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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студентка гр. 8383 Сырцова Е.А.

Преподаватель Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Изучить алгоритм Ахо-Корасик поиска множества подстрок в строке, а также реализовать данный алгоритм на языке программирования C++.

Постановка задачи.

1. Разработайте программу, решающую задачу точного поиска набора образцов.

   Входные данные:

   Первая строка содержит текст (T, $1 \le |T| \le 100000$).

   Вторая - число n ($1 \le n \le 3000$), каждая следующая из n строк содержит шаблон из набора $P = \{p_1, \ldots, p_n\}$ $1 \le |p_i| \le 75$.

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

   Выходные данные:

   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

   CCCA
   1
   CC

   Соответствующие выходные данные

   1 1
   2 1

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

   В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному

содержащему шаблоны образцу P необходимо найти все вхождения P в текст T.

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

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

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

Входные данные:

Текст (T, $1 \leq |T| \leq 100000$).

Шаблон (P, $1 \leq |P| \leq 40$).

Символ джокера.

Выходные данные:

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

Номера должны выводиться в порядке возрастания.

Пример входных данных

ACTANCA
A$$A$
$

Соответствующие выходные данные

1

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

Реализация задачи

Описание алгоритма

1. Алгоритм принимает строки-образцы и строит по ним бор следующим образом: корнем бора является корневая вершина, из которой по символу есть переход в вершину уровнем ниже. При добавлении строки, у неё перебираются все символы. Если перехода по считанному символу из текущей вершины нет, то создается новая вершина, которая является ребенком текущей, совершается переход в нее (т.е. она становится текущей). Если же переход есть, он происходит. Вершина, в которую выполняется переход по последней букве строки-образца помечается, как терминальная.

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.

Суффиксная ссылка для каждой вершины v — это вершина, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине v. Для корня бора суффиксная ссылка – ссылка на себя же. Принцип нахождения суффиксной ссылки таков: выполняется переход в предка текущей вершины, затем выполняется переход по его суффиксной ссылке, а затем переход по заданному символу.

После выполненного перехода, происходит проверка. Из текущей вершины по суффиксным ссылкам совершается проход до корня, если встречается терминальная вершина (лист) – вхождение найдено.

Алгоритм завершает работу, когда каждый символ строки-текста был обработан. Алгоритм имеет сложность по памяти O(m), где m – длина всех строк-образцов, т.к. в худшем случае в автомате хранится каждая буква строк-образцов.

Алгоритм строит бор за O(m * log(k)), где k – количество символов алфавита, m – длина всех строк-образцов т.к. в худшем случае в бор

добавляется каждая вершина, а вставка в контейнер имеет временную сложность O(log(k)). Алгоритм пройдется по всей длине текста t, получая переходы из словаря за log(k), после каждого перехода будут проверены все суффиксные ссылки до корня, которых максимально m штук, итого сложность алгоритма составит O((t+2m)*log(k)).

2. Был реализован алгоритм, который используя реализацию точного множественного поиска, решит задачу точного поиска для одного образца с джокером. Для корректной работы программы структура вершины бора была модифицирована. Теперь одна вершина может хранить информацию о нескольких шаблонах, которые в ней заканчиваются в массиве int pattern_num[40]. Для поиска в тексте шаблона с джокером, шаблон разбивается на части без джокеров, по ним строится бор, далее алгоритм идентичен предыдущему. После завершения выполняется проверка на найденные шаблоны с джокерами. Между позициями в тексте найденных последовательно подшаблонов считается количество пропущенных символов, если оно равно числу джокеров в этом месте (разность между позицией подшаблона в шаблоне и концом предыдущего подшаблона), то найден шаблон в тексте, в противном случае – не найден. Позиции найденных шаблонов сохраняются.

Также для алгоритма с джокером используется структура struct numbers, которая хранит индекс int index и номер подшаблона int pattern_num, найденного в тексте. Все найденные подшаблоны хранятся в векторе vector<numbers> num.

Описание функций и структур данных

Была использована структура для вершин бора: struct bohr_vertex.

Бор хранится как вектор вершин созданной структуры.

Переменные структуры bohr_vertex:

- int next_vertex[] – массив вершин, в которые можно попасть из данной;

- int path_num – номер текущего шаблона;
- bool flag – терминальная вершина для шаблона;
- int suff_link – переменная для хранения суффиксной ссылки;
- int auto_move[] – массив переходов из одного состояния в другое;
- int par – номер вершины родителя;
- char symbol – символ, по которому осуществляется переход от родителя;
- int suff_flink – переменная для хранения сжатой суффиксной ссылки.

Для алгоритма с джокером вместо int path_num используется

- int pattern_num[40] – массив с номерами шаблонов.

Методы структуры bohr_vertex:

1. bohr_vertex make_bohr_vertex(int par, char symbol) – функция для создания вершины бора, принимает номер вершины родителя par и символ symbol, по которому осуществляется переход от родителя. Остальные характеристики задаются -1, а метка терминальной вершины – false. Функция возвращает созданную вершину бора.

Описание используемых функций:

2. void init_bohr() – функция для создания пустого бора, создает вершину корня бора с номером -1 и символом -1.
3. int find(char symbol) – функция, хранящая алфавит допустимых символов. Принимает символ шаблона symbol и возвращает его номер для хранения.
4. char vertex(char v) – вспомогательная функция, принимает номер символа шаблона v и возвращает его отображение для вывода на консоль.
5. void info() – вспомогательная функция, выводит на консоль информацию о построенном боре и считает максимальное число дуг, исходящих от одной вершины.
6. void add_string_to_bohr(string s) – функция добавления строки s в бор. Проходит по всей строке переданного шаблона, если в боре уже есть вершина с переходом по заданному символу из текущей вершины, то переходит к ней,

если нет – создаем новую вершину и выполняем переход. Когда добавляем последнюю вершину из шаблона, помечаем её как терминальную.

7. int get_auto_move(int v, char ch) – функция вычисляемых переходов. Принимает текущую вершину v и символ ch, по которому осуществляется переход. Если такая вершина есть, осуществляется переход, если нет – выполняется переход по суффиксной ссылке. Возвращает переходов из массива от данной вершины по символу.

8. int get_suff_link(int v) – реализует получение суффиксной ссылки для переданной вершины v. Для корня бора и его дочерних вершин суффиксная ссылка равна ссылке на корень бора. Для остальных (еще не вычисленных) вершин выполняется переход по суффиксной ссылке предка по заданному символу. Полученная вершина будет суффиксной ссылкой для вершины v. Функция возвращает суффиксную ссылку вершины.

9. int get_suff_flink(int v) – реализует получение сжатой суффиксной ссылки для вершины v. Для корня бора и его дочерних вершин – равна ссылке на корень бора. Для остальных вершин, если суффиксальная ссылка на терминальную вершину – равна суффиксальной ссылке, если нет, то вычисляется рекурсивно от вершины по суффиксальной ссылке. Функция возвращает сжатую суффиксную ссылку вершины.

10. void check(int v, int i) – функция отмечает по сжатым суффиксным ссылкам вершины v шаблоны, найденные в тексте, где i – позиция найденного шаблона в тексте.

11. void find_all_pos(string s) – реализует поиск заданных шаблонов в тексте s. Проходит по всему тексту и вычисляет функции переходов для каждой вершины от корня бора, отмечает найденные шаблоны в тексте, вызывая функции get_auto_move() и check().

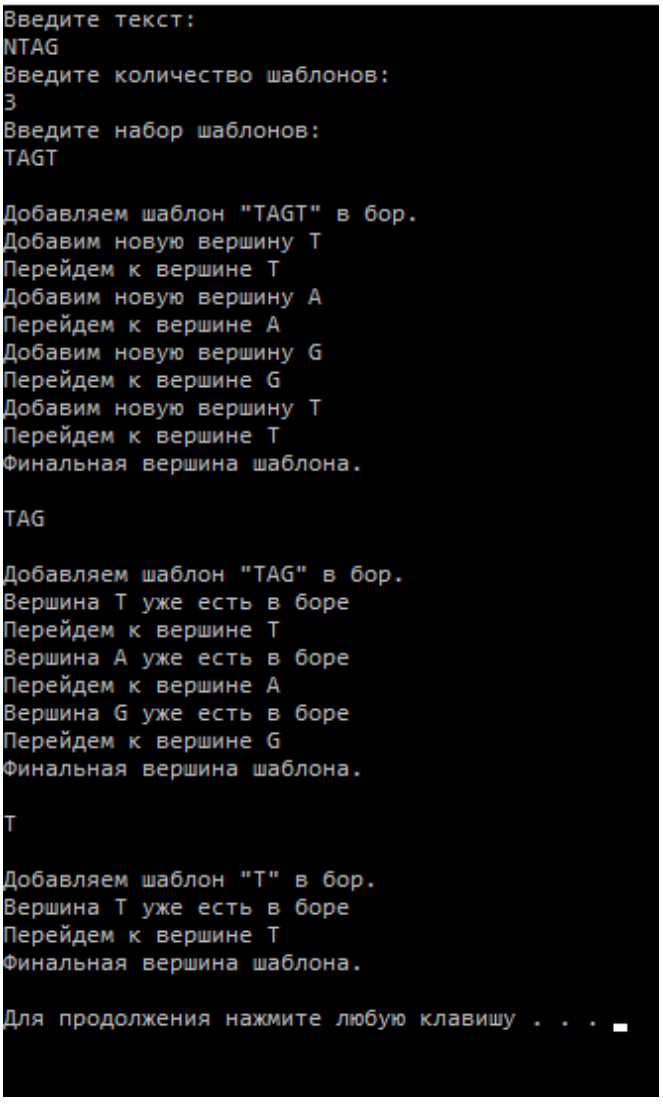
Тестирование первой программы

Входные данные	Выходные данные
NTAG 3	Ответ: Позиция в тексте/номер шаблона

TAGT TAG T	2 3 2 2 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов: N
AAAAAG 2 AAAA AG	Ответ: Позиция в тексте/номер шаблона 1 1 2 1 5 2 Максимальное количество дуг, исходящих из одной вершины 2 Строка после удаления найденных шаблонов:
ACGTATA 6 AA AC AT C G T	Ответ: Позиция в тексте/номер шаблона 1 2 2 4 3 5 4 6 5 3 6 6 Максимальное количество дуг, исходящих из одной вершины 4 Строка после удаления найденных шаблонов: A
AAAAAAAA 2 AAAA AA	Ответ: Позиция в тексте/номер шаблона 1 2 2 2 1 1 3 2 2 1 4 2 3 1 5 2 4 1 6 2 5 1 7 2 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов:
ACTGNA 1 CTGNAA	Ответ: Позиция в тексте/номер шаблона

	Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов: ACTGNA

Пример вывода промежуточных данных первой программы



```
Индивидуализация:
Вычислим количество дуг из вершин.

Вершина номер 0
Это корень бора
Соседние вершины:
Вершина 1 по символу T
Суффиксная ссылка: еще не посчитана.

Вершина номер 1
Вершина-родитель с номером 0. Переход по символу T
Соседние вершины:
Вершина 2 по символу A
Суффиксная ссылка: еще не посчитана.

Вершина номер 2
Вершина-родитель с номером 1. Переход по символу A
Соседние вершины:
Вершина 3 по символу G
Суффиксная ссылка: еще не посчитана.

Вершина номер 3
Вершина-родитель с номером 2. Переход по символу G
Соседние вершины:
Вершина 4 по символу T
Суффиксная ссылка: еще не посчитана.

Вершина номер 4
Вершина-родитель с номером 3. Переход по символу T
Соседние вершины:
Это финальная вершина.
Суффиксная ссылка: еще не посчитана.

Максимальное количество дуг, исходящих из одной вершины 1
Для продолжения нажмите любую клавишу . . .
```

```
Найдем шаблоны в тексте.

Вычислим функции переходов.

Из вершины 0 есть ребро с символом T
Переходим по этому ребру.

Найден шаблон с номером 3, позиция в тексте 2

Найдем суффиксную ссылку для вершины T
Текущая вершина - начало шаблона. Суффиксная ссылка равна 0.
Из вершины T есть ребро с символом A
Переходим по этому ребру.

Найдем суффиксную ссылку для вершины A
Пройдем по суффиксной ссылке предка T и запустим переход по символу A

Найдем суффиксную ссылку для вершины T
Суффиксная ссылка для вершины T равна A

Значит суффиксная ссылка для вершины A равна A

Из вершины A есть ребро с символом G
Переходим по этому ребру.

Найден шаблон с номером 2, позиция в тексте 2

Найдем суффиксную ссылку для вершины G
Пройдем по суффиксной ссылке предка A и запустим переход по символу G

Найдем суффиксную ссылку для вершины A
Суффиксная ссылка для вершины A равна A

Значит суффиксная ссылка для вершины G равна A

Для продолжения нажмите любую клавишу . . .
```

```
Ответ:
Позиция в тексте/номер шаблона
2 3
2 2
Строка после удаления найденных шаблонов: N
Для продолжения нажмите любую клавишу . . .
```

Тестирование второй программы

Входные данные	Выходные данные
ACTANCA A$$A$ $	Ответ: 1 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов: CA
AAACATGNA A!!!A !	Ответ: 1 5 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов:
ACATCTNCG C33C 3	Ответ: 2 5 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов: AG
ACTNGCTAACTA CTQQCT Q	Ответ: 2 6 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов: AA
AAAAAAAAA A@A @	Ответ: 1 2 3 4 5 6 7 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов:
AACNNAANN AC*N*AN *	Ответ: 2 Максимальное количество дуг, исходящих из одной вершины 2 Строка после удаления найденных шаблонов: AN

Пример вывода промежуточных данных второй программы

```
Введите текст:
ACTANCA
Введите шаблон:
A$$A$
Введите джокер:
$

Разделим шаблон на подстроки без джокера

Добавляем шаблон "A" в бор.
Добавим новую вершину A
Перейдем к вершине A
Финальная вершина шаблона.


Добавляем шаблон "A" в бор.
Вершина A уже есть в боре
Перейдем к вершине A
Финальная вершина шаблона.

Для продолжения нажмите любую клавишу . . .
------------------------------------------------------------
Индивидуализация:
Вычислим количество дуг из вершин.

Вершина номер 0
Это корень бора
Соседние вершины:
Вершина 1 по символу A
Суффиксная ссылка: еще не посчитана.

Вершина номер 1
Вершина-родитель с номером 0
Соседние вершины:
Это финальная вершина.
Суффиксная ссылка: еще не посчитана.

Максимальное количество дуг, исходящих из одной вершины 1
Для продолжения нажмите любую клавишу . . .
```

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

Найдем шаблон в тексте.

Вычислим функции переходов.

Из вершины 0 есть ребро с символом A
Переходим по этому ребру.
Найден подшаблон с номером 0, позиция в тексте 0
Найден подшаблон с номером 1, позиция в тексте 0

Найдем суффиксную ссылку для вершины A
Текущая вершина - начало шаблона. Суффиксная ссылка равна 0.
Из вершины A нет ребра с символом C
Перейдем по суффиксной ссылке.


Найдем суффиксную ссылку для вершины A
Суффиксная ссылка для вершины A равна A

Найден подшаблон с номером 0, позиция в тексте 3
Найден подшаблон с номером 1, позиция в тексте 3
Из вершины A нет ребра с символом N
Перейдем по суффиксной ссылке.


Найдем суффиксную ссылку для вершины A
Суффиксная ссылка для вершины A равна A

Найден подшаблон с номером 0, позиция в тексте 6
Найден подшаблон с номером 1, позиция в тексте 6

Ответ:
1
Строка после удаления найденных шаблонов: CA
```

Вывод

В результате работы была написана полностью рабочая программа, решающая поставленную задачу при использовании изученных теоретических материалов. Программа было протестирована, результаты тестов удовлетворительны.

ПРИЛОЖЕНИЕ А

```
#include <iostream>
#include <vector>
#include <string>
#include <cstring>
#include <algorithm>
#define ALP 6

using namespace std;

struct bohr_vertex {//структура вершины бора
	int next_vertex[ALP];//массив вершин, в которые можно попасть из данной
	int path_num;//номер шаблона
	bool flag;//финальная вершина для шаблона
	int suff_link;//переменная для хранения суффиксной ссылки
	int auto_move[ALP];//массив переходов из одного состояния в другое
	int par;//номер вершины родителя
	char symbol;//символ по которому осуществляется переход от родителя
	int suff_flink;//сжатые суффиксные ссылки
};

vector <bohr_vertex> bohr;//бор
vector <string> pattern;//шаблоны

bohr_vertex make_bohr_vertex(int par, char symbol) {//создание вершины бора
	bohr_vertex vertex;
	memset(vertex.next_vertex, 255, sizeof(vertex.next_vertex));
	vertex.flag = false;
	vertex.suff_link = -1;
	memset(vertex.auto_move, 255, sizeof(vertex.auto_move));
	vertex.par = par;
	vertex.symbol = symbol;
	vertex.suff_flink = -1;
	return vertex;
}

void init_bohr() {//создает пустой бор
	bohr.push_back(make_bohr_vertex(-1, -1));
}

int find(char symbol) {//алфавит
	int ch;
	switch (symbol)
	{
	case 'A':
		ch = 0;
		break;
```

```
	case 'C':
		ch = 1;
		break;
	case 'G':
		ch = 2;
		break;
	case 'T':
		ch = 3;
		break;
	case 'N':
		ch = 4;
		break;
	default:
		break;
	}
	return ch;
}

char vertex(char v) {//для вывода на консоль
	char ch;
	switch (v)
	{
	case 0:
		ch = 'A';
		break;
	case 1:
		ch = 'C';
		break;
	case 2:
		ch = 'G';
		break;
	case 3:
		ch = 'T';
		break;
	case 4:
		ch = 'N';
		break;
	default:
		ch = '0';
		break;
	}
	return ch;
}

void info() {
	int count[20];
	int max = 0;
	for (int i = 0; i < bohr.size(); i++) {
		count[i] = 0;
		cout << endl << "Вершина номер " << i << endl;
		if (i == 0) cout << "Это корень бора" << endl;
```

```
            else cout << "Вершина-родитель с номером " << bohr[i].par <<
". Переход по символу " << vertex(bohr[i].symbol) << endl;
            cout << "Соседние вершины:" << endl;
            for (int j = 0; j < ALP; j++) {
                  if (bohr[i].next_vertex[j] != -1) {
                        cout << "Вершина " << bohr[i].next_vertex[j] << "
по символу " << vertex(bohr[bohr[i].next_vertex[j]].symbol) << endl;
                        count[i]++;
                  }
            }
            if (count[i] == 0) cout << "Это финальная вершина." << endl;
            cout << "Суффиксная ссылка: ";
            if (bohr[i].suff_link == -1) cout << "еще не посчитана." <<
endl;
            else cout << vertex(bohr[i].suff_link) << endl;
      }
      for (int i = 0; i < bohr.size(); i++)
            if (count[i] > max) max = count[i];
      cout << endl << "Максимальное количество дуг, исходящих из одной
вершины " << max << endl;
}

void add_string_to_bohr(string s) {//вставляет строку в бор
      cout << endl << "Добавляем шаблон \"" << s << "\" в бор." <<
endl;
      int num = 0;
      for (int i = 0; i < s.length(); i++) {//проходится по строке
            char ch = find(s[i]);//находит номер символа
            if (bohr[num].next_vertex[ch] == -1) {//добавляется новая
вершина если её не было
                  cout << "Добавим новую вершину " << s[i] << endl;
                  bohr.push_back(make_bohr_vertex(num, ch));
                  bohr[num].next_vertex[ch] = bohr.size() - 1;
            }
            else cout << "Вершина " << s[i] << " уже есть в боре" <<
endl;
            cout << "Перейдем к вершине " << s[i] << endl;
            num = bohr[num].next_vertex[ch];//переходим к следующей
вершине
      }
      cout << "Финальная вершина шаблона." << endl << endl;
      bohr[num].flag = true;
      pattern.push_back(s);
      bohr[num].path_num = pattern.size() - 1;
}

int get_auto_move(int v, char ch);

int get_suff_link(int v) {//реализует получение суффиксной ссылки для
данной вершины
```

```
	cout << endl << "Найдем суффиксную ссылку для вершины " <<
vertex(bohr[v].symbol) << endl;
	if (bohr[v].suff_link == -1) {
		if (v == 0 || bohr[v].par == 0) {//если это корень или
начало шаблона
			if (v==0) cout << "Текущая вершина - корень бора.
Суффиксная ссылка равна 0." << endl;
			else cout << "Текущая вершина - начало шаблона.
Суффиксная ссылка равна 0." << endl;
			bohr[v].suff_link = 0;
		}
		else {
			cout << "Пройдем по суффиксной ссылке предка " <<
vertex(bohr[bohr[v].par].symbol) << " и запустим переход по символу "
<< vertex(bohr[v].symbol) << endl;
			bohr[v].suff_link =
get_auto_move(get_suff_link(bohr[v].par), bohr[v].symbol);  //пройдем
по суф.ссылке предка и запустим переход по символу.
			cout << "Значит суффиксная ссылка для вершины " <<
vertex(bohr[v].symbol) << " равна " << vertex(bohr[v].suff_link) <<
endl << endl;
		}
	}
	else cout << "Суффиксная ссылка для вершины " <<
vertex(bohr[v].symbol) << " равна " << vertex(bohr[v].suff_link) <<
endl << endl;
	return bohr[v].suff_link;
}

int get_auto_move(int v, char ch) {			//вычисляемая функция
переходов
	if (bohr[v].auto_move[ch] == -1) {
		if (bohr[v].next_vertex[ch] != -1) {			//если из
текущей вершины есть ребро с символом ch
			cout << "Из вершины " << vertex(bohr[v].symbol) << "
есть ребро с символом " << vertex(ch) << endl;
			cout << "Переходим по этому ребру." << endl;
			bohr[v].auto_move[ch] = bohr[v].next_vertex[ch];
//то идем по нему
		}
		else {//если нет
			if (v == 0) {//если это корень бора
				//cout << "Текущая вершина " <<
vertex(bohr[v].symbol) << " - корень бора." << endl;
				bohr[v].auto_move[ch] = 0;
			}
			else {
				cout << "Из вершины " << vertex(bohr[v].symbol)
<< " нет ребра с символом " << vertex(ch) << endl;
				cout << "Перейдем по суффиксной ссылке." << endl
<< endl;
```

```
                    bohr[v].auto_move[ch] =
get_auto_move(get_suff_link(v), ch);  //иначе перейдем по
суффиксальной ссылке
                }
            }
        }
        return bohr[v].auto_move[ch];
}

int get_suff_flink(int v) {//функция вычисления сжатых суффиксальных
ссылок
        if (bohr[v].suff_flink == -1) {
            int u = get_suff_link(v);
            if (u == 0) {//если корень или начало шаблона
                bohr[v].suff_flink = 0;
            }
            else {//если по суффиксальной ссылке конечная вершина-равен
суффиксальной ссылке, если нет-рекурсия.
                bohr[v].suff_flink = (bohr[u].flag) ? u :
get_suff_flink(u);
            }
        }
        return bohr[v].suff_flink;
}

void check(int v, int i) {
        for (int u = v; u != 0; u = get_suff_flink(u)) {
            if (bohr[u].flag) {
                cout << endl << "Найден шаблон с номером " <<
bohr[u].path_num + 1 << ", позиция в тексте " << i -
pattern[bohr[u].path_num].length() + 1 << endl;
            }
        }
}

void find_all_pos(string s) {//поиск шаблонов в строке
        int u = 0;//текущая вершина
        int n = 0;
        int arr[20][2];
        cout << endl << "Вычислим функции переходов." << endl << endl;
        for (int i = 0; i < s.length(); i++) {
            u = get_auto_move(u, find(s[i]));

            if (bohr[u].flag) {
                arr[n][0] = i - pattern[bohr[u].path_num].length() +
2;
                arr[n][1] = bohr[u].path_num + 1;
                n++;
            }
```

```
            check(u, i + 1);//отмечаем по сжатым суффиксным ссылкам
строки, которые нам встретились и их позицию
      }
      system("pause");
      cout << "-----------------------------------------------------
----------------------" << endl;
      cout << endl << "Ответ:" << endl << "Позиция в тексте/номер
шаблона" << endl;
      for (int i = 0; i < n; i++)
            cout << arr[i][0] << " " << arr[i][1] << endl;
      for (int i = n-1; i >=0; i--) {
            string::const_iterator sub = find_end(s.begin(), s.end(),
(pattern[arr[i][1] - 1]).begin(), (pattern[arr[i][1] - 1]).end());
            if (sub != s.end()) {
                  s.erase(sub, sub + pattern[arr[i][1] -
1].size());
            }
      }
      cout << "Строка после удаления найденных шаблонов: " << s <<
endl;
}

int main() {
      setlocale(LC_ALL, "Russian");
      string text;
      int n;//количество шаблонов
      init_bohr();
      cout << "Введите текст:" << endl;
      cin >> text;
      cout << "Введите количество шаблонов:" << endl;
      cin >> n;
      cout << "Введите набор шаблонов:" << endl;
      for (int i = 0; i < n; i++) {
            string temp;//шаблон
            cin >> temp;
            add_string_to_bohr(temp);
      }
      system("pause");
      cout << "-----------------------------------------------------
----------------------" << endl;
      cout << "Индивидуализация:\nВычислим количество дуг из вершин."
<< endl;
      info();
      system("pause");
      cout << "-----------------------------------------------------
----------------------" << endl;
      cout << endl << "Найдем шаблоны в тексте." << endl;
      find_all_pos(text);
      system("pause");
      return 0;
}
```

ПРИЛОЖЕНИЕ В

```
#include <iostream>
#include <vector>
#include <string>
#include <cstring>
#include <algorithm>
#define ALP 6

using namespace std;

struct numbers {
	int index;
	int pattern_num;
};

struct bohr_vertex {
	int next_vertex[ALP];//массив вершин, в которые можно попасть из данной
	bool flag;//финальная вершина для шаблона
	int suff_link;//переменная для хранения суффиксной ссылки
	int auto_move[ALP];//массив переходов из одного состояния в другое
	int par;//номер вершины родителя
	char symbol;//символ по которому осуществляется переход от родителя
	int suff_flink;//сжатые суффиксные ссылки
	int pattern_num[40];//номера подшаблонов
};

vector<numbers> num;
vector <bohr_vertex> bohr;//бор
vector <string> pattern;//шаблоны

bohr_vertex make_bohr_vertex(int par, char symbol) {//создание вершины бора
	bohr_vertex vertex;
	memset(vertex.next_vertex, 255, sizeof(vertex.next_vertex));
	vertex.flag = false;
	vertex.suff_link = -1;
	memset(vertex.auto_move, 255, sizeof(vertex.auto_move));
	vertex.par = par;
	vertex.symbol = symbol;
	vertex.suff_flink = -1;
	memset(vertex.pattern_num, 255, sizeof(vertex.pattern_num));
	return vertex;
}

void init_bohr() {//создание пустого бора
	bohr.push_back(make_bohr_vertex(-1, -1));
}
```

```
int find(char symbol) {//алфавит
	int ch;
	switch (symbol)
	{
	case 'A':
		ch = 0;
		break;
	case 'C':
		ch = 1;
		break;
	case'G':
		ch = 2;
		break;
	case 'T':
		ch = 3;
		break;
	case 'N':
		ch = 4;
		break;
	default:
		break;
	}
	return ch;
}

char vertex(char v) {//для вывода на консоль
	char ch;
	switch (v)
	{
	case 0:
		ch = 'A';
		break;
	case 1:
		ch = 'C';
		break;
	case 2:
		ch = 'G';
		break;
	case 3:
		ch = 'T';
		break;
	case 4:
		ch = 'N';
		break;
	default:
		ch = '0';
		break;
	}
	return ch;
}
```

```
void info() {
	int count[20];
	int max = 0;
	for (int i = 0; i < bohr.size(); i++) {
		count[i] = 0;
		cout << endl << "Вершина номер " << i << endl;
		if (i == 0) cout << "Это корень бора" << endl;
		else cout << "Вершина-родитель с номером " << bohr[i].par <<
endl;
		cout << "Соседние вершины:" << endl;
		for (int j = 0; j < ALP; j++) {
			if (bohr[i].next_vertex[j] != -1) {
				cout << "Вершина " << bohr[i].next_vertex[j] << "
по символу " << vertex(bohr[bohr[i].next_vertex[j]].symbol) << endl;
				count[i]++;
			}
		}
		if (count[i] == 0) cout << "Это финальная вершина." << endl;
		cout << "Суффиксная ссылка: ";
		if (bohr[i].suff_link == -1) cout << "еще не посчитана." <<
endl;
		else cout << vertex(bohr[i].suff_link) << endl;
	}
	for (int i = 0; i < bohr.size(); i++)
		if (count[i] > max) max = count[i];
	cout << endl << "Максимальное количество дуг, исходящих из одной
вершины " << max << endl;
}

void add_string_to_bohr(string s) {//вставляет строку в бор
	cout << endl << "Добавляем шаблон \"" << s << "\" в бор." <<
endl;
	int num = 0;
	for (int i = 0; i < s.length(); i++) {//проходится по строке
		char ch = find(s[i]);//находит номер символа
		if (bohr[num].next_vertex[ch] == -1) {//добавляется новая
вершина если её не было
			cout << "Добавим новую вершину " << s[i] << endl;
			bohr.push_back(make_bohr_vertex(num, ch));
			bohr[num].next_vertex[ch] = bohr.size() - 1;
		}
		else cout << "Вершина " << s[i] << " уже есть в боре" <<
endl;
		cout << "Перейдем к вершине " << s[i] << endl;
		num = bohr[num].next_vertex[ch];//переходим к следующей
вершине
	}
	cout << "Финальная вершина шаблона." << endl << endl;
	bohr[num].flag = true;
	pattern.push_back(s);
```

```
	for (int i = 0; i < 40; i++) {
		if (bohr[num].pattern_num[i] == -1) {
			bohr[num].pattern_num[i] = pattern.size() - 1;
			break;
		}
	}
}

int get_auto_move(int v, char ch);

int get_suff_link(int v) {//реализует получение суффиксной ссылки для данной вершины
	cout << endl << "Найдем суффиксную ссылку для вершины " << vertex(bohr[v].symbol) << endl;
	if (bohr[v].suff_link == -1) {
		if (v == 0 || bohr[v].par == 0) {//если это корень или начало шаблона
			if (v == 0) cout << "Текущая вершина - корень бора. Суффиксная ссылка равна 0." << endl;
			else cout << "Текущая вершина - начало шаблона. Суффиксная ссылка равна 0." << endl;
			bohr[v].suff_link = 0;
		}
		else {
			cout << "Пройдем по суффиксной ссылке предка " << vertex(bohr[bohr[v].par].symbol) << " и запустим переход по символу " << vertex(bohr[v].symbol) << endl;
			bohr[v].suff_link = get_auto_move(get_suff_link(bohr[v].par), bohr[v].symbol);  //пройдем по суф.ссылке предка и запустим переход по символу.
			cout << "Значит суффиксная ссылка для вершины " << vertex(bohr[v].symbol) << " равна " << vertex(bohr[v].suff_link) << endl << endl;
		}
	}
	else cout << "Суффиксная ссылка для вершины " << vertex(bohr[v].symbol) << " равна " << vertex(bohr[v].suff_link) << endl << endl;
	return bohr[v].suff_link;
}

int get_auto_move(int v, char ch) {			//вычисляемая функция переходов
	if (bohr[v].auto_move[ch] == -1) {
		if (bohr[v].next_vertex[ch] != -1) {			//если из текущей вершины есть ребро с символом ch
			cout << "Из вершины " << vertex(bohr[v].symbol) << " есть ребро с символом " << vertex(ch) << endl;
			cout << "Переходим по этому ребру." << endl;
			bohr[v].auto_move[ch] = bohr[v].next_vertex[ch];
//то идем по нему
```

```
            }
            else {//если нет
                  if (v == 0) {//если это корень бора
                        //cout << "Текущая вершина " <<
vertex(bohr[v].symbol) << " - корень бора." << endl;
                        bohr[v].auto_move[ch] = 0;
                  }
                  else {
                        cout << "Из вершины " << vertex(bohr[v].symbol)
<< " нет ребра с символом " << vertex(ch) << endl;
                        cout << "Перейдем по суффиксной ссылке." << endl
<< endl;
                        bohr[v].auto_move[ch] =
get_auto_move(get_suff_link(v), ch);  //иначе перейдем по
суффиксальной ссылке
                  }
            }
      }
      return bohr[v].auto_move[ch];
}

int get_suff_flink(int v) {//функция вычисления сжатых суффиксальных
ссылок
      if (bohr[v].suff_flink == -1) {
            int u = get_suff_link(v);
            if (u == 0) {//если корень или начало шаблона
                  bohr[v].suff_flink = 0;
            }
            else {//если по суффиксальной ссылке конечная вершина-равен
суффиксальной ссылке, если нет-рекурсия.
                  bohr[v].suff_flink = (bohr[u].flag) ? u :
get_suff_flink(u);
            }
      }
      return bohr[v].suff_flink;
}

void check(int v, int i) {
      struct numbers s;
      for (int u = v; u != 0; u = get_suff_flink(u)) {
            if (bohr[u].flag) {
                  for (int j = 0; j < 40; j++) {
                        if (bohr[u].pattern_num[j] != -1) {
                              s.index = i -
pattern[bohr[u].pattern_num[j]].length();
                              s.pattern_num = bohr[u].pattern_num[j];
                              cout << "Найден подшаблон с номером " <<
s.pattern_num << ", позиция в тексте " << s.index << endl;
                              num.push_back(s);
                        }
                        else
```

```
                    break;
                }
            }
        }
    }

    void find_all_pos(string s) {
        int u = 0;
        cout << endl << "Вычислим функции переходов." << endl << endl;
        for (int i = 0; i < s.length(); i++) {
            u = get_auto_move(u, find(s[i]));
            check(u, i + 1);
        }
    }

    int main() {
        setlocale(LC_ALL, "Russian");
        vector<string> patterns;  //подстроки при делении по джокеру
        vector<int> patterns_pos; //позиции подстрок
        string text;//текст
        string temp;//шаблон
        char joker;//джокер
        string pat;
        cout << "Введите текст:" << endl;
        cin >> text;
        cout << "Введите шаблон:" << endl;
        cin >> temp;
        cout << "Введите джокер:" << endl;
        cin >> joker;
        init_bohr();
        cout << endl << "Разделим шаблон на подстроки без джокера" <<
    endl;
        for (int i = 0; i < temp.length(); i++) {
            if (temp[i] != joker) {
                patterns_pos.push_back(i + 1);
                for (int j = i; temp[j] != joker && j !=
    temp.length(); j++) {
                    pat += temp[j];
                    i++;
                }
                add_string_to_bohr(pat);
                patterns.push_back(pat);
                pat.clear();
            }
        }
        system("pause");
        cout << "-------------------------------------------------------
    ----------------------" << endl;
        cout << "Индивидуализация:\nВычислим количество дуг из вершин."
    << endl;
        info();
```

```
	system("pause");
	cout << "-------------------------------------------------------
----------------------" << endl;
	cout << endl << "Найдем шаблон в тексте." << endl;
	find_all_pos(text);
	int arr[10];
	int n = 0;
	vector<int> c(text.length(), 0);
	cout << endl << "Ответ:" << endl;
	for (int i = 0; i < num.size(); i++) {
		if (num[i].index < patterns_pos[num[i].pattern_num] - 1)
continue;//если индекс подшаблона меньше позиции подшаблона в строке
		c[num[i].index - patterns_pos[num[i].pattern_num] +
1]++;//увеличиваем счетчик количества пропусков
		if (c[num[i].index - patterns_pos[num[i].pattern_num] + 1]
== patterns.size() && //если количество пропусков равно колличеству
джокеров
			num[i].index - patterns_pos[num[i].pattern_num] + 1 <=
text.length() - temp.length()) { //и есть место чтобы закончить шаблон
			cout << num[i].index -
patterns_pos[num[i].pattern_num] + 2 << endl;
			arr[n] = num[i].index -
patterns_pos[num[i].pattern_num] + 2;
			n++;
		}
	}
	for (int i = n - 1; i >= 0; i--) {
		if (i != 0 && arr[i - 1] + temp.size() - 1 >= arr[i])
			text.erase(arr[i - 1] + temp.size() - 1, arr[i]-arr[i-
1] );
		else
			text.erase(arr[i] - 1, temp.size());
	}
	cout << "Строка после удаления найденных шаблонов: " << text <<
endl;
	return 0;
}
```