Лабораторная работа № 5 по курсу: Дискретный анализ

Выполнил студент группы М8О-307Б-17 МАИ *Лопатин Александр.*

Задача

Вариант 2 : Поиск с использованием суффиксного массива
Найти в заранее известном тексте поступающие на вход образцы с использованием суффиксного массива. Формат входных и выходных данных, а так же примеры аналогичны указанным во первом задании.

Входные данные: Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

Выходные данные: Для каждого образца, найденного в тексте, нужно распечатать строчку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания.

Информация

Суффиксное дерево (сжатое суффиксное дерево) T для строки s (где |s| = n) — дерево с n листьями, обладающее следующими свойствами:

1. каждая внутренняя вершина дерева имеет не меньше двух детей;
2. каждое ребро помечено непустой подстрокой строки s;
3. никакие два ребра, выходящие из одной вершины, не могут иметь пометок, начинающихся с одного и того же символа;
4. дерево должно содержать все суффиксы строки s, причем каждый суффикс заканчивается точно в листе и нигде кроме него.

Суффиксным массивом (англ. suffix array) строки $s[1..n]$ называется массив suf целых чисел от 1 до n, такой, что суффикс $s[suf[i]..n]$ — i-й в лексикографическом порядке среди всех непустых суффиксов строки s.

Метод решения

Можно построить суффиксное дерево с помощью алгоритма Укконена за линейное время, потом пройтись по листам (чтобы каждый суффикс имел свой лист, добавим в конец строки символ, который ранее в тексте не встречался) и в лексикографическом порядке добавить их в суффиксный массив. Так как в дальнейшем мы будем работать только с суффиксным массивом, удалим из памяти суффиксное дерево. Далее,

на каждый запрос поиска образца с помощью бинарного поиска находим левую и правую границы в суффиксном массиве всех суффиксов, для которых образец является префиксом, и сортируем их по длине суффикса (т.к. по условию задания нужно перечислить номера позиций, где встречается образец в порядке возрастания). Алгоритм Укконена и построение суффиксного массива имеют асимптотическую сложность $O(n)$, бинарный поиск и сортировка номеров позиций в сумме будут иметь асимпотитечкую сложность $O(p * logn + q * logq)$, где n - длина исходной строки, p - длина образца, q - длина интервала результата работы бинарного поиска (сортировка занимает $O(q * logq)$ времени).

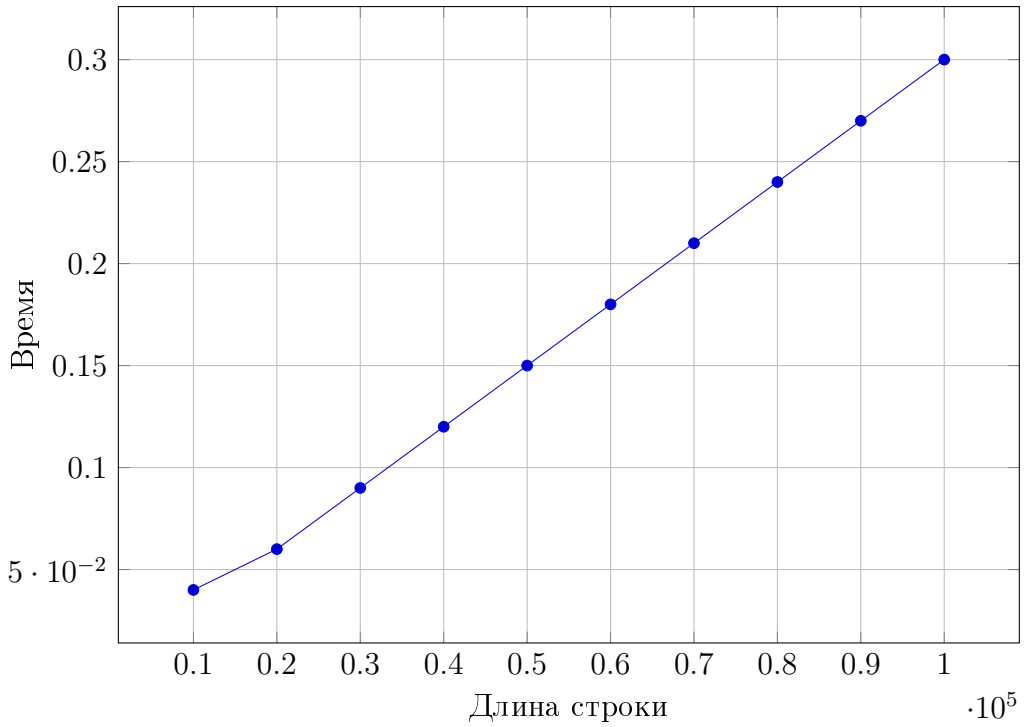
Генератор тестов

tests.py

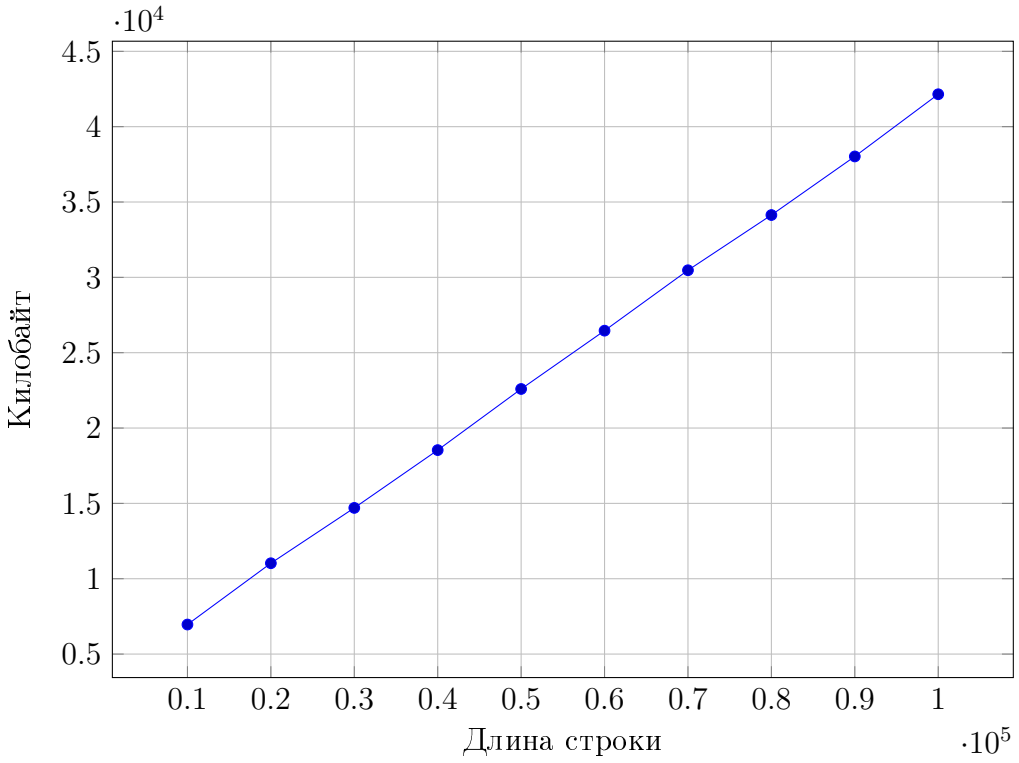
```
import random

for i in range(10000, 110000, 10000):
    file = open('tests/test'+str(i)+'.txt','w')
    for j in range(1, i+1):
        file.write(str(chr(random.randint(97, 122))))
    file.close()
```

Проверка производительности

Время построения суффиксного дерева с помощью алгоритма Укконена:



Использование памяти:



Выводы

Алгоритм Укконена позволяет построить суффиксное дерево для заранее известного текста за линейное относительно длины строки время. Плюсом является то, что это on-line алгоритм, который дополняет на каждом шаге суффиксное дерево следующим символом. Причем, как было сказано ранее, текст заранее известен, поэтому он хорошо подходит для задач, когда шаблоны поиска заранее неизвестны. Большой минус этого алгоритма в том, что он требователен к памяти: для каждого следующего шага ему требуется содержать все суффиксное дерево в оперативной памяти, однако можно модифицировать алгоритм, чтобы он не хранил все дерево в ней, но это не так и просто реализовать. Можно заметить, что алгоритм Укконена на самом деле работает за время $O(k*n)$ и использует столько же памяти, где k - размер алфавита. Поэтому, если алфавит очень большой требуется чрезмерный объем памяти. Можно сэкономить на памяти, храня в каждой вершине только те символы, по которым из нее есть переходы, но тогда поиск перехода будет занимать $O(logk)$ времени.