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

Студент гр. 8303 _____ Курлин Н.Н.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Научиться использовать алгоритм Ахо-Корасика множественного поиска индексов вхождений строк-паттернов в строку-текст и для для строк-паттернов без символа джокера, и с учётом этого символа путём разработки программы.

Вариант 5. Вычислить максимальное количество дуг, исходящих из одной вершины в боре; вырезать из строки поиска все найденные образцы и вывести остаток строки поиска.

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

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 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

Описание алгоритма.

Алгоритм принимает строки-паттерны и строит по ним бор следующим образом: корнем бора является корневая вершина, из которой по символу есть переход в вершину уровнем ниже. При добавлении строки, у неё перебираются все символы. Если перехода по считанному символу из текущей вершины нет, то она создаётся, если переход есть, текущей вершиной становится инцидентная этому ребру. Когда выполнен последний переход в вершину, она помечается терминальной.

Далее алгоритм считывает по одному символу из строки-текста, выполняется переход из текущей вершины (при нулевой итерации это корень) по символу специальной функцией. Если есть прямой переход из текущей вершины по символу, то функция возвращает вершину, в которую перешла. Если прямого перехода нет, то выполняется переход по суффиксной ссылке и из новой вершины предпринимается новая попытка перейти по символу. Если достигнут корень, то он же и возвращается. Представить можно так:

$$\delta(u, c) = \begin{cases} v, & \text{if } v \text{ is son by symbol } c \text{ in trie;} \\ root, & \text{if } u \text{ is root and } u \text{ has no child by symbol } c \text{ in trie} \\ \delta(\pi(u), c), & \text{else.} \end{cases}$$ — функция перехода.

, где u – вершина, c – символ, по которому нужно выполнить переход.

После выполненного перехода для текущей вершины и всех её рекурсивных суффиксных ссылок проверяется, являются ли они конечными (терминальными) – сама идея множественного поиска. Если являются, то вхождение паттерна в строку-текст найдено. Из текста считывается новый символ, начинается следующая итерация цикла алгоритма.

Алгоритм завершает работу, когда каждый символ строки-текста был обработан.

Функция, выполняющая поиск суффиксной ссылки работает следующим образом: если для заданной вершины суффиксная ссылка ещё не была найдена (ленивая инициализация), то проверяем, является ли вершина корнем или прямым сыном корня. Если является, то ссылка этой вершины – корень, иначе выполняется рекурсивный поиск суффиксной ссылки: суффиксная ссылка ищется у вершины родителя и в ней функцией перехода осуществляется переход по символу, по которому в исходную вершину найден путь от родителя. Представимо в таком виде:

$$\pi(\mathrm{u}) = \begin{cases} 0\,(\mathrm{root}), & \text{if u is root or straight son of root} \\ \delta(\pi(\mathrm{parent(u)}), \mathrm{c}) & \text{else} \end{cases}$$

Сложности алгоритма.

Сложность алгоритма по операциям:

Алгоритм строит бор за |P| * log(|E|), где |P| - сумма длин всех паттернов, |E| - мощность алфавита. Поскольку для вставки новых строк в бор необходимо выполнить |P| операций поиска по ключу и добавления пары в map, которые занимают у контейнера log(|E|) операций. Чтобы обойти бор, считывая символы из текста нужно ещё |T|*log(|E|) операций, где |T| - длина текста, поскольку по каждому символу снова нужно в мэпе находить значение по ключу. Во время обхода бора, также будут совершаться переходы по конечным ссылкам, число таких переходов максимально равно общему числу совпадений всех паттернов с текстом (t).

Получаем O ((|P| + |T|) * log(|E|) + t).

Сложность алгоритма по памяти:

O (|P| + |T|), где |P| в худшем случае обозначает число всех вершин в боре, которые нужно хранить, а |T| - длину текста.

Описание функций и структур данных.

1.

```
struct Vertex
{
    std::map<char, int> next;
    std::map<char, int> go;
    bool isTerminal = false;
    int prev;
    char prevChar;
    int suffix;
    int number;
    int deep;
};
```

Структура представления вершины бора и автомата.

next - контейнер прямых переходов по символу char в вершину с номером int

go - массив переходов (запоминаем переходы в ленивой рекурсии), используемый для вычисления суффиксных ссылок

isTerminal - является ли терминальной (конечной) вершиной (на которой заканчивается паттерн)

prev - номер предыдущей вершины (родителя)

prevChar - символ, по которому пришли в вершину

suffix - суффиксная ссылка

number - какой по счёту считанный паттерн заканчивается на этой вершине

deep - глубина в боре, равная длине строки-паттерна, заканчивающегося в этой терминальной вершине

2.

```
void addString(const std::string& str, std::vector<Vertex>&
vertexArr, int& count)
```

Функция предназначена для заполнения бора строками-паттернами. str – строка-паттерн, добавляемая в бор, vertexArr – массив вершин бора, count – число строк в боре.

3.

```
int getSuffix(int index, std::vector<Vertex>& vertexArr)
```

Функция поиска суффиксной ссылки для вершины. index – номер вершины в боре, для которой ищем ссылку. vertexArr – массив вершин бора. Возвращает номер вершины, на которую указывает суффиксная ссылка

4.

```
int go(int index, char symb, std::vector<Vertex>& vertexArr)
```

Функция перехода из вершины по символу. index – номер вершины в боре, из которой ищем путь. symb – символ, по которому нужно найти переход. Возвращает номер достигнутой вершины.

5.

```
void search(const std::string& text, std::vector<Vertex>&
vertexArr, std::vector<std::pair<int, int>>& res, const
std::vector<std::string>& patternArr)
```

Функция поиска вхождений паттернов. text – введённая строка-текст для поиска в ней, res – массив пар <индекс вхождения, номер паттерна>, patternArr – массив паттернов, vertexArr – массив вершин бора.

6.

```
void printRes(const std::vector<std::pair<int, int>>& res, const
std::vector<std::string>& patternArr, const std::string& text,
std::string& textRest)
```

Функция вывода ответа. Выводит индекс вхождения и номер паттерна, вырезает из строки-текста паттерны. res – массив пар <индекс вхождения, номер паттерна>, patternArr – массив паттернов, text – строка-текст, textRest – остаточная строка.

7.

```
void readPattern(std::vector<Vertex>& vertexArr, int& count, std::vector<std::string>& patternArr)
```

Функция считывания паттернов. vertexArr – массив вершин бора, patternArr – массив паттернов, count – число строк в боре.

8.

```
int findMaxSons(std::vector<Vertex> vertexArr)
```

Функция подсчёта максимального исходящего числа дуг в боре из одной вершины. vertexArr – массив вершин бора. Возвращает максимальное число исходящих дуг в боре из одной вершины.

9.

```
void automatePrint(std::vector <Vertex> vertexArr)
```

Функция вывода полученного автомата. vertexArr – массив вершин бора.

Тестирование.

1.

```
--------------------------------------------------------------------------------
Enter text:
asdfsddssdasdafds
Enter pattern count:
2
Enter pattern for searching in:
dfs
```

```
Adding string "dfs" in the bohr
    Current symbol: 'd'
    Current vertex: 0
    Way through 'd' wasn't found. Adding new vertex with number 1
    *previous vertex is 0, the symbol of incoming path 'd'

    Current symbol: 'f'
    Current vertex: 1
    Way through 'f' wasn't found. Adding new vertex with number 2
    *previous vertex is 1, the symbol of incoming path 'f'

    Current symbol: 's'
    Current vertex: 2
    Way through 's' wasn't found. Adding new vertex with number 3
    *previous vertex is 2, the symbol of incoming path 's'

The number of this pattern is 1
Vertex 3 is terminal, deep of the vertex is 3

Enter pattern for searching in:
sd
Adding string "sd" in the bohr
    Current symbol: 's'
    Current vertex: 0
    Way through 's' wasn't found. Adding new vertex with number 4
    *previous vertex is 0, the symbol of incoming path 's'

    Current symbol: 'd'
    Current vertex: 4
    Way through 'd' wasn't found. Adding new vertex with number 5
    *previous vertex is 4, the symbol of incoming path 'd'

The number of this pattern is 2
Vertex 5 is terminal, deep of the vertex is 2

Searching begin
    Current symbol is 'a' from text...
    Current vertex is 0
            *Finding the way from 0 through 'a'
            *This is root
            *Found way from 0 through 'a' is 0
    Achieved vertex 0
    Finding possible entrance with end suffix-links:
        Root is arrived, reading new symbol from the text
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------
    Current symbol is 's' from text...
    Current vertex is 0
            *Finding the way from 0 through 's'
            *Found way from 0 through 's' is 4
    Achieved vertex 4
    Finding possible entrance with end suffix-links:
        Current suffix-link vertex: 4
        It's not terminal vertex, getting suffix-link from this vertex

            Getting suffix-link from vertex 4
            This is a vertex with deep = 1, suffix-link = 0
            Suffix-link from vertex 4 is 0

        Root is arrived, reading new symbol from the text
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------
```

```
Current symbol is 'd' from text...
Current vertex is 4
        *Finding the way from 4 through 'd'
        *Found way from 4 through 'd' is 5
Achieved vertex 5
Finding possible entrance with end suffix-links:
    Current suffix-link vertex: 5
    The vertex is terminal (end suffix-link). The entrance found, index = 2 (pattern = "sd")
Getting suffix-link from vertex 5
        Finding suffix-link from suffix of parent-vertex (4) through d
        Getting suffix-link from vertex 4
        Suffix-link from vertex 4 is 0

        *Finding the way from 0 through 'd'
        *Found way from 0 through 'd' is 1
        Suffix-link from vertex 5 is 1

    Current suffix-link vertex: 1
    It's not terminal vertex, getting suffix-link from this vertex

        Getting suffix-link from vertex 1
        This is a vertex with deep = 1, suffix-link = 0
        Suffix-link from vertex 1 is 0

    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'f' from text...
Current vertex is 5
        *Finding the way from 5 through 'f'
        *No straight path. Finding the way from suffix-link of this vertex through 'f'
        Getting suffix-link from vertex 5
        Suffix-link from vertex 5 is 1

        *Finding the way from 1 through 'f'
        *Found way from 1 through 'f' is 2
        *Found way from 5 through 'f' is 2
Achieved vertex 2
Finding possible entrance with end suffix-links:
    Current suffix-link vertex: 2
    It's not terminal vertex, getting suffix-link from this vertex

        Getting suffix-link from vertex 2
        Finding suffix-link from suffix of parent-vertex (1) through f
        Getting suffix-link from vertex 1
        Suffix-link from vertex 1 is 0

        *Finding the way from 0 through 'f'
        *This is root
        *Found way from 0 through 'f' is 0
        Suffix-link from vertex 2 is 0

    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 's' from text...
Current vertex is 2
        *Finding the way from 2 through 's'
        *Found way from 2 through 's' is 3
Achieved vertex 3
Finding possible entrance with end suffix-links:
    Current suffix-link vertex: 3
```

```
    The vertex is terminal (end suffix-link). The entrance found, index = 3 (pattern = "dfs")
Getting suffix-link from vertex 3
        Finding suffix-link from suffix of parent-vertex (2) through s
        Getting suffix-link from vertex 2
        Suffix-link from vertex 2 is 0

        *Finding the way from 0 through 's'
        *Found way from 0 through 's' is 4
        Suffix-link from vertex 3 is 4

    Current suffix-link vertex: 4
    It's not terminal vertex, getting suffix-link from this vertex

        Getting suffix-link from vertex 4
        Suffix-link from vertex 4 is 0

    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'd' from text...
Current vertex is 3
        *Finding the way from 3 through 'd'
        *No straight path. Finding the way from suffix-link of this vertex through 'd'
        Getting suffix-link from vertex 3
        Suffix-link from vertex 3 is 4

        *Finding the way from 4 through 'd'
        *Found way from 4 through 'd' is 5
        *Found way from 3 through 'd' is 5
Achieved vertex 5
Finding possible entrance with end suffix-links:
    Current suffix-link vertex: 5
    The vertex is terminal (end suffix-link). The entrance found, index = 5 (pattern = "sd")
Getting suffix-link from vertex 5
        Suffix-link from vertex 5 is 1

    Current suffix-link vertex: 1
    It's not terminal vertex, getting suffix-link from this vertex

        Getting suffix-link from vertex 1
        Suffix-link from vertex 1 is 0

    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'd' from text...
Current vertex is 5
        *Finding the way from 5 through 'd'
        *No straight path. Finding the way from suffix-link of this vertex through 'd'
        Getting suffix-link from vertex 5
        Suffix-link from vertex 5 is 1

        *Finding the way from 1 through 'd'
        *No straight path. Finding the way from suffix-link of this vertex through 'd'
        Getting suffix-link from vertex 1
        Suffix-link from vertex 1 is 0

        *Finding the way from 0 through 'd'
        *Found way from 0 through 'd' is 1
        *Found way from 1 through 'd' is 1
        *Found way from 5 through 'd' is 1
Achieved vertex 1
```

```
Finding possible entrance with end suffix-links:
	Current suffix-link vertex: 1
	It's not terminal vertex, getting suffix-link from this vertex

		Getting suffix-link from vertex 1
		Suffix-link from vertex 1 is 0

	Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 's' from text...
Current vertex is 1
		*Finding the way from 1 through 's'
		*No straight path. Finding the way from suffix-link of this vertex through 's'
		Getting suffix-link from vertex 1
		Suffix-link from vertex 1 is 0

		*Finding the way from 0 through 's'
		*Found way from 0 through 's' is 4
		*Found way from 1 through 's' is 4
Achieved vertex 4
Finding possible entrance with end suffix-links:
	Current suffix-link vertex: 4
	It's not terminal vertex, getting suffix-link from this vertex

		Getting suffix-link from vertex 4
		Suffix-link from vertex 4 is 0

	Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 's' from text...
Current vertex is 4
		*Finding the way from 4 through 's'
		*No straight path. Finding the way from suffix-link of this vertex through 's'
		Getting suffix-link from vertex 4
		Suffix-link from vertex 4 is 0

		*Finding the way from 0 through 's'
		*Found way from 0 through 's' is 4
		*Found way from 4 through 's' is 4
Achieved vertex 4
Finding possible entrance with end suffix-links:
	Current suffix-link vertex: 4
	It's not terminal vertex, getting suffix-link from this vertex

		Getting suffix-link from vertex 4
		Suffix-link from vertex 4 is 0

	Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'd' from text...
Current vertex is 4
		*Finding the way from 4 through 'd'
		*Found way from 4 through 'd' is 5
Achieved vertex 5
Finding possible entrance with end suffix-links:
	Current suffix-link vertex: 5
	The vertex is terminal (end suffix-link). The entrance found, index = 9 (pattern = "sd")
Getting suffix-link from vertex 5
		Suffix-link from vertex 5 is 1
```

```
        Current suffix-link vertex: 1
        It's not terminal vertex, getting suffix-link from this vertex

                Getting suffix-link from vertex 1
                Suffix-link from vertex 1 is 0

        Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'a' from text...
Current vertex is 5
                *Finding the way from 5 through 'a'
                *No straight path. Finding the way from suffix-link of this vertex through 'a'
                Getting suffix-link from vertex 5
                Suffix-link from vertex 5 is 1

                *Finding the way from 1 through 'a'
                *No straight path. Finding the way from suffix-link of this vertex through 'a'
                Getting suffix-link from vertex 1
                Suffix-link from vertex 1 is 0

                *Finding the way from 0 through 'a'
                *Found way from 0 through 'a' is 0
                *Found way from 1 through 'a' is 0
                *Found way from 5 through 'a' is 0
Achieved vertex 0
Finding possible entrance with end suffix-links:
        Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 's' from text...
Current vertex is 0
                *Finding the way from 0 through 's'
                *Found way from 0 through 's' is 4
Achieved vertex 4
Finding possible entrance with end suffix-links:
        Current suffix-link vertex: 4
        It's not terminal vertex, getting suffix-link from this vertex

                Getting suffix-link from vertex 4
                Suffix-link from vertex 4 is 0

        Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'd' from text...
Current vertex is 4
                *Finding the way from 4 through 'd'
                *Found way from 4 through 'd' is 5
Achieved vertex 5
Finding possible entrance with end suffix-links:
        Current suffix-link vertex: 5
        The vertex is terminal (end suffix-link). The entrance found, index = 12 (pattern = "sd")
Getting suffix-link from vertex 5
                Suffix-link from vertex 5 is 1

        Current suffix-link vertex: 1
        It's not terminal vertex, getting suffix-link from this vertex

                Getting suffix-link from vertex 1
                Suffix-link from vertex 1 is 0
```

```
        Root is arrived, reading new symbol from the text
    ------------------------------------------------------------------------------
    ------------------------------------------------------------------------------
    Current symbol is 'a' from text...
    Current vertex is 5
            *Finding the way from 5 through 'a'
            *Found way from 5 through 'a' is 0
    Achieved vertex 0
    Finding possible entrance with end suffix-links:
        Root is arrived, reading new symbol from the text
    ------------------------------------------------------------------------------
    ------------------------------------------------------------------------------
    Current symbol is 'f' from text...
    Current vertex is 0
            *Finding the way from 0 through 'f'
            *Found way from 0 through 'f' is 0
    Achieved vertex 0
    Finding possible entrance with end suffix-links:
        Root is arrived, reading new symbol from the text
    ------------------------------------------------------------------------------
    ------------------------------------------------------------------------------
    Current symbol is 'd' from text...
    Current vertex is 0
            *Finding the way from 0 through 'd'
            *Found way from 0 through 'd' is 1
    Achieved vertex 1
    Finding possible entrance with end suffix-links:
        Current suffix-link vertex: 1
        It's not terminal vertex, getting suffix-link from this vertex

            Getting suffix-link from vertex 1
            Suffix-link from vertex 1 is 0

        Root is arrived, reading new symbol from the text
    ------------------------------------------------------------------------------
    ------------------------------------------------------------------------------
    Current symbol is 's' from text...
    Current vertex is 1
            *Finding the way from 1 through 's'
            *Found way from 1 through 's' is 4
    Achieved vertex 4
    Finding possible entrance with end suffix-links:
        Current suffix-link vertex: 4
        It's not terminal vertex, getting suffix-link from this vertex

            Getting suffix-link from vertex 4
            Suffix-link from vertex 4 is 0

        Root is arrived, reading new symbol from the text
    ------------------------------------------------------------------------------
    ------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
2 2
3 1
5 2
9 2
12 2
Rest string from text after cutting patterns from it: adsaafds
Max count of sons: 2
```

Total automate:
Connections from vertex 0:
0 --a-> 0
0 --d-> 1
0 --f-> 0
0 --s-> 4
Connections from vertex 1:
1 --a-> 0
1 --d-> 1
1 --f-> 2
1 --s-> 4
Connections from vertex 2:
2 --s-> 3
Connections from vertex 3:
3 --d-> 5
Connections from vertex 4:
4 --d-> 5
4 --s-> 4
Connections from vertex 5:
5 --a-> 0
5 --d-> 1
5 --f-> 2

2. Sample Input:

abaabaabses

3

ba

bse

ses

Sample Output:

2 1

5 1

8 2

9 3

3. Sample Input:

bbebbeaas

2

be

bb

Sample Output:

1 2

2 1

4 2

5 1

Алгоритм Ахо-Корасик с джокером.

Задание.

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с *джокером*.

В шаблоне встречается специальный символ, именуемого 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 неопределенной длины. В шаблоне входит хотя бы один символ не джокер, те шаблоны вида ??? недопустимы.

Все строки содержат символы из алфавита {*A*, *C*, *G*, *T*, *N*}

Вход:

Текст (T, $1 \le |T| \le 100000$)

Шаблон (P,$1 \le |P| \le 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания.

Sample Input:

ACT

A$

$

Sample Output:

1

Описание алгоритма.

Алгоритм принимает строку-паттерн с джокером и делит её по джокерам на подстроки-паттерны, запоминая их индексы в исходной строке. По каждой подстроке-паттерну строит бор следующим образом: корнем бора является корневая вершина, из которой по символу есть переход в вершину уровнем ниже. При добавлении строки, у неё перебираются все символы. Если перехода по считанному символу из текущей вершины нет, то она создаётся, если переход есть, текущей вершиной становится инцидентная этому ребру. Когда выполнен последний переход в вершину, она помечается терминальной.

Далее алгоритм считывает по одному символу из строки-текста, выполняется переход из текущей вершины (при нулевой итерации это корень) по символу специальной функцией. Если есть прямой переход из текущей вершины по символу, то функция возвращает вершину, в которую перешла. Если прямого перехода нет, то выполняется переход по суффиксной ссылке и из новой вершины предпринимается новая попытка перейти по символу. Если достигнут корень, то он же и возвращается. Представить можно так:

$$\delta(u, c) = \begin{cases} v, & \text{if } v \text{ is son by symbol } c \text{ in trie;} \\ root, & \text{if } u \text{ is root and } u \text{ has no child by symbol } c \text{ in trie} \\ \delta(\pi(u), c), & \text{else.} \end{cases}$$

— функция перехода.

, где u – вершина, c – символ, по которому нужно выполнить переход.

После выполненного перехода для текущей вершины и всех её рекурсивных суффиксных ссылок проверяется, являются ли они конечными (терминальными) – сама идея множественного поиска. Если являются, то вхождение паттерна в строку-текст найдено. Из текста считывается новый символ, начинается следующая итерация цикла алгоритма.

Алгоритм завершает работу, когда каждый символ строки-текста был обработан.

Функция, выполняющая поиск суффиксной ссылки работает следующим образом: если для заданной вершины суффиксная ссылка ещё не была найдена (ленивая инициализация), то проверяем, является ли вершина корнем или прямым сыном корня. Если является, то ссылка этой вершины – корень, иначе выполняется рекурсивный поиск суффиксной ссылки: суффиксная ссылка ищется у вершины родителя и в ней функцией перехода осуществляется переход по символу, по которому в исходную вершину найден путь от родителя. Представимо в таком виде:

$$\pi(\mathrm{u}) = \begin{cases} 0\,(\mathrm{root}), & \text{if u is root or straight son of root} \\ \delta(\pi(\mathrm{parent(u)}), \mathrm{c}) & \text{else} \end{cases}$$

Сложности алгоритма.

Сложность алгоритма по операциям:

Алгоритм строит бор за |P| * log(|E|), где |P| - сумма длин всех паттернов, |E| - мощность алфавита. Поскольку для вставки новых строк в бор необходимо выполнить |P| операций поиска по ключу и добавления пары в map, которые занимают у контейнера log(|E|) операций. Чтобы обойти бор, считывая символы из текста нужно ещё |T|*log(|E|) операций, где |T| - длина текста, поскольку по каждому символу снова нужно в мэпе находить значение по ключу. Во время обхода бора, также будут совершаться переходы по конечным ссылкам, число

таких переходов максимально равно общему числу совпадений всех паттернов с текстом (t). Ещё понадобится |T| операций, чтобы в массиве числа совпадений паттернов найти индексы в тексте, для которых паттерн совпал.

Получаем O ((|P| + |T|) * log(|E|) + t + |T|).

Сложность алгоритма по памяти:

O (2|P| + 2|T| + |p|), где |P| в худшем случае обозначает число всех вершин в боре, которые нужно хранить, ещё один |P| - общая длина строк с паттернами (без джокеров), необходимыми для вывода ответа, |T| - длина текста, ещё один |T| - массив количества вхождений паттернов под каждый символ строки, |p| - количество паттернов (без джокеров) для которых хранятся столько же индексов смещений.

Описание функций и структур данных.

1.

```
struct Vertex
{
    std::map<char, int> next;
    std::map<char, int> go;
    bool isTerminal = false;
    int prev;
    char prevChar;
    int suffix;
    std::vector<int> number;
    int deep;
};
```

Структура представления вершины бора и автомата.

next - контейнер прямых переходов по символу char в вершину с номером int

go - массив переходов (запоминаем переходы в ленивой рекурсии), используемый для вычисления суффиксных ссылок

isTerminal - является ли терминальной (конечной) вершиной (на которой заканчивается паттерн)

prev - номер предыдущей вершины (родителя)

prevChar - символ, по которому пришли в вершину

suffix - суффиксная ссылка

number - какой по счёту считанный паттерн заканчивается на этой вершине

deep - глубина в боре, равная длине строки-паттерна, заканчивающегося в этой терминальной вершине

2.

```
void addString(const std::string& str, std::vector<Vertex>& vertexArr, int& count)
```

Функция предназначена для заполнения бора строками-паттернами. str – строка-паттерн, добавляемая в бор, vertexArr – массив вершин бора, count – число строк в боре.

3.

```
int getSuffix(int index, std::vector<Vertex>& vertexArr)
```

Функция поиска суффиксной ссылки для вершины. index – номер вершины в боре, для которой ищем ссылку. vertexArr – массив вершин бора. Возвращает номер вершины, на которую указывает суффиксная ссылка.

4.

```
int go(int index, char symb, std::vector<Vertex>& vertexArr)
```

Функция перехода из вершины по символу. index – номер вершины в боре, из которой ищем путь. symb – символ, по которому нужно найти переход. Возвращает номер достигнутой вершины.

5.

```
void search(const std::string& text, std::vector<Vertex>&
vertexArr, std::vector<int>& res, const std::vector<int>&
patternOffsetArr, int patternLen, const std::vector<std::string>&
patternArr)
```

Функция поиска вхождений паттернов. text – введённая строка-текст для поиска в ней, res – массив числел найденных паттернов под индексом строки, patternArr – массив паттернов, vertexArr – массив вершин бора, patternOffsetArr – массив смещений подстрок-паттернов в исходной строке-паттерне с джокерами, patternLen – длина исходного паттерна с джокерами.

6.

```
void printRes(const std::vector<int>& res, int patternCount,
std::string& textRest, int patternLen, const std::string& text)
```

Функция вывода ответа. Выводит индекс вхождения, вырезает из строки-текста паттерны. res – массив чисел найденных паттернов под индексом строки, patternCount – число паттернов, patternLen – длина исходного паттерна, textRest – остаточная строка, text – строка-текст.

7.

```
void readPattern(std::vector<Vertex>& vertexArr, char& joker,
std::vector<int>& patternOffsetArr, int& patternLen,
std::vector<std::string>& patternArr)
```

Функция считывания паттернов. vertexArr – массив вершин бора, joker – символ джокера, patternArr – массив паттернов, patternOffsetArr – массив смещений подстрок-паттернов в исходной строке-паттерне с джокерами, patternLen – длина исходного паттерна с джокерами.

8.

```
int findMaxSons(std::vector<Vertex> vertexArr)
```

Функция подсчёта максимального исходящего числа дуг в боре из одной вершины. vertexArr – массив вершин бора. Возвращает максимальное число исходящих дуг в боре из одной вершины.

9.

```
void automatePrint(std::vector <Vertex> vertexArr)
```

Функция вывода полученного автомата. vertexArr – массив вершин бора.

10.

```
void split(std::string str, char joker, std::vector<std::string>&
patternArr, std::vector<int>& patternOffsetArr){
```

Функция разбиения строки-паттерна с джокерами на подпаттерны без джокеров и их индексы смещений относительно исходного паттерна. str – строка-паттерн для разбиения, joker – символ джокера, patternArr – массив паттернов, patternOffsetArr – массив смещений подстрок-паттернов в исходной строке-паттерне с джокерами.

Тестирование.

1.

```
---------------------------------------------------------------------------
Enter text:
asffaasssafasf
Enter pattern:
$fa
Enter joker:
$
---------------------------------------------------------------------------
Begin splitting
    Was found new pattern: fa
    Index of entrance in total pattern: 1
---------------------------------------------------------------------------
---------------------------------------------------------------------------
Begin bohr building
```

```
Adding string "fa" in the bohr
    Current symbol: 'f'
    Current vertex: 0
    Way through 'f' wasn't found. Adding new vertex with number 1
    *previous vertex is 0, the symbol of incoming path 'f'

    Current symbol: 'a'
    Current vertex: 1
    Way through 'a' wasn't found. Adding new vertex with number 2
    *previous vertex is 1, the symbol of incoming path 'a'

The number of this pattern is 1
Vertex 2 is terminal, deep of the vertex is 2

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Searching begin
    Current symbol is 'a' from text...
    Current vertex is 0
            *Finding the way from 0 through 'a'
            *This is root
            *Found way from 0 through 'a' is 0
    Achieved vertex 0
    Finding possible entrance with end suffix-links:
        Root is arrived, reading new symbol from the text
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------
    Current symbol is 's' from text...
    Current vertex is 0
            *Finding the way from 0 through 's'
            *This is root
            *Found way from 0 through 's' is 0
    Achieved vertex 0
    Finding possible entrance with end suffix-links:
        Root is arrived, reading new symbol from the text
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------
    Current symbol is 'f' from text...
    Current vertex is 0
            *Finding the way from 0 through 'f'
            *Found way from 0 through 'f' is 1
    Achieved vertex 1
    Finding possible entrance with end suffix-links:
        Current suffix-link vertex: 1
        It's not terminal vertex, getting suffix-link from this vertex

            Getting suffix-link from vertex 1
            This is a vertex with deep = 1, suffix-link = 0
            Suffix-link from vertex 1 is 0

        Root is arrived, reading new symbol from the text
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------
    Current symbol is 'f' from text...
    Current vertex is 1
            *Finding the way from 1 through 'f'
            *No straight path. Finding the way from suffix-link of this vertex through 'f'
            Getting suffix-link from vertex 1
            Suffix-link from vertex 1 is 0

            *Finding the way from 0 through 'f'
            *Found way from 0 through 'f' is 1
```

```
                *Found way from 1 through 'f' is 1
        Achieved vertex 1
        Finding possible entrance with end suffix-links:
            Current suffix-link vertex: 1
            It's not terminal vertex, getting suffix-link from this vertex

                Getting suffix-link from vertex 1
                Suffix-link from vertex 1 is 0

            Root is arrived, reading new symbol from the text
        ------------------------------------------------------------------------------
        ------------------------------------------------------------------------------
        Current symbol is 'a' from text...
        Current vertex is 1
                *Finding the way from 1 through 'a'
                *Found way from 1 through 'a' is 2
        Achieved vertex 2
        Finding possible entrance with end suffix-links:
            Current suffix-link vertex: 2
            The vertex is terminal (end suffix-link). The entrance found, index = 2 (pattern = "fa"). Count of entra
nce is 1 from 1 possible

                Getting suffix-link from vertex 2
                Finding suffix-link from suffix of parent-vertex (1) through a
                Getting suffix-link from vertex 1
                Suffix-link from vertex 1 is 0

                *Finding the way from 0 through 'a'
                *Found way from 0 through 'a' is 0
                Suffix-link from vertex 2 is 0

            Root is arrived, reading new symbol from the text
        ------------------------------------------------------------------------------
        ------------------------------------------------------------------------------
        Current symbol is 'a' from text...
        Current vertex is 2
                *Finding the way from 2 through 'a'
                *No straight path. Finding the way from suffix-link of this vertex through 'a'
                Getting suffix-link from vertex 2
                Suffix-link from vertex 2 is 0

                *Finding the way from 0 through 'a'
                *Found way from 0 through 'a' is 0
                *Found way from 2 through 'a' is 0
        Achieved vertex 0
        Finding possible entrance with end suffix-links:
            Root is arrived, reading new symbol from the text
        ------------------------------------------------------------------------------
        ------------------------------------------------------------------------------
        Current symbol is 's' from text...
        Current vertex is 0
                *Finding the way from 0 through 's'
                *Found way from 0 through 's' is 0
        Achieved vertex 0
        Finding possible entrance with end suffix-links:
            Root is arrived, reading new symbol from the text
        ------------------------------------------------------------------------------
        ------------------------------------------------------------------------------
        Current symbol is 's' from text...
        Current vertex is 0
                *Finding the way from 0 through 's'
                *Found way from 0 through 's' is 0
```

```
Achieved vertex 0
Finding possible entrance with end suffix-links:
    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 's' from text...
Current vertex is 0
        *Finding the way from 0 through 's'
        *Found way from 0 through 's' is 0
Achieved vertex 0
Finding possible entrance with end suffix-links:
    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'a' from text...
Current vertex is 0
        *Finding the way from 0 through 'a'
        *Found way from 0 through 'a' is 0
Achieved vertex 0
Finding possible entrance with end suffix-links:
    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'f' from text...
Current vertex is 0
        *Finding the way from 0 through 'f'
        *Found way from 0 through 'f' is 1
Achieved vertex 1
Finding possible entrance with end suffix-links:
    Current suffix-link vertex: 1
    It's not terminal vertex, getting suffix-link from this vertex

        Getting suffix-link from vertex 1
        Suffix-link from vertex 1 is 0

    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 'a' from text...
Current vertex is 1
        *Finding the way from 1 through 'a'
        *Found way from 1 through 'a' is 2
Achieved vertex 2
Finding possible entrance with end suffix-links:
    Current suffix-link vertex: 2
    The vertex is terminal (end suffix-link). The entrance found, index = 9 (pattern = "fa"). Count of entra
nce is 1 from 1 possible

        Getting suffix-link from vertex 2
        Suffix-link from vertex 2 is 0

    Root is arrived, reading new symbol from the text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Current symbol is 's' from text...
Current vertex is 2
        *Finding the way from 2 through 's'
        *No straight path. Finding the way from suffix-link of this vertex through 's'
        Getting suffix-link from vertex 2
        Suffix-link from vertex 2 is 0

        *Finding the way from 0 through 's'
```

```
                *Found way from 0 through 's' is 0
                *Found way from 2 through 's' is 0
        Achieved vertex 0
        Finding possible entrance with end suffix-links:
            Root is arrived, reading new symbol from the text
        ------------------------------------------------------------------------------
        ------------------------------------------------------------------------------
        Current symbol is 'f' from text...
        Current vertex is 0
                *Finding the way from 0 through 'f'
                *Found way from 0 through 'f' is 1
        Achieved vertex 1
        Finding possible entrance with end suffix-links:
            Current suffix-link vertex: 1
            It's not terminal vertex, getting suffix-link from this vertex

                Getting suffix-link from vertex 1
                Suffix-link from vertex 1 is 0

            Root is arrived, reading new symbol from the text
        ------------------------------------------------------------------------------
        ------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Total indexes of entrance (beginning from 1):
3
10
Rest string from text after cutting patterns from it: asassssf
Max count of sons: 1

------------------------------------------------------------------------------
Total automate:
Connections from vertex 0:
    0  --a->  0
    0  --f->  1
    0  --s->  0
Connections from vertex 1:
    1  --a->  2
    1  --f->  1
Connections from vertex 2:
    2  --a->  0
    2  --s->  0
```

2. Sample Input:

abaabaabses

ab?

?

Sample Output:

1

4

7

3. Sample Input:

bbebbeaas

?b?

?

Sample Output:

1

3

4

Выводы.

Были получены умения по использованию алгоритма Ахо-Карассик множественного поиска индексов вхождений паттернов в строку, когда паттерны состоят только из символов алфавита и для случая, когда паттерн содержит джокер(ы). Написана программа, реализующая алгоритм Ахо-Корасик и выводящая индексы вхождений паттернов в строку, максимальное число исходящих дуг одной вершины в боре, остаточный текст после вырезки паттернов и уонечный полученный автомат.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Алгоритм Ахо-Корасик

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


struct Vertex
{
    std::map<char, int> next;   //Контейнер переходов по символу char в вершину с номером int
    std::map<char, int> go;     //массив переходов (запоминаем переходы в ленивой рекурсии), используемый для вычисления суффиксных ссылок
    bool isTerminal = false;    //Является ли терминальной (конечной) вершиной (на которой заканчивается паттерн)
    int prev;                   //Номер предыдущей вершины (родителя)
    char prevChar;              //Символ, по которому пришли в вершину
    int suffix;                 //Суффиксная ссылка
    int number;                 //Какой по счёту считанный паттерн заканчивается на этой вершине (только для вывода)
    int deep;                   //глубина в боре, равная длине строки-паттерна, заканчивающегося в этой терминальной вершине
};

int go(int index, char symb, std::vector<Vertex>& vertexArr);

void addString(const std::string& str, std::vector<Vertex>& vertexArr, int& count)
{
    std::cout << "Adding string \"" << str << "\" in the bohr\n";
    int current = 0;
```

```
    for (int i = 0; i < str.size(); i++) {
        std::cout << "\tCurrent symbol: \'" << str[i] << "\'\n";
        std::cout << "\tCurrent vertex: " << current << "\n";
        if (vertexArr[current].next.find(str[i]) ==
vertexArr[current].next.end()) {    //Если переход по символу не
обнаружен и итератор указывает на конец мэпа
            std::cout << "\tWay through \'" << str[i] << "\' wasn't
found. Adding new vertex with number " << vertexArr.size() << "\n";
            Vertex ver;
//Создаём новую вершину
            ver.suffix = -1;
            ver.prev = current;
            std::cout << "\t*previous vertex is " << current << ", the
symbol of incoming path \'" << str[i] << "\'\n";
            ver.prevChar = str[i];
            vertexArr.push_back(ver);
            vertexArr[current].next[str[i]] = vertexArr.size() - 1;
        }
        else{
            std::cout << "The way through the symbol exist\n";
        }
        std::cout << std::endl;
        current = vertexArr[current].next[str[i]];      //Переход к
следующей вершине
    }

    std::cout << "The number of this pattern is " << count + 1 << "\n";
    std::cout << "Vertex " << current << " is terminal, deep of the
vertex is " << str.size() << "\n\n";
    vertexArr[current].number = ++count;                //Устанавливаем
номер считанного паттерна,
    vertexArr[current].isTerminal = true;               //Терминальную
вершину
    vertexArr[current].deep = str.size();               //Глубину
```

```
}


int getSuffix(int index, std::vector<Vertex>& vertexArr)    //Функция
поиска суффиксной ссылки для вершины index
{
    std::cout << "\t\t\tGetting suffix-link from vertex " << index <<
"\n";
    if (vertexArr[index].suffix == -1) {                    //Если
суффиксная ссылка ещё не была найдена
        if (index == 0 || vertexArr[index].prev == 0) {     //Если
вершина - корень или сын корня
            vertexArr[index].suffix = 0;
            (index == 0) ? std::cout << "\t\t\tThis is root, suffix-link
vertex = 0\n" : std::cout << "\t\t\tThis is a vertex with deep = 1,
suffix-link = 0\n";
        }
        else {                                              //Рекурсивный
поиск суфф. ссылки. Получаем ссылку родителя и выполняем
            std::cout << "\t\t\tFinding suffix-link from suffix of
parent-vertex (" << vertexArr[index].prev << ") through " <<
vertexArr[index].prevChar << "\n";
            vertexArr[index].suffix = go(getSuffix(vertexArr[index].prev,
vertexArr), vertexArr[index].prevChar, vertexArr);
        }                                                   //из неё
переход по символу, по которому попали в вершину, для
    }                                                       //которой и
ищется суфф. ссылка
    std::cout << "\t\t\tSuffix-link from vertex " << index << " is " <<
vertexArr[index].suffix << "\n\n";
    return vertexArr[index].suffix;
}
```

```
int go(int index, char symb, std::vector<Vertex>& vertexArr)
//Функция перехода из вершины index по символу symb. Если прямой переход
{
//невозможен, перейдёт по ссылке
    std::cout << "\t\t\t*Finding the way from " << index << " through \'"
<< symb << "\'\n";
    if (vertexArr[index].go.find(symb) == vertexArr[index].go.end()) {
//Если путь ещё не был найден
        if (vertexArr[index].next.find(symb) !=
vertexArr[index].next.end()) {    //Если найден прямой переход по символу
в боре
            vertexArr[index].go[symb] = vertexArr[index].next[symb];
//Добавляем в контейнер возможных переходов
        }
        else {
//Если прямого перехода нет,  получаем суфф. ссылку
            if (index == 0)
//и ищем переход из суффиксной ссылки по заданному символу
                std::cout << "\t\t\t*This is root\n";
            else
                std::cout << "\t\t\t*No straight path. Finding the way
from suffix-link of this vertex through \'" << symb << "\'\n";
            vertexArr[index].go[symb] = (index == 0 ? 0 :
go(getSuffix(index, vertexArr), symb, vertexArr));
        }
    }
    std::cout << "\t\t\t*Found way from " << index << " through \'" <<
symb << "\' is " << vertexArr[index].go[symb] << "\n";
    return vertexArr[index].go[symb];
}


void search(const std::string& text, std::vector<Vertex>& vertexArr,
std::vector<std::pair<int, int>>& res, const std::vector<std::string>&
```

```
patternArr)
{
    std::cout << "Searching begin\n";
    int curr = 0;
    for (int i = 0; i < text.size(); i++) {
        std::cout << "\tCurrent symbol is \'" << text[i] <<  "\' from
text...\n";
        std::cout << "\tCurrent vertex is " << curr << "\n";
        curr = go(curr, text[i], vertexArr);
        std::cout << "\tAchieved vertex " << curr << "\n";
        std::cout << "\tFinding possible entrance with end suffix-
links:\n";
        for (int tmp = curr; tmp != 0; tmp = getSuffix(tmp, vertexArr)) {
            std::cout << "\t\tCurrent suffix-link vertex: " << tmp
<<"\n";
            if (vertexArr[tmp].isTerminal) {
                res.push_back(std::make_pair(i + 2 - vertexArr[tmp].deep,
vertexArr[tmp].number));
                std::cout << "\t\tThe vertex is terminal (end suffix-
link). The entrance found, index = " <<
                    i + 2 - vertexArr[tmp].deep << " (pattern = \"" <<
patternArr[vertexArr[tmp].number - 1] << "\")";
            } else
                std::cout << "\t\tIt's not terminal vertex, getting
suffix-link from this vertex\n\n";
        }
        std::cout << "\t\tRoot is arrived, reading new symbol from the
text\n";
        std::cout << "\t-----------------------------------------------
--------------------------------\n";
        std::cout << "\t-----------------------------------------------
--------------------------------\n";
    }
    std::cout << "-------------------------------------------------------
```

```
------------------------------\n";
    std::cout << "-------------------------------------------------------
------------------------------\n";
}


void printRes(const std::vector<std::pair<int, int>>& res, const
std::vector<std::string>& patternArr, const std::string& text,
std::string& textRest)
{
    std::vector<bool> cutStr(text.size());          //Индексы символов в
строке, которые будут вырезаны
    for (int i = 0; i < res.size(); i++) {
        std::cout << res[i].first << " " << res[i].second << '\n';
        for (int j = 0; j < patternArr[res[i].second - 1].size(); j++)
            cutStr[res[i].first - 1 + j] = true;
    }

    for (int i = 0; i < cutStr.size(); i++){
        if (!cutStr[i])
            textRest.push_back(text[i]);            //Сохраняем только
неудалённые символы
    }
}

void readPattern(std::vector<Vertex>& vertexArr, int& count,
std::vector<std::string>& patternArr)
{
    Vertex root;
    root.prev = -1;
    root.suffix = -1;
    vertexArr.push_back(root);
    count = 0;
```

```
    int patternNumb;
    std::cout << "Enter pattern count:\n";
    std::cin >> patternNumb;

    for (int i = 0; i < patternNumb; i++) {
        std::cout << "Enter pattern for searching in:\n";
        std::string pattern;
        std::cin >> pattern;
        patternArr.push_back(pattern);
        addString(pattern, vertexArr, count);
    }
}

//Функция поиска максимального числа исходящих дуг из одной вершины бора
int findMaxSons(std::vector<Vertex> vertexArr)
{
    int max = vertexArr[0].next.size();

    for(int i = 1; i < vertexArr.size(); i++){
        if (vertexArr[i].next.size() > max)
            max = vertexArr[i].next.size();
    }
    return max;
}

void automatePrint(std::vector <Vertex> vertexArr)
{
    std::cout << "------------------------------------------------------
------------------------------\n";
    std::cout << "Total automate:\n";

    for (int i = 0; i < vertexArr.size(); i++){
        std::cout << "Connections from vertex " << i << ":\n";
        auto iter = vertexArr[i].go.begin();
```

```
        for (int j = 0; j < vertexArr[i].go.size(); j++){
            std::cout << "\t" << i << "  --" << iter->first << "->  " <<
iter->second << "\n";
            iter++;
        }
    }
}

int main() {
    std::cout << "-----------------------------------------------------
------------------------------\n";
    std::cout << "Enter text:\n";
    std::string text, textRest;
    std::cin >> text;

    std::vector<Vertex> vertexArr;
    std::vector<std::string> patternArr;
    std::vector<std::pair<int, int>> res;   //<Индекс паттерна в тексте,
номер паттерна>
    int count;

    readPattern(vertexArr, count, patternArr);
    search(text, vertexArr, res, patternArr);
    printRes(res, patternArr, text, textRest);

    std::cout << "Rest string from text after cutting patterns from it: "
<< textRest << "\n";

    int maxSonsCount = findMaxSons(vertexArr);
    std::cout << "Max count of sons: " << maxSonsCount << "\n\n";


    automatePrint(vertexArr);
```

```
    return 0;
}
```

Алгоритм Ахо-Корасик с джокером

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


struct Vertex
{
    std::map<char, int> next;   //Контейнер переходов по символу char в вершину с номером int
    std::map<char, int> go;     //массив переходов (запоминаем переходы в ленивой рекурсии), используемый для вычисления суффиксных ссылок
    bool isTerminal = false;    //Является ли терминальной (конечной) вершиной (на которой заканчивается паттерн)
    int prev;                   //Номер предыдущей вершины (родителя)
    char prevChar;              //Символ, по которому пришли в вершину
    int suffix;                 //Суффиксная ссылка
    std::vector<int> number;    //Какой по счёту считанный паттерн заканчивается на этой вершине (только для вывода)
    int deep;                   //глубина в боре, равная длине строки-паттерна, заканчивающегося в этой терминальной вершине
};

int go(int index, char symb, std::vector<Vertex>& vertexArr);

void addString(const std::string& str, std::vector<Vertex>& vertexArr, int& count)  //Функция добавления строки-паттерна в бор
{
    std::cout << "Adding string \"" << str << "\" in the bohr\n";
```

```
    if (str.empty())
        return;

    int current = 0;
    for (int i = 0; i < str.size(); i++) {
        std::cout << "\tCurrent symbol: \'" << str[i] << "\'\n";
        std::cout << "\tCurrent vertex: " << current << "\n";
        if (vertexArr[current].next.find(str[i]) ==
vertexArr[current].next.end()) {    //Если переход по символу не
обнаружен и
            std::cout << "\tWay through \'" << str[i] << "\' wasn't
found. Adding new vertex with number " << vertexArr.size() << "\n";
            Vertex ver;
//итератор указывает на конец мэпа, то
            ver.suffix = -1;
//создаём новую вершину
            ver.prev = current;
            std::cout << "\t*previous vertex is " << current << ", the
symbol of incoming path \'" << str[i] << "\'\n";
            ver.prevChar = str[i];
            vertexArr.push_back(ver);
            vertexArr[current].next[str[i]] = vertexArr.size() - 1;
//У предыдущей вершины переход в эту
        }
//по текущему символу
        else{
            std::cout << "The way through the symbol exist\n";
        }
        std::cout << std::endl;
        current = vertexArr[current].next[str[i]];
//Переход к следующей вершине
    }

    std::cout << "The number of this pattern is " << count + 1 << "\n";
```

```
    std::cout << "Vertex " << current << " is terminal, deep of the
vertex is " << str.size() << "\n\n";
    vertexArr[current].number.push_back(++count);       //Устанавливаем
номер считанного паттерна,
    vertexArr[current].isTerminal = true;               //Терминальную
вершину
    vertexArr[current].deep = str.size();               //Глубину
}


int getSuffix(int index, std::vector<Vertex>& vertexArr)    //Функция
поиска суффиксной ссылки для вершины index
{
    std::cout << "\t\t\tGetting suffix-link from vertex " << index <<
"\n";
    if (vertexArr[index].suffix == -1) {                    //Если
суффиксная ссылка ещё не была найдена
        if (index == 0 || vertexArr[index].prev == 0) {     //Если
вершина - корень или сын корня
            vertexArr[index].suffix = 0;
            (index == 0) ? std::cout << "\t\t\tThis is root, suffix-link
vertex = 0\n" : std::cout << "\t\t\tThis is a vertex with deep = 1,
suffix-link = 0\n";
        }
        else {                                              //Рекурсивный
поиск суфф. ссылки. Получаем ссылку родителя и выполняем
            std::cout << "\t\t\tFinding suffix-link from suffix of
parent-vertex (" << vertexArr[index].prev << ") through " <<
vertexArr[index].prevChar << "\n";
            vertexArr[index].suffix = go(getSuffix(vertexArr[index].prev,
vertexArr), vertexArr[index].prevChar, vertexArr);
        }                                                   //из неё
переход по символу, по которому попали в вершину, для
    }                                                       //которой и
```

```
ищется суфф. ссылка
    std::cout << "\t\t\tSuffix-link from vertex " << index << " is " <<
vertexArr[index].suffix << "\n\n";
    return vertexArr[index].suffix;
}


int go(int index, char symb, std::vector<Vertex>& vertexArr)
//Функция перехода из вершины index по символу symb. Если прямой переход
{
//невозможен, перейдёт по ссылке
    std::cout << "\t\t\t*Finding the way from " << index << " through \'"
<< symb << "\'\n";
    if (vertexArr[index].go.find(symb) == vertexArr[index].go.end()) {
//Если путь в массиве переходов ещё не был найден
        if (vertexArr[index].next.find(symb) !=
vertexArr[index].next.end()) {    //Если найден прямой переход по символу
в боре
            vertexArr[index].go[symb] = vertexArr[index].next[symb];
//Добавляем в контейнер возможных переходов
        }
        else {
//Если прямого перехода нет, получаем суфф. ссылку
            if (index == 0)
//и ищем переход из суффиксной ссылки по заданному символу
                std::cout << "\t\t\t*This is root\n";
            else
                std::cout << "\t\t\t*No straight path. Finding the way
from suffix-link of this vertex through \'" << symb << "\'\n";
            vertexArr[index].go[symb] = (index == 0 ? 0 :
go(getSuffix(index, vertexArr), symb, vertexArr));
        }
    }
    std::cout << "\t\t\t*Found way from " << index << " through \'" <<
```

```
symb << "\' is " << vertexArr[index].go[symb] << "\n";
    return vertexArr[index].go[symb];
}


void search(const std::string& text, std::vector<Vertex>& vertexArr,
std::vector<int>& res, const std::vector<int>& patternOffsetArr, int
patternLen, const std::vector<std::string>& patternArr)
{
    std::cout << "Searching begin\n";
    int curr = 0;
    for (int i = 0; i < text.size(); i++) {
//Перебираем все символы текста
        std::cout << "\tCurrent symbol is \'" << text[i] <<  "\' from
text...\n";
        std::cout << "\tCurrent vertex is " << curr << "\n";
        curr = go(curr, text[i], vertexArr);
//Осуществляем переход в автомате по считанному символу
        std::cout << "\tAchieved vertex " << curr << "\n";
        std::cout << "\tFinding possible entrance with end suffix-
links:\n";
        for (int tmp = curr; tmp != 0; tmp = getSuffix(tmp, vertexArr)) {
//Сам множественный поиск через суфф. ссылки
            std::cout << "\t\tCurrent suffix-link vertex: " << tmp
<<"\n";
            if (vertexArr[tmp].isTerminal) {
//Если какая-то из них конечная,
                for (int j = 0; j < vertexArr[tmp].number.size(); j++) {
//увеличиваем под символом текста число вхождений паттернов
                    if (i + 1 - patternOffsetArr[vertexArr[tmp].number[j]
- 1] - vertexArr[tmp].deep >= 0 &&
                        i + 1 - patternOffsetArr[vertexArr[tmp].number[j]
- 1] - vertexArr[tmp].deep <= text.size() - patternLen){
                        res[i + 1 -
```

```
patternOffsetArr[vertexArr[tmp].number[j] - 1] - vertexArr[tmp].deep]++;
                        std::cout << "\t\tThe vertex is terminal (end
suffix-link). The entrance found, index = " <<
                            i + 1 -
patternOffsetArr[vertexArr[tmp].number[j] - 1] - vertexArr[tmp].deep << "
(pattern = \"" << patternArr[vertexArr[tmp].number[j] - 1] <<
                            "\"). Count of entrance is " << res[i + 1 -
patternOffsetArr[vertexArr[tmp].number[j] - 1] - vertexArr[tmp].deep] <<
                            " from " << patternOffsetArr.size() << "
possible\n\n";
                    }
                }
            } else
                std::cout << "\t\tIt's not terminal vertex, getting
suffix-link from this vertex\n\n";
        }
        std::cout << "\t\tRoot is arrived, reading new symbol from the
text\n";
        std::cout << "\t-------------------------------------------------
--------------------------------\n";
        std::cout << "\t-------------------------------------------------
--------------------------------\n";
    }
    std::cout << "-------------------------------------------------------
------------------------------\n";
    std::cout << "-------------------------------------------------------
------------------------------\n";
}


void printRes(const std::vector<int>& res, int patternCount, std::string&
textRest, int patternLen, const std::string& text)
{
    std::cout << "Total indexes of entrance (beginning from 1):\n";
```

```
    std::vector<bool> cutStr(text.size());          //Индексы символов в
строке, которые будут вырезаны

    for (int i = 0; i < res.size(); i++) {
        if (res[i] == patternCount) {                 //Если под текущим
символом текста совпали все паттерны,
            std::cout << i + 1 << "\n";             //то вхождение
найдено
            for (int j = 0; j < patternLen; j++)    //Перебираем все
символы строки, образующие паттерн
                cutStr[i + j] = true;               //Помечаем индексы
символов в строке, подлежащие удалению
        }
    }

    for (int i = 0; i < cutStr.size(); i++){
        if (!cutStr[i])
            textRest.push_back(text[i]);            //Сохраняем только
неудалённые символы
    }
}

//Функция разбивает строку-паттерн с джокерами на массив строк-паттернов
без них и запоминает их индексы в первоначальной строке
void split(std::string str, char joker, std::vector<std::string>&
patternArr, std::vector<int>& patternOffsetArr){
    std::cout << "Begin splitting\n";
    std::string buf = "";
    for (int i=0; i<str.size(); i++){
        if (str[i] == joker){
            if (buf.size() > 0) {                   //Пропускаем пустые
строки (если джокеры идут подряд)
                patternArr.push_back(buf);            //Сохраняем паттерн
                std::cout << "\tWas found new pattern: " << buf << "\n";
```

```
                patternOffsetArr.push_back(i - buf.size());   //и его
индекс вхождения в строку с джокерами
                std::cout << "\tIndex of entrance in total pattern: " <<
i - buf.size() << "\n";
                buf = "";
            }
        }
        else {
            buf.push_back(str[i]);                  //Формируем строку-
паттерна без джокеров
            if (i == str.size() - 1){               //Если достигнут
конец паттерна
                patternArr.push_back(buf);            //Сохраняем
последний полученный паттерн без джокера
                std::cout << "\tWas found new pattern: " << buf << "\n";
                patternOffsetArr.push_back(i - buf.size() + 1);
                std::cout << "\tIndex of entrance in total pattern: " <<
i - buf.size() + 1 << "\n";
            }
        }
    }
}

void readPattern(std::vector<Vertex>& vertexArr, char& joker,
std::vector<int>& patternOffsetArr, int& patternLen,
std::vector<std::string>& patternArr)
{
    Vertex root;                                //Инициализация корня
    root.prev = -1;
    root.suffix = -1;
    vertexArr.push_back(root);
    int count = 0;

    std::cout << "Enter pattern:\n";
```

```
    std::string patternStr;                                    //Строка-
паттерн
    std::cin >> patternStr;
    std::cout << "Enter joker:\n";
    std::cin >> joker;
    patternLen = patternStr.size();                            //Длина
паттерна
    std::cout << "-------------------------------------------------------
------------------------------\n";

    split(patternStr, joker, patternArr, patternOffsetArr);
    std::cout << "-------------------------------------------------------
------------------------------\n";
    std::cout << "-------------------------------------------------------
------------------------------\n";
    std::cout << "Begin bohr building\n";
    for (auto pattern : patternArr) {
        addString(pattern, vertexArr, count);    //Формируем бор
    }
    std::cout << "-------------------------------------------------------
------------------------------\n";
    std::cout << "-------------------------------------------------------
------------------------------\n";
}

//Функция поиска максимального числа исходящих дуг из одной вершины бора
int findMaxSons(std::vector<Vertex> vertexArr)
{
    int max = vertexArr[0].next.size();

    for(int i = 1; i < vertexArr.size(); i++){
        if (vertexArr[i].next.size() > max)
            max = vertexArr[i].next.size();
    }
```

```
    return max;
}

void automatePrint(std::vector <Vertex> vertexArr)
{
    std::cout << "-------------------------------------------------------
------------------------------\n";
    std::cout << "Total automate:\n";

    for (int i = 0; i < vertexArr.size(); i++){
        std::cout << "Connections from vertex " << i << ":\n";
        auto iter = vertexArr[i].go.begin();
        for (int j = 0; j < vertexArr[i].go.size(); j++){
            std::cout << "\t" << i << "  --" << iter->first << "->  " <<
iter->second << "\n";
            iter++;
        }
    }
}

int main() {
    std::cout << "-------------------------------------------------------
------------------------------\n";
    std::cout << "Enter text:\n";
    std::string text, textRest;
    std::cin >> text;

    std::vector<Vertex> vertexArr;      //Массив вершин
    std::vector<std::string> patternArr;
    std::vector<int> res(110000);    //Массив числа совпадений паттернов
под каждым символом строки
    std::vector<int> patternOffsetArr;
    int patternLen;                     //Длина паттерна
```

```
    for (int i = 0; i < 110000; i++){
        res[i] = 0;
    }

    char joker;

    readPattern(vertexArr, joker, patternOffsetArr, patternLen,
patternArr);
    search(text, vertexArr, res, patternOffsetArr, patternLen,
patternArr);
    printRes(res, patternArr.size(), textRest, patternLen, text);

    std::cout << "Rest string from text after cutting patterns from it: "
<< textRest << "\n";

    int maxSonsCount = findMaxSons(vertexArr);
    std::cout << "Max count of sons: " << maxSonsCount << "\n\n";

    automatePrint(vertexArr);

    return 0;
}
```