ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Программная инженерия»

СОГЛАСОВАНО	УТВЕРЖДАЮ
Научный руководитель, Заместитель декана по учебно-методической работе факультета компьютерных наук доцент департамента больших данных и информационного поиска	Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук
_____ И. Ю. Самоненко	_____ В. Шилов
«_09_»_____мая_____ 2022 г.	«_09_»_____мая_____ 2022 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

ПРИЛОЖЕНИЕ ДЛЯ ВИЗУАЛИЗАЦИИ АЛГОРИТМА АХО-КОРАСИК

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.10.03-01 81 01-1-ЛУ

Исполнитель
студент группы БПИ201
_____/ М. М. Ким /
«_09_»_____мая_____ 2022 г.

Москва 2022

УТВЕРЖДЕН
RU.17701729.10.03-01 81 01-1-ЛУ

ПРИЛОЖЕНИЕ ДЛЯ ВИЗУАЛИЗАЦИИ АЛГОРИТМА АХО-КОРАСИК

Пояснительная записка

RU.17701729.10.03-01 81 01-1

Листов 19

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Москва 2022

Оглавление

1. ВВЕДЕНИЕ 3
1.1. Наименование программы 3
1.2. Документы, на основании которых ведётся разработка 3
2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ 4
2.1. Назначение программы 4
2.1.1 Функциональное назначение программы 4
2.1.2 Эксплуатационное назначение программы 4
2.2. Краткая характеристика области применения 4
3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ 5
3.1. Постановка задачи на разработку программы 5
3.2. Описание алгоритма и функционирования программы 5
3.2.1. Общий алгоритм работы программы 5
3.2.2. Устройство хранения действий алгоритма. Класс ActionList и ListNode. 6
3.2.3. Алгоритм отрисовки дерева. Класс Tree. 6
3.2.4. Алгоритм отображения действий работы алгоритма 7
3.3. Описание и обоснование выбора метода организации входных и выходных данных 7
3.3.1. Описание метода организации входных данных 7
3.3.2. Обоснование метода организация входных данных 8
3.3.3. Описание метода организации выходных данных 8
3.3.4. Обоснование метода организации выходных данных 8
3.4. Описание и обоснование выбора метода выбора технических и программных средств 8
3.4.1. Описание метода выбора технических и программных средств 9
3.4.2. Обоснование метода выбора технических и программных средств 9
4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ 10
4.1. Ориентировочная экономическая эффективность 10
4.2. Предполагаемая потребность 10
4.3. Экономические преимущества разработки по сравнению с отечественными или зарубежными аналогами 10
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 11
ПРИЛОЖЕНИЕ 1 12
ПРИЛОЖЕНИЕ 2 13
ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 19

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование программы – «Приложение для визуализации алгоритма Ахо-Корасик».

Наименование программы на английском языке – «Application for Aho-Corasick algorithm visualization».

1.2. Документы, на основании которых ведётся разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденная академическим руководителем тема курсового проекта.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение программы

2.1.1 Функциональное назначение программы

Функциональным назначением разработанной программы является предоставление пользователям возможности работы с алгоритмом Ахо-Корасик для поиска всех вхождений строк из заданного набора в текст, получение информации о каждой вершине построенного в ходе работы с алгоритмом конечного автомата [1, 7], а также ознакомление с текстовым описанием шагов работы алгоритма и его визуальным представлением.

2.1.2 Эксплуатационное назначение программы

Приложение может быть использовано в образовательных целях преподавателями технических высших учебных заведений; студентами, изучающими компьютерные науки; учениками общеобразовательных учреждений, интересующимися алгоритмами, разработчиками программного обеспечения. Разработанная программа позволит упростить и ускорить процесс изучения алгоритма Ахо-Корасик благодаря пошаговым объяснениям и визуальному отображению работы алгоритма.

2.2. Краткая характеристика области применения

«Приложение для визуализации алгоритма Ахо-Корасик» - веб-приложение, которое может быть использовано для изучения алгоритма Ахо-Корасик. Программа позволяет ознакомиться с процессом работы алгоритма: добавлением слова в словарь, а также поиском всех вхождений слов из заданного словаря в текст (позиция вхождения, само слово). Разработанный продукт несет образовательный характер и может быть применен в целях упрощения изучения и понимания работы алгоритма. Также программа может быть применена в целях поиска всех вхождений строк из заданного набора в текст [1, 7], что может быть полезно для, например, цензурирования текста.

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Приложение состоит только из клиентской части.

В приложении должны быть реализованы:

1. Ввод входных данных (словарь, а также строка для поиска в ней множества подстрок их словаря).
2. Построение автомата.
3. Поиск всех вхождение подстрок из словаря в строке.
4. Отображение текущего состояния автомата.
5. Отображение данных о каждой вершине автомата.
6. Возможность перейти к следующему и предыдущему шагу.
7. Отображение подсказок для пользователя (корректность входных данных, назначение кнопок).
8. Возможность сброса всех данных, хранимых в автомате.

3.2. Описание алгоритма и функционирования программы

3.2.1. Общий алгоритм работы программы

За работу приложения отвечает класс Controller, обрабатывающий нажатия на кнопки: добавление слова в словарь; поиск всех вхождений слов в текст; сброс всех данных, содержащихся в автомате; переход между шагами работы алгоритма. В качестве полей он содержит сущности других классов, отвечающих за: алгоритм Ахо-Корасик (класс AhoCorasick); отрисовка автомата в виде дерева (класс Tree); хранение шагов алгоритма (Класс ActionList). В файлах main.js и init.js содержится привязка действий к кнопкам, привязка действий к действиям пользователя, а также инициализация глобальных переменных, используемых во всей программе [1, 7].

При добавлении слова в словарь, класс Controller запускает добавление слова в классе AhoCorasick, отвечающим за работу алгоритма. После выполнения действия Controller получает список действий (действие может состоять из нескольких составных частей – функций, см. подробнее в описании класса ActionList) в виде сущности класса ActionList, после чего добавляет полученные действия к существующим. Аналогичный алгоритм имеет и операция поиска. После этого создаётся новая сущность класса Tree, в которую передаётся корень дерева из объекта класса AhoCorasick и высчитывается позиция дерева по середине окна с деревом.

При нажатии кнопок, отвечающих за переход между действиями, класс Controller использует список действий (сущность класса ActionList), доставая из списка в зависимости от шага (следующий / предыдущий) действие и отправляя его далее на обработку. В случае, если необходимо получить предыдущий шаг, в функцию обработки действия в качестве аргумента посылается специальный флаг, сигнализирующий о том, что действие необходимо инвертировать. При обработке действия программа перебирает все функции в действии и запускает их с соответствующими списками аргументов.

3.2.2. Устройство хранения действий алгоритма. Класс ActionList и ListNode.

Класс ListNode представляет из себя элемент двусвязного списка, хранящий в себе предыдущий и следующий объекты, а также действие работы алгоритма в виде массива, состоящего из функции и массива её аргументов для вызова. Имеет возможность добавления действия в конец массива действий.

Класс ActionList представляет из себя двусвязный список, хранящий действия работы алгоритма в виде объектов класса ListNode. Также в классе хранится первый и последний элемент двусвязного списка, его размер, а также указатель на текущее действие. В классе имеется возможность добавления действия в конец списка; получения текущего / следующего / предыдущего действия; проверки существования следующего / предыдущего действия; конкатенации с другим объектом ActionList; очистки списка.

Класс ActionList используется для хранения и получения действий работы алгоритма. Все функции, из которых состоят действия, содержатся в файле actions.js.

3.2.3. Алгоритм отрисовки дерева. Класс Tree.

Отрисовка дерева происходит на объекте типа canvas с помощью библиотеки Konva. Каждая вершина отображается в виде круга с буквой по центру (в случае корня «root»). Связи между вершинами дерева отображаются в виде линий. Отдельные переходы между вершинами отображаются в виде стрелок. В процессе работы алгоритма вершины, линии, стрелки могут быть подсвечены.

При создании класса Tree в конструктор передаётся корень дерева, после чего просчитываются все необходимые поля вершин класса TrieNode, хранящихся в классе AhoCorasick. Каждый раз при изменении вершин дерева пересчитываются все позиции его вершин, соединяющих линий, подписей вершин. Определение координат каждой из

вершин осуществляется с помощью алгоритма Уолкера [3]. Также в файле tree.js содержатся параметры дерева. При необходимости переподсчитать координаты вершин, вызывается необходимая функция.

3.2.4. Алгоритм отображения действий работы алгоритма

Все функции по непосредственному отображению действий работы алгоритма находятся в файлу actions.js. Как уже было сказано в пункте 3.2.1 функции вызываются из класса Controller. Для каждого действия (отрисовка вершины, её удаление, выделение вершины и пр.) существуют отдельные функции, из которых составляются полноценные действия работы алгоритма в классе AhoCorasick при добавлении слова в словарь и при поиске всех вхождений.

Большинство функции отображения элементов имеют их инверсированные версии. Так, при вызове функции с флагом обратного действия, функция вместо выполнения вызовет противоположную себе функцию, но уже без флага обратного действия. Таким образом реализован откат состояний работы алгоритма до предыдущих действий.

3.3. Описание и обоснование выбора метода организации входных и выходных данных

3.3.1. Описание метода организации входных данных

При добавлении слова в словарь, пользователь вводит слово в отдельно выделенное для этого поле и нажимает на кнопку «Add». Слово должно состоять исключительно из символов латинского алфавита. Минимум 1 символ, максимум 20.

При поиске всех вхождений слов из словаря в текст, пользователь вводит текст в отдельно выделенное для этого поле и нажимает на кнопку «Find». Текст должен состоять исключительно из символов латинского алфавита и пробелов. После поиска добавление нового слова в словарь будет недоступно. Минимум 1 символ, максимум 100.

При необходимости переходов между шагами алгоритма пользователь нажимает на кнопки «Next step» или «Previous step».

При необходимости очистки всех данных пользователь нажимает на кнопку «Clear».

3.3.2. Обоснование метода организация входных данных

Приведённая организация входных данных полностью отвечает требованиям технического задания и является одной из наиболее простых.

Максимум символов в слове обусловлено необходимостью избежания излишне громоздкого дерева, что помешает простоте восприятия алгоритма.

Невозможность добавления нового слова в словарь обусловлено необходимостью алгоритма пересчитывать необходимую для поиска информацию после добавления слова.

3.3.3. Описание метода организации выходных данных

При добавлении нового слова в словарь и поиске программа должна начинать отображать информацию о первом шаге алгоритма (текстовое описание шага, выделение нужных элементов).

При переходе на следующий шаг программа должна отобрать информацию о следующем шаге алгоритма (добавление нового блока информации о вершине, отрисовка вершины, текстовая информация о шаге алгоритма, выделение нужных элементов, добавление информации о результатах действия).

При переходе на предыдущий шаг программа должны отображать информацию о предыдущем шаге алгоритма (удаление / добавление нового блока информации о вершине, отрисовка вершины / удаление вершины, текстовая информация о шаге алгоритма, выделение нужных элементов, добавление / удаление информации о результатах действия).

При очистке всех данных программа должна отображать обновленное состояние (удаление всех вершин, блоков информации о вершинах, / удаление информации о результатах действия, очистка текстовой информации о шаге алгоритма).

3.3.4. Обоснование метода организации выходных данных

Приведённая организация выходных данных полностью отвечает требованиям технического задания и является одной из наиболее простых.

3.4. Описание и обоснование выбора метода выбора технических и программных средств

3.4.1. Описание метода выбора технических и программных средств

Для работы программы необходимы следующие компоненты:

- Операционная система: Windows 7 / Windows 8 / Windows 10 / MacOS X 11 и новее.
- Процессор: Intel Pentium 4 / Athlon 64 или новее.
- Свободное место на жёстком диске: не менее 256 МБ.
- Оперативная память: не менее 2ГБ.
- Клавиатура.
- Мышь.
- Стабильное подключение к сети интернет.
- Браузер Mozilla Firefox версии не ниже 54 / Google Chrome версии не ниже 51 / Safari версии не ниже 10.
- Монитор с разрешением не менее чем 1600 на 900.

3.4.2. Обоснование метода выбора технических и программных средств

Выбор свободного места на жестком диске и оперативной памяти обусловлен потребностями браузера.

Выбор браузера обусловлен тем, что библиотека Konva и сам код написаны с учетом стандарта ES6, который полностью поддерживается начиная с этих версий браузера.

Выбор разрешения монитора обусловлено громоздкостью визуализации алгоритма, поэтому это разрешение будет минимальным для комфортного изучения работы алгоритма.

Клавиатура и мышь необходимы для взаимодействия с программой.

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая эффективность

Программа способна сэкономить большое количество времени благодаря значительному упрощению изучения принципа работы алгоритма.

4.2. Предполагаемая потребность

Программа может быть необходима:

1. Преподавателям ВУЗов и школ, ведущим курсы, связанные с алгоритмами и структурами данных.
2. Студентам ВУЗов и школ, изучающих алгоритмы и структуры данных.
3. Разработчикам программного обеспечения.

4.3. Экономические преимущества разработки по сравнению с отечественными или зарубежными аналогами

Таблица 1 – Экономические преимущества разработки по сравнению с отечественными или зарубежными аналогами

	Приложение для визуализации алгоритма Ахо-Корасик	Jovilab sinaapp [5]	Bruno's Code [4]	Algorithm Visualizer [2]	VisuAlgo [6]
Визуализация алгоритма Ахо—Корасик	+	+	+	-	-
Подробная информация о каждом шаге алгоритма	+	-	-	+	+
Отображение информации о вершинах автомата	+	+	+	-	-
Возможность перехода к следующему / предыдущему шагу	+	-	-	+	+
Возможность ввода собственных данных	+	+	+	+	+

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Aho–Corasick algorithm [Электронный ресурс]: Режим доступа: https://en.wikipedia.org/wiki/Aho–Corasick_algorithm, свободный. (дата обращения: 07.05.2022)
2. Algorithm Visualizer [Электронный ресурс]: Режим доступа: https://algorithm-visualizer.org/, свободный. (дата обращения: 07.05.2022)
3. A node-positioning algorithm for general trees [Электронный ресурс]: Режим доступа: https://onlinelibrary.wiley.com/doi/10.1002/spe.4380200705, свободный. (дата обращения: 07.05.2022)
4. Bruno’s code [Электронный ресурс]: Режим доступа: https://brunorb.com/aho-corasick/, свободный. (дата обращения: 07.05.2022)
5. Jovilab sinaapp [Электронный ресурс]: Режим доступа: http://jovilab.sinaapp.com/, свободный. (дата обращения: 07.05.2022)
6. VisuAlgo [Электронный ресурс]: Режим доступа: https://visualgo.net/en, свободный. (дата обращения: 07.05.2022)
7. Алгоритм Ахо-Корасик [Электронный ресурс]: Режим доступа: https://e-maxx.ru/algo/aho_corasick, свободный. (дата обращения: 07.05.2022)

ПРИЛОЖЕНИЕ 1

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ

Таблица 2. Описание и функциональное назначение классов

Класс	**Назначение**
ListNode	Представляет собой элемент двусвязного списка. Хранит в себе действие работы алгоритма (действие состоит из нескольких функций).
Action List	Представляет собой двусвязный список из элементов класса ListNode. Используется для хранения и получения действий работы алгоритма.
TrieNode	Представляет собой вершину автомата, строящегося в ходе работы алгоритма. Хранит в себе множество полей необходимых для отрисовки дерева.
AhoCorasick	Представляет собой класс, содержащий все функции алгоритма Ахо-Корасик. Заполняет и возвращает ActionList в Controller.
Controller	Управляет работой программы.
Tree	Представляет собой алгоритм Уолкера [3]. Необходим для определения координат вершин дерева.
Queue	Представляет собой очередь. Необходим для алгоритма Уолкера [3].

ПРИЛОЖЕНИЕ 2

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ

Типы полей обозначены как предполагаемые, так как JavaScript динамически типизированный язык.

Таблица 3.1 – Описание полей и методов класса ListNode

Имя	**Предполагаемый тип**	**Назначение**
value	Object {func, args}	Хранит действие в виде массива функций и аргументов для их вызова.
prev	ListNode	Хранит указатель на предыдущий элемент в двусвязном списке.
next	ListNode	Хранит указатель на следующий элемент в двусвязном списке.
push	Function	Добавляет функцию и аргументы для её вызова в конец массива действия.

Таблица 3.2. – Описание полей и методов класса ActionList

Имя	**Предполагаемый тип**	**Назначение**
head	ListNode	Хранит указатель на первый элемент двусвязного списка.
tail	ListNode	Хранит указатель на последний элемент двусвязного списка.
size	Integer	Хранит текущий размер списка.
current	ListNode	Хранит указатель на текущее действие.
push	Function	Добавляет действие в конец двусвязного списка.
concat	Function	Конкатенирует с другим двусвязным списком.
getCurrent	Function	Возвращает текущее действие.

Продолжение Таблицы 3.2.

Имя	**Предполагаемый тип**	**Назначение**
getPrev	Function	Возвращает предыдущее действие.
getNext	Function	Возвращает следующее действие.
isNext	Function	Проверяет, если существует следующее действие и возвращает true иначе false.
isPrev	Function	Проверяет, если существует предыдущее действие и возвращает true иначе false.
clear	Function	Очищает двусвязный список.

Таблица 3.3. – Описание полей и методов класса TrieNode

Имя	**Предполагаемый тип**	**Назначение**
to	Map	Хранит переходы между вершинами [1,7]
go	Map	Хранит автоматные переходы между вершинами [1,7]
link	TrieNode	Хранит суффиксную ссылку [1, 7]
up	TrieNode	Хранит «хорошую» суффиксную ссылку [1]
char	Char	Хранит символ текущей вершины
parent	TrieNode	Хранит указатель на родительскую вершину
isLeaf	Boolean	Указывает на то, является ли вершина листом
num	Integer	Хранит номер слова, оканчивающегося в вершине

Продолжение Таблицы 3.3.

Имя	**Предполагаемый тип**	**Назначение**
id	Integer	Хранит номер вершины
leftSibling	TrieNode	Хранит указатель на левого sibling вершины [3]
rightSibling	TireNode	Хранит указатель на правого sibling вершины [3]
leftNeighbor	TireNode	Хранит указатель на левого neighbor [3]
children	Array	Хранит массив детей вершины
preliminary	Integer	Хранит предварительную позицию вершниы
modifier	Integer	Хранит модификатор позиции [3]
level	Integer	Хранит уровень текущей вершины от корня
nodeBlock	DOM element	Содержит элемент с описанием вершины
nodeElem	Konva element	Содержит элемент вершины для отрисовки дерева
nodeLine	Konva element	Содержит элемент линии до вершины родителя для отрисовки дерева
nodeLink	Konva element	Содержит элемент стрелки на суффиксную ссылку [3] для отрисовки дерева
nodeGo	Konva element	Содержит элемент стрелки на автоматный переход [3] для отрисовки дерева
nodeName	Konva element	Содержит элемент текста с названием текущей вершины для отрисовки дерева

Продолжение Таблицы 3.3.

Имя	**Предполагаемый тип**	**Назначение**
nextInPostorder	Function	Применяет переданную функцию ко всем вершинам дерева в обратном порядке
addChild	Function	Добавляет в вершину ребенка
setLeftSibling	Function	Устанавливает левого sibling [3]
setRightSibling	Function	Устанавливает правого sibling [3]
setLeftNeighbor	Function	Устанавливает левого neighbor [3]
getLeftMost	Function	Возвращает leftmost вершину [3]
getLevel	Function	Возвращает уровень текущей вершины от корня дерева

Таблица 3.4. – Описание полей и методов класса AhoCorasick

Имя	**Предполагаемый тип**	**Назначение**
counter	Integer	Счетчик текущего количества вершин. Необходим для присваивания вершинам id
root	TrieNode	Указатель на корень дерев
words	Array	Массив слов, хранящихся в автомате
actions	ActionList	Двусвязный список действий работы алгоритма
addString	Function	Функция добавления слова в автомат. Возвращает заполненный ActionList с действиями алгоритма
link	Function	Находит и возвращает суффиксную ссылку из нужной вершины

Продолжение Таблицы 3.4.

Имя	**Предполагаемый тип**	**Назначение**
go	Function	Находит и возвращает автоматный переход по заданному символу из нужной вершины
linkC	Function	То же самое что и link, но не заполняет ActionList
goC	Function	То же самое что и go, но не заполняет ActionList
up	Function	Находит и возвращает «хорошую» суффиксную ссылку из нужной вершины
check	Function	Проверяет при поиске все нужные вершины и находит вхождения
find	Function	Функция поиска вхождений слов в текст. Возвращает заполненный ActionList с действиями алгоритма

Таблица 3.5. – Описание полей и методов класса Controller

Имя	**Предполагаемый тип**	**Назначение**
actions	ActionList	Хранит список действий работы алгоритма
algo	AhoCorasick	Хранит класс AhoCorasick для работы с алгоритмом
tree	Tree	Хранит класс Tree для отрисовки дерева
algoAdd	Function	Обрабатывает добавление слова в словарь
algoFind	Function	Обрабатывает поиск вхождений
currentStep	Function	Получает текущее действие и обрабатывает его
nextStep	Function	Получает следующее действие и обрабатывает его

Продолжение Таблицы 3.5.

previousStep	Function	Получает предыдущее действие и обрабатывает его
handleStep	Function	Обрабатывает действие

Таблица 3.6. – Описание полей и методов класса Tree

Имя	**Предполагаемый тип**	**Назначение**
queue	Queue	Хранит очередь для алгоритма Уолкера
root	TrieNode	Хранит указатель на корень дерева
callPost	Function	Применяет переданную функцию ко всем вершинам дерева в порядке прохода в глубину
setX	Function	Устанавливает координату оси абсцисс дерева
setY	Function	Устанавливает координату оси ординат дерева

Таблица 3.7. – Описание полей и методов класса Queue

Имя	**Предполагаемый тип**	**Назначение**
queue	Array	Содержит список вершин для очереди
isEmpty	Function	Возвращает true если очередь пуста, иначе false
push	Function	Добавляет элемент в конец очереди
pop	Function	Возвращает и удаляет первый элемент в очереди
peek	Function	Возвращает первый элемент в очереди

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					