Структура данных для представления текста в текстовом редакторе

Шильдяков А.В.

ИУ9-121

Оглавление

[Введение 4](#__RefHeading__470_1909124723)

[1 Аналитическая часть 5](#__RefHeading__13906_910725910)

[2 Научно-исследовательская часть 6](#__RefHeading__472_1909124723)

[2.1 Структура данных 6](#__RefHeading___Toc14420_1050734236)

[2.1.1 Построение структуры из исходного текста 7](#__RefHeading___Toc14422_1050734236)

[2.1.2 Вставка нового текста 8](#__RefHeading___Toc14424_1050734236)

[2.1.3 Удаление фрагмента текста 10](#__RefHeading___Toc14426_1050734236)

[2.1.4 Получение фрагмента текста 11](#__RefHeading___Toc19037_1050734236)

[2.1.5 Разделение дерева на две части по определённому символу 13](#__RefHeading___Toc12040_1050734236)

[2.1.6 Объединение двух AVL-сбалансированных деревьев вида Rope 14](#__RefHeading___Toc12042_1050734236)

[2.2 Параллельный лексический анализатор 17](#__RefHeading___Toc14430_1050734236)

[2.2.1 Получение лексем фрагмента текста 18](#__RefHeading__7954_243819943)

[2.2.2 Определение таблицы переходов 19](#__RefHeading__7956_243819943)

[3 Проектно-конструкторская часть 22](#__RefHeading__478_1909124723)

[3.1 Структура данных хранения текста 22](#__RefHeading__7958_243819943)

[3.2 Параллельный лексический анализатор 22](#__RefHeading__7960_243819943)

[3.3 Интерфейс пользователя 22](#__RefHeading__7962_243819943)

[3.3.1 Отображение текста 22](#__RefHeading__7964_243819943)

[3.3.2 Редактирование текста 24](#__RefHeading__9343_243819943)

[3.3.3 Управление курсором 25](#__RefHeading__7966_243819943)

[3.3.4 Работа с буфером обмена 28](#__RefHeading__9345_243819943)

[4 Технологическая часть 30](#__RefHeading__484_1909124723)

[4.1 Методология разработки и используемые средств 30](#__RefHeading__7970_243819943)

[4.2 Руководство пользователя 30](#__RefHeading__7972_243819943)

[5 Организационно-экономическая часть 31](#__RefHeading__488_1909124723)

[5.1 Введение 31](#__RefHeading__3609_1816927962)

[5.2 Организация и планирование процесса разработки программы 31](#__RefHeading__7564_822303622)

[5.2.1 Техническое задание 31](#__RefHeading___Toc15830_745179007)

[5.2.2 Расчёт стоимости проекта 32](#__RefHeading__7566_822303622)

[5.2.3 Затраты на выплату исполнителям 34](#__RefHeading___Toc15832_745179007)

[5.2.4 Определение количества исполнителей 49](#__RefHeading__7568_822303622)

[5.2.5 Календарный график выполнения работ 50](#__RefHeading__7570_822303622)

[5.3 Расчёт сметы затрат 53](#__RefHeading__7572_822303622)

[5.3.1 Затраты на выплату исполнителям 53](#__RefHeading__3611_1816927962)

[5.3.2 Затраты на оборудование 56](#__RefHeading__7576_822303622)

[5.3.3 Затраты на организацию рабочих мест 59](#__RefHeading__7578_822303622)

[5.3.4 Накладные расходы 60](#__RefHeading__7580_822303622)

[5.3.5 Суммарные затраты 60](#__RefHeading__7582_822303622)

[5.4 Цена продукта 61](#__RefHeading__7584_822303622)

[5.5 Дисконтирование дохода 62](#__RefHeading__7586_822303622)

[5.6 Вывод 64](#__RefHeading__6943_348067787)

[Заключение 65](#__RefHeading__490_1909124723)

[Приложение А. Диаграмма Ганта выполненных работ 68](#__RefHeading__4092_62578531)

# Введение

В настоящее время большую популярность набирают веб-приложения, работающие через Интернет и предоставляющие совместный доступ нескольких пользователей к одним и тем же данным, мгновенное сохранение изменений и обработку в «облаке». Помимо указанных положительных свойств использование таких приложений позволяет абстрагироваться от практически любой платформы и операционной системы устройства, поскольку для работы таких приложений необходим лишь доступ к Интернету и браузер, который уже реализован на всех платформах.

Существующие текстовые редакторы предполагают работу с малым объёмом текста. Подсветка программного исходного кода в таких текстовых редакторах на больших файлах работает достаточно долго. Такое можно наблюдать при одновременном раздельном просмотре одного файла при создании многострочного комментария.

В данной работе проводится исследование по определению оптимального типа данных для хранения и изменения большого количества текста, а также для возможности быстрой обработки данных для подсветки программного кода. Предполагается создание структуры данных, тестирование работоспособности и создание работающего примера веб-приложения для демонстрации скорости обработки файла размером 100 МБ. Также будут проведено сравнение скорости работы данного примера программы с другими текстовыми редакторами. Эти текстовые редакторы могут быть как веб так и нативными приложениями.

# Аналитическая часть

# Научно-исследовательская часть

Поставленную задачу разработки можно разделить на три составляющие:

* структура данных для представления текста в текстовом редакторе, выполняющая основные операции по вставке, получению и удалению текста;
* лексический анализатор и дополнения в структуре для поддержки получения лексем текста;
* пользовательский интерфейс, отображающий текстовый редактор и использующий созданную структуру.

Описание структуры данных, лексического анализатора и доказательство асимптотической сложности приведены в этой главе. Описание архитектуры приложения и пользовательского интерфейса описаны в следующей главе 3 Проектно-конструкторская часть.

## Структура данных

Данная структура должна поддерживать следующие операции:

* построение структуры на основе исходного текста
* вставка нового текста в любое место исходного текста
* удаление фрагмента исходного текста
* получение информации о типах лексем выбранного фрагмента текста

Особенностью данной структуры должна являться быстрая обработка части текста большого размера, также позволяющая быстро реагировать на изменения в тексте. Быстрота реагирования определяется асимптотической сложностью операций, и в техническом задании для новой структуры указаны значения, приведённые в таблице 1. Можем заметить, что асимптотическая сложность операций определяемой структуры очень похожа на сложность операций при работе со сбалансированным деревом.

Исходя из требований предлагается использовать структуру данных Rope с балансировкой AVL-дерева. Структура данных Rope представляет собой бинарное дерево в листах которого хранятся подстроки исходной строки. В AVL сбалансированном дереве для каждой вершины должно быть выполнено правило после окончания операции и перестроения дерева: высота поддеревьев‑потомков вершины не должна отличаться больше, чем на 1.

Асимптотическая сложность операций представлена в таблице 1. В формулах используется следующее обозначение:

* K — количество символов, с которыми производится операция вставки/удаления;
* N — количество символов в исходном тексте

|  |  |
| --- | --- |
| Операция | Сложность |
| Построение структуры |  |
| Вставка текста |  |
| Удаление фрагмента |  |
| Получение фрагмента |  |
| Получение лексем |  |

Таблица 1 — Асимптотическая сложность операций

Рассмотрим алгоритм операций и докажем асимптотическую сложность требуемых методов.

### Построение структуры из исходного текста

Алгоритм:

1. Разбиваем строку на подстроки определённой величины
2. Строим листья из этих подстрок
3. Для каждых двух вершин на текущем уровне строим вершину — родитель.
4. Когда все вершины-родители текущего уровня построены — переходим на уровень выше.
5. Повторяем пункты 3 и 4 до достижения первого уровня — корня дерева.

Операция построения дерева из строки имеет асимптотическую сложность , где K — количество символов в тексте. Докажем это:

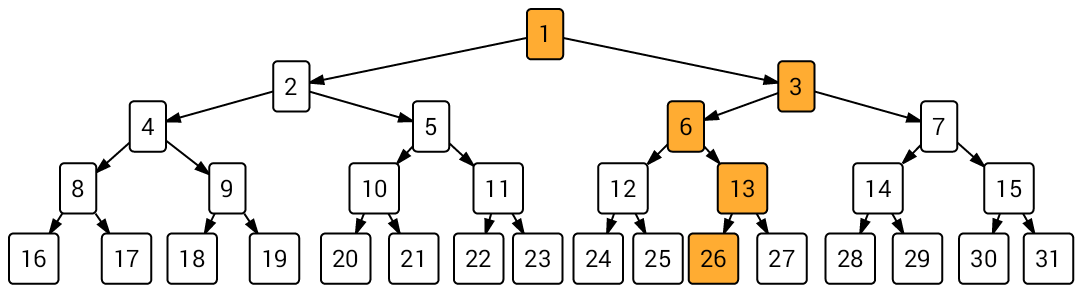
1. Выделение подстрок в листья занимает операций, где K — количество символов, M — количество символов в листе
2. Количество операций построения вершин-родителей рассчитывается по формуле:

В итоге асимптотическая сложность построения дерева равна

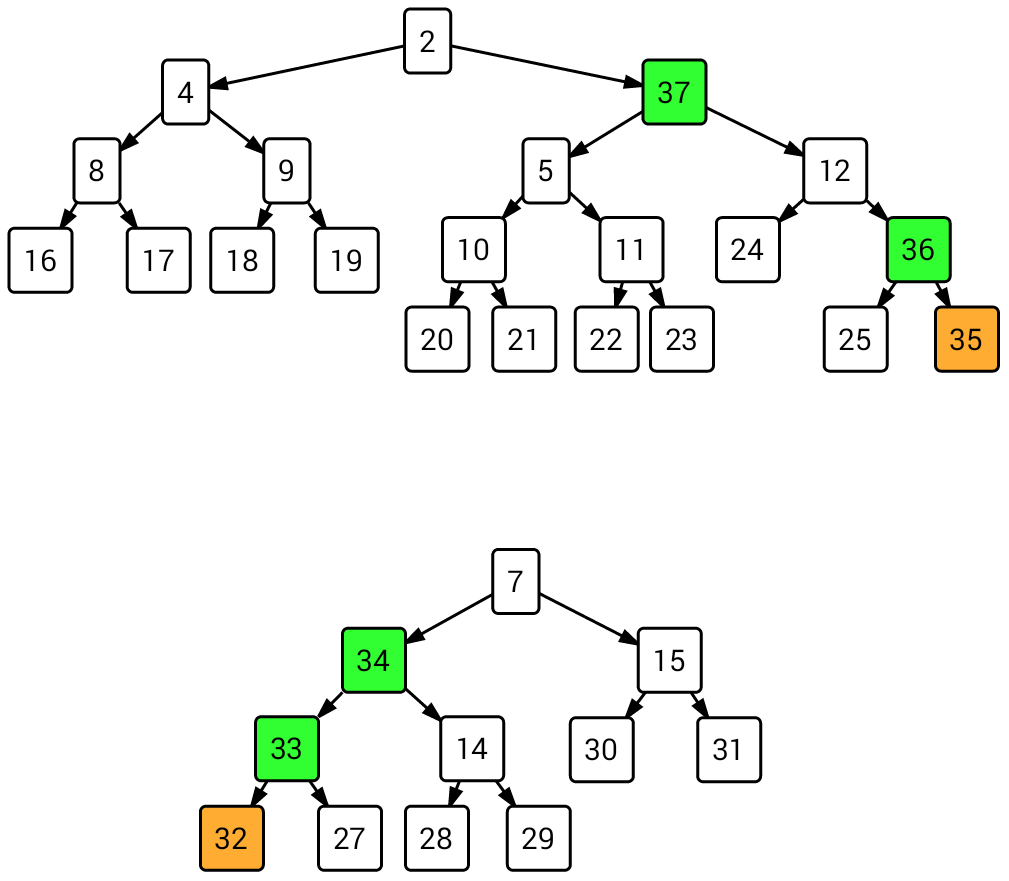
### Вставка нового текста

Алгоритм:

1. Для вставляемой строки строим своё rope-дерево
2. Исходное дерево (рисунок 1) при поиске листа, куда нужно вставить новый текст, разделяем на два новых дерева, которые также являются сбалансированными (доказано в параграфе 2.1.5 Разделение дерева на две части по определённому символу) как показано на рисунке 2.

Рисунок 1 – Путь в исходном дереве до листа, в котором находится разделяемый символ

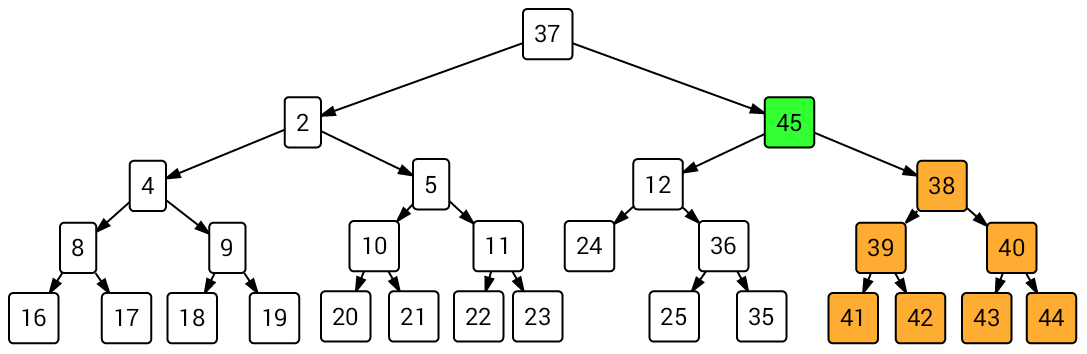
1. Левое новое дерево объединяем с вставляемым как показано на рисунке 3.

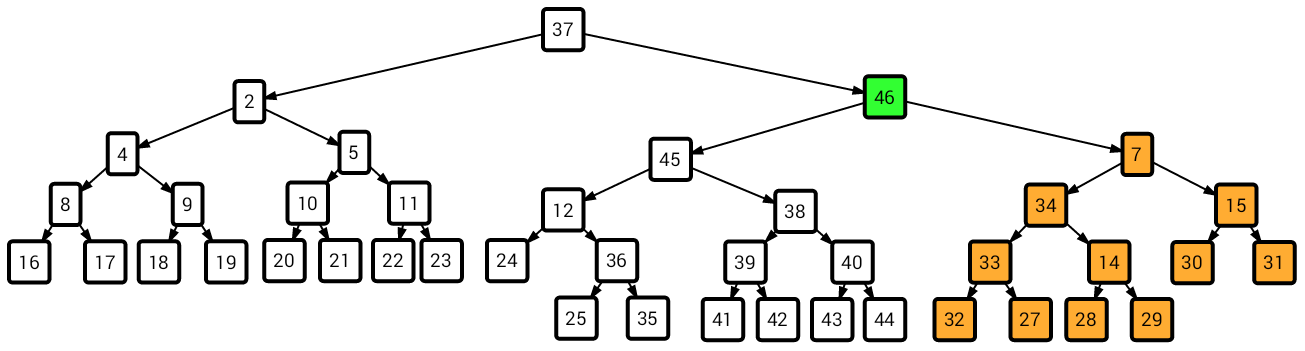
Рисунок 2 – Два дерева, полученных разделением по листу 26 из исходного

1. Полученное дерево объединяем с правым новым деревом как показано на рисунке 4.

Операция вставки текста имеет асимптотическую сложность , где N — количество символов в исходном тексте, а K — количество символов во вставляемом тексте. Пункты при рассмотрении асимптотик совпадают с пунктами алгоритма.

1. По ранее рассмотренному алгоритму, асимптотическая сложность равна
2. Данная операция имеет асимптотическую сложность, поскольку каждая вершина на пути от корня до листа используется только однажды. Длина пути при этом равна высоте дерева, которая и является

Рисунок 3 – Объединение левого поддерева с вставляемым

Рисунок 4 – Объединение дерева с правым поддеревом

1. Асимптотическая сложность операции объединения, что доказывается в параграфе 2.1.6 Объединение двух AVL-сбалансированных деревьев вида Rope.
2. Аналогично предыдущему пункту —

### Удаление фрагмента текста

Алгоритм:

1. Разделяем исходное дерево на два поддереваипо символу, определяющему начало фрагмента.
2. Разделяем поддеревона два других поддереваипо символу, определяющему конец фрагмента.
3. Отбрасывая дерево, объединим поддеревьяи . Таким образом мы получим новое дерево с удалённым фрагментом.

Докажем, что асимптотическая сложность данной операции равна. Пункты при рассмотрении асимптотик совпадают с пунктами алгоритма.

1. Операция разделения имеет сложность, что доказано в параграфе 2.1.5 Разделение дерева на две части по определённому символу
2. Аналогично предыдущему пункту сложность
3. Сложность объединения двух поддеревьев имеет сложность . Можно отметить, что , таким образом

Таким образом сложность выполнения всех операций составит .

### Получение фрагмента текста

Алгоритм:

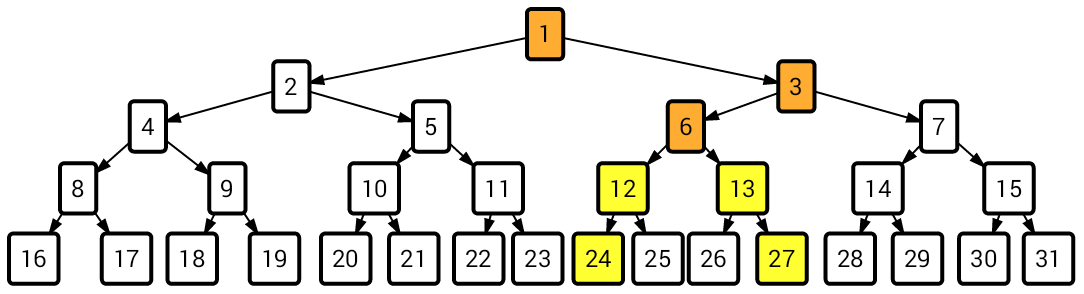
1. Находим лист в дереве, в котором располагается первый символ выбранного фрагмента текста. Получаем подстроку в данном листе, соответствующую началу фрагмента.
2. Ищем следующий по порядку лист и добавляем в общую подстроку.
3. Повторяем пункт 2 до достижения листа, в котором располагается последний символ выбранного фрагмента текста.

Таким образом мы получаем все необходимые подстроки выбранного фрагмента и, соответственно, сам фрагмент.

Докажем, что асимптотическая сложность данной операции составляет .

Данная операция разделяется на два действия. Во-первых, поиск листа, который имеет асимптотическую сложность. Во-вторых, на процедуру перебора листьев, сложность которой мы рассмотрим.

При переборе листьев дерева мы просматриваем все вершины поддерева, лежащего между первым и последним листом. Корнем такого дерева является самая нижняя вершина V, общая для пути для первого и последнего листа, как показано на рисунке 5. При этом в каждой вершине, не являющейся листом, мы бываем дважды: когда идём вниз до листа и когда поднимаемся вверх для поиска следующего листа. Это поддерево можно рассматривать как дерево, количество листов которого не превышает , где K — количество символов в просматриваемом фрагменте, M — минимальное количество символов в листе. Как было доказано в параграфе 2.1.1 Построение структуры из исходного текста, количество вершин в таком дереве не превышает . Количество вершин в пути от корня исходного дерева до вершины V не превышает высоты этого дерева . Таким образом общее количество действий не превышает , асимптотическая сложность которых составляет .

Рисунок 5 – Общая часть двух путей

### Разделение дерева на две части по определённому символу

Ищем путь от корня дерева до листа, в котором располагается символ

1. На каждом уровне мы выбираем, в левом поддереве или в правом поддереве находится нужный лист. Отсекаем поддерево, в котором не лежит путь до листа, от основного дерева, как это показано на рисунке .
2. Повторяем операцию 1 до достижения листа, в котором располагается указанный символ. Разделим подстроку, располагающуюся в найденном листе, по указанному символу. Получим лист, расположенный левее, и лист, расположенный правее символа включительно.
3. Получаем множество поддеревьев, расположенных левее этого символа, и множество поддеревьев, расположенных правее указанного символа включительно.

Все эти поддеревья являются AVL‑сбалансированными по определению AVL-сбалансированности исходного дерева, откуда поддеревья были получены.

Для получения 2 деревьев, разделённых указанным символом, необходимо объединить поддеревья в соответствующих множествах. Результирующее дерево также будет AVL-сбалансированным по его построению.

Асимптотическая сложность данных операций составляет. Докажем это:

1. Путь до листа с указанным символом мы проходим, выполняя операций, где — высота исходного дерева. Сложность операций составляет
2. Объединение поддеревьев одного множества имеет асимптотическую сложность . Нам нужно объединять поддеревья в двух множествах, поэтому сложность равна . Докажем сложность объединения.

Каждое поддерево в множестве расположим в последовательности, построенному по убыванию высоты дерева , поскольку каждое поддерево было построено разделением поддерева большей высоты. При этом высота каждого поддерева отличается от более верхнего минимум на один: .

Объединение вида «добавлением справа/слева» двух поддеревьев высотой и,имеет асимптотическую сложность , доказательство которого описывается в 2.1.6 Объединение двух AVL-сбалансированных деревьев вида Rope. Таким образом асимптотическая сложность получения AVL‑сбалансированного дерева из множества сбалансированных поддеревьев вычисляется по формуле:

Все операции имеют асимптотическую сложность

### Объединение двух AVL-сбалансированных деревьев вида Rope

Введём обозначения:

— первое AVL-сбалансированное дерево

— второе AVL-сбалансированное дерево

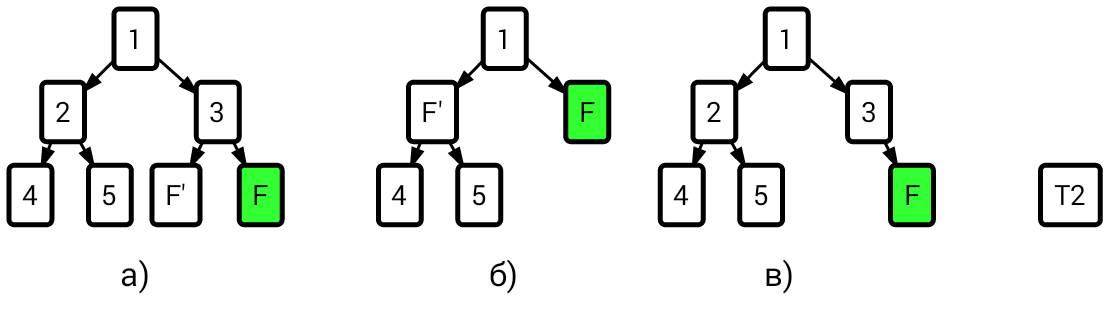
Без ограничения общности допустим, что , где — высота поддерева T.

Алгоритм объединения:

1. Находим соответствующую (слева или справа) вершину F в дереве, находящуюся на глубине (длина пути от корня дерева до вершины) как это показано на рисунке 6а.

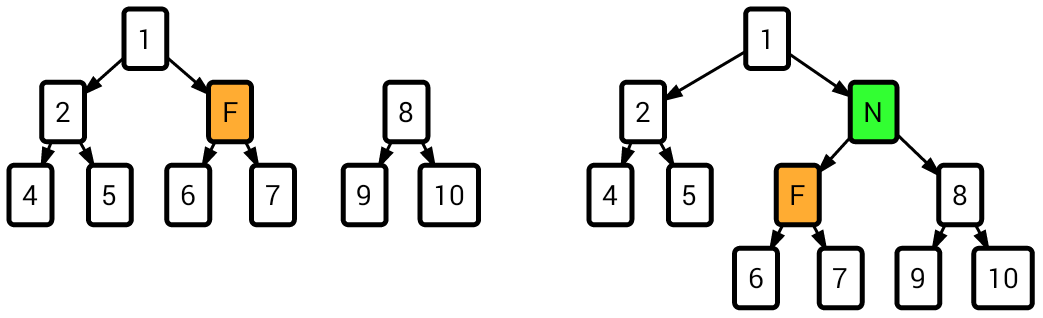
Найденное поддерево с корнем F назовём . Глубина вершины F: . Это связано с определением AVL‑сбалансированности, которое допускает отличие высот в потомках на 1. В этом случае потомка с нужной нам стороны не существует, и мы используем родителя, что видно на рисунке 6б.

Также возможен случай, когда высота поддерева— соседа , у которого тот же родитель, может иметь высоту, меньшую чем у : как это показано на рисунке 6в.

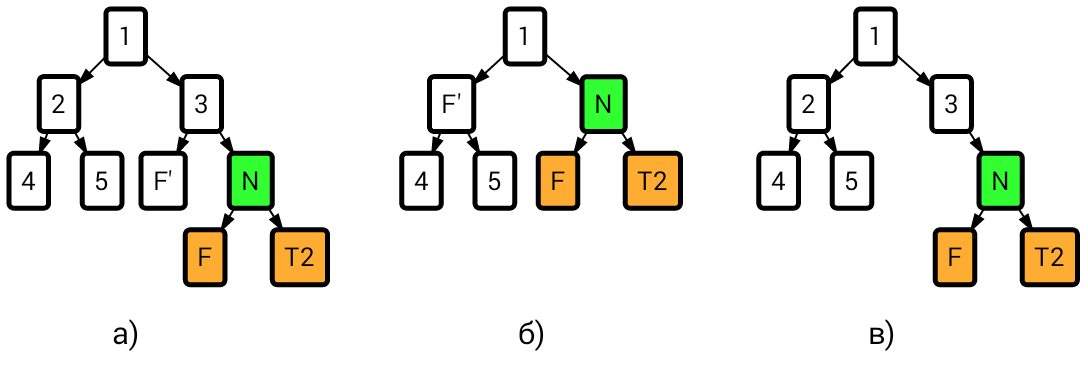
Рисунок 6 – Выбор вершины в левом поддереве для объединения

1. Создаём новую вершину N, назначая потомками вершину F (поддерево бо́льшего дерева ) и корень дерева . При этом вершину N расположим вместо вершины F в дереве (рисунок 7)
2. Высота нового, соответствующего, поддерева с корнем в вершине N может увеличится на 1 от предыдущего. На рисунке 8 показано новые деревья, соответствующие рисунку 7.

Возможен случай, когда поддерево имело высоту меньше, чем Тогда разница высот поддеревьев родителя будет равна 2 как в пункте в) рисунка. В этом случае потребуется балансировка.

Рисунок 7 – Создание вершины для объединения поддерева

1. После балансировки высота поддерева может также увеличиться на 1, по сравнению с исходной высотой, что потребует балансировку родителя. Таким образом может потребоваться каскадная балансировка вершин на пути от корня до вершины N.

Рисунок 8 – Варианты объединения поддерева

1. В результате получим объединённое AVL-сбалансированное дерево.

Докажем, что асимптотическая сложность операции объединения AVL-сбалансированных деревьев вида «добавление слева/справа» равна , где — высота поддерева T. Пункты при рассмотрении асимптотик совпадают с пунктами алгоритма.

1. Операция займёт не более действий и будет иметь асимптотическую сложность .
2. Количество действий является константным, поэтому асимптотическая сложность .
3. Каждая AVL-балансировка выполняется одинаковым константным количеством операций, поэтому асимптотическая сложность балансировки одной вершины равна .
4. Может потребоваться балансировок до корня большего дерева. Асимптотическая сложность данных действий будет равна

Сложив сложности, получим результирующую асимптотическую сложность объединения AVL-сбалансированных деревьев вида “добавление слева/справа” равной .

При этом высота полученного дерева может быть на 1 больше высоты двух объединяемых поддеревьев:

## Параллельный лексический анализатор

Лексический анализатор используется для определения лексем и соответствующей расцветке программного кода текста в интерфейсе пользователя. Особенностью данного лексического анализатора является быстрая работа на больших текстах. При этом любое изменение текста должно укладываться в асимптотическую сложность и изменение лексем не должно превышать это значение. Таким образом определение лексемы для фрагмента не должно превышать по времени получение фрагмента этого текста, сложность которой составляет , где N — количество символов в исходном тексте, а K — количество символов фрагмента, лексемы которого можно получить. Это означает, что получение лексем для фрагмента текста не должно превышать линейное время .

В качестве лексического анализатора мы будем использовать конечный детерминированный автомат. Для того, чтобы определить лексемы фрагмента, нам необходимо знать состояние автомата на первом символе этого фрагмента. При известном начальном состоянии определение лексем займёт линейное время ввиду использования детерменированного автомата.

Чтобы найти начальное состояние мы будем использовать таблицу переходов состояний в каждой вершине. Таблица переходов должна определять, какое состояние будет иметь конечный автомат после обработки последнего символа текста, определённого поддеревом с корнем, являющейся данной вершиной.

### Получение лексем фрагмента текста

Алгоритм получения лексем в таком случае следующий:

1. Текущее состояние автомата на начале текста считаем нулевым.
2. Определяем, в левом или правом поддереве находится первый символ фрагмента получаемого текста. Если в правом, то определяем состояние автомата на начале текста, определяемого в правом поддереве, используя таблицу переходов соседней вершины — корня левого поддерева.
3. Таким образом при достижении листа, в котором располагается первый символ фрагмента, мы определили состояние автомата на начале текста, определяемого этим листом. Используя конечный автомат, найдём его состояние на начало фрагмента.
4. Используя конечный автомат получаем состояния на каждом символе. В случае, когда состояние является конечным — мы нашли лексему.

Докажем асимптотическую сложность, равную:

1. Поиск состояния автомата на начало текста в листе, в котором расположен первый символ фрагмента занимает, потому что мы посещаем вершины дерева, составляющие путь от корня до найденного листа.
2. Поиск состояния автомата на начало фрагмента в листе не может занять более M операций, где M — максимальное количество символов в листе. Это значение является константой, поэтому сложность составит
3. Для того чтобы определить лексемы достаточно иметь фрагмент текста и начальное состояние автомата. Получение фрагмента текста, как определено в 2.1.4 Получение фрагмента текста имеет сложность
4. В итоге, все операции имеют сложность

### Определение таблицы переходов

Таблица переходов в листе определяется по тексту в этом листе. Таблица переходов в вершине, не являющейся листом, определяется как суперпозиция таблиц переходов прямых потомков по формуле:

, где:

* — левое поддерево указанной вершины
* — правое поддерево указанной вершины
* — таблица переходов для указанного поддерева с вершиной T

Рассмотрим принцип построения таблицы переходов для листа. Она может быть построена последовательным перебором: используя различные начальные состояния мы можем просматривать каждый символ текста в листе. Количество операций в таком случае равно , где:

* M — максимальное количество символов в листе,
* S — количество состояний лексического анализатора.

Асимптотическая сложность данной операции , потому что— количество состояний не зависит от текста, а зависит от языка лексического анализатора. Линейная сложность имеет большую константу и для языка C составляет порядка 450 состояний. Но мы уменьшим эту константу, оптимизировав построение таблицы переходов и используя следующую эвристику:

* в случае большого M — количества символов в листе — количество возможных конечных состояний достаточно мало.

Представим следующий алгоритм:

1. На первом символе конечный автомат может перейти изразличных начальных состояний вразличных состояний, где
2. На втором символе конечный автомат может перейти изразличных начальных состояний вразличных состояний, где
3. Повторим действия до завершения строки

Таким образом количество различных состояний на символах текста составляет следующую последовательность:

Большинство ключевых слов и символов заканчиваются внутри текста, что означает что практически все первоначальные состояния придут в конечное ещё в середине текста. Это, в свою очередь, означает что большинство состояний начинаются с нулевого в середине текста с k-го символа. Таким образом мы можем перебирать не всесостояний на каждом символе, а лишь ограниченноесостояний. Это уменьшит константу, но асимптотическая сложность останется линейной. Последовательность различных состояний на символах текста будет:

Общая асимптотическая сложность определения таблиц переходов для всех вершин при первоначальном построении переходе составляет , где N — количество символов в тексте.

Описать конкретный алгоритм с массивами

# Проектно-конструкторская часть

## Структура данных хранения текста

## Параллельный лексический анализатор

Структура лексического анализатора формируется автоматически на основе лексического описания языка Lex с помощью программы курсовой работы. [ссылка] В результате работы указанной программы формируется готовый исходный код однопоточного лексического анализатора на языке Go. Этот код, во-первых, преобразуется в код программы на JavaScript, во-вторых, добавляется параллельный анализ для составления таблицы переходов определённой строки.

Структура формируемого лексического анализатора достаточно простая. Поскольку анализатор работает с использованием конечного детерминированного автомата, требуется лишь несколько таблиц — массивов, определяющих состояния и переходы между ними и дополнительную информацию о лексемах:

* Таблица определения конечных состояний и соответствующих им доменов лексем. Размер таблица определяется количеством состояний конечного автомата анализатора.
* Таблица переходов для каждого состояния, определяющая рёбра переходов между всеми состояниями анализатора. Таблица является двумерным массивом. Количество массивов на первом уровне определяется количеством состояний, на втором уровне — количеством рёбер переходов.
* Таблица определения номера ребра перехода по символу перехода. Размер таблицы равен 256 — количество ASCII символов, которые могут использоваться для определения программы на языке ANSI C. Данная таблица служит для сжатия размеров таблиц переходов.

Конечным результатом работы лексического анализатора является список подстрок, определяющих лексемы, и соответствующим им доменов лексем. Поэтому мы используем таблицу соответствия доменов лексем, пропуская шаг определения конкретной лексемы и устраняя дополнительную таблицу соответствия.

Поскольку исходная программа лексического анализатора [ссылка] предполагает частью алфавита исходного языка любой Unicode символ, в ней используется сжатие количества рёбер переходов между состояниями. Данное сжатие предполагает наличие и работу только с одним ребром (переходом) для группы символов. Это позволяет существенно в несколько десятков сократить количество переходов — от нескольких десятков тысяч символов Unicode до пары сотен групп символов с одним переходом. Группа символов определяется как начало и конец последовательности символов. Такое объединение должно удовлетворять правилу: переход по любому символу группы от любого состояния конечного автомата должен вести в одно и то же состояние конечного автомата.

Для поиска ребра в исходной программе используется функция, которая определяет принадлежность символа определённой группе. Такой поиск имеет асимптотическую сложность , где — количество групп символов, что является существенным при работе нашего лексического анализатора. Для уменьшения такой сложности и ввиду гораздо меньшего алфавита определения программ языка ANSI C мы будем использовать массив прямого поиска соответствия, где каждому символу будет соответствовать ячейка в массиве, определяемая кодом символа. Значением ячейки будет являться номер ребра, используемый в таблице переходов. В таком случае асимптотическая сложность определения номера ребра перехода становится равной, что в несколько сотен раз меньше исходного алгоритма в абсолютном значении.

При использовании такого сжимающего алгоритма количество рёбер переходов изменится несущественно — меньше, чем в 2 раза — с 255 до 160. Использование же обусловлено сжатием таблицы переходов, где общее количество переходов по всем рёбрам (представляя конечный автомат в виде полного графа) составляет , где— количество групп символов, — количество состояний. Это позволит уменьшить количество используемой памяти.

Работу лексического анализатора можно разделить на 2 основные функции:

* получение списка лексем, соответствующих определённой строке, и равнозначно определение состояния конечного автомата на определённом символе строки;
* определение таблицы переходов для строки.

Первый алгоритм достаточно простой:

1. Для каждого символа определяем следующее состояние конечного автомата по таблице переходов.
2. Если состояние является ошибочным, значит мы полностью распознали лексему, иначе переходим к пункту 1.
3. При определении лексемы смотрим таблицу соответствия конечного состояния, предшествующего ошибочному, домену лексемы.
4. После определения лексемы, продолжаем алгоритм с начального состояния.

При определении лексем возникает сложность с многострочными комментариями или длинными строками. Данная особенность возникает ввиду отображения только видимой части строки, в которую может входить часть комментария или длинной строки. Поскольку определение начала лексемы происходит со стопроцентной точностью и имеет асимптотикув связи с определённой структурой хранения данных, состояние конечного автомата на начало отображаемой подстроки будет определено верно. Для определение же лексемы длинной строки или многострочного комментария потребуется просмотреть все символы до конца этой лексемы, что имеет линейную асимптотическую зависимость от количества символов в этой лексеме и в общем случае зависит от длины всего текста, что неприемлимо.

Быстрым решением является некоторая эвристика, которая не гарантирует абсолютное определение лексемы, но позволяет идентифицировать многострочные комментарии или длинные строки на основе части этих лексем. Для этого промежуточные состояния комментария и строки определим как конечные. Это возможно благодаря установленным лексическим правилам языка C. Такая эвристика поможет верно определять лексемы комментариев и строк, но не гарантирует определение ключевых слов языка при частичном их отображении. Для улучшения возможно указание промежуточных состояний других лексем как конечные, в случае если они находятся на пути, определяющем лексемы только одного домена.

Применение такой эвристики, конечно, является нарушением классической работы лексического анализатора и возможность применения может быть ограничена лишь цветовой синтаксической подсветки исходного кода программ. Такая эвристика может не работать для выполнения других задач.

Для универсального разрешения данной ситуации вне зависимости от языка, для которого строится лексический анализатор и определяемых лексем предлагается возможность добавления дополнительной информации в структуру хранения текста, являющейся не только таблицей переходов состояний для подстроки, представленной вершиной дерева, но и таблицей переходов в ближайшее конечное состояние. То есть начальному состоянию должен соответствовать домен первой лексемы, находящейся в представленной данной вершиной подстроке, если бы распознавание начиналось с этого состояния. Обновление данной таблицы должно быть аналогично обновлению таблицы переходов. При этом асимптотическая сложность операций будет совпадать с таковой для таблицы переходов, а значит не изменит сложность операций над всей структурой данных хранения текста.

### Параллельное построение таблицы переходов

Базовый принцип второго алгоритма описан в главе 2.2.2 Определение таблицы переходов. Для работы алгоритма потребуется список массивов, целью которых является объединение перехода от нескольких различных состояний в одно. Пример такого списка массивов представлен на рисунке. В списке выполняются следующие правила:

* значения элементов последнего массива списка являются состояниями автомата на текущем обрабатываемом символе,
* значение элементов предыдущих массивов являются индексом элемента последующего массива, определяющего состояние в которое переходит группа состояний, определяемых текущим элементом
* каждый последующий массив имеет количество элементов строго меньшее, чем предыдущий

Опишем алгоритм построения:

1. Определяем значения первого массива размером совпадающим с количеством состояний совпадающим с индексом элемента в массиве.
2. Создаём последующий пустой массив
3. Для каждого состояния последнего значимого массива по текущему символу определяем новое состояние.
4. При необходимости добавляем в последний, наполняемый, массив переходное состояние.
5. Изменяем значение обрабатываемого элемента (состояния) последнего значимого массива на индекс соответствующего элемента в последнем, наполняемом, массиве.
6. Если количество элементов в новом, наполняемом, массиве совпадает с количеством элементов предыдущего, значимого, массива — переписываем значения предыдущего массива на соответствующие значения нового массива. Удаляем новый массив
7. Повторяем до завершения обработки строки

Пункт 6 описанного алгоритма позволяет сократить используемую память, поскольку сохраняет только массивы, размеры которых составляют строго уменьшающуюся последовательность. В упрощённом случае количество массивов было бы равно количеству символов в обрабатываемой строке.

Архитектура - классы

## Интерфейс пользователя

Для разработки пользовательского интерфейса будем использовать библиотеку ReactJS. Особенностью данной библиотеки является быстрое отображение интерфейса при его изменениях. Такое быстродействие обеспечивается за счёт построения виртуального DOM-дерева и практически линейного построения разницы между новым и старым интерфейсом благодаря использованию некоторых эвристик [ссылка]. Использование такого алгоритма также упрощает программирование. Благодаря автоматическому рассчёту изменений достаточно указывать полные данные для отображения. В случае их изменений между различными состояниями, ReactJS самостоятельно отобразит их.

Использование библиотеки также накладывает определённые правила на архитектуру приложения. Необходимо использовать классы ReactJS, переопределяя методы render() для отображения интерфейса. Рассмотрим построение архитектуры приложения.

### Отображение текста

Текст представляет собой набор строк. Каждая строка может иметь длину, превышающую количество колонок для отображения. Количество строк текста также может превышать количество отображаемых строк. Таким образом отображается часть текста, ограниченная определённым окном отображения. Окно отображения зададим следующими параметрами:

* количество отображаемых строк (высота отображаемого окна),
* количество отображаемых символов (ширина отображаемого окна),
* порядковый номер в тексте первой строки окна отображения,
* порядковый номер в тексте первого символа окна отображения.

Таким образом алгоритм отображения окна является простым. Требуется получить отображаемые подстроки для всех отображаемых строк, находящихся в окне отображения. Таким же образом можем получить список лексем для заданного окна отображения для реализации синтаксической подсветки. Ограничение на получение лишь подстрок, а не всей строки введено ввиду возможного наличия очень длинных строк в отображаемом тексте. И в случае получения всей строки может расходоваться дополнительные ресурсы на отображение ненужной информации.

Основной сложностью является изменение координат окна отображения при навигации по тексту. Окно отображения напрямую связано с позицией курсора. Алгоритм должен удовлетворять основному правилу: курсор должен находится в окне отображения.

Встроенная в браузер реализация многострочных текстовых полей в HTML с помощью тега <input type=textarea> не позволит реализовать синтаксическую подсветку кода, поскольку не позволяет использовать теги внутри. Поэтому мы будем использовать большой div-контейнер для всего текстового поля. Этот контейнер будет содержать несколько div-контейнеров, представляющих отдельную строку. Для реализации использования различных цветовых стилей для лексем будем использовать тег <span> с назначенным CSS-классом, определённым для каждого домена лексемы. В этом случае определение цветовой схемы будет дано в отдельном CSS-файле, что упростит его изменение.

Для удовлетворения требования по обработке больших текстовых документов размером более 100 МБ мы будет отображать не весь текст документа, а текст лишь в окне отображения. В этом случае мы не будем использовать встроенные в браузер механизмы отображения полос прокрутки.

### Редактирование текста

Как было определено ранее, мы не будем использовать встроенное в браузер текстовое поле, поддерживающее изменение текста, а будем использовать вложенные div-контейнеры. В HTML5 был введен атрибут contenteditable, определённых для всех тегов. В случае наличия в них текста, содержимое можно редактировать, как при использовании встроенного текстового поля textarea. Плюсом в такой реализации является поддержка всех внутренних тегов, а значит и возможность цветового оформления лексем текста.

Ввиду отображения текстового поля с помощью ReactJS, изменения в тексте должны производиться в последовательности, представленной на рисунке. Поэтому все события клавиатуры, связанные с изменением текста будут прерывать стандартную реакцию браузера (например, отображение нового символа или удаление). Как это могло бы быть без использования ReactJS можно увидеть на рисунке.

В остальном реализация функционала редактирования текста не представляет сложности, ввиду распределения функций между другими классами:

1. Получение события изменения текста
   * добавление символа или вставляемой подстроки
   * удаление символа или выделенной подстроки
   * изменение выделенной подстроки
2. Изменение текста в структуре хранения
3. Изменение положения курсора
4. При необходимости изменение окна отображения
5. Отображение текста, находящегося в окне отображения с помощью ReactJS

### Управление курсором

ReactJS берёт на себя функцию отображения содержимого, но не занимается управлением курсора. Поскольку получение текста для отображения является достаточно трудоёмкой операцией, а при перемещении курсора в пределах окна отображения текст не изменяется, было решено использовать нативную, встроенную в браузер, поддержку курсора, которая представлена классом Selection. Данный класс также представляет выделение текста, определяемого координатами начала и конца выделения. Курсор в данном случае представляется вырожденным выделением нулевой длины, в котором координаты начала и конца выделения совпадают.

Особенностью реализации текстового редактора является отсутствие автоматического переноса слов (Word wrap) и, соответственно, наличие окна отображения. Встроенный класс Selection оперирует координатами отображаемого документа, в то время, как мы должны оперировать абсолютными координатами курсора и выделения в тексте. Для разрешения данной ситуации мы будем использовать дополнительный класс AbsoluteSelection, который будет отвечать за определение абсолютных координат выделения.

В то время как класс AbsoluteSelection должен получать информацию при любых изменениях курсора и выделения, непосредственно само отображение курсора и выделения может обновляться встроенным в браузер механизмом в случае, если изменения происходят в окне отображения и не изменяют его. Мы используем это условие для определения необходимости обновления фактического курсора и выделения. Это позволяет сократить количество выполняемых операций с учётом необходимости их выполнения, что является принципом аналогичным в используемой библиотеке ReactJS.

Рассмотрим возможные операции изменения местоположения курсора и соответствующие им клавиши:

* перемещение курсора влево (Left);
* перемещение курсора вправо (Right);
* перемещение курсора на начало строки (Home);
* перемещение курсора на конец строки (End);
* перемещение курсора вниз (Down);
* перемещение курсора вверх (Up);
* перемещение курсора на несколько строк вверх (PageUp);
* перемещение курсора на несколько строк вниз (PageDown);
* указание положения курсора в пределах окна отображения с помощью мыши.

Изменение позиции курсора происходит не только при непосредственных действиях по их изменению, но и косвенно при выполнении любых операций изменения текста: вставке и удалению символа или текста. В общем случае, алгоритм при изменении положения курсора следующий:

1. При необходимости, определение абсолютной позиции курсора на основе отображаемого
2. Изменение положения курсора
3. Проверка изменения окна отображения для соответствия правилу нахождения курсора внутри окна
4. При необходимости изменение окна отображения
5. При необходимости отображение текста, находящегося в окне отображения с помощью ReactJS

Текстовые редакторы имеют достаточно продолжительную историю развития. Вследствие этого есть устоявшиеся правила изменения курсора при определённых обстоятельствах. Рассмотрим их:

* Если курсор находится на первой строке текста, операция перемещения курсора вверх переместит курсор на начало строки (ввиду отсутствия предшествующей)
* Аналогично прошлому, если курсор находится на последней строке текста, операция перемещения курсора вниз переместит курсор на конец строки (ввиду отсутствия следующей)
* Описанные выше операции справедливы и при перемещении страницы на несколько строк (PageUp/PageDown)
* Если курсор находится перед первым символом непервой строки, то операция перемещения курсора влево переместит его на конец предыдущей строки
* Аналогично прошлому, если курсор находится после последнего символа непоследней строки, то операция перемещения курсора вправо переместит его на начало следующей строки

Эти правила мы определили в классе текстового поля ReactJS при обработке события нажатия клавиши.

Более умные редакторы также реализовывают следующий принцип:

1. Перед началом перемещения позиция символа, перед которым стоит курсор запоминается
2. Если при перемещении курсора между строками вверх и вниз длина строки меньше запомненной позиции курсора, то курсор устанавливается в конец строки
3. В случае, если курсор вновь попадёт на строку, длина которой не меньше запомненной позиции курсора, то курсор занимает позицию перед запомненной позицией символа
4. В случае перемещения курсора влево или вправо, запомненная позиция сбрасывается.

Упростив, можем сказать, что курсор при вертикальных перемещениях всегда стремится занять запомненное положение. При этом запоминается последнее положение после горизонтального перемещения. Это упрощает получение верного отступа при прокрутке больших исходных текстов программ.

Данная функциональность реализована в классе AbsoluteSelection. При горизонтальных изменениях положения курсора в классе ReactJS вызывается метод resetSavedCursorColumn(), который сохраняет текущую позицию курсора перед символом в строке.

Отметим также особенность изменения окна отображения при изменении местоположения курсора. Это изменение происходит только в случае, когда курсор перемещается за пределы отображаемого окна. И здесь различают относительное местоположение курсора в новом окне. Так, если курсор переместился слева направо, пользователь предполагает, что большая левая часть текста ему остаётся видна. В случае же, когда курсор перемещается справа налево, пользователь предполагает отображение большей правой части текста. Это справедливо и при перемещениях курсора вверх/вниз. Поскольку обновление окна отображения на соответствие курсору относится к работе с окном отображения, для возможности разделения на функциональные объекты класс AbsoluteSelection также будет хранить состояние, указывающее направление изменения курсора.

### Работа с буфером обмена

Для работы с буфером обмена будем использовать класс ClipboardEvent. Объект этого класса передаётся в обработчик соответствующий событий работы с буфером обмена:

* вставка текста (onPaste)
* вырезание выделенного подтекста (onCut)
* копирование выделенного подтекста (onCopy)

Обработка данных событий является простой ввиду использования методов других классов. Рассмотрим конкретные алгоритмы:

* вставка
  1. удаление выделенного текста в структуре хранения
  2. вставка текста из буфера обмена в структуру хранения
  3. обновление абсолютной позиции курсора
  4. при необходимости обновление окна отображения
  5. Отображение текста, находящегося в окне отображения с помощью ReactJS
* вырезание
  1. вставка выделенного текста в буфер обмена
  2. удаление выделенного текста в структуре хранения
  3. обновление абсолютной позиции курсора
  4. при необходимости обновление окна отображения
  5. Отображение текста, находящегося в окне отображения с помощью ReactJS
* копирование
  1. вставка выделенного текста в буфер обмена

### Архитектура приложения

Исходя из описанных выше действий, предложений и ограничений по ним выделим функциональные объекты приложения и определим связи и основные публичные методы этих классов. На рисунке представлена UML‑диаграмма отношений и основные классы программы, использующей структуру хранения текста. На рисунке также представлена полная UML‑диаграмма классов приложения, включая библиотеку, представляющую структуру хранения текста и лексический анализатор.

# Технологическая часть

## Методология разработки и используемые средства

## Руководство пользователя

Requirejs

Создание объекта из функции

# Организационно-экономическая часть

## Введение

Разрабатываемое в рамках дипломной работы программное обеспечение является инструментом редактирования больших текстовых файлов, рассчитанное на редактирование исходных файлов программ на языках программирования. Продукт позволяет загружать, просматривать и редактировать текстовые документы и поддерживает синтаксическую подсветки кода. Продукт также может использоваться в качестве модуля для разработки другого программного обеспечения.

На данный момент существует множество систем, являющихся как текстовыми редакторами, выполняемых самостоятельно в операционной системе пользователя или являющихся интернет приложениями. Такие приложения также поддерживают подсветку исходного кода программ большинства языков программирования. В отличие от существующих аналогов в дипломе ведётся разработка модуля, алгоритм которого позволяет гораздо быстрее вносить и обрабатывать изменения в текстовых документах за счёт использования специальной структуры данных, большего размера, чем при использовании в аналогах.

Целью данного раздела является расчет трудоемкости, продолжительности разработки программного обеспечения и сметы затрат.

## Организация и планирование процесса разработки программы

### Техническое задание

Для расчёта затрат на выполнение дипломного проекта используем техническое задание, представленное в таблице 2.

Таблица 2 — Техническое задание

| № | Наименование | Значение |
| --- | --- | --- |
| 1 | Срок начала проекта | 10 марта 2015 г. |
| 2 | Срок окончания проекта | 31 мая 2015 г. |
| 3 | Количество листов А4 записки проекта | 85 |
| 4 | Тип конечного носителя разрабатываемого проекта | Электронный информационный носитель, к которому прилагается описание проекта в виде отчёта |
| 5 | Планируемое число копий | Тиражирование данной продукции не планируется |

### Расчёт стоимости проекта

Базовой экономической характеристикой для оценки дипломного проекта является её стоимость. Стоимость дипломного проекта, как экономическая категория, определяется по формуле 1 [ссылка].

| , | (1) |
| --- | --- |

где

* — стоимость проектных работ,
* — себестоимость проектных работ,
* — прибыль.

Выразим прибыль через себестоимость работ в формуле 2:

| , | (2) |
| --- | --- |

где— желаемый для исполнителя уровень рентабельности.

Исходя из формул 1 и 2, можем выразить стоимость проекта как показано в формуле 3:

| , | (3) |
| --- | --- |

Таким образом для расчёта стоимости проекта потребуется спрогнозировать себестоимость. Для её определения необходимо рассчитать каждую стадию сметы затрат по формуле 4 [ссылка]:

| , | (4) |
| --- | --- |

где соответствующиепредставлены в таблице 3.

Таблица 3— Структура затрат на выполнение проекта

| № | Наименование статьи затрат | Затраты, % |
| --- | --- | --- |
| 1 | Материальные | 20 |
| 2 | Заработная плата (основная и дополнительная) | 30 |
| 3 | Отчисления на социальные нужды | 15 |
| 4 | Амортизация оборудования | 20 |
| 5 | Прочие затраты | 15 |

Прочие затраты обычно составляют 15% [ссылка] и включают в себя:

* оплату налогов;
* подготовку специальной научно-технической информации;
* проведение патентных исследований, научно-технических конкурсов и экспертиз;
* услуги всех видов связи;
* служебные командировки работников в Российской федерации и за рубежом;
* расходы на сертификацию продукции;
* представительские расходы.

Практика использования таких данных и экономическая целесообразность показывает, что наилучший результат по точности прогноза получается, если в качестве искомой величины взять расчет затрат на заработную плату и отчисления на социальные нуждыи через это значение определить себестоимость проекта. Исходя из значений доли статьи расходов к общей стоимости проекта и из формулы 4, рассчитаем себестоимость проекта в формуле 5:

| , | (5) |
| --- | --- |

### Затраты на выплату исполнителям

В заработную плату включается основная и дополнительная заработная плата всех исполнителей, непосредственно занятых разработкой, с учетом их должностного оклада и времени участия в разработке. Дополнительную заработную плату в составе обобщённой определим в формуле 6 [ссылка]:

| , | (6) |
| --- | --- |

где

* — заработная плата,
* — основная заработная плата,
* — дополнительная заработная плата,
* — коэффициент отчислений на дополнительную заработную плату.

Коэффициент примем равным 100%, учитывающим расходы на очередные отпуска, выплаты за выслугу лет и прочее. Таким образом, подставив это значение в формулу 6, получим значение заработной платы, определённой в формуле :

| , | (7) |
| --- | --- |

В настоящее время федеральным законом РФ №212-ФЗ от 24.07.2009 вместо единого социального налога определяются страховые взносы для отчисления в:

* пенсионный фонд РФ,
* фонд социального страхования,
* фонды обязательного медицинского страхования (федеральный и территориальный фонды).

Ставки страховых взносов в 2015 году для организаций, осуществляющих деятельность в области информационных технологий, за исключением организаций, заключивших с органами управления особыми экономическими зонами соглашения об осуществлении технико-внедренческой деятельности, [ссылка] указаны в таблице 4.

Таблица 4 — Ставки страховых взносов на 2014 год

| Получатель | Ставка страхового взноса |
| --- | --- |
| Пенсионный фонд РФ | 8% |
| Фонд социального страхования | 2% |
| Фонд обязательного медицинского страхования | 4% |

Рассчитаем отчисления на социальные нужды с заработной платы в формуле (8):

|  | (8) |
| --- | --- |

К основной заработной плате при выполнении проектных работ относится фонд оплаты труда (ФОТ) научных, инженерных и технических работников, рабочих научно-исследовательских и научно-технических отделов, принимающих непосредственное участие в НИР. ФОТ работников за выполнение разработки определяется по формуле 9:

| , | (9) |
| --- | --- |

где

* — среднедневный ФОТ рабочих i-й специальности,
* — количество дней работ для рабочих i-й специальности,
* — количество рабочих i-й специальности.

Для расчёта среднедневного фонда оплаты труда для рабочего по определённой специальности необходимо рассчитать месячный оклад по формуле :

| , | (10) |
| --- | --- |

где

* — среднедневный ФОТ рабочих i-й специальности,
* — количество дней работ для рабочих i-й специальности,

, величина которого зависит:

* от требуемой квалификации работника (разряда работы),
* от значения МРОТ на момент начала проектирования,
* от тарифного коэффициента и вычисляется по формуле :

Минимальный размер оплаты труда (МРОТ) устанавливается соответствующим законом [ссылка] и составляет 5965 рублей. С учетом принятого в организации тарифного коэффициента на работы (KT ≈ 10.9) размер оплаты труда инженера по защите информации и специалиста по защите информации первой категории составляет:

Для расчёта количества рабочих дней и количества рабочих потребуется рассчитать трудоёмкость проекта. Разработка программного продукта состоит из пяти основных этапов, состав работ которых указан в таблице 5 [ссылка]:

* техническое задание;
* эскизный проект;
* технический проект;
* рабочий проект;
* внедрение.

Таблица 5 — Этапы разработки программного продукта

| **№ этапа** | **Название этапа** | **Общий состав работ этапа** |
| --- | --- | --- |
| 1 | Техническое задание (ТЗ) | Разработка ТЗ. |
| 2 | Эскизный проект (ЭП) | Исследование существующего программного продукта.  Уточнение структуры и формы представления входных и выходных данных. Разработка алгоритма решения задачи. Разработка структуры программы.  Разработка пояснительной записки. Согласование и утверждение технического проекта. |
| 3 | Технический проект (ТП) | Разработка алгоритмов (общих алгоритмов и структуры данных, структуры основных и вспомогательных модулей и др.) |
| 4 | Рабочий проект (РП) | Описание программы на языке программирования.  Разработка, создание и утверждение порядка и методики испытаний, корректировка программы. |
| 5 | Внедрение (В) | Разработка программной документации.  Подготовка и передача программы и программной документации для сопровождения и изготовления, оформления и утверждения акта о передаче ПП на сопровождение. Передача ПП заказчику. |

Исходя из расчёта трудоёмкости всего проекта, определим трудоёмкость каждого этапа и по заданным срокам проекта требуемое количество работников.

Вначале рассчитаем трудоёмкость проекта по нормативно-статистическому методу [ссылка] по формуле 11. За единицу нормирования принимается разработка одного листа технической документации формата A4 эскизного проекта, что составляетот общей трудоёмкости проекта, согласно [ссылка].

| , | (11) |
| --- | --- |

где

* — трудоёмкость всего проекта, чел/час;
* — количество требуемых листов документации, шт;
* — норма времени на разработку одного листа формата А4, час/шт;
* — коэффициент, учитывающий объём входной информации;
* — коэффициент, учитывающий сложность контроля информации;
* — коэффициент, учитывающий вид обработки информации (режим обработки информации);
* — поправочный коэффициент по степени применения типовых проектных решений, пакетов прикладных программ, типовых проектов, типовых программ и стандартных модулей;
* — коэффициент учёта уровня алгоритмического языка программирования;
* — доля трудозатрат в общем проекте.

Коэффициент, учитывающий объём входной информации зависит от количества наборов входных данных и выражается по формуле 12:

| , | (12) |
| --- | --- |

где

* , и – значения коэффициентов учета вида используемой информации для переменной, нормативно-справочной информации и баз данных соответственно;
* , и – количество наборов данных переменной, нормативно-справочной информации и базы данных соответственно.

На вход программного продукта должна подаваться информация четырёх видов:

* положение и размеры окна отображения,
* положение курсора/положение вставляемого или удаляемого текста,
* вставляемый или удаляемый текст,
* сведения о лексемах языка программирования.

На выходе алгоритма два вида информации:

* текст, разделённый на лексемы,
* домен лексемы.

По степени новизны программной продукт может быть отнесён к одной из четырех групп, представленный в таблице 6. А по степени сложности алгоритма к одной из трёх групп, указанных в таблице 7.

В данном случае программа относится к группе «В», поскольку существуют программные комплексы, реализующие аналогичный функционал. А по степени сложности к группе 3.

Для группы новизны «В» и сложности алгоритма группы 3 значения коэффициентов равны [ссылка]: ;;. Подставив их в уравнение 12 получим значение в уравнении 13:

| , | (13) |
| --- | --- |

Таблица 6 — Классификация степени новизны разрабатываемого программного продукта

|  |  |
| --- | --- |
| **Название группы** | **Описание** |
| А | Разработка программных комплексов, требующих использования принципиально новых методов их создания, проведение НИРС и т.п. |
| Б | Разработка программной продукции, не имеющей аналогов, в том числе разработка пакетов прикладных программ. |
| В | Разработка программной продукции, имеющей аналоги. |
| Г | Разработка программной продукции, основанной на привязке типовых проектных решений. |

Таблица 7 — Классификация степени сложности алгоритма программной продукции

| Степень сложности | Описание |
| --- | --- |
| 1 | Программная продукция, реализующая оптимизационные и моделирующие алгоритмы |
| 2 | Программная продукция, реализующая учётно-статистические алгоритмы |
| 3 | Программная продукция, реализующая алгоритмы стандартных методов решения задач |

Определим требуемые в уравнении 11 коэффициенты. Количество требуемых листов документации исходя из технического задания равно . Норма времени на разработку документации одного листа формата А4 равно часам [ссылка]. Согласно таблицам [ссылка] коэффициент учёта режима обработки информациидля технического проекта с обработкой информации в реальном времени группы новизны «В» равен . Коэффициент, учитывающий сложность контроля информации, для данной специфики задачи равен . Поскольку данный программный продукт использует библиотеку для визуализации — ReactJS — только для создания примера текстового редактора на основе полученной структуры хранения текста, поправочный коэффициент по степени применения типовых проектных решений, пакетов прикладных программ возьмём равным . Программный код разрабатывается на языке высокого уровня, поэтому коэффициент учёта уровня алгоритмического языка программирования равен .

Таким образом, подставив коэффициенты в уравнение 11 получим трудоёмкость рабочего проекта в формуле 14:

| , | (14) |
| --- | --- |

Для того, чтобы определить количество человек, требуемых для выполнения каждого из этапов разработки, необходимо определить трудоёмкость каждого этапа. В таблице 8 представлены доли трудоёмкости каждого этапа, согласно [ссылка], и рассчитанная по формуле 15 абсолютное его значение.

| , | (15) |
| --- | --- |

где — доля каждого этапа.

Таблица 8 — Трудоёмкость этапов разработки программного продукта

| **№ этапа** | **Название этапа** | **Трудоёмкость** | |
| --- | --- | --- | --- |
| **%** | **чел/час** |
| 1 | Техническое задание | 10 | 104,4 |
| 2 | Эскизный проект | 20 | 208,8 |
| 3 | Технический проект | 25 | 261,0 |
| 4 | Рабочий проект | 30 | 313,2 |
| 5 | Внедрение | 15 | 156,6 |
|  | Всего | 100 | 1044 |

При выполнении разработки требуемое количество исполнителей для выполнения этапа в заданный срок определяется по формуле 16:

| , | (16) |
| --- | --- |

где

* – трудоёмкость этапа, чел/час;
* – коэффициент дополнительных работ, учитывающий затраты времени на работы, не предусмотренные нормативами, ;
* — фонд рабочего времени исполнителя за период, определяемый сроками;
* — коэффициент, учитывающий выполнение норм, .

Фонд рабочего времени каждого исполнителя за период с 10 марта 2015 года по 31 мая 2015 года рассчитывается по формуле 17:

| , | (17) |
| --- | --- |

где

* — время выполнения проекта в месяцах (устанавливается в ТЗ и для этого проекта равно 2,6 месяца),
* — фонд времени в текущем месяце, который рассчитывается из учета общего числа дней в году, числа выходных и праздничных дней, рассчитываемого по формуле 18:

| , | (18) |
| --- | --- |

где

* — продолжительность рабочего дня,
* — общее число дней в году,
* — число выходных дней в году,
* — число праздничных дней в году.

Нерабочие праздничные дни в году устанавливается соответствующим законом [ссылка на статью 112 главы 18 трудового кодекса №197-ФЗ]. Подставив соответствующие значения рассчитаем среднемесячный фонд времени в формуле :

| , | (19) |
| --- | --- |

Подставив значения в формулу 17 получим фонд времени на период работы, определённых в техническом задании равным 428,1 часам. Таким образом, при равномерном распределении работ для выполнения проекта в срок, установленный техническим заданием — 31 мая 2015 года —подставив в формулу 16 соответствующие значения и используя полные трудозатраты по всему проекту получим среднее значение требуемого количества исполнителей в формуле 20:

| , | (20) |
| --- | --- |

Для целей планирования и контроля работ проекта можно применить календарный ленточный график (диаграмма Ганта) — на оси Х показывают календарные дни (по рабочим неделям) от начала проекта до его завершения, а по оси Y — выполняемые этапы работ.

Продолжительность выполнения работ без учёта выходных и праздничных дней по этапам определяется из формулы (21):

|  | (21) |
| --- | --- |

где

* – трудоёмкость i-й работы, чел.-часы;
* – трудоёмкость дополнительных работ для исполнителя, чел.-часы;
* – количество исполнителей для i-й работы.

Этапы разработки разбиваются на несколько работ, продолжительность которых указана с учётом выходных дней в таблице 9.

Таблица 9 — Описание работ

| Этап | Номер работы | Наименование работы | Должности исполнителей | Число исполнителей | Трудозатраты, чел-час | Продолжи-тельность , дни |
| --- | --- | --- | --- | --- | --- | --- |
| ТЗ | 1 | Составление технического задания | Разработчик, проектировщик | 2 | 31 | 2 |
| ЭП | 2 | Сбор информации о существующих методах и разработках | Разработчик,  проектировщик | 2 | 8 | 1 |
| 3 | Изучение технической литературы | Разработчик, проектировщик | 2 | 11 | 1 |
| 4 | Изучение внутренней структуры программы | Разработчик | 1 | 48 | 6 |
| ТРП | 5 | Исследование оптимальной структуры модели | Разработчик,  проектировщик | 2 | 152 | 10 |
| 6 | Разработка алгоритмов | Проектировщик | 1 | 54 | 7 |
| 7 | Разработка и изменение внутренних структур программы | Разработчик | 1 | 118 | 15 |
| В | 8 | Создание сопроводительной документации | Разработчик | 1 | 10 | 2 |
| 9 | Проведение испытаний и сравнение полученных результатов | Проектировщик | 1 | 40 | 5 |

В программе Calligra Plan была построена диаграмма Ганта указанных работ с учётом праздничных и выходных дней. Диаграмма представлена в приложении на рисунке А.1.

Длительность стадий проектных работ определяется по формуле 22:

| , | (22) |
| --- | --- |

где

* – продолжительность стадии, рабочие дни;
* – трудоёмкость этапа, чел/час;
* – доля дополнительных работ;
* – учитываемая продолжительность рабочего дня, часов/рабочий день.

Необходимо распределить этапы работ для целого числа исполнителей с таким расчётом, чтобы уложиться в требуемые сроки. Такое распределение приведено в таблице :

Таблица 10 — Оптимизированное количество исполнителей проектных работ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № стадии | 1 | 2 | 3 | 4 | 5 |
| Трудоёмкость, чел/час | 104,4 | 208,8 | 261,0 | 313,2 | 156,6 |
| Доля дополнительных работ | 0,1 | 0,2 | 0,3 | 0,2 | 0,2 |
| Количество исполнителей | 2 | 3 | 3 | 3 | 2 |

В соответствии с Единым Тарифно-Квалификационным справочником квалификация работников, выполняющих этапы работ [ссылка **Квалификационный справочник должностей руководителей, специалистов и других служащих** (утвержден постановлением Минтруда РФ от 21 августа 1998 г. N 37)], назначается следующим образом:

* инженер-программист 1 категории,
* инженер по научно-технической информации 1 категории.

Разрабатываемое в ходе дипломного проектирования программное обеспечение выполняет функции управления входящим и исходящим потоками информации по специальным условиям, поэтому его можно отнести к функциональному назначению ПП «Управление технической подготовкой производства». По степени новизны продукция относится к группе «В», и время на подготовку ТЗ,

### Определение количества исполнителей

Средняя численность состава исполнителей при реализации проекта разработки и внедрения ПО определяется по формуле (23):

| , | (23) |
| --- | --- |

где

* - затраты труда на выполнение проекта (разработка и внедрение ПО),
* — фонд рабочего времени.

Величина фонда рабочего времени определяется по формуле (24):

| , | (24) |
| --- | --- |

где

* — время выполнения проекта в месяцах (как правило, устанавливается заказчиком проекта),
* — фонд времени в текущем месяце, который рассчитывается из учета общего числа дней в году, числа выходных и праздничных дней, рассчитываемого по формуле (25):

| , | (25) |
| --- | --- |

где

* — продолжительность рабочего дня,
* — общее число дней в году,
* — число выходных дней в году,
* — число праздничных дней в году.

Рассчитаем фонд времени в одном месяце 2014 года в формуле (26):

| . | (26) |
| --- | --- |

Время выполнения проекта — 2 месяца. Следовательно, в формуле (27) получим фонд рабочего времени:

| . | (27) |
| --- | --- |

Рассчитанная в предыдущем пункте трудоёмкость ПП человеко‑часов. В соответствии с этими данными, рассчитаем среднее количество исполнителей в формуле (28):

|  | (28) |
| --- | --- |

Округлив до «целых программистов», получаем число исполнителей проекта человека. Их будем называть разработчиком и проектировщиком ТЗ.

### Календарный график выполнения работ

## Расчёт сметы затрат

Затраты на выполнение проекта состоят из прямых затрат (заработная плата исполнителям, затраты на закупку или аренду оборудования, затраты на организацию рабочих мест), и косвенных затрат (т.н. накладные расходы) вычисляются по формуле (29):

| , | (29) |
| --- | --- |

где

* — заработная плата исполнителей;
* — затраты на обеспечение необходимым оборудованием;
* — затраты на организацию рабочих мест;
* — накладные расходы.

Рассчитаем все составляющие затрат на разработку программного продукта.

### Затраты на выплату исполнителям

Затраты на выплату исполнителям заработной платы линейно связаны с трудоемкостью и определяются по формуле (30):

| , | (30) |
| --- | --- |

где

* — основная заработная плата;
* — дополнительная заработная плата;
* — отчисление с заработной платы.

Расчёт основной заработной платы (оплаты труда непосредственных исполнителей) производится по формуле (31):

| , | (31) |
| --- | --- |

где

* — число дней, отработанных исполнителем проекта;
* — дневной оклад исполнителя.

При 8-и часовом рабочем дне он рассчитывается по формуле (32):

| , | (32) |
| --- | --- |

где

* — месячный оклад;
* — месячный фонд рабочего времени.

С учетом налога на доходы физических лиц размер оклада увеличивается, что отражено в формуле (33):

| , | (33) |
| --- | --- |

где

* — «чистый» оклад;
* — налог на доходы физических лиц в размере 13% .

В нашем проекте разработчиком ПО является программист C++, а проектировщиком — системный аналитик. Средние заработные платы по Москве были определены с помощью электронного ресурса Яндекс.Работа и для программиста составляют 89000 рублей в месяц при полной рабочей неделе[1], а аналитика — 104000 рублей в месяц[2].

Итоговые заработные платы для работников указаны ниже (Таблица 11).

Следовательно, общие затраты на заработную плату исполнителям проекта составят руб. Расходы на дополнительную заработную плату учитывают все выплаты непосредственным исполнителям за время, не проработанное на производстве, но предусмотренное законодательством. Величина этих выплат составляет 20% от размера основной заработной платы и  вычисляется по формуле (34):

Таблица 11 — Затраты на основную заработную плату сотрудников

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | **Должность** | **«Чистый» оклад, руб.** | **Почасовой оклад, руб.** | **Трудозатраты, чел.–час** | **Затраты на зарплату, руб.** |
| 1 | Проектировщик | 104 000 | 650,00 | 195 | 143 227,5 |
| 2 | Разработчик | 89 000 | 556,25 | 277 | 174 111,8 |
| Итого |  | | | | 317 339,3 |

|  | (34) |
| --- | --- |

Рассчитаем расходы на дополнительную заработную плату в формуле (35):

| руб. | (35) |
| --- | --- |

В настоящее время федеральным законом РФ №212-ФЗ от 24.07.2009 вместо единого социального налога определяются страховые взносы для отчисления в:

* пенсионный фонд РФ,
* фонд социального страхования,
* фонды обязательного медицинского страхования (федеральный и территориальный фонды).

Ставки страховых взносов в 2014 году для организаций, осуществляющих деятельность в области информационных технологий, за исключением организаций, заключивших с органами управления особыми экономическими зонами соглашения об осуществлении технико-внедренческой деятельности, [3] указаны в таблице 12.

Таблица 12 — Ставки страховых взносов на 2014 год

| Получатель | Ставка страхового взноса |
| --- | --- |
| Пенсионный фонд РФ | 8% |
| Фонд социального страхования | 2% |
| Фонд обязательного медицинского страхования | 4% |

Рассчитаем отчисления с заработной платы в формуле (36):

| руб. | (36) |
| --- | --- |

Таким образом, получим общие затраты на заработную плату в формуле (37):

| руб. | (37) |
| --- | --- |

### Затраты на оборудование

В совокупные затраты на реализацию проекта также необходимо включить стоимость оборудования. Оборудование, необходимое для работы разработчиков ПП перечислено ниже (Таблица 13).

Таблица 13 — Затраты на оборудование

|  |  |  |
| --- | --- | --- |
| **Наименование оборудования** | **Количество, шт.** | **Общая стоимость, руб.** |
| ПК (средней категории) | 1 | 20 000 |
| ПК (высокопроизводительный) | 1 | 30 000 |
| Итого | | 50 000 |

Срок полезного использования персональных компьютеров будем считать не менее трёх лет. Норма амортизации оборудования (при использовании нелинейного метода) рассчитывается по формуле (38):

| , | (38) |
| --- | --- |

где — срок полезного использования данного оборудования в месяцах.

В нашем случае и норма амортизации в месяц

Амортизация по месяцам представлена в таблице ниже (Таблица 14).

Амортизационные отчисления осуществляются по норме амортизации до того как остаточная стоимость основного средства станет равна 20 % от его первоначальной стоимости. За оставшееся время эксплуатации остаточная стоимость разделяется равномерно.

Таблица 14 — Амортизационные отчисления

| Месяц | Амортизационные отчисления | Остаточная стоимость |
| --- | --- | --- |
| 1 | 2775,00 | 47225,00 |
| 2 | 2620,99 | 44604,01 |
| 3 | 2475,52 | 42128,49 |
| 4 | 2338,13 | 39790,36 |
| 5 | 2208,36 | 37581,99 |
| 6 | 2085,80 | 35496,19 |
| 7 | 1970,04 | 33526,15 |
| 8 | 1860,70 | 31665,45 |
| 9 | 1757,43 | 29908,02 |
| 10 | 1659,90 | 28248,13 |
| 11 | 1567,77 | 26680,35 |
| 12 | 1480,76 | 25199,59 |
| 13 | 1398,58 | 23801,02 |
| 14 | 1320,96 | 22480,06 |
| 15 | 1247,64 | 21232,42 |
| 16 | 1178,40 | 20054,02 |
| 17 | 1113,00 | 18941,02 |
| 18 | 1051,23 | 17889,79 |
| 19 | 992,88 | 16896,91 |
| 20 | 937,78 | 15959,13 |
| 21 | 885,73 | 15073,40 |
| 22 | 836,57 | 14236,83 |
| 23 | 790,14 | 13446,68 |
| 24 | 746,29 | 12700,39 |
| 25 | 704,87 | 11995,52 |
| 26 | 665,75 | 11329,77 |
| 27 | 628,80 | 10700,97 |
| 28 | 593,90 | 10107,06 |
| 29 | 1263,3825 | 8843,68 |
| 30 | 1263,3825 | 7580,30 |
| 31 | 1263,3825 | 6316,91 |
| 32 | 1263,3825 | 5053,53 |
| 33 | 1263,3825 | 3790,15 |
| 34 | 1263,3825 | 2526,77 |
| 35 | 1263,3825 | 1263,38 |
| 36 | 1263,3825 | 0 |

Расход по амортизационным отчислениями зависит от времени использования оборудования. В нашем случае, это 2,33 месяца. Рассчитаем суммарный расход на оборудование за время разработки в формуле (39):

| руб. | (39) |
| --- | --- |

### Затраты на организацию рабочих мест

Расчёт затрат, связанных с организацией рабочих мест для исполнителей проекта, проводится на основе требований СНИПа (санитарные нормы и правила) и стоимости аренды помещения требуемого уровня сервиса.

В соответствии с санитарными нормами, расстояние между рабочими столами с видеомониторами должно быть не менее 2 м., а между боковыми поверхностями видеомониторов - не менее 1,2 м. Площадь на одно рабочее место с терминалом или ПК должна составлять не менее 6 кв.м., а объём — не менее 20 куб.м.. Расположение рабочих мест в подвальных помещениях не допускается. Помещения должны быть оборудованы системами отопления, кондиционирования воздуха или эффективной приточно-вытяжной вентиляцией.

Исходя из данных требований и учитывая количество сотрудников, площадь рабочего помещения должна составлять не менее 15 кв. м. При этом выбор географического расположения рабочего помещения значения не имеет. Средняя цена аренды офисов класса B в пределах Москвы составляет 22438 рублей за кв. м. в год [4].

Затраты на аренду помещения вычисляются по формуле (40):

| , | (40) |
| --- | --- |

где

* — стоимость аренды одного кв. метра площади за год;
* — арендуемая площадь рабочего помещения;
* — срок аренды (календарный) в месяцах.

Рассчитаем эти затраты в формуле (41):

| руб. | (41) |
| --- | --- |

### Накладные расходы

Накладные расходы, связанные с выполнением проекта, вычисляются в соответствии с расходами на основную заработную плату. Обычно они составляют от 60% до 100% от их размера.

Величина этих затрат характеризует зарплату «неосновным» исполнителям и вычисляется по формуле (42):

|  | (42) |
| --- | --- |

Вычислим накладные расходы в формуле (43):

| руб. | (43) |
| --- | --- |

### Суммарные затраты

Суммарные затраты вычисляются как сумма всех затрат по формуле (44):

|  | (44) |
| --- | --- |

Определим затраты на реализацию проекта в формуле (45):

| руб. | (45) |
| --- | --- |

График распределения затрат по категориям представлен на рисунке 9.

## Цена продукта

Если ПП рассматривается и создаётся как продукция производственно-технического назначения, допускает многократное тиражирование и отчуждение от непосредственных разработчиков, то её цена определяется по формуле (46):

| , | (46) |
| --- | --- |

где

* – затраты на разработку ПО;
* – коэффициент учёта затрат на изготовление опытного образца ПП как продукта производственно-технического назначения (К = 1,1…1,2);
* – нормативная прибыль, рассчитываемая по формуле (47):

Рисунок 9 — Распределение затрат на реализацию проекта

| , | (47) |
| --- | --- |

где

* – материальные затраты (в нашем случае, затраты на оборудование);
* – норматив рентабельности, в %.

Минимальное значение рентабельности должно быть выше средней процентной ставки по депозитным вкладам в коммерческом банке. Возьмём за точку отсчёта .

Вычислим нормативную прибыль в формуле (48):

| руб. | (48) |
| --- | --- |

Коэффициент учёта затрат на изготовление опытного образца ПП возьмём минимальный .

Следовательно, цена разрабатываемого ПП получим в формуле (49):

| руб. | (49) |
| --- | --- |

## Дисконтирование дохода

Полный программный продукт представляет собой клиентскую часть системы распространения видео и аудиопродукции. Существуют коммерческие сервисы, предоставляющие услуги предоставления доступа к такой продукции, работающие по другой технологии. Потенциальные клиенты — компании, предполагающие развивать бизнес в данной области с использованием P2P технологий. Наши пользователи — клиенты «онлайн кинотеатров». Они будут предлагать клиентам бесплатное клиентское приложение, модификация которого была произведена в дипломной работе. Также существует компания, создающая полный программный комплекс и предоставляющий сервис онлайн кинотеатра — AceStream. На данный момент программный комплекс находится в тестовом периоде и не имеет коммерческой эксплуатации.

Предполагаем, что количество продаж будет составлять 30% от количестве таких компаний (в виду насщения рынка). На данный момент на российском рынке представлено 12 компаний (Now, Ivi, Zoomby, Megogo, Playfamily, Stream, TVzavr, Tvigle, Zabava, Amediateka, Viaplay, Molodejj). Количество продаж получим равной 3,6 или около 4-х продаж. Будем продавать продукт равномерно в течение 12 месяцев после создания 4 покупателям по цене в 350 000 рублей. Требуется провести дисконтирование полученного дохода.

Смысл дисконтирования заключается в том, что текущая стоимость будущих финансовых потоков может существенно отличаться от их номинальной стоимости. Для расчёта используется формула (50):

|  | (50) |
| --- | --- |

где

* — текущая стоимость;
* — будущая стоимость;
* — номер месяца;
* — постоянная нормы дисконта, в %;
* — расчётный период, в месяцах.

В качестве нормы дисконта обычно берётся ставка рефинансирования. На 2014 год она равна 8,25 %.

По результатам предыдущего пункта, цена программного продукта равна 906 931 рублям.

За три месяца разработки затраты разделим в пропорции 6:3:1 (шесть частей в первый месяц, три части во втором и одна часть в третьем месяцах). Доход от продукта мы получим лишь в четвёртом, седьмом, одиннадцатом и четырнадцатом месяцах с момента старта проекта. Произведём дисконтирование в формуле (51):

| руб. | (51) |
| --- | --- |

## Вывод

Для реализации данного проекта необходимы программист C++ (трудозатраты составят 277 часов) и системный аналитик (трудозатраты составят 195 часов). Продолжительность выполнения проекта составляет 67 дней с учётом выходных и праздничных дней.

Расходы на разработку продукта равны 746 088 рублей. Дисконтированный доход после 14 месяцев от начала работы над проектом составит 147 270 рублей при сроке окупаемости в 11 месяцев. Дисконтированный доход является положительным, следовательно проект выгоден.

# Заключение

В данной работе были собраны статистические сведения по распределению вероятностей скоростей загрузки данных в реальном окружении с различными параметрами, имитирующие разные условия использования. На основе полученных данных было произведено моделирование системы массового обслуживания — структуры торрент-клиента — и выбрана оптимальная его архитектура.

После проведения анализа внутренней структуры торрент-клиента с открытым исходным кодом KTorrent была произведена его доработка для обеспечения оптимальной модели загрузки.

Анализ полученных результатов показал, что при скорости загрузки, превышающей необходимую скорость на 5% не наблюдается прерываний при воспроизведении, в отличие от предыдущего алгоритма. По сравнению с аналогичными программными продуктами ...

Библиография

1: , The BitTorrent Protocol Specification, , http://www.bittorrent.org/beps/bep\_0003.html

2: , Fast Extension, , http://www.bittorrent.org/beps/bep\_0006.html

3: , NAT - Википедия, , https://ru.wikipedia.org/wiki/NAT

4: , Phonon - KDE UserBase Wiki, , http://userbase.kde.org/Phonon/ru

5: , MediaSource CLass Reference | Documentation | Qt Project, , http://qt-project.org/doc/qt-4.8/phonon-mediasource.html#Type-enum

6: , Phonon.AbstractMediaStream, , http://api.kde.org/pykde-4.2-api/phonon/Phonon.AbstractMediaStream.html

7: , ID3 (метаданные), , https://ru.wikipedia.org/wiki/ID3\_(метаданные)

8: , Статистика зарплат "программист c++" в Москве — Яндекс.Работа, , http://rabota.yandex.ru/salary.xml?text=%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82%20c%2B%2B&rid=213

9: , Статистика зарплат "системный аналитик" в Москве - Яндекс.Работа, , http://rabota.yandex.ru/salary.xml?text=%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D1%8B%D0%B9%20%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D1%82%D0%B8%D0%BA

10: , Для категорий страхователей, перечисленных в пп. 4, 5, 6 п. 4 ст. 33 Федерального закона от 15.12.2001 N 167-ФЗ - Справочная информация: "Страховые взносы, установленные начиная с 2010 г., в Пенсионный фонд РФ, Фонд социального страхования РФ и Федеральный фонд обязательного медицинского страхования вместо единого социального налога" (Материал подготовлен специалистами КонсультантПлюс) (15 декабря 2001 г.) \ Консультант Плюс, , http://www.consultant.ru/document/cons\_doc\_LAW\_93256/?frame=4

1. Диаграмма Ганта выполненных работ

В столбцах приведены номера недель и дни. Серыми указаны выходные дни.

В строках указан задачи и их наименование.

Стрелки обозначают зависимость выполнения работ.

Рисунок А.1 — Диаграмма Ганта