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. 8383 _____ Шишкин И.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить работу и реализовать алгоритм Ахо-Корасик для нахождения набора образцов в тексте.

Постановка задачи.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T, 1≤|T|≤100000).

Вторая - число n (1≤n≤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.

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, $1 \leq |T| \leq 100000$)

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.

Базовая часть лаб. работы № 5 состоит в выполнении обоих заданий на Stepik из раздела 6. Для обоих заданий на программирование должны быть версии кода с выводом промежуточных данных. В них, в частности, должны выводиться построение бора и автомата, построенный автомат (в виде, например, описания каждой вершины автомата), процесс его использования.

Вариант 1. На месте джокера может быть любой символ, за исключением заданного.

Основные теоретические положения.

Бор — это дерево, в котором каждая вершина обозначает какую-то строку (корень обозначает нулевую строку).

Суффиксная ссылка для каждой вершины p – это вершина, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине p.

Описание алгоритма Ахо-Корасик.

Алгоритм реализует поиск множества подстрок из словаря в данной строке. Его можно разбить на 2 этапа: построение бора, автомата и поиск шаблонов в тексте.

1. Создание бора и автомата.

На ребрах между вершинами написана 1 буква, таким образом, добираясь по ребрам из корня в какую-нибудь вершину и контангенируя буквы из ребер в порядке обхода, мы получим строку, соответствующую этой вершине.

Строить бор будем последовательным добавление исходных строк. Изначально у нас есть 1 вершина, корень (root) – пустая строка. Добавление строки происходит так: начиная в корне, двигаемся по дереву, выбирая каждый раз ребро, соответствующее очередной букве строки. Если такого ребра нет, то мы создаем его вместе с вершиной. Для каждой строки необходимо дополнительно хранить признак того, является она строкой из условия или нет. На рис. 1 пример построенного бора для строк: 1) acab, 2) accc, 3) acac, 4) baca, 5) abb, 6) z, 7) ac.

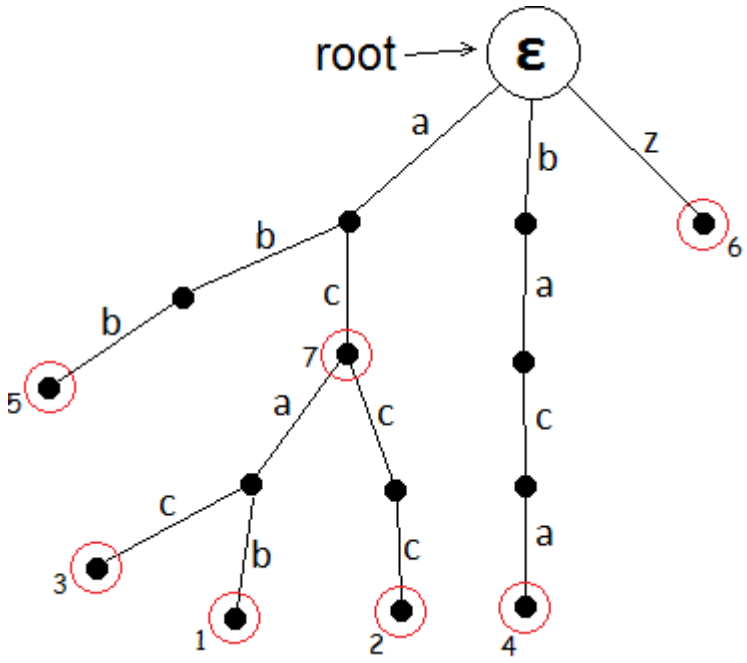

Рисунок 1 – Пример построенного бора

После построения бора по нему нужно построить конечный детерминированный автомат, состояния которого можно понимать, как вершины бора. Переход из состояний осуществляется по 2 параметрам – текущей вершине v и символу ch, по которому нам надо сдвинуться из этой вершины. Поконкретнее, необходимо найти вершину u, которая обозначает наидлиннейшую строку, состоящую из суффикса строки v (возможно нулевого) + символа ch. Если такого в боре нет, то идем в корень. Например, пусть бор построен по строкам “ab” и “bc”, и мы под воздействием строки “ab” перешли в некоторое состояние, являющееся листом. Тогда под воздействием буквы “c” мы вынуждены перейти в состояние, соответствующее строке “b”, и только оттуда выполнить переход по букве “c”.

Для удобства суффиксную ссылку из него проведём в себя же. Теперь мы можем переформулировать утверждение по поводу переходов в автомате так: пока из текущей вершины бора нет перехода по соответствующей букве (или пока мы не придём в корень бора), мы должны переходить по суффиксной ссылке. Таким образом, задача нахождения перехода свелась к задаче нахождения суффиксной ссылки, а задача нахождения суффиксной ссылки – к задаче нахождения суффиксной ссылки и перехода, но уже для более близких к корню вершин. Мы получили рекурсивную зависимость, но не бесконечную, и, более того, разрешить которую можно за линейное время.

2. Поиск шаблонов в тексте.
    - Текущая вершина – корень бора
    - Пока есть символы в строке:
        - Совершается переход по следующей букве строки (по правилам, указанным выше)

- Из текущей вершины по суффиксным ссылкам проходим до корня бора, проверяя, найдено ли вхождение (если встречается лист, то вхождение найдено)

Описание структур.

```
struct Vertex {
	std::vector <int> nextVrtx;
	bool flag;
	int suffLink;
	std::vector <int> autoMove;
	int par;
	char symb;
	int patternNumber;
	int deep;
};
```

Структура для хранения бора/автомата. Вектор next_vrtx[i] – номер вершины, в которую мы придем по символу с номером i в алфавите; flag – бит, указывающий на то, является ли наша вершина исходной строкой; suffLink – суффиксная ссылка, par – вершина-отец в дереве; symb – символ на ребре от par к этой вершине; autoMove – запоминание перехода автомата; patternNumber – номер шаблона; deep – глубина вершины в боре.

```
struct ResForStepik {
	int position;
	int number;
};
```

Структура для хранения ответа, используется только в 1-й программе. position – позиция вхождения шаблона, number – номер шаблона.

Описание функции makeBohr.

Функция Vertex makeBohr(int par, char c) создает новую вершину в боре. На вход принимает par, который записывается в поле структуры Vertex par, и

символ c, который записывается в поле symb. Значения suffLink и patternNumber становятся равны 1, flag становится false.

Описание функции addStringsToBohr.

Функция void addStringsToBohr(std::vector <Vertex>& bohr, int n) производит добавление строк в бор. На вход принимает вектор bohr – сам бор, и переменную n – количество шаблонов. С помощью цикла for сначала считывается очередной шаблон, затем по длине этого шаблона производится проход, высчитывается значение ch = s[j] - 'A' (где s – шаблон. Так как алфавит состоит только из символов A, C, G, T, N, то значение ch может изменяться от 0 до 25 – количество всех заглавных латинских букв). Затем производится проверка на то, существует ли ребро по этому символу, и если оно существует, то в бор добавляется новая вершина с помощью функции makeBohr. В конце прохода по строке значение flag становится true, глубина deep становится равным длине шаблона, patternNumber – номеру шаблона, но т.к. нумерация производится с 1, то оно принмает значение i+1.

Описание функции cmpForRes.

Функция bool cmpForRes(ResForStepik res1, ResForStepik res2) – компаратор для сортировки ответа по возрастанию (сначала номер позиции, затем номер шаблона).

Описание функции getSuffLink.

int getSuffLink(int v, std::vector <Vertex>& bohr) – функция получения вершины, доступной по суффиксной ссылке. На вход принимает v – номер вершины, для которой нужно найти суффиксную ссылку; bohr – бор.

Описание функции getAutoMove.

int getAutoMove(int v, int ch, std::vector <Vertex>& bohr) – функция получения вершины для перехода. На вход принимает v – номер вершины, из которой совершается переход; ch – номер символа для перехода; bohr – бор.

Описание функции findAllPos.

void findAllPos(const std::string& text, std::vector <Vertex>& bohr, std::vector <ResForStepik>& result) – функция поиска шаблонов в строке. На вход принимает text – текст, в котором производится поиск; bohr – бор; result – вектор для хранения результата.

Описание функции printAuto.

Функция void printAuto(std::vector <Vertex>& bohr) печатает состояния автомата/бора. На вход принимает только бор bohr.

Отличие 2-й программы от 1-й.

Во 2-й программе появляется специальный символ «джокер». Во-первых, отсутствует структура для печати результата на степике, так как теперь достаточно вывести лишь строки с номерами позиций вхождений шаблона. Во-вторых, в структуру Vertex была добавлена переменная bool isNextWC. Так же, теперь вводится лишь один шаблон, поэтому он считывается в функции main и передается в функцию void addStringToBohr(std::vector <Vertex>& bohr, const std::string& pat, char wildCard), где wildCard – символ джокера. В этой функции, при встрече в шаблоне pat символа wildCard, переменная ch принимается за букву ‘J’ (т.к. в алфавит данная буква не входит, значит ее можно использовать, потому что пользователь может выбрать джокером любой символ, а функция обрабатывает лишь заглавные латинские буквы). Помимо этого, при встрече джокера переменная isNextWC принимает значение true. Последнее отличие располагается в функции getAutoMove: если в словаре переходов нет проверяемого символа, то далее проверяется переменная isNextWC, и если она true, то возвращается значение bohr[v].nextVrtx['J' - 'A'].

Отличие 3-й программы от 2-й.

В этой программе появляется символ, который считывается после символа «джокер». Данный символ не может быть джокером, в программе он обозначается как char bannedWC – передается в функцию поиска вхождений, получения суффиксной ссылки и получения перехода автомата. Например, если был введен текст “AAA”, шаблон “A$A”, джокером выступает “$”, и если следующим символом будет введен “A”, то вхождений не будет найдено, т.к. этот символ не может быть джокером.

Отличие программ заключается в том, что при получении перехода автомата, когда в словаре переходов еще нет перехода по обрабатываемому символу, после проверки, что isNextWc == true, производится еще одна проверка: обрабатываемый символ не должен совпадать с «заблокированным» джокером. Если он не совпадает, то возвращается значение bohr[v].nextVrtx['J' - 'A']. Если совпадает, то функция продолжает свою обычную проверку.

Сложность алгоритма по времени.

Сложность алгоритма Ахо-Корасик по времени можно оценить, как

$$O((N + M) * A + k).$$

Так как алгоритм проходится по всему тексту длиной N, добавляет в бор шаблоны общей длиной M (при этом символов в строке определенное количество, поэтому A – размер алфавита), k – общая длина всех совпадений

Сложность алгоритма по памяти.

Сложность алгоритма Ахо-Корасик по памяти можно оценить, как

$$O(M * A).$$

Так как на каждую вершину приходится $O(A)$ памяти.

Спецификация программы.

Программа написана на языке C++. В 1-й программе на вход подается сначала текст, затем количество шаблонов, и шаблоны через ‘\n’. Во 2-й программе на вход подается текст, шаблон и символ джокера. В 3-й программе на вход подается текст, шаблон, символ джокера, и символ, который не может быть джокером. Важно помнить, что во всех программах алфавит состоит лишь из символов A, C, G, T, N, при этом джокер и «заблокированный» джокер могут быть любыми символами.

Тестирование 1-й программы.

Пример вывода результата для 1-го теста.

```
CCCA
1
CC
Добавление в бор шаблона "CC"...
Добавление символа 'C'...
Ребра нет. Создается новая вершина
Переход по символу 'C'
Добавление символа 'C'...
Ребра нет. Создается новая вершина
Переход по символу 'C'
Текущая вершина - лист

----------------------------------------------------------------
Производится поиск шаблона в тексте
Переход по символу 'C':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'C'
Существует ребро бора с этим символом. Этот переход добавляется в словарь переходов
Завершается функция getAutoMove

Вызывается функция getSuffLink...
Суффикс ссылка еще не была посчитана
Суффиксная ссылка из вершины = 0
Завершается функция getSuffLink

Переход по символу 'C':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'C'
Существует ребро бора с этим символом. Этот переход добавляется в словарь переходов
Завершается функция getAutoMove

Проверяемая вершина - лист. Найдено вхождение шаблона в текст
Вызывается функция getSuffLink...
Суффикс ссылка еще не была посчитана
Поиск суффиксной ссылки путем попытки перехода по символу из вершины предка
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getAutoMove...
Завершается функция getAutoMove

Завершается функция getSuffLink

Вызывается функция getSuffLink...
Завершается функция getSuffLink

Переход по символу 'C':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'C'
Не существует ребра бора с этим символом. Попытка совершения перехода из вершины, доступной по суффиксной ссылке
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getAutoMove...
Завершается функция getAutoMove
```

```
Завершается функция getAutoMove

Проверяемая вершина - лист. Найдено вхождение шаблона в текст
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getSuffLink...
Завершается функция getSuffLink

Переход по символу 'A':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'A'
Не существует ребра бора с этим символом. Попытка совершения перехода из вершины, доступной по суффиксной ссылке
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'A'
Не существует ребра бора с этим символом. Попытка совершения перехода из вершины, доступной по суффиксной ссылке
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'A'
Не существует ребра бора с этим символом. Попытка совершения перехода из вершины, доступной по суффиксной ссылке
Завершается функция getAutoMove

Завершается функция getAutoMove

Завершается функция getAutoMove

----------------------------------------------------------------
Построенный автомат:
Номер вершины 0:
Номер вершины предка 0, символ предка A
Ребра бора, доступные из вершины: 1
Переходы, доступные из вершины: 0 1
Номер вершины 1:
Номер вершины предка 0, символ предка C
Суффиксная ссылка 0
Ребра бора, доступные из вершины: 2
Переходы, доступные из вершины: 0 2
Номер вершины 2:
Вершина - лист по шаблону номер 1
Номер вершины предка 1, символ предка C
Суффиксная ссылка 1
Переходы, доступные из вершины: 0 2
----------------------------------------------------------------
Ответ:
1 1
2 1
Для продолжения нажмите любую клавишу . . .
```

Таблица 1 – тестирование программы

№	Input	Output
1	CCCA 1 CC	1 1 2 1
2	NANNNANNANAN 2 NAN A	1 1 2 2 5 1 6 2 8 1 9 2 10 1

		11 2
3	ACACGTNNNNAGT 3 G AG NNN	5 1 7 3 8 3 11 2 12 1

Тестирование 2-й программы.

Пример вывода результата для 1-го теста.

```
ACTANCA
A$$A$
$
Добавление в бор шаблона "A$$A$"...
Добавление символа 'A'...
Ребра нет. Создается новая вершина
Переход по символу 'A'
Добавление символа '$'...
Ребра нет. Создается новая вершина
Текущий символ - джокер, поэтому переменная isNextWC = true
Переход по символу '$'
Добавление символа '$'...
Ребра нет. Создается новая вершина
Текущий символ - джокер, поэтому переменная isNextWC = true
Переход по символу '$'
Добавление символа 'A'...
Ребра нет. Создается новая вершина
Переход по символу 'A'
Добавление символа '$'...
Ребра нет. Создается новая вершина
Текущий символ - джокер, поэтому переменная isNextWC = true
Переход по символу '$'
Конец добавления шаблона в бор

Производится поиск шаблона в тексте
Переход по символу 'A':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'A'
Существует ребро бора с этим символом. Этот переход добавляется в словарь переходов
Завершается функция getAutoMove

Вызывается функция getSuffLink...
Суффикс ссылка еще не была посчитана
Суффиксная ссылка из вершины = 0
Завершается функция getSuffLink

Переход по символу 'C':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'C'
Из вершины существует переход по символу джокеру
Завершается функция getAutoMove

Вызывается функция getSuffLink...
Суффикс ссылка еще не была посчитана
Поиск суффиксной ссылки путем попытки перехода по символу из вершины предка
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'J'
Не существует ребра бора с этим символом. Попытка совершения перехода из вершины, доступной по суффиксной ссылке
Завершается функция getAutoMove

Завершается функция getSuffLink

Переход по символу 'T':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'T'
Из вершины существует переход по символу джокеру
Завершается функция getAutoMove
```

```
Вызывается функция getSuffLink...
Суффикс ссылка еще не была посчитана
Поиск суффиксной ссылки путем попытки перехода по символу из вершины предка
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getAutoMove...
Завершается функция getAutoMove

Завершается функция getSuffLink

Переход по символу 'A':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'A'
Существует ребро бора с этим символом. Этот переход добавляется в словарь переходов
Завершается функция getAutoMove

Вызывается функция getSuffLink...
Суффикс ссылка еще не была посчитана
Поиск суффиксной ссылки путем попытки перехода по символу из вершины предка
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getAutoMove...
Завершается функция getAutoMove

Завершается функция getSuffLink

Вызывается функция getSuffLink...
Завершается функция getSuffLink

Переход по символу 'N':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'N'
Из вершины существует переход по символу джокеру
Завершается функция getAutoMove


!!!Проверяемая вершина - лист. Найдено вхождение шаблона в текст: 1

Вызывается функция getSuffLink...
Суффикс ссылка еще не была посчитана
Поиск суффиксной ссылки путем попытки перехода по символу из вершины предка
Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'J'
Из вершины существует переход по символу джокеру
Завершается функция getAutoMove

Завершается функция getSuffLink

Вызывается функция getSuffLink...
Завершается функция getSuffLink

Переход по символу 'C':
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'C'
Не существует ребра бора с этим символом. Попытка совершения перехода из вершины, доступной по суффиксной ссылке
Вызывается функция getSuffLink...
Завершается функция getSuffLink
```

```
Вызывается функция getAutoMove...
В словаре переходов еще нет перехода по символу 'C'
Из вершины существует переход по символу джокеру
Завершается функция getAutoMove

Завершается функция getAutoMove

Вызывается функция getSuffLink...
Завершается функция getSuffLink

Переход по символу 'A':
Вызывается функция getAutoMove...
Завершается функция getAutoMove

Вызывается функция getSuffLink...
Завершается функция getSuffLink

Вызывается функция getSuffLink...
Завершается функция getSuffLink


Ответ:
1
```

Таблица 2 – тестирование программы

№	Input	Output
1	ACTANCA A$$A$ $	1
2	ACACAGGAGA A%A %	1, 3, 8
3	ACG A&& &	1

Тестирование 3-й программы.

Пример вывода результата для 1-го теста.

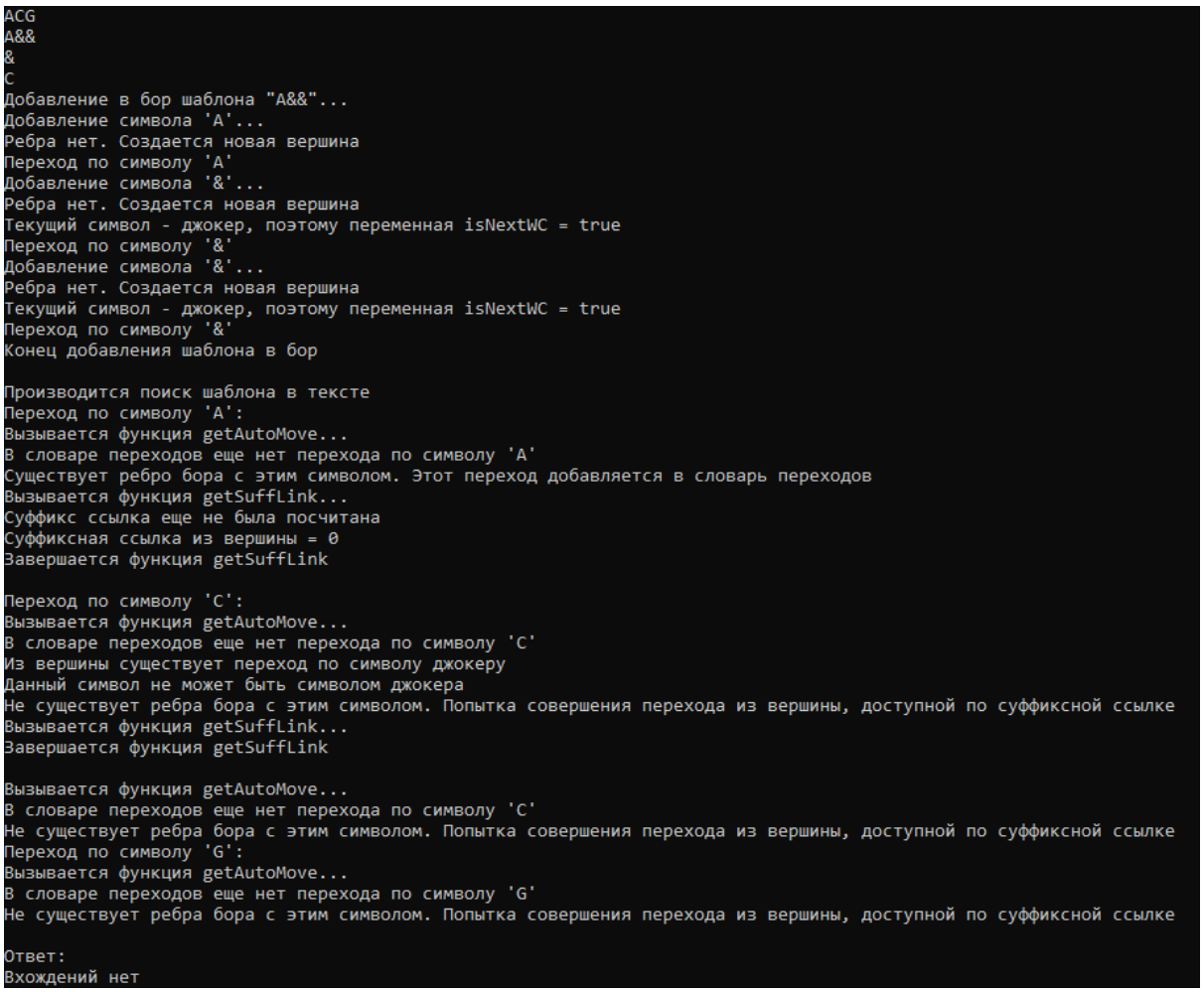

Таблица 3 – тестирование программы

№	Input	Output
1	ACG A&& & C	Вхождений нет
2	NANNNANNANAN NXN X N	1, 5, 8, 10
3	ACACAGGAGA A%A % G	1, 3

Выводы.

В ходе выполнения лабораторной работы был реализован на языке C++ алгоритм Ахо-Корасик, на его основе построены алгоритмы для поиска вхождений шаблона с «джокером» в строку.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ 1

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

struct Vertex {
	std::vector <int> nextVrtx;  //nextVrtx[i] — номер вершины, в которую мы придем по символу с номером i в алфавите
	bool flag;	//бит, указывающий на то, является ли наша вершина исходной строкой
	int suffLink; //суффиксная ссылка
	std::vector <int> autoMove; //запоминание перехода автомата
	int par;	//вершина-отец в дереве
	char symb; //символ на ребре от par к этой вершине
	int patternNumber; //номер шаблона
	int deep;	//глубина
};

struct ResForStepik {
	int position;
	int number;
};

Vertex makeBohr(int par, char c);	//функция инициализации бора
void addStringsToBohr(std::vector <Vertex>& bohr, int n);	//добавление строк к бору
int getSuffLink(int v, std::vector <Vertex>& bohr);		//получение суффиксной ссылки
int getAutoMove(int v, int ch, std::vector <Vertex>& bohr);		//получение перехода автомата
void findAllPos(const std::string& text, std::vector <Vertex>& bohr, std::vector <ResForStepik>& result);	//поиск вхождений
bool cmpForRes(ResForStepik res1, ResForStepik res2);	//компаратор для результата
void printAuto(std::vector <Vertex>& bohr);	//печать состояний автомата

int main() {
	setlocale(LC_ALL, "RUS");
	std::string text;
	int n;
	std::vector <Vertex> bohr;	//бор
	std::vector <ResForStepik> result;
```

```
	std::cin >> text;
	std::cin >> n;

	if (n <= 0) {
		std::cout << "Вхождения не найдены\n";
		system("pause");
		return 0;
	}

	bohr.push_back(makeBohr(0, NULL));

	addStringsToBohr(bohr, n);
	std::cout << "----------------------------------------------------------------" <<
std::endl;

	findAllPos(text, bohr, result);

	std::sort(result.begin(), result.end(), cmpForRes);

	printAuto(bohr);
	std::cout << "----------------------------------------------------------------" <<
std::endl;
	std::cout << "Ответ:" << std::endl;

	if (result.size() == 0) {
		std::cout << "Вхождения не найдены\n";
	}

	for (int i = 0; i < result.size(); i++) {
		std::cout << result[i].position << " " << result[i].number << std::endl;
	}

	system("pause");
	return 0;
}

Vertex makeBohr(int par, char c) {
	Vertex b;
	for (int i = 0; i < 26; i++) {   //инициализация бора (до 26, т.к. в ASCII table
заглавные латинские буквы идут от 65 до 90)
		b.nextVrtx.push_back(-1);
		b.autoMove.push_back(-1);
	}
```

```
	b.suffLink = -1;
	b.flag = false;
	b.par = par;
	b.symb = c;
	b.patternNumber = -1;
	return b;
}

void addStringsToBohr(std::vector <Vertex>& bohr, int n) {
	for (int i = 0; i < n; i++) {
		std::string s;
		std::cin >> s;

		std::cout << "Добавление в бор шаблона \"" << s << "\"..." <<
std::endl;

		int num = 0; //начинаем с корня
		for (size_t j = 0; j < s.length(); j++) {
			std::cout << "Добавление символа '" << s[i] << "'..." <<
std::endl;
			int ch = s[j] - 'A';

			if (bohr[num].nextVrtx[ch] == -1) { //-1 - признак отсутствия
ребра
				std::cout << "Ребра нет. Создается новая вершина" <<
std::endl;
				bohr.push_back(makeBohr(num, ch));
				bohr[num].nextVrtx[ch] = bohr.size() - 1;
			}

			num = bohr[num].nextVrtx[ch];
			std::cout << "Переход по символу '" << s[i] << "'" << std::endl;
		}

		std::cout << "Текущая вершина - лист" << std::endl << std::endl;
		bohr[num].flag = true;
		bohr[num].deep = s.length();
		bohr[num].patternNumber = i + 1;
	}
}

int getSuffLink(int v, std::vector <Vertex>& bohr) {
	std::cout << "Вызывается функция getSuffLink..." << std::endl;
```

```
	if (bohr[v].suffLink == -1) {    //если еще не считали
		std::cout << "Суффикс ссылка еще не была посчитана" << std::endl;

		if (v == 0 || bohr[v].par == 0) {    //если v - корень или предок v - 
корень
			std::cout << "Суффиксная ссылка из вершины = 0" << std::endl;
			bohr[v].suffLink = 0;
		}

		else {
			std::cout << "Поиск суффиксной ссылки путем попытки 
перехода по символу из вершины предка\n";
			bohr[v].suffLink = getAutoMove(getSuffLink(bohr[v].par, bohr), 
bohr[v].symb, bohr);
		}
	}

	std::cout << "Завершается функция getSuffLink" << std::endl << std::endl;
	return bohr[v].suffLink;
}

int getAutoMove(int v, int ch, std::vector <Vertex>& bohr) {
	std::cout << "Вызывается функция getAutoMove..." << std::endl;

	if (bohr[v].autoMove[ch] == -1) {   // если в словаре переход нет текущего 
символа
		std::cout << "В словаре переходов еще нет перехода по символу '" 
<< (char)(ch + 'A') << "'" << std::endl;

		if (bohr[v].nextVrtx[ch] != -1) {
			std::cout << "Существует ребро бора с этим символом. Этот 
переход добавляется в словарь переходов" << std::endl;
			bohr[v].autoMove[ch] = bohr[v].nextVrtx[ch];
		}

		else {
			std::cout << "Не существует ребра бора с этим символом. 
Попытка совершения перехода из вершины, доступной по суффиксной ссылке" 
<< std::endl;

			if (v == 0)
				bohr[v].autoMove[ch] = 0;
			else
```

```
                        bohr[v].autoMove[ch] = getAutoMove(getSuffLink(v,
bohr), ch, bohr);
                }
        }

        std::cout << "Завершается функция getAutoMove" << std::endl << std::endl;
        return bohr[v].autoMove[ch];
}

void findAllPos(const std::string& text, std::vector <Vertex>& bohr, std::vector
<ResForStepik>& result) {
        std::cout << "Производится поиск шаблона в тексте" << std::endl;
        int curr = 0;

        for (int i = 0; i < text.length(); i++) {
                std::cout << "Переход по символу '" << text[i] << "':" << std::endl;

                curr = getAutoMove(curr, text[i] - 'A', bohr);     //Переход из текущей
вершины по текущему символу
                for (int tmp = curr; tmp != 0; tmp = getSuffLink(tmp, bohr)) {          //
Обход автомата по суффикс ссылкам
                        if (bohr[tmp].flag) {               //Если при обходе встретился
flag = 1
                                std::cout << "Проверяемая вершина - лист. Найдено
вхождение шаблона в текст" << std::endl;
                                ResForStepik res;
                                res.number = bohr[tmp].patternNumber;
                                res.position = i + 2 - bohr[tmp].deep;
                                result.push_back(res);
                        }
                }
        }
}

bool cmpForRes(ResForStepik res1, ResForStepik res2) {
        if (res1.position != res2.position) {
                return res1.position < res2.position;
        }
        return res1.number < res2.number;
}

void printAuto(std::vector <Vertex>& bohr) {
        std::cout << "------------------------------------------------------------" <<
std::endl;
```

```
	std::cout << "Построенный автомат:" << std::endl;

	for (int i = 0; i < bohr.size(); i++) {
		Vertex curr = bohr[i];
		std::cout << "Номер вершины " << i << ":\n";

		if (curr.flag) {
			std::cout << "Вершина - лист по шаблону номер " <<
curr.patternNumber << std::endl;
		}

		if (curr.par != -1) {
			std::cout << "Номер вершины предка " << curr.par << ", символ
предка " << (char)(curr.symb + 'A') << std::endl;
		}

		if (curr.suffLink != -1) {
			std::cout << "Суффиксная ссылка " << curr.suffLink <<
std::endl;
		}

		if (!curr.flag) {
			std::cout << "Ребра бора, доступные из вершины: ";
			for (int j = 0; j < curr.nextVrtx.size(); j++) {
				if (curr.nextVrtx[j] != -1) std::cout << curr.nextVrtx[j] << "
";
			}
			std::cout << std::endl;
		}

		std::cout << "Переходы, доступные из вершины: ";
		for (int j = 0; j < curr.autoMove.size(); j++) {
			if (curr.autoMove[j] != -1) std::cout << curr.autoMove[j] << " ";
		}
		std::cout << std::endl;

	}
}
```

ПРИЛОЖЕНИЕ Б

КОД ПРОГРАММЫ 2

```
#include <iostream>
#include <vector>

struct Vertex {
	std::vector <int> nextVrtx;  //nextVrtx[i] – номер вершины, в
которую мы придем по символу с номером i в алфавите
	bool flag;	//бит, указывающий на то, является ли наша вершина
исходной строкой
	int suffLink; //суффиксная ссылка
	std::vector <int> autoMove; //запоминание перехода автомата
	int par;	//вершина-отец в дереве
	char symb; //символ на ребре от par к этой вершине
	int deep;	//глубина
	bool isNextWC;	//является ли следующий элемент джокером
};

Vertex makeBohr(int par, char c);	//функция инициализации бора
void addStringToBohr(std::vector <Vertex>& bohr, const std::string& pat,
char wildCard);	//добавление паттерна к бору
int getSuffLink(int v, std::vector <Vertex>& bohr);	//получение
суффиксной ссылки
int getAutoMove(int v, int ch, std::vector <Vertex>& bohr);
	//получение перехода автомата
void findAllPos(const std::string& text, std::vector <Vertex>& bohr,
std::vector <int>& answer);	//поиск вхождений
void printAuto(std::vector <Vertex>& bohr);	//печать состояний
автомата

int main() {
	setlocale(LC_ALL, "RUS");
	std::string text;
	std::string pattern;
	std::vector <Vertex> bohr;	//бор
	char wildCard;
	std::vector <int> answer;	//индексы вхождения

	std::cin >> text;
	std::cin >> pattern;
	std::cin >> wildCard;

	addStringToBohr(bohr, pattern, wildCard);

	findAllPos(text, bohr, answer);

	std::cout << std::endl << "Ответ:" << std::endl;
	for (int i = 0; i < answer.size(); i++) {
		if (i == 0) std::cout << answer[i];
```

```
            else std::cout << ", " << answer[i];
      }
      std::cout << std::endl;

      system("pause");
      return 0;
}

Vertex makeBohr(int par, char c) {
      Vertex b;
      for (int i = 0; i < 26; i++) {   //инициализация бора (до 26, т.к.
в ASCII table заглавные латинские буквы идут от 65 до 90)
            b.nextVrtx.push_back(-1);
            b.autoMove.push_back(-1);
      }
      b.suffLink = -1;
      b.flag = false;
      b.par = par;
      b.symb = c;
      b.isNextWC = false;
      return b;
}

void addStringToBohr(std::vector <Vertex>& bohr, const std::string& pat,
char wildCard) {
      std::cout << "Добавление в бор шаблона \"" << pat << "\"..." <<
std::endl;
      bohr.push_back(makeBohr(0, NULL));
      int num = 0; //начинаем с корня

      for (size_t j = 0; j < pat.length(); j++) {
            std::cout << "Добавление символа '" << pat[j] << "'..." <<
std::endl;
            int ch = 0;
            if (pat[j] == wildCard) ch = 'J' - 'A';
            else ch = pat[j] - 'A';

            if (bohr[num].nextVrtx[ch] == -1) { //-1 - признак отсутствия
ребра
                  std::cout << "Ребра нет. Создается новая вершина" <<
std::endl;
                  bohr.push_back(makeBohr(num, ch));
                  bohr[num].nextVrtx[ch] = bohr.size() - 1;
                  if (pat[j] == wildCard) {
                        std::cout << "Текущий символ - джокер, поэтому
переменная isNextWC = true" << std::endl;
                        bohr[num].isNextWC = true;
                  }
            }

            num = bohr[num].nextVrtx[ch];
```

```
            std::cout << "Переход по символу '" << pat[j] << "'" <<
std::endl;
      }

      bohr[num].flag = true;
      bohr[num].deep = pat.length();
      std::cout << "Конец добавления шаблона в бор" << std::endl <<
std::endl;
}

int getSuffLink(int v, std::vector <Vertex>& bohr) {
      std::cout << "Вызывается функция getSuffLink..." << std::endl;

      if (bohr[v].suffLink == -1) {    //если еще не считали
            std::cout << "Суффикс ссылка еще не была посчитана" <<
std::endl;

            if (v == 0 || bohr[v].par == 0) {    //если v - корень или
предок v - корень
                  std::cout << "Суффиксная ссылка из вершины = 0" <<
std::endl;
                  bohr[v].suffLink = 0;
            }

            else {
                  std::cout << "Поиск суффиксной ссылки путем попытки
перехода по символу из вершины предка\n";
                  bohr[v].suffLink = getAutoMove(getSuffLink(bohr[v].par,
bohr), bohr[v].symb, bohr);
            }
      }

      std::cout << "Завершается функция getSuffLink" << std::endl <<
std::endl;
      return bohr[v].suffLink;
}

int getAutoMove(int v, int ch, std::vector <Vertex>& bohr) {
      std::cout << "Вызывается функция getAutoMove..." << std::endl;

      if (bohr[v].autoMove[ch] == -1) {   // если нет текущего символа
            std::cout << "В словаре переходов еще нет перехода по символу
'" << (char)(ch + 'A') << "'" << std::endl;

            if (bohr[v].isNextWC) {
                  std::cout << "Из вершины существует переход по символу
джокеру" << std::endl;
                  std::cout << "Завершается функция getAutoMove" <<
std::endl << std::endl;
                  return bohr[v].nextVrtx['J' - 'A'];
            }
```

```
          if (bohr[v].nextVrtx[ch] != -1) {
               std::cout << "Существует ребро бора с этим символом.
Этот переход добавляется в словарь переходов" << std::endl;
               bohr[v].autoMove[ch] = bohr[v].nextVrtx[ch];
          }

          else {
               std::cout << "Не существует ребра бора с этим символом.
Попытка совершения перехода из вершины, доступной по суффиксной ссылке"
<< std::endl;

               if (v == 0)
                    bohr[v].autoMove[ch] = 0;
               else
                    bohr[v].autoMove[ch] = getAutoMove(getSuffLink(v,
bohr), ch, bohr);
          }
     }

     std::cout << "Завершается функция getAutoMove" << std::endl <<
std::endl;
     return bohr[v].autoMove[ch];
}

void findAllPos(const std::string& text, std::vector <Vertex>& bohr,
std::vector <int>& answer) {
     std::cout << "Производится поиск шаблона в тексте" << std::endl;
     int curr = 0;

     for (int i = 0; i < text.length(); i++) {
          std::cout << "Переход по символу '" << text[i] << "':" <<
std::endl;

          curr = getAutoMove(curr, text[i] - 'A', bohr);     //Переход
из текущей вершины по текущему символу
          for (int tmp = curr; tmp != 0; tmp = getSuffLink(tmp, bohr)) {
          // Обход автомата по суффикс ссылкам
               if (bohr[tmp].flag) {      //Если при обходе встретился
flag = 1
                    std::cout << std::endl << "!!!Проверяемая вершина -
лист. Найдено вхождение шаблона в текст: ";
                    std::cout << i + 2 - bohr[tmp].deep << std::endl <<
std::endl;
                    answer.push_back(i + 2 - bohr[tmp].deep);
               }
          }
     }
}

void printAuto(std::vector <Vertex>& bohr) {
```

```
	std::cout << "-----------------------------------------------------
------------" << std::endl;
	std::cout << "Построенный автомат:" << std::endl;

	for (int i = 0; i < bohr.size(); i++) {
		Vertex curr = bohr[i];
		std::cout << "Номер вершины " << i << ":\n";

		if (curr.flag) {
			std::cout << "Вершина - лист" << std::endl;
		}

		if (curr.par != -1) {
			std::cout << "Номер вершины предка " << curr.par << ",
символ предка " << (char)(curr.symb + 'A') << std::endl;
		}

		if (curr.suffLink != -1) {
			std::cout << "Суффиксная ссылка " << curr.suffLink <<
std::endl;
		}

		if (!curr.flag) {
			std::cout << "Ребра бора, доступные из вершины: ";
			for (int j = 0; j < curr.nextVrtx.size(); j++) {
				if (curr.nextVrtx[j] != -1) std::cout <<
curr.nextVrtx[j] << " ";
			}
			std::cout << std::endl;
		}

		std::cout << "Переходы, доступные из вершины: ";
		for (int j = 0; j < curr.autoMove.size(); j++) {
			if (curr.autoMove[j] != -1) std::cout <<
curr.autoMove[j] << " ";
		}
		std::cout << std::endl;

	}
}
```

ПРИЛОЖЕНИЕ В

КОД ПРОГРАММЫ 3

```
#include <iostream>
#include <vector>

struct Vertex {
	std::vector <int> nextVrtx;  //nextVrtx[i] – номер вершины, в
которую мы придем по символу с номером i в алфавите
	bool flag;	//бит, указывающий на то, является ли наша вершина
исходной строкой
	int suffLink; //суффиксная ссылка
	std::vector <int> autoMove; //запоминание перехода автомата
	int par;	//вершина-отец в дереве
	char symb; //символ на ребре от par к этой вершине
	int deep;	//глубина
	bool isNextWC;	//является ли следующий элемент джокером
};

Vertex makeBohr(int par, char c);	//функция инициализации бора
void addStringToBohr(std::vector <Vertex>& bohr, const std::string& pat,
char wildCard);	//добавление паттерна к бору
int getSuffLink(int v, std::vector <Vertex>& bohr, char wildCard, char
bannedWC);	//получение суффиксной ссылки
int getAutoMove(int v, int ch, std::vector <Vertex>& bohr, char wildCard,
char bannedWC);	//получение перехода автомата
void findAllPos(const std::string& text, std::vector <Vertex>& bohr,
std::vector <int>& answer, char wildCard, char bannedWC);  //поиск
вхождений
void printAuto(std::vector <Vertex>& bohr);	//печать состояний
автомата

int main() {
	setlocale(LC_ALL, "RUS");
	std::string text;	//текст
	std::string pattern;  // шаблон
	std::vector <Vertex> bohr;	//бор
	char wildCard;	//джокер
	char bannedWC;	// запрещенный джокер
	std::vector <int> answer;	//ответ

	std::cin >> text;
	std::cin >> pattern;
	std::cin >> wildCard;
	std::cin >> bannedWC;

	addStringToBohr(bohr, pattern, wildCard);

	findAllPos(text, bohr, answer, wildCard, bannedWC);
```

```
	std::cout << std::endl << "Ответ:" << std::endl;
	if (answer.size() == 0) std::cout << "Вхождений нет" << std::endl;
	for (int i = 0; i < answer.size(); i++) {
		if (i == 0) std::cout << answer[i];
		else std::cout << ", " << answer[i];
	}
	std::cout << std::endl;

	system("pause");
	return 0;
}

Vertex makeBohr(int par, char c) {
	Vertex b;
	for (int i = 0; i < 26; i++) {   //инициализация бора (до 26, т.к. 
в ASCII table заглавные латинские буквы идут от 65 до 90)
		b.nextVrtx.push_back(-1);
		b.autoMove.push_back(-1);
	}
	b.suffLink = -1;
	b.flag = false;
	b.par = par;
	b.symb = c;
	b.isNextWC = false;
	return b;
}

void addStringToBohr(std::vector <Vertex>& bohr, const std::string& pat, 
char wildCard) {
	std::cout << "Добавление в бор шаблона \"" << pat << "\"..." << 
std::endl;
	bohr.push_back(makeBohr(0, NULL));
	int num = 0; //начинаем с корня

	for (size_t j = 0; j < pat.length(); j++) {
		std::cout << "Добавление символа '" << pat[j] << "'..." << 
std::endl;

		int ch = 0;
		if (pat[j] == wildCard) ch = 'J' - 'A';
		else ch = pat[j] - 'A';

		if (bohr[num].nextVrtx[ch] == -1) { //-1 - признак отсутствия 
ребра
			std::cout << "Ребра нет. Создается новая вершина" << 
std::endl;
			bohr.push_back(makeBohr(num, ch));
			bohr[num].nextVrtx[ch] = bohr.size() - 1;

			if (pat[j] == wildCard) {
```

```
                    std::cout << "Текущий символ - джокер, поэтому
переменная isNextWC = true" << std::endl;
                    bohr[num].isNextWC = true;
               }
          }

          num = bohr[num].nextVrtx[ch];
          std::cout << "Переход по символу '" << pat[j] << "'" <<
std::endl;
     }

     bohr[num].flag = true;
     bohr[num].deep = pat.length();
     std::cout << "Конец добавления шаблона в бор" << std::endl <<
std::endl;
}

int getSuffLink(int v, std::vector <Vertex>& bohr, char wildCard, char
bannedWC) {
     std::cout << "Вызывается функция getSuffLink..." << std::endl;

     if (bohr[v].suffLink == -1) {   //если еще не считали
          std::cout << "Суффикс ссылка еще не была посчитана" <<
std::endl;

          if (v == 0 || bohr[v].par == 0) {   //если v - корень или
предок v - корень
               std::cout << "Суффиксная ссылка из вершины = 0" <<
std::endl;
               bohr[v].suffLink = 0;
          }

          else {
               std::cout << "Поиск суффиксной ссылки путем попытки
перехода по символу из вершины предка" << std::endl;
               bohr[v].suffLink = getAutoMove(getSuffLink(bohr[v].par,
bohr, wildCard, bannedWC), bohr[v].symb, bohr, wildCard, bannedWC);
          }
     }

     std::cout << "Завершается функция getSuffLink" << std::endl <<
std::endl;
     return bohr[v].suffLink;
}

int getAutoMove(int v, int ch, std::vector <Vertex>& bohr, char wildCard,
char bannedWC) {
     std::cout << "Вызывается функция getAutoMove..." << std::endl;

     if (bohr[v].autoMove[ch] == -1) {
```

```
            std::cout << "В словаре переходов еще нет перехода по символу
'" << (char)(ch + 'A') << "'" << std::endl;
            if (bohr[v].isNextWC) {
                  std::cout << "Из вершины существует переход по символу
джокеру" << std::endl;
                  if (ch != bannedWC - 'A') {
                        std::cout << "Данный символ может быть джокером.
Завершается функция getAutoMove" << std::endl << std::endl;
                        return bohr[v].nextVrtx['J' - 'A'];
                  }

                  std::cout << "Данный символ не может быть символом
джокера" << std::endl;
            }
      }
      if (bohr[v].nextVrtx[ch] != -1) {
            std::cout << "Существует ребро бора с этим символом. Этот
переход добавляется в словарь переходов" << std::endl;
            bohr[v].autoMove[ch] = bohr[v].nextVrtx[ch];
      }
      else {
            std::cout << "Не существует ребра бора с этим символом.
Попытка совершения перехода из вершины, доступной по суффиксной ссылке"
<< std::endl;
            if (v == 0)
                  bohr[v].autoMove[ch] = 0;
            else
                  bohr[v].autoMove[ch] = getAutoMove(getSuffLink(v, bohr,
wildCard, bannedWC), ch, bohr, wildCard, bannedWC);
      }
      return bohr[v].autoMove[ch];
}

void findAllPos(const std::string& text, std::vector <Vertex>& bohr,
std::vector <int>& answer, char wildCard, char bannedWC) {
      std::cout << "Производится поиск шаблона в тексте" << std::endl;
      int curr = 0;

      for (int i = 0; i < text.length(); i++) {
            std::cout << "Переход по символу '" << text[i] << "':" <<
std::endl;
            int sym = text[i] - 'A';

            curr = getAutoMove(curr, sym, bohr, wildCard, bannedWC);
//Переход из текущей вершины по текущему символу
            for (int tmp = curr; tmp != 0; tmp = getSuffLink(tmp, bohr,
wildCard, bannedWC)) {     // Обход автомата по суффикс ссылкам
                  if (bohr[tmp].flag) {      //Если при обходе встретился
flag = 1
                        std::cout << std::endl << "!!!Проверяемая вершина -
лист. Найдено вхождение шаблона в текст: ";
```

```
                    std::cout << i + 2 - bohr[curr].deep << std::endl;
                    answer.push_back(i + 2 - bohr[tmp].deep);
                }
            }
        }
}

void printAuto(std::vector <Vertex>& bohr) {
    std::cout << "-------------------------------------------------------
------------" << std::endl;
    std::cout << "Построенный автомат:" << std::endl;

    for (int i = 0; i < bohr.size(); i++) {
        Vertex curr = bohr[i];
        std::cout << "Номер вершины " << i << ":\n";

        if (curr.flag) {
            std::cout << "Вершина - лист" << std::endl;
        }

        if (curr.par != -1) {
            std::cout << "Номер вершины предка " << curr.par << ",
символ предка " << (char)(curr.symb + 'A') << std::endl;
        }

        if (curr.suffLink != -1) {
            std::cout << "Суффиксная ссылка " << curr.suffLink <<
std::endl;
        }

        if (!curr.flag) {
            std::cout << "Ребра бора, доступные из вершины: ";
            for (int j = 0; j < curr.nextVrtx.size(); j++) {
                if (curr.nextVrtx[j] != -1) std::cout <<
curr.nextVrtx[j] << " ";
            }
            std::cout << std::endl;
        }

        std::cout << "Переходы, доступные из вершины: ";
        for (int j = 0; j < curr.autoMove.size(); j++) {
            if (curr.autoMove[j] != -1) std::cout <<
curr.autoMove[j] << " ";
        }
        std::cout << std::endl;

    }
}
```