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. О. Дубинин
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.

Вариант поиска: Найти в заранее известном тексте поступающие на вход образцы.

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 Описание

Сперва нужно определится с терминами. Суффиксное дерево T для m-символьной строки S:

1. Ориентированное дерево, имеющее ровно m листьев, пронумерованных от 1 до m.
2. Каждая внутренняя вершина, отличная от корня, имеет не меньше двух детей.
3. Каждая дуга помечена непустой подстрокой строки S (дуговая метка).
4. Никакие две дуги, выходящие из одной вершины, не могут иметь меток, начинающихся с одинаковых символов.
5. Для каждого листа i конкатенация меток от корня составляет S[i..m].

Поскольку суффиксное дерево нельзя построить для любой строки: если существует суффикс, совпадающий с префиксом другого суффикса, то не будет выполнено условие о количестве листьев, то в конец строки добавляется терминальный символ – не принадлежащий алфавиту строки. В данной лабораторной, для наглядности, я использовал символ $.

Алгоритм Укконена заключается в следующем берутся префиксы строки, и для них строится неявное дерево, последовательным добавлением их суффиксов начиная с большего. Такой алгоритм получается слишком трудоёмким лишь на первый взгляд к нему применяются улучшения. При продолжении суффиксов возможно три случая: продлить лист, создать новую вершину, ничего не делать (такой суффикс уже есть в дереве неявно). Сперва вводятся суффиксные связи, причём при прохождении по ним происходит прыжок по счётчику: зная размер строки заранее прыгаем по вершинам. Затем происходит сокращение требуемой памяти: вместо хранения строк, хранятся указатели на конкретные символы в строке (в моей программе это реализовано с помощью итераторов). И переосмысление правил: правило один - это листы остаются листами на всегда, правило три обозначает, что такой суффикс уже добавлен и ничего делать не надо. Таким образом прохождение фазы сводится к последовательному применению правила 2.

Поиск образца по суффиксному дереву текста:

Проходим по суффиксному дереву по символам образца. При этом складываем длину пройденых ребёр – они потребуются для определения конкретного индекса образца. Если в каком-то моменте обнаружилось несовпадение – это обозначает, что текст не содержит суффикса префиксом которого является образец. То есть образец не встречается в тексте. Если же мы успешно прошли по дереву и остановились в какой-то вершине, то применям поиск Depth-first search (DFS) – поиск в глубину к

данной вершине, реализована функцией SearchLeafs, и каждый лист тогда – ровно одно вхождение пройденного образца. Тогда его позиция в тексте будет определяться длиной текста минус длина суффикса, которую мы подсчитывали при поиске.

2 Исходный код

TSuffixTree	
std::string text;	Строка по которой построено дерево
std::vector<int> Search(std::string pattern);	Поиск образца с помощью суффиксного дерева. Возвращает массив с индексами вхождений
TSuffixTree(std::string str);	Конструктор строящий суффиксное дерево.
void Build(std::string::iterator &position);	Функция запускающая фазу с буквой position
void ImprovedAlgo(std::string::iterator positionBegin, std::string::iterator positionEnd);	Выполнения "Дополнений"(extensions)(добавление суффикса positionBegin...positionEnd-1)
void NaiveAlgo(std::string::iterator positionBegin, std::string::iterator positionEnd);	Вспомогательная функция для "Дополнений"(extensions)
void SearchLeafs(TNode *node, std::vector<int> &result, int deep);	Поиск всех листов, в которые есть пути из данной вершины. используется для поиска
void RecursiveDestroy(TNode *node);	Рекурсивное удаление
TNode	
std::map<char, TNode *> to;	Пути из вершины
std::string::iterator begin, end;	Итераторы на начало и на конец подстроки текста, которой помещена грань
TNode *sufflink;	Суффиксная ссылка

3 Производительность

Реализованный поиск сравнивается со стандартным методом поиска в строке $std::string::find$. Для этого был реализована следующая программа

```
#include <string>
#include <iostream>
#include <vector>
#include <chrono>
int main(void)
{
    std::chrono::high_resolution_clock::time_point start = std::chrono::
        high_resolution_clock::now();
    std::string text;
    std::string pattern;
    std::cin >> text;
    int count = 1;
    size_t found;
    std::vector<int> res;
    while (std::cin >> pattern){
        found = text.find(pattern);
        while(found != std::string::npos){
            res.push_back(found+1);
            found = text.find(pattern, found+1);
        }
        if (!res.empty()) {
            std::cout << count << ": ";
            for (int i = 0; i < res.size()-1; ++i)
                std::cout << res[i] << ", ";
            std::cout << res[res.size()-1] << std::endl;
        }
        res.clear();
        ++count;
    }

    std::chrono::high_resolution_clock::time_point end = std::chrono::
        high_resolution_clock::now();
    std::cout << "Search: std::string::find" << std::endl;
    std::cout << "Time of working ";
    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>( end - start ).
        count();
    std::cout << " ms." << std::endl;
    return 0;
}
```

Согласно [4] в общем случае сложность find не определена, но худший случай: длина текста умноженная на индекс, с которого осуществляется поиск (второй аргумент).

Тест из 1000 символов с часто повторящимися подстроками, образцы похожи на текст:

```
art@mars:~/study/semester_3/DA/labs/lab_5/test_code/ ../main <tests/00.t >tests/00.my
art@mars:~/study/semester_3/DA/labs/lab_5/test_code/ benchmark/find <tests/00.t
>tests/00.ans
art@mars:~/study/semester_3/DA/labs/lab_5/test_code/ diff tests/00.my tests/00.ans
122,123c122,123
<Search: suff
<Time of working 22 ms.
---
>Search: std::string::find
>Time of working 23 ms.
```

Тест из 10000 символов:

```
art@mars:~/study/semester_3/DA/labs/lab_5/test_code/ ../main <tests/00.t >tests/00.my
art@mars:~/study/semester_3/DA/labs/lab_5/test_code/ benchmark/find <tests/00.t
>tests/00.ans
art@mars:~/study/semester_3/DA/labs/lab_5/test_code/ diff tests/00.my tests/00.ans
1732,1733c1732,1733
<Search: suff
<Time of working 680 ms.
---
>Search: std::string::find
>Time of working 924 ms.
```

4 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я изучил алгоритм Укконена построения суффиксного дерева за линейное время и реализовал его применение для поиска множества образцов онлайн.

Он выглядит очень практичным и быстрым на фоне, например, поиска КМП, который требует предварительной подготовки образцов. Например можно потратить значительное время и построить суффиксное дерево для огромной базы данных: базы информации о компьютерных вирусах и производить очень быстрый поиск по нему – за длину образца плюс их количество в тексте. Однако, есть более эффективная по памяти (но чуть менее по производительности) реализация такого сценария – суффиксный масив.

Конкретно в реализации моей программы, он очень напоминает алгоритм Ахо-Корасик: такая же сложность и похожая структура – бор с суффиксными ссылками. Однако, суффиксное дерево позволяет решать по истине впечатляющее количество прикладных задач: поиск в наборе строк, линеаризацию циклической строки, поиск общих подстрок и т.д.

Список литературы

[1] Дэн Гасфилд *Строки деревъя и последовательности в алгоритмах: Информатика и вычислительная биология* — Издательство « Невский диалект», 2003. – 654 с.: ил. (ISBN 5-7940-0103-8)

[2] *Построение суффиксного дерева за линейное время*
URL: `http://yury.name/internet/01ianote.pdf` (дата обращения: 18.01.2018).

[3] *Алгоритм Укконена – Викиконспекты*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Укконена` (дата обращения: 18.01.2018).

[4] *string::find - C++ Reference*
URL: `http://www.cplusplus.com/reference/string/string/find/` (дата обращения: 18.01.2018).