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

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

СОГЛАСОВАНО
Научный руководитель,
Заместитель декана по учебно-методической работе факультета компьютерных наук
доцент департамента больших данных и информационного поиска, канд. социологических. наук

_____ И.Ю. Самоненко
«13» мая 2022 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия»
профессор департамента программной инженерии, канд. техн. наук

_____ В.В. Шилов
«13» мая 2022 г.

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

Приложение для визуализации алгоритма Фараха

Техническое задание

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.10.03-01 ТЗ 01-1-ЛУ

Исполнитель:
студент группы БПИ206

_____ / Г. В. Вавилов /
«16» февраля 2022 г.

Москва 2022

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

Приложение для визуализации алгоритма Фараха

Техническое задание

RU.17701729.10.03-01 ТЗ 01-1

Листов 22

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

Москва 2022

Содержание

1. Введение

1.1. **Наименование:**

Приложение для визуализации алгоритма Фараха.

1.2. **Характеристика и область назначения:**

Алгоритм Фараха [1] используется для построения сжатого суффиксного дерева [9]. Алгоритм интересен тем, что имеет линейную временную сложность и может работать с бесконечными алфавитами.

Суффиксное дерево имеет множество применений. При помощи него можно осуществлять следующие операции:

1) За линейное время искать подстроки в строке.
2) За линейное время искать количество различных подстрок в строке.
3) За линейное время строить суффиксный массив. [10]
4) Сжатие данных. [2]

Алгоритм очень объемный и сложный для понимания. Приложение для визуализации должно помочь пользователю с формированием представления о работе алгоритма и об используемых в нем способах хранения данных.

Основная область применения – сфера образования.

2. Основания для разработки

2.1. Документ, на основании которого ведется разработка

Программа выполнена в рамках учебного плана подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденная академическим руководителем программы тема курсового проекта «Приложение для визуализации алгоритма Фараха».

2.2. Наименование темы разработки

2.2.1. **Название темы разработки на русском языке:**

Приложение для визуализации алгоритма Фараха.

2.2.2. **Название темы разработки на английском языке:**

Farach Algorithm Visualization Application

3. Назначение разработки

3.1. Функциональное назначение

Приложение наглядно показывает, как должно строиться сжатое суффиксное дерево для введенной пользователем строки символов при помощи алгоритма Фараха. Приложение визуализирует каждый этап алгоритма и объясняет выполненные шаги.

3.2. Эксплуатационное назначение

Приложение позволяет детально и наглядно рассмотреть каждый этап работы алгоритма с целью его понимания. Также визуализация должна помочь при реализации алгоритма.

Пользователь сможет посмотреть на построение сжатого суффиксного дерева для произвольной строки символов.

4. Требования к программе

4.1. Требования к функциональным характеристикам

Приложение состоит только из клиентской части, которая должна быть реализована в виде веб-приложения, запускаемого в браузере.

В приложении должно быть реализовано следующее:

1. Поле ввода для строки, по которой будет строиться сжатое суффиксное дерево.
2. Поэтапная визуализация алгоритма Фараха
   - 2.1. Разбиение строки на пары символов и дальнейшее сжатие.
     - 2.1.1. Манипуляции с символами и парами символов показываются при помощи перемещения текстовых элементов.
   - 2.2. Построение четного суффиксного дерева.
     - 2.2.1. Дерево состоит из вершин и ребер. Вершина показывается в виде круга. Ребро показывается в виде отрезка, соединяющего вершины. Ребро подписывается текстом, который оно обозначает.
   - 2.3. Построение нечетного суффиксного дерева.
     - 2.3.1. Требования к дереву аналогичны требованиям в пункте “ Построение четного суффиксного дерева”.
     - 2.3.2. Суффиксный массив представляется в виде таблицы, содержащей суффиксы и их индексы в строке.
     - 2.3.3. Построение суффиксного дерева по суффиксному массиву.
   - 2.4. Слияние деревьев.
     - 2.4.1. Требования к дереву аналогичны требованиям в пункте “ Построение четного суффиксного дерева”.
     - 2.4.2. Ребра четного дерева имеют цвет, отличный от цвета ребер нечетного дерева.
   - 2.5. Удаление двойных ребер.
3. Возможность приостановить визуализацию.
4. Возможность перейти к следующему кадру визуализации.
5. Возможность перейти к предыдущему кадру визуализации.
6. Возможность построения сжатого суффиксного дерева для новой строки без перезапуска приложения.

4.2. Требования к надежности.

4.2.1. Должна быть предусмотрена корректная обработка ошибок, связанных с некорректными действиями пользователя. В случае некорректных входных данных программа не должна завершаться аварийно или уходить в бесконечный цикл. Пользователь должен быть уведомлен о некорректных данных.

4.2.2. Во время загрузки программы должно быть обеспечено стабильное соединение с сетью интернет.

4.2.3. После загрузки программа должна иметь возможность корректной работы без соединения с сетью интернет.

4.3. Условия эксплуатации.

4.3.1. **Климатические условия эксплуатации.**

Требования к климатическим условиям эксплуатации не предъявляются.

4.3.2. **Требования к видам обслуживания**

Обслуживание не требуется.

4.3.3. **Требования к численности и квалификации персонала.**

До работы с программой допускается один оператор и любое количество наблюдателей. Оператор должен владеть базовыми навыками работы с компьютером и иметь опыт использования веб-браузера.

4.4. Требования к составу и параметрам технических средств.

Для работы программы необходимы следующие компоненты:

- Процессор: Intel Pentium 4 / Athlon 64 или более поздней версии с поддержкой SSE2
- Видеокарта GeForce GTX 470/Radeon R7 260X или лучше.
- Операционная система: Windows 7 / Windows 8 / Windows 10 / Windows 11 / MacOS X 11 и выше / Linux
- Свободное место на жёстком диске: 256 МБ.
- Оперативная память: 4ГБ.
- Доступ в интернет.
- Клавиатура.
- Мышь или заменяющее устройство ввода.

Требования обусловлены необходимостью использования веб-браузера для работы программы.

4.5. **Требования к информационной и программной совместимости.**

4.5.1. **Требования к информационным структурам на входе.**

Входные данные – строка символов, по которой требуется построить суффиксное дерево. Вводится пользователем в поле ввода на веб-странице.

4.5.2. **Требования к информационным структурам на выходе.**

В процессе работы программы должен быть визуализирован каждый этап алгоритма. Результат работы алгоритма – построенное сжатое суффиксное дерево. Данные и структуры данных, использованные во время работы алгоритма, должны быть представлены пользователю в удобном для восприятия виде.

Узлы дерева могут быть представлены в виде окружностей, ребра дерева в виде отрезков, соединяющих вершины.

4.5.3. **Требования к методам решения.**

Среда разработки: Visual Studio Code.

Используемые библиотеки и технологии: Node.js, TypeScript, Babel, Jest, D3.js.

4.5.4. **Требования к исходным кодам и языкам программирования.**

Исходный код программы должен быть написан на языках программирования HTML, CSS и TypeScript с возможностью транспиляции в JavaScript.

4.5.5. **Требования к программным средствам, используемым программой.**

Для работы приложения необходима установленная операционная система из п. 4.4 “Требования к составу и параметрам технических средств”.

Также необходим веб-браузер Firefox > 96.0.0 / Chrome > 97.0.0 или их аналоги на движках Gecko/Blink.

4.6. **Требования к маркировке и упаковке.**

Программа состоит из трех частей:

1) .html файл с разметкой страницы.
2) .css файл со стилями страницы.
3) .js файл – результат транспиляции и сборки исходного кода.

Программа должна храниться на выделенном сервере или на устройстве пользователя.

4.7. **Требования к транспортированию и хранению.**

Исходный код программы хранится в открытом репозитории сервиса GitHub.

Программа размещается на удаленном сервере, откуда при необходимости запрашивается пользователем через веб-браузер при помощи http-запросов на удаленный сервер. После загрузки программы она хранится на жестком диске пользователя. Транспортирование и хранение программы осуществляет веб-браузер.

4.8. **Специальные требования**

Специальные требования к программе не предъявляются

5. Требования к программной документации

5.1. Предварительный состав программной документации

5.1.1. «Приложение для визуализации алгоритма Фараха» Техническое задание (ГОСТ 19.201-78) [4]

5.1.2. «Приложение для визуализации алгоритма Фараха» Программа и методика испытаний (ГОСТ 19.301-79) [5]

5.1.3. Приложение для визуализации алгоритма Фараха» Пояснительная записка (ГОСТ 19.404-79) [7]

5.1.4. «Приложение для визуализации алгоритма Фараха» Текст программы (ГОСТ 19.401-78) [6]

5.1.5. «Приложение для визуализации алгоритма Фараха» Руководство оператора (ГОСТ 19.505-79) [8]

6. Технико-экономические показатели

6.1. Предполагаемая потребность

Программа будет использоваться преподавателями и учениками технических ВУЗов в образовательных целях. Также программа будет актуальна для людей, занимающихся спортивным программированием.

6.2. Ориентировочная экономическая эффективность

Программа существенно упрощает восприятие и понимание алгоритма, что будет экономить время, затрачиваемое на обучение.

6.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами.

Название	Algorithm Visualizer	Vamonos	Visualgo	Data Structure Visualizations	Приложение для визуализации
Визуализация алгоритма Фараха	-	-	-	-	+
Визуализация алгоритма построения суффиксного дерева	-	-	+	-	+
Пошаговый просмотр	+	+	+	+	+
Приостановка	+	+	+	+	+
Представление данных в удобном для человека виде	+	+	+	+	+
Возможность задать входные данные	+	+	+	+	+

Ни один из аналогов не предоставляет возможность визуализации алгоритма Фараха для построения сжатого суффиксного дерева. В этом заключается главное преимущество данного программного продукта.

7. Стадии и этапы разработки

7.1. Техническое задание

7.1.1. Разработка технического задания.

7.1.2. Согласование технического задания.

7.1.3. Утверждение технического задания.

7.2. Рабочий проект

7.2.1. Изучение визуализируемого алгоритма.

7.2.2. Программирование.

7.2.3. Отладка.

7.2.4. Написание тестов.

7.2.5. Разработка программной документации в соответствии с требованиями ГОСТ 19.101-77 [3].

7.2.6. Испытание программы.

7.2.7. Корректировка программы по результатам испытания.

7.3. Внедрение

7.3.1. Подготовка презентации и комплекта документов для защиты.

7.3.2. Защита проекта.

8. Сроки разработки и исполнители

8.1. Сроки

- Контрольная точка 1. (17.02.2022)
    - Подготовить техническое задание «Приложение для визуализации алгоритма Фараха» (ГОСТ 19.201-78) [4]
- Защита проекта. (05.2022 – 06.2022)

8.2. Исполнитель

Студент группы БПИ206 Г. В. Вавилов

9. Порядок контроля и приемки

9.1. Виды испытаний.

Тестируется правильность выполнения программой выполняемого ей функционала.

Тестирование осуществляется в соответствии с документом «Программа и методика испытаний» (ГОСТ 19.301-79) [5].

9.2. Общие требования к приемке работы.

Работа принимается в случае полной работоспособности программы и успешного прохождения тестирования.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Алгоритм Фараха [Электронный ресурс] : Викиконспекты / Университет ИТМО – Электрон. текст. дан. – Москва: ИТМО, 2010 – URL: http://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Фараха, свободный (дата обращения 23.12.2021)
2. Алгоритмы LZ77 и LZ78 [Электронный ресурс] : Викиконспекты / Университет ИТМО – Электрон. текст. дан. – Москва: ИТМО, 2010 – URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритмы_LZ77_и_LZ78, свободный (дата обращения 30.01.2022)
3. ГОСТ 19.101-77 Виды программ и программных документов. Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению. Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.401-78 Текст программы. Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.505-79 Руководство оператора. Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. Сжатое суффиксное дерево [Электронный ресурс] : Викиконспекты / Университет ИТМО – Электрон. текст. дан. – Москва: ИТМО, 2010 – URL: https://neerc.ifmo.ru/wiki/index.php?title=Сжатое_суффиксное_дерево, свободный (дата обращения 30.01.2022)
10. Суффиксный массив [Электронный ресурс] : Викиконспекты / Университет ИТМО – Электрон. текст. дан. – Москва: ИТМО, 2010 – URL: https://neerc.ifmo.ru/wiki/index.php?title=Суффиксный_массив, свободный (дата обращения 30.01.2022)

Приложение 1

10. СПИСОК ТЕРМИНОВ

10.1. Суффикс строки

Подстрока данной строки, в которую входит последний символ строки.

10.2. Суффиксное дерево

Дерево, которое содержит все суффиксы данной строки. Каждое ребро дерева помечено одним символом. Каждый узел не может содержать два одинаковых исходящих ребра.

10.3. Сжатое суффиксное дерево

Суффиксное дерево, в котором ребра могут быть помечены подстрокой данной строки. Каждая вершина не может содержать исходящие ребра с одинаковыми первыми символами.

10.4. Четное суффиксное дерево

Сжатое суффиксное дерево, которое содержит суффиксы строки, находящиеся на четных позициях.

10.5. Нечетное суффиксное дерево

Сжатое суффиксное дерево, которое содержит суффиксы строки, находящиеся на нечетных позициях.

10.6. Временная сложность

Функция, которая зависит от размера входных данных. Представляет время работы алгоритма.

10.7. Линейная временная сложность

Время работы алгоритма пропорционально функции вида $y = kx+b$, где x – размер входных данных.

10.8. Алфавит

Множество символов, из которых составлена строка.

10.9. Транспиляция

Эквивалентное преобразование кода программы из одного языка программирования в другой с полным сохранением работоспособности.

Приложение 2

11. ОПИСАНИЕ АЛГОРИТМА ФАРАХА ДЛЯ ПОСТРОЕНИЯ СЖАТОГО СУФФИКСНОГО ДЕРЕВА

Алгоритм выполняется рекурсивно и состоит из пяти этапов.

1) Сжатие алфавита.
2) Построение четного суффиксного дерева.
3) Построение нечетного суффиксного дерева.
4) Слияние четного и нечетного деревьев.
5) Удаление двойных ребер.

11.1. Этап 1. Сжатие алфавита

На вход принимается строка.
Возвращается построенное суффиксное дерево.

Если длина строки равна 1, то суффиксное дерево для нее тривиально (рис. 1). Дерево возвращается, шаг 1 завершается.

Рисунок 1 - Тривиальное дерево

Если длина строки нечетна, в конец строки добавляется специальный символ, который не равен ни одному из символов текущего алфавита.
Строка разбивается на пары подряд идущих символов.
Пары символов сортируются лексикографически при помощи поразрядной сортировки.
Удаляются копии пар.
В исходной строке пары символов заменяются на индексы этих пар в массиве, полученном на предыдущем шаге.
Получилась строка длиной в два раза меньше, чем была изначально.
Выполнить этап 1 для сжатой строки.
Выполнить этап 2. Построить четное дерево.
Выполнить этап 3. Построить нечетное дерево.
Выполнить этап 4. Произвести слияние четного и нечетного деревьев.
Выполнить этап 5. Удалить двойные ребра в полученном дереве.

11.2. Этап 2. Построение четного дерева

На вход принимается суффиксное дерево для сжатой строки, полученное на предыдущем этапе.

Возвращается четное суффиксное дерево, построенное для текущей строки.

Каждый символ из ребер суффиксного дерева заменяется обратно на его пару символов. Массив с парами был получен на предыдущем этапе.

После раскрытия пар могут возникнуть ребра с одинаковыми первыми символами, принадлежащие одной вершине. Создаем новую вершину между такой вершиной и ребрами с одинаковым первым символом (рис. 2).

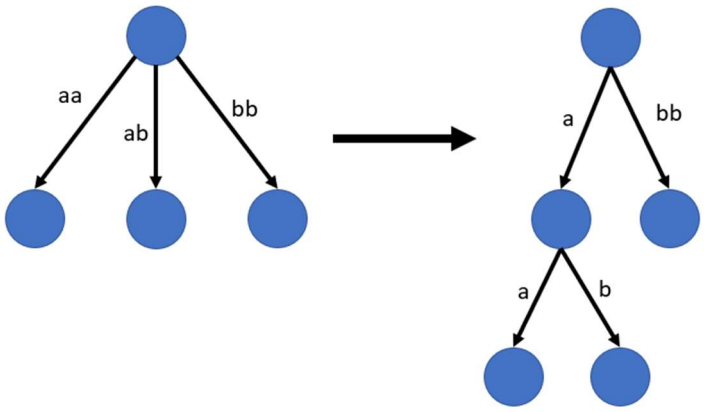

Рисунок 2 - Исправление ребер с одинаковыми первыми символами

Четное суффиксное дерево построено.

11.3. Этап 3. Построение нечетного дерева

На вход принимается четное суффиксное дерево для сжатой строки, полученное на предыдущем этапе.

Возвращается нечетное суффиксное дерево, построенное для сжатой строки.

Строим суффиксный массив по нечетному дереву.

Дописываем к каждому суффиксу в массиве символ, предшествующий этому суффиксу в изначальной строке.

Сортируем суффиксы по дописанному символу.

Получили суффиксный массив для нечетного дерева.

Строим нечетное суффиксное дерево по суффиксному массиву.

11.4. Этап 4. Слияние деревьев

На вход принимается четное и нечетное дерево, полученные на двух предыдущих этапах.

Возвращается новое дерево – результат слияния.

Рекурсивно выполняется следующий алгоритм:

Имеем два указателя на корни четного и нечетного поддерева и указатель на корень нового поддерева.

Просматриваем ребра по одному в каждой вершине.

- Если первые символы ребер различны, копируем ребро с меньшим первым символом в новое дерево. Переходим к следующему ребру в вершине, ребро которой было скопировано.
- Если первые символы ребер одинаковы и длины строк в ребрах равны, копируем оба ребра в новое дерево. Переходим к следующему ребру в обоих вершинах.
- Если первые символы ребер одинаковы и длины строк в ребрах различаются, в новое дерево добавляются оба ребра. Создается дополнительная вершина, которая разбивает большее ребро на два ребра. Первое ребро содержит префикс длины строки в меньшем ребре.

Рекурсивно спускаемся в следующую вершину после ее добавления в новое дерево.

11.5. Этап 5. Удаление двойных ребер

На вход принимается дерево, в котором могут быть двойные ребра, полученное на предыдущем этапе.

Возвращается дерево без двойных ребер.

Рекурсивно проходится по всем вершинам дерева, и выполняем следующие действия для каждого двойного ребра:

Если два ребра имеют полностью одинаковые строки – удаляем одно из ребер.

Иначе находим общий префикс и создаем новую вершину и ребро с этим префиксом, ведущее в эту вершину.

Из новой вершины выпускаем два ребра, к которых будут находиться строки двойного ребра без общего префикса.

Разделяем ребра, выходящие из двойного ребра на четные и нечетные и присоединяем их к соответствующему четному/нечетному новому ребру. (рис. 3)

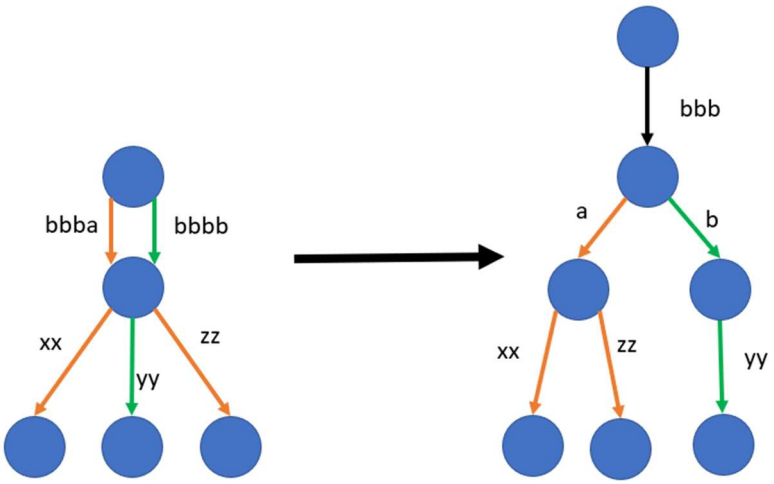

Рисунок 3 - Удаление двойного ребра

Лист регистрации изменений

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					