МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Ахо-Корасик

Студент гр. 9383 Мосин К.К.

Преподаватель Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Найти набор подстрок в исходном тексте, используя алгоритм Ахо-Корасик.

Задание.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T,$1 \leq |T| \leq 100000$).

Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P=\{p1, \ldots, pn\}$ $1 \leq |pi| \leq 75$

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.

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 P в текст T.

Например, образец ab??c? 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}

Выполнение работы.

Для хранения шаблонов строк использовался бор. Помимо бора, строились суффиксные ссылки для каждого узла. На каждой итерации обхода текста алгоритм Ахо-Корасик выбирает наибольший суффикс данной подстроки, обходя бор и проверяя является ли он искомым шаблоном. После нахождения или отсутствия шаблона происходит перемещение по суффиксной ссылке и продолжение считывание текста.

Для реализации поиска с джокером, исходный текст разбивается на подстроки с запоминанием индексов разбиения.

Анализ алгоритма.

Для поиска шаблона строк строится бор и происходит обход текста. Построение бора выполняется за O(n), где n - сумма длин всех шаблонов. Для поиска заданных шаблонов просматривается весь текст и идет поиск совпадений. Все это занимает O(n + m + k), где n - сумма длин шаблонов, m - длинна текста и k - число совпадений.

Вывод

В ходе лабораторной работы был реализован алгоритм Ахо-Корасик. Для хранения шаблонов строк использовалась структура бор, а также был разработан алгоритм, определяющий наибольший суффикс для каждой вершины в боре.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5_1.h

```
#pragma once

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

struct Trie {
    std::map<char, Trie*> children;
    Trie* suffix_link = nullptr;
    int order = 0;
    int depth = 0;
};

void add(Trie* root, std::string& string, int order);
void build(Trie* root);
std::vector<std::pair<int, int>> Aho_Corasick(Trie* root, std::string& text);
```

Название файла: lab5_2.h

```
#pragma once

#include <algorithm>
#include <iostream>
#include <cstring>
#include <string>
```

```
#include <vector>
#include <queue>
#include <map>

struct Trie {
    std::map<char, Trie*> children;
    Trie* suffix_link = nullptr;
    std::vector<int> wild_card_position;
};

void add(Trie* root, std::string& string, int position);
int split(Trie* root, std::string& pattern, char wild_card);
void build(Trie* root);
std::vector<int> Aho_Corasick(Trie* root, std::string& text, int count, int size);
```

Название файла: lab5_1.cpp

```
#include "lab5_1.h"

void add(Trie* root, std::string& string, int order) {
    Trie* node = root;
    int depth = 1;
    for (auto symbol : string) {
        if (!node->children[symbol]) {
            node->children[symbol] = new Trie();
        }
        node = node->children[symbol];
        node->depth = depth++;
    }
    node->order = order;
}
```

```
void build(Trie* root) {
    root->suffix_link = root;

    std::queue<Trie*> queue;
    for (auto vertex : root->children) {
        vertex.second->suffix_link = root;
        queue.push(vertex.second);
    }

    while (!queue.empty()) {
        Trie* node = queue.front();
        queue.pop();

        for (auto vertex : node->children) {
            char name = vertex.first;
            Trie* child = vertex.second;
            Trie* suffix_link = node->suffix_link;
            while (suffix_link != root && !suffix_link->children[name]) {
                suffix_link = suffix_link->suffix_link;
            }

            if (suffix_link == root && !suffix_link->children[name]) {
                child->suffix_link = suffix_link;
            }
            else {
                suffix_link = suffix_link->children[name];
                child->suffix_link = suffix_link;
            }
```

```
            queue.push(vertex.second);
        }
    }
}

std::vector<std::pair<int, int>> Aho_Corasick(Trie* root, std::string& text) {
    std::vector<std::pair<int, int>> result;

    Trie* node = root;
    for (int i = 0; i < text.size(); ++i) {
        while (!node->children[text[i]] && node != root) {
            node = node->suffix_link;
        }

        if (node->children[text[i]]) {
            node = node->children[text[i]];
        }

        Trie* temp = node;
        while (temp != root) {
            if (temp->order != 0) {
                result.push_back(std::pair<int, int>(i + 2 - temp->depth, temp->order));
            }

            temp = temp->suffix_link;
        }
    }

    return result;
}
```

Название файла: lab5_2.cpp

```
#include "lab5_2.h"

void add(Trie* root, std::string& string, int position) {
    Trie* node = root;
    for (auto symbol : string) {
        if (!node->children[symbol]) {
            node->children[symbol] = new Trie();
        }
        node = node->children[symbol];
    }
    node->wild_card_position.push_back(position);
}

int split(Trie* root, std::string& pattern, char wild_card) {
    int count = 0;
    int index = 0;
    std::string temp;
    for (auto symbol : pattern) {
        if (symbol == wild_card) {
            if (!temp.empty()) {
                add(root, temp, index - 1);
                count++;
            }
            temp.clear();
        }
        else {
            temp.push_back(symbol);
        }
```

```
        index++;
    }

    if (!temp.empty()) {
        add(root, temp, index - 1);
        count++;
    }

    return count;
}

void build(Trie* root) {
    root->suffix_link = root;

    std::queue<Trie*> queue;
    for (auto vertex : root->children) {
        vertex.second->suffix_link = root;
        queue.push(vertex.second);
    }

    while (!queue.empty()) {
        Trie* node = queue.front();
        queue.pop();

        for (auto vertex : node->children) {
            char name = vertex.first;
            Trie* child = vertex.second;
            Trie* suffix_link = node->suffix_link;
            while (suffix_link != root && !suffix_link->children[name]) {
                suffix_link = suffix_link->suffix_link;
```

```
            }

            if (suffix_link == root && !suffix_link->children[name]) {
                child->suffix_link = suffix_link;
            }
            else {
                suffix_link = suffix_link->children[name];
                child->suffix_link = suffix_link;
            }

            queue.push(vertex.second);
        }
    }
}

std::vector<int> Aho_Corasick(Trie* root, std::string& text, int count, int size) {
    std::vector<int> result;

    Trie* node = root;
    int number_of_matches[text.size()];
    std::memset(number_of_matches, 0, sizeof(number_of_matches));

    for (int i = 0; i < text.size(); ++i) {
        while (!node->children[text[i]] && node != root) {
            node = node->suffix_link;
        }

        if (node->children[text[i]]) {
            node = node->children[text[i]];
        }
```

```
        Trie* temp = node;
        while (temp != root) {
            for (auto position : temp->wild_card_position) {
                if (i - position >= 0) {
                    number_of_matches[i - position]++;
                }
            }
            temp = temp->suffix_link;
        }
    }

    for (int i = 0; i < text.size(); ++i) {
        if (number_of_matches[i] == count && i + size <= text.size()) {
            result.push_back(i + 1);
        }
    }

    return result;
}
```

Название файла: main1.cpp

```
#include "lab5_1.h"

int main(int argc, char *argv[]) {
    std::string text;
    std::cin >> text;

    int count;
    std::cin >> count;
```

```
    Trie* root = new Trie;
    std::string string;
    for (int i = 0; i < count; ++i) {
        std::cin >> string;
        add(root, string, i+1);
    }
    build(root);

    std::vector<std::pair<int, int>> result = Aho_Corasick(root, text);
    std::sort(result.begin(), result.end(), [](std::pair<int, int>& a, std::pair<int, int>& b)
-> bool {return a.first == b.first ? a.second < b.second : a.first < b.first;});
    for (auto it : result) {
        std::cout << it.first << " " << it.second << std::endl;
    }

    return 0;
}
```

Название файла: main2.cpp

```
#include "lab5_2.h"

int main(int argc, char *argv[]) {
    std::string text, pattern;
    std::cin >> text >> pattern;

    char wild_card;
    std::cin >> wild_card;

    Trie* root = new Trie;
```

```
    int count = split(root, pattern, wild_card);
    build(root);

    std::vector<int> result = Aho_Corasick(root, text, count, pattern.size());
    for (auto position : result) {
        std::cout << position << std::endl;
    }

    return 0;
}
```