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: Т. А. Габдуллин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №5

Задача: Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из входных строк, необходимо воспользоваться полученным суффиксным деревом для решения своего варианта задания. Алфавит строк: строчные буквы латинского алфавита (т.е., от a до z).

Вариант алгоритма: 2. Поиск с использованием суффиксного массива

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.

1 Алгоритм Уккокена

Рассмотрим сначала наивный метод, который строит дерево за время $\emptyset(n^3)$, где n — длина исходной строки s. В дальнейшем данный алгоритм будет оптимизирован таким образом, что будет достигнута линейная скорость работы.

Алгоритм последовательно строит неявные суффиксные деревья для всех префиксов исходного текста $S = s_1 s_2 \dots s_n$. На i-ой фазе неявное суффиксное дерево ${}_i1$ для префикса $s[1 \dots i1]$ достраивается до ${}_i$ для префикса $s[1 \dots i]$. Достраивание происходит следующим образом: для каждого суффикса подстроки $s[1 \dots i1]$ необходимо спуститься от корня дерева до конца этого суффикса и дописать символ s_i.

Алгоритм состоит из n фаз. На каждой фазе происходит продление всех суффиксов текущего префикса строки, что требует $\emptyset(n^2)$ времени. Следовательно, общая асимптотика алгоритма составляет $\emptyset(n^3)$.

Чтобы улучшить время работы данного алгоритма до $\emptyset(n)$, нужно использовать линейное количество памяти, поэтому метка каждого ребра будет храниться как два числа — позиции её самого левого и самого правого символов в исходном тексте.

Линейная оценка: $\emptyset(n)$
Описание взято с сайта *neerc.ifmo.ru*.

2 Суффиксный массив

Строго говоря, описываемый ниже алгоритм будет выполнять сортировку не суффиксов, а циклических сдвигов строки. Однако из этого алгоритма легко получить и алгоритм сортировки суффиксов: достаточно приписать в конец строки произвольный символ, который заведомо меньше любого символа, из которого может состоять строка (например, это может быть доллар или шарп; в языке C в этих целях можно использовать уже имеющийся нулевой символ).

Сразу заметим, что поскольку мы сортируем циклические сдвиги, то и подстроки мы будем рассматривать циклические: под подстрокой $s[i \dots j]$, когда $i > j$, понимается подстрока $s[i \dots n-1] + s[0 \dots j]$. Кроме того, предварительно все индексы берутся по модулю длины строки (в целях упрощения формул я буду опускать явные взятия индексов по модулю).

Рассматриваемый нами алгоритм состоит из примерно $\log n.k - (k = 0 \dots \lceil \log n \rceil)2^k$. На последней, $\lceil \log n \rceil - , 2^{\lceil \log n \rceil} > n$, что эквивалентно сортировке циклических сдвигов.

На каждой фазе алгоритм помимо перестановки $p[0 \dots n-1]$ индексов циклических подстрок будет поддерживать для каждой циклической подстроки, начинающейся в позиции i с длиной 2^k, номер $c[i]$ класса эквивалентности, которому эта подстрока принадлежит. В самом деле, среди подстрок могут быть одинаковые, и алгоритму понадобится информация об этом. Кроме того, номера $c[i]$ классов эквивалентности будем давать таким образом, чтобы они сохраняли и информацию о порядке: если один суффикс меньше другого, то и номер класса он должен получить меньший. Классы будем для удобства нумеровать с нуля. Количество классов эквивалентности будем хранить в переменной classes.

3 Листинг

1. main.cpp

```cpp
#include <bits/stdc++.h>

#include "tsuffixtree.h"
using namespace std;
int main() {
    TSuffixTree tree;

    ios::sync_with_stdio(false);
    string str;
    cin >> str;

    tree.Build(str);

    size_t cnt = 1;
    while (cin >> str) {
        vector<size_t> result;
        tree.Find(str, result);

        if (!result.empty()) {
            cout << cnt << ": ";

            for (size_t i = 0; i < result.size() - 1; ++i) {
                cout << result[i] + 1 << ", ";
            }

            cout << result[result.size() - 1] + 1 << "\n";
        }

        ++cnt;
    }
}
```

2. TSuffixTree.h

```cpp
#ifndef TSUFFIXTREE_H
#define TSUFFIXTREE_H
#include <bits/stdc++.h>

const int LEAF_LENGTH = 10e8;

class TNode {
public:
    std::map<char, TNode *> children;

    TNode *parent;
    TNode *suffixLink;
```

```
13
14      std::size_t first;
15      std::size_t length;
16
17      TNode();
18      TNode(const std::string &, TNode *&node, std::size_t &pos);
19
20      ~TNode();
21  };
22
23  class TSuffixTree {
24      std::string str;
25      TNode *root;
26      TNode *activeEdge;
27      std::size_t activeLength;
28      std::size_t reminder;
29
30      void DeepFirstSearch(TNode *, std::vector<std::size_t> &, std::size_t &total)
            ;
31  public:
32      TSuffixTree();
33
34      void Build(const std::string &str);
35      void Insert(char c);
36      void Find(const std::string &pattern, std::vector<std::size_t> &result);
37
38      ~TSuffixTree();
39  };
40
41  #endif //TSUFFIXTREE_H
```

4 Выводы

Благодаря данной лабораторной работе, я познакомился с Алгоритмом Уккокена, и с таким определением, как суффиксный массив, который является массивом лексикографически отсортированных суффиксов строк. А так же узнал про наивную реализацию суффиксного массива, и его более быстрый аналог через *lcp(longest common prefix.*

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 2-е издание. — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Суффиксный массив*
URL:`http://www.e-maxx-ru.1gb.ru/algo/suffix`$_a rray$
URL : `http://neerc.ifmo.ru/wiki/index.php?title=Алгоритм-Уккокена`