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: Д. С. Пивницкий
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата: 01.01.2021
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 задания.

Алфавит строк: строчные буквы латинского алфавита (т.е. от a до z).

Формат входных данных: Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

Формат выходных данных: Для каждого образца, найденного в тексте, нужно распечатать строку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания.

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

1 Описание

Требуется реализовать алгоритм Укконена для построения суффиксного дерева, из суффиксного дерева построить суффиксный массив и написать эффективный алгоритм поиска паттерна в тексте при помощи суффиксного массива. Суффиксный массив получается при обходе в глубину дерева в лексикографическом порядке. Поиск паттерна в тексте при помощи суффиксного массива основывается на поиске левой границы, где паттерн $\geq$ суффикса в массиве и правой границы, где паттерн $<$ суффикса в массиве, т.к. суффиксный массив упорядочен лексикографически, то все суффиксы лежащие между левой и правой границей являются вхождениями, вхождение границ определяется реализацией.

2 Исходный код

Основная функция считывает текст, строит по нему суффиксное дерево, по суффиксному дереву строит суффиксный массив, считывает все паттерны и находит вхождения в текст, для каждого паттерна выводит упорядоченные индексы начала каждого вхождения паттерна в текст.

Код: main.cpp

```
#include <iostream>
#include "st.hpp"

int main()
{
std::string text, pattern;
std::cin >> text;
TSuffTree tree(text + "$");
TSuffArr sa(&tree);
size_t test_number = 1;
while(std::cin >> pattern)
{
std::vector<int> result = sa.Find(pattern);
if(!result.empty())
{
std::cout << test_number << ": ";
for(size_t i = 0; i < result.size(); ++i)
{
std::cout << result[i] + 1;
if(i < result.size() - 1) { std::cout << ", "; }
}
std::cout << "\n";
}
++test_number;
}
return 0;
}
```

Используются конструкторы: TSuffTree::TSuffTree производит инициализацию, void TSuffTree::Destroy для удаления вершины дерева и всех его потомков, void TSuffTree::Create для вставки в дерево суффикса начинающегося с pos. TSuffArr::TSuffArr это конструктор построения суффиксного массива из суффиксного дерева tree.

Так же реализованы функции поиска левого и правого индекса суффиксного массива в котором суффикс содержит паттерн.

Код: st.cpp

```
#include "st.hpp"

TSuffTree::TSuffTree(std::string const& string) : text(string), root(new TNode(text.
    end(), text.end())), remainder(0)
```

```
4  {
5  activ_edge = text.begin();
6  activ_node = root;
7  root->suff_link = root;
8  curr_suff_link = root;
9  activ_length = 0;
10 for(std::string::iterator suff = text.begin(); suff != text.end(); ++suff) { Create(
       suff); }
11 }
12
13 TNode::TNode(std::string::iterator start, std::string::iterator end) : start(start),
       end(end), suff_link(nullptr) {}
14
15 void TSuffTree::Destroy(TNode* node)
16 {
17 for(std::map<char, TNode*>::iterator it = node->v.begin(); it != node->v.end(); ++it)
       { Destroy(it->second); }
18 delete node;
19 }
20
21 TSuffTree::~TSuffTree() { Destroy(root); }
22
23 void TSuffTree::Create(std::string::iterator pos)
24 {
25 curr_suff_link = root;
26 ++remainder;
27 while(remainder)
28 {
29 if(!activ_length) { activ_edge = pos; }
30 std::map<char, TNode*>::iterator v = activ_node->v.find(*activ_edge);
31 TNode* next;
32 if(v == activ_node->v.end())
33 {
34 TNode* leaf = new TNode(pos, text.end());
35 activ_node->v[*activ_edge] = leaf;
36 SuffLinkAdd(activ_node);
37 }
38 else
39 {
40 next = v->second;
41 if(GoDown(pos, next)) { continue; }
42 if(*(next->start + activ_length) == *pos)
43 {
44 ++activ_length;
45 SuffLinkAdd(activ_node);
46 break;
47 }
48 TNode* split = new TNode(next->start, next->start + activ_length);
49 TNode* leaf = new TNode(pos, text.end());
```

```
50
51 activ_node->v[*activ_edge] = split;
52 split->v[*pos] = leaf;
53 next->start += activ_length;
54 split->v[*next->start] = next;
55 SuffLinkAdd(split);
56 }
57 --remainder;
58 if(activ_node == root && activ_length)
59 {
60 --activ_length;
61 activ_edge = pos - remainder + 1;
62 }
63 else
64 {
65 activ_node = (activ_node->suff_link) ? activ_node->suff_link : root;
66 }
67 }
68 }
69
70 bool TSuffTree::GoDown(std::string::iterator pos, TNode* node)
71 {
72 int length = 0;
73 if(pos + 1 < node->end) { length = pos + 1 - node->start; }
74 else { length = node->end - node->start; }
75 if(activ_length >= length)
76 {
77 activ_edge += length;
78 activ_length -= length;
79 activ_node = node;
80 return true;
81 }
82 return false;
83 }
84
85 void TSuffTree::SuffLinkAdd(TNode* node)
86 {
87 if(curr_suff_link != root) { curr_suff_link->suff_link = node; }
88 curr_suff_link = node;
89 }
90
91 void TSuffTree::DFS(TNode* node, std::vector<int>& result, int const& deep)
92 {
93 if(node->v.empty())
94 {
95 result.push_back(text.size() - deep);
96 return;
97 }
98 for(std::map<char, TNode*>::iterator it = node->v.begin(); it != node->v.end(); ++it)
```

```
{
int tmp = deep;
tmp += it->second->end - it->second->start;
DFS(it->second, result, tmp);
}
}

TSuffArr::TSuffArr(TSuffTree* tree) : text(tree->text), arr()
{
tree->DFS(tree->root, arr, 0);
}


int TSuffArr::FindLeft(std::string const& pattern)
{
int left = 0;
int right = text.size() - 1;
int length = pattern.size();
while(left <= right)
{
int mid = (left + right) / 2;
std::string tmp = text.substr(arr[mid], length);
if(pattern > tmp) { left = mid + 1; }
else { right = mid - 1; }
}
return left;
}

int TSuffArr::FindRight(std::string const& pattern)
{
int left = 0;
int right = text.size() - 1;
int length = pattern.size();
while(left <= right)
{
int mid = (left + right) / 2;
std::string tmp = text.substr(arr[mid], length);
if(pattern >= tmp) { left = mid + 1; }
else { right = mid - 1; }
}
return left;
}

std::vector<int> TSuffArr::Find(std::string const& pattern)
{
int left = FindLeft(pattern);
int right = FindRight(pattern);
std::vector<int> result;
for(int i = left; i < right; ++i) { result.push_back(arr[i]); }
```

```
148 std::sort(result.begin(), result.end());
149 return result;
150 }
```

3 Консоль

```
(py37) ~ /DA_labs/lab5$ make
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
(py37) ~ /DA_labs/lab5$ cat test.txt
abcgfhdeghheababctcehjtceghjtcehjdadeabcghheghhhheghhdeabcacbabchhjdetcehj
abc
de
ghhe
tcehj
(py37) ~ /DA_labs/lab5$ ./solution<test.txt
1: 1,15,38,56,62
2: 7,36,54,68
3: 9,41
4: 18,29,70
```

4 Тест производительности

Тест состоит из 400 тыс. поисков паттерна в тексте для моего поиска с использованием суффиксного массива и для стандартного метода строк find, который модифицирован для поиска нескольких образцов в тексте. Паттерны повторяются по 4 штуки - p1: «abc», p2: «de», p3: «ghhe», p4: «tcehj», p5: «abc» и т.д.

Текст: «abcgfhdeghheababctcehjtceghjtcehjdadeabcghheghhhheghhdeabcacbabchhjdetcehj»

Моя реализация:

```
(py37) ~ /DA_labs/lab5$ make
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
(py37) ~ /DA_labs/lab5$ make bench
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror benchmark.cpp -o benchmark
(py37) ~ /DA_labs/lab5$ ./benchmark
Time for create suffix tree and suffix array: 0.0007106 seconds
Time for my find: 6.08816 seconds
Time for standart find: 1.66517 seconds
```

Как можно увидеть, время работы на построение суффиксново дерева и массива крайне мало, а поиск в несколько раз медленнее, но если например нам важно сэкономить память, но не так критично время, то поиск с использованием суффиксного массива вполне подходит.

5 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», я познакомился с алгоритмом Укконена построения суффиксного дерева, построения суффиксного массива из суффиксного дерева, поиска подстроки в строке с использованием суффиксного массива.

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] *Поразрядная сортировка — Вики университета ITMO.* URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 01.10.2020).