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. А. Почечура
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №5

Задача: Линеаризовать циклическую строку, то есть найти минимальный в лексикографическом смысле разрез циклической строки с помощью суффиксного дерева.

1 Описание

Требуется написать реализацию суффиксного дерева. Оно представляет из себя сжатый $Trie$, который содержит в себе все суффиксы строки, которая была "скормлена" дереву. Алгоритм Укконена за счёт своих ускорений (суффиксные ссылки, прыжки по счётчику, хранение на рёбрах начала и конца отрезка вместо всего куска строки, добавление строки по префиксам) даёт возможность построить суффиксное дерево за $O(n)$, где n - длина строки. Суффиксное дерево позволяет решать большое количество различных задач. Например, если в лабораторной работе №4 нам нужно было найти указанный паттерн в разных текстах, то с помощью суффиксного дерева мы можем загнать какой-то текст, и уже в нём искать любые паттерны за $O(m)$, где m - длина паттерна. В моей лабораторной работе использован для решения поставленной задачи принцип циклического буфера. К изначальной строке в конец была "приклеена" она же, и уже по этой строке мы строим суффиксное дерево. Такой ход даём нам возможность найти в суффиксном дереве изначальную строку, прочитанную циклически с любой позиции. Остаётся лишь найти минимальную лексикографическую строку, которую мы можем получить таким образом, проходя по наименьшим буквам из всех возможных в каждом узле.

2 Исходный код

Терминальный символ, который ставится в конце строки перед её добавлением в суффиксное дерево, здесь берётся как самый большой лексикографический символ (т.к. в ответ он входить не должен). Каждая нода состоит из мапы, которая отражает переход в другие узлы по всем возможным буквам, суффиксную ссылку из данного узла и пару, которая отражает отрезок строки, покрываемый ребром, ведущим в данную ноду. При добавлении нового префикса, учитывается уже пройденный путь при добавлении прошлых префиксов. Если продолжить путь по дереву мы не можем, а пройденный путь у нас итак нулевой, то просто добавим новый лист. Если же пройденный путь не нулевой, то добавим текущую букву требуемым способом (либо на ребро, если мы стоим где-то на ребре, либо создадим новый лист, если мы прошли всё ребро). Далее идёт переход по суффиксным ссылкам, или проход этого же пути, только "обрезанного" спереди на один символ, и повторение тех же действий. Помимо этого, при проходе нужно создавать суффиксные ссылки для вершин, у которых их нет. При выводе ответа идём по дереву, постоянно проходя по самой наименьшей лексокографически букве, по которой мы можем перейти. Путь наш должен составлять ровно длину данной нам строки, поэтому он может закончиться даже где-то на ребре, а не только на его конце.

lab2.cpp	
TSuffixTree:: TSuffixTree()	Деструктор дерева
void TSuffixTree::Delete(TSuffixNode* node)	Рекурсивное удаление дерева
void TSuffixTree::Build()	Построение дерева
void TSuffixTree::AddSuffixes()	Добавление каждого нового префикса строки в дерево
void TSuffixTree::Print(TSuffixNode* node, int k)	Печать ответа

```
const char TERMINAL = 'z' + 1;
const int ABSOLUTE_END = 1e7;
int endOfPrefix = -1; //
int remainingPart = 0; //
int remainingPass = 0; //
string pattern;

class TSuffixNode {
public:
    map<char, TSuffixNode*> mapData;
    TSuffixNode* SuffixLink = nullptr;
    pair<int,int> Key = {-1,-1};
};
```

```
14
15 TSuffixNode* node = nullptr; //
16 pair<int, int> keyNode; //
17 char alphaNode; //
18 TSuffixNode* suffLink = nullptr; //
19 int suffNumber = -1; //
20
21 class TSuffixTree {
22 public:
23     TSuffixTree(const string pattern) { Build(); }
24     ~TSuffixTree();
25 private:
26     void Build();
27     void AddSuffixes();
28     void Print(TSuffixNode*, int);
29     void Delete(TSuffixNode*);
30 private:
31     TSuffixNode* Root = nullptr;
32 };
```

3 Консоль

```
root@DESKTOP-5HM2HTK:~# cat <test
xabcd
root@DESKTOP-5HM2HTK:~# g++ lab5.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <test
abcdx
root@DESKTOP-5HM2HTK:~#
```

4 Тест производительности

```
root@DESKTOP-5HM2HTK:~# g++ -pedantic -Wall benchmark.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <tests/1.t
Count of lines is 84
Suffix Tree time: 2477ms
Square algorithm time: 7868ms
root@DESKTOP-5HM2HTK:~#
```

Как видно, суффиксное дерево на тестах с длинной строки ≈ 10000 работает больше чем в 3 раза быстрее, чем алгоритм за квадрат. Из-за того, что алгоритм суффиксного дерева требует сначала построение самого дерева, и лишь потом осуществляет поиск, в его сложности присутствует некоторая константа, которая, при увеличении длины строки, будет всё меньше ощущаться.

5 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», я разобрался, как работает суффиксное дерево и научился его реализовывать. Программировать алгоритм Укконена было очень интересно, немалая часть времени была потрачена для улучшения алгоритма перехода и создания суффиксных ссылок, которые очень сильно влияют на время работы всего алгоритма. Принцип циклического буфера также был интересен и полезен для решения будущих задач.

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] *Суффиксное дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Суффиксное_дерево` (дата обращения: 5.09.2022).