Лабораторная работа № 5 по курсу дискретного анализа: линеаризация циклической строки

Выполнил студент группы М80-308Б-18 МАИ *Марков Александр.*

Условие

1. Общая постановка задачи

   Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из выходных строк, необходимо воспользоваться полученным суффиксным деревом для решения своего варианта задания.

   Алфавит строк: строчные буквы латинского алфавита (т.е. от a до z).

2. Вариант задания

   Вариант 4. Вариант алгоритма: Линеаризовать циклическую строку, то есть найти минимальный в лексикографическом смысле разрез циклической строки.

Метод решения

Алгоритм Укконена реализован следующим образом. Каждый узел суффиксного дерева содержит целочисленные переменные для левого и правого концов подстроки, указатель на int, необходимый для неявного изменения правого конца подстроки листа (для неявных добавлений первого типа), суффиксную ссылку, которая либо указывает на вершину с таким же суффиксом только без первого символа, либо при отсутствии вершины на nullptr, указатель на родителя, булевская переменная, помечающая является ли узел листом, целочисленная переменная, содержащая номер листа. Также есть словарь с дочерними узлами для данной вершины. В дереве хранится текст, в конец которого добавляется терминальный символ, указатель на корень, указатель на последний элемент, с которым выполнялось добавление. Этот указатель необходим для добавления суффиксных ссылок.

При создании дерева сначала добавляем первый суффикс. Начинаем с первой фазы. Сначала происходит поиск места, в котором должно происходить добавление. Поиск может привести к следующим результатам:

- Завершение на ребре. Могут быть случаи добавления 2 и 3. Если случай 2, то создастся вершина на ребре и дочерний узел для этой вершины. Такой дочерний узел будет листом. Если случай 3, то фаза заканчивается.

- Завершение в вершине. В моей реализации случай 1 невозможен, т.к. сразу в начале фазы добавляется конец суффикса, соответствующего данной фазе. А значит если поиск закончится в листе, то это приведет только к случаю 3. Случай 3

заканчивает фазу. Если поиск закончился во внутренней вершине, то создается дочерний узел для этой вершины, который становится листом.

После случая 2, если длина подстроки вершины больше единицы или равна единице, но родителем этой вершины не является корень, то должна создаться суффиксная связь между этой вершиной и вершиной, которая либо создается в следующем продолжении, либо уже существует, поэтому переменная do_link устанавливается в true. Иначе создается суффиксная связь с корнем.

Линеаризация циклической строки: Подается строка S.

1. Строю суффиксное дерево по удвоенной строке SS$, где $ - терминальный символ.
2. Прохожу дерево таким образом, что в каждой вершине прохожу по дуге с символом наименьшем в лексикографическом порядке.

Описание программы

Проект состоит из 3 файлов:

- main.cpp - главный файл, в котором реализована функция main
- suffix_tree.hpp - заголовочный файл, в котором находятся объявления классов вершины дерева и суффиксного дерева.
- suffix_tree.cpp - файл, в котором содержатся реализации методов классов вершины дерева и суффиксного дерева.

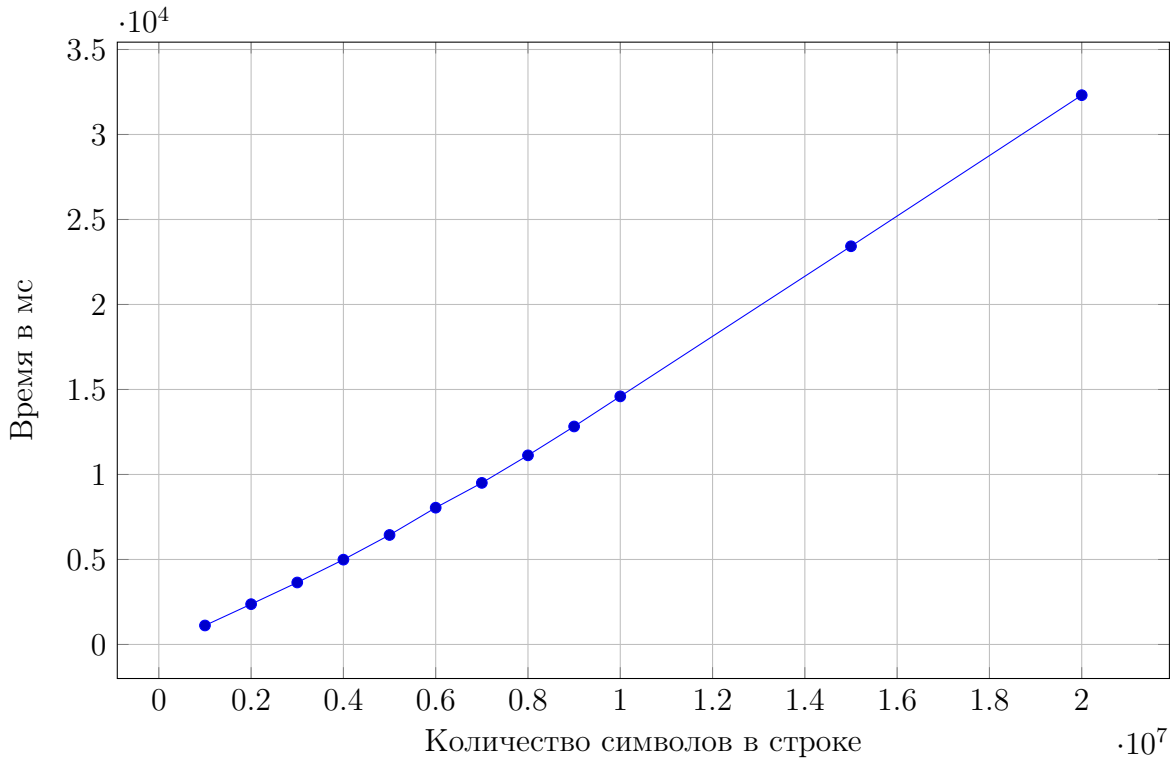
Дневник отладки

При создании этой таблицы была использована история посылок.

№	Время	Проблема	Описание
1-8	2020/10/30	RE	Если фаза заканчивалась добавлением case3, то вершина, в которой завершался поиск, присваивалась last_node. Но в analysis_links, не учитывалось, что фаза может завершаться раньше времени. Поэтому если в last_node была суффиксная ссылка, то в следующей фазе при поиске происходил переход по этой суффиксной ссылке, хотя этого происходить не должно было. Поэтому дерево строилось неправильно и при некоторых тестовых данных программа падала с segmentation fault.
9	2020/10/31	Ожидает подтверждения	Ошибка была исправлена, путем того, что в случае завершения предыдущей фазы case3, поиск начинался с корня. Хоть чекер принял такое решение, но оно не оптимально.
10-11	2020/10/31	RE	При оптимазации возникали ошибки в реализации построения суффиксного дерева, из-за которых программа падала с segmentation fault.
12	2020/01/08	Ожидает подтверждения	Все ошибки были исправлены.

Тест производительности

Тесты создавались с помощью небольших программы generator.py. Создавались строки с n-ым количеством символов.



Выводы

В данной лабораторной работе был реализован алгоритм построения суффиксного за линейное время, а также алгоритм линеаризации циклической строки.

Так как я использовал словарь в вершинах для хранения дочерних узлов, построение суффиксного дерева имеет следующую временную оценку $O(m * logk)$, где m - количество символов в тексте, k - размер алфавита.

Сложность по времени алгоритма линеаризации циклической строки равна $O(2n * logk + n * logk) = O(n * logk)$, где n - длина изначального среза, k - размер алфавита.

Суффиксное дерево показалось мне одной из самых сложных структур данных для понимания. Но его можно применять для множества задач таких, как линеаризацию циклической строки, количество подстроки в текст, общие подстроки. Все эти задачи решаются за линейное время.