

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. 8304

Щука А. А.

Преподаватель

Размочаева Н. В.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритм Ахо-Корасик и алгоритм поиска вхождений шаблонов с “джокерами” в строку. Написать программу, реализующую эти алгоритмы работы со строками.

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.

Пример входных данных

СССА

1

СС

Пример выходных данных

1 1

2 1

Описание алгоритма.

В начале алгоритма бор заполняется символами шаблонов. Для этого поочередно обрабатывается каждый символ шаблона. Если перехода в боре для текущей вершины нет, то вершина создается, добавляется в бор и в нее совершается переход по текущему символу. Если вершина с переходом по текущему символу уже существует, то в нее совершается переход.

Далее осуществляется поиск шаблонов в текстовой строке. Для этого обрабатывается автомат, полученный из созданного бора путем добавления суффиксных ссылок.

Обрабатывается текущий символ текстовой строки. Если в автомате уже существует ребро-переход по символу в вершину, то осуществляется переход в эту вершину. Если ребра-перехода в автомате еще нет, но существует переход по текущему символу в вершину-сына, то этот переход осуществляется и добавляется в ребра автомата. Если такого перехода также не существует, то переход осуществляется по суффиксной ссылке и также заносится в ребра автомата.

Для нахождения суффиксной ссылки для вершины, осуществляется переход в предка вершины, затем переход по суффиксной ссылке предка и переход по текущему символу. Если предок не имеет суффиксной ссылки, то для него она определяется аналогичным образом рекурсивно.

Если во время перехода в автомате встречается терминальная вершина, это означает, что шаблон в подстроке найден. Вычисляется индекс его в строке и заносится в вектор результата.

Для вывода максимального числа дуг, исходящих из одной вершины бора перебираются вершины-дети бора. Если число дуг для текущей вершины

больше переменной, хранящей это максимальное число, то в переменную заносится это новое значение. Результатом является значение, хранящееся в этой переменной.

Для вывода строки, из которой были удалены найденные шаблоны заводится булевский вектор. Индексы, соответствующие индексам с символами шаблона в строке, помечаются. Строка формируется путем добавления в нее символов, индексы которых не были помечены.

Сложность алгоритма по операциям:

Таблица переходов автомата хранится в структуре `std::map`, которая реализована как красно-черное дерево. Тогда сложность алгоритма по операциям будет равна $O((M+N)*\log(k)+t)$, M – длина всех символов слов шаблонов, N – длина текста, в котором осуществляется поиск, k – размер алфавита, t – длина всех возможных вхождений всех строк-образцов.

Сложность алгоритма по памяти: $O(M+N)$, M – длина всех символов слов шаблонов, N – длина текста, в котором осуществляется поиск.

Описание функций и структур данных.

Структура вершины

struct Vertex

```
{
    std::map<char, int> next; - переходы по ребрам в боре
    std::map<char, int> go;  - переходы в автомате
    int prev = 0;           - индекс вершины-предка
    char prevChar = 0;      - символ перехода в вершину из родительской
    int suffix = -1;        - индекс вершины в которую можно попасть по
суффиксной ссылке
    int number = 0;         - номер шаблона терминальной вершины
    int deep = 0;           - глубина вершины
}
```

```
bool isLeaf = false;    - является ли вершина терминальной  
};
```

void addPattern(const std::string& str)

Функция добавления символов шаблона в бор

str – шаблон для добавления в бор

void search(const std::string& str)

Функция поиска шаблонов в строке

str – текст, в котором будет осуществляться поиск

void printResult(const std::string& text) const

Функция вывода результата работы алгоритма и строки, из которой были удалены найденные шаблоны.

text – текст, в котором осуществляется поиск шаблонов.

void printMaxArcs() const

Функция, вычисляющая максимальное количество дуг, исходящих из одной вершины в боре

int getSuffix(int index)

Функция получения вершины, доступной по суффиксной ссылке.

index – индекс вершины, для которой осуществляется поиск по суффиксной ссылке.

Возвращаемым значением является индекс вершины, доступной по суффиксной ссылке, в векторе всех вершин автомата.

int getGo(int index, char ch)

Функция получения вершины, для перехода в нее.

index - индекс вершины, из которой осуществляется переход

ch – символ, по которому осуществляется переход

Возвращаемым значением является индекс вершины для перехода в векторе всех вершин автомата.

void printMachine() const

Функция, печатающая автомат, который был построен во время работы алгоритма.

Тестирование.

Входные данные:

СССА

1

СС

Результат работы программы:

Add symbols of new pattern in prefix tree

Received character C

Add new vertex in prefix tree with number 1

Symbol path to the vertex C

Transition in vertex by symbol C

Received character C

Add new vertex in prefix tree with number 2

Symbol path to the vertex C

Transition in vertex by symbol C

New terminal vertex is C

Count of terminal vertex 1

Deep of terminal vertex 2

Search for patterns in the line is started

Search for symbols C

Go to the vertex 1 by symbol C

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link

Suffix link follow to the root

Terminal vertex was not found, go to the next symbol

Search for symbols C

Go to the vertex 2 by symbol C

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern with index 1 and number 1

Search for symbols C

Follow the new suffix link and add the transition to the path of machine

Start find path by suffix

Go to the vertex 1 by path from vectors of paths

Go to the vertex 2 by path from vectors of paths

Received the path to the vertex 2 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern with index 2 and number 1

Search for symbols A

Follow the new suffix link and add the transition to the path of machine

Follow the new suffix link and add the transition to the path of machine

Go to the root

Received the path to the vertex 0 through the suffix link

Received the path to the vertex 0 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Machine built during the operation of the algorithm

Vertex 0 with possible path:

Vertex 0 with path A

Vertex 1 with path C

Vertex 1 with possible path:

Vertex 0 with path A

Vertex 2 with path C

Vertex 2 with possible path:

Vertex 0 with path A

Vertex 2 with path C

Max arcs of one vertex prefix tree: 1

The result of algorithm work:

1 1

2 1

String without found patterns:

A

Входные данные:

BAAACBACB

4

ACB

ACA

BA

BC

Результат работы программы:

Add symbols of new pattern in prefix tree

Received character A

Add new vertex in prefix tree with number 1

Symbol path to the vertex A

Transition in vertex by symbol A

Received character C

Add new vertex in prefix tree with number 2

Symbol path to the vertex C

Transition in vertex by symbol C

Received character B

Add new vertex in prefix tree with number 3

Symbol path to the vertex B

Transition in vertex by symbol B

New terminal vertex is B

Count of terminal vertex 1

Deep of terminal vertex 3

Add symbols of new pattern in prefix tree

Received character A

Transition in vertex by symbol A

Received character C

Transition in vertex by symbol C

Received character A
Add new vertex in prefix tree with number 4
Symbol path to the vertex A
Transition in vertex by symbol A

New terminal vertex is A
Count of terminal vertex 2
Deep of terminal vertex 3

Add symbols of new pattern in prefix tree
Received character B
Add new vertex in prefix tree with number 5
Symbol path to the vertex B
Transition in vertex by symbol B

Received character A
Add new vertex in prefix tree with number 6
Symbol path to the vertex A
Transition in vertex by symbol A

New terminal vertex is A
Count of terminal vertex 3
Deep of terminal vertex 2

Add symbols of new pattern in prefix tree
Received character B
Transition in vertex by symbol B

Received character C
Add new vertex in prefix tree with number 7
Symbol path to the vertex C
Transition in vertex by symbol C

New terminal vertex is C
Count of terminal vertex 4
Deep of terminal vertex 2

Search for patterns in the line is started

Search for symbols B

Go to the vertex 5 by symbol B

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link

Suffix link follow to the root

Terminal vertex was not found, go to the next symbol

Search for symbols A

Go to the vertex 6 by symbol A

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern with index 1 and number 3

Search for symbols A

Follow the new suffix link and add the transition to the path of machine

Start find path by suffix

Go to the vertex 1 by symbol A

Add the transition to the paths of machine

Follow the new suffix link and add the transition to the path of machine

Suffix link follow to the root

Go to the vertex 1 by path from vectors of paths

Received the path to the vertex 1 through the suffix link

Received the path to the vertex 1 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link
Terminal vertex was not found, go to the next symbol

Search for symbols A

Go to the vertex 1 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link
Terminal vertex was not found, go to the next symbol

Search for symbols C

Go to the vertex 2 by symbol C

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link
Start find path by suffix

Go to the root

Terminal vertex was not found, go to the next symbol

Search for symbols B

Go to the vertex 3 by symbol B

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link
Terminal vertex found, found pattern with index 4 and number 1

Search for symbols A

Follow the new suffix link and add the transition to the path of machine

Start find path by suffix

Go to the vertex 5 by path from vectors of paths

Go to the vertex 6 by path from vectors of paths

Received the path to the vertex 6 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link
Terminal vertex found, found pattern with index 6 and number 3

Search for symbols C

Follow the new suffix link and add the transition to the path of machine
Go to the vertex 2 by path from vectors of paths

Received the path to the vertex 2 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link
Terminal vertex was not found, go to the next symbol

Search for symbols B
Go to the vertex 3 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link
Terminal vertex found, found pattern with index 7 and number 1

Machine built during the operation of the algorithm

Vertex 0 with possible path:
 Vertex 1 with path A
 Vertex 5 with path B
 Vertex 0 with path C

Vertex 1 with possible path:
 Vertex 1 with path A
 Vertex 2 with path C

Vertex 2 with possible path:
 Vertex 3 with path B

Vertex 3 with possible path:
 Vertex 6 with path A

Vertex 4 with possible path:

Vertex 5 with possible path:
 Vertex 6 with path A

Vertex 6 with possible path:

Vertex 1 with path A

Vertex 2 with path C

Vertex 7 with possible path:

Max arcs of one vertex prefix tree: 2

The result of algorithm work:

1 3

4 1

6 3

7 1

String without found patterns:

A

Алгоритм поиска шаблона 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 Р необходимо найти все вхождения Р в текст Т.

Например, образец *ab??c?* с джокером *?* встречается дважды в тексте *xabvccbababcaх*.

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, 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.

Пример выходных данных

АСТ

А\$

\$

Пример выходных данных

1

Описание алгоритма.

В начале работы алгоритма считывается шаблон, поиск которого будет осуществляться. Этот шаблон разделяется функцией на подшаблоны, которые были разделены друг от друга символом джокера в строке-шаблоне. Также запоминаются индексы этих подшаблонов в строке-шаблоне для дальнейшей работы алгоритма.

Далее с помощью алгоритма Ахо-Корасик подшаблоны заносятся в бор и осуществляется их поиск в строке. Когда подшаблон находится в строке поиска, то инкрементируется значение, находящееся в индексе вектора совпадений подшаблонов. Этот индекс определяется как индекс вхождения подшаблона в строку минус индекс подшаблона в строке-шаблоне.

После того, как вся строка поиска будет обработана и все подшаблоны найдены, то проверяются значения вектора вхождения подшаблонов. Если в каком-либо индексе этого вектора хранится число, равное количеству всех подшаблонов шаблона, значит строка-шаблон входит в строку поиска на этом индексе полностью. Индекс вхождения этого шаблона запоминается и заносится в вектор результата.

Для вывода максимального числа дуг, исходящих из одной вершины бора перебираются вершины-дети бора. Если число дуг для текущей вершины больше переменной, хранящей это максимальное число, то в переменную заносится это новое значение. Результатом является значение, хранящееся в этой переменной.

Для вывода строки, из которой были удалены найденные шаблоны заводится булевский вектор. Индексы, соответствующие индексам с символами шаблона в строке, помечаются. Строка формируется путем добавления в нее символов, индексы которых не были помечены.

Сложность алгоритма по операциям:

Аналогично алгоритму Ахо-Корасик и проход по вектору совпадений подшаблонов в тексте: $O((M+N)*\log(k)+t+N)$, M – длина всех символов слов шаблона, N – длина текста, в котором осуществляется поиск, k – размер алфавита, t – длина всех возможных вхождений всех строк-образцов.

Сложность алгоритма по памяти:

Помимо данных, которые хранятся в алгоритме Ахо-Корасик, еще необходимо хранить массив подшаблонов, массив длин подшаблонов и массив, в котором отмечается количество входящих подшаблонов в каждый символ текста-поиска. Длина этого массива будет равна количеству символов текста-поиска: $O(2*M+2*N+W)$, M – длина всех символов слов шаблона, N – длина текста, в котором осуществляется поиск, W – количество подшаблонов

Описание функций и структур данных.

Структура вершины

struct Vertex

```
{  
    std::map<char, int> next; - переходы по ребрам в боре  
    std::map<char, int> go;   - переходы в автомате  
    int prev = 0;            - индекс вершины-предка  
    char prevChar = 0;       - символ перехода в вершину из родительской  
    int suffix = -1;         - индекс вершины в которую можно попасть по  
суффиксной ссылке  
    int number = 0;          - номер шаблона терминальной вершины  
    int deep = 0;            - глубина вершины  
    bool isLeaf = false;     - является ли вершина терминальной  
};
```

void addPattern(const std::string& str)

Функция добавления символов шаблона в бор

str — шаблон для добавления в бор

void search(const std::string& str)

Функция поиска шаблонов в строке

str — текст, в котором будет осуществляться поиск

void printResult(const std::string& text) const

Функция вывода результата работы алгоритма и строки, из которой были удалены найденные шаблоны.

text – текст, в котором осуществляется поиск шаблонов.

void printMaxArcs() const

Функция, вычисляющая максимальное количество дуг, исходящих из одной вершины в боре

int getSuffix(int index)

Функция получения вершины, доступной по суффиксной ссылке.

index – индекс вершины, для которой осуществляется поиск по суффиксной ссылке.

Возвращаемым значением является индекс вершины, доступной по суффиксной ссылке, в векторе всех вершин автомата.

int getGo(int index, char ch)

Функция получения вершины, для перехода в нее.

index - индекс вершины, из которой осуществляется переход

ch – символ, по которому осуществляется переход

Возвращаемым значением является индекс вершины для перехода в векторе всех вершин автомата.

void printMachine() const

Функция, печатающая автомат, который был построен во время работы алгоритма.

void readPattern(std::string& str)

Функция обработки считанного шаблона

str - считанный шаблон

void split(std::string str)

Функция разбиения шаблонов на подшаблоны

str – шаблон, который будет разбит на подшаблоны

Тестирование.

Входные данные:

ABBBACBCA

\$B\$A

\$

Результат работы программы:

Read pattern processing

Read pattern: \$B\$A

Subpattern and his index in patterns:

B with index 2

A with index 4

Add subpatterns in prefix tree

Add symbols of new pattern in prefix tree

Received character B

Add new vertex in prefix tree with number 1

Symbol path to the vertex B

Transition in vertex by symbol B

New terminal vertex is B

Deep of terminal vertex 1

Add symbols of new pattern in prefix tree

Received character A

Add new vertex in prefix tree with number 2

Symbol path to the vertex A

Transition in vertex by symbol A

New terminal vertex is A

Deep of terminal vertex 1

Search for patterns in the line is started

Search for symbols A

Go to the vertex 2 by symbol A

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link

Suffix link follow to the root

Terminal vertex was not found, go to the next symbol

Search for symbols B

Follow the new suffix link and add the transition to the path of machine

Go to the vertex 1 by symbol B

Add the transition to the paths of machine

Received the path to the vertex 1 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern B with index in text 2

Count subpatterns in this index 1 of 2

Suffix link follow to the root

Search for symbols B

Follow the new suffix link and add the transition to the path of machine

Go to the vertex 1 by path from vectors of paths

Received the path to the vertex 1 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern B with index in text 3

Count subpatterns in this index 1 of 2

Search for symbols B

Go to the vertex 1 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern B with index in text 4

Count subpatterns in this index 1 of 2

Search for symbols A

Follow the new suffix link and add the transition to the path of machine

Go to the vertex 2 by path from vectors of paths

Received the path to the vertex 2 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern A with index in text 5

Count subpatterns in this index 2 of 2

Search for symbols C

Follow the new suffix link and add the transition to the path of machine

Go to the root

Received the path to the vertex 0 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols B

Go to the vertex 1 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern B with index in text 7

Count subpatterns in this index 1 of 2

Search for symbols C

Follow the new suffix link and add the transition to the path of machine

Go to the vertex 0 by path from vectors of paths

Received the path to the vertex 0 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols A

Go to the vertex 2 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern A with index in text 9

Count subpatterns in this index 2 of 2

Machine built during the operation of the algorithm

Vertex 0 with possible path:

Vertex 2 with path A

Vertex 1 with path B

Vertex 0 with path C

Vertex 1 with possible path:

Vertex 2 with path A

Vertex 1 with path B

Vertex 0 with path C

Vertex 2 with possible path:

Vertex 1 with path B

Vertex 0 with path C

Max arcs of one vertex prefix tree: 2

The result of algorithm work:

2

6

String without found patterns:

A

Входные данные:

BACECAACAACBE

AC\$

\$

Результат работы программы:

Read pattern processing

Read pattern: AC\$

Subpattern and his index in patterns:

AC with index 1

Add subpatterns in prefix tree

Add symbols of new pattern in prefix tree

Received character A

Add new vertex in prefix tree with number 1

Symbol path to the vertex A

Transition in vertex by symbol A

Received character C

Add new vertex in prefix tree with number 2

Symbol path to the vertex C

Transition in vertex by symbol C

New terminal vertex is C

Deep of terminal vertex 2

Search for patterns in the line is started

Search for symbols B

Go to the root

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols A

Go to the vertex 1 by symbol A

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link

Suffix link follow to the root

Terminal vertex was not found, go to the next symbol

Search for symbols C

Go to the vertex 2 by symbol C

Add the transition to the paths of machine

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern AC with index in text 2

Count subpatterns in this index 1 of 1

Start find path by suffix

Go to the root

Search for symbols E

Follow the new suffix link and add the transition to the path of machine

Go to the root

Received the path to the vertex 0 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols C

Go to the vertex 0 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols A

Go to the vertex 1 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols A

Follow the new suffix link and add the transition to the path of machine

Go to the vertex 1 by path from vectors of paths

Received the path to the vertex 1 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols C

Go to the vertex 2 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern AC with index in text 7

Count subpatterns in this index 1 of 1

Search for symbols A

Follow the new suffix link and add the transition to the path of machine

Go to the vertex 1 by path from vectors of paths

Received the path to the vertex 1 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols A

Go to the vertex 1 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols C

Go to the vertex 2 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex found, found pattern AC with index in text 10

Count subpatterns in this index 1 of 1

Search for symbols B

Follow the new suffix link and add the transition to the path of machine

Go to the vertex 0 by path from vectors of paths

Received the path to the vertex 0 through the suffix link

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Search for symbols E

Go to the vertex 0 by path from vectors of paths

Start check terminal vertex and check terminal vertex by suffix link

Terminal vertex was not found, go to the next symbol

Machine built during the operation of the algorithm

Vertex 0 with possible path:

Vertex 1 with path A

Vertex 0 with path B

Vertex 0 with path C

Vertex 0 with path E

Vertex 1 with possible path:

Vertex 1 with path A

Vertex 2 with path C

Vertex 2 with possible path:

Vertex 1 with path A

Vertex 0 with path B

Vertex 0 with path E

Max arcs of one vertex prefix tree: 1

The result of algorithm work:

2

7

10

String without found patterns:

BCAE

Выводы.

В ходе выполнения лабораторной работы были получены навыки работы с алгоритмом Ахо-Корасик и алгоритмом поиска подстроки с “джокером”. Были написаны программы, реализующую эти алгоритмы работы со строками.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

АЛГОРИТМ АХО-КОРАСИК

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

struct Vertex          //Структура вершины
{
    std::map<char, int> next;    // сыновья вершины в боре
    std::map<char, int> go;     // пути автомата
    int prev = 0;              // индекс родительской вершины
    char prevChar = 0;         // символ родительской вершины
    int suffix = -1;           // индекс вершины суффиксного
перехода
    int number = 0;            // номер терминальной вершины
    int deep = 0;              // глубина вершины
    bool isLeaf = false;       // является ли вершина листом
(терминалом)
};

class AhoCorasick      // класс реализующий алгоритм Ахо-
Корасик
{
public:

    AhoCorasick(){
        Vertex root;          // создание корневой вершины
        root.prev = -1;
        vertexs.push_back(root); // добавление вершины в вектор
вершин
        countVertex = 0;
    }

    void addPattern(const std::string& str) // добавление символов
паттерна в бор
    {
        std::cout << "-----" <<
std::endl;
        std::cout << "Add symbols of new pattern in prefix tree" <<
std::endl;
        currentVertex = 0;
        for (char i : str) {
            std::cout << "\tReceived character " << i << std::endl;
            if (vertexs[currentVertex].next.find(i) ==
vertexs[currentVertex].next.end()) { // если для текущей вершины нет
перехода по символу
                Vertex vertex;
```



```

// вершина создается и добавляется в бор
    vertex.prev = currentVertex;
    vertex.prevChar = i;
    vertexs.push_back(vertex);
    vertexs[currentVertex].next[i] = vertexs.size() - 1;
    std::cout << "\tAdd new vertex in prefix tree with
number " << vertexs.size() - 1 << std::endl;
    std::cout << "\tSymbol path to the vertex " << i <<
std::endl;
    }
    currentVertex = vertexs[currentVertex].next[i];
    std::cout << "\tTransition in vertex by symbol " << i <<
std::endl << std::endl;
    }

    countVertex++;
    patternsArr.push_back(str);
    vertexs[currentVertex].number = countVertex;
// номер терминальной вершины
    vertexs[currentVertex].isLeaf = true; // вершина объявляется
терминальной
    vertexs[currentVertex].deep = str.size();
    std::cout << "New terminal vertex is " <<
vertexs[currentVertex].prevChar << std::endl;
    std::cout << "Count of terminal vertex " <<
vertexs[currentVertex].number << std::endl;
    std::cout << "Deep of terminal vertex " <<
vertexs[currentVertex].deep << std::endl;
    std::cout << "-----" <<
std::endl;
    }

    void search(const std::string& str)
// поиск паттернов в строке
    {
        std::cout << std::endl << "\nSearch for patterns in the line
is started" << std::endl << std::endl << std::endl;
        std::cout << "-----" <<
std::endl;
        int curr = 0;
        bool terminalVertexFound;
        for (size_t i = 0; i < str.size(); i++) {
            std::cout << "Search for symbols " << str[i] << std::endl;
            curr = getGo(curr, str[i]);
// по каждому символу переходим в новую вершину бора
            std::cout << "\nStart check terminal vertex and check
terminal vertex by suffix link" << std::endl;
            terminalVertexFound = false;
            for (int tmp = curr; tmp != 0; tmp = getSuffix(tmp)) {
// также осуществляем переходы по суффиксным ссылкам
                if (vertexs[tmp].isLeaf) {
// если символ терминальный, то добавляем индекс в массив результата
                    result.push_back(i - vertexs[tmp].deep + 2);
                    result.push_back(vertexs[tmp].number);

```

```

        terminalVertexFound = true;
        std::cout << "Terminal vertex found, found pattern
with index " << i - vertexs[tmp].deep + 2 << " and number " <<
vertexs[tmp].number << std::endl;
        break;
    }
}
if (!terminalVertexFound){
    std::cout << "Terminal vertex was not found, go to the
next symbol" << std::endl;
}
std::cout << "-----"
<< std::endl;
}
}

void printResult(const std::string& text) const{
// функция печати результата
    printMachine();
    printMaxArcs();
    std::vector<bool> cutStr(text.size());
// вектор попадания символов, вошедших в паттерн
    std::string textRest;
// входной текст без паттернов
    std::cout << "The result of algorithm work:" <<std::endl;
    for (size_t i = 0; i < result.size(); i += 2) {
        std::cout << result[i] << " " << result[i + 1] <<
std::endl;
        for (int j = 0; j < patternsArr[result[i+1] - 1].size();
j++)
            cutStr[result[i] - 1 + j] = true;
    }

    for (int i = 0; i < cutStr.size(); i++){
        if (!cutStr[i])
            textRest.push_back(text[i]);
// заполняем строку символов, которые не вошли в паттерн
    }

    std::cout << std::endl << "String without found patterns:" <<
std::endl;
    std::cout << textRest << std::endl;
}

private:
    void printMaxArcs() const{ //функция, вычисляющая
максимальное количество дуг, исходящих из одной вершины в боре
        auto current = vertexs.begin();
        int maxArcs = 0;
        while (current != vertexs.end()){
            if (current->next.size() > maxArcs)
                maxArcs = current->next.size();
            current++;
        }
    }
}

```

```

        std::cout << std::endl << "Max arcs of one vertex prefix tree:
" << maxArcs << std::endl << std::endl;
    }

    int getSuffix(int index) //
    получение вершины перехода по суффиксной ссылке
    {
        if (vertexs[index].suffix == -1) { //
            если суффиксная ссылка еще не определена
                if (index == 0 || vertexs[index].prev == 0) {
                    vertexs[index].suffix = 0; //
                }
                если корень или родитель корень - то суффиксная ссылка ведет в корень
                std::cout << "Suffix link follow to the root" <<
std::endl;
            }
            else {
                std::cout << "Start find path by suffix " <<
std::endl;
                vertexs[index].suffix =
getGo(getSuffix(vertexs[index].prev), vertexs[index].prevChar); //
            }
            иначе переходим ищем суффикс через суффикс родителя
        }
    }

    return vertexs[index].suffix; //
    возвращаем индекс суффиксной вершины в векторе вершин
}

int getGo(int index, char ch) //
получить путь автомата из текущей вершины
{
    if (vertexs[index].go.find(ch) == vertexs[index].go.end()) {
        // если пути по символу из текущей вершины нет
        if (vertexs[index].next.find(ch) !=
vertexs[index].next.end()) {
            vertexs[index].go[ch] = vertexs[index].next[ch];
            // если из вершины есть дети, то путь прокладывается через них
            std::cout << "\nGo to the vertex " <<
vertexs[index].go[ch] << " by symbol " << ch << std::endl;
            std::cout << "Add the transition to the paths of
machine" << std::endl;
        }
        else {
            if (index == 0) {
                vertexs[index].go[ch] = 0;
                std::cout << "\nGo to the root " << std::endl;
            }
            else {
                std::cout << "\nFollow the new suffix link and add
the transition to the path of machine" << std::endl;
                vertexs[index].go[ch] =
getGo(getSuffix(index), ch); // иначе путь прокладывается через
суффиксную ссылку
            }
        }
    }
}

```

```

        std::cout << "\nReceived the path to the vertex "
<< vertexs[index].go[ch] << " through the suffix link" << std::endl;
    }
}
else{
    std::cout << "Go to the vertex " << vertexs[index].go[ch]
<< " by path from vectors of paths" << std::endl;
}

    return vertexs[index].go[ch];        // возвращаем индекс
вершины пути в векторе вершин
}

void printMachine() const {
//функция печати автомата, полученного во время работы алгоритма
    std::cout << "-----" <<
std::endl;
    std::cout << "Machine built during the operation of the
algorithm" << std::endl << std::endl;
    for (int i=0; i<vertexs.size(); i++){
        std::cout << "Vertex " << i << " with possible path: " <<
std::endl;
        auto cur = vertexs[i].go.begin();
        while (cur != vertexs[i].go.end()){
            std::cout << "\tVertex " << cur->second << " with path
" << cur->first << std::endl;
            cur++;
        }
        std::cout << std::endl;
    }
    std::cout << "-----" <<
std::endl;
}

private:
    std::vector<Vertex> vertexs;        // вектор вершин
    std::vector<int> result;            // вектор результата
    int countVertex;                   // количество терминальных вершин
в боре
    int currentVertex;                  // текущая вершина
    std::vector<std::string> patternsArr;
};

int main() {
    std::string str;
    int count = 0;
    std::cout << "Enter text string:" << std::endl;
    std::cin >> str;
    std::cout << "Enter count patterns:" << std::endl;
    std::cin >> count;

    std::string pattern;

```

```

std::vector<std::string> patterns;
std::cout << "Enter all patterns:" << std::endl;
for (int i = 0; i < count; i++){
    std::cin >> pattern;
    patterns.push_back(pattern);
}

auto* ahoCorasick = new AhoCorasick();
for (int i = 0; i < count; i++) {           // заполнение бора
    ahoCorasick->addPattern(patterns[i]);
}
символами паттернов

ahoCorasick->search(str);                   // запуск поиска
ahoCorasick->printResult(str);               // печать
результата

return 0;
}

```

АЛГОРИТМ ПОИСКА ШАБЛОНА С “ДЖОКЕРОМ”

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

struct Vertex //Структура вершины
{
    std::map<char, int> next; //сыновья вершины в боре
    std::map<char, int> go; // пути автомата
    std::vector<int> number; // массив номеров паттернов
    int prev = 0; // индекс родительской вершины
    int deep = 0; // глубина вершины
    int suffix = -1; // индекс вершины суффиксного
перехода
    bool isLeaf = false; // является ли вершина листом
(терминалом)
    char prevChar = 0; // символ родительской вершины
};

class AhoCorasick // класс реализующий алгоритм Ахо-
Корасик
{
public:
    explicit AhoCorasick(char joker): matchPatterns(1100000){
        Vertex root; // создание корневой вершины
        root.prev = -1;
        vertexs.push_back(root); // добавление вершины в вектор
вершин
        this->joker = joker; // определяется джокер
        countTerminalVertex = 0; // определяется число
терминальных вершин
    }

    void readPattern(std::string& str){ //функция считывание
паттерна
        std::cout << "-----" <<
std::endl;
        std::cout << "Read pattern processing" << std::endl;
        std::cout << "Read pattern: " << str << std::endl;
        patternLen = str.size();
        split(str); //разбиение паттерна по
джокерам

        std::cout << "Subpattern and his index in patterns: " <<
std::endl;
        for (int i = 0; i < patternArr.size(); i++){
            std::cout << patternArr[i] << " with index " <<
patternLength[i] + 1 << std::endl;
        }
    }
};
```

```

        std::cout << "-----";
        std::cout << std::endl << std::endl << "Add subpatterns in
prefix tree " << std::endl << std::endl << std::endl;

        for (const auto& pattern : patternArr) {
            addPattern(pattern);          // добавление каждого
подпатерна в бор
        }

    }

    void search(const std::string& str)          // поиск
паттернов в строке
    {
        std::cout << std::endl << "\nSearch for patterns in the line
is started" << std::endl << std::endl << std::endl;
        std::cout << "-----" <<
std::endl;
        int curr = 0;
        bool terminalVertexFound;
        for (int i = 0; i < str.size(); i++) {
            std::cout << "Search for symbols " << str[i] << std::endl;
            curr = getGo(curr, str[i]);          // по каждому
символу переходим в новую вершину бора
            std::cout << "\nStart check terminal vertex and check
terminal vertex by suffix link" << std::endl;
            terminalVertexFound = false;
            for (int tmp = curr; tmp != 0; tmp = getSuffix(tmp)) {
                // также осуществляем переходы по суффиксным ссылкам
                if (vertexs[tmp].isLeaf) {
                    for (int j = 0; j < vertexs[tmp].number.size();
j++) {
                        if (i + 1 -
patternsLength[vertexs[tmp].number[j] - 1] - vertexs[tmp].deep >= 0 &&
i + 1 -
patternsLength[vertexs[tmp].number[j] - 1] - vertexs[tmp].deep <=
str.size() - patternLen){ //если паттерн не выходит за границы
(слева и справа)
                            matchPatterns[i + 1 -
patternsLength[vertexs[tmp].number[j] - 1] - vertexs[tmp].deep]++;
                            // добавляем индекс совпадения в вектор совпадений подпаттернов
                            terminalVertexFound = true;
                            std::cout << "Terminal vertex found, found
pattern " << patternArr[vertexs[tmp].number[j] - 1] << " with index
in text " << i - vertexs[tmp].deep + 2 << std::endl;
                            std::cout << "Count subpatterns in this
index " << matchPatterns[i + 1 - patternsLength[vertexs[tmp].number[j]
- 1] - vertexs[tmp].deep] << " of " << patternsLength.size() <<
std::endl;
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
        if (!terminalVertexFound){
            std::cout << "Terminal vertex was not found, go to the
next symbol" << std::endl;
        }
        std::cout << "-----"
<< std::endl;
    }

}

    void printResult(const std::string& text) const{
// функция печати результата
        printMachine();
        printMaxArcs();
        std::vector<bool> cutStr(text.size());
// вектор попадания символов, вошедших в паттерн
        std::string textRest;
// входной текст без паттернов
        std::cout << "The result of algorithm work:" <<std::endl;
        for (int i = 0; i < matchPatterns.size(); i++) {
            if (matchPatterns[i] == patternsLength.size()) {
// если число вошедших подпаттернов в индексе совпадет с числом всех
подпаттнов, то это индекс вхождения паттерна
                std::cout << i + 1 << "\n";
                for (int j = 0; j < patternLen; j++)
                    cutStr[i + j] = true;
// помечаем, что символ вошел в паттерн
            }
        }
        for (int i = 0; i < cutStr.size(); i++){
            if (!cutStr[i])
                textRest.push_back(text[i]);
// заполняем строку символов, которые не вошли в паттерн
        }

        std::cout << std::endl << "String without found patterns:" <<
std::endl;
        std::cout << textRest << std::endl;
    }

private:

    void printMaxArcs() const{           //функция, вычисляющая
максимальное количество дуг, исходящих из одной вершины в боре
        auto current = vertexs.begin();
        int maxArcs = 0;
        while (current != vertexs.end()){
            if (current->next.size() > maxArcs)
                maxArcs = current->next.size();
            current++;
        }
        std::cout << std::endl << "Max arcs of one vertex prefix tree:

```



```

" << maxArcs << std::endl << std::endl;
    }

    void split(std::string str) {          // функция разделения
паттернов на подпаттерны
        std::string buf = "";
        for (int i=0; i<str.size(); i++){
            if (str[i] == joker){
                if (!buf.empty()) {
                    patternArr.push_back(buf);
//заполняет массив подпаттернов
                    patternsLength.push_back(i - buf.size());
//и массив их вхождения в паттерне
                    buf = "";
                }
            }
            else {
                buf.push_back(str[i]);
                if (i == str.size() - 1){
                    patternArr.push_back(buf);
                    patternsLength.push_back(i - buf.size() + 1);
                }
            }
        }
    }

    void addPattern(const std::string& str)    // добавление символов
паттерна в бор
    {
        std::cout << "-----" <<
std::endl;
        std::cout << "Add symbols of new pattern in prefix tree" <<
std::endl;
        int current = 0;
        for (char i : str) {
            std::cout << "\tReceived character " << i << std::endl;
            if (vertexs[current].next.find(i) ==
vertexs[current].next.end()) { // если для текущей вершины нет
перехода по символу
                Vertex ver;
// вершина создается и добавляется в бор
                ver.suffix = -1;
                ver.prev = current;
                ver.prevChar = i;
                vertexs.push_back(ver);
                vertexs[current].next[i] = vertexs.size() - 1;
                std::cout << "\tAdd new vertex in prefix tree with
number " << vertexs.size() - 1 << std::endl;
                std::cout << "\tSymbol path to the vertex " << i <<
std::endl;
            }
            current = vertexs[current].next[i];
            std::cout << "\tTransition in vertex by symbol " << i <<
std::endl << std::endl;

```

```

    }
    countTerminalVertex++;
    vertexs[current].number.push_back(countTerminalVertex);
//номера подпаттернов
    vertexs[current].isLeaf = true;
//
вершина объявляется терминальной
    vertexs[current].deep = str.size();
    std::cout << "New terminal vertex is " <<
vertexs[current].prevChar << std::endl;
    std::cout << "Deep of terminal vertex " <<
vertexs[current].deep << std::endl;
    std::cout << "-----" <<
std::endl;

}

int getSuffix(int index)
//
получение вершины перехода по суффиксной ссылке
{
    if (vertexs[index].suffix == -1) {
//
если суффиксная ссылка еще не определена
        if (index == 0 || vertexs[index].prev == 0) {
            vertexs[index].suffix = 0;
//
если корень или родитель корень - то суффиксная ссылка ведет в корень
            std::cout << "Suffix link follow to the root" <<
std::endl;
        }
        else {
            std::cout << "Start find path by suffix " <<
std::endl;
            vertexs[index].suffix =
getGo(getSuffix(vertexs[index].prev), vertexs[index].prevChar);
//
иначе переходим ищем суффикс через суффикс родителя
        }
    }
    return vertexs[index].suffix;
// возвращаем
индекс суффиксной вершины в векторе вершин
}

int getGo(int index, char ch)
// получить путь
автомата из текущей вершины
{
    if (vertexs[index].go.find(ch) == vertexs[index].go.end()) {
// если пути по символу из текущей вершины нет
        if (vertexs[index].next.find(ch) !=
vertexs[index].next.end()) {
            vertexs[index].go[ch] = vertexs[index].next[ch];
// если из вершины есть дети, то путь прокладывается через них
            std::cout << "\nGo to the vertex " <<
vertexs[index].go[ch] << " by symbol " << ch << std::endl;
            std::cout << "Add the transition to the paths of
machine" << std::endl;
        }
        else {

```

```

        if (index == 0){
            vertexs[index].go[ch] = 0;
            std::cout << "\nGo to the root " << std::endl;
        }
        else{
            std::cout << "\nFollow the new suffix link and add
the transition to the path of machine" << std::endl;
            vertexs[index].go[ch] = getGo(getSuffix(index),
ch);    // иначе путь прокладывается через суффиксную ссылку
            std::cout << "\nReceived the path to the vertex "
<< vertexs[index].go[ch] << " through the suffix link" << std::endl;
        }
    }
    }
    else{
        std::cout << "Go to the vertex " << vertexs[index].go[ch]
<< " by path from vectors of paths" << std::endl;
    }
    return vertexs[index].go[ch];    // возвращаем индекс
вершины пути в векторе вершин
}

void printMachine() const {    //функция печати автомата,
полученного во время работы алгоритма
    std::cout << "-----" <<
std::endl;
    std::cout << "Machine built during the operation of the
algorithm" << std::endl << std::endl;
    for (int i=0; i<vertexs.size(); i++){
        std::cout << "Vertex " << i << " with possible path: " <<
std::endl;
        auto cur = vertexs[i].go.begin();
        while (cur != vertexs[i].go.end()){
            std::cout << "\tVertex " << cur->second << " with path
" << cur->first << std::endl;
            cur++;
        }
        std::cout << std::endl;
    }
    std::cout << "-----" <<
std::endl;
}

private:
    std::vector<Vertex> vertexs;    // вектор вершин
    char joker;    // символ джокера
    int countTerminalVertex;    // количество терминальных
вершин
    std::vector<std::string> patternArr;    // вектор подпаттернов
    int patternLen{};    // длина паттерна
    std::vector<int> matchPatterns;    // вектор попаданий
подпаттернов
    std::vector<int> patternsLength;    // вектор длин
подпаттернов

```

```
};
```

```
int main() {  
    std::string str;  
    std::string pattern;  
    char joker;  
    std::cout << "Enter text string:" << std::endl;  
    std::cin >> str;  
    std::cout << "Enter pattern:" << std::endl;  
    std::cin >> pattern;  
    std::cout << "Enter joker:" << std::endl;  
    std::cin >> joker;  
  
    auto* ahoCorasick = new AhoCorasick(joker);  
    ahoCorasick->readPattern(pattern);           // обработка паттерна  
    ahoCorasick->search(str);                     // запуск поиска  
    ahoCorasick->printResult(str);                 // печать  
    результата  
  
    return 0;  
}
```