }
}
return pos;
}

SuffixAutomaton::~SuffixAutomaton() {
    std::vector<Node*> q;
    q.push_back(this->m_root);
    this->m_root->position = -2;
    for (std::size_t i = 0; i < q.size(); ++i) {
        auto v = q[i];
        for (auto &p : v->child) {
            if (p.second->position != -2) {
                p.second->position = -2;
                q.push_back(p.second);
            }
        }
    }
    for (auto &v : q) {
        delete v;
    }
    this->m_root = nullptr;
    this->m_last = nullptr;
}

SuffixAutomaton::Node *SuffixAutomaton::getRoot() const noexcept {
    return this->m_root;
}

SuffixAutomaton::Node::Node(std::size_t length) {
    suffixLink = nullptr;
    this->length = length;
    this->position = static_cast<int>(length) - 1;
}

}

#endif //COMPETITIVE_ALGORITHMS_SUFFIXAUTOMATON_H

```

Листинг Б.13 – Файл Treap.h

```

#ifndef COMPETITIVE_ALGORITHMS_TREAP_H
#define COMPETITIVE_ALGORITHMS_TREAP_H

#include <random>
#include <chrono>
#include <memory>
#include <stack>
#include "ImmutableDataStructure.h"
#include "ISegmentSolver.h"

namespace cpa {

    template<class TType, class TFunction = std::function<TType(const TType &, const
TType &>>>
    class Treap :
        public ImmutableDataStructure<TType>,
        public ISegmentSolver<TType, TFunction> {
    private:

```

```

std::mt19937 m_rnd;
TFunction m_function;

struct node {
    std::size_t size;
    unsigned int prior;
    TType value;
    TType segmentValue;
    std::unique_ptr<node> left, right;

    explicit node(const TType &value, unsigned int prior) :
        size(1),
        prior(prior),
        value(value),
        segmentValue(value),
        left(nullptr),
        right(nullptr) {}

    node(const node &other) :
        size(other.size),
        prior(other.prior),
        value(other.value),
        segmentValue(other.segmentValue),
        left(other->left == nullptr ? nullptr :
std::make_unique<node>(*other->left)),
        right(other->right == nullptr ? nullptr :
std::make_unique<node>(*other->right)) {}

    void update(const TFunction &t_function);
};

void initial_random();

template<class TIter>
void build(TIter t_begin, TIter t_end);

const TType &getValueAt(const std::unique_ptr<node> &t_current, std::size_t
t_index) const;

void changeValueAt(std::unique_ptr<node> &t_current, std::size_t t_index,
const TType &t_value);

TType
valueOnSegment(const std::unique_ptr<node> &t_current, std::size_t
t_left_index,
                std::size_t t_right_index) const;

public:
    std::unique_ptr<node> m_root = nullptr;

    explicit Treap(const TFunction &t_function = nullptr);

    template<class TIter>
    Treap(TIter t_begin, TIter t_end, const TFunction &t_function = nullptr);

    Treap(const Treap<TType, TFunction> &other);

    Treap(Treap<TType, TFunction> &&other) noexcept;

    const TType &getValueAt(std::size_t t_index) const override;

    void changeValueAt(std::size_t t_index, const TType &t_value) override;

```

```

        TType valueOnSegment(std::size_t t_left_index, std::size_t t_right_index)
const override;
    };
}

#endif //COMPETITIVE_ALGORITHMS_TREAP_H

```

Листинг Б.14 – Файл WeightedEdge.h

```

#ifndef COMPETITIVE_ALGORITHMS_WEIGHTEDEDGE_H
#define COMPETITIVE_ALGORITHMS_WEIGHTEDEDGE_H

#include <algorithm>
#include "IEdge.h"

namespace cpa {
    template<class TWeight>
    class WeightedEdge : public IEdge<TWeight> {

    public:
        explicit WeightedEdge(std::size_t t_from, std::size_t t_to, const TWeight
&t_weight = TWeight());

        WeightedEdge(const WeightedEdge<TWeight> &other);

        WeightedEdge(WeightedEdge<TWeight> &&other) noexcept;

        void setWeight(const TWeight &t_weight) override;
    };

    template<class TWeight>
    WeightedEdge<TWeight>::WeightedEdge(std::size_t t_from, std::size_t t_to, const
TWeight &t_weight) {
        this->m_from = t_from;
        this->m_to = t_to;
        this->m_weight = t_weight;
    }

    template<class TWeight>
    WeightedEdge<TWeight>::WeightedEdge(const WeightedEdge<TWeight> &other) {
        this->m_from = other.m_from;
        this->m_to = other.m_to;
        this->m_weight = other.m_weight;
    }

    template<class TWeight>
    WeightedEdge<TWeight>::WeightedEdge(WeightedEdge<TWeight> &&other) noexcept {
        this->m_from = other.m_from;
        this->m_to = other.m_to;
        this->m_weight = std::move(other.m_weight);
    }

    template<class TWeight>
    void WeightedEdge<TWeight>::setWeight(const TWeight &t_weight) {
        this->m_weight = t_weight;
    }
}

#endif //COMPETITIVE_ALGORITHMS_WEIGHTEDEDGE_H

```

Приложение В
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)

```

#include <windows.h>
#include <iostream>
#include "../competitiveProgAlgoLib/SegmentTree.h"

int main() {
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);
    auto minimum = [&](const int &a, const int &b) {
        if (a < b) return a;
        return b;
    };
    std::vector data{4, 3, 0, 7, 8};
    cpa::SegmentTree<int> segmentTree(data.begin(), data.end(), minimum);
    std::cout << "Дерево отрезков успешно построено" << std::endl;
    for (int i = 0; i < segmentTree.size(); ++i) {
        for (int j = i; j < segmentTree.size(); ++j) {
            std::cout << "Минимум на отрезке [" << i + 1 << ", " << j + 1 << "] = ";
            std::cout << segmentTree.valueOnSegment(i, j) << std::endl;
        }
    }
    return 0;
}

```

segTreeTest1 ×

D:\projects\olympiadAlgoLib\competitive_algorithms\cmake-build-debug\segTreeTest1.e

Дерево отрезков успешно построено

Минимум на отрезке [1, 1] = 4
 Минимум на отрезке [1, 2] = 3
 Минимум на отрезке [1, 3] = 0
 Минимум на отрезке [1, 4] = 0
 Минимум на отрезке [1, 5] = 0
 Минимум на отрезке [2, 2] = 3
 Минимум на отрезке [2, 3] = 0
 Минимум на отрезке [2, 4] = 0
 Минимум на отрезке [2, 5] = 0
 Минимум на отрезке [3, 3] = 0
 Минимум на отрезке [3, 4] = 0
 Минимум на отрезке [3, 5] = 0
 Минимум на отрезке [4, 4] = 7
 Минимум на отрезке [4, 5] = 7
 Минимум на отрезке [5, 5] = 8

Рисунок В.1 – Результаты тестирования варианта использования «Решить задачу на подотрезках массива»

```
#include <windows.h>
#include <iostream>
#include "../competitiveProgAlgoLib/SegmentTree.h"

int main() {
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);
    auto maximum = [&](const int &a, const int &b) {
        if (a > b) return a;
        return b;
    };
    std::vector data{89, 12, 17, 4, 9};
    cpa::SegmentTree<int> segmentTree(data.begin(), data.end(), maximum);
    std::cout << "Дерево отрезков успешно построено" << std::endl;
    std::cout << "Максимум на отрезке [2, 4] = " << segmentTree.valueOnSegment( t_left_index: 1, t_right_index: 3) << std::endl;
    return 0;
}
```

segTreeTest2 ×

D:\projects\olympiadAlgoLib\competitive_algorithms\cmake-build-debug\segTreeTest2.exe

Дерево отрезков успешно построено

Максимум на отрезке [2, 4] = 17

Process finished with exit code 0

Рисунок В.2 – Результат тестирования вариантов использования «Построить структуру данных» и «Получить значение функции на отрезке массива»

```
#include <windows.h>
#include <iostream>
#include "../competitiveProgAlgoLib/Graph.h"
#include "../competitiveProgAlgoLib/Edge.h"

int main() {
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);
    cpa::Graph<cpa::Edge> graph( t_vertexCount: 4, t_isDirected: true);
    graph.addEdge( t_edge: cpa::Edge( t_from: 0, t_to: 1));
    graph.addEdge( t_edge: cpa::Edge( t_from: 1, t_to: 3));
    graph.addEdge( t_edge: cpa::Edge( t_from: 0, t_to: 2));
    graph.addEdge( t_edge: cpa::Edge( t_from: 2, t_to: 3));
    std::cout << "Граф успешно построен" << std::endl;
    if (graph.isAcyclic()){
        std::cout << "Граф является ациклическим" << std::endl;
    } else {
        std::cout << "Граф является циклическим" << std::endl;
    }
    return 0;
}
```

graphTest1 ×

D:\projects\olympiadAlgoLib\competitive_algorithms\cmake-build-debug\graphTest1.exe

Граф успешно построен

Граф является ациклическим

Process finished with exit code 0

Рисунок В.3 – Результат тестирования варианта использования «Решить задачу на графах»

```
#include <windows.h>
#include <iostream>
#include "../competitiveProgAlgoLib/Graph.h"
#include "../competitiveProgAlgoLib/Edge.h"

int main() {
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);
    cpa::Graph<cpa::Edge> graph( t_vertexCount: 5, t_isDirected: false);
    graph.addEdge( t_edge: cpa::Edge( t_from: 0, t_to: 1));
    graph.addEdge( t_edge: cpa::Edge( t_from: 1, t_to: 2));
    graph.addEdge( t_edge: cpa::Edge( t_from: 2, t_to: 0));
    graph.addEdge( t_edge: cpa::Edge( t_from: 2, t_to: 4));
    graph.addEdge( t_edge: cpa::Edge( t_from: 4, t_to: 3));
    graph.addEdge( t_edge: cpa::Edge( t_from: 3, t_to: 2));
    std::cout << "Граф успешно построен" << std::endl;
    std::cout << "Найденные точки сочленения:" << std::endl;
    for(auto v : graph.cutVertexes()){
        std::cout << v + 1 << std::endl;
    }
    return 0;
}
```

graphTest2 ×

D:\projects\olympiadAlgoLib\competitive_algorithms\cmake-build-debug\graphTest2.exe

Граф успешно построен

Найденные точки сочленения:

3

Process finished with exit code 0

Рисунок В.4 – Результат тестирования вариантов использования «Построить граф» и «Использовать необходимый алгоритм»

```
#include <windows.h>
#include <iostream>
#include "../competitiveProgAlgoLib/SuffixAutomaton.h"

int main() {
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);
    std::string data = "abacaba";
    cpa::SuffixAutomaton suffixAutomaton(data);
    std::cout << "Суффиксный автомат успешно построен" << std::endl;
    std::cout << "Позиция первого вхождения \"bac\": " << suffixAutomaton.find( t_s: "bac") + 1 << std::endl;
    return 0;
}
```

f main

suffixAutomatonTest1 ×

D:\projects\olympiadAlgoLib\competitive_algorithms\cmake-build-debug\suffixAutomatonTest1.exe

Суффиксный автомат успешно построен

Позиция первого вхождения "bac": 2

Рисунок В.5 – Результат тестирования варианта использования «Решить задачу на обработку строк»

```

#include ...
int main() {
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);
    std::string data = "ababb";
    cpa::SuffixAutomaton suffixAutomaton(data);
    std::cout << "Суффиксный автомат успешно построен" << std::endl;
    std::cout << "Позиция всех вхождения \"ab\": " << std::endl;
    for (auto i : suffixAutomaton.findAll(t_s: "ab")) std::cout << i + 1 << std::endl;
    return 0;
}

```

suffixAutomatonTest2 x

D:\projects\olympiadAlgoLib\competitive_algorithms\cmake-build-debug\suffixAutomatonTest2.exe

Суффиксный автомат успешно построен

Позиция всех вхождения "ab":

1

3

Рисунок В.6 – Результат тестирования вариантов использования «Построить суффиксный автомат» и «Использовать суффиксный автомат»

		Tests passed: 33 of 33 tests – 11 s 136 ms
Test Results		11 s 136 ms
Directed graph testing	0 ms	Testing started at 16:05 ... D:\projects\olympiadAlgoLib\compe Process finished with exit code 0
add edge	0 ms	
connectivity components	0 ms	
topological sort	0 ms	
is acyclic	0 ms	
strong connectivity components	0 ms	
Dijkstra	0 ms	
Ford-Bellman	0 ms	
Range query	5 s 766 ms	
handle tests	0 ms	
array	0 ms	
sparse table	0 ms	
segment tree	0 ms	
fenwick tree	0 ms	
sqrt decomposition	0 ms	
treap	0 ms	
random static tests	5 s 363 ms	
Shortest path in graph	1 ms	
bamboo graph	1 ms	
star graph	0 ms	
directed graph	0 ms	
start from 1	0 ms	
start from 4	0 ms	
undirected graph	0 ms	
graph with negative weight	0 ms	
Suffix automaton	1 ms	
copy constructor	0 ms	
long string of equals character	1 ms	
Undirected graph testing	3 ms	
connectivity components	0 ms	
is acyclic	0 ms	
BRIDGES	0 ms	
CUT VERTEX	0 ms	

Рисунок В.7 – Результаты модульного тестирования

Приложение Г
ИСХОДНЫЙ КОД МОДУЛЬНОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

Листинг Г.1 – Исходный код модульного тестирования

```

#define CATCH_CONFIG_MAIN

#include "../catch2/catch.h"

#include "../competitiveProgAlgoLib/Graph.h"
#include "../competitiveProgAlgoLib/Edge.h"
#include "../competitiveProgAlgoLib/WeightedEdge.h"
#include "../competitiveProgAlgoLib/SuffixAutomaton.h"
#include "../competitiveProgAlgoLib/Array.h"
#include "../competitiveProgAlgoLib/ISparseTable.h"
#include "../competitiveProgAlgoLib/SegmentTree.h"
#include "../competitiveProgAlgoLib/FenwickTree.h"
#include "../competitiveProgAlgoLib/SqrtDecomposer.h"
#include "../competitiveProgAlgoLib/Treap.h"

TEST_CASE("Directed graph testing", "[graph]") {

    cpa::Graph<cpa::Edge> g(5, true);
    REQUIRE(g.vertexCount() == 5);
    REQUIRE(g.isDirected());

    g.addEdge(cpa::Edge(2, 1));
    g.addEdge(cpa::Edge(1, 4));
    g.addEdge(cpa::Edge(0, 3));
    g.addEdge(cpa::Edge(3, 4));
    g.addEdge(cpa::Edge(2, 0));
    REQUIRE(g.edgeCount() == 5);

    SECTION("add edge") {
        g.addEdge(cpa::Edge(0, 4));
        REQUIRE(g.edgeCount() == 6);
        g.addEdge(cpa::Edge(0, 4));
        g.addEdge(cpa::Edge(0, 4));
        REQUIRE(g.edgeCount() == 8);
    }

    SECTION("connectivity components") {
        auto comp = g.connectivityComponents();
        REQUIRE(comp.size() == 1);
        REQUIRE_THAT(comp[0], Catch::UnorderedEquals(std::vector{0u, 1u, 2u, 3u,
4u}));
    }

    SECTION("topological sort") {
        auto order = g.topologicalSort();
        REQUIRE(order.size() == g.vertexCount());
        std::vector<std::size_t> pos(g.vertexCount());
        for (auto i = 0u; i < order.size(); ++i) {
            pos[order[i]] = i;
        }
        for (auto &e : g.edges()) {
            if (pos[e.from()] >= pos[e.to()]) {
                REQUIRE(pos[e.from()] < pos[e.to()]);
            }
        }
    }

    SECTION("is acyclic") {
        REQUIRE(g.isAcyclic());
        g.addEdge(cpa::Edge(0, 4));
        REQUIRE(g.isAcyclic());
    }
}

```

```

    g.addEdge(cpa::Edge(3, 2));
    REQUIRE_FALSE(g.isAcyclic());
}

SECTION("strong connectivity components") {
    auto comp = g.strongConnectivityComponents();
    REQUIRE(comp.size() == 5);
    std::vector<bool> was(g.vertexCount());
    for (auto &a : comp) {
        REQUIRE(a.size() == 1);
        for (auto &v : a) {
            INFO("vertex should be unique")
            REQUIRE(!was[v]);
            was[v] = true;
        }
    }

    g.addEdge(cpa::Edge(0, 4));
    comp = g.strongConnectivityComponents();
    REQUIRE(comp.size() == 5);
    was.assign(g.vertexCount(), false);
    for (auto &a : comp) {
        REQUIRE(a.size() == 1);
        for (auto &v : a) {
            INFO("vertex should be unique")
            REQUIRE(!was[v]);
            was[v] = true;
        }
    }

    g.addEdge(cpa::Edge(3, 2));
    comp = g.strongConnectivityComponents();
    REQUIRE(comp.size() == 3);
    was.assign(g.vertexCount(), false);
    for (auto &a : comp) {
        REQUIRE((a.size() == 1 || a.size() == 3));
        for (auto &v : a) {
            INFO("vertex should be unique")
            REQUIRE(!was[v]);
            was[v] = true;
        }
        if (a.size() == 3) {
            REQUIRE_THAT(a, Catch::UnorderedEquals(std::vector{0u, 2u, 3u}));
        }
    }

    g.addEdge(cpa::Edge(4, 2));
    comp = g.strongConnectivityComponents();
    REQUIRE(comp.size() == 1);
    was.assign(g.vertexCount(), false);
    for (auto &a : comp) {
        REQUIRE(a.size() == 5);
        for (auto &v : a) {
            INFO("vertex should be unique")
            REQUIRE(!was[v]);
            was[v] = true;
        }
        REQUIRE_THAT(a, Catch::UnorderedEquals(std::vector{0u, 1u, 2u, 3u, 4u}));
    }
}

SECTION("Dijkstra") {
    auto[dist, p] = g.Dijkstra(2);
    REQUIRE_THAT(dist, Catch::Equals(std::vector{1, 1, 0, 2, 2}));
}

```

```

    REQUIRE_THAT(p, Catch::Equals(std::vector{2u, 2u, 2u, 0u, 1u}));
}

SECTION("Ford-Bellman") {
    auto[dist, p] = g.FordBellman(2, -1);
    REQUIRE_THAT(dist, Catch::Equals(std::vector{1, 1, 0, 2, 2}));
    REQUIRE_THAT(p, Catch::Equals(std::vector{2u, 2u, 2u, 0u, 1u}));
}

}

TEST_CASE("Undirected graph testing", "[graph]") {

    cpa::Graph<cpa::Edge, int> g(6, false);
    REQUIRE(g.vertexCount() == 6);
    REQUIRE(g.edgeCount() == 0);

    SECTION("connectivity components") {
        auto comp = g.connectivityComponents();
        REQUIRE(comp.size() == 6);

        g.addEdge(cpa::Edge(3, 2));
        g.addEdge(cpa::Edge(2, 4));
        g.addEdge(cpa::Edge(3, 4));
        comp = g.connectivityComponents();
        REQUIRE(comp.size() == 4);

        g.addEdge(cpa::Edge(0, 1));
        g.addEdge(cpa::Edge(1, 5));
        comp = g.connectivityComponents();
        REQUIRE(comp.size() == 2);

        g.addEdge(cpa::Edge(1, 3));
        comp = g.connectivityComponents();
        REQUIRE(comp.size() == 1);
        REQUIRE_THAT(comp[0], Catch::UnorderedEquals(std::vector{0u, 1u, 2u, 3u, 4u,
5u}));
    }

    g.addEdge(cpa::Edge(3, 2));
    g.addEdge(cpa::Edge(3, 4));
    g.addEdge(cpa::Edge(0, 1));
    g.addEdge(cpa::Edge(1, 5));

    SECTION("is acyclic") {
        REQUIRE(g.isAcyclic());

        g.addEdge(cpa::Edge(2, 4));
        REQUIRE_FALSE(g.isAcyclic());
    }

    SECTION("BRIDGES") {
        auto bridges = g.bridges();
        REQUIRE(bridges.size() == 4);

        g.addEdge(cpa::Edge(2, 4));
        bridges = g.bridges();
        REQUIRE(bridges.size() == 2);

        g.addEdge(cpa::Edge(5, 0));
        bridges = g.bridges();
        REQUIRE(bridges.empty());

        g.addEdge(cpa::Edge(1, 3));
    }
}

```

```

        bridges = g.bridges();
        REQUIRE(bridges.size() == 1);
    }

    SECTION("CUT VERTEX") {
        auto cutVertexes = g.cutVertexes();
        REQUIRE(cutVertexes.size() == 2);

        g.addEdge(cpa::Edge(2, 4));
        cutVertexes = g.cutVertexes();
        REQUIRE(cutVertexes.size() == 1);

        g.addEdge(cpa::Edge(5, 0));
        cutVertexes = g.cutVertexes();
        REQUIRE(cutVertexes.empty());

        g.addEdge(cpa::Edge(1, 3));
        cutVertexes = g.cutVertexes();
        REQUIRE(cutVertexes.size() == 2);
    }
}

TEST_CASE("Shortest path in graph", "[graph]") {

    SECTION("bamboo graph") {
        cpa::Graph<cpa::WeightedEdge<int>, int> g(10);
        g.addEdge(cpa::WeightedEdge<int>(0, 1, 1));
        g.addEdge(cpa::WeightedEdge<int>(1, 2, 2));
        g.addEdge(cpa::WeightedEdge<int>(2, 3, 3));
        g.addEdge(cpa::WeightedEdge<int>(3, 4, 4));
        g.addEdge(cpa::WeightedEdge<int>(4, 5, 5));
        g.addEdge(cpa::WeightedEdge<int>(5, 6, 6));
        g.addEdge(cpa::WeightedEdge<int>(6, 7, 7));
        g.addEdge(cpa::WeightedEdge<int>(7, 8, 8));
        g.addEdge(cpa::WeightedEdge<int>(8, 9, 9));

        auto[dist1, p1] = g.Dijkstra(0);
        REQUIRE_THAT(dist1, Catch::Equals(std::vector{0, 1, 3, 6, 10, 15, 21, 28, 36,
45}));
        REQUIRE_THAT(p1, Catch::Equals(std::vector<unsigned>{0, 0, 1, 2, 3, 4, 5, 6,
7, 8}));

        auto[dist2, p2] = g.FordBellman(0, -1);
        REQUIRE_THAT(dist2, Catch::Equals(std::vector{0, 1, 3, 6, 10, 15, 21, 28, 36,
45}));
        REQUIRE_THAT(p2, Catch::Equals(std::vector<unsigned>{0, 0, 1, 2, 3, 4, 5, 6,
7, 8}));
    }

    SECTION("star graph") {
        cpa::Graph<cpa::WeightedEdge<int>, int> g(10);
        g.addEdge(cpa::WeightedEdge<int>(0, 1, 1));
        g.addEdge(cpa::WeightedEdge<int>(0, 2, 2));
        g.addEdge(cpa::WeightedEdge<int>(0, 3, 3));
        g.addEdge(cpa::WeightedEdge<int>(0, 4, 4));
        g.addEdge(cpa::WeightedEdge<int>(0, 5, 5));
        g.addEdge(cpa::WeightedEdge<int>(0, 6, 6));
        g.addEdge(cpa::WeightedEdge<int>(0, 7, 7));
        g.addEdge(cpa::WeightedEdge<int>(0, 8, 8));
        g.addEdge(cpa::WeightedEdge<int>(0, 9, 9));

        auto[dist1, p1] = g.Dijkstra(0);
    }
}

```

```

    REQUIRE_THAT(dist1, Catch::Equals(std::vector{0, 1, 2, 3, 4, 5, 6, 7, 8,
9}));
    REQUIRE_THAT(p1, Catch::Equals(std::vector<unsigned>{0, 0, 0, 0, 0, 0, 0, 0,
0, 0}));

    auto[dist2, p2] = g.FordBellman(0, -1);
    REQUIRE_THAT(dist2, Catch::Equals(std::vector{0, 1, 2, 3, 4, 5, 6, 7, 8,
9}));
    REQUIRE_THAT(p2, Catch::Equals(std::vector<unsigned>{0, 0, 0, 0, 0, 0, 0, 0,
0, 0}));
}

SECTION("directed graph") {
    cpa::Graph<cpa::WeightedEdge<int>, int> g(10, true);
    g.addEdge(cpa::WeightedEdge(0, 2, 4));
    g.addEdge(cpa::WeightedEdge(2, 3, 0));
    g.addEdge(cpa::WeightedEdge(3, 0, 2));
    g.addEdge(cpa::WeightedEdge(2, 7, 15));
    g.addEdge(cpa::WeightedEdge(3, 7, 11));
    g.addEdge(cpa::WeightedEdge(0, 1, 3));
    g.addEdge(cpa::WeightedEdge(0, 9, 12));
    g.addEdge(cpa::WeightedEdge(7, 1, 6));
    g.addEdge(cpa::WeightedEdge(7, 4, 3));
    g.addEdge(cpa::WeightedEdge(1, 5, 100));
    g.addEdge(cpa::WeightedEdge(9, 8, 13));
    g.addEdge(cpa::WeightedEdge(5, 8, 4));
    g.addEdge(cpa::WeightedEdge(8, 6, 0));
    g.addEdge(cpa::WeightedEdge(6, 0, 7));
    g.addEdge(cpa::WeightedEdge(8, 4, 17));

    SECTION("start from 1") {
        auto[dist1, p1] = g.Dijkstra(1);
        REQUIRE_THAT(dist1, Catch::Equals(std::vector{111, 0, 115, 115, 121, 100,
104, 126, 104, 123}));
        REQUIRE_THAT(p1, Catch::Equals(std::vector<unsigned>{6, 1, 0, 2, 8, 1, 8,
3, 5, 0}));

        auto[dist2, p2] = g.FordBellman(1, -1);
        REQUIRE_THAT(dist2, Catch::Equals(std::vector{111, 0, 115, 115, 121, 100,
104, 126, 104, 123}));
        REQUIRE_THAT(p2, Catch::Equals(std::vector<unsigned>{6, 1, 0, 2, 8, 1, 8,
3, 5, 0}));
    }

    SECTION("start from 4") {
        auto[dist1, p1] = g.Dijkstra(4);
        REQUIRE_THAT(dist1, Catch::Equals(std::vector{0, 0, 0, 0, 0, 0, 0, 0, 0,
0}));
        REQUIRE_THAT(p1, Catch::Equals(std::vector<unsigned>{0, 1, 2, 3, 4, 5, 6,
7, 8, 9}));

        auto[dist2, p2] = g.FordBellman(4, -1);
        REQUIRE_THAT(dist2, Catch::Equals(std::vector{0, 0, 0, 0, 0, 0, 0, 0, 0,
0}));
        REQUIRE_THAT(p2, Catch::Equals(std::vector<unsigned>{0, 1, 2, 3, 4, 5, 6,
7, 8, 9}));
    }
}

SECTION("undirected graph") {
    cpa::Graph<cpa::WeightedEdge<int>, int> g(6, false);
    g.addEdge(cpa::WeightedEdge(1, 3, 8));
    g.addEdge(cpa::WeightedEdge(1, 4, 9));

```

```

    g.addEdge(cpa::WeightedEdge(4, 3, 1));
    g.addEdge(cpa::WeightedEdge(1, 2, 12));
    g.addEdge(cpa::WeightedEdge(4, 2, 0));
    g.addEdge(cpa::WeightedEdge(4, 5, 25));
    g.addEdge(cpa::WeightedEdge(5, 3, 13));
    g.addEdge(cpa::WeightedEdge(2, 5, 4));

    auto[dist1, p1] = g.Dijkstra(3);
    REQUIRE_THAT(dist1, Catch::Equals(std::vector{0, 8, 1, 0, 1, 5}));
    REQUIRE_THAT(p1, Catch::Equals(std::vector<unsigned>{0, 3, 4, 3, 3, 2}));

    auto[dist2, p2] = g.FordBellman(3, -1);
    REQUIRE_THAT(dist2, Catch::Equals(std::vector{0, 8, 1, 0, 1, 5}));
    REQUIRE_THAT(p2, Catch::Equals(std::vector<unsigned>{0, 3, 4, 3, 3, 2}));
}

SECTION("graph with negative weight") {
    cpa::Graph<cpa::WeightedEdge<int>, int> g(9, true);
    g.addEdge(cpa::WeightedEdge<int>(0, 4, 1));
    g.addEdge(cpa::WeightedEdge<int>(5, 4, -9));
    g.addEdge(cpa::WeightedEdge<int>(4, 0, 2));
    g.addEdge(cpa::WeightedEdge<int>(4, 8, -3));
    g.addEdge(cpa::WeightedEdge<int>(8, 0, 2));
    g.addEdge(cpa::WeightedEdge<int>(0, 1, 4));
    g.addEdge(cpa::WeightedEdge<int>(1, 7, 3));
    g.addEdge(cpa::WeightedEdge<int>(7, 2, 1));
    g.addEdge(cpa::WeightedEdge<int>(2, 1, -5));
    g.addEdge(cpa::WeightedEdge<int>(7, 3, 2));
    g.addEdge(cpa::WeightedEdge<int>(3, 6, 7));
    g.addEdge(cpa::WeightedEdge<int>(6, 7, 6));
    const int NEG_INF = -10000;
    auto[dist, p] = g.FordBellman(4, NEG_INF);
    REQUIRE_THAT(dist, Catch::Equals(std::vector{-1, NEG_INF, NEG_INF, NEG_INF,
0, 0, NEG_INF, NEG_INF, -3}));
    REQUIRE_THAT(p, Catch::Equals(std::vector<unsigned>{8, 1, 2, 3, 4, 5, 6, 7,
4}));
}

TEST_CASE("Suffix automaton", "[suf_automaton]") {
    std::string s = "qwerty_q";
    cpa::SuffixAutomaton a(s);

    INFO("find all characters besides last")
    for (std::size_t i = 0; i + 1 < s.length(); ++i)
        REQUIRE(a.find(std::string(1, s[i])) == i);

    REQUIRE(a.find("qwe") == 0);
    REQUIRE(a.find("wer") == 1);
    REQUIRE(a.find(s) == 0);
    REQUIRE(a.find("_q") == 6);
    REQUIRE(a.find("") == 0);
    REQUIRE(a.find("z") == -1);
    REQUIRE(a.find(s + "asd") == -1);
    REQUIRE(a.find("qwerty_w") == -1);
    REQUIRE(a.find("erty2") == -1);

    auto pos_q = a.findAll("q");
    REQUIRE_THAT(pos_q, Catch::UnorderedEquals(std::vector{0, 7}));
    auto pos_empty = a.findAll("");
    REQUIRE_THAT(pos_empty, Catch::UnorderedEquals(std::vector{0, 1, 2, 3, 4, 5, 6,
7, 8}));
    SECTION("copy constructor") {
        auto a2 = a;

```

```

    REQUIRE(a2.find("qwe") == 0);
    REQUIRE(a2.find(s) == 0);
    REQUIRE(a2.find("_q") == 6);
    REQUIRE(a2.find(s + "asd") == -1);
    REQUIRE(a2.find("erty2") == -1);
}
SECTION("long string of equals character") {
    std::string sa(100, 'a');
    cpa::SuffixAutomaton a2(sa);
    REQUIRE(a2.find("a") == 0);
    std::vector<int> pos(sa.length());
    std::iota(pos.begin(), pos.end(), 0);
    REQUIRE_THAT(a2.findAll("a"), Catch::UnorderedEquals(pos));
}
}
TEST_CASE("Range query", "[data_structure]") {
    auto minimum = [](const int &a, const int &b) {
        if (a < b) {
            return a;
        }
        return b;
    };
    SECTION("handle tests") {
        std::vector data{12, 4, 6, -5, 2, 90};
        SECTION("array") {
            cpa::Array<int> ds(data.begin(), data.end(), minimum);
            for (std::size_t i = 0; i < data.size(); ++i) {
                REQUIRE(ds.valueOnSegment(i, i) == data[i]);
                REQUIRE(ds.getValueAt(i) == data[i]);
            }
            REQUIRE(ds.valueOnSegment(0, 5) == -5);
            REQUIRE(ds.valueOnSegment(0, 2) == 4);
            REQUIRE(ds.valueOnSegment(4, 5) == 2);

            ds.changeValueAt(3, 7);
            REQUIRE(ds.getValueAt(3) == 7);
            REQUIRE(ds.valueOnSegment(0, 5) == 2);
        }
        SECTION("sparse table") {
            cpa::SparseTable<int> ds(data.begin(), data.end(), minimum);
            for (std::size_t i = 0; i < data.size(); ++i) {
                REQUIRE(ds.valueOnSegment(i, i) == data[i]);
                REQUIRE(ds.getValueAt(i) == data[i]);
            }
            REQUIRE(ds.valueOnSegment(0, 5) == -5);
            REQUIRE(ds.valueOnSegment(0, 2) == 4);
            REQUIRE(ds.valueOnSegment(4, 5) == 2);
        }
        SECTION("segment tree") {
            cpa::SegmentTree<int> ds(data.begin(), data.end(), minimum);
            for (std::size_t i = 0; i < data.size(); ++i) {
                REQUIRE(ds.valueOnSegment(i, i) == data[i]);
                REQUIRE(ds.getValueAt(i) == data[i]);
            }
            REQUIRE(ds.valueOnSegment(0, 5) == -5);
            REQUIRE(ds.valueOnSegment(0, 2) == 4);
            REQUIRE(ds.valueOnSegment(4, 5) == 2);
            ds.changeValueAt(3, 7);
            REQUIRE(ds.getValueAt(3) == 7);
            REQUIRE(ds.valueOnSegment(0, 5) == 2);
        }
        SECTION("fenwick tree") {
            cpa::FenwickTree<int> ds(data.begin(), data.end(), minimum);
            for (std::size_t i = 0; i < data.size(); ++i) {

```

```

        REQUIRE(ds.valueOnSegment(i, i) == data[i]);
        REQUIRE(ds.getValueAt(i) == data[i]);
    }
    REQUIRE(ds.valueOnSegment(0, 5) == -5);
    REQUIRE(ds.valueOnSegment(0, 2) == 4);
    REQUIRE(ds.valueOnSegment(4, 5) == 2);
    ds.changeValueAt(5, -80);
    REQUIRE(ds.getValueAt(5) == -80);
    REQUIRE(ds.valueOnSegment(0, 5) == -80);
}
SECTION("sqrt decomposition") {
    cpa::SqrtDecomposer<int> ds(data.begin(), data.end(), minimum);
    for (std::size_t i = 0; i < data.size(); ++i) {
        REQUIRE(ds.valueOnSegment(i, i) == data[i]);
        REQUIRE(ds.getValueAt(i) == data[i]);
    }
    REQUIRE(ds.valueOnSegment(0, 5) == -5);
    REQUIRE(ds.valueOnSegment(0, 2) == 4);
    REQUIRE(ds.valueOnSegment(4, 5) == 2);
    ds.changeValueAt(3, 7);
    REQUIRE(ds.getValueAt(3) == 7);
    REQUIRE(ds.valueOnSegment(0, 5) == 2);
}
SECTION("treap") {
    cpa::Treap<int> ds(data.begin(), data.end(), minimum);
    for (std::size_t i = 0; i < data.size(); ++i) {
        REQUIRE(ds.valueOnSegment(i, i) == data[i]);
        REQUIRE(ds.getValueAt(i) == data[i]);
    }
    REQUIRE(ds.valueOnSegment(0, 5) == -5);
    REQUIRE(ds.valueOnSegment(0, 2) == 4);
    REQUIRE(ds.valueOnSegment(4, 5) == 2);

    ds.changeValueAt(3, 7);
    REQUIRE(ds.getValueAt(3) == 7);
    REQUIRE(ds.valueOnSegment(0, 5) == 2);
}
}
SECTION("random static tests") {
    std::vector<int> data(GENERATE(take(100, random(1, 100))));
    for (auto &val : data) {
        val = GENERATE(take(1, random(-1000, 1000)));
    }
    std::vector<cpa::ISegmentSolver<int>*> testing_struct{
        new cpa::Array<int>(data.begin(), data.end(), minimum),
        new cpa::SparseTable<int>(data.begin(), data.end(), minimum),
        new cpa::SegmentTree<int>(data.begin(), data.end(), minimum),
        new cpa::FenwickTree<int>(data.begin(), data.end(), minimum),
        new cpa::SqrtDecomposer<int>(data.begin(), data.end(), minimum),
        new cpa::Treap<int>(data.begin(), data.end(), minimum),
    };
    for (auto &ds : testing_struct) {
        for (std::size_t l = 0; l < data.size(); ++l) {
            auto mn = data[l];
            for (std::size_t r = l; r < data.size(); ++r) {
                mn = minimum(mn, data[r]);
                REQUIRE(ds->valueOnSegment(l, r) == mn);
            }
        }
        delete ds;
    }
}
}
}

```