

**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  
Вариант 1**

Студент гр. 8383

Степанов В.Д.

Преподаватель

Фирсов М. А.

Санкт-Петербург

2020

## **Цель работы.**

Изучить работу и реализовать алгоритм Ахо-Корасик для нахождения набора строк в тексте.

## **Постановка задачи.**

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  $P = \{p_1 \dots p_n\} 1 \leq |p_i| \leq 75$

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 -  $i$  и  $p$

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:

CCCA

1

CC

Sample Output:

1 1

2 1

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}

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

Базовая часть лаб. работы № 5 состоит в выполнении обоих заданий на Stepik из раздела 6. Для обоих заданий на программирование должны быть версии кода с выводом промежуточных данных. В них, в частности, должны выводиться построение бора и автомата, построенный автомат (в виде, например, описания каждой вершины автомата), процесс его использования.

Вариант 1. На месте джокера может быть любой символ, за исключением заданного.

### **Описание алгоритма Ахо-Корасик.**

#### 1. Создание бора.

Для создания бора все шаблоны поиска поочередно добавляются в бор. Для этого добавляется начальная вершина, она становится текущей (корень) и для каждой буквы шаблона:

- Если переход по букве существует, он совершается.
- Иначе в боре создается новая вершина, добавляется и совершается переход в нее.

#### 2. Поиск шаблонов в строке.

Суффиксная ссылка для каждой вершины  $v$  — это вершина, в которой оканчивается найденнейший собственный суффикс строки, соответствующей вершине  $v$ .

Правило перехода в боре:

Пусть мы находимся в состоянии  $r$ , которому соответствует строка  $t$ , и хотим выполнить переход по символу  $c$ .

- Если в боре уже есть переход по букве  $c$ , этот переход совершается и мы попадаем в вершину, соответствующую строке  $tc$ .
- Если же такого ребра нет, то мы должны найти состояние, соответствующее найденнейшему собственному суффиксу строки  $t$  (найденнейшему из имеющихся в боре), и попытаться выполнить переход по букве  $c$  из него. То есть задача сводится к поиску суффиксных ссылок для вершин.

Если мы хотим узнать суффиксную ссылку для некоторой вершины  $v$ , то мы можем перейти в предка  $r$  текущей вершины (пусть  $c$  — буква, по которой из  $r$  есть переход в  $v$ ), затем перейти по его суффиксной ссылке, а затем из неё выполнить переход в автомата по букве  $c$ .

Сам поиск шаблонов в строке:

#### 1. Текущая вершина – корень бора.

2. Пока есть символы в строке:

- Совершается переход по следующей букве строки (по правилам, указанным выше).
- Из текущей вершины по суффиксам проходим до корня бора, проверяя встречу вхождений (если встречается лист, вхождение найдено).

Так как таблица переходов автомата храниться как индексный массив — расход памяти  $O(n\sigma)$ , вычислительная сложность  $O(n\sigma + H + k)$ , где  $H$  — длина текста, в котором производится поиск,  $n$  — общая длина всех слов в словаре,  $\sigma$  — размер алфавита,  $k$  — общая длина всех совпадений.

## Описание основных функций.

### Программа 1:

```
struct vertex {  
    int next[COUNT_OF_SYM];  
    bool leaf;  
    int numStr;  
    int p;  
    char pch;  
    int link;  
    int go[COUNT_OF_SYM];  
};
```

Структура для хранения бора. `int next[COUNT_OF_SYM]` – массив указателей на вершину, в которую ведёт ребро по символам, `bool leaf` – является ли концом строки, `int numStr` – индекс строки, `int p` – вершина предок, `char pch` – символ, по которому перешли из вершины предка, `int link` – суффиксная ссылка, `int go[COUNT_OF_SYM]` – переходы в автомате по каждому из символов

```
struct answer{  
    int pos;  
    int word;  
};
```

Структура для хранения ответа. `int pos` – позиция в тексте, `int word` – индекс строки.

```
bool cmpAnswer (answer a, answer b)
```

Компаратор, для сортировки вывода ответа.

```
int indOfSym (char sym)
```

Функция, которая возвращает индекс соответствующего символа. char sym – символ.

```
void addString (const std::string & s, std::vector <vertex> & t,  
int i)
```

Функция, которая добавляет строку в бор. const std::string & s – строка, std::vector <vertex> & t – бор, int i – индекс строки.

```
int go (int v, char c, std::vector <vertex> & t)
```

Функция, которая возвращает состояние в которое нужно перейти из вершины v по символу c. int v – вершина, из которой совершается переход, char c – индекс символа по которому совершается переход, std::vector <vertex> & t – бор.

```
int getLink (int v, std::vector <vertex> & t)
```

Функция состояние, в которое можно перейти по суффиксом ссылке. int v – вершина, std::vector <vertex> & t – бор.

## Программа 2:

```
struct saveString {  
    std::string stroc;  
    std::vector<int> ind;  
};
```

Структура для хранения подстроки. std::string stroc – подстрока, std::vector<int> ind – индекс начала постромки в строке.

## Тестирование

### Программа 1:

Ввод	Выход
CCCA	
1	1 1
CC	2 1

CCCA 2 CC CA	1 1 2 1 2 2
ACAACAA 2 AT CG	

```

vladislavstepanov — firstStep_out — 76x59
CCCA
1
CC
Добавляем строку: CC
    Ребра С из 0 не существует
    Создаем новое состояние 2
    Переходим в состояние 1
    Ребра С из 1 не существует
    Создаем новое состояние 3
    Переходим в состояние 2
Строка добавлена
-----
Ищем вершину для перехода из 0 по С
|----- Получение перехода оп символу
|   Добавление нового перехода
*---- Вершина найдена
Переходим в 1
Проверяем вершину 1
|----- Получение вершины по суффиксной ссылке
|   Добавляем состоянию 1 суффиксную ссылку на 0
*---- Вершина найдена
-----
Ищем вершину для перехода из 1 по С
|----- Получение перехода оп символу
|   Добавление нового перехода
*---- Вершина найдена
Переходим в 2
Проверяем вершину 2
Вершина является конечной, строка 1 найдена на с индекса 1
|----- Получение вершины по суффиксной ссылке
|----- Получение вершины по суффиксной ссылке
*---- Вершина найдена
|----- Получение перехода оп символу
*---- Вершина найдена
|   Добавляем состоянию 2 суффиксную ссылку на 1
*---- Вершина найдена
Проверяем вершину 1
|----- Получение вершины по суффиксной ссылке
*---- Вершина найдена
-----
Ищем вершину для перехода из 2 по С
|----- Получение перехода оп символу
|----- Получение вершины по суффиксной ссылке
*---- Вершина найдена
|----- Получение перехода оп символу
*---- Вершина найдена
|   Добавление нового перехода
*---- Вершина найдена
Переходим в 2
Проверяем вершину 2
Вершина является конечной, строка 1 найдена на с индекса 2
|----- Получение вершины по суффиксной ссылке
*---- Вершина найдена
Проверяем вершину 1
|----- Получение вершины по суффиксной ссылке
*---- Вершина найдена
-----
Ищем вершину для перехода из 2 по А
|----- Получение перехода оп символу

```

Рисунок 1 – Вывод программы 1

```

*----- Вершина найдена
Проверяем вершину 1
| ----- Получение вершины по суффиксной ссылке
*----- Вершина найдена
-----
Ищем вершину для перехода из 2 по А
| ----- Получение перехода оп символу
| ----- Получение вершины по суффиксной ссылке
*----- Вершина найдена
| ----- Получение перехода оп символу
| ----- Получение вершины по суффиксной ссылке
*----- Вершина найдена
| ----- Получение перехода оп символу
|   Добавление нового перехода
*----- Вершина найдена
|   Добавление нового перехода
*----- Вершина найдена
Переходим в 0
1 1
2 1

```

Рисунок 2 – Продолжение вывода программы 1

### Программа 2:

Ввод	Выход
ACTANCA A\$\$AS \$	1
ACTANCA A\$\$A \$	1 4
ATGTNGT AC!GN !	

```

ACTANCA
A$$A$
$
Разбиваем строку на подстроки
Найдена подстрока: A
Сохраняем новую подстроку A и ее индекс 0
Найдена подстрока: A
Подстрока A уже существует, добавляем индекс 3
Ребра A из 0 не существует
Создаем новое состояние 2
Переходим в состояние 1

-----
Ищем вершину для перехода из 0 по A
|----- Получение перехода оп символу
|   |----- Добавление нового перехода
*----- Вершина найдена
Переходим в 1
Проверяем вершину 1
Найдено вхождение подстроки A, увеличиваем ячейку с индексом 1
|----- Получение вершины по суффиксной ссылке
|   |----- Добавляем состоянию 1 суффиксную ссылку на 0
*----- Вершина найдена

-----
Ищем вершину для перехода из 1 по С
|----- Получение перехода оп символу
|   |----- Получение вершины по суффиксной ссылке
*----- Вершина найдена
|----- Получение перехода оп символу
|   |----- Добавление нового перехода
*----- Вершина найдена
|----- Вершина найдена
|   |----- Добавление нового перехода
*----- Вершина найдена
Переходим в 0

-----
Ищем вершину для перехода из 0 по Т
|----- Получение перехода оп символу
|   |----- Добавление нового перехода
*----- Вершина найдена
Переходим в 0

-----
Ищем вершину для перехода из 0 по А
|----- Получение перехода оп символу
*----- Вершина найдена
Переходим в 1
Проверяем вершину 1
Найдено вхождение подстроки A, увеличиваем ячейку с индексом 4
Найдено вхождение подстроки A, увеличиваем ячейку с индексом 1
|----- Получение вершины по суффиксной ссылке
*----- Вершина найдена

-----
Ищем вершину для перехода из 1 по Н
|----- Получение перехода оп символу
|   |----- Получение вершины по суффиксной ссылке
*----- Вершина найдена
|----- Получение перехода оп символу
|   |----- Добавление нового перехода
*----- Вершина найдена
|----- Вершина найдена
|   |----- Добавление нового перехода
*----- Вершина найдена
Переходим в 0

```

Рисунок 3 – Вывод программы 2

### Программа 3:

Ввод	Выход
ACTAGCANCAAAAAANA	1
AXXA	4
X	10
N	11
	12

ACTAGCACCAAAAAANA	1
AXXA	4
X	7
N	10
	11
	12
ACNAGCANCAAAAAANA	1
AXXA	4
X	7
T	10
	11
	12
	14

```

vladislavstepanov — secondStep_out — 80x27
Ищем вершину для перехода из 1 по N
|----- Получение перехода оп символу
|----- Получение вершины по суффиксной ссылке
*---- Вершина найдена
|----- Получение перехода оп символу
|----- Добавление нового перехода
*---- Вершина найдена
|----- Добавление нового перехода
*---- Вершина найдена
Переходим в 0
-----
Ищем вершину для перехода из 0 по С
|----- Получение перехода оп символу
*---- Вершина найдена
Переходим в 0
-----
Ищем вершину для перехода из 0 по А
|----- Получение перехода оп символу
*---- Вершина найдена
Переходим в 1
Проверяем вершину 1
Найдено вхождение подстроки А, увеличиваем ячейку с индексом 7
Найдено вхождение подстроки А, увеличиваем ячейку с индексом 4
|----- Получение вершины по суффиксной ссылке
*---- Вершина найдена
1

```

Рисунок 4 – Продолжение вывода программы 2

## **Выходы.**

В ходе лабораторной работы был реализован на языке C++ алгоритм Ахорн-Корасик для нахождения набора строк в тексте.

The terminal window displays the output of a C++ program. The title bar shows 'vladislavstepanov — opti — 80x60'. The program's code is as follows:

```
ACTANCA
ASSA
$
N
Разбиваем строку на подстроки
Найдена построка: A
Сохраняем новую подстроку A и ее индекс 0
Найдена построка: A
Подстрока A уже существует, добавляем индекс 3
Ребра N из 0 не существует
Создаем новое состояние 2
Переходим в состояние 1
Ребра A из 0 не существует
Создаем новое состояние 3
Переходим в состояние 2
-----
Ищем вершину для перехода из 0 по A
| ----- Получение перехода оп символу
|   | Добавление нового перехода
*----- Вершина найдена
Переходим в 2
Проверяем вершину 2
Найдено входжение подстроки A, увеличиваем ячейку с индексом 1
| ----- Получение вершины по суффиксной ссылке
|   | Добавляем состоянию 2 суффиксную ссылку на 0
*----- Вершина найдена
-----
Ищем вершину для перехода из 2 по С
| ----- Получение перехода оп символу
| ----- Получение вершины по суффиксной ссылке
*----- Вершина найдена
| ----- Получение перехода оп символу
|   | Добавление нового перехода
*----- Вершина найдена
|   | Добавление нового перехода
*----- Вершина найдена
Переходим в 0
-----
Ищем вершину для перехода из 0 по Т
| ----- Получение перехода оп символу
|   | Добавление нового перехода
*----- Вершина найдена
Переходим в 0
Проверяем вершину 2
Найдено входжение подстроки A, увеличиваем ячейку с индексом 4
Найдено входжение подстроки A, увеличиваем ячейку с индексом 1
| ----- Получение вершины по суффиксной ссылке
*----- Вершина найдена
-----
Ищем вершину для перехода из 2 по N
| ----- Получение перехода оп символу
| ----- Получение вершины по суффиксной ссылке
*----- Вершина найдена
| ----- Получение перехода оп символу
|   | Добавление нового перехода
```

Рисунок 5 – Вывод программы 3

```
vladislavstepanov — opti — 80x43  
-----  
Ищем вершину для перехода из 2 по N  
|----- Получение перехода оп символу  
|----- Получение вершины по суффиксной ссылке  
*---- Вершина найдена  
|----- Получение перехода оп символу  
| Добавление нового перехода  
*---- Вершина найдена  
|----- Вершина найдена  
*----- Вершина найдена  
Переходим в 1  
Проверяем вершину 1  
Найдено вхождение символа, который не может быть джокером, уменьшаем ячейку с индексом 4  
Найдено вхождение символа, который не может быть джокером, уменьшаем ячейку с индексом 3  
|----- Получение вершины по суффиксной ссылке  
| Добавляем состоянию 1 суффиксную ссылку на 0  
*---- Вершина найдена  
-----  
Ищем вершину для перехода из 1 по С  
|----- Получение перехода оп символу  
|----- Получение вершины по суффиксной ссылке  
*---- Вершина найдена  
|----- Получение перехода оп символу  
*---- Вершина найдена  
| Добавление нового перехода  
*----- Вершина найдена  
Переходим в 0  
-----  
Ищем вершину для перехода из 0 по А  
|----- Получение перехода оп символу  
*---- Вершина найдена  
Переходим в 2  
Проверяем вершину 2  
Найдено вхождение подстрки А, увеличиваем ячейку с индексом 7  
Найдено вхождение подстрки А, увеличиваем ячейку с индексом 4  
|----- Получение вершины по суффиксной ссылке  
*---- Вершина найдена  
1  
[Процесс завершен]
```

Рисунок 6 – Продолжение вывода программы 3

## ПРИЛОЖЕНИЕ А

### ПРОГРАММА 1

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

#define COUNT_OF_SYM 5

struct vertex {
    int next[COUNT_OF_SYM];           // массив указателей на вершину, в
    которую ведёт ребро по символам
    bool leaf;                      // является ли концом строки

    int numStr;                     // индекс строки

    int p;                          // вершина предок
    char pch;                       // символ, по которому перешли из
    вершины предка
    int link;                       // суффиксная ссылка
    int go[COUNT_OF_SYM];           // переходы в автомате по каждому
    из символов
};

struct answer{                      // структура для хранения ответа
    int pos;
    int word;
};

bool cmpAnswer (answer a, answer b){   // компаратор, для
    сортировки вывода ответа

    if (a.pos == b.pos) return a.word < b.word;
    return a.pos < b.pos;
}

int indOfSym (char sym){             // функция, которая
    возвращает индекс соответствующего символа
    switch (sym) {
        case 'A':
            return 0;
    }
}
```

```

        case 'C':
            return 1;
        case 'G':
            return 2;
        case 'T':
            return 3;
        case 'N':
            return 4;
        default:
            return -1;
    }
}

void addString (const std::string & s, std::vector <vertex> & t,
int i) {

    int v = 0;                      // начинаем добавление из корня
    size_t sz = t.size();           // количество вершин бора

    for (int i = 0; i < s.length(); ++i) {      // проходимся по
всей строке

        char c = indOfSym(s[i]);           // получаем индекс
элемента

        if (t[v].next[c] == -1) {          // если перехода не
существует, то создаем его

            vertex a;
            t.push_back(a);

            for (int j = 0; j < COUNT_OF_SYM; j++) {
                t[sz].next[j] = -1;
                t[sz].go[j] = -1;
            }

            t[sz].link = -1;
            t[sz].numStr = -1;
            t[sz].p = v;
        }
    }
}

```

```

        t[sz].pch = c;
        t[v].next[c] = (int) sz++; // добавляем к старой
вершине ребро до новой
    }

    v = t[v].next[c]; // переходим в новую вершину
}

t[v].leaf = true; // последняя вершина является концом
строки
t[v].numStr = i; // строки с индексом i
}

int go (int v, char c, std::vector <vertex> & t);

int getLink (int v, std::vector <vertex> & t) {
    if (t[v].link == -1) // если еще не переходили
        if (v == 0 || t[v].p == 0) // если находимся в корне
или предок корень
        t[v].link = 0;
    else // переходим по суффиксной ссылки родителя
        t[v].link = go (getLink (t[v].p, t), t[v].pch, t); // пытаемся получить состояние, которое можно получить по символу
    return t[v].link; // по которому пришли из родителя
}

int go(int v, char c, std::vector <vertex> & t) {
    if (t[v].go[c] == -1) // если еще не совершали переход по символу c
        if (t[v].next[c] != -1) // если в боре существует переход по символу c
            t[v].go[c] = t[v].next[c]; // записываем переход
        else // иначе не существует перехода в боре
            t[v].go[c] = v == 0 ? 0 : go (getLink (v,t), c, t); // пытаемся перейти в вершине, доступной по суффиксной ссылке
    return t[v].go[c];
}

int main () {

```

```

    std::string text;                                // текст поиска
    int k;                                         // количество строк
    std::vector<answer> answers;                   // вектор для формирования
    ответа
    std::vector<vertex> t;                         // вектор, в котором храниться
    бор

//----- Считывание

    std::cin >> text;

    std::cin >> k;

    std::string *arr = new std::string[k];

    for (int i = 0; i < k; i++) {
        std::cin >> arr[i];
    }

//----- Создание бора

    vertex a;
    t.push_back(a);                                // добавление
    корня

    for (int i = 0; i < COUNT_OF_SYM; i++) {
        t[0].next[i] = -1;
        t[0].go[i] = -1;
    }

    for (int i = 0; i < k; i++) {                  // добавление
строк в бор
        addString(arr[i], t, i);
    }

    for (int i = 0; i < COUNT_OF_SYM; i++) {      // создание петли
из корня в корень
        if (t[0].next[i] == -1)                      // для символов,
которые не имеют ребра с корнем
            t[0].next[i] = 0;
    }

//----- Поиск

```

```

        int curr = 0;                                // начинаем поиск
из корня

        for (int i = 0; i < text.length(); i++) {      // проходимся по
всему тексту

            curr = go(curr, indOfSym(text[i]), t);    // переходим к
следующей вершине

            for (int next = curr; next != 0; next = getLink(next, t)) {
// проеверяем данную вершину и все вершины

// по которым можно перейти по суффиксным ссылкам
                if (t[next].leaf) {
// если вершина является концом какой-то строки
                    answer a;
// то записываем номер строки и положение в ответ
                    a.pos = i - arr[t[next].numStr].size() + 2;
                    a.word = t[next].numStr + 1;
                    answers.push_back(a);
                }
            }
        }

//----- Вывод

std::sort(answers.begin(), answers.end(), cmpAnswer);
// сортируем ответ

for (int i = 0; i < answers.size(); i++) {
    std::cout << answers[i].pos << " " << answers[i].word
<< std::endl;
}

}

```

## ПРИЛОЖЕНИЕ Б

### ПРОГРАММА 2

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

#define COUNT_OF_SYM 5

struct vertex {
    int next[COUNT_OF_SYM];           // массив указателей на вершину, в
которую ведёт ребро по символам
    bool leaf;                      // является ли концом строки
    int numStr;                     // индекс строки
    int p;                          // вершина предок
    char pch;                       // символ, по которому перешли из
вершины предка
    int link;                       // суффиксная ссылка
    int go[COUNT_OF_SYM];           // переходы в автомате по каждому
из символов
};

struct saveString {                // структура для хранения
подстроки
    std::string stroc;             // подстрока
    std::vector<int> ind;          // индекс в строке
};

bool cmpAnswer (int a, int b){     // компаратор, для сортировки
вывода ответа
    return a < b;
}

int indOfSym (char sym){          // функция, которая
возвращает индекс соответствующего символа
    switch (sym) {
        case 'A':
            return 0;
        case 'C':
```

```

        return 1;
    case 'G':
        return 2;
    case 'T':
        return 3;
    case 'N':
        return 4;
    default:
        return -1;
    }
}

void addString (const std::string & s, std::vector <vertex> & t,
int i) {

    int v = 0;                      // начинаем добавление из корня
    size_t sz = t.size();           // количество вершин бора

    for (int i = 0; i < s.length(); ++i) {      // проходимся по
        всей строке

        char c = indOfSym(s[i]);           // получаем индекс
        элемента

        if (t[v].next[c] == -1) {          // если перехода не
            существует, то создаем его

            vertex a;
            t.push_back(a);

            for (int j = 0; j < COUNT_OF_SYM; j++) {
                t[sz].next[j] = -1;
                t[sz].go[j] = -1;
            }

            t[sz].link = -1;
            t[sz].numStr = -1;
            t[sz].p = v;
            t[sz].pch = c;
        }
    }
}

```

```

        t[v].next[c] = (int) sz++; // добавляем к старой
вершине ребро до новой
    }

    v = t[v].next[c]; // переходим в новую вершину
}

t[v].leaf = true; // последняя вершина является концом
строки
t[v].numStr = i; // строки с индексом i
}

int go (int v, char c, std::vector <vertex> & t);

int getLink (int v, std::vector <vertex> & t) {
    if (t[v].link == -1) // если еще не переходили
        if (v == 0 || t[v].p == 0) // если находимся в корне
или предок корень
        t[v].link = 0;
    else // переходим по суффиксной ссылки родителя
        t[v].link = go (getLink (t[v].p, t), t[v].pch, t); // пытаемся получить состояние, которое можно получить по символу
    return t[v].link; // по которому пришли из родителя
}

int go(int v, char c, std::vector <vertex> & t) {
    if (t[v].go[c] == -1) // если еще не совершали переход по символу c
        if (t[v].next[c] != -1) // если в боре существует переход по символу c
            t[v].go[c] = t[v].next[c]; // записываем переход
        else // иначе не существует перехода в боре
            t[v].go[c] = v == 0 ? 0 : go (getLink (v,t), c, t); // пытаемся перейти в вершине, доступной по суффиксной ссылке
    return t[v].go[c];
}

int main () {

```

```

    std::string text;           // текст поиска
    int k;                     // количество строк
    std::vector<int> answers;   // вектор для формирования
ответа
    std::vector<vertex> t;      // вектор, в котором храниться
бор

    std::string str;           // строка с джокерами
    char joker;                // джокер

//----- Считывание

    std::cin >> text;
    std::cin >> str;
    std::cin >> joker;

    std::vector <int> C(text.length()); // вектор для записи
количество вхождений подстрок
    std::vector <saveString> arr;      // вектор для хранения
подстрок

    for (int j = 0; j < str.length(); j++) {           // разделение
строки на подстроки

        if (str[j] != joker) {

            std::string s;
            int saveJ = j;
            k++;

            for (j; str[j] != joker && j < str.length(); j++) {
                s.push_back(str[j]);
            }

            bool inArr = false;

            int i = 0;

            for (i; i < arr.size(); i++) {

                if (arr[i].stroc == s) {

```

```

        inArr = true;
        break;
    }
}

if (inArr == false) {

    saveString ss;

    ss.stroc = s;
    ss.ind.push_back(saveJ);

    arr.push_back(ss);
} else {
    arr[i].ind.push_back(saveJ);
}
}

//----- Создание бора
vertex a;
t.push_back(a); // добавление
корня

for (int i = 0; i < COUNT_OF_SYM; i++) {
    t[0].next[i] = -1;
    t[0].go[i] = -1;
}

for (int i = 0; i < arr.size(); i++) { // добавление строк в бор
    addString(arr[i].stroc, t, i);
}

for (int i = 0; i < COUNT_OF_SYM; i++) { // создание петли
из корня в корень
    if (t[0].next[i] == -1) // для символов,
которые не имеют ребра с корнем
        t[0].next[i] = 0;
}

}

```

```

//----- Поиск

    int curr = 0;                                // начинаем поиск
из корня

    for (int i = 0; i < text.length(); i++) {      // проходимся по
всему тексту

        curr = go(curr, indOfSym(text[i]), t);    // получение
следующей вершины

        for (int next = curr; next != 0; next = getLink(next, t)) {
// проверяем данную вершину и все вершины

// по которым можно перейти по суффиксным ссылкам

// если вершина является концом какой-то строки
        if (t[next].leaf) {
// то записываем номер строки и положение в ответ

            int indInText = i -
arr[t[next].numStr].stroc.length()+1;    // индекс подстроки в
тексте

            for (int j = 0; j <
arr[t[next].numStr].ind.size(); j++) {      // проходимся по всем
индексам вхождения подстроки в строку
                int indInStr = arr[t[next].numStr].ind[j];
// индекс подстроки в строке

                if ((indInText - indInStr >= 0)) {
                    C[indInText - indInStr]++;
// увеличиваем счетчик
                }
            }
        }
    }

//----- Вывод

```

```
for (int i = 0; i <= text.length() - str.length(); i++) {      //  
    проходимся по всевозможным вхождениям  
  
    if (C[i] == k) {                                         //  
        если строка входит в текст начиная с индекса i  
        answers.push_back(i+1);                                //  
        то в ячейке i должно быть число равное кол-ву подстрок  
    }  
}  
  
std::sort(answers.begin(), answers.end(), cmpAnswer);          //  
сортируем ответ  
  
  
for (int i = 0; i < answers.size(); i++) {  
    std::cout << answers[i]<< std::endl;  
}  
}
```

## ПРИЛОЖЕНИЕ В

### ПРОГРАММА 3

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

#define COUNT_OF_SYM 5
int deep = 0;

struct vertex {
    int next[COUNT_OF_SYM];           // массив указателей на вершину, в
которую ведёт ребро по символам
    bool leaf;                      // является ли концом строки
    int numStr;                     // индекс строки
    int p;                          // вершина предок
    char pch;                       // символ, по которому перешли из
вершины предка
    int link;                       // суффиксная ссылка
    int go[COUNT_OF_SYM];           // переходы в автомате по каждому
из символов
};

struct saveString {                // структура для хранения
подстроки
    std::string stroc;             // подстрока
    std::vector<int> ind;          // индекс в строке
};

void printSpace () {
    for (int i = 0; i < deep; i++)
        std::cout << " ";
}

bool cmpAnswer (int a, int b){      // компаратор, для сортировки
вывода ответа
    return a < b;
}
```

```

int indOfSym (char sym) {                                // функция, которая
    возвращае индекс соответствующего символа
    switch (sym) {
        case 'A':
            return 0;
        case 'C':
            return 1;
        case 'G':
            return 2;
        case 'T':
            return 3;
        case 'N':
            return 4;
        default:
            return -1;
    }
}

```

```

void addString (const std::string & s, std::vector <vertex> & t,
int i) {

    int v = 0;                      // начинаем добавление из корня
    size_t sz = t.size();           // количество вершин бора

    for (int i = 0; i < s.length(); ++i) {      // проходимся по
        всей строке

        char c = indOfSym(s[i]);           // получаем индекс
        элемента

        if (t[v].next[c] == -1) {          // если перехода не
            существует, то создаем его

                std::cout    << "\tРебра " << s[i] << " из " << v << " не
            существует" << std::endl
                << "\tСоздаем новое состояние " << sz+1 <<
            std::endl;

        vertex a;

```

```

        t.push_back(a);

        for (int j = 0; j < COUNT_OF_SYM; j++) {
            t[sz].next[j] = -1;
            t[sz].go[j] = -1;
        }

        t[sz].link = -1;
        t[sz].numStr = -1;
        t[sz].p = v;
        t[sz].pch = c;
        t[v].next[c] = (int) sz++; // добавляем к старой
вершине ребро до новой
    } else {
        std::cout << "\tРебра " << s[i] << " из " << v << "
найдено" << std::endl;
    }

    std::cout << "\tПереходим в состояние " << t[v].next[c] <<
std::endl;

    v = t[v].next[c]; // переходим в новую вершину
}

t[v].leaf = true; // последняя вершина является концом
строки
t[v].numStr = i; // строки с индексом i
}

int go (int v, char c, std::vector <vertex> & t);

int getLink (int v, std::vector <vertex> & t) {

    deep++;
    printSpace();
    std::cout << " |----- Получение вершины по суффиксной
ссылке" << std::endl;

    if (t[v].link == -1){ // если еще не переходили
        if (v == 0 || t[v].p == 0) // если находимся в корне
или предок корень
            t[v].link = 0;
    }
}

```

```

        else // переходим по суффиксной ссылки родителя
            t[v].link = go (getLink (t[v].p, t), t[v].pch, t); // пытаемся получить состояние, которое можно получить по символу
            printSpace();
            std::cout << "      | \tДобавляем состоянию " << v << " суффиксную ссылку на " << t[v].link << std::endl;
        }
        printSpace();
        std::cout << "      ----- Вершина найдена" << std::endl;
        deep--;
        return t[v].link;
    // по которому пришли из родителя
}

int go(int v, char c, std::vector <vertex> & t) {
    deep++;
    printSpace();
    std::cout << "      | ----- Получение перехода оп символу " << c << std::endl;
    if (t[v].go[c] == -1) { // если еще не совершали переход по символу с
        if (t[v].next[c] != -1) // если в боре существует переход по символу с
            t[v].go[c] = t[v].next[c]; // записываем переход
        else // иначе не существует перехода в боре
            t[v].go[c] = v == 0 ? 0 : go (getLink (v,t), c, t); // пытаемся перейти в вершине, доступной по суффиксной ссылке
        printSpace();
        std::cout << "      | \t Добавление нового перехода" << std::endl;
    }
    printSpace();
    std::cout << "      ----- Вершина найдена" << std::endl;
    deep--;
    return t[v].go[c];
}

int main () {

```

```

    std::string text; // текст поиска
    int k; // количество строк
    std::vector<int> answers; // вектор для формирования
ответа
    std::vector<vertex> t; // вектор, в котором храниться
бор

    std::string str; // строка с джокерами
    char joker; // джокер
    char contSym; // символ, который не может
быть джокером

//----- Считывание

    std::cin >> text;
    std::cin >> str;
    std::cin >> joker;
    std::cin >> contSym;

    std::vector <int> C(text.length()); // вектор для записи
количество вхождений подстрок
    std::vector <saveString> arr; // вектор для хранения
подстрок

    saveString c; // добавляем символ
contSym как подстроку
    c.stroc.push_back(contSym);
    arr.push_back(c);

    std::cout << "Разбиваем строку на подстроки" << std::endl;

    for (int j = 0; j < str.length(); j++) { // разделение
строки на подстроки

        if (str[j] != joker) {

            std::string s;
            int saveJ = j;
            k++;

            for (j; str[j] != joker && j < str.length(); j++)
                s.push_back(str[j]);

```

```

    }

    std::cout << " Найдена построка: " << s << std::endl;

    bool inArr = false;

    int i = 0;

    for (i; i < arr.size(); i++) {

        if (arr[i].strof == s) {
            inArr = true;
            break;
        }
    }

    if (inArr == false) {

        saveString ss;

        ss.strof = s;
        ss.ind.push_back(saveJ);

        arr.push_back(ss);
        std::cout << "\tСохраняем новую подстроку " << s
        << " и ее индекс " << saveJ << std::endl;

    } else {
        arr[i].ind.push_back(saveJ);
        std::cout << "\tПодстрока " << s
        << " уже существует, добавляем индекс " << saveJ
        << std::endl;
    }

}

if (str[j] == joker)
arr[0].ind.push_back(j);
}

```

```

//----- Создание бора
vertex a;

```

```

    t.push_back(a);                                // добавление
корня

    for (int i = 0; i < COUNT_OF_SYM; i++) {
        t[0].next[i] = -1;
        t[0].go[i] = -1;
    }

    for (int i = 0; i < arr.size(); i++) {           // добавление строк в бор
        addString(arr[i].stroc, t, i);
    }

}

for (int i = 0; i < COUNT_OF_SYM; i++) {           // создание петли
из корня в корень
    if (t[0].next[i] == -1)                         // для символов,
которые не имеют ребра с корнем
        t[0].next[i] = 0;

}

//----- Поиск

int curr = 0;                                     // начинаем поиск
из корня

for (int i = 0; i < text.length(); i++) {          // проходимся по
всему тексту

    std::cout << "-----" << std::endl
    << "Ищем вершину для перехода из " << curr << " по " <<
text[i] << std::endl;

    curr = go(curr, indOfSym(text[i]), t); // получение
следующей вершины

    std::cout << " Переходим в " << curr << std::endl;

    for (int next = curr; next != 0; next = getLink(next, t)) {
// проверяем данную вершину и все вершины

// по которым можно перейти по суффиксным ссылкам

```

```

        std::cout << "    Проверяем вершину " << next <<
std::endl;

// если вершина является концом какой-то строки
if (t[next].leaf) {
// то записываем номер строки и положение в ответ

        int indInText = i -
arr[t[next].numStr].stroc.length()+1; // индекс подстроки в
тексте

        for (int j = 0; j <
arr[t[next].numStr].ind.size(); j++) { // проходимся по всем
индексам вхождения подстроки в строку
            int indInStr = arr[t[next].numStr].ind[j];
// индекс подстроки в строке

            if ((indInText - indInStr >= 0)) {

                if (contSym != text[i])
// если это не contSym
                    C[indInText - indInStr]++;
// то увеличиваем счетчик
                std::cout << "Найдено вхождение
подстроки " << arr[t[next].numStr].stroc
                            << ", увеличиваем ячейку с индексом "
<< indInText - indInStr +1<< std::endl;
                } else {
                    C[indInText - indInStr]--;
// иначе уменьшаем
                std::cout << "Найдено вхождение
символа, который не может быть джокером, уменьшаем ячейку с
индексом " << indInText - indInStr +1<< std::endl;
                }
            }
        }
    }
}

//----- Вывод

```

```
for (int i = 0; i <= text.length() - str.length(); i++) {      //  
    проходимся по всевозможным вхождениям  
  
    if (C[i] == k) {                                         //  
        если строка входит в текст начиная с индекса i  
        answers.push_back(i+1);                                //  
        то в ячейке i должно быть число равное кол-ву подстрок  
    }  
}  
  
std::sort(answers.begin(), answers.end(), cmpAnswer);          //  
сортируем ответ  
  
for (int i = 0; i < answers.size(); i++) {  
    std::cout << answers[i]<< std::endl;  
}  
}
```