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: "Алгоритм Ахо-Корасик".

Студентка гр. 8383 _____ Аверина О.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Научиться реализовывать алгоритм Ахо-Корасик и реализовать с его помощью программу для поиска вхождений шаблонов в текст и поиска вхождений в текст шаблона с джокером .

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.

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 PP необходимо найти все вхождения PP в текст TT.

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}

Вход:

Текст (T, $1 \leq |T| \leq 100000$)

Шаблон (P, $1 \leq |P| \leq 40$)

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

Индивидуализации для лаб. работы № 5:

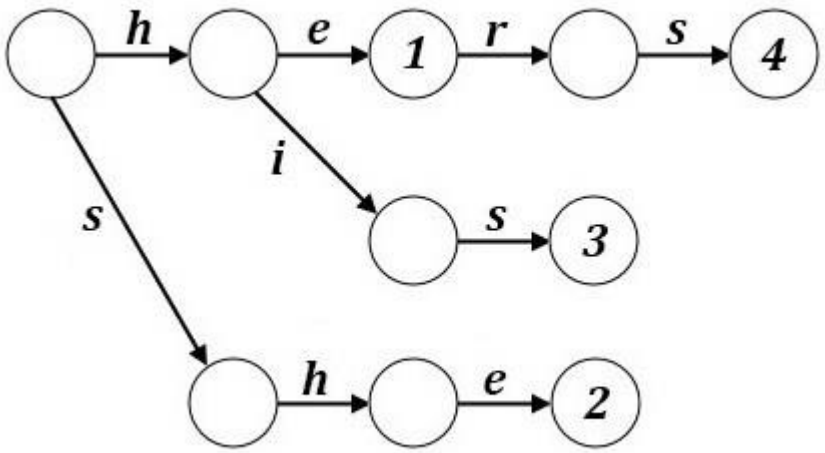
Вар. 3. Вычислить длину самой длинной цепочки из суффиксных ссылок и самой длинной цепочки из конечных ссылок в автомате.

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.

Широко применяется в системном программном обеспечении, например, используется в утилите поиска grep.

Бор (англ. *trie*) — это структура данных для компактного хранения строк. Она устроена в виде дерева, где на рёбрах между вершинами написана символы, а некоторые вершины помечены *терминальными*.



Суть алгоритма заключена в использовании структуры данных — бора из данных шаблонов и построения по нему конечного детерминированного автомата. Автомат получает по очереди все символы текста и переходит по соответствующим рёбрам. Если автомат пришёл в конечное состояние, соответствующая строка словаря присутствует в строке поиска, иначе если нельзя перейти к следующему состоянию, переходит по суффиксной ссылке и пробует снова. Суффиксная ссылка — это ссылка на узел, соответствующий самому длинному суффиксу, который не заводит бор в тупик.

Реализация алгоритма Ахо-Корасика.

На вход принимаются текст, количество шаблонов и сами шаблоны. По всем шаблонам строится бор, состоящий из экземпляров класса TNode, хранящих ссылку на родителя и потомков. Вершина бора для последнего символа каждого шаблона отмечается как терминальная. Сама структура хранится в классе Trie. Затем строится автомат, где состояния - вершины бора. Строятся суффиксные ссылки, для каждой вершины v - это ссылка на вершину в боре, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине v. Суффиксная ссылка находится следующим образом: происходит переход в вершину по суффиксной ссылке родителя, а затем из нее совершается переход по заданному символу.

После построения автомата начитается обработка теста на поиск вхождений шаблонов. Для каждого символа вызывается функция поиска потомка: если из текущей вершины есть потомок с рассматриваемым символом, переходим в него, иначе переходим по суффикс-ссылке к другой вершине и ищем потомка от нее. Если символ в боре отсутствует, текущей вершиной становится корень. После проверки символа, функция возвращает вектор с номерами шаблонов, входящими в найденный шаблон, который добавляется в вектор со всеми найденными индексами вхождения. Далее они выводятся на экран.

Для индивидуализации в функции нахождения потомка в боре в момент поиска шаблонов в найденной подстроке, при каждом переходе по суффиксной ссылке увеличивается счетчик количества суффиксных ссылок. Для подсчета максимального количества конечных ссылок счетчик увеличивается только в том случае, если встреченная вершина является терминальной.

Описание функций и структур данных

Код представлен в приложении С.

- Класс TNode - содержит элементы бора и функции для работы с ними
    - char symbol - символ, которому соответствует вершина
    - unordered_map<char, TNode*> sons - ассоциативный массив для хранения потомков вершины.
    - TNode* parent - указатель на предка вершины.
    - TNode* suffLink - суффиксная ссылка.
    - string str - подстрока, которой соответствует
    - int terminated - номер шаблона, концу которого соответствует вершина или 0, если вершина не является терминальной.
    - explicit TNode(char c): symbol(c), terminated(0){} - конструктор класса.
    - void insert(string temp, int numPattern) - функция для вставки шаблона в бор.
    - vector<int> getChain(char c, int *maxSuffLen, int *maxEndLen) - функция для поиска в боре потомка по символу.
    - void makeSuffixLinks() - функция для построения суффиксных ссылок в боре.
    - void printTrie(TNode* root) - функция для печати информации об элементах бора.
- Класс Trie - обертка над классом TNode для работы с бором.
    - TNode node - корень бора.
    - int maxSuffLen - максимальная длина цепочки суффиксных ссылок(для индивидуализации)
    - int maxEndLen - максимальная длина цепочки конечных ссылок(для индивидуализации).
    - void printMaxLenghts() - функция для вывода максимальных длин цепочек суффиксных и конечных ссылок.

- TNode* getRoot() - функция получения указателя на корень бора для вывода элементов бора.
- Остальные методы аналогичны методам класса TNode.

Оценка сложности:

Построение бора имеет линейную сложность O(n), где n - сумма длин паттернов. Построение суффиксных ссылок реализуется через обход в ширину со сложностью O(V + E), где E – кол-во ребер, V - кол-во вершин, эту сложность можно адаптировать для бора: O(n + n) = O(2n) = O(n). Перебор символов текста в боре занимает O(m), где m - длина текста. В конечном итоге сложность алгоритма составляет O(2n + m) = O(n + m).

Каждый символ шаблона представляет собой вершину бора, поэтому сложность по памяти составляет O(n). Т.к. на каждой позиции в тексте могут встретиться все *p* шаблонов, полная сложность по памяти составляет O(n + m * p).

Тестирование:

Ввод	Вывод
asdfasfaslkas 4 as fas fasf a	Max suffix link chain lenght: 2 Max end link chain lenght: 2 Index Pattern 1 1 1 4 4 2 4 3 5 1 5 4 7 2 8 1 8 4 12 1 12 4
abababa 1 aba	Max suffix link chain lenght: 2 Max end link chain lenght: 1 Index Pattern 1 1

	3 1 5 1
aacaa 2 aa ca	Max suffix link chain lenght: 2 Max end link chain lenght: 1 Index Pattern 1 1 3 2 4 1
asdfasdased 3 asd asdf as	Max suffix link chain lenght: 1 Max end link chain lenght: 1 Index Pattern 1 1 1 2 1 3 5 1 5 3 8 3

Подробный тест представлен в приложении A.

Алгоритм поиска подстрок с джокером.

Построение бора и суффиксных ссылок реализуется так же, как в первом алгоритме, за исключением того, что бор состоит не из шаблонов, а из подстрок данного шаблона без джокеров $\{P_1, P_2, \ldots, P_k\}$. При вставке их в бор, в терминальной вершине в векторе сохраняется позиция начала шаблона и его размер. Затем запускается поиск для каждого символа. Появление подстроки P_i означает возможное появление шаблона на позиции $j - i + 1$, где j - текущая позиция в тексте, i - позиция начала подстроки в маске. Затем в дополнительном векторе подсчитываются такие позиции, и если в ячейке количество вхождений равно k, значит там было вхождение шаблона.

Описание функций и структур данных

Все классы и методы аналогичны программе поиска подстрок.

Код представлен в приложении D.

Оценка сложности:

В случае с поиском шаблона с джокером построение бора будет иметь сложность O(h), где h - сумма длин подстрок. Построение суффиксных ссылок так же, как и в первом алгоритме имеет сложность O(h). Прохождение текста по бору также составляет O(n), где n - длина текста. Не учитывается прохождение дополнительно вектора, т.к. его длина равна длине исходного текста. В конечном итоге сложность алгоритма составляет O(2h + n) = O(2h + n)

Сложность по памяти, как и в первом алгоритме, составляет O(n+m*p).

Тестирование:

Ввод	Вывод
actataaha ax x	Max suffix link chain lenght: 1 Max end link chain lenght: 1 Index Pattern 1 4 6 7
asdf axsx x	Max suffix link chain lenght: 1 Max end link chain lenght: 1 Index Pattern
asasas a$a $	Max suffix link chain lenght: 1 Max end link chain lenght: 1 Index Pattern 1 3
xabvccbababcax ab??c? ?	Max suffix link chain lenght: 1 Max end link chain lenght: 1 Index Pattern 2 8

Подробный тест представлен в приложении В.

Вывод.

Было получено теоретическое представление об алгоритме Кнута-Морриса-Пратта и на основе него были реализованы программы для поиска вхождений подстроки и поиска циклического сдвига строки.

ПРИЛОЖЕНИЕ А
ПОДРОБНЫЙ ТЕСТ ДЛЯ ЗАДАНИЯ 1

```
ASDFASF
2
AS
FASF

STARTED THE CONSTRUCTION OF THE TRIE...
INSERT PATTERN: AS
INSERT PATTERN: FASF
------------------------- SUFFIX LINKS ----------------------------

THE PROCESS OF CREATING SUFFIX LINKS...
CONSIDERED VERTEX: F SUBSTRING: F

CONSIDERED VERTEX: A SUBSTRING: A

CONSIDERED VERTEX: A SUBSTRING: FA
SUFFIX LINK: A SUBSTRING: A

CONSIDERED VERTEX: S SUBSTRING: AS

CONSIDERED VERTEX: S SUBSTRING: FAS
SUFFIX LINK: S SUBSTRING: AS

CONSIDERED VERTEX: F SUBSTRING: FASF
SUFFIX LINK: F SUBSTRING: F

------------------------BUILT THE TRIE ----------------------------

STRING:
   ROOT
   CHILDREN: F A

STRING: F
   SYMBOL: '\0'     PARENT: ROOT
   SUFFIX LINK:
   CHILDREN: A

STRING: FA
   SYMBOL: A        PARENT: F
```

```
   SUFFIX LINK: A
   CHILDREN: S

STRING: FAS
   SYMBOL: S       PARENT: A
   SUFFIX LINK: AS
   CHILDREN: F

STRING: FASF
--->TERMINATED!
   SYMBOL: F       PARENT: S
   SUFFIX LINK: F
   CHILDREN:  NONE

STRING: A
   SYMBOL: '\0'     PARENT: ROOT
   SUFFIX LINK:
   CHILDREN: S

STRING: AS
--->TERMINATED!
   SYMBOL: S       PARENT: A
   SUFFIX LINK:
   CHILDREN:  NONE
---------------------- SUBSTRING SEARCH ----------------------------

SYMBOL: A  INDEX: 0
CHILD: F
CHILD: A
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 0

SYMBOL: S  INDEX: 1
CHILD: S
THIS IS THE CHILD THEY WERE LOOKING FOR!
A TERMINAL VERTEX 'S' WAS FOUND FOR THE TEMPLATE 1
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: D  INDEX: 2
CHILD: F
CHILD: A
THE END OF THE TEMPLATE FOR THIS POSITION WAS NOT FOUND.
```

SYMBOL: F INDEX: 3
CHILD: F
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: A INDEX: 4
CHILD: A
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 2
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: S INDEX: 5
CHILD: S
THIS IS THE CHILD THEY WERE LOOKING FOR!
A TERMINAL VERTEX 'S' WAS FOUND FOR THE TEMPLATE 1
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 2
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: F INDEX: 6
CHILD: F
THIS IS THE CHILD THEY WERE LOOKING FOR!
A TERMINAL VERTEX 'F' WAS FOUND FOR THE TEMPLATE 2
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 2
CURRENT MAX END LINK CHAIN LENGHT: 1

MAX SUFFIX LINK CHAIN LENGHT: 2
MAX END LINK CHAIN LENGHT: 1
INDEX PATTERN
1 1
4 2
5 1

ПРИЛОЖЕНИЕ В
ПОДРОБНЫЙ ТЕСТ ДЛЯ ЗАДАНИЯ 1

XABVCCBABA

AB??C?

?

STARTED THE CONSTRUCTION OF THE TRIE...

INSERT SUBSTRING: AB

INSERT SUBSTRING: C

---------- SUFFIX LINKS ----------

THE PROCESS OF CREATING SUFFIX LINKS...

CONSIDERED VERTEX: C SUBSTRING: C

CONSIDERED VERTEX: A SUBSTRING: A

CONSIDERED VERTEX: B SUBSTRING: AB

----------BUILT THE TRIE ----------

STRING:

ROOT

CHILDREN:C A

STRING:C

--->TERMINATED!

PARENT: ROOT

SUFFIX LINK:

CHILDREN: NONE

STRING:A

PARENT: ROOT

```
   SUFFIX LINK:
   CHILDREN:B

STRING:AB
--->TERMINATED!
   SYMBOL:B
   PARENT:A
   SUFFIX LINK:
   CHILDREN: NONE
---------- SUBSTRING SEARCH ----------

SYMBOL: X  INDEX: 0
CHILD: C
CHILD: A

SYMBOL: A  INDEX: 1
CHILD: C
CHILD: A
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 0

SYMBOL: B  INDEX: 2
CHILD: B
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: V  INDEX: 3
CHILD: C
```

CHILD: A

SYMBOL: C INDEX: 4
CHILD: C
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: C INDEX: 5
CHILD: C
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: B INDEX: 6
CHILD: C
CHILD: A

SYMBOL: A INDEX: 7
CHILD: C
CHILD: A
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: B INDEX: 8
CHILD: B
THIS IS THE CHILD THEY WERE LOOKING FOR!
CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1
CURRENT MAX END LINK CHAIN LENGHT: 1

SYMBOL: A INDEX: 9

CHILD: C

CHILD: A

THIS IS THE CHILD THEY WERE LOOKING FOR!

CURRENT MAX SUFFIX LINK CHAIN LENGHT: 1

CURRENT MAX END LINK CHAIN LENGHT: 1

MAX SUFFIX LINK CHAIN LENGHT: 1

MAX END LINK CHAIN LENGHT: 1

INDEX PATTERN

2

ПРИЛОЖЕНИЕ С

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

using namespace std;
void inputConsole(string *word, int *n ,vector<string> *templates);

class TNode{
private:
    char symbol;
    unordered_map<char, TNode*> sons;
    TNode* parent = nullptr;
    TNode* suffLink = nullptr;
    string str = "";
    int terminated;

public:
    explicit TNode(char c): symbol(c), terminated(0){}

    void insert(string temp, int numPattern){
        TNode* curr = this;

        for (char symbol: temp) {
            if (curr->sons[symbol] == nullptr) {
                curr->sons[symbol] = new TNode(symbol);  // создаем нового ребенка
                curr->sons[symbol]->parent = curr;
                curr->sons[symbol]->str = curr->str + symbol;
            }
            curr = curr->sons[symbol];
        }
        cout << "Insert pattern: " << temp << endl;
        curr->terminated = numPattern + 1;
    }


    //  поиск символа префикса в боре и всех вхождений шаблонов в его путь
    vector<int> getChain(char c, int *maxSuffLen, int *maxEndLen){
        vector<int> templatesInside;
        int currSuffLen = 0;
        int currEndLen = 0;
        static const TNode* curr = this;

        for (; curr != nullptr ; curr = curr->suffLink) {
            for (auto son: curr->sons) {
                cout << "Child: " << son.first << endl;
                if(son.first == c) {
```

```
                cout << "This is the child they were looking for! \n";
                curr = son.second;
                for (auto node = curr; node->suffLink != nullptr; node = node->suffLink,
currSuffLen++)
                    if(node->terminated > 0) {
                        currEndLen++;
                        cout << "A terminal vertex "'<<node->symbol <<"' was found for the template "
<< node->terminated << endl;
                        templatesInside.push_back(node->terminated);
                    }
                *maxSuffLen = (*maxSuffLen < currSuffLen) ? currSuffLen : *maxSuffLen;
                *maxEndLen = (*maxEndLen < currEndLen) ? currEndLen : *maxEndLen;
                cout << "Current max suffix link chain lenght: " << *maxSuffLen << endl;
                cout << "Current max end link chain lenght: " << *maxEndLen << endl;
                return templatesInside;
            }
        }
    }
    curr = this;
    cout << "The end of the template for this position was not found.\n";
    return templatesInside;
  }


  // функция для построения суффиксных ссылок
  void makeSuffixLinks(){


    queue<TNode*> q;
    for (auto son: sons) {     // можно внести это в цикл
        q.push(son.second);
    }
    while(!q.empty()) {
        TNode* curr = q.front();  //  берем вершину из очереди для обработки
        cout << "Considered vertex: " << curr->symbol << " Substring: " << curr->str << endl;
        for(pair<const char, TNode *> son: curr->sons) {
            q.push(son.second);
        }
        q.pop();

        TNode* par = curr->parent;
        if(par != nullptr)   // переходим по суфф. ссылке предыдущей вершины
            par = par->suffLink;


        while(par && par->sons.find(curr->symbol) == par->sons.end()) //проверка, есть ли
нужный символ
        {

            par = par->suffLink;                    // в потомках рассматриваемой вершины,
        }                                       // если нет, то переходим по суфф ссылке
```

```
            if(par)  {curr->suffLink = par->sons[curr->symbol]; cout << " Suffix link: " << curr->suffLink->symbol << " Substring: " << curr->suffLink->str << endl;}
                //  присваиваем суффиксную ссылку, если она найдена
            else curr->suffLink = this;      //  иначе присваиваем ссылку в себя
            cout << endl;
        }

    }


    void printTrie(TNode* root){
        TNode* curr = root;

        cout <<"\nString: " << curr->str << endl;
        if(curr->terminated > 0)
            cout <<"--->Terminated!" << "\n";

        if(curr->parent) {
            if (curr->parent->symbol != '\0') {
                cout << "    Symbol: " << curr->symbol << "    ";
                cout << "    Parent: " << curr->parent->symbol << endl;
            }
            if (curr->parent->symbol == '\0') {
                cout << "    Symbol: " << "'\\0' ";
                cout << "    Parent: root" << endl;
            }
        }
        else
            cout << "    Root" << endl;
        if(curr->suffLink)
            cout << "    Suffix link: " << curr->suffLink->str << endl;

        cout << "    Children: ";
        if(curr->sons.size() > 0) {
            for (auto c:curr->sons) {
                cout << c.first << " ";
            }
            cout << endl;
        }
        else cout << " none \n";
        for(auto tmp:curr->sons) {
            if (tmp.second) {
                printTrie(tmp.second);
            }
        }
    }
};

class Trie{
private:
    TNode node;
    int maxSuffLen;
```

```
    int maxEndLen;
public:
    Trie(): node('\0'), maxSuffLen(0), maxEndLen(0) {}

    void printMaxLenghts(){
        cout << "\nMax suffix link chain lenght: " << maxSuffLen << endl;
        cout << "Max end link chain lenght: " << maxEndLen << endl;
    }
    TNode* getRoot(){
        return &node;
    }
    vector<int> getChain(char c){
        return node.getChain(c, &maxSuffLen, &maxEndLen);
    }
    void makeSuffixLinks(){
        node.makeSuffixLinks();
    }
    void insert(string temp, int numPattern){
        node.insert(temp, numPattern);
    }
};


int main() {
    string str;
    int n;
    map<int, vector<int>> res;
    vector<string> templates(10);
//    inputConsole(&str, &n, &templates);
//----------------------------------------
    cin >> str >> n;
    templates.resize(n);
    for (int i = 0; i < n; ++i) { cin >> templates[i]; }
    //----------------------------------------

    Trie root;
    // построение бора
    cout << "\nStarted the construction of the Trie... \n";
    for (int j = 0; j < n; ++j) {
        root.insert(templates[j], j);
    }
    cout << "------------------------- Suffix links ----------------------------\n";
    cout << "\nThe process of creating suffix links...\n";
    root.makeSuffixLinks();
    cout << "-------------------------Built the trie ----------------------------\n";
    root.getRoot()->printTrie(root.getRoot());

    cout << "----------------------- Substring search ----------------------------\n";
    for (int i = 0; i < str.length(); ++i) {
        cout << "\nSymbol: " << str[i] << "  Index: " << i << endl;
        vector<int> tmp = root.getChain(str[i]);
        for (auto index: tmp) {
```

```
            res[i - templates[index - 1].size() + 2].push_back(index);
            sort(res[i - templates[index - 1].size() + 2].begin(),
                res[i - templates[index - 1].size() + 2].end());
        }
    }
    root.printMaxLenghts();

    cout << "Index Pattern\n";
    for (auto it: res) {
        for (auto k: it.second) {
            cout << "" << it.first << "      " <<  k << endl;

        }
    }
    return 0;
}

void inputConsole(string *word,int *n ,vector<string> *templates){
    ifstream file;
    file.open("input.txt");
    if (file.is_open()) {
        file >> *word >> *n;
        templates->resize(*n);
        for (int i = 0; i < *n; ++i) {
            file >> (*templates)[i];
        }
        file.close();
    } else {
        cout << "File isn't open!";
    }
}
```

ПРИЛОЖЕНИЕ D

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

using namespace std;
void inputConsole(string *word, int *n ,vector<string> *templates);

class TNode{
private:
   char symbol;
   unordered_map<char, TNode*> sons;
   TNode* parent = nullptr;
   TNode* suffLink = nullptr;
   string str = "";
   int terminated;

public:
   explicit TNode(char c): symbol(c), terminated(0){}

   void insert(string temp, int numPattern){
      TNode* curr = this;

      for (char symbol: temp) {
         if (curr->sons[symbol] == nullptr) {
            curr->sons[symbol] = new TNode(symbol);  // создаем нового ребенка
            curr->sons[symbol]->parent = curr;
            curr->sons[symbol]->str = curr->str + symbol;
         }
         curr = curr->sons[symbol];
      }
      cout << "Insert pattern: " << temp << endl;
      curr->terminated = numPattern + 1;
   }


   //  поиск символа префикса в боре и всех вхождений шаблонов в его путь
   vector<int> getChain(char c, int *maxSuffLen, int *maxEndLen){
      vector<int> templatesInside;
      int currSuffLen = 0;
      int currEndLen = 0;
      static const TNode* curr = this;

      for (; curr != nullptr ; curr = curr->suffLink) {
         for (auto son: curr->sons) {
            cout << "Child: " << son.first << endl;
            if(son.first == c) {
```

```
                cout << "This is the child they were looking for! \n";
                curr = son.second;
                for (auto node = curr; node->suffLink != nullptr; node = node->suffLink,
currSuffLen++)
                    if(node->terminated > 0) {
                        currEndLen++;
                        cout << "A terminal vertex "'<<node->symbol <<"' was found for the template "
<< node->terminated << endl;
                        templatesInside.push_back(node->terminated);
                    }
                *maxSuffLen = (*maxSuffLen < currSuffLen) ? currSuffLen : *maxSuffLen;
                *maxEndLen = (*maxEndLen < currEndLen) ? currEndLen : *maxEndLen;
                cout << "Current max suffix link chain lenght: " << *maxSuffLen << endl;
                cout << "Current max end link chain lenght: " << *maxEndLen << endl;
                return templatesInside;
            }
        }
    }
    curr = this;
    cout << "The end of the template for this position was not found.\n";
    return templatesInside;
  }


  // функция для построения суффиксных ссылок
  void makeSuffixLinks(){


    queue<TNode*> q;
    for (auto son: sons) {     // можно внести это в цикл
        q.push(son.second);
    }
    while(!q.empty()) {
        TNode* curr = q.front();  //  берем вершину из очереди для обработки
        cout << "Considered vertex: " << curr->symbol << " Substring: " << curr->str << endl;
        for(pair<const char, TNode *> son: curr->sons) {
            q.push(son.second);
        }
        q.pop();

        TNode* par = curr->parent;
        if(par != nullptr)   // переходим по суфф. ссылке предыдущей вершины
            par = par->suffLink;


        while(par && par->sons.find(curr->symbol) == par->sons.end()) //проверка, есть ли
нужный символ
        {

            par = par->suffLink;                    // в потомках рассматриваемой вершины,
        }                                       // если нет, то переходим по суфф ссылке
```

```
            if(par)  {curr->suffLink = par->sons[curr->symbol]; cout << " Suffix link: " << curr->suffLink->symbol << " Substring: " << curr->suffLink->str << endl;}
                //  присваиваем суффиксную ссылку, если она найдена
            else curr->suffLink = this;      //  иначе присваиваем ссылку в себя
            cout << endl;
        }

    }


    void printTrie(TNode* root){
        TNode* curr = root;

        cout <<"\nString: " << curr->str << endl;
        if(curr->terminated > 0)
            cout <<"--->Terminated!" << "\n";

        if(curr->parent) {
            if (curr->parent->symbol != '\0') {
                cout << "    Symbol: " << curr->symbol << "    ";
                cout << "    Parent: " << curr->parent->symbol << endl;
            }
            if (curr->parent->symbol == '\0') {
                cout << "    Symbol: " << "'\\0' ";
                cout << "    Parent: root" << endl;
            }
        }
        else
            cout << "    Root" << endl;
        if(curr->suffLink)
            cout << "    Suffix link: " << curr->suffLink->str << endl;

        cout << "    Children: ";
        if(curr->sons.size() > 0) {
            for (auto c:curr->sons) {
                cout << c.first << " ";
            }
            cout << endl;
        }
        else cout << " none \n";
        for(auto tmp:curr->sons) {
            if (tmp.second) {
                printTrie(tmp.second);
            }
        }
    }
};

class Trie{
private:
    TNode node;
    int maxSuffLen;
```

```
    int maxEndLen;
public:
    Trie(): node('\0'), maxSuffLen(0), maxEndLen(0) {}

    void printMaxLenghts(){
        cout << "\nMax suffix link chain lenght: " << maxSuffLen << endl;
        cout << "Max end link chain lenght: " << maxEndLen << endl;
    }
    TNode* getRoot(){
        return &node;
    }
    vector<int> getChain(char c){
        return node.getChain(c, &maxSuffLen, &maxEndLen);
    }
    void makeSuffixLinks(){
        node.makeSuffixLinks();
    }
    void insert(string temp, int numPattern){
        node.insert(temp, numPattern);
    }
};


int main() {
    string str;
    int n;
    map<int, vector<int>> res;
    vector<string> templates(10);
//    inputConsole(&str, &n, &templates);
//----------------------------------------
    cin >> str >> n;
    templates.resize(n);
    for (int i = 0; i < n; ++i) { cin >> templates[i]; }
    //----------------------------------------

    Trie root;
    // построение бора
    cout << "\nStarted the construction of the Trie... \n";
    for (int j = 0; j < n; ++j) {
        root.insert(templates[j], j);
    }
    cout << "------------------------- Suffix links ----------------------------\n";
    cout << "\nThe process of creating suffix links...\n";
    root.makeSuffixLinks();
    cout << "-------------------------Built the trie ----------------------------\n";
    root.getRoot()->printTrie(root.getRoot());

    cout << "----------------------- Substring search ----------------------------\n";
    for (int i = 0; i < str.length(); ++i) {
        cout << "\nSymbol: " << str[i] << "  Index: " << i << endl;
        vector<int> tmp = root.getChain(str[i]);
        for (auto index: tmp) {
```

```
            res[i - templates[index - 1].size() + 2].push_back(index);
            sort(res[i - templates[index - 1].size() + 2].begin(),
                res[i - templates[index - 1].size() + 2].end());
        }
    }
    root.printMaxLenghts();

    cout << "Index Pattern\n";
    for (auto it: res) {
        for (auto k: it.second) {
            cout << "" << it.first << "      " <<  k << endl;

        }
    }
    return 0;
}

void inputConsole(string *word,int *n ,vector<string> *templates){
    ifstream file;
    file.open("input.txt");
    if (file.is_open()) {
        file >> *word >> *n;
        templates->resize(*n);
        for (int i = 0; i < *n; ++i) {
            file >> (*templates)[i];
        }
        file.close();
    } else {
        cout << "File isn't open!";
    }
}
```