

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

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритм Ахо-Корасик

Студент гр. 8303

Быков А.В.

Преподаватель

Фирсов М.А.

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 2. Подсчитать количество вершин в автомате; вывести список найденных образцов, имеющих пересечения с другими найденными образцами в строке 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

СС

СА

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 ля 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.

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.

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.

Сложность алгоритма по операциям:

Таблица переходов автомата хранится как индексный массив, следовательно

- расход памяти $O(n * \text{Б} + H)$
- вычислительная сложность $O(n * \text{Б} + H + k)$

Где H — длина текста, в котором производится поиск, n — общая длина всех слов в словаре, Б — размер алфавита, k — общая длина всех совпадений.

Описание функций и структур данных.

Структура вершины

struct BohrPoint

```
{  
    int directLinks[5]; // переходы в боре  
    bool terminal; // является ли вершина конечной  
    int num; // номер паттерна  
    int parentIndex; // индекс вершины родителя  
    int suffixLink; // суффиксная ссылка  
    int charLinks[5]; // переходы в автомате  
    int charFromParent; // символ перехода от родителя
```

```
int compressedLink; // переход по конечной ссылке  
};
```

Структура хранения найденных паттернов

```
struct Adjacent  
{  
    int size; // размер паттерна  
    int index; // индекс в тексте  
    int num; // номер паттерна  
};
```

void pushPoint(string str, int number) Функция добавления символов шаблона в бор **str** – шаблон для добавления в бор, **number** – номер шаблона

void find(int v, int i)

Функция поиска шаблонов в строке

i – индекс в тексте, в котором будет осуществляться поиск, **v** – номер текущей вершины в автомате

void AHO_COR()

Функция осуществляет проход по строке поиска и реализует алгоритм Ахо-Корасик.

int getSuffixLink(int point)

Функция получения вершины, доступной по суффиксной ссылке.

point – индекс вершины, для которой осуществляется поиск по суффиксной ссылке. Возвращаемым значением является индекс вершины, доступной по суффиксной ссылке, в векторе всех вершин автомата.

int getCompressedLink (int point)

Функция получения вершины, доступной по конечной ссылке.

int getLink(int point, int charr)

Функция получения вершины, для перехода в нее.

point - индекс вершины, из которой осуществляется переход

charr – символ, по которому осуществляется переход

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

void printBohr()

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

Such way not found, adding new point with number 1

Switching to point - 1

Received character C

Such way not found, adding new point with number 2

Switching to point - 2

Searching patterns in text

New char: C

link to point 1 by char C

Current point: 1

Suffix link not found

Compressed link not found

New char: C

link to point 2 by char C

Current point: 2

Terminal point was found!

Pattern was found in index 1. Patter number - 1

Suffix link not found

link to point 1 by char C

Suffix link to point 1!

Compressed link not found

Compressed link not found

New char: C

Suffix link to point 1!

link to point 2 by char C

link to point 2 by char C

Current point: 2

Terminal point was found!

Pattern was found in index 2. Patter number - 1

Compressed link not found

New char: A

Suffix link to point 1!

Suffix link not found

link to point 0 by char A

link to point 0 by char A

link to point 0 by char A

Current point: 0

Variant 2 -- Individualization

Points number: 3

No adjusting patterns!

Machine built during the operation of the algorithm

Point 0 with paths:

Point 0 with path A

Point 1 with path C

Point 1 with paths:

Point 0 with path A

Point 2 with path C

Point 2 with paths:

Point 0 with path A

Point 2 with path C

Входные данные:

ACGTNNN

3

ACG

ACGT

AC

Результат работы программы:

Creating bohr

ACGTNNN

3

ACG ACGT

Add symbols of new pattern in prefix tree

Received character A

Such way not found, adding new point with number 1

Switching to point - 1

Received character C

Such way not found, adding new point with number 2

Switching to point - 2

Received character G

Such way not found, adding new point with number 3

Switching to point - 3

Add symbols of new pattern in prefix tree

Received character A

Switching to point - 1

Received character C

Switching to point - 2

Received character G

Switching to point - 3

Received character T

Such way not found, adding new point with number 4

Switching to point - 4

AC

Add symbols of new pattern in prefix tree

Received character A

Switching to point - 1

Received character C

Switching to point - 2

Searching patterns in text

New char: A

link to point 1 by char A

Current point: 1

Suffix link not found

Compressed link not found

New char: C

link to point 2 by char C

Current point: 2

Terminal point was found!

Pattern was found in index 1. Patter number - 3

Suffix link not found

link to point 0 by char C

Suffix link not found

Compressed link not found

New char: G

link to point 3 by char G

Current point: 3

Terminal point was found!

Pattern was found in index 1. Patter number - 1

Suffix link not found

link to point 0 by char G

Suffix link not found

Compressed link not found

New char: T

link to point 4 by char T

Current point: 4

Terminal point was found!

Pattern was found in index 1. Patter number - 2

Suffix link not found

link to point 0 by char T

Suffix link not found

Compressed link not found

New char: N

Suffix link not found

link to point 0 by char N

link to point 0 by char N

Current point: 0

New char: N

link to point 0 by char N

Current point: 0

New char: N

link to point 0 by char N

Current point: 0

Variant 2 -- Individualization

Adjacent patterns: pattern 1 and 3, index 1 and 1

Adjacent patterns: pattern 2 and 3, index 1 and 1

Adjacent patterns: pattern 1 and 2, index 1 and 1

Points number: 5

Machine built during the operation of the algorithm

Point 0 with paths:

Point 1 with path A

Point 0 with path C

Point 0 with path G

Point 0 with path T

Point 0 with path N

Point 1 with paths:

Point 2 with path C

Point 2 with paths:

Point 3 with path G

Point 3 with paths:

Point 4 with path T

Point 4 with paths:

Point 0 with path N

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 шаблон образцу P необходимо найти все вхождения P в текст T .

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

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

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

Вход:

Текст (T ,
 $1 \leq |T| \leq 100000$)
Шаблон (P , $1 \leq |P| \leq 40$)
Символ джокера

Выход:

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

Номера должны выводиться в порядке возрастания.

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.

Сложность алгоритма по операциям:

Таблица переходов автомата хранится как индексный массив, следовательно

- вычислительная сложность $O(n * \text{Б} + H + k)$

Где H — длина текста, в котором производится поиск, n — общая длина всех слов в словаре, Б — размер алфавита, k — общая длина всех совпадений.

- расход памяти $O(n * \text{Б} + n + 2 * H)$

Так как помимо данных, которые хранятся в алгоритме Ахо-Корасик, еще необходимо хранить массив подшаблонов и массив, в котором

отмечается количество входящих подшаблонов в каждый символ текста-поиска. Длина этого массива будет равна количеству символов текста-поиска.

Описание функций и структур данных.

Структура вершины

struct BohrPoint

```
{
    int directLinks[5]; // переходы в боре
    int patternNum[40]; // номер паттерна
    int suffixLink; // суффиксная ссылка
    int charLink[5]; // переходы в автомате
    int parentIndex; // индекс вершины родителя
    int compressedLink; // переход по конечной ссылке
    bool terminal; // является ли вершина конечной
    char charr; // символ перехода от родителя
};
```

Структура хранения найденных паттернов (подстрок)

struct Numbers

```
{
    int index; // индекс в тексте
    int patternNum; // индекс в строке паттерна
};
```

void addString(string& s)

Функция добавления символов шаблона в бор **s** – шаблон для добавления в бор

void find(int v, int i)

Функция поиска шаблонов в строке

i – индекс в тексте, в котором будет осуществляться поиск, **v** – номер текущей вершины в автомате

BohrPoint createPoint(int p, char c)

Функция для создания вершины в боре

p – индекс родителя, **c** – символ перехода

void АНО()

Функция осуществляет проход по строке поиска и реализует алгоритм Ахо-Корасик.

int getSuffixLink(int point)

Функция получения вершины, доступной по суффиксной ссылке.

point – индекс вершины, для которой осуществляется поиск по суффиксной ссылке. Возвращаемым значением является индекс вершины, доступной по суффиксной ссылке, в векторе всех вершин автомата.

int getCompressedLink (int point)

Функция получения вершины, доступной по конечной ссылке.

int getLink(int point, int charr)

Функция получения вершины, для перехода в нее.

point - индекс вершины, из которой осуществляется переход

charr – символ, по которому осуществляется переход

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

void printBohr()

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

void print ()

Функция вывода результата работы.

Тестирование.

Входные данные:

ACTANCA

A\$\$\$A\$

\$

Результат работы программы:

Creating bohr

ACTANCA

A\$\$\$A\$

\$

New pattern - A

Add symbols of new pattern in prefix tree

Received character A

Such way not found, adding new point with number 1

Switching to point - 1

New pattern - A

Add symbols of new pattern in prefix tree

Received character A

Switching to point - 1

Searching patterns in text

New char: A

link to point 1 by char A

Terminal point was found!

Pattern was found in index 0. Pattern number - 0

Pattern was found in index 0. Pattern number - 1

Suffix link not found

Compressed link not found

New char: C

Suffix link not found

link to point 0 by char C

link to point 0 by char C

New char: T

link to point 0 by char T

New char: A

link to point 1 by char A

Terminal point was found!

Pattern was found in index 3. Pattern number - 0

Pattern was found in index 3. Pattern number - 1

Compressed link not found

New char: N

Suffix link not found

link to point 0 by char N

link to point 0 by char N

New char: C

link to point 0 by char C

New char: A

link to point 1 by char A

Terminal point was found!

Pattern was found in index 6. Pattern number - 0

Pattern was found in index 6. Pattern number - 1

Compressed link not found

Analysing found patterns

Current pattern - A

On index 1 - coincidence 1 out of 2

Current pattern - A

Current pattern - A

On index 4 - coincidence 1 out of 2

Current pattern - A

On index 1 - coincidence 2 out of 2

Whole patter was found on 1

Current pattern - A

On index 7 - coincidence 1 out of 2

Current pattern - A

On index 4 - coincidence 2 out of 2

Variant 2 -- Individualization

Adjacent patterns: pattern 1 and 0, index 0 and 0

Adjacent patterns: pattern 1 and 0, index 3 and 3

Adjacent patterns: pattern 1 and 0, index 6 and 6

Points number: 2

Machine built during the operation of the algorithm

Point 0 with paths:

Point 1 with path A

Point 0 with path C

Point 0 with path T

Point 0 with path N

Point 1 with paths:

Point 0 with path C

Point 0 with path N

Входные данные:

ACGNCGTACGT

ACG%CGT

%

Результат работы программы:

Creating bohr

ACGNCGTACGT ACG%CGT %

New pattern - ACG

Add symbols of new pattern in prefix tree

Received character A

Such way not found, adding new point with number 1

Switching to point - 1

Received character C

Such way not found, adding new point with number 2

Switching to point - 2

Received character G

Such way not found, adding new point with number 3

Switching to point - 3

New pattern - CGT

Add symbols of new pattern in prefix tree

Received character C

Such way not found, adding new point with number 4

Switching to point - 4

Received character G

Such way not found, adding new point with number 5

Switching to point - 5

Received character T

Such way not found, adding new point with number 6

Switching to point - 6

Searching patterns in text

New char: A

link to point 1 by char A

Suffix link not found

Compressed link not found

New char: C

link to point 2 by char C

Suffix link not found

link to point 4 by char C

Suffix link to point 4!

Suffix link not found

Compressed link not found

Compressed link not found

New char: G

link to point 3 by char G

Terminal point was found!

Pattern was found in index 0. Pattern number - 0

Suffix link to point 4!

link to point 5 by char G

Suffix link to point 5!

Suffix link not found

link to point 0 by char G

Suffix link not found

Compressed link not found

Compressed link not found

New char: N

Suffix link to point 5!

Suffix link not found

link to point 0 by char N

link to point 0 by char N

link to point 0 by char N

New char: C

link to point 4 by char C

Compressed link not found

New char: G

link to point 5 by char G

Compressed link not found

New char: T

link to point 6 by char T

Terminal point was found!

Pattern was found in index 4. Pattern number - 1

Suffix link not found

link to point 0 by char T

Suffix link not found

Compressed link not found

New char: A

Suffix link not found

link to point 1 by char A

link to point 1 by char A

Compressed link not found

New char: C

link to point 2 by char C

Compressed link not found

New char: G

link to point 3 by char G

Terminal point was found!

Pattern was found in index 7. Pattern number - 0

Compressed link not found

New char: T

Suffix link to point 5!

link to point 6 by char T

link to point 6 by char T

Terminal point was found!

Pattern was found in index 8. Pattern number - 1

Compressed link not found

Analysing found patterns

Current pattern - ACG

On index 1 - coincidence 1 out of 2

Current pattern - CGT

On index 1 - coincidence 2 out of 2

Whole patten was found on 1

Current pattern - ACG

On index 8 - coincidence 1 out of 2

Current pattern - CGT

On index 5 - coincidence 1 out of 2

Variant 2 -- Individualization

Adjacent patterns: pattern 0 and 1, index 7 and 8

Points number: 7

Machine built during the operation of the algorithm

Point 0 with paths:

Point 1 with path A

Point 4 with path C

Point 0 with path G

Point 0 with path T

Point 0 with path N

Point 1 with paths:

Point 2 with path C

Point 2 with paths:

Point 3 with path G

Point 3 with paths:

Point 6 with path T

Point 0 with path N

Point 4 with paths:

Point 5 with path G

Point 5 with paths:

Point 6 with path T

Point 0 with path N

Point 6 with paths:

Point 1 with path A

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 <vector>
#include <string>

using namespace std;

class Bohr
{
private:
    string alphabet = "ACGTN";
    struct BohrPoint
    {
        int directLinks[5]; // переходы в боре
        bool terminal; // является ли вершина конечной
        int num; // номер паттерна
        int parentIndex; // индекс вершины родителя
        int suffixLink; // суффиксная ссылка
        int charrLinks[5]; // переходы в автомате
        int charFromParent; // символ перехода от родителя
        int compressedLink; // переход по конечной ссылке
    };

    struct Adjacent
    {
        int size; // размер паттерна
```

```

        int index; // индекс в тексте
        int num; // номер паттерна
    };

    vector <Adjacent> pats; // вектор для хранения
найденных паттернов
    string text; // строка поиска
    vector<BohrPoint> bohr; // бор
public:
    Bohr() // создание бора
    {
        cout << "Creating bohr\n";
        bohr.push_back({ {-1, -1, -1, -1, -1}, false,
0, 0, -1, {-1, -1, -1, -1, -1}, -1, -1 }); // добавляем
корневую вершину
        cin >> text;
        int c;
        cin >> c;
        for (int i = 0; i < c; i++)
        {
            string curr;
            cin >> curr;
            cout << "Add symbols of new pattern in
prefix tree\n";
            pushPoint(curr, i + 1); // добавляем паттерн в
бор
        }
    }

    void pushPoint(string str, int number) // функция

```

для добавления паттерна

```
{
    int index = 0;
    int charr;
    for (int i(0); i < str.size(); i++)
    {
        switch (str.at(i)) // определяем символ
перехода
        {
            case 'A':
                charr = 0;
                cout << "\nReceived character A" <<
endl;

                break;
            case 'C':
                charr = 1;
                cout << "\nReceived character C" <<
endl;

                break;
            case 'G':
                charr = 2;
                cout << "\nReceived character G" <<
endl;

                break;
            case 'T':
                charr = 3;
                cout << "\nReceived character T" <<
endl;

                break;
            case 'N':
```

```

        charr = 4;
        cout << "\nReceived character N" <<
endl;

        break;
    }
    if (bohr[index].directLinks[charr] == -1)
// если такого состояния в боре нет, то добавляем новое
    {
        cout << "Such way not found, adding
new point with number " << bohr.size() << endl;
        bool isEnd = false;
        if (i == str.size() - 1)
            isEnd = true;
        bohr.push_back({ {-1, -1, -1, -1, -1},
isEnd, number, index, -1, {-1, -1, -1, -1, -1}, charr,
-1 });

        bohr[index].directLinks[charr] =
bohr.size() - 1;
    }
    index = bohr[index].directLinks[charr];
    cout << "Switching to point - " << index
<< "\n";

    if (i == str.size() - 1) // проверка
терминальности вершины
    {
        bohr[index].terminal = true;
        bohr[index].num = number;
    }

}

```

```

    }

    int getSuffixLink(int point) // функция для получения
    суффиксной ссылки
    {
        if (bohr.at(point).suffixLink == -1) // если
        суффиксная ссылка не существует
        {
            if (point == 0 ||
            bohr.at(point).parentIndex == 0) // если текущая вершина -
            корень или если родительская вершина - корень
                bohr.at(point).suffixLink = 0; //
            возвращаем 0
        }
        else // иначе ищем ссылку на нижнем уровне
            bohr.at(point).suffixLink =
            getLink(getSuffixLink(bohr.at(point).parentIndex),
            bohr.at(point).charFromParent);
    }
    if (bohr.at(point).suffixLink)
        cout << "Suffix link to point " <<
        bohr.at(point).suffixLink << "!\n";
    else
        cout << "Suffix link not found\n";
    return bohr.at(point).suffixLink;
}

    int getLink(int point, int charr) // функция для
    получения обычной ссылки
    {
        if (bohr.at(point).charrLinks[charr] == -1) //

```

если ссылки нет

```
{
    if (bohr.at(point).directLinks[charr] != -
1) // проверяем есть ли ссылка в боре
        bohr.at(point).charrLinks[charr] =
bohr.at(point).directLinks[charr];
    else // пробуем найти суффиксную ссылку
    {
        if (point == 0)
            bohr.at(point).charrLinks[charr] =
0;
        else
            bohr.at(point).charrLinks[charr] =
getLink(getSuffixLink(point), charr);
    }
}
cout << "link to point " <<
bohr.at(point).charrLinks[charr] << " by char " <<
alphabet[charr] << endl;
return bohr.at(point).charrLinks[charr];
}
```

int getCompressedLink(int point) // // функция для
получения конечной ссылки

```
{
    if (bohr.at(point).compressedLink == -1) //
если конечной ссылки нет
    {
        int curr = getSuffixLink(point); // получаем
суффиксную ссылку
```

```

        if (curr == 0)
            bohr.at(point).compressedLink = 0;
        else
        {
            if (bohr.at(curr).terminal) // если
суффиксная ссылка указывает на конечную вершину
                bohr.at(point).compressedLink =
curr; // возвращаем конечную ссылку на конечную вершину
            else // иначе пробуем запустить функцию для
найденной вершины
                bohr.at(point).compressedLink =
getCompressedLink(curr);
        }
    }
    if (bohr.at(point).compressedLink)
        cout << "Compressed link to point " <<
bohr.at(point).compressedLink << "!\n";
    else
        cout << "Compressed link not found" <<
endl;

    return bohr.at(point).compressedLink;
}

void find(int v, int i) // функция для проверки
терминальности текущей вершины и запуска поиска конечных ссылок
{
    int j = 0;
    for (int u(v); u != 0; u =
getCompressedLink(u), j++)
    {

```

```

        if (bohr.at(u).terminal) // если найдена
конечная вершина
        {
            cout << "Terminal point was found!" <<
endl;

            int delta = 0;
            int curr = u;
            while (bohr.at(curr).parentIndex != 0)
            {
                curr = bohr.at(curr).parentIndex;
                delta++; // получаем размер паттерна,
с помощью возврата до корня
            }
            cout << "\n-----\nPattern was found in
index "<< i - delta << ". Patter number - " <<
bohr.at(u).num << "\n-----\n\n";
            pats.push_back({ delta, i - delta,
bohr.at(u).num });
        }
    }

void AHO_COR() // запуск алгоритма Ахо-Корасик
{
    cout << "\n-----\nSearching patterns in text
\n\n";

    int point = 0, charr = 0;
    for (int i(0); i < text.length(); i++) //
перебор всех символов текста

```



```

        {
            cout << "-----\nNew char: " << text.at(i)
<< endl << endl;
            switch (text.at(i))
            {
                case 'A':
                    charr = 0;
                    break;
                case 'C':
                    charr = 1;
                    break;
                case 'G':
                    charr = 2;
                    break;
                case 'T':
                    charr = 3;
                    break;
                case 'N':
                    charr = 4;
                    break;
            }
            point = getLink(point, charr);
            cout << "Current point: " << point <<
endl;

            find(point, i + 1);
        }

        cout << "\n-----\nVariant 2 -- Individalization"
<< endl;
        bool flag = true;

```

```

        for (int i = 0; i < pats.size(); i++)
        {
            int a, b;
            for (int j = 0; j < pats.size(); j++)
            {
                if (pats.at(j).index >
pats.at(i).index)
                {
                    a = j;
                    b = i;
                }
                else
                {
                    a = i;
                    b = j;
                }
                if (i != j && pats.at(a).index <
pats.at(b).index + pats.at(b).size)
                {
                    flag = false;
                    cout << "Adjacent patterns:
pattern " << pats.at(b).num << " and " <<
pats.at(a).num;

                    cout << ", index " <<
pats.at(b).index << " and " << pats.at(a).index <<
endl;
                }
            }
        }
        pats.erase(pats.begin());
    }
}

```

```

    }
    cout << "Points number: " << bohr.size() <<
endl;

    if (flag) cout << "No adjusting patterns!\n";
    printBohr(); // ВЫВОД АВТОМАТА
}

void printBohr()
{
    cout << "\n-----\nMachine built during the
operation of the algorithm\n\n";
    for (int i = 0; i < bohr.size(); i++)
    {
        cout << "Point " << i << " with paths:
\n";

        for (int j = 0; j < 5; j++)
            if (bohr.at(i).charrLinks[j] != -1)
                cout << "\tPoint " <<
bohr.at(i).charrLinks[j]<< " with paht " << alphabet[j]
<< endl;
    }
}

};

int main()
{
    Bohr bor;
    bor.AHO_COR();
    return 0;
}

```

АЛГОРИТМ ПОИСКА ШАБЛОНА С “ДЖОКЕРОМ”

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

using namespace std;
class Bohr
{
private:
    string alphabet = "ACGTN";

    struct Numbers
    {
        int index; // индекс в тексте
        int patternNum; // индекс в строке паттерна
    };

    struct BohrPoint
    {
        int directLinks[5]; // переходы в боре
        int patternNum[40]; // номер паттерна
        int suffixLink; // суффиксная ссылка
        int charrLink[5]; // переходы в автомате
        int parentIndex; // индекс вершины родителя
        int compressedLink; // переход по конечной ссылке
        bool terminal; // является ли вершина конечной
        char charr; // символ перехода от родителя
    };
};
```

```

};

vector <BohrPoint> bohr; // бор
string text; // строка поиска
string patternStr; // строка с паттерном
vector <string> patterns; // вектор хранения подстрок –
паттернов
vector <int> patternPos; // вектор хранения индексов
паттернов
vector<Numbers> num; // вектор хранения найденных
паттернов

public:
    Bohr() // создание бора
    {
        cout << "Creating bohr\n";
        bohr.push_back(createPoint(-1, -1));
        char joker;
        cin >> text >> patternStr >> joker;
        for (int i = 0; i < patternStr.size(); i++)
        {
            string pat;
            if (patternStr[i] != joker)
            {
                patternPos.push_back(i + 1);
                for (int j(i); patternStr[j] != joker
&& j != patternStr.length(); j++)
                {
                    pat += patternStr[j];
                    i++;

```

```

        }
        cout << "\n-----\nNew pattern - " <<
pat << endl;

        cout << "Add symbols of new pattern in
prefix tree\n";

        patterns.push_back(pat);
        addString(pat); // добавляем паттерн в бор
    }
}
}

```

BohrPoint createPoint(int p, char c) // функция для
создания вершины в боре

```

{
    BohrPoint v;
    memset(v.directLinks, -1,
sizeof(v.directLinks));
    memset(v.charrLink, -1, sizeof(v.charrLink));
    memset(v.patternNum, -1, sizeof(v.patternNum));
    v.terminal = false;
    v.suffixLink = -1;
    v.parentIndex = p;
    v.charr = c;
    v.compressedLink = -1;
    return v;
}

```

void addString(string& s) // функция для добавления
паттерна

```

{

```

```

int index = 0;
for (int i = 0; i < s.size(); i++)
{
    cout << "\nReceived character " << s[i] <<
endl;

    char ch = alphabet.find(s[i]);
    if (bohr[index].directLinks[ch] == -1) //
если такого состояния в боре нет, то добавляем новое
    {
        cout << "Such way not found, adding
new point with number " << bohr.size() << endl;
        bohr.push_back(createPoint(index,
ch));

        bohr[index].directLinks[ch] =
bohr.size() - 1;
    }
    index = bohr[index].directLinks[ch];
    cout << "Switching to point - " << index
<< "\n";
}
bohr[index].terminal = true;
for (int i = 0; i < 40; i++)
{
    if (bohr[index].patternNum[i] == -1)
    {
        bohr[index].patternNum[i] =
patterns.size() - 1; // запоминаем номер паттерна
        break;
    }
}
}

```

```

    }

    int getSuffixLink(int point) // функция для получения
    суффиксной ссылки
    {
        if (bohr[point].suffixLink == -1) // если
        суффиксная ссылка не существует
            if (point == 0 || bohr[point].parentIndex
            == 0) // если текущая вершина - корень или если родительская
            вершина - корень
                bohr[point].suffixLink = 0; //
        возвращаем 0
            else // иначе ищем ссылку на нижнем уровне
                bohr[point].suffixLink =
                getLink(getSuffixLink(bohr[point].parentIndex),
                bohr[point].charr);
            if (bohr.at(point).suffixLink)
                cout << "Suffix link to point " <<
                bohr.at(point).suffixLink << "!\n";
            else
                cout << "Suffix link not found\n";
            return bohr.at(point).suffixLink;
        }

    int getLink(int point, char charr) // функция для
    получения обычной ссылки
    {
        if (bohr[point].charrLink[charr] == -1) // если
        ссылки нет
            if (bohr[point].directLinks[charr] != -1)

```



```

// проверяем есть ли ссылка в боре
        bohr[point].charrLink[charr] =
bohr[point].directLinks[charr];

        else // пробуем найти суффиксную ссылку
        {
            if (point == 0)
                bohr[point].charrLink[charr] = 0;
            else
                bohr[point].charrLink[charr] =
getLink(getSuffixLink(point), charr);
        }

        cout << "link to point " <<
bohr.at(point).charrLink[charr] << " by char " <<
alphabet[charr] << endl;
        return bohr.at(point).charrLink[charr];
    }

    int getCompressedLink(int point) // функция для
получения конечной ссылки
    {
        if (bohr[point].compressedLink == -1) // если
конечной ссылки нет
        {
            int curr = getSuffixLink(point);
            if (curr == 0)
                bohr[point].compressedLink = 0;
            else
            {
                if (bohr[curr].terminal) // если

```

суффиксная ссылка указывает на конечную вершину

```
        bohr[point].compressedLink = curr;
// возвращаем конечную ссылку на конечную вершину
        else // иначе пробуем запустить функцию для
найденной вершины
            bohr[point].compressedLink =
getCompressedLink(curr);
    }
}
if (bohr.at(point).compressedLink)
    cout << "Compressed link to point " <<
bohr.at(point).compressedLink << "!\n";
else
    cout << "Compressed link not found" <<
endl;
return bohr.at(point).compressedLink;
}
```

```
void find(int v, int i) // функция для проверки
терминальности текущей вершины и запуска поиска конечных ссылок
{
    struct Numbers s;
    for (int u(v); u != 0; u =
getCompressedLink(u))
    {
        if (bohr[u].terminal) // если найдена конечная
вершина
        {
            cout << "Terminal point was found!" <<
endl;
```

```

        for (int j(0); j < 40; j++)
        {
            if (bohr[u].patternNum[j] != -1)
            {
                s.index = i -
patterns[bohr[u].patternNum[j]].size(); // запоминаем
индекс в строке

                s.patternNum =
bohr[u].patternNum[j]; // запоминаем номер паттерна
                num.push_back(s);
                cout << "\n-----\nPattern was
found in index " << s.index << ". Pattern number - "
<< s.patternNum << "\n-----\n\n";
            }
            else
                break;
        }
    }
}

void АНО() // запуск алгоритма Ахо-Корасик
{
    cout << "\n-----\nSearching patterns in text
\n\n";

    int u = 0;
    for (int i(0); i < text.size(); i++) // перебор
ВСЕХ СИМВОЛОВ ТЕКСТА
    {
        cout << "-----\nNew char: " << text.at(i)

```

```

<< endl << endl;

        u = getLink(u, alphabet.find(text[i]));
        find(u, i + 1);
    }
}

void print() // ВЫВОД результатов
{
    cout << "\n\n-----\nAnalysing found
patterns\n";
    vector <int> c(text.size(), 0);
    for (int i(0); i < num.size(); i++)
    {
        cout << "Current pattern - "<<
patterns[num[i].patternNum] << endl;
        if (num[i].index <
patternPos[num[i].patternNum] - 1)
            continue;
        c[num[i].index -
patternPos[num[i].patternNum] + 1]++;
        cout << "On index " << num[i].index -
patternPos[num[i].patternNum] + 2 << " - coincidence "
<< c[num[i].index - patternPos[num[i].patternNum] + 1]
<< " out of " << patterns.size() << endl;
        if (c[num[i].index -
patternPos[num[i].patternNum] + 1] == patterns.size()
&&
            num[i].index -
patternPos[num[i].patternNum] + 1 <= text.size() -
patternStr.size())

```

```

        cout << "Whole patter was found on "<<
num[i].index - patternPos[num[i].patternNum] + 2 <<
endl;
    }

    cout << "\n-----\nVariant 2 -- Individalization"
<< endl;
    bool flag = true;
    for (int i = 0; i < num.size(); i++)
    {

        int a, b;
        for (int j = 0; j < num.size(); j++)
        {
            if (num.at(j).index > num.at(i).index)
            {
                a = j;
                b = i;
            }
            else
            {
                a = i;
                b = j;
            }
            if (i != j && num.at(a).index <
num.at(b).index + patterns[num[b].patternNum].size())
            {
                flag = false;
                cout << "Adjacent patterns:
pattern " << num.at(b).patternNum << " and " <<

```

```

num.at(a).patternNum;

                cout << ", index " <<
num.at(b).index << " and " << num.at(a).index << endl;
            }
        }
        num.erase(num.begin());

    }
    cout << "Points number: " << bohr.size() <<
endl;

    if (flag) cout << "No adjusting patterns!\n";
    printBohr(); // ВЫВОД АВТОМАТА

}

void printBohr()
{
    cout << "\n-----\nMachine built during the
operation of the algorithm\n\n";
    for (int i = 0; i < bohr.size(); i++)
    {
        cout << "Point " << i << " with paths:
\n";

        for (int j = 0; j < 5; j++)
            if (bohr.at(i).charrLink[j] != -1)
                cout << "\tPoint " <<
bohr.at(i).charrLink[j] << " with paht " <<
alphabet[j] << endl;
    }
}

```

```
};
```

```
int main()
```

```
{
```

```
    Bohr obj;
```

```
    obj.AHO();
```

```
    obj.print();
```

```
    return 0;
```

```
}
```