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 №10

**по дисциплине**

**“ Алгоритмы построение и анализ”**

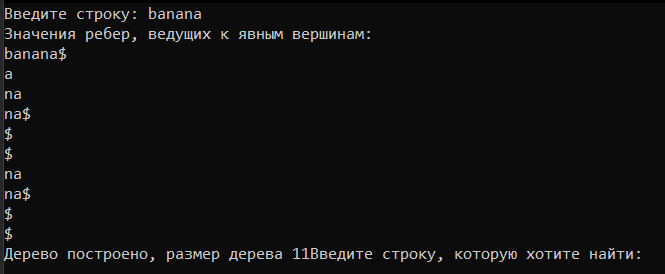
Выполнил: студент гр. ФИб-3302-51-00 ё. \_\_\_\_\_\_\_\_\_\_\_

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

Киров 2020

**Задание 1.**

- написать программу построения суффиксного дерева для заданной пользователем строки.



#include <functional>

#include <iostream>

#include <vector>

using namespace std;

struct Node

{

string sub = ""; // подстрока входной строки

vector<int> ch; // вектор детей вершины

Node()

{

// empty

}

Node(const string& sub, initializer\_list<int> children) : sub(sub)

{

ch.insert(ch.end(), children);

}

};

struct SuffixTree

{

vector<Node> nodes;

SuffixTree(const string& str)

{

nodes.push\_back(Node{});

for (size\_t i = 0; i < str.length(); i++)

addSuffix(str.substr(i));

}

void BuildTree()

{

cout << "Значения ребер, ведущих к явным вершинам: ";

function<void(int, const string&)> f;

f = [&](int n, const string& pre)

{

auto children = nodes[n].ch;

if (children.size() == 0)

{

cout << nodes[n].sub << '\n';

return;

}

cout << nodes[n].sub << '\n';

auto it = begin(children);

if (it != end(children)) do

{

if (next(it) == end(children)) break;

cout << pre;

f(\*it, pre + "");

it = next(it);

} while (true);

cout << pre;

f(children[children.size() - 1], pre + " ");

};

f(0, "");

cout << "Дерево построено, размер дерева ";

cout << (int)nodes.size();

}

private:

void addSuffix(const std::string& suf)

{

int n = 0;

size\_t i = 0;

while (i < suf.length())

{

char b = suf[i];

int x2 = 0;

int n2;

while (true)

{

auto children = nodes[n].ch;

if (x2 == children.size())

{

// нет подходящего ребенка остаток суффикса становится новой вершиной

n2 = nodes.size();

nodes.push\_back(Node(suf.substr(i), {}));

nodes[n].ch.push\_back(n2);

return;

}

n2 = children[x2];

if (nodes[n2].sub[0] == b)

{

break;

}

x2++;

}

// найдем префикс оставшегося суффикса общего с дочерним

auto sub2 = nodes[n2].sub;

size\_t j = 0;

while (j < sub2.size())

{

if (suf[i + j] != sub2[j])

{

auto n3 = n2;

// новый узел для общей части

n2 = nodes.size();

nodes.push\_back(Node(sub2.substr(0, j), { n3 }));

nodes[n3].sub = sub2.substr(j);

nodes[n].ch[x2] = n2;

break; // продолжаем дальше по дереву

}

j++;

}

i += j;

n = n2;

}

}

};

int main()

{

setlocale(0, "");

cout << "Введите строку: ";

string str;

cin >> str;

str += "$";

SuffixTree(str).BuildTree();

cout << "Введите строку, которую хотите найти: ";

string str2;

cin >> str2;

str2 += "$";

}