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: Ю. Ю. Обыденкова
Преподаватель: А. Н. Ридли
Группа: М8О-308Б-18
Дата:
Оценка:
Подпись:

Москва, 2021

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.

Вариант задания Поиск с использованием суффиксного массива. Необходимо по полученному суффиксному дереву построить суффиксный массив и искать в нём строки с помощью бинарного поиска.

Вариант алфавита Латинские строчные буквы от a до z.

1 Описание

Суффиксное дерево — структура данных, выявляющая внутреннее строение строки более глубоко, нежели Z-функция. Суффиксные деревья имеют множество применений, одно из которых — задача о точном совпадении подстрок(решаемая за линейное время). Задача эта формулируется для суффиксного дерева так — за препроцессорное время $O(m)$(где m — длина текста) необходимо подготовиться к тому, чтобы, получив неизвестную строку длины n, за время $O(n)$ либо найти вхождение этой строки в текст, либо определить, что данная строка в него не входит.

Определим, что же такое суффиксное дерево. [1]: «Суффиксным деревом для m-символьной строки S называется ориентированное дерево с корнем, имеющее ровно m листьев, пронумерованных от 1 до m. Каждая внутренняя вершина такого дерева имеет не менее двух детей, а каждая дуга помечена непустой подстрокой строки S(дуговой меткой). Никакие две дуги, выходящие из одной и той же вершины не могут иметь пометок, начинающихся с одного символа. Основной особенностью этой структуры является то, что для каждого листа i конкатенация меток дуг на пути от корня к листу i в точности составляет суффикс строки S, который начинается в позиции i.»

Существует несколько способов построения этого дерева:

- Наивное построение с использованием алгоритма Ахо-Корасик. Дерево строится за время $O(n^2)$(где n — длина текста) и занимает $O(n^2)$ памяти.
- Алгоритм Вайнера.
- Алгоритм МакКрейта.
- Алгоритм Укконена.

Из вышеперечисленных нас прежде всего интересует последний. От остальных он отличается тем, что его можно считать online алгоритмом, так как суффиксное дерево строится последовательно для всех символов строки, начиная с первого, и переход между такими этапами построения занимает $O(1)$ времени.

Перед тем, как описывать алгоритм, работающий за линейное время, проще описать его худшую версию, работающую за кубическое время, после чего, проведя ряд улучшений, уменьшить его время работы до линейного.

Говоря кратко суть алгоритма за кубическое время состоит в том, чтобы брать каждый префикс строки, начиная с префикса длины 1, и добавлять все его суффиксы, начиная с самого длинного, в дерево. Для того, чтобы улучшить время работы до $O(n^2)$ необходимо в первую очередь сделать дерево сжатым и ввести в алгоритм суффиксные ссылки. Суффиксной ссылкой для вершины, соответствующей метке $x\alpha$, называется ссылка на вершину, метка которой α. Суффиксные ссылки ведут только

из внутренних вершин во внутренние. Для последующего ускорения нужно сжать дуговые метки(теперь вместо подстрок исходной строки там хранятся пары индексов). Также нужно заметить, что для перемещения по дуге не обязательно сравнивать все её символы, достаточно только первого. Следующим шагом к ускорению является осознание того, что листься дерева всегда остаются листьями, а значит их не обязательно продолжать на каждой итерации алгоритма.Последним шагом является понимание, что если попытка продолжения очередного суффикса является неудачной по причине того, что дуга с таким стартовым символом уже идет из данной вершины, то можно сразу переходить к следующей итерации алгоритма, а стартовой позицией для этой итерации будет позиция, получающаяся из текущей спуском по дуге на один символ(тот, который мы хотели добавить).

В задании фигурирует суффиксный массив — массив, состоящий из стартовых индексов суффиксов строки, отсортированных в лексикографическом порядке. Для построения этой структуры также существует алгоритм, но мы построим её, основываясь на суффиксном дереве. Сделать это можно с помощью обхода в глубину, при котором все суффиксы обходятся в порядке лексикографического возрастания. Задача поиска подстроки в строке, с использованием суффиксного массива сводится к бинарному поиску искомой строки внутри.

2 Исходный код

SuffixTree — публичные функции	
explicit SuffixTree(std::string)	Конструктор суффиксного дерева, именно в нем выполняется алгоритм Укконена
~SuffixTree()	Рекурсивный деструктор
std::set<long long> SearchForString(const std::string&) const	Поиск вхождений подстроки в дерево
SuffixArray ConstructArray() const	Строит суффиксный массив на основе суффиксного дерева
SuffixTree — приватные функции	
void LexicographicalTraverse(SuffixNode* node, std::vector<long long>& result) const	Выполняет обход в глубину в лексикографическом порядке, записывая в result получаемый суффиксный массив
void FindOcurencies(SuffixNode* node, std::set<long long>& set) const	Находит все листья, начиная с переданного узла и помещает их номера в set
void DeleteSubtree(SuffixNode* node)	Вспомогательная рекурсивная функция для удаления поддерева
void NumerateLeaves(SuffixNode* node, long long depth)	Нумерует листья дерева
SuffixTree — внутренняя структура ArcInfo, хранящая позицию в дереве	
ArcInfo(const std::string* str, SuffixNode* link, long long arcStart, long long arcLength)	Конструктор для конкретной позиции.
bool OnArc()	Проверяет, находится ли текущая позиция на дуге дерева
bool CheckNextSymbol(char s)	Проверяет, является ли s следующим символом дуги, если позиция находится на дуге. Если позиция в узле, проверяет, есть ли из нее дуга с таким символом
ArcInfo GoDown(std::pair<long long, long long>)	Принимает дугу в виде пары чисел — начальной позиции и длины, и проходит по ней вниз из текущей позиции
AddNode(long long index, SuffixNode*& prev)	Добавляет в текующую позицию новый узел, попутно инициализируя суффиксные ссылки.
SuffixArray	

SuffixArray(std::vector<long long> newArray, std::string newData)	Приватный конструктор, вызываемый в дружеском классе SuffixTree
std::set<long long> SearchForString(std::string sub) const	Поиск подстроки в тексте с использованием суффиксного массива.

3 Консоль

```
julia@julia21novo:~/CLionProjects/da_05/solution$ make
g++ -std=c++11 -O2 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic -o solution main.cpp SuffixArray.cpp SuffixTree.cpp
julia@julia21novo:~/CLionProjects/da_05/solution$ ./solution
abacaba
a
1: 1,3,5,7
aba
2: 1,5
caba
3: 4
ba
4: 2,6
```

4 Тест производительности

Тест производительности представляет из себя следующее: в тексте длины 100000 производится поиск сначала 1000 строк длины 100, гарантировано в него входящих, а потом производится поиск такого же количества строк, входящих в текст с малой вероятностью. Сгенерируем текст для примера и замерим время выполнения программ.

```
#include <iostream>
#include <string>
#include <random>

using namespace std;

default_random_engine rng;

uint64_t get_number(uint64_t min = 0,uint64_t max = numeric_limits<unsigned long long
    >::max()) {
    uniform_int_distribution<unsigned long long> dist_ab(min, max);
    return dist_ab(rng);
}

string get_string(int size) {
    size_t string_size = get_number(size,size);
    string string;
    string.resize(string_size);
    for (size_t i = 0; i < string_size; ++i) {
        string[i] = 'a' + get_number(0,25);
    }
    return string;
}


int main(int argc, char** argv) {

    string text = get_string(100000);
    std::cout << text << '\n';
    for (int i = 0; i < 1000; ++i) {
        std::cout << text.substr(get_number(0,99000), 150) << "\n";
    }
    for (int j = 0; j < 1000; ++j) {
        std::cout << get_string(150) << "\n";
    }
    return 0;

}
```

Запустим тесты и замерим время работы.

```
julia@julia21novo:~/CLionProjects/da_05/cmake-build-debug$ ./test >test_file
julia@julia21novo:~/CLionProjects/da_05/cmake-build-debug$ ./bench <test_file
>/dev/null
Suffix tree construction 492 ms
Suffix array construction 70 ms
Suffix array construction 80 ms
julia@julia21novo:~/CLionProjects/da_05/cmake-build-debug$ ./naive <test_file
>/dev/null
Naive search 3376 ms
julia@julia21novo:~/CLionProjects/da_05/cmake-build-debug$
```

Видно, что поиск с использованием суффиксного массива значительно быстрее, нежели наивный.

5 Выводы

Суффиксные структуры(деревья, массивы, автоматы) позволяют тщательно исследовать внутреннее строение строки. Благодаря этому, они являются крайне мощными инструментами. Каждая из этих структур имеет свои особенности и недостатки. Суффиксное дерево, например, тратит довольно много памяти, но поиск в нем происходит быстрее, чем в суффиксном массиве. В результате выполнения этой работы я познакомилась с этими структурами данных и улучшила свои навыки программирования на C++.

Дерево корректно строится за O(n), доказательство этого лежит в доказательстве корректности алгоритма Укконена.

Т.к. переход по прямой ссылке работает за O(1), а их не более N , то достаточно только рассмотреть переход вниз по рёбрам. Для этого рассмотрим длину смещения по ребру, по которому мы поднялись к предку. Пусть она равна L. Тогда при каждом прохождении во рёбрам вниз эта длина уменьшается хотя бы на 1. А увеличивается она только в основоной процедуре при проходе по ребру по символу C, причём не более чем на 1 за раз. А таких проходов N . Конечное значение смещения лежит в диапазоне [0, N], количество удлинений N , то и суммарное количество сокращений может быть не более 2N . Значит, суммарное количество продвижений вниз при переходе по суффикной ссылке O(3N). В итоге, количество каждого вида действий O(N), а значит и суммарное количество всех действий O(N).

Первая, и возможно самая распространённая, практическая задача, связанная с суффиксными деревьями, это нахождение включения одной строки в текст. Для этой задачи есть и другие алгоритмы, но если мы её немного модифицируем, например, нужно будет искать много раз, но мы не может обработать все запросы заранее, как

в алгоритме Ахо-корасика, то тогда просто проходя по дереву по символам из поискового запроса, мы либо попытаемся выйти из дерева, тогда ответ отрицательный, либо, если запрос закончился, ответ положительный.

Список литературы

[1] Дэн Гасфилд. *Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология* — Издательство «Невский Диалект», 2003. Перевод с английского: И. В. Романовский. — 654 с. (ISBN 5-7940-0103-8(рус.))