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: Р. С. Лисин
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б-20

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №5

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

Формат входных данных Некий разрез циклической строки.

Формат результата Минимальный в лексикографическом смысле разрез.

1 Описание

Согласно [1], суффиксное дерево для m-символьной строки S - это ориентированное дерево с корнем, имеющее ровно m листьев, занумерованных от 1 до m. Каждая внутренняя вершина, отличная от корня, имеет не меньше двух детей, а каждая дуга помечена непустой подстрокой строки S. Никакие две дуги, выходящие из одной и той же вершины, не могут иметь пометок, начинающихся с одного и того же символа. Главная особенность суффиксного дерева заключается в том, что для каждого листа i конкатенация меток дуг на пути от корня к листу i составляет суффикс строки S, которые начинается в позиции i.

Для построения суффиксного дерева используется алгоритм Укконена. Каждую дугу дерева будем помечать не подстрокой, а индексами границ этой подстроки. Алгоритм состоит из нескольких фаз. Количество фаз совпадает с количеством символов в строке.

В структуре будем хранять активную вершину. Это положение, в которое мы будем вставлять следующий суффикс. Для однозначного определения активной вершины у нас будет три компоненты: сама активная вершина, активное ребро (ребро, по которому нужно перейти) и активная длина (количество элементов, через которое нужно перепрыгнуть на активном ребре.

Если мы вставили новый лист в дерево, то он останется листом навсегда. Поэтому удобно вместо правой границы этого узла использовать глобальную переменную end, которая будет увеличиваться на каждом шаге и всегда будет «указывать» на конец строки. Для обновления значения end нужно инкрементировать эту переменную при старте каждой фазы.

Введем понятие суффиксной ссылки. Пусть $x\alpha$ обозначает произвольную строку, где x - ее первый символ, а α - оставшаяся подстрока (возможно, пустая). Если для внутренней вершины v с путевой меткой $x\alpha$ существует другая вершина $s(v)$ с путевой меткой α, то указатель из v в $s(v)$ называется суффискной ссылкой. Суффиксная ссылка нужна для быстрого перехода по дереву при вставке суффиксов.

Если при вставки мы не нашли вершины, начинающейся с вставляемого суффикса, то создаем новую вершину. Иначе спускаемся в эту вершину и смотрим, содержится ли наш суффикс целиком. Если да, то завершаем фазу, иначе создаем новую внутреннюю вершину и добавляем суффикс. После каждого создания узла необходимо обновить суффиксные ссылки, а после каждой фазы - активную вершину.

Сложность алгоритма Укконена - $O(n)$ по времени и по памяти.

Для нахождения минимального разреза циклической строки необходимо построить суффиксное дерево для удвоенной строки, выполнить его обход, всегда переходя по лексикографически минимальному ребру. Как только мы достигли листовой вершины, мы отсекаем вторую половину полученной строки и получаем ответ.

2 Исходный код

Создадим класс для вершины суффиксного дерева $TNode$. Его атрибуты - хеш-таблица с ребрами, суффиксная ссылка, позиции начала и конца подстроки и лексикографически минимальное ребро. Для поиска минимального разреза за линейное время будем находить для каждой вершины минимальное ребро при построении дерева.

Перейдем к классу самого дерева $TSuffixTree$. Его атрибуты - сама строка $text$, глобальная переменная для обозначения текущего конца строки $EndPos$, корень $Root$, активная вершина $ActiveNode$, активное ребро $ActiveEdge$, активная длина $ActiveLength$ и количество суффиксов $Remaining$, которое нужно вставить. При создании объекта класса построение дерева выполняется автоматически. Для полноты дерева к концу строки добавляется сентинел (\$).

Выполнение одной фазы алгоритма описано в методе $Extend$. Там реализован алгоритм, описанный выше. Для нахождения лексикографически минимального разреза используется метод $FindLexMinString$, который по сути просто выполняет проход по дереву.

```
1  class TNode {
2  public:
3      TNode(int left = 0, int right = 0) : start(left), end(right) {}
4
5      std::unordered_map<char, TNode*> edges;
6      TNode* suffixLink = nullptr;
7
8      int start;
9      int end;
10
11     char lexMinEdge = 0; // edge with lexicographically minimal char
12 };
13
14
15 class TSuffixTree {
16 public:
17     std::string text;
18     int endPos = -1; // variable for end position
19
20     TNode* root = new TNode();
21
22     // a point where we will start the next phase
23     TNode* activeNode = root; // start node
24     int activeEdge = -1; // id of the edge
25     int activeLength = 0; // how far should we go along the edge
26
27     int remaining = 0; // amount of suffixed we should add
28
```

```
29 ||
30 ||     TSuffixTree(std::string& str);
31 ||     ~TSuffixTree();
32 ||
33 ||     void Extend(int phase);
34 ||
35 ||     std::string FindlexMinString();
36 ||
37 ||     void Delete(TNode* node);
38 ||
39 || };
```

3 Консоль

```
roma@DESKTOP-JD58QU2:~/Diskran/lab5$ ./solution
xabcd
abcdx
roma@DESKTOP-JD58QU2:~/Diskran/lab5$ ./solution
abxabc
abcabx
```

4 Тест производительности

Сравним полученный алгоритм с наивным алгоритмом. Наивный алгоритм просто перебирает все возможные варианты разрезов и ищет среди них лексикографически наименьший.

Для сравнения я подготовил 3 теста. Первый тест - со строкой длины 500 символов, второй тест - 10^4 символов, третий - 10^6.

```
roma@DESKTOP-JD58QU2:~/Diskran/lab5$ ./suftree <test500
Time: 0.001716 ms
roma@DESKTOP-JD58QU2:~/Diskran/lab5$ ./naive <test500
Time: 0.00017 ms
roma@DESKTOP-JD58QU2:~/Diskran/lab5$ ./suftree <test1e4
Time: 0.0329882 ms
roma@DESKTOP-JD58QU2:~/Diskran/lab5$ ./naive <test1e4
Time: 0.0042473 ms
roma@DESKTOP-JD58QU2:~/Diskran/lab5$ ./suftree <test1e6
Time: 4.08355 ms
roma@DESKTOP-JD58QU2:~/Diskran/lab5$ ./naive <test1e6
Time: 73.1969 ms
```

Видим, что на маленьких строках суффиксное дерево немного уступает наивному алгоритму. Это связано с тем, что построение дерева тоже требует времени. Однако на большой строке видно все преимущество суффиксных деревьев - выигрыш примерно в 18 раз. Это связано с тем, что наивный алгоритм работает за $O(n^2)$ - мы проверям n разрезов, каждый из которых сравниваем с текущим минимальным (сравнение строк происходит за $O(n)$.

5 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», я познакомился с такой структурой данных как суффиксное дерево. Основное ее предназначение - поиск образцов в тексте. Поиск отличается от всеми известных алгоритмов КМП, Ахо-Корасик и др. тем, что мы предобрабатываем текст, а не образец.

Однако у суффиксных деревьев есть еще много применений. Благодаря ему мы можем за линейное время найти наибольшую общую подстроку, можем рассчитать статистику совпадений, построить суффиксный массив или найти минимальный лексикографический разрез.

Список литературы

[1] Дэн Гасфилд. *Строки, деревья и последовательности в алгоритмах.* — БХВ-Петербург, 2003. Перевод с английского: И. В. Романовский. — 658 с. (ISBN 5-94157-321-9)