**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №5**

**по дисциплине «Построение и анализ алгоритмов» Тема: Алгоритм Ахо-Корасик**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Ягудин Д.Р. |
| Преподаватель |  | Фирсов М.А. |

# 

# Задание 1.

Разработайте программу, решающую задачу точного поиска набора образцов.

**Вход:**

Первая строка содержит текст (*T*,1≤∣*T*∣≤100000 ).

Вторая — число *n* (1≤*n*≤3000), каждая следующая из *n* строк содержит шаблон из набора *P*={*p*1,…,*pn*}1≤∣*pi*∣≤75

Все строки содержат символы из алфавита {*A*,*C*,*G*,*T*,*N*}

**Выход:**

Все вхождения образцов из *P* в *T*.

Каждое вхождение образца в текст представить в виде двух чисел - *i p* Где *i* - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером *p*

(нумерация образцов начинается с 1).

Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

# Sample Input:

NTAG 3 TAGT TAG T

# Sample Output:

2 2

2 3

# Задание 2.

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с *джокером*.

В шаблоне встречается специальный символ, именуемый джокером (wild card),

который "совпадает" с любым символом. По заданному содержащему шаблоны образцу *P* необходимо найти все вхождения Р в текст Т.

Например, образец а*b*??с? с джокером ? встречается дважды в тексте

*xabvccbababcax*.

Символ джокер не входит в алфавит, символы которого используются в *T*. Каждый джокер соответствует одному символу, а не подстроке неопределённой длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы.

Все строки содержат символы из алфавита {*A*,*C*,*G*,*T*,*N*}

**Вход:**

Текст (*T*,1≤∣*T*∣≤100000) Шаблон (*P*,1≤∣*P*∣≤40) Символ джокера **Выход:**

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания. Sample Input:

ACTANCA

A$$A$

$

# Sample Output:

1

# Основные теоретические положения.

Пусть дан набор строк в алфавите размера k суммарной длины m. Алгоритм Ахо-Корасик строит для этого набора строк структуру данных "бор", а затем по этому бору строит автомат, всё за O(m) времени и O(mk) памяти. Полученный автомат уже может использоваться в различных задачах, простейшая из которых — это нахождение всех вхождений каждой строки из данного набора в некоторый текст за линейное время.

*Сложности алгоритма* по операциям O(nA + T + k), где n – общая длина всех слов в словаре, A – размер алфавита, T – длина текста, в котором проводится поиск, k – общая длина всех совпадений

# Выполнение работы.

Для представления вершины бора был реализован класс *Node*. Данный класс содержит в себе следующие поля:

* *name*, которое хранит имя узла;
* *children*, представляющее из себя вектор указателей на узлы- потомки;
* parent являющееся указателем на узел-родитель;
* *suffixLink\_* – суффиксная ссылка;
* *endLink\_* – конечная ссылка;
* *terminal* – вектор номеров образцов, соответствующих терминальной вершине.

В классе *Node* реализованы следующие методы:

* *getName*, возвращающий имя данного узла;
* *getChildren*, возвращающий вектор *children*;
* *getChild*, возвращающий конкретный узел-потомок;
* *addChild()* для добавления узлов-потомков;
* *deleteChildren()* для удаления узлов-потомков;
* changeTerminal() для изменения терминальной вершины;
* *getTerminal() -* возвращает значение терминала;
* getSuffixLink (), возвращающий *suffixLink\_*;
* *getEndLink ()*, возвращающий *endLink\_*.

Для представления бора был реализован класс *Machine*, в котором содержатся следующие поля:

* *root* – ссылка на корень бора;
* *patterns* – вектор строк-образцов; и реализованы следующие методы:
* *AhoCorasick()*, реализующий алгоритм Ахо-Корасик;
* *addSuffixLinks()* для добавления суффиксных ссылок каждой вершине в боре.

Разработанный программный код см. в приложении А.

# Тестирование.

*Задание1.*

# Входные данные:

NTAG 3 TAGT TAG T

# Выходные данные:

2 2

2 3

# Входные данные:

ACGT 3 CGTA GT

A

# Выходные данные:

1 3

3 2

# Входные данные:

GGTA 3 GTA AG

T

# Выходные данные:

2 1

3 3

*Задание2.*

1. Входные данные:

ACTANCA

A$$A$

$

Выходные данные:

1

1. Входные данные:

CGTTTTNCGAS

CG\*\*

\*

Выходные данные:

1

8

# Входные данные:

TNAAAGCAAGAAG

AA& &

# Выходные данные:

3

4

8

11

# Выводы.

В ходе работы был изучен алгоритм Ахо-Корасик поиска набора образцов в строке, применён в решении поставленной задачи на языке программирования C++. Реализовано тестирование программы.

Задание 1:

**ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: main.cpp

#include "lb5\_1.hpp"

int main () {

std::string text; std::cin >> text; int n;

std::cin >> n; std::vector<std::string> patterns; std::string pattern;

for (int i = 0; i < n; i++) { std::cin >> pattern; patterns.push\_back(pattern);

}

Machine machine(patterns);

auto result = machine.AhoCorasick(text); for (auto &obj : result) {

std::cout << obj.first << ' ' << obj.second << '\n';

}

return 0;

}

Название файла: lb5\_1.cpp

#include "lb5\_1.hpp"

Название файла: lb5\_1.hpp

#include <iostream> #include <string> #include <vector> #include <queue> #include <algorithm>

class State { public:

State (char name, State \*parent) : name(name), parent(parent) { this->suffixLink = nullptr;

this->endLink = nullptr; this->terminal = 0;

}

char getName (){ return this->name;

}

std::vector<State \*> getChildren (){ return this->children;

}

State \*getChild (char newChild){

for (auto &node : children) {

if (node->getName() == newChild) { return node;

}

}

return nullptr;

}

void addChild (State \*descendent) {

this->children.push\_back(descendent);

}

void deleteChildren () {

for (auto &node : this->children) { node->deleteChildren();

delete node;

}

}

void changeTerminal (int numOfString) { this->terminal = numOfString;

}

int getTerminal (){

return this->terminal;

}

void addSuffixLink () {

if (this->getName() == ' ') { this->suffixLink = this;

} else if (this->parent->getName() == ' ') { this->suffixLink = this->parent;

} else {

auto curState = this->parent->suffixLink;

auto nextState = curState->getChild(this->getName()); while (nextState == nullptr) {

if (curState->getName() == ' ') { nextState = curState;

break;

}

curState = curState->suffixLink;

nextState = curState->getChild(this->getName());

}

this->suffixLink = nextState; auto temp = nextState;

while (temp->getTerminal() == 0) { if (temp->getName() == ' ') {

break;

}

temp = temp->getSuffixLink();

}

if (temp->getTerminal() > 0) { this->endLink = temp;

}

}

}

State \*getSuffixLink (){ return this->suffixLink;

}

State \*getEndLink (){ return this->endLink;

}

private:

char name;

std::vector<State \*> children; State \*parent;

State \*suffixLink; State \*endLink; int terminal;

};

class Machine { public:

explicit Machine (std::vector<std::string> &patterns) { this->root = new State(' ', nullptr);

this->patterns = patterns; this->addPatterns(patterns); this->addSuffixLinks();

}

~Machine () {

root->deleteChildren(); delete root;

}

&text) {

std::vector<std::pair<int, int>> AhoCorasick (std::string

auto curState = root; std::vector<std::pair<int, int>> result;

for (int index = 0; index < text.size(); index++) { auto nextState = curState->getChild(text[index]); while (nextState == nullptr) {

if (curState == this->root) { nextState = curState; break;

}

curState = curState->getSuffixLink(); nextState = curState->getChild(text[index]);

}

if (nextState->getTerminal() > 0) {

int strIndex = index - this->patterns[nextState-

>getTerminal() - 1].size() + 2;

result.emplace\_back(strIndex, nextState-

>getTerminal());

}

if (nextState->getEndLink() != nullptr) { auto temp = nextState->getEndLink(); while (temp != nullptr) {

result.emplace\_back(index - this-

>patterns[temp->getTerminal() - 1].size() + 2, temp->getTerminal());

temp = temp->getEndLink();

}

}

curState = nextState;

}

std::sort(result.begin(), result.end(), [](std::pair<int,

int> entry1, std::pair<int, int> entry2){

if (entry1.first != entry2.first) { return entry1.first < entry2.first;

} else {

return entry1.second < entry2.second;

}

});

return result;

}

protected:

void addPatterns (std::vector<std::string> &newPatterns) { int numOfString = 1;

for (auto &str : newPatterns) { State \*curState = root;

for (int i = 0; i < str.size(); i++) {

auto nextState = curState->getChild(str[i]); if (nextState == nullptr) {

nextState = new State(str[i], curState); curState->addChild(nextState);

}

if (i == str.size() - 1) {

nextState->changeTerminal(numOfString);

} else {

curState = nextState;

}

}

numOfString++;

}

}

void addSuffixLinks () { std::queue<State \*> queue; State \*curState = root; queue.push(root);

while (!queue.empty()) { curState->addSuffixLink(); queue.pop();

for (auto &node : curState->getChildren()) { queue.push(node);

}

}

private:

}

curState = queue.front();

State \*root; std::vector<std::string> patterns;

};

Задание 2:

Название файла: main.cpp

#include "lb5\_2.hpp"

int main () {

std::string text; std::cin >> text; std::string pattern; std::cin >> pattern; char joker;

std::cin >> joker; std::vector<std::pair<std::string, int>> patterns; std::string subPattern;

int endJokers;

for (int i = 0; i < pattern.size(); i++) { if (pattern[i] == joker) {

if (!subPattern.empty()) patterns.emplace\_back(subPattern, i);

subPattern = ""; endJokers++;

} else {

subPattern += pattern[i]; endJokers = 0;

}

}

if (!subPattern.empty()) { patterns.emplace\_back(subPattern, pattern.size());

}

Machine machine(patterns, endJokers); auto result = machine.AhoCorasick(text); for (auto &obj : result) {

std::cout << obj << '\n';

}

return 0;

}

Название файла: lb5\_2.cpp

#include "lb5\_2.hpp"

Название файла: lb5\_2.hpp

#include <iostream> #include <string> #include <vector> #include <queue>

class State { public:

State (char name, State \*parent) { this->name = name;

this->parent = parent; this->suffixLink = nullptr; this->endLink = nullptr;

}

char getName (){ return this->name;

}

std::vector<State \*> getChildren (){ return this->children;

}

State \*getChild (char newName){ for (auto &node : children) {

if (node->getName() == newName) { return node;

}

}

return nullptr;

}

void addChild (State \*descendent) {

this->children.push\_back(descendent);

}

void deleteChildren () {

for (auto &node : this->children) { node->deleteChildren();

delete node;

}

}

void changeTerminal (int numOfString) { this->terminals.push\_back(numOfString);

}

std::vector<int> getTerminal () { return this->terminals;

}

void addSuffixLink () {

if (this->getName() == ' ') { this->suffixLink = this;

} else if (this->parent->getName() == ' ') { this->suffixLink = this->parent;

} else {

auto curState = this->parent->suffixLink;

auto nextState = curState->getChild(this->getName()); while (nextState == nullptr) {

if (curState->getName() == ' ') { nextState = curState;

break;

}

curState = curState->suffixLink;

nextState = curState->getChild(this->getName());

}

this->suffixLink = nextState; auto temp = nextState;

while (temp->getTerminal().empty()) {

if (temp->getName() == ' ') { break;

}

temp = temp->getSuffixLink();

}

if (!temp->getTerminal().empty()) { this->endLink = temp;

}

}

}

State \*getSuffixLink () { return this->suffixLink;

}

State \*getEndLink () { return this->endLink;

}

private:

char name;

std::vector<State \*> children; State \*parent;

State \*suffixLink; State \*endLink;

std::vector<int> terminals;

};

class Machine { public:

Machine (std::vector<std::pair<std::string, int>> &patterns, int endJokers) {

this->root = new State(' ', nullptr); this->patterns = patterns;

this->endJokers = endJokers; this->addPatterns(patterns); this->addSuffixLinks();

}

~Machine () {

root->deleteChildren(); delete root;

+) {

}

std::vector<int> AhoCorasick (std::string &text) { auto curState = root;

std::vector<int> strNums(text.size(), 0);

for (int index = 0; index < text.size() - endJokers; index+

auto nextState = curState->getChild(text[index]); while (nextState == nullptr) {

if (curState == this->root) { nextState = curState; break;

}

curState = curState->getSuffixLink();

nextState = curState->getChild(text[index]);

}

for (auto &terminal : nextState->getTerminal()) { if (index - terminal + 1 >= 0) {

strNums[index - terminal + 1]++;

}

}

auto tempState = nextState->getEndLink(); while (tempState != nullptr) {

for (auto &terminal : tempState->getTerminal()) { if (index - terminal + 1 >= 0) {

strNums[index - terminal + 1]++;

}

}

tempState = tempState->getEndLink();

}

curState = nextState;

}

std::vector<int> result;

for (int index = 0; index < strNums.size(); index++) { if (strNums[index] == patterns.size()) {

result.push\_back(index + 1);

}

}

return result;

}

protected:

void addPatterns (std::vector<std::pair<std::string, int>> &newPatterns) {

for (auto &str : newPatterns) { State \*curState = root;

for (int i = 0; i < str.first.size(); i++) {

auto nextState = curState->getChild(str.first[i]); if (nextState == nullptr) {

nextState = new State(str.first[i], curState); curState->addChild(nextState);

}

if (i == str.first.size() - 1) { nextState->changeTerminal(str.second);

} else {

curState = nextState;

}

}

}

}

void addSuffixLinks () { std::queue<State \*> queue; State \*curState = root; queue.push(root);

while (!queue.empty()) { curState->addSuffixLink(); queue.pop();

for (auto &node : curState->getChildren()) { queue.push(node);

}

}

private:

}

curState = queue.front();

State \*root;

std::vector<std::pair<std::string, int>> patterns; int endJokers;

};