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: Ахо-Корасик

Студент гр. 9383 _____ Нистратов Д.Г.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Ахо-Корасик для решения задачи по поиску точного набора образцов.

Задание.

1. Разработайте программу, решающую задачу точного поиска набора образцов.

Вход: Первая строка содержит (T, 1 <= |T| <= 10000). Вторая - число n (1 <= n <= 3000), каждая следующая из n строк содержит шаблон из набора P= {p_1, …, p_n}1 <= |p| <= 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 pp (нумерация образцов начинается с 1). Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

Вариант 4.

Реализовать режим поиска, при котором все найденные образцы не пересекаются в строке поиска (т.е. некоторые вхождения не будут найдены; решение задачи неоднозначно).

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 P в текст T.

Например, образец ab??c?ab??c? с джокером ?? встречается дважды в тексте xabvccbababcaxxabvccbababcax.

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| <= 10000)

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.

Описание работы алгоритма.

Алгоритм Ахо-Корасика построен на обходе дерева Бора, реализованный отдельной структурой, хранящей в себе список соседей (Edges), а также значение, определяющее является ли узел терминальным. Так же дерево Бора было расширено суффиксными ссылками, выполняющее переход к узлу с наибольшим суффиксом.

Алгоритм Ахо-Корасика считывает заданный текст и обходит дерево Бора, до нахождения отсутствующего символа. Затем перемещается по просчитанной суффиксной ссылке и продолжает считывание символов текста.

Отсутствие пересекающихся шаблонов было реализовано с помощью удаления терминальных узлов, которые являлись ветвями другого терминального узла.

Поиск с джокером реализован с помощью разделения шаблона на подстроки и хранения позиций в отдельном массиве.

Анализ алгоритма.

Построение дерева Бора выполняется за O(m), где m – суммарная длинна строк.

Поиск заданного шаблона в алгоритме Ахо-Карасика выполняется за O(m + n + a), где m – длина текста, n – суммарная длина шаблонов, a – длина совпадений.

Поиск заданного шаблона в алгоритме Ахо-Карасика в задачи с Джокером выполняется за O(m + n + a), где n – суммарная длина строк, m – длина текста, a – количество появлений подстрок шаблона. Обход по массиву позиций выполняется за O(m).

Описание основных функций.

addPatern(std::string str, int pattern_state) – функция добавления строки в дерево Бора.

createSuffixLinks() – обход дерева Бора для создания суффикс ссылок для каждого узла.

Find_ak() – алгоритм Ахо-Корасика

Split_by_joker() – разделение шаблона на подстроки.

Тестирование.

Тесты проведены с помощью сторонней библиотеки catch2 и описаны в файле test_1.cpp, для Ахо-Корасика, и test_2.cpp, для Ахо-Корасика с Джокером.

Для проверки результатов была написана сторонняя функция проверяющая контейнеры результатов. Примеры тестов см. в таблице 1 и таблице 2.

Таблица 1. Результаты тестирования для Ахо-Корасика

Ввод	Вывод
NTAH 3 TA N G	1 2 2 1 4 3
NTAH 3 TAGT TAG T	2 2 2 3
AAAA 1 A	1 1 2 1 3 1 4 1
AAAA 3 C T G	

Таблица 2. Результаты тестирования для Ахо-Корасика с Джокером

Ввод	Вывод
ACTANCA A$$A$ $	1
ACTANCA CA $	2

ACTANCA A44A4 4	1
ACTANCA C$$$ $	2
ACAD A$ $	1 3

Выводы.

При выполнении работы был изучен и реализован алгоритм Ахо-Корасика для поиска вхождения строки в текст, с помощью дерева Бора и суффикс ссылок. На основе данного алгоритма был описан поиск позиций вхождений подстроки в строку. Создание дерева бора реализовано за O(m). Алгоритм Ахо-Корасика реализован линейно, за O(m+ n + a).

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

Название файла: main_1.cpp

```
#include "lab5_1.hpp"

int main(){
    std::string text;
    std::cin >> text;
    AhoCorasick ak(text);
    int n = 0;
    std::cin >> n;
    for (int i = 0; i < n; i++){
        std::string temp;
        std::cin >> temp;
        ak.addPattern(temp, i+1);
    }

    ak.createSuffixLinks();
    auto result = ak.find_ak();
    std::sort(result.begin(), result.end(), [](auto a, auto b){
        if (a.first != b.first){
            return a.first < b.first;
        }
        return a.second < b.second;
    });
    for (auto pos : result){
        std::cout << pos.first << ' ' << pos.second << std::endl;
    }
    return 0;
}
```

Название файла: main_2.cpp

```
#include "lab5_2.hpp"

int main(){
    std::string text, pattern;
    std::cin >> text >> pattern;
```

```
    AhoCorasick ak(text, pattern);
    char joker;
    std::cin >> joker;
    ak.split_by_joker(joker);
    ak.createSuffixLinks();
    auto result = ak.find_ak();
    for (auto i : result){
        std::cout << i << std::endl;
    }
    return 0;
}
```

Название файла: lab5_2.hpp

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

struct BorTree;
using Edges = std::map<char, BorTree*>;
using Result = std::vector<int>;

struct BorTree
{
    Edges links;
    BorTree* suffixLink = nullptr;
    std::vector<int> pos_of_split_pattern;
};

class AhoCorasick
{
private:
    BorTree* root;
    std::string text, pattern;
    int split_pattern_ammount = 0;
    Result result;
public:
```

```
    AhoCorasick(std::string str, std::string ptrn) : root(new BorTree), text(str), pattern(ptrn){};

    void addPattern(std::string str, int level);

    void createSuffixLinks();

    void split_by_joker(char joker);

    Result find_ak();
};
```

Название файла: lab5_1.hpp

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

struct BorTree;
using Edges = std::map<char, BorTree*>;
using Result = std::vector<std::pair<int, int>>;

struct BorTree
{
    Edges links;
    BorTree* suffixLink = nullptr;
    int term = 0;
    int depth = 0;
};

class AhoCorasick
{
private:
    BorTree* root;
    std::string text;
    Result result;
```

```
public:
    AhoCorasick(std::string str) : root(new BorTree), text(str) {};

    void addPattern(std::string str, int pattern_state);

    void createSuffixLinks();

    Result find_ak();
};
```

Название файла: lab5_1.cpp

```
#include "lab5_1.hpp"

void AhoCorasick::addPattern(std::string str, int pattern_state){
    BorTree* curr_node = root;
    int d = 1;
    for (size_t i = 0; i < str.size(); i++){
        if (!curr_node->links[str[i]]){
            curr_node->links[str[i]] = new BorTree;
        }
        curr_node = curr_node->links[str[i]];
        curr_node->depth = d++;
    }
    curr_node->term = pattern_state;
}

void AhoCorasick::createSuffixLinks(){
    std::queue<BorTree*> q;
    root->suffixLink = root;
    for (auto pair : root->links){
        pair.second->suffixLink = root;
        q.push(pair.second);
    }
    while (!q.empty()){
        auto node = q.front();
        q.pop();
        for (auto pair : node->links){
            while(node->suffixLink != root && !node->suffixLink->links[pair.first]){
```

```
            node->suffixLink = node->suffixLink->suffixLink;
        }
        if (node->suffixLink == root && !node->suffixLink->links[pair.first]){
            pair.second->suffixLink = node->suffixLink;
        }
        else{
            node->suffixLink = node->suffixLink->links[pair.first];
            pair.second->suffixLink = node->suffixLink;
        }
        if (pair.second->term > 0){
            pair.second->suffixLink = root;
            pair.second->links.clear();
        }
        q.push(pair.second);
    }
  }
}

Result AhoCorasick::find_ak(){
  BorTree* curr_node = root;
  for (size_t i = 0; i < text.size(); i++){
    while (!curr_node->links[text[i]] && curr_node != root){
      curr_node = curr_node->suffixLink;
    }
    if (curr_node->links[text[i]]){
      curr_node = curr_node->links[text[i]];
    }

    BorTree* answer_node = curr_node;
    while (answer_node != root){
      if (answer_node->term > 0){
        result.push_back(std::pair<int, int>(i + 2 - answer_node->depth, answer_node->term));
      }
      answer_node = answer_node->suffixLink;
    }

  }
  return result;
```

```
}
```

Название файла: lab5_2.cpp

```
#include "lab5_2.hpp"

void AhoCorasick::addPattern(std::string str, int pattern_state){
    BorTree* curr_node = root;
    for (size_t i = 0; i < str.size(); i++){
        if (!curr_node->links[str[i]]){
            curr_node->links[str[i]] = new BorTree;
        }
        curr_node = curr_node->links[str[i]];
    }
    curr_node->pos_of_split_pattern.push_back(pattern_state);
}

void AhoCorasick::createSuffixLinks(){
    std::queue<BorTree*> q;
    root->suffixLink = root;
    for (auto pair : root->links){
        pair.second->suffixLink = root;
        q.push(pair.second);
    }
    while (!q.empty()){
        auto node = q.front();
        q.pop();
        for (auto pair : node->links){
            while(node->suffixLink != root && !node->suffixLink->links[pair.first]){
                node->suffixLink = node->suffixLink->suffixLink;
            }
            if (node->suffixLink == root && !node->suffixLink->links[pair.first]){
                pair.second->suffixLink = node->suffixLink;
            }
            else{
                node->suffixLink = node->suffixLink->links[pair.first];
                pair.second->suffixLink = node->suffixLink;
            }
```

```
            q.push(pair.second);
        }
    }
}

void AhoCorasick::split_by_joker(char joker){
    std::string split_str;
    for (int i = 0; i < pattern.size(); i++){
        if (pattern[i] == joker){
            if (!split_str.empty()){
                addPattern(split_str, i - 1);
                split_pattern_ammount++;
            }
            split_str.clear();
        }
        else{
            split_str.push_back(pattern[i]);
        }
    }
    if (!split_str.empty()){
        addPattern(split_str, pattern.size() - 2);
        split_pattern_ammount++;
    }
}

Result AhoCorasick::find_ak(){
    BorTree* curr_node = root;
    int numberMatches[text.size()];
    for (int i = 0; i < text.size(); i++){
        numberMatches[i] = 0;
    }
    for (size_t i = 0; i < text.size(); i++){
        while (!curr_node->links[text[i]] && curr_node != root){
            curr_node = curr_node->suffixLink;
        }
        if (curr_node->links[text[i]]){
            curr_node = curr_node->links[text[i]];
        }
```

```
        BorTree* answer_node = curr_node;
        while (answer_node != root){
            if (!answer_node->pos_of_split_pattern.empty()){
                for (auto pos : answer_node->pos_of_split_pattern){
                    if(i - pos >= 0){
                        numberMatches[i - pos]++;
                    }
                }
            }
            answer_node = answer_node->suffixLink;
        }
    }
    for (int i = 0; i < text.size(); i++){
        if (numberMatches[i] == split_pattern_ammount && i + pattern.size() <= text.size() ){
            result.push_back(i + 1);
        }
    }
    return result;
}
```