Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: А. А. Пустовалова
Преподаватель: А. А. Кухтичев
Группа: 8О-206
Дата:
Оценка:
Подпись:

Москва, 2016

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 (от a до z).

Вариант: Линеаризация циклической строки.

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

Входные данные: некий разрез циклической строки.

Выходные данные: минимальный в лексикографическом смысле разрез.

1 Описание

Требуется реализовать алгоритм Укконена построения суффиксного дерева для заданной строки за линейное время. Суффиксное дерево для m-символьной строки S с терминальным элементом — 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. [1] Слиянием строк дуг на пути от корня до листа с номером i является суффикс данной строки с позиции i. Неявное суффиксное дерево представляет собой суффиксное дерево строки без терминального элемента. На каждой из m итераций алгоритм Укконена строит неявное суффиксное дерево для префикса строки S. Для этого необходимо для каждого суффикса подстроки $S[1..i-1]$ пройти по дереву от корня и добавить символ $S[i]$. При добавлении символа рассматриваются три случая. Если текущий символ – лист, просто продливаем данный суффикс, добавляя нужный элемент. Если текущий символ не лист, не имеет пути, помеченного добавляемым символом, и находится в конце строки данного ребра, добавляем путь, помеченный вставляемым элементом, из текущего элемента в новую вершину. Если же элемент не находится в конце строки ребра, нужно сделать разветвление от текущего символа, одна дуга будет помечена второй половиной исходной строки, а другая добавляемым символом. Остался случай, когда из текущего символа есть переход по заданному. В этом случае никакие дополнительные действия производить не нужно. Для ускорения алгорима используются переходы по суффиксным ссылкам. Суффиксная ссылка из вершины v с путевой меткой ведет в вершину с путевой меткой a. Также запись строковых меток индексами начала и конца в строке S, наблюдения о том, что 3 правило вставки завершает работу данной фазы, что лист всегда остается листом, позволяют добиться ассимптотики работы алгоритма $O(M)$.

2 Исходный код

Считаем входную строку, затем по алгоритму Укконена построим суффиксное дерево для удвоенной строки. Создадим переменные *blank* – мнимая вершина, необходимая для того, чтобы не рассматривать отдельные случаи при вставке, если приходим в корень. У этой вершины будем считать, что имеются пути длины 1 по всем символам алфавита в корень, а из корня в эту переменную ведет суффиксная ссылка. *root* – корень дерева. Переменная *sp* указывает на рассматриваемую вершину и номер элемента в строке ребра, которое входит в эту вершину. После построения суффиксного дерева проходим его таким образом, чтобы в каждом узле двигаться по дуге с наименьшим первым символом. Обход заканчиваем при получении строки, длина которой равна длине исходной строки. Выводим ответ.

1.c	
int len())	Подсчет длины строки ребра
bool IsNext(int i)	Проверка, что из текущего символа есть переход по заданному
void Split(int i, TVirtualNode &sp)	Разбиение ребра и вставка элемента
void Go(TVirtualNode &v)	Переход по суффиксной ссылке
void NextLetter(TVirtualNode &v, int i)	Переход по символу i на 1 позицию

```
struct TNode {
    int l, r;
    TNode* par;
    TNode* link;
    map<char, TNode*> next;
    TNode(int l, int r, TNode* par)
        : l(l), r(r), par(par), link(NULL) {}
    int len() {
        return r - l + 1;
    }
};

struct TVirtualNode {
    TNode* n;
    int c;
    bool IsNext(int i) {
        if (c == n->r || n == root) {
            return ((n->next).count(s[i]) > 0);
        }
        else {
            return (s[c + 1] == s[i]);
        }
    }
};
```

3 Консоль

```
aacaabbcbaacaabaa
aaaacaabbcbaacaab
```

4 Тест производительности

Тест производительности представляет из себя следующее: поиск минимального разреза циклической строки с помощью построения суффиксного дерева сравнивается с наивной реализацией, время измеряется в тактах процессора.

Входные данные состоят из 10^6 символов английского алфавита.

```
С помощью суффиксного дерева: 330
Наивная реализация: 45771
```

Тестирование показало, что решение с помощью суффиксного в несколько раз эффективнее наивной реализации решения данной задачи.

5 Выводы

Выполнив пятую лабораторную работу, я научилась реализовывать алгоритм Укконена построения суффиксного дерева, находить минимальный разрез циклической строки с его помощью. Также я провела сравнение скорости работы алгоритма с наивной реализацией решения.

Список литературы

[1] Дэн Гасфилд *Строки, деревья и последовательности в алгоритмах* — Издательский дом «Невский Диалект», 2003. Перевод с английского: И. В. Романовский. — 1296 с. (ISBN5-7940-0103-8)