

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №5
по дисциплине «ПОСТРОЕНИЕ И АНАЛИЗ
АЛГОРИТМОВ»
Тема: Алгоритм Ахо-Корасик
Вариант 5

Студент гр. 8382

Торосян Т.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить и применить на практике алгоритм Ахо-Корасик множественного поиска вхождений строковых паттернов в тексте, в том числе с учетом символа-«джокера».

Алгоритм Ахо-Корасик на обычном тексте

Постановка задачи

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст T ($1 \leq |T| \leq 100000$)

Вторая — число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P = p_1, \dots, p_n$ $1 \leq |p_i| \leq 75$

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

Выход:

Все вхождения образцов из P в T .

Каждое вхождение образца в текст представить в виде двух чисел — i и p

Где i — позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p (нумерация с 1).

Строки вывода должны быть отсортированы по возрастанию, сначала номер позиции, затем номер шаблона.

Индивидуализация

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

Описание алгоритма

Получив строки-паттерны, алгоритм строит бор по следующим

правилам: корнем бора является вершина, из которой строятся переходы к вершинам нижнего уровня (уровень корневой вершины - 0). Каждый переход соответствует своему символу. Каждая добавленная строка увеличивает граф. Если нужного перехода из текущей вершины нет, создается новая вершина, иначе следующей текущей вершиной становится инцидентная нынешней текущей. Вершина, соответствующая последнему переходу, объявляется терминальной.

Затем алгоритм начинает посимвольное считывание текста, выполняя переходы по символу в боре при помощи отдельной функции. Если есть прямой переход из текущей вершины, эта функция возвращает ту вершину, в которую выполнен переход. Иначе выполняется переход по суффиксной ссылке и уже из другой вершины функция повторяет попытку перехода. При достижении корня, он же и возвращается.

После выполненного перехода для текущей вершины и рекурсивно для ее суффиксных ссылок выполняется проверка на терминальность. Если она успешна — перед нами вхождение паттерна в текст. После этого считывается следующий символ текста и начинается следующая итерация.

Алгоритм завершает работу когда пройдены все символы текста.

Функция, ищущая суффиксную ссылку работает по следующему принципу: если для заданной вершины суффиксная ссылка ещё не была найдена, то функция проверяет, является ли вершина корнем или вершиной 1-го уровня. Если это так, то суфф. Ссылка этой вершины — корень, иначе функция выполняет рекурсивный поиск: ищет суффиксную ссылку у вершины-родителя и в ней осуществляет переход по символу, по которому был выполнен переход в исходную вершину.

Описание функций

- *struct Vertex*

{

std::map<char, int> next;

```

std::map<char, int> next;

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 (запоминание переходов в ленивой рекурсии), используемый для вычисления суффиксных ссылок.

IsTerminal – флаг терминальности данной вершины.

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

- *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.

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

Функция поиска суффиксной ссылки для вершины. `index` – номер

вершины в бору, для которой ищем ссылку. `vertexArr` – массив вершин бора. Возвращает номер вершины, на которую указывает суффиксная ссылка

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

Функция перехода из вершины по символу. `index` – номер вершины в

бору, из которой ищем путь. `symb` – символ, по которому нужно найти переход. Возвращает номер достигнутой вершины.

- *`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.

- *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.

- *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.

- *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.

- *void automatePrint(std::vector<Vertex> vertexArr)*

Функция вывода полученного автомата. vertexArr – массив вершин бора.

Сложность алгоритма по времени

Алгоритм строит бор за $|P|\log(|E|)$, где $|P|$ - сумма длин всех паттернов, $|E|$ - мощность алфавита, т.к. для вставки новых строк в бор нужно выполнить $|P|$ операций поиска по ключу и добавления пары в map, которые занимают $|T|\log(|E|)$ операций, где $|T|$ - длина текста, поскольку по каждому символу снова нужно находить значение по ключу. В процессе обхода бора будут совершаться переходы по конечным ссылкам, число таких переходов максимально равно общему числу совпадений всех паттернов с текстом (t).

Отсюда получаем $O((|P|+|T|)*\log(|E|) + t)$

Сложность алгоритма по памяти

.Сложность по памяти составляет $O(|P|+|T|)$, где $|P|$ в худшем случае равно числу вершин в боре, которые нужно хранить, а $|T|$ - длину текста.

Тестирование

```
-----
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
    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
  -----

```


through 'f'

(pattern = "dfs")

```
-----
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

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
        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'

through 's'

*No straight path. Finding the way from suffix-link of this vertex

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'

through 's'

*No straight path. Finding the way from suffix-link of this vertex

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

Process returned 0 (0x0)   execution time : 50.957 s
Press any key to continue.

```

№ теста	Ввод	Вывод
1	SHE_BELIEVED 3 HE LIE LIVE	2 1 7 2
2	KOLOKOL 4 LOL OLO KOL OK	1 3 2 2 4 4 5 3
3	HISHER 6 HIS HER IS I HE SHE	1 1 2 3 2 4 3 6 4 2 4 5

Алгоритм Ахо-Корасик с джокером.

Постановка задачи

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 Р необходимо найти все вхождения Р в текст Т.

Например, образец $ab??c?$ с джокером $?$ встречается дважды в тексте $xabvccbababcsax$.

Символ джокер не входит в алфавит, символы которого используются в Т. Каждый джокер соответствует одному символу, а не подстроке неопределенной длины. В шаблоне входит хотя бы один символ не джокер, т.е. шаблоны вида $???$ недопустимы.

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

Ввод:

Текст ($T, 1 \leq |T| \leq 100000$)

Шаблон ($P, 1 \leq |P| \leq 40$)

Символ джокера

Вывод:

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

Номера должны выводиться в порядке возрастания.

Описание алгоритма.

Изначально алгоритм инициализирует массив для записи результата, равный тексту по длине. Алгоритм принимает строку-паттерн с джокером и делит её по джокерам на подстроки-паттерны, запоминая их индексы в исходной строке. Затем с помощью алгоритма Ахо-Корасик находятся все вхождения данных подстрок. При нахождении очередной подстроки вычисляется индекс в позиции массива. Если индекс попал в заданный диапазон массива, то счетчик в массиве увеличивается на единицу. Затем производится поиск на совпадение полей массива с итоговым количеством подстрок. Если числа совпадают, то в тексте с данного индекса начинается искомая строка с джокерами.

Каждую подстроку-паттерн алгоритм заносит в 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.

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.

Сложность алгоритма по времени

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.

- *struct Vertex*

```
{  
  
    std::map<char, int> next;  
  
    std::map<char, int> next;  
  
    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 (запоминание переходов в ленивой рекурсии), используемый для вычисления суффиксных ссылок.

IsTerminal – флаг терминальности данной вершины.

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

- *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.

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

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

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

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

- *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 – введенная строка-текст для поиска в ней, res – массив пар <индекс вхождения, номер паттерна>, patternArr

– массив паттернов, vertexArr – массив вершин бора.

- `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.

- `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.

- `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.

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

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

- `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

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 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: 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

Process returned 0 (0x0)   execution time : 88.728 s
Press any key to continue.

```

№ теста	Ввод	Вывод
1	SMALLDUCKMAKESQUACK D*CK *	6
2	Bananabanhammer \$ban \$	6
3	PunPun_pins_a_pen_to_the_p an <u>p@n</u> @	8 15 26

Вывод.

В ходе выполнения работы были реализованы практические приложения алгоритма Ахо-Корасик, в том числе, с символом-«джокером» для выполнения множественного поиска вхождений в тексте. Также был реализован функционал отображения максимального числа исходящих дуг для вершины в боре, остаточного текста после удаления вхождений и результирующего конечного автомата.

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл task1.cpp

```
#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 мпа
            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;
}

```

Файл task2.cpp

```

#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 deer; //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 не удалённые
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;
}

```