

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

Нам Ё Себ

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

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 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:

СССА

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:

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

, 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:

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

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 тар, 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> jump;  
    bool is_terminal = false;  
    int prev;  
    char prev_char;  
    int suffix;  
    int number;  
    int deep;  
};
```

Структура представления вершины бора и автомата.

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

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

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

`prev` - номер предыдущей вершины (родителя)

`prev_char` - 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>&  
arr_vertex, int& count)
```

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

3.

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

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

4.

```
int jump(int index, char symb, std::vector<Vertex>& arr_vertex)
```

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>& arr_vertex, std::vector<std::pair<int, int>>& res, const std::vector<std::string>& arr_pattern)
```

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

6.

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

Функция вывода ответа. Выводит индекс вхождения и номер паттерна, вырезает из строки-текста паттерны. `res` – массив пар <индекс вхождения,

номер паттерна>, arr_pattern – массив паттернов, text – строка-текст, cutted_text – остаточная строка.

7.

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

Функция считывания паттернов. arr_vertex – массив вершин бора, arr_pattern – массив паттернов, count – число строк в боре.

8.

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

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

9.

```
void printAutomate(std::vector <Vertex> arr_vertex)
```

Функция вывода полученного автомата. arr_vertex – 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

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 $(T, 1 \leq |T| \leq 100000)$

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:

АСТ

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:

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

, 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:

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

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 тар, 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> jump;
    bool is_terminal = false;
    int prev;
    char prev_char;
    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`

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

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

`prev` - номер предыдущей вершины (родителя)

`prev_char` - символ, по которому пришли в вершину

`suffix` - суффиксная ссылка

`number` - какой по счёту считанный паттерн заканчивается на этой вершине

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

2.

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

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

3.

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

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

4.

```
int jump(int index, char symb, std::vector<Vertex>& arr_vertex)
```

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>&
arr_vertex, std::vector<int>& res, const std::vector<int>&
pattern_offset_arr, int pattern_len, const std::vector<std::string>&
arr_pattern)
```

Функция поиска вхождений паттернов. `text` – введенная строка-текст для поиска в ней, `res` – массив чисел найденных паттернов под индексом строки, `arr_pattern` – массив паттернов, `arr_vertex` – массив вершин бора, `pattern_offset_arr` – массив смещений подстрок-паттернов в исходной строке-паттерне с джокерами, `pattern_len` – длина исходного паттерна с джокерами.

6.

```
void printRes(const std::vector<int>& res, int pattern_cnt,
std::string& cutted_text, int pattern_len, const std::string& text)
```

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

7.

```
void readPattern(std::vector<Vertex>& arr_vertex, char& joker,
std::vector<int>& pattern_offset_arr, int& pattern_len,
std::vector<std::string>& arr_pattern)
```

Функция считывания паттернов. `arr_vertex` – массив вершин бора, `joker` – символ джокера, `arr_pattern` – массив паттернов, `pattern_offset_arr` – массив смещений подстрок-паттернов в исходной строке-паттерне с джокерами, `pattern_len` – длина исходного паттерна с джокерами.

8.

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

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

9.

```
void printAutomate(std::vector<Vertex> arr_vertex)
```

Функция вывода полученного автомата. `arr_vertex` – массив вершин бора.

10.

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

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, `arr_pattern` – массив паттернов, `pattern_offset_arr` – массив смещений подстрок-паттернов в исходной строке-паттерне с джокерами.

Тестирование.

1.

```
-----
Enter text:
asffaasssafaf
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 entrance 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 entrance 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: asasssf
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

```

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-Карассик 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>

#define LONGLINE "-----\n"

struct Vertex {
    std::map<char, int> next;    //Контейнер переходов по символу char в
    //вершину с номером int
    std::map<char, int> jump;    //массив переходов (запоминаем переходы в
    //ленивой рекурсии), используемый для вычисления суффиксных ссылок
    bool is_terminal = false;    //Является ли терминальной (конечной)
    //вершиной (на которой заканчивается паттерн)
    int prev;                    //Номер предыдущей вершины (родителя)
    char prev_char;              //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 jump(int index, char symb, std::vector<Vertex>& arr_vertex);

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

    for (int i = 0; i < str.size(); i++)
    {
        std::cout << "\tCurrent symbol: '" << str[i] << "'\n";
        std::cout << "\tCurrent vertex: " << current << std::endl;
        if (arr_vertex[current].next.find(str[i]) ==
arr_vertex[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 " << arr_vertex.size() << std::endl;
            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.prev_char = str[i];
            arr_vertex.push_back(ver);
            arr_vertex[current].next[str[i]] = arr_vertex.size() - 1;
        }
        else
            std::cout << "The way through the symbol exist\n";

        std::cout << std::endl;
    }
}
```

```

        current = arr_vertex[current].next[str[i]];          //Переход к
следующей вершине
    }

    std::cout << "The number of this pattern is " << count + 1 << std::endl;
    std::cout << "Vertex " << current << " is terminal, deep of the vertex is
" << str.size() << "\n\n";

    arr_vertex[current].number = ++count;                  //Устанавливаем номер
считанного паттерна,
    arr_vertex[current].is_terminal = true;                //Терминальную
вершину
    arr_vertex[current].deep = str.size();                  //Глубину
}

int getSuffix(int index, std::vector<Vertex>& arr_vertex)    //Функция поиска
суффиксной ссылки для вершины index
{
    std::cout << "\t\t\tGetting suffix-link from vertex " << index <<
std::endl;
    if (arr_vertex[index].suffix == -1)                    //Если суффиксная
ссылка ещё не была найдена
    {
        if (index == 0 || arr_vertex[index].prev == 0)    //Если вершина -
корень или сын корня
        {
            arr_vertex[index].suffix = 0;
            (index == 0) ? std::cout << "\t\t\tThis is root, suffix-link
vertex = 0" << std::endl : std::cout << "\t\t\tThis is a vertex with deep = 1,
suffix-link = 0" << std::endl;
        }
        else                                                //Рекурсивный
поиск суфф. ссылки. Получаем ссылку родителя и выполняем
        {
            std::cout << "\t\t\tFinding suffix-link from suffix of parent-
vertex (" << arr_vertex[index].prev << ") through " <<
arr_vertex[index].prev_char << std::endl;
            arr_vertex[index].suffix =
jump(getSuffix(arr_vertex[index].prev, arr_vertex), arr_vertex[index].prev_char,
arr_vertex);
        }                                                    //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 " <<
arr_vertex[index].suffix << "\n\n";
        return arr_vertex[index].suffix;
    }
}

int jump(int index, char symb, std::vector<Vertex>& arr_vertex) { //Функция
перехода из вершины index по символу symb. Если прямой переход
    std::cout << "\t\t\t*Finding the way from " << index << " through \' " <<
symb << "\'\n"; //невозможен, перейдёт по ссылке

    if (arr_vertex[index].jump.find(symb) == arr_vertex[index].jump.end())
//Если путь ещё не был найден
    {
        if (arr_vertex[index].next.find(symb) !=
arr_vertex[index].next.end()) //Если найден прямой переход по символу в боре

```

```

        arr_vertex[index].jump[symb] = arr_vertex[index].next[symb];
//Добавляем в контейнер возможных переходов
    else
//Если прямого перехода нет, получаем суфф. ссылку
    {
        //и ищем переход из суффиксной ссылки
        по заданному символу
        if (index == 0)
            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";

        arr_vertex[index].jump[symb] = (index == 0 ? 0 :
jump(getSuffix(index, arr_vertex), symb, arr_vertex));
    }
}

std::cout << "\t\t\t*Found way from " << index << " through \' " << symb <<
"\' is " << arr_vertex[index].jump[symb] << std::endl;
return arr_vertex[index].jump[symb];
}

void search(const std::string& text, std::vector<Vertex>& arr_vertex,
std::vector<std::pair<int, int>>& res, const std::vector<std::string>&
arr_pattern)
{
    std::cout << "Searching begin" << std::endl;
    int curr = 0;

    for (int i = 0; i < text.size(); i++)
    {
        std::cout << "\tCurrent symbol is \' " << text[i] << "\' from
text..." << std::endl;
        std::cout << "\tCurrent vertex is " << curr << std::endl;
        curr = jump(curr, text[i], arr_vertex);
        std::cout << "\tAchieved vertex " << curr << std::endl;
        std::cout << "\tFinding possible entrance with end suffix-links:" <<
std::endl;

        for (int tmp = curr; tmp != 0; tmp = getSuffix(tmp, arr_vertex))
        {
            std::cout << "\t\tCurrent suffix-link vertex: " << tmp <<
std::endl;
            if (arr_vertex[tmp].is_terminal)
            {
                res.push_back(std::make_pair(i + 2 -
arr_vertex[tmp].deep, arr_vertex[tmp].number));
                std::cout << "\t\tThe vertex is terminal (end suffix-
link). The entrance found, index = " <<
i + 2 - arr_vertex[tmp].deep << " (pattern = \' "
<< arr_pattern[arr_vertex[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"
<< std::endl;
        std::cout << "\t" << LONGLINE;
        std::cout << "\t" << LONGLINE;
    }
}

```

```

    }

    std::cout << LONGLINE;
    std::cout << LONGLINE;
}

void printRes(const std::vector<std::pair<int, int>>& res, const
std::vector<std::string>& arr_pattern, const std::string& text, std::string&
cutted_text)
{
    std::vector<bool> cut_str(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 << std::endl;
        for (int j = 0; j < arr_pattern[res[i].second - 1].size(); j++)
            cut_str[res[i].first - 1 + j] = true;
    }

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

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

    int pattern_num;
    std::cout << "Enter pattern count:" << std::endl;
    std::cin >> pattern_num;

    for (int i = 0; i < pattern_num; i++)
    {
        std::cout << "Enter pattern for searching in:" << std::endl;
        std::string pattern;
        std::cin >> pattern;
        arr_pattern.push_back(pattern);
        addString(pattern, arr_vertex, count);
    }
}

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

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

```

```

}

void printAutomate(std::vector<Vertex> arr_vertex)
{
    std::cout << LONGLINE;
    std::cout << "Total automate:" << std::endl;

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

int main() {
    std::cout << LONGLINE;
    std::cout << "Enter text:" << std::endl;
    std::string text, cutted_text;
    std::cin >> text;

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

    readPattern(arr_vertex, count, arr_pattern);
    search(text, arr_vertex, res, arr_pattern);
    printRes(res, arr_pattern, text, cutted_text);

    std::cout << "Rest string from text after cutting patterns from it: " <<
cutted_text << std::endl;

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

    printAutomate(arr_vertex);

    return 0;
}

```

Алгоритм Ахо-Корасик с джокером

```

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

```

```

#define LONGLINE "-----\n"

```

```

struct Vertex {
    std::map<char, int> next;    //Контейнер переходов по символу char в
    вершину с номером int
    std::map<char, int> jump;    //массив переходов (запоминаем переходы в
    ленивой рекурсии), используемый для вычисления суффиксных ссылок
    bool is_terminal = false;    //Является ли терминальной (конечной)
    вершиной (на которой заканчивается паттерн)
    int prev;                    //Номер предыдущей вершины (родителя)
    char prev_char;              //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 deer;                    //глубина в боре, равная длине строки-
    паттерна, заканчивающегося в этой терминальной вершине
};

int jump(int index, char symb, std::vector<Vertex>& arr_vertex);

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

        if (arr_vertex[current].next.find(str[i]) ==
arr_vertex[current].next.end())    //Если переход по символу не обнаружен и
        {
            std::cout << "\tWay through \"" << str[i] << "\" wasn't found.
Adding new vertex with number " << arr_vertex.size() << std::endl;
            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.prev_char = str[i];
        arr_vertex.push_back(ver);
        arr_vertex[current].next[str[i]] = arr_vertex.size() - 1;
//У предыдущей вершины переход в эту
    }
//по текущему символу
    else
        std::cout << "The way through the symbol exist" << std::endl;

        std::cout << std::endl;
        current = arr_vertex[current].next[str[i]];
//Переход к следующей вершине
    }
    std::cout << "The number of this pattern is " << count + 1 << std::endl;
    std::cout << "Vertex " << current << " is terminal, deep of the vertex is
" << str.size() << "\n\n";

    arr_vertex[current].number.push_back(++count);           //Устанавливаем номер
считанного паттерна,
    arr_vertex[current].is_terminal = true;                   //Терминальную
вершину
    arr_vertex[current].deep = str.size();                     //Глубину
}

int getSuffix(int index, std::vector<Vertex>& arr_vertex) {    //Функция поиска
суффиксной ссылки для вершины index
    std::cout << "\t\t\tGetting suffix-link from vertex " << index <<
std::endl;
    if (arr_vertex[index].suffix == -1)                       //Если суффиксная
ссылка ещё не была найдена
    {
        if (index == 0 || arr_vertex[index].prev == 0)       //Если вершина -
корень или сын корня
        {
            arr_vertex[index].suffix = 0;
            (index == 0) ? std::cout << "\t\t\tThis is root, suffix-link
vertex = 0" << std::endl : std::cout << "\t\t\tThis is a vertex with deep = 1,
suffix-link = 0" << std::endl;
        }
        else                                                    //Рекурсивный
поиск суфф. ссылки. Получаем ссылку родителя и выполняем

```



```

        {
            std::cout << "\t\t\tFinding suffix-link from suffix of parent-
vertex (" << arr_vertex[index].prev << ") through " <<
arr_vertex[index].prev_char << std::endl;
            arr_vertex[index].suffix =
jump(getSuffix(arr_vertex[index].prev, arr_vertex), arr_vertex[index].prev_char,
arr_vertex);
        } //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 " <<
arr_vertex[index].suffix << "\n\n";
    return arr_vertex[index].suffix;
}

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

            arr_vertex[index].jump[symb] = (index == 0 ? 0 :
jump(getSuffix(index, arr_vertex), symb, arr_vertex));
        }
    }
}

```

```

    }
    std::cout << "\t\t\t*Found way from " << index << " through \'' << symb <<
    "\' is " << arr_vertex[index].jump[symb] << std::endl;
    return arr_vertex[index].jump[symb];
}

void search(const std::string& text, std::vector<Vertex>& arr_vertex,
std::vector<int>& res, const std::vector<int>& pattern_offset_arr, int
pattern_len, const std::vector<std::string>& arr_pattern){
    std::cout << "Searching begin" << std::endl;
    int curr = 0;

    for (int i = 0; i < text.size(); i++)
//Перебираем все символы текста
    {
        std::cout << "\tCurrent symbol is \'' << text[i] << "\' from
text..." << std::endl;
        std::cout << "\tCurrent vertex is " << curr << std::endl;
        curr = jump(curr, text[i], arr_vertex);
//Осуществляем переход в автомате по считанному символу
        std::cout << "\tAchieved vertex " << curr << std::endl;
        std::cout << "\tFinding possible entrance with end suffix-links:\n";

        for (int tmp = curr; tmp != 0; tmp = getSuffix(tmp, arr_vertex))
//Сам множественный поиск через суфф. ссылки
        {
            std::cout << "\t\tCurrent suffix-link vertex: " << tmp <<
std::endl;

            if (arr_vertex[tmp].is_terminal)
//Если какая-то из них конечная,
            {
                //увеличиваем под символом текста число
                вхождений паттернов
                for (int j = 0; j < arr_vertex[tmp].number.size(); j++)
                {
                    if (i + 1 -
pattern_offset_arr[arr_vertex[tmp].number[j] - 1] - arr_vertex[tmp].deep >= 0 &&
i + 1 -
pattern_offset_arr[arr_vertex[tmp].number[j] - 1] - arr_vertex[tmp].deep <=
text.size() - pattern_len)
                    {

```

```

        res[i + 1 -
pattern_offset_arr[arr_vertex[tmp].number[j] - 1] - arr_vertex[tmp].deep]++;
        std::cout << "\t\tThe vertex is terminal
(end suffix-link). The entrance found, index = " <<
        i + 1 -
pattern_offset_arr[arr_vertex[tmp].number[j] - 1] - arr_vertex[tmp].deep << "
(pattern = \"\" << arr_pattern[arr_vertex[tmp].number[j] - 1] <<
        "\"). Count of entrance is " << res[i
+ 1 - pattern_offset_arr[arr_vertex[tmp].number[j] - 1] - arr_vertex[tmp].deep]
<<
        " from " << pattern_offset_arr.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"
<< std::endl;
    std::cout << "\t" << LONGLINE;
    std::cout << "\t" << LONGLINE;
}
std::cout << LONGLINE;
std::cout << LONGLINE;
}

```

```

void printRes(const std::vector<int>& res, int pattern_cnt, std::string&
cutted_text, int pattern_len, const std::string& text)
{
    std::cout << "Total indexes of entrance (beginning from 1):" << std::endl;
    std::vector<bool> cut_str(text.size()); //Индексы символов в
строке, которые будут вырезаны

    for (int i = 0; i < res.size(); i++)
        if (res[i] == pattern_cnt) //Если под текущим
символом текста совпали все паттерны,
        {
            //то вхождение найдено
            std::cout << i + 1 << std::endl;
        }
    }
}

```

```

        for (int j = 0; j < pattern_len; j++)    //Перебираем все
символы строки, образующие паттерн
            cut_str[i + j] = true;                //Помечаем индексы
символов в строке, подлежащие удалению
        }

        for (int i = 0; i < cut_str.size(); i++)
            if (!cut_str[i])
                cutted_text.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>& arr_pattern,
std::vector<int>& pattern_offset_arr) {
    std::cout << "Begin splitting" << std::endl;
    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)
            {
                arr_pattern.push_back(buf);        //Сохраняем
паттерн

                std::cout << "\tWas found new pattern: " << buf <<
std::endl;

                pattern_offset_arr.push_back(i - buf.size());    //и его
индекс вхождения в строку с джокерами
                std::cout << "\tIndex of entrance in total pattern: " <<
i - buf.size() << std::endl;
                buf = "";
            }
        }
        else
        {
            buf.push_back(str[i]);                //Формируем строку-
паттерна без джокеров

```

```

        if (i == str.size() - 1)                                //Если достигнут конец
паттерна
    {
        arr_pattern.push_back(buf);                            //Сохраняем
последний полученный паттерн без джокера
        std::cout << "\tWas found new pattern: " << buf <<
std::endl;
        pattern_offset_arr.push_back(i - buf.size() + 1);
        std::cout << "\tIndex of entrance in total pattern: " <<
i - buf.size() + 1 << std::endl;
    }
}
}

void readPattern(std::vector<Vertex>& arr_vertex, char& joker, std::vector<int>&
pattern_offset_arr, int& pattern_len, std::vector<std::string>& arr_pattern) {
    Vertex root;                                                //Инициализация корня
    root.prev = -1;
    root.suffix = -1;
    arr_vertex.push_back(root);
    int count = 0;

    std::cout << "Enter pattern:" << std::endl;
    std::string pattern_str;                                    //Строка-
паттерн
    std::cin >> pattern_str;
    std::cout << "Enter joker:" << std::endl;
    std::cin >> joker;
    pattern_len = pattern_str.size();                            //Длина
паттерна
    std::cout << LONGLINE;

    split(pattern_str, joker, arr_pattern, pattern_offset_arr);
    std::cout << LONGLINE;
    std::cout << LONGLINE;
    std::cout << "Begin bohr building" << std::endl;

    for (auto pattern : arr_pattern)
        addString(pattern, arr_vertex, count);                //Формируем бор

    std::cout << LONGLINE;

```

```

        std::cout << LONGLINE;
    }

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

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

void printAutomate(std::vector <Vertex> arr_vertex) {
    std::cout << LONGLINE;
    std::cout << "Total automate:" << std::endl;

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

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

int main() {
    std::cout << LONGLINE;
    std::cout << "Enter text:" << std::endl;
    std::string text, cutted_text;
    std::cin >> text;

    std::vector<Vertex> arr_vertex;          //Массив вершин
    std::vector<std::string> arr_pattern;

```

```

        std::vector<int> res(110000);    //Массив числа совпадений паттернов под
каждым символом строки
        std::vector<int> pattern_offset_arr;
        int pattern_len;                //Длина паттерна

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

        char joker;

        readPattern(arr_vertex, joker, pattern_offset_arr, pattern_len,
arr_pattern);
        search(text, arr_vertex, res, pattern_offset_arr, pattern_len,
arr_pattern);
        printRes(res, arr_pattern.size(), cutted_text, pattern_len, text);

        std::cout << "Rest string from text after cutting patterns from it: " <<
cutted_text << std::endl;

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

        printAutomate(arr_vertex);

        return 0;
}

```