Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: А. В. Барсов
Преподаватель: Н. С. Капралов
Группа: М8О-307Б-19

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №4

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

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

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

1 Описание

Требуется реализовать алгоритм Укконена построения суффиксного дерева за линейное время.

Алгоритм Укконена в самой простоей реализации имеет сложность $O(n^3)$, так как добавляет каждый суффикс каждого префикса строки в отличии от добавления всех суффиксов строки. Основная идея в том, чтобы оптимизировать его и получить сложность $O(n)$.

Заметим, что проще будет продлевать суффиксы на один символ. В некоторых случаях при продлении можно идти не по всем символам, а сразу по ребру дерева, что значительно уменьшает число шагов.

Пусть в суффиксном дереве есть строка xa (x — первый символ строки, a — оставшаяся строка), тогда a тоже будет в суффиксном дереве, потому что a является суффиксом xa. Если для строки xa существует некоторая вершина u, то существует и вершина u для a. Ссылка из u в v называется суффиксной ссылкой.

Суффиксные ссылки позволяют не проходить каждый раз по дереву из корня. Для построения суффиксных ссылок достаточно хранить номер последней созданной вершины при продлении. Если на этой же фазе мы создаём ещё одну новую вершину, то нужно построить суффиксную ссылку из предыдущей в текущую.

Сложность алгоритма с использованием продления суффиксов и суффиксных ссылок $O(n^2)$. Подробное доказательство изложено в [1].

Для ускорения до $O(n)$ нужно уменьшить объём потребляемой памяти. Будем в каждом ребре дерева хранить не подстроку, а только индекс начала и конца подстроки.

У исходной строки n суффиксов и будет создано не более n внутренних вершин, в среднем продление суффиксов работает за $O(1)$.

При использовании всех вышеописанных эвристик получим временную и пространственную сложность $O(n)$.

Для нахождения минимального лексикографического разреза строки s построим суффиксное дерево от удвоенной строки и найдём лексикографически минимальный путь длины |s| в дереве. Сложность $O(n)$.

2 Консоль

```
[artem@IdeaPad solution]$ make
g++ -pedantic -Wall -std=c++11 -Werror -Wno-sign-compare -O2 -lm -c main.cpp
g++ -pedantic -Wall -std=c++11 -Werror -Wno-sign-compare -O2 -lm -c suffix_tree.cpp
g++ -pedantic -Wall -std=c++11 -Werror -Wno-sign-compare -O2 -lm main.o suffix_tree.o
-o solution
[artem@IdeaPad solution]$ ./solution
xabcd
abcdx
[artem@IdeaPad solution]$ ./solution
abracadabra
aabracadabr
[artem@IdeaPad solution]$ ./solution
abaacaababaab
aababaababaac
```

3 Выводы

Во время выполнения лабораторной работы я вспомнил префиксное дерево, способы хранения графов, реализовал алгоритм Укконена и ознакомился с приложениями суффиксного дерева.

Сложнее всего было понять, как алгоритм достигает линейной сложности и как работают суффиксные ссылки. Помогли рисунки в тетради и визуализатор [2].

Суффиксное дерево позволяет быстро искать множество шаблонов в тексте, чего не могут другие алгоритмы. Но в повседневных задачах чаще требуется найти один шаблон в тексте, где лучше использовать более простые алгоритмы: алгоритм КнутаМорриса-Пратта, Бойера-Мура, Рабина-Карпа.

Список литературы

[1] *Алгоритм Укконена* — Викиконспекты
URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Укконена .

[2] *Visualization of Ukkonen's Algorithm.*
URL: http://brenden.github.io/ukkonen-animation/ .