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

Студент гр. 9382 Сорокумов С.В.

Преподаватель Фирсов М.А.

Санкт-Петербург

2021

Цель работы

Изучить алгоритм Ахо-Корасик поиска множества подстрок в строке, а также реализовать данный алгоритм.

Постановка задачи.

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

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

   Выход:

   Все вхождения образцов из P в T.

   Каждое вхождение образца в текст представить в виде двух чисел - i p, где i - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p (нумерация образцов начинается с 1).

   Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

   Sample Input:

   NTAG

   3

   TAGT

   TAG

   T

   Sample Output:

   2 2

   2 3

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 $\{A, C, G, T, N\}$

Вход:

Текст (T, $1 \le |T| \le 100000$).

Шаблон (P, $1 \le |P| \le 40$).

Символ джокера.

Выход:

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

Номера должны выводиться в порядке возрастания.

Sample Input:

ACTANCA

A$$A$

$

Sample Output:

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.

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.

Для вывода ответов, отсортированных по возрастанию, сначала номера позиции, затем номера шаблона, используется вектор ответов `std::vector< std::pair<int, int>> answer`, где первое значение – номер позиции, а второе – номер вектора. Этот вектор сортируется с помощью библиотечной функции sort и компилятора cmp.

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 `std::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 num` – номер текущего шаблона;

`bool flag` – терминальная вершина для шаблона;

`int link` – переменная для хранения суффиксной ссылки;

`int move[]` – массив переходов из одного состояния в другое;

`int par` – номер вершины родителя;

`char symbol` – символ, по которому осуществляется переход от родителя;

`int flink` – переменная для хранения сжатой суффиксной ссылки.

Для алгоритма с джокером вместо `int num` используется

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

Так же для работы с `bohr_vertex` был реализовал класс `Borh`

Методы класса `Borh`:

1. `bohr_vertex make_vertex(int par, char symbol)` – метод 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. Метод возвращает созданную вершину бора.

`Borh()` – конструктор, который создаёт пустой бор, создает вершину корня бора с номером -1 и символом -1.

2. `int find_letter(char symbol)` – метод, хранящая алфавит допустимых символов. Принимает символ шаблона `symbol` и возвращает его номер для хранения.

3. `char vertex(char v)` – вспомогательный метод, принимает номер символа шаблона `v` и возвращает его отображение для вывода на консоль.

4. `void find_count_of_arch()` – вспомогательный метод, выводит на консоль информацию о построенном боре и считает максимальное число дуг, исходящих от одной вершины.

5. `void add_string_to_bohr(std::string s)` – метод 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.

6. `int get_auto_move(int v, char ch)` – метод 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.

7. `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`. Метод возвращает суффиксную ссылку вершины.

8. `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. Метод возвращает сжатую суффиксную ссылку вершины.

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

10. `void find_all_possitions(std::string str)` – 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 методы `get_auto_move()` и `check()`.

Тестирование первой программы

Входные данные	Выходные данные
NTAG 3 TAGT TAG T	Ответ: Позиция в тексте/номер шаблона 2 2 2 3 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов: N
AACCTTNN 2 AACCTTG TTNN	Ответ: Позиция в тексте/номер шаблона 5 2 Максимальное количество дуг, исходящих из одной вершины 2 Строка после удаления найденных шаблонов: AACC
TTTTTTT 4 TTT T TTTT TTGTT	Позиция в тексте/номер шаблона 1 1 1 2 1 3 2 1 2 2 2 3 3 1 3 2 3 3 4 1 4 2 4 3

	5 1 5 2 6 2 7 2 Максимальное количество дуг, исходящих из одной вершины 2 Строка после удаления найденных шаблонов:
ACTNACNT 4 CAN NT TN CNT	Ответ: Позиция в тексте/номер шаблона 3 3 6 4 7 2 Максимальное количество дуг, исходящих из одной вершины 3 Строка после удаления найденных шаблонов: ACA

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
ACANTAATT A@ @	Ответ: 1 3 6 7 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов: TT
TTTTTACTNNTTTT!!!!T !	Ответ: 3 8 Максимальное количество дуг, исходящих из одной вершины 1 Строка после удаления найденных шаблонов: TT
AACNNAANN AC*N*AN *	Ответ: 2 Максимальное количество дуг, исходящих из одной вершины 2 Строка после удаления найденных шаблонов: AN

Вывод

Был изучен алгоритм Ахо-Корасик поиска множества подстрок в строке, а также реализован данный алгоритм.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Название файла: 1.cpp

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

int compare(std::pair<int, int> a, std::pair<int, int>  b) {
    if (a.first == b.first)
        return a.second < b.second;
    else
        return a.first < b.first;
}

struct bohr_vertex {//структура вершины бора
    int next_vertex[6];//массив вершин, в которые можно попасть из
данной
    int num;//номер шаблона
    bool flag;//финальная вершина для шаблона
    int link;//переменная для хранения суффиксной ссылки
    int move[6];//массив переходов из одного состояния в другое
    int par;//номер вершины родителя
    char symbol;//символ по которому осуществляется переход от
родителя
    int flink;//сжатые суффиксные ссылки
};

class Bohr{
public:
    Bohr() {
        bohr.push_back(make_vertex(-1, -1));
    }
    void find_count_of_arch() {
        int count[20];
        max_count_of_arch = 0;
        for (int i = 0; i < bohr.size(); i++) {
            count[i] = 0;
            std::cout << std::endl << "Вершина номер " << i <<
std::endl;
            if (i == 0) std::cout << "Это корень бора" << std::endl;
            else std::cout << "Вершина-родитель с номером " <<
bohr[i].par << " по символу " << vertex(bohr[i].symbol) << std::endl;
            std::cout << "Соседние вершины:" << std::endl;
            for (int j = 0; j < 6; j++) {
                if (bohr[i].next_vertex[j] != -1) {
                    std::cout << "Вершина " << bohr[i].next_vertex[j]
<< " по символу " << vertex(bohr[bohr[i].next_vertex[j]].symbol) <<
std::endl;
```

```
                    count[i]++;
                }
            }
            if (count[i] == 0) std::cout << "Это финальная вершина."
<< std::endl;
            std::cout << "Суффиксная ссылка: ";
            if (bohr[i].link == -1) std::cout << "еще не посчитана."
<< std::endl;
            else std::cout << vertex(bohr[i].link) << std::endl;
        }
        for (int i = 0; i < bohr.size(); i++)
            if (count[i] > max_count_of_arch) max_count_of_arch =
count[i];

    }

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

    void find_all_possitions(std::string str) {//поиск шаблонов в
строке
        int current = 0;//текущая вершина
        for (int i = 0; i < str.length(); i++) {
            std::cout << std::endl << "Символ текста " << str[i] << "
с индексом " << i + 1 << std::endl;
            std::cout << std::endl << "Текущая вершина " << current <<
std::endl << "Вычислим функцию переходов." << std::endl << std::endl;
```

```
            current = get_auto_move(current, find_letter(str[i]));
            if (current != 0) std::cout << "Переход к вершине " <<
current << " по символу " << vertex(bohr[current].symbol) <<
std::endl;
            else std::cout << "Из вершины " << current << " нет ребра
с символом " << str[i] << std::endl;
            for (int v = current; v != 0; v = get_suff_flink(v)) {
                if (bohr[v].flag) {
                    std::pair<int, int> res(i -
pattern[bohr[v].num].length() + 2, bohr[v].num + 1);
                    answer.push_back(res);
                }
            }
            std::cout << std::endl << "Перейдем по хорошим суффиксным
ссылкам вершины  " << current;
            if (i + 1 != str.length()) std::cout << " по символу " <<
str[i + 1] << std::endl;
            check(current, i + 1);//отмечаем по сжатым суффиксным
ссылкам строки, которые нам встретились и их позицию
        }
        std::cout << std::endl << "Проход по строке текста завершен."
<< std::endl;
        std::cout << "----------------------------------------------
------------------------------" << std::endl;
        for (int i = answer.size() -1; i >=0; i--) {
            std::string::const_iterator sub = find_end(str.begin(),
str.end(), (pattern[answer[i].second - 1]).begin(),
(pattern[answer[i].second - 1]).end());
            if (sub != str.end()) {
                if (i != 0 && answer[i - 1].first + pattern[answer[i -
1].second - 1].size() - 1 >= answer[i].first) {
                    str.erase(sub + answer[i - 1].first +
pattern[answer[i - 1].second - 1].size() - answer[i].first, sub +
pattern[answer[i].second - 1].size());
                }
                else
                    str.erase(sub, sub + pattern[answer[i].second -
1].size());
            }
        }
        sort(answer.begin(), answer.end(), compare);
        std::cout << std::endl << "Ответ:" << std::endl << "Позиция в
тексте/номер шаблона" << std::endl;
        for (int i = 0; i < answer.size(); i++)
            std::cout << answer[i].first << " " << answer[i].second <<
std::endl;
        std::cout << std::endl << "Максимальное количество дуг,
исходящих из одной вершины " << max_count_of_arch << std::endl;
        std::cout << "Строка после удаления найденных шаблонов: " <<
str << std::endl;
    }
```

```
private:
    int max_count_of_arch;
    bohr_vertex make_vertex(int par, char symbol) {//создание вершины
бора
        bohr_vertex vertex;
        memset(vertex.next_vertex, 255, sizeof(vertex.next_vertex));
        vertex.flag = false;
        vertex.link = -1;
        memset(vertex.move, 255, sizeof(vertex.move));
        vertex.par = par;
        vertex.symbol = symbol;
        vertex.flink = -1;
        return vertex;
    }
    int find_letter(char symbol) {
        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;
    }

    int get_auto_move(int v, char ch) {             //вычисляемая
функция переходов
        if (bohr[v].move[ch] == -1) {
            if (bohr[v].next_vertex[ch] != -1) {            //если из
текущей вершины есть ребро с символом ch
                std::cout << "Из вершины " << v << " есть ребро с
символом " << vertex(ch) << std::endl;
                std::cout << "Переходим по этому ребру в вершину " <<
bohr[v].next_vertex[ch] << std::endl;
                bohr[v].move[ch] = bohr[v].next_vertex[ch];    //то
идем по нему
            }
            else {//если нет
                if (v == 0) {//если это корень бора
                    bohr[v].move[ch] = 0;
                }
                else {
                    std::cout << "Перейдем по суффиксной ссылке." <<
std::endl << std::endl;
                    bohr[v].move[ch] = get_auto_move(get_suff_link(v),
ch);  //иначе перейдем по суффиксальной ссылке
                }
            }
        }
        return bohr[v].move[ch];
    }

    int get_suff_link(int v) {//реализует получение суффиксной ссылки
для данной вершины
        std::cout << std::endl << "Найдем суффиксную ссылку для
вершины " << v << std::endl;
        if (bohr[v].link == -1) {
            if (v == 0 || bohr[v].par == 0) {//если это корень или
начало шаблона
                if (v == 0) std::cout << "Текущая вершина - корень
бора. Суффиксная ссылка равна 0." << std::endl;
```

```
                else std::cout << "Текущая вершина - начало шаблона.
Суффиксная ссылка равна 0." << std::endl;
                bohr[v].link = 0;
            }
            else {
                std::cout << "Пройдем по суффиксной ссылке предка " <<
bohr[v].par << " и запустим переход по символу " <<
vertex(bohr[v].symbol) << std::endl;
                bohr[v].link =
get_auto_move(get_suff_link(bohr[v].par), bohr[v].symbol);  //пройдем
по суф.ссылке предка и запустим переход по символу.
                std::cout << "Значит суффиксная ссылка для вершины "
<< v << " равна " << bohr[v].link << std::endl << std::endl;
            }
        }
        else std::cout << "Суффиксная ссылка для вершины " << v << "
равна " << bohr[v].link << std::endl << std::endl;
        return bohr[v].link;
    }

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

    void check(int v, int i) {
        for (int u = v; u != 0; u = get_suff_flink(u)) {
            if (bohr[u].flag) {
                std::cout << std::endl << "Вершина " << u << "
конечная для шаблона " << bohr[u].num + 1 << std::endl;
                std::cout << std::endl << "Найден шаблон с номером "
<< bohr[u].num + 1 << ", позиция в тексте " << i -
pattern[bohr[u].num].length() + 1 << std::endl;
            }
            else std::cout << std::endl << "Вершина " << u << " не
конечная" << std::endl;
        }
    }

    std::vector<std::pair<int, int>> answer;
```

```
    std::vector <bohr_vertex> bohr;//бор
    std::vector <std::string> pattern;//шаблоны
};


int main() {
    std::string text;
    int n;//количество шаблонов
    Bohr bohr;
    std::cout << "Введите текст:" << std::endl;
    std::cin >> text;
    std::cout << "Введите количество шаблонов:" << std::endl;
    std::cin >> n;
    std::cout << "Введите набор шаблонов:" << std::endl;
    for (int i = 0; i < n; i++) {
        std::string temp;//шаблон
        std::cin >> temp;
        bohr.add_string_to_bohr(temp);
    }
    std::cout << "Вычисление количество дуг из вершин." << std::endl;
    bohr.find_count_of_arch();
    std::cout << std::endl << "Нахождение шаблонов в тексте." <<
std::endl;
    bohr.find_all_possitions(text);
    return 0;
}
```

Название файла: 2.cpp

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

struct numbers {
    int index;
    int pattern_num;
};


struct bohr_vertex {
    int next_vertex[6];//массив вершин, в которые можно попасть из
данной
    bool flag;//финальная вершина для шаблона
    int link;//переменная для хранения суффиксной ссылки
    int move[6];//массив переходов из одного состояния в другое
    int par;//номер вершины родителя
    char symbol;//символ по которому осуществляется переход от
родителя
    int flink;//сжатые суффиксные ссылки
    int pattern_num[40];//номера подшаблонов
};

class Borh{
public:
    Borh(){
        bohr.push_back(make_vertex(-1, -1));
    }
    void find_count_of_arch() {
        int count[20];
        max_count_of_arch = 0;
        for (int i = 0; i < bohr.size(); i++) {
            count[i] = 0;
            std::cout << std::endl << "Вершина номер " << i <<
std::endl;
            if (i == 0) std::cout << "Это корень бора" << std::endl;
            else std::cout << "Вершина-родитель с номером " <<
bohr[i].par << " по символу " << vertex(bohr[i].symbol) << std::endl;
            std::cout << "Соседние вершины:" << std::endl;
            for (int j = 0; j < 6; j++) {
                if (bohr[i].next_vertex[j] != -1) {
                    std::cout << "Вершина " << bohr[i].next_vertex[j]
<< " по символу " << vertex(bohr[bohr[i].next_vertex[j]].symbol) <<
std::endl;
                    count[i]++;
                }
            }
```

```
            if (count[i] == 0) std::cout << "Это финальная вершина."
<< std::endl;
            std::cout << "Суффиксная ссылка: ";
            if (bohr[i].link == -1) std::cout << "еще не посчитана."
<< std::endl;
            else std::cout << vertex(bohr[i].link) << std::endl;
        }
        for (int i = 0; i < bohr.size(); i++)
            if (count[i] > max_count_of_arch) max_count_of_arch =
count[i];
    }

    void add_string_to_bohr(std::string s) {//вставляет строку в бор
        std::cout << std::endl << "Добавляем шаблон \"" << s << "\" в
бор." << std::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
                bohr.push_back(make_vertex(num, ch));
                bohr[num].next_vertex[ch] = bohr.size() - 1;
                std::cout << "Добавим новую вершину " <<
bohr[num].next_vertex[ch] << " по символу " << s[i] << std::endl;
            }
            else std::cout << "Вершина по символу " << s[i] << " уже
есть в боре" << std::endl;
            std::cout << "Перейдем к вершине " <<
bohr[num].next_vertex[ch] << std::endl;
            num = bohr[num].next_vertex[ch];//переходим к следующей
вершине
        }
        std::cout << "Финальная вершина шаблона." << std::endl <<
std::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;
            }
        }
    }

    void find_all_pos(std::string s) {
        int u = 0;
        for (int i = 0; i < s.length(); i++) {
            std::cout << std::endl << "Символ текста " << s[i] << " с
индексом " << i + 1 << std::endl;
            std::cout << std::endl << "Текущая вершина " << u <<
std::endl << "Вычислим функцию переходов." << std::endl << std::endl;
```

```
            u = get_auto_move(u, find(s[i]));
            if (u != 0) std::cout << "Переход к вершине " << u << " по
символу " << vertex(bohr[u].symbol) << std::endl;
            else std::cout << "Из вершины " << u << " нет ребра с
символом " << s[i] << std::endl;
            std::cout << std::endl << "Перейдем по хорошим суффиксным
ссылкам вершины  " << u;
            if (i+1!= s.length()) std::cout << " по символу " << s[i +
1] << std::endl;
            check(u, i + 1);
        }
        std::cout << std::endl << "Проход по строке текста завершен."
<< std::endl;
    }

    void answer(std::vector<std::string> patterns, std::vector<int>
patterns_pos, std::string text, std::string temp){
        int arr[10];
        int n = 0;
        std::vector<int> c(text.length(), 0);

        std::cout << "-----------------------------------------------
------------------------------" << std::endl;
        std::cout << std::endl << "Найдем шаблон с джокерами в тексте"
<< std::endl;
        for (int i = 0; i < num.size(); i++) {
            std::cout << std::endl << "Текущий подшаблон " <<
pattern[num[i].pattern_num] << " с индексом в тексте " << num[i].index
+ 1 << " и индексом в шаблоне " << patterns_pos[num[i].pattern_num] <<
std::endl;
            if (num[i].index < patterns_pos[num[i].pattern_num] - 1)
{//если индекс подшаблона меньше позиции подшаблона в строке
                std::cout << "индекс подшаблона равен " <<
num[i].index + 1 << " и меньше позиции подшаблона в шаблоне " <<
patterns_pos[num[i].pattern_num] << std::endl;
                continue;
            }
            c[num[i].index - patterns_pos[num[i].pattern_num] +
1]++;//увеличиваем счетчик количества пропусков
            std::cout << std::endl << "Количество пропусков между
шаблонами равно " << c[num[i].index - patterns_pos[num[i].pattern_num]
+ 1] << std::endl;
            std::cout << "Количество джокеров между шаблонами равно "
<< patterns.size() << std::endl << std::endl;
            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
```

```
                std::cout << "Количество джокеров между шаблонами
равно количеству пропусков. Найден исходный шаблон в тексте" <<
std::endl;
                std::cout << "На позиции " << num[i].index -
patterns_pos[num[i].pattern_num] + 2 << std::endl;
                arr[n] = num[i].index -
patterns_pos[num[i].pattern_num] + 2;
                n++;
            }

        }

        std::cout << "-----------------------------------------------
------------------------------" << std::endl;
        std::cout << std::endl << "Ответ:" << std::endl;
        for (int i = 0; i < n; i++) std::cout << arr[i] << std::endl;
        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());
        }
        std::cout << std::endl << "Максимальное количество дуг,
исходящих из одной вершины " << max_count_of_arch << std::endl;
        std::cout << "Строка после удаления найденных шаблонов: " <<
text << std::endl;
    }
private:
    int max_count_of_arch;
    std::vector<numbers> num;
    std::vector <bohr_vertex> bohr;//бор
    std::vector <std::string> pattern;//шаблоны

    bohr_vertex make_vertex(int par, char symbol) {//создание вершины
бора
        bohr_vertex vertex;
        memset(vertex.next_vertex, 255, sizeof(vertex.next_vertex));
        vertex.flag = false;
        vertex.link = -1;
        memset(vertex.move, 255, sizeof(vertex.move));
        vertex.par = par;
        vertex.symbol = symbol;
        vertex.flink = -1;
        memset(vertex.pattern_num, 255, sizeof(vertex.pattern_num));
        return vertex;
    }

    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;
}
```

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

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

```
            }
        }

        return bohr[v].move[ch];
    }

    int get_suff_flink(int v) {//функция вычисления сжатых
суффиксальных ссылок
        if (bohr[v].flink == -1) {
            int u = get_suff_link(v);
            if (u == 0) {//если корень или начало шаблона
                bohr[v].flink = 0;
            }
            else {//если по суффиксальной ссылке конечная вершина-
равен суффиксальной ссылке, если нет-рекурсия.
                bohr[v].flink = (bohr[u].flag) ? u :
get_suff_flink(u);
            }
        }
        return bohr[v].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];
                        std::cout << std::endl << "Вершина " << u << "
конечная для шаблона " << s.pattern_num+1 << std::endl;
                        std::cout << "Найден подшаблон с номером " <<
s.pattern_num+1 << ", позиция в тексте " << s.index << std::endl;
                        num.push_back(s);
                    }
                    else
                        break;
                }
            }
            else std::cout << std::endl << "Вершина " << u << " не
конечная" << std::endl;
        }
    }




};
```

```
int main() {
    setlocale(LC_ALL, "Russian");
    std::vector<std::string> patterns;  //подстроки при делении по
джокеру
    std::vector<int> patterns_pos; //позиции подстрок
    std::string text;//текст
    std::string temp;//шаблон
    char joker;//джокер
    std::string pat;
    std::cout << "Введите текст:" << std::endl;
    std::cin >> text;
    std::cout << "Введите шаблон:" << std::endl;
    std::cin >> temp;
    std::cout << "Введите джокер:" << std::endl;
    std::cin >> joker;
    Borh borh;

    std::cout << std::endl << "Разделим шаблон на подстроки без
джокера" << std::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++;
            }
            borh.add_string_to_bohr(pat);
            patterns.push_back(pat);
            pat.clear();
        }
    }


    std::cout << "Вычисление количество дуг из вершин." << std::endl;
    borh.find_count_of_arch();


    std::cout << std::endl << "Нахождение шаблонов в тексте." <<
std::endl;
    borh.find_all_pos(text);
    borh.answer(patterns, patterns_pos, text, temp);

    return 0;
}
```