МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное

образовательное учреждение

высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №11

**по дисциплине**

**“ Алгоритмы построение и анализ”**

Выполнил: студент гр. ФИб-3302-51-00 ё. \_\_\_\_\_\_\_\_\_\_\_

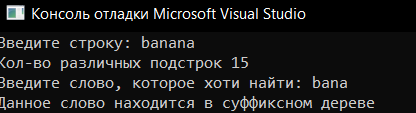
Проверил: доцент кафедры ПМиИ Разова Е. В. \_\_\_\_\_\_\_\_\_\_\_

Киров 2020

**Задание 1.** Поиск подстроки в s   
*Идея: Заметим, что строка u является подстрокой строки s тогда и только тогда, когда u можно прочитаться из корня суффиксного дерева (т.к. u = s[i..j] для некоторых i и j и поэтому суффикс s[i..n-1] начинается с u).*

**Задание 2.** Количество различных подстрок в строке s

*Идея: Теми же рассуждениями можно установить, что каждая подстрока s соответствует некоторой позиции на метке какого-то ребра суффиксного дерева. Значит число различных подстрок – это число таких позиций и оно равно сумме len[v] по всем вершинам v кроме корня.*



#include <iostream>

#include <clocale>

#include <cmath>

#include <algorithm>

#include <vector>

#include <string>

using namespace std;

const int alph = 256;

const int INF = 1e9;

const int MPOW = 16;

const int N = 1 << MPOW - 1;

const int N2 = N << 1;

struct sg\_tree

{

int arr[N2];

void build(vector<int> x, int n)

{

fill(arr, arr + N2, INF);

for (int i = 0; i < n; i++)

arr[i + N] = x[i];

for (int i = N - 1; i > 0; i--)

arr[i] = min(arr[i << 1], arr[(i << 1) + 1]);

}

int get\_min(int c, int cl, int cr, int l, int r)

{

if (l == cl && r == cr)

return arr[c];

if (l > r)

return INF;

int cm = cl + cr >> 1;

return min(get\_min(c << 1, cl, cm, l, min(r, cm)), get\_min((c << 1) + 1, cm + 1, cr, max(l, cm + 1), r));

}

int get\_min(int l, int r)

{

return get\_min(1, 0, N - 1, l, r);

}

};

pair<vector<int>, vector<int>> compute(string& s)

{

int n = s.size();

int maxn = n + alph;

vector<int> p(n), c(n), cnt(maxn, 0);

for (int i = 0; i < n; i++)

cnt[s[i]]++;

for (int i = 1; i < maxn; i++)

cnt[i] += cnt[i - 1];

for (int i = 0; i < n; i++)

p[--cnt[s[i]]] = i;

int cl = 0;

c[p[0]] = cl;

for (int i = 1; i < n; i++)

{

if (s[p[i]] != s[p[i - 1]])cl++;

c[p[i]] = cl;

}

vector<int> lcp(n, 0);

for (int i = 1; i < n; i++)

lcp[i] = c[p[i]] == c[p[i - 1]];

vector<int> pn(n), cn(n), lcpn(n);

vector<int> rpos(n), lpos(n);

sg\_tree rmq;

int k = 1;

while (k < n)

{

fill(begin(cnt), end(cnt), 0);

for (int i = 0; i < n; i++)

rpos[c[p[i]]] = i;

for (int i = n - 1; i >= 0; i--)

lpos[c[p[i]]] = i;

for (int i = 0; i < n; i++)

{

pn[i] = p[i] - k;

if (pn[i] < 0)pn[i] += n;

}

for (int i = 0; i < n; i++)

cnt[c[i]]++;

for (int i = 1; i < maxn; i++)

cnt[i] += cnt[i - 1];

for (int i = n - 1; i >= 0; i--)

p[--cnt[c[pn[i]]]] = pn[i];

cl = 0;

cn[p[0]] = 0;

for (int i = 1; i < n; i++)

{

int m1 = (p[i] + k) % n, m2 = (p[i - 1] + k) % n;

if (c[p[i]] != c[p[i - 1]] || c[m1] != c[m2])cl++;

cn[p[i]] = cl;

}

rmq.build(lcp, n);

for (int i = 1; i < n; i++)

{

int a = p[i], b = p[i - 1];

if (c[a] != c[b])

lcpn[i] = lcp[lpos[c[a]]];

else

{

int aa = (a + k) % n, bb = (b + k) % n;

if (c[aa] == c[bb])

lcpn[i] = k << 1;

else

lcpn[i] = k + rmq.get\_min(lpos[c[bb]] + 1, rpos[c[aa]]);

}

lcpn[i] = min(n, lcpn[i]);

}

copy(begin(cn), end(cn), begin(c));

copy(begin(lcpn), end(lcpn), begin(lcp));

k <<= 1;

}

return { p,lcp };

}

struct suffix\_tree

{

struct edge

{

int from;

int to;

int next\_vert;

int suffix\_here;

};

struct vertex

{

vector<edge> go;

};

string str;

vector<vertex> data;

static bool comp(const edge& b, const char& a)

{

return 1;

}

void build(string& s)

{

// Получим суффиксный массив строки, а также массив lcp

pair<vector<int>, vector<int>> info = compute(s);

vector<int> p = info.first, lcp = info.second;

int n = s.size();

str = s;

// Будем хранить здесь стек рёбер, из которых мы ещё не вышли

vector<int> p\_vert;

vector<int> p\_edge;

vector<int> p\_dist;

// Добавим первую строку в дерево

vertex v;

edge e;

e.from = p[0];

e.to = n;

e.next\_vert = -1;

e.suffix\_here = p[0];

v.go.push\_back(e);

data.push\_back(v);

p\_vert.push\_back(0);

p\_edge.push\_back(0);

p\_dist.push\_back(0);

for (int i = 1; i < n; i++)

{

int c\_lcp = lcp[i];

// Поднимаемся до lcp.

while (p\_dist.back() > c\_lcp)

{

// Найдём минимальный суффикс, покрывающий ребро для ответа на задачу.

edge& E = data[p\_vert.back()].go[p\_edge.back()];

if (E.next\_vert + 1)

{

int m = data[E.next\_vert].go.size();

for (int j = 0; j < m; j++)

E.suffix\_here = min(E.suffix\_here, data[E.next\_vert].go[j].suffix\_here);

}

// Удалим ребро из стека. Больше мы в него не вернёмся

p\_vert.pop\_back();

p\_edge.pop\_back();

p\_dist.pop\_back();

}

vertex v;

edge e;

int c\_v = p\_vert.back();

int c\_e = p\_edge.back();

int At = data[c\_v].go[c\_e].from + c\_lcp - p\_dist.back(); // Индекс для разделения ребра

p\_dist.push\_back(c\_lcp);

// Ребро, которое надо добавить

e.next\_vert = -1;

e.suffix\_here = p[i];

e.from = p[i] + c\_lcp;

e.to = n;

// Либо добавляем ребро к вершине, либо разделяем текущее

if (At == data[c\_v].go[c\_e].from)

{

data[c\_v].go.push\_back(e);

p\_vert.push\_back(c\_v);

p\_edge.push\_back(data[c\_v].go.size() - 1);

}

else

{

v.go.push\_back(data[c\_v].go[c\_e]);

v.go.back().from = At;

v.go.push\_back(e);

data.push\_back(v);

data[c\_v].go[c\_e].next\_vert = data.size() - 1;

data[c\_v].go[c\_e].to = At;

p\_vert.push\_back(data.size() - 1);

p\_edge.push\_back(1);

}

}

while (!p\_dist.empty())

{

// Найдём минимальный суффикс, покрывающий ребро для ответа на задачу.

edge& E = data[p\_vert.back()].go[p\_edge.back()];

if (E.next\_vert + 1)

{

int m = data[E.next\_vert].go.size();

for (int j = 0; j < m; j++)

E.suffix\_here = min(E.suffix\_here, data[E.next\_vert].go[j].suffix\_here);

}

// Удалим ребро из стека. Больше мы в него не вернёмся

p\_vert.pop\_back();

p\_edge.pop\_back();

p\_dist.pop\_back();

}

}

int search\_str(string& s)

{

int n = s.size();

int cur\_v = 0;

int cur\_e;

char t;

for (int i = 0; i < n;)

{

if (cur\_v == -1)break;

t = s[i];

int cur\_e;

for (cur\_e = 0; cur\_e < data[cur\_v].go.size(); cur\_e++)

if (str[data[cur\_v].go[cur\_e].from] >= t)break;

for (int j = data[cur\_v].go[cur\_e].from; i < n && j < data[cur\_v].go[cur\_e].to; j++, i++)

if (str[j] != s[i])

i = n + 1;

if (i == n)

return data[cur\_v].go[cur\_e].suffix\_here;

cur\_v = data[cur\_v].go[cur\_e].next\_vert;

}

return -1;

}

int print(int x)

{

if (x == -1)return 0;

int ans = 0;

for (int i = 0; i < data[x].go.size(); i++)

ans += data[x].go[i].to - data[x].go[i].from + print(data[x].go[i].next\_vert) - (data[x].go[i].to == str.size());

return ans;

}

};

int main()

{

setlocale(0, "");

cout << "Введите строку: ";

string a, str2;

cin >> a;

a += '$';

suffix\_tree sf;

sf.build(a);

cout << "Кол-во различных подстрок "<< sf.print(0) << endl;

cout << "Введите слово, которое хоти найти: ";

cin >> str2;

if (sf.search\_str(str2) == -1)

cout << "Слово не находится в суффиксном дереве";

else

cout << "Данное слово находится в суффиксном дереве";

}