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

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Изучить принцип работы алгоритмов на графах (алгоритм Ахо-Корасик) Применить знания на практике, решив поставленную задачу.

Основные теоретические положения.

Алгоритм Ахо — Корасик — алгоритм поиска подстроки, разработанный Альфредом Ахо и Маргарет Корасик в 1975 году, реализует поиск множества подстрок из словаря в данной строке.

Задание.

1) 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.

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? 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 2 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.

Выполнение работы.

Алгоритм строит конечный автомат, которому затем передаёт строку поиска. Автомат получает по очереди все символы строки и переходит по соответствующим рёбрам. Если автомат пришёл в конечное состояние, соответствующая строка словаря присутствует в строке поиска. Бор является конечным автоматом, распознающим одну строку из m — но при условии, что начало строки известно. Суффиксная ссылка — это ссылка на узел, соответствующий самому длинному суффиксу, который не заводит бор в тупик (в данном случае a).

Для корневого узла суффиксная ссылка — петля. Для остальных правило таково: если последний распознанный символ — c, то осуществляется обход по суффиксной ссылке родителя, если оттуда есть дуга, нагруженная символом c, суффиксная ссылка направляется в тот узел, куда эта дуга ведёт. Иначе — алгоритм проходит по суффиксной ссылке ещё и ещё раз, пока либо не пройдёт по корневой ссылке-петле, либо не найдёт дугу, нагруженную символом c.

Переменные и структуры данных:

class Node – класс, представители которого – узлы автомата (вершины бора).

{ *char edgename;* - символ ребра родитель->вершина.

int pattern_count = 0; - номер шаблона для вершины (для 1 задания).

std::vector<int> position; - позиции строк (для 2 задания).

Bool logical = false; - логическая переменная для терминальной вершины.

Int link = -1; - суффиксная ссылка на вершину.

Int previous_node; - родитель вершины.

Int next_nodes[alphabet]; - вершины, в которые можно перейти.

Int path[alphabet]; - путь (хранит индексы вершин).

};

class Bor – класс, реализующий бор.

{ *std::vector<Node> Bor_array;* - массив, представляющий бор с суфф.ссылками.

std::vector<std::string> patterns; - массив, хранящий строки.

int link_suff(...); - функция, возвращающая индекс следующей вершины через суфф.ссылку.

int next_node(...); - функция, возвращающая индекс следующей вершины в поиске шаблонов.

void result(...); - функция для форматирования результата, приводящая данные в необходимый для задания формат.

void node_add(...); - функция, добавляющая шаблонную строку в бор.

void node_find(...); - функция, выполняющая поиск шаблонов в тексте.

std::vector<int> pattern(...); - функция для работы с джокером (разделяет строку с ним на подстроки для корректной работы бора).

void print(...); - функция для вывода результата на экран.

}

Тестирование.

Таблица 1. – результат тестирования.

Номер задания	Тест	Результат	Вывод

1	NTAG 3 TAGT TAG T	2 2 2 3	Программа работает корректно и выводит верный ответ в требуемом формате.
2	ACTANCA A$$A$ $	1	Программа работает корректно и выводит верный ответ в требуемом формате.

Выводы.

В результате работы была написана программа, решающая поставленную задачу при использовании изученных теоретических материалов. Программа было протестирована, результаты тестов удовлетворительны.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл lb5_1.cpp

```
#include <iostream>
#include <vector>
#include <map>
#define alphabet 5
class Node
{
public:
    char edgename;
    int pattern_count = 0;
    bool logical = false;
    int link = -1;
    int previous_node;
    int next_nodes[alphabet];
    int path[alphabet];
    Node(int prev, char name) : previous_node(prev), edgename(name)
    {
        for (int i = 0; i < alphabet; i++) { next_nodes[i] = -1; path[i] = -1; }
    }
};
class Bor
{   std::vector<Node> Bor_array;
    int link_suff(int node_count)
    {    if (Bor_array[node_count].link == -1)
        {   if (node_count == 0 || Bor_array[node_count].previous_node == 0)
                Bor_array[node_count].link = 0;
            else
                Bor_array[node_count].link =
next_node(link_suff(Bor_array[node_count].previous_node),

Bor_array[node_count].edgename);}
        return Bor_array[node_count].link;
    }

    int next_node(int node_count, int name)
    {   if (Bor_array[node_count].path[name] == -1){
            if (Bor_array[node_count].next_nodes[name] != -1)
                Bor_array[node_count].path[name] =
Bor_array[node_count].next_nodes[name];
            else
                if (node_count == 0)
                    Bor_array[node_count].path[name] = 0;
                else Bor_array[node_count].path[name] =
next_node(link_suff(node_count), name);
        }
        return Bor_array[node_count].path[name];
    }
```

```
    void result(std::vector <std::string>& arraystr, int node_count, int
numstring, std::string string, std::vector<std::pair<int, int>> &res)
    {
        for (int i = node_count; i != 0; i = link_suff(i))
            if (Bor_array[i].logical)  res.push_back(std::make_pair(numstring -
arraystr[Bor_array[i].pattern_count].size() + 1, Bor_array[i].pattern_count +
1));
    }
public:
    Bor()
    {
        Bor_array.push_back(Node(0, 0));
    }
    void node_add(std::map<char, int> symbols, int count, std::string string)
    {   int number = 0;
        int node;
        for (int index = 0; index < string.size(); index++)
        { char symbol = string[index];
            node = symbols[symbol];
            if (Bor_array[number].next_nodes[node] == -1){
                Bor_array.push_back(Node(number, node));
                Bor_array[number].next_nodes[node] = Bor_array.size() - 1;
            }
            number = Bor_array[number].next_nodes[node];
        }
         Bor_array[number].logical = true;
        Bor_array[number].pattern_count = count;
    }
    void node_find(std::vector<std::string> array, std::string string,
std::map<char, int> alphabets, std::vector<std::pair<int, int>> &res)
    {   int count_letter = 0;
        for (int index = 0; index < string.size(); index++)
        {   char letter = string[index];
            int s_edge = alphabets[letter];
            count_letter = next_node(count_letter, s_edge);
            result(array, count_letter, index + 1, string, res);
        }
    }};

int main()
{ std::map<char, int> letters{
        {'A', 0},
        {'C', 1},
        {'G', 2},
        {'T', 3},
        {'N', 4} };
    Bor Bor1;
    std::string document;
    int count;
    std::cin >> document >> count;
    std::string pattern;
    std::vector<std::string> pattern_array;
    std::vector<std::pair<int, int>> results;
```

```
    for (int index = 0; index < count; index++)
    {   std::cin >> pattern;
        Bor1.node_add(letters, index, pattern);
        pattern_array.push_back(pattern);  }
    Bor1.node_find(pattern_array, document, letters, results);
    sort(results.begin(), results.end());
    for (int index = 0; index < results.size(); index++)  std::cout <<
results[index].first << ' ' << results[index].second << std::endl;
    return 0;
}
```

Файл lb5_2.cpp

```
#include <iostream>
#include <sstream>
#include <vector>
#include <map>
#define alphabet 5
class Node
{
public:
    char edgename;
    std::vector<int> position;
    bool logical = false;
    int link = -1;
    int previous_node;
    int next_nodes[alphabet];
    int path[alphabet];
    Node(int prev, char name) : previous_node(prev), edgename(name)
    {position.resize(0); {
        for (int i = 0; i < alphabet; i++) { next_nodes[i] = -1; path[i] = -1; }
    }}
};
class Bor
{   std::vector<Node> Bor_array;
    std::vector<std::string> patterns;
    int link_suff(int node_count)
    {    if (Bor_array[node_count].link == -1)
        {   if (node_count == 0 || Bor_array[node_count].previous_node == 0)
                Bor_array[node_count].link = 0;
            else
                Bor_array[node_count].link =
next_node(link_suff(Bor_array[node_count].previous_node),

Bor_array[node_count].edgename);}
        return Bor_array[node_count].link;
    }

    int next_node(int node_count, int name)
    {   if (Bor_array[node_count].path[name] == -1){
            if (Bor_array[node_count].next_nodes[name] != -1)
                Bor_array[node_count].path[name] =
Bor_array[node_count].next_nodes[name];
            else
                if (node_count == 0)
```

```
                    Bor_array[node_count].path[name] = 0;
                else Bor_array[node_count].path[name] =
next_node(link_suff(node_count), name);
        }
        return Bor_array[node_count].path[name];
    }
    void result(int node, int i, std::vector<int>& extraarray, std::vector<int>
len_of_pattern)
    { for (int n = node; n != 0; n = link_suff(n)) {
            if (Bor_array[n].logical) {
                for (const auto& j : Bor_array[n].position)
                    if ((i - len_of_pattern[j] < extraarray.size()))
extraarray[i - len_of_pattern[j]]++;
            } } }
public:
    Bor()
    {
        Bor_array.push_back(Node(0, 0));
    }
    void node_add(std::map<char, int> symbols, std::string string){   int number
= 0;
        int node;
        for (int index = 0; index < string.size(); index++) {
            char letter = string[index];
            node = symbols[letter];
            if (Bor_array[number].next_nodes[node] == -1) {
                Bor_array.push_back(Node(number, node));
                Bor_array[number].next_nodes[node] = Bor_array.size() - 1;
            }
            number = Bor_array[number].next_nodes[node];
        }
            Bor_array[number].logical = true;
            patterns.push_back(string);
            Bor_array[number].position.push_back(patterns.size() - 1);
        }
    void node_find(std::map<char, int> letters, std::string& string,
                   std::vector <int>& extraarray, const std::vector <int>&
len_of_pattern)
    {   int n = 0;
        for (int index = 0; index < string.length(); index++)
        {   char symbol = string[index];
            int node = letters[symbol];
            n = next_node(n, node);
            result(n, index + 1, extraarray, len_of_pattern);
        }
    }
    std::vector<int> pattern(std::map<char, int> letters, std::stringstream&
pattern, char jester)
    {   std::vector<int> len_of_patterns;
        int len = 0;
        std::string cast;
        while (getline(pattern, cast, jester))
        { if (cast.size() > 0)
```

```
            {  len += cast.size();
                len_of_patterns.push_back(len);
                node_add(letters, cast);
            } len++;
        }
        return len_of_patterns;
    }
    void print(const std::vector<int>& extraarray, int size, int len)
    {
        for (int i = 0; i < size; i++)
            if ((extraarray[i] == patterns.size()) && (i + len <= size))
std::cout << i + 1 << std::endl;
    }};

int main()
{ std::map<char, int> letters{
        {'A', 0},
        {'C', 1},
        {'G', 2},
        {'T', 3},
        {'N', 4} };
    Bor Bor1;
    std::string string;
    std::string pattern;
    std::vector<int> len_of_patterns;
    char jester;
    std::cin >> string >> pattern >> jester;
    std::stringstream stream(pattern);
    len_of_patterns = Bor1.pattern(letters, stream, jester);
    std::vector<int> extraarray(string.size(), 0);
    Bor1.node_find(letters, string, extraarray, len_of_patterns);
    Bor1.print(extraarray, string.size(), pattern.size());
    return 0;
}
```