

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. 9382

Русинов Д.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Разработать программу для поиска шаблонов в строке, используя алгоритм Ахо-Корасик. Также разработать программу, которая ищет шаблон, включающий в себя символы джокера, в строке поиска.

Задание 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 $P = \{p_1, \dots, p_n\}$ $1 \leq |p_i| \leq 75$

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.

Sample Input:

```
NTAG
3
TAGT
TAG
T
```

Sample Output:

```
2 2
2 3
```

Задание 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 PP необходимо найти все вхождения P в текст T .

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

Символ джокер не входит в алфавит, символы которого используются в T . Каждый джокер соответствует одному символу, а не подстроке 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.

Sample Input:

ACTANCA
A\$ \$A\$
\$

Sample Output:

1

Индивидуальное задание

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

Описание алгоритма

Первоначально необходимо построить бор из шаблонов, которые были даны на вход. Шаблон в боре представляется в виде узлов, которые имеют уникальный числовой идентификатор. Считываются поочередно символы шаблона, создается узел, до которого можно добраться по считанному символу, узел добавляется в бор. Добавление в бор узлов для шаблона начинается с корня, затем новые узлы могут прикрепляться к ранее добавленным узлам. Так, если по символу 'а' от корня в боре уже есть узел, тогда создавать еще один такой узел нет нужды. Узел, который пришелся на конечный символ шаблона, является терминальным.

После построения бора начинается работа алгоритма. Поочередно считываются символы данного на вход текста, по ним выполняется переход в боре. Если из текущего узла есть ребро с рассматриваемым символом до следующего узла, то выполняется переход по этому ребру. Если текущий узел корневой и нет ребра с рассматриваемым символом, то переход будет выполнен в корневой узел. В ином случае вычисляется суффиксальная ссылка текущего узла, а уже от нее выполняется переход по рассматриваемому символу. Суффиксальная ссылка – указатель на узел, что символы, начиная с корня, до этого узла образуют максимальный собственный суффикс слова, образованного от текущего узла.

Суффиксальная ссылка вычисляется переходом по суффиксальной ссылке узла родителя по символу, который ведет к текущему узлу. Если

текущий узел находится первым после корня или же узел является корнем, то суффиксальная ссылка указывается на корень. Суффиксальная ссылка вычисляется рекурсивно, основываясь на суффиксальных ссылках родителей, поэтому она будет вычисляться до тех пор, пока не дойдет до корня, либо же не будет выполнен успешный переход.

После осуществления перехода, проверяется узел, является ли он терминальным. Если он терминальный, значит было найдено вхождение образца в текст. После этого необходимо вычислить хорошие суффиксальные ссылки от этого узла. Хорошая суффиксальная ссылка – это суффиксальная ссылка, которая указывает на терминальный узел. Хорошая суффиксальная ссылка вычисляется до тех пор, пока узел не станет корневым. При нахождении образца в тексте, все его слова, построенные от корня до узла, на который указывает хорошая суффиксальная ссылка, также будут входить в текст. Хорошая суффиксальная ссылка необходима, чтобы в дальнейшем избежать лишних скачков при нахождении образца вновь.

Во второй программе используется шаблон, в котором могут присутствовать джокеры – символ, который является любым символом из алфавита. В этом случае строится бор с алфавитом, в котором присутствует символ джокера. При осуществлении перехода по рассматриваемому символу, если от текущего узла есть ребро с джокером, то переход выполняется по джокеру. При этом рассматриваемый символ запоминается, чтобы была возможность вычислить суффиксальные ссылки по символу, который находился “под джокером”. Для корректной работы алгоритма, суффиксальные ссылки вычисляются каждый раз заново, поскольку они могут зависеть от символа, который находился “под джокером”.

Оценка сложности по времени.

Алгоритм рассматривает каждый символ строки (N символов), вычисления суффиксальных ссылок (S – количество узлов бора, M – размер алфавита), проверяются хорошие суффиксальные ссылки (T – количество вхождений образцов в строке поиска), итоговая сложность $O(N + T + S * M)$.

Оценка сложности по памяти.

N – сумма длин всех шаблонов, K – количество символов в алфавите, итоговая сложность $O(N * K)$.

Тестирование.

Тестирование первой программы.

Нумерация	Входные данные	Выходные данные
1	NTAG 3 TAGT TAG T	----- Добавление шаблона TAGT в бор Рассматривается символ 'T' Для данного символа был добавлен узел в бор, его номер: 1 Рассматривается символ 'A' Для данного символа был добавлен узел в бор, его номер: 2 Рассматривается символ 'G' Для данного символа был добавлен узел в бор, его номер: 3

	<p>Рассматривается символ 'T'</p> <p>Для данного символа был добавлен узел в бор, его номер: 4</p> <p>-----</p> <p>-----</p> <p>Добавление шаблона TAG в бор</p> <p>Рассматривается символ 'T'</p> <p>Узел для данного символа уже есть в боре, его номер: 1</p> <p>Рассматривается символ 'A'</p> <p>Узел для данного символа уже есть в боре, его номер: 2</p> <p>Рассматривается символ 'G'</p> <p>Узел для данного символа уже есть в боре, его номер: 3</p> <p>-----</p> <p>-----</p> <p>Добавление шаблона T в бор</p> <p>Рассматривается символ 'T'</p> <p>Узел для данного символа уже есть в боре, его номер: 1</p> <p>-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 0</p> <p>Рассматривается символ текста 'N' по индексу 0</p> <p>Выполняется переход по символу 'N' из узла 0</p> <p>Для данного символа из данного узла не вычислен переход</p>
--	--

	<p>Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0</p> <p>Переход по символу 'N' из узла 0 будет выполнен в узел 0</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>-----ПОИСК ХОРОШИХ ССЫЛОК-----</p> <p>-----КОНЕЦ ПОИСКА ХОРОШИХ ССЫЛОК-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 0</p> <p>Рассматривается символ текста 'T' по индексу 1</p> <p>Выполняется переход по символу 'T' из узла 0</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла по данному символу возможно совершить переход в узел 1</p> <p>Переход по символу 'T' из узла 0 будет выполнен в узел 1</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>-----ПОИСК ХОРОШИХ ССЫЛОК-----</p> <p>■ По индексу 2 располагается шаблон T</p> <p>Вычисляется хорошая суффиксная ссылка для узла: 1</p> <p>Вычисляется суффиксная ссылка для узла: 1</p> <p>Узел корневой или первый после корня, поэтому суффиксная ссылка - 0</p> <p>■ Суффиксная ссылка узла 1 - 0</p> <p>Суффиксная ссылка указывает на 0, поэтому хорошая суффиксная ссылка тоже 0</p> <p>■ Хорошая суффиксная ссылка узла 1 - 0</p> <p>-----КОНЕЦ ПОИСКА ХОРОШИХ ССЫЛОК-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 1</p> <p>Рассматривается символ текста 'A' по индексу 2</p>
--	--

	<p>Выполняется переход по символу 'А' из узла 1</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла по данному символу возможно совершить переход в узел 2</p> <p>Переход по символу 'А' из узла 1 будет выполнен в узел 2</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>-----ПОИСК ХОРОШИХ ССЫЛОК-----</p> <p>Вычисляется хорошая суффиксная ссылка для узла: 2</p> <p>Вычисляется суффиксная ссылка для узла: 2</p> <p>Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 1 по символу 'А'</p> <p>■ Суффиксная ссылка узла 1 - 0</p> <p>Выполняется переход по символу 'А' из узла 0</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0</p> <p>Переход по символу 'А' из узла 0 будет выполнен в узел 0</p> <p>■ Суффиксная ссылка узла 2 - 0</p> <p>Суффиксная ссылка указывает на 0, поэтому хорошая суффиксная ссылка тоже 0</p> <p>■ Хорошая суффиксная ссылка узла 2 - 0</p> <p>-----КОНЕЦ ПОИСКА ХОРОШИХ ССЫЛОК-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 2</p> <p>Рассматривается символ текста 'G' по индексу 3</p> <p>Выполняется переход по символу 'G' из узла 2</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла по данному символу возможно совершить переход в узел 3</p>
--	---

		<p>Переход по символу 'G' из узла 2 будет выполнен в узел 3</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>-----ПОИСК ХОРОШИХ ССЫЛОК-----</p> <p>■ По индексу 2 располагается шаблон TAG</p> <p>Индекс прошлого найденного образца: 2</p> <p>Размер прошлого найденного образца: 3</p> <p>Индекс текущего найденного образца: 2</p> <p>Прошлый и текущий образцы пересекаются, так как индекс текущего образца попал в область прошлого образца</p> <p>Вычисляется хорошая суффиксная ссылка для узла: 3</p> <p>Вычисляется суффиксная ссылка для узла: 3</p> <p>Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 2 по символу 'G'</p> <p>■ Суффиксная ссылка узла 2 - 0</p> <p>Выполняется переход по символу 'G' из узла 0</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0</p> <p>Переход по символу 'G' из узла 0 будет выполнен в узел 0</p> <p>■ Суффиксная ссылка узла 3 - 0</p> <p>Суффиксная ссылка указывает на 0, поэтому хорошая суффиксная ссылка тоже 0</p> <p>■ Хорошая суффиксная ссылка узла 3 - 0</p> <p>-----КОНЕЦ ПОИСКА ХОРОШИХ ССЫЛОК-----</p> <p>2 2</p> <p>2 3</p> <p>Пересекающиеся шаблоны: TAG, T,</p> <p>Количество узлов в автомате 5</p>
2	AAAAA CCCCC	1 1 2 1

	3 A C AC	3 1 4 1 5 1 5 3 6 2 7 2 8 2 9 2 10 2 11 2 Пересекающиеся шаблоны: A, C, AC, Количество узлов в автомате 4
3	AGNTG NTTTA 5 AAAA TTT TT T GN	2 5 4 4 5 5 7 2 7 3 7 4 8 3 8 4 9 4 Пересекающиеся шаблоны: TTT, TT, T, Количество узлов в автомате 10

Тестирование второй программы.

Нумерация	Входные данные	Выходные данные
1	ACTANCA A\$\$\$ \$	----- Добавление шаблона A\$\$\$ в бор Рассматривается символ 'A'

	<p>Для данного символа был добавлен узел в бор, его номер: 1</p> <p>Рассматривается символ '\$'</p> <p>Для данного символа был добавлен узел в бор, его номер: 2</p> <p>Рассматривается символ '\$'</p> <p>Для данного символа был добавлен узел в бор, его номер: 3</p> <p>Рассматривается символ 'A'</p> <p>Для данного символа был добавлен узел в бор, его номер: 4</p> <p>Рассматривается символ '\$'</p> <p>Для данного символа был добавлен узел в бор, его номер: 5</p> <p>-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 0</p> <p>Рассматривается символ текста 'A' по индексу 0</p> <p>Выполняется переход по символу 'A' из узла 0</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла по данному символу возможно совершить переход в узел 1</p> <p>Переход по символу 'A' из узла 0 будет выполнен в узел 1</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p>
--	---

	<p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 1</p> <p>Рассматривается символ текста 'С' по индексу 1</p> <p>Выполняется переход по символу 'С' из узла 1</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла возможно совершить переход по джокеру в узел2</p> <p>Переход по символу 'С' из узла 1 будет выполнен в узел 2</p> <p>Был совершен переход по Джокеру, поэтому рассматриваемый символ был добавлен в список символов, которые находились 'под джокером', чтобы была возможность вычислить суффиксную ссылку</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 2</p> <p>Рассматривается символ текста 'Т' по индексу 2</p> <p>Выполняется переход по символу 'Т' из узла 2</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла возможно совершить переход по джокеру в узел3</p> <p>Переход по символу 'Т' из узла 2 будет выполнен в узел 3</p> <p>Был совершен переход по Джокеру, поэтому рассматриваемый символ был добавлен в список символов, которые находились 'под</p>
--	---

	<p>джокером', чтобы была возможность вычислить суффиксную ссылку</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 3</p> <p>Рассматривается символ текста 'A' по индексу 3</p> <p>Выполняется переход по символу 'A' из узла 3</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла по данному символу возможно совершить переход в узел 4</p> <p>Переход по символу 'A' из узла 3 будет выполнен в узел 4</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 4</p> <p>Рассматривается символ текста 'N' по индексу 4</p> <p>Выполняется переход по символу 'N' из узла 4</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла возможно совершить переход по джокеру в узел 5</p> <p>Переход по символу 'N' из узла 4 будет выполнен в узел 5</p> <p>Был совершен переход по Джокеру, поэтому рассматриваемый символ был добавлен в список символов, которые находились 'под джокером', чтобы была возможность вычислить суффиксную ссылку</p>
--	---

	<p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>Текущий узел является терминальным, найден шаблон по индексу 1</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 5</p> <p>Рассматривается символ текста 'С' по индексу 5</p> <p>Выполняется переход по символу 'С' из узла 5</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Невозможно совершить переход по заданному символу из заданного узла, поэтому необходимо вычислить переход по суффиксной ссылке</p> <p>Вычисляется суффиксная ссылка для узла: 5</p> <p>Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 4 по символу ПОД ДЖОКЕРОМ в тексте 'N'</p> <p>Вычисляется суффиксная ссылка для узла: 4</p> <p>Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 3 по СТАТИЧНОМУ символу 'А'</p> <p>Вычисляется суффиксная ссылка для узла: 3</p> <p>Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 2 по символу ПОД ДЖОКЕРОМ в тексте 'Т'</p>
--	--

		<p>Вычисляется суффиксная ссылка для узла: 2</p> <p>Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 1 по символу ПОД ДЖОКЕРОМ в тексте 'С'</p> <p>Вычисляется суффиксная ссылка для узла: 1</p> <p>Узел корневой или первый после корня, поэтому суффиксная ссылка - 0</p> <p>■ Суффиксная ссылка узла 1 - 0</p> <p>Выполняется переход по символу 'С' из узла 0</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0</p> <p>Переход по символу 'С' из узла 0 будет выполнен в узел 0</p> <p>■ Суффиксная ссылка узла 2 - 0</p> <p>Выполняется переход по символу 'Т' из узла 0</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0</p> <p>Переход по символу 'Т' из узла 0 будет выполнен в узел 0</p> <p>■ Суффиксная ссылка узла 3 - 0</p> <p>Выполняется переход по символу 'А' из узла 0</p>
--	--	--

	<p>Переход по символу 'A' из узла 0 будет выполнен в узел 1</p> <p>■ Суффиксная ссылка узла 4 - 1</p> <p>Выполняется переход по символу 'N' из узла 1</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла возможно совершить переход по джокеру в узел2</p> <p>Переход по символу 'N' из узла 1 будет выполнен в узел 2</p> <p>■ Суффиксная ссылка узла 5 - 2</p> <p>Выполняется переход по символу 'C' из узла 2</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла возможно совершить переход по джокеру в узел3</p> <p>Переход по символу 'C' из узла 2 будет выполнен в узел 3</p> <p>Переход по символу 'C' из узла 5 будет выполнен в узел 3</p> <p>Был совершен переход по Джокеру, поэтому рассматриваемый символ был добавлен в список символов, которые находились 'под джокером', чтобы была возможность вычислить суффиксную ссылку</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>-----ПЕРЕХОД-----</p> <p>Текущее состояние автомата: 3</p> <p>Рассматривается символ текста 'A' по индексу 6</p>
--	--

		<p>Выполняется переход по символу 'A' из узла 3</p> <p>Для данного символа из данного узла не вычислен переход</p> <p>Из данного узла по данному символу возможно совершить переход в узел 4</p> <p>Переход по символу 'A' из узла 3 будет выполнен в узел 4</p> <p>-----КОНЕЦ ПЕРЕХОДА-----</p> <p>Индексы найденных образцов:</p> <p>1</p> <p>Нет экземпляров образца, которые пересекаются между собой</p> <p>Количество узлов в автомате 6</p>
2	ACTACTACTNG \$TACT\$ \$	2 5 Имеются экземпляры образца \$TACT\$, которые пересекаются между собой Количество узлов в автомате 7
3	NATNATNATNATACGTNAT NAT\$\$\$ \$	1 4 7 10 Имеются экземпляры образца NAT\$\$\$, которые пересекаются между собой Количество узлов в автомате 7
4	ANAA A\$ \$	1 3

		Нет экземпляров образца, которые пересекаются между собой
--	--	---

Выводы.

Был изучен и реализован алгоритм Ахо-Корасика. Также была реализована модификация программы, которая ищет в тексте шаблоны, включающие в себя символ джокера.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5_1.cpp

```
#include "vector"
#include "iostream"
#include "map"
#include "cstring"
#include "algorithm"
#include "set"

#define DEBUG

const int ALPHABET_LENGTH = 5;
std::map<char, int> ALPHABET = {{'A', 0},
                                {'C', 1},
                                {'G', 2},
                                {'T', 3},
                                {'N', 4}};

struct BohrVertex {
    int nextVertices[ALPHABET_LENGTH];
    int patternNumber;
    bool isTerminal;
    int suffixLink;
    int goodSuffixLink;
    int parent;
    char symbol;
    int transitions[ALPHABET_LENGTH];

    static BohrVertex createVertex(int parentNumber, char symbolOfEdge) {
        auto vertex = BohrVertex();
        memset(vertex.nextVertices, 255, sizeof(vertex.nextVertices));
        memset(vertex.transitions, 255, sizeof(vertex.transitions));
        vertex.isTerminal = false;
        vertex.suffixLink = -1;
        vertex.parent = parentNumber;
        vertex.symbol = symbolOfEdge;
        vertex.goodSuffixLink = -1;
        return vertex;
    }
};

class Bohr {

    std::vector<BohrVertex> vertexes {};
    std::vector<std::string> patterns {};
    std::string & text;

    int indexOfLastPattern;
    int numOfLastPattern;
    std::set<int> crosses {};

public:

    Bohr(std::string & text) : text(text) {
        auto root = BohrVertex::createVertex(0, '\0');
```

```

        vertexes.push_back(root);

        indexOfLastPattern = 0;
        numOfLastPattern = -1;
    }

    void addStringToBohr(const std::string & string, int indexOfPattern) {
        int number = 0;

#ifdef DEBUG
        std::cout << "-----\n" <<
std::endl;
        std::cout << "Добавление шаблона " << string << " в бор\n";
#endif

        for (char symbol : string) {

#ifdef DEBUG
            std::cout << "Рассматривается символ " << "'" << symbol << "'\n";
#endif

            int ordinal = ALPHABET[symbol];
            if (vertexes[number].nextVertices[ordinal] == -1) {

                auto vertex = BohrVertex::createVertex(number, symbol);
                vertexes.push_back(vertex);
                vertexes[number].nextVertices[ordinal] = (int) vertexes.size() - 1;

#ifdef DEBUG
                std::cout << "Для данного символа был добавлен узел в бор,
его номер: " << vertexes[number].nextVertices[ordinal] << "\n\n";
#endif

            }

#ifdef DEBUG
            else { std::cout << "Узел для данного символа уже есть в боре,
его номер: " << vertexes[number].nextVertices[ordinal] << "\n\n"; }
#endif

            number = vertexes[number].nextVertices[ordinal];
        }

#ifdef DEBUG
        std::cout << "-----" <<
std::endl;
#endif

        vertexes[number].isTerminal = true;
        vertexes[number].patternNumber = indexOfPattern;
        patterns.push_back(string);
    }

    int getSuffixLink(int vertex);
    int getTransition(int vertex, char symbol);
    int getGoodSuffixLink(int vertex);
    void doAlgorithm();
};

int Bohr::getSuffixLink(int vertex) {

```

```

        if (vertexes[vertex].suffixLink == -1) {

#ifdef DEBUG
            std::cout << "\nВычисляется суффиксная ссылка для узла: " << vertex
<< "\n";
#endif

            if (vertex == 0 || vertexes[vertex].parent == 0) {

#ifdef DEBUG
                std::cout << "Узел корневой или первый после корня, поэтому суф-
фиксная ссылка - 0\n";
#endif

                vertexes[vertex].suffixLink = 0;
            }

            else {
                int parent = vertexes[vertex].parent;
                char symbol = vertexes[vertex].symbol;
#ifdef DEBUG
                std::cout << "Для вычисления суффиксной ссылки узла нужно совер-
шить переход"
                        " по суффиксной ссылке родителя " << parent << " по
символу '" << symbol << "'\n";
#endif
                vertexes[vertex].suffixLink = getTransition(getSuffixLink(par-
ent), symbol);
            }

        }

#ifdef DEBUG
        std::cout << "■ Суффиксная ссылка узла " << vertex << " - " << ver-
texes[vertex].suffixLink << "\n";
#endif

        return vertexes[vertex].suffixLink;
    }

    int Bohr::getTransition(int vertex, char symbol) {

        char ordinal = ALPHABET[symbol];

#ifdef DEBUG
        std::cout << "\nВыполняется переход по символу '" << symbol << "' << "
из узла " << vertex << "\n";
#endif

        if (vertexes[vertex].transitions[ordinal] == -1) {

#ifdef DEBUG
            std::cout << "Для данного символа из данного узла не вычислен пере-
ход\n";
#endif

            if (vertexes[vertex].nextVertices[ordinal] != -1) {

                int nextVertex = vertexes[vertex].nextVertices[ordinal];
#ifdef DEBUG

```

```

        std::cout << "Из данного узла по данному символу возможно совер-
шить переход в узел " << nextVertex << "\n";
    #endif

    vertexes[vertex].transitions[ordinal] = nextVertex;

    } else {

        if (vertex == 0) {
    #ifdef DEBUG
            std::cout << "Данный узел является корнем, и невозможно со-
вершить переход по заданному символу, поэтому переход будет в узел 0\n";
    #endif
            vertexes[vertex].transitions[ordinal] = 0;
        }

        else {
    #ifdef DEBUG
            std::cout << "Невозможно совершить переход по заданному сим-
волу из заданного узла, поэтому необходимо вычислить переход по суффиксной
ссылке\n";
    #endif
            vertexes[vertex].transitions[ordinal] = getTransition(getSuf-
fixLink(vertex), symbol);
        }
    }

    }

    #ifdef DEBUG
        std::cout << "Переход по символу '" << symbol << "' из узла " << vertex
<< " будет выполнен в узел " << vertexes[vertex].transitions[ordinal] <<
"\n";
    #endif

    return vertexes[vertex].transitions[ordinal];
}

int Bohr::getGoodSuffixLink(int vertex) {

    #ifdef DEBUG
        std::cout << "\nВычисляется хорошая суффиксная ссылка для узла: " << ver-
tex << "\n";
    #endif

    if (vertexes[vertex].goodSuffixLink == -1) {
        int suffixLink = getSuffixLink(vertex);

        if (suffixLink == 0) {
    #ifdef DEBUG
            std::cout << "Суффиксная ссылка указывает на 0, поэтому хорошая
суффиксная ссылка тоже 0\n";
    #endif
            vertexes[vertex].goodSuffixLink = 0;
        }
        else {
            if (vertexes[suffixLink].isTerminal) {
    #ifdef DEBUG
            std::cout << "Узел по суффиксной ссылке является терминаль-
ным, поэтому суффиксная ссылка хорошая\n";
    #endif
            vertexes[vertex].goodSuffixLink = suffixLink;
        }
    }
}

```

```

        }
        else {
#ifdef DEBUG
            std::cout << "Узел по суффиксной ссылке не терминальный,
            поэтому вычисляется хорошая суффиксная ссылка узла" << suffixLink << "\n";
#endif
            vertexes[vertex].goodSuffixLink = getGoodSuffixLink(suffixLink);
        }
    }
}

#ifdef DEBUG
    std::cout << "■ Хорошая суффиксная ссылка узла " << vertex << " - " <<
    vertexes[vertex].goodSuffixLink << "\n";
#endif
    return vertexes[vertex].goodSuffixLink;
}

bool resultComparator(std::pair<int, int> first, std::pair<int, int> second)
{
    if (first.first < second.first) return true;
    if (first.first == second.first)
        return first.second < second.second;
    return false;
}

void Bohr::doAlgorithm() {
    int vertex = 0;
    auto result = std::vector<std::pair<int, int>>();

    for (int i = 0; i < text.size(); ++i) {
        char symbol = text[i];

#ifdef DEBUG
            std::cout << "-----ПЕРЕХОД-----\n";
            std::cout << "Текущее состояние автомата: " << vertex << "\n";
            std::cout << "Рассматривается символ текста '" << symbol << "' " <<
            "по индексу " << i << "\n";
#endif

            vertex = getTransition(vertex, symbol);

#ifdef DEBUG
            std::cout << "-----КОНЕЦ ПЕРЕХОДА-----\n";
#endif

            bool maxSizePatternMarked = false;

#ifdef DEBUG
            std::cout << "-----ПОИСК ХОРОШИХ ССЫЛОК-----\n";
#endif

            for (int link = vertex; link; link = getGoodSuffixLink(link)) {
                if (vertexes[link].isTerminal) {

```



```

        int patternNumber = vertexes[link].patternNumber;
        int indexOfPattern = i - (int) patterns[patternNumber].size()
+ 2;

#ifdef DEBUG
        std::cout << "■ По индексу " << indexOfPattern << "
располагается шаблон " << patterns[patternNumber] << "\n";
#endif

        if (numOfLastPattern != - 1 && indexOfPattern < in-
dexOfLastPattern + patterns[numOfLastPattern].size()) {
#ifdef DEBUG
            std::cout << "Индекс прошлого найденного образца: " <<
indexOfLastPattern << "\n";
            std::cout << "Размер прошлого найденного образца: " << pat-
terns[vertexes[vertex].patternNumber].size() << "\n";
            std::cout << "Индекс текущего найденного образца: " << in-
dexOfPattern << "\n";
            std::cout << "Прошлый и текущий образцы пересекаются, так как
индекс"
                                " текущего образца попал в область прошлого об-
разца" << "\n";
#endif

            crosses.insert(numOfLastPattern);
            crosses.insert(vertexes[link].patternNumber);
        }

        result.emplace_back(indexOfPattern, patternNumber + 1);

        if (!maxSizePatternMarked) {
            numOfLastPattern = patternNumber;
            indexOfLastPattern = indexOfPattern;
            maxSizePatternMarked = true;
        }
    }
}

#ifdef DEBUG
    std::cout << "-----КОНЕЦ ПОИСКА ХОРОШИХ ССЫЛОК-----\n";
#endif

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

    for (auto pair : result) {
        std::cout << pair.first << " " << pair.second << std::endl;
    }

#ifdef DEBUG
    std::cout << "Пересекающиеся шаблоны: ";
    for (auto cross : crosses) {
        std::cout << patterns[cross] << ", ";
    }

    if (crosses.empty())
        std::cout << " нет\n";

    std::cout << "\nКоличество узлов в автомате " << vertexes.size() << "\n";
#endif
}

```

```

int main() {
    int countPatterns;
    std::string text, pattern;
    std::cin >> text >> countPatterns;

    auto bohr = Bohr(text);

    auto patterns = std::vector<std::string>();
    for (int i = 0; i < countPatterns; ++i) {
        std::cin >> pattern;
        patterns.push_back(pattern);
        bohr.addStringToBohr(pattern, i);
    }

    bohr.doAlgorithm();
    return 0;
}

```

Название файла: lab5_2.cpp

```
#include "vector"
#include "iostream"
#include "map"
#include "cstring"
#include "algorithm"

const int ALPHABET_LENGTH = 6;
std::map<char, int> ALPHABET = {{'A', 0},
                                {'C', 1},
                                {'G', 2},
                                {'T', 3},
                                {'N', 4}};

struct BohrVertex {
    int nextVertices[ALPHABET_LENGTH];
    bool isTerminal;
    int suffixLink;

    int parent;
    char symbol;
    int transitions[ALPHABET_LENGTH];

    static BohrVertex createVertex(int parentNumber, char symbolOfEdge) {
        auto vertex = BohrVertex();
        memset(vertex.nextVertices, 255, sizeof(vertex.nextVertices));
        memset(vertex.transitions, 255, sizeof(vertex.transitions));
        vertex.isTerminal = false;
        vertex.suffixLink = -1;
        vertex.parent = parentNumber;
        vertex.symbol = symbolOfEdge;

        return vertex;
    }
};

class Bohr {
    std::vector<BohrVertex> vertexes;

    char joker;
    std::string text;
    std::string pattern;

    int indexOfLastFoundedPattern;
    bool isThereCrossing;

    void addStringToBohr(const std::string & string);
    int getSuffixLink(int vertex);
    int getTransition(int vertex, char symbol);
    std::string getPatternFromText(int i, int sizeOfPattern);

public:
    std::vector<char> symbolsUnderJoker;
    int numOfSymbolUnderJoker{};
    Bohr(char joker, std::string & text, std::string & pattern);
    void doAlgorithm();
};
```

```

};

Bohr::Bohr(char joker, std::string &text, std::string &pattern)
    : joker(joker), text(text), pattern(pattern) {
    vertexes = std::vector<BohrVertex>();
    auto root = BohrVertex::createVertex(0, '\\0');
    vertexes.push_back(root);

    indexOfLastFoundedPattern = - pattern.size();
    isThereCrossing = false;
}

void Bohr::addStringToBohr(const std::string &string) {
    int number = 0;

#ifdef DEBUG
    std::cout << "-----\\n" <<
std::endl;
    std::cout << "Добавление шаблона " << string << " в бор\\n";
#endif

    for (char symbol : string) {

#ifdef DEBUG
        std::cout << "Рассматривается символ " << "'" << symbol << "'\\n";
#endif

        int ordinal = ALPHABET[symbol];
        if (vertexes[number].nextVertices[ordinal] == -1) {

            auto vertex = BohrVertex::createVertex(number, symbol);
            vertexes.push_back(vertex);
            vertexes[number].nextVertices[ordinal] = (int) vertexes.size() -
1;

#ifdef DEBUG
            std::cout << "Для данного символа был добавлен узел в бор, его
номер: " << vertexes[number].nextVertices[ordinal] << "\\n\\n";
#endif

        }
#ifdef DEBUG
        else { std::cout << "Узел для данного символа уже есть в боре, его
номер: " << vertexes[number].nextVertices[ordinal] << "\\n\\n"; }
#endif
        number = vertexes[number].nextVertices[ordinal];
    }

#ifdef DEBUG
    std::cout << "-----" << std::endl;
#endif

    vertexes[number].isTerminal = true;
}

int Bohr::getSuffixLink(int vertex) {

#ifdef DEBUG
    std::cout << "\\nВычисляется суффиксная ссылка для узла: " << vertex <<

```

```

"\n";
#endif

    if (vertex == 0 || vertexes[vertex].parent == 0) {

#ifdef DEBUG
        std::cout << "Узел корневой или первый после корня, поэтому суффикс-
ная ссылка - 0\n";
#endif
        vertexes[vertex].suffixLink = 0;
    }
    else {
        int parent = vertexes[vertex].parent;
        if (vertexes[vertex].symbol == joker) {
            char symbol = symbolsUnderJoker[numOfSymbolUnderJoker--];
#ifdef DEBUG
                std::cout << "Для вычисления суффиксной ссылки узла нужно совер-
шить переход"
                        " по суффиксной ссылке родителя " << parent << " по
символу ПОД ДЖОКЕРОМ в тексте '" << symbol << "'\n";
#endif
                vertexes[vertex].suffixLink = getTransition(getSuffixLink(par-
ent), symbol);
            }
            else {
                char symbol = vertexes[vertex].symbol;
#ifdef DEBUG
                    std::cout << "Для вычисления суффиксной ссылки узла нужно совер-
шить переход"
                            " по суффиксной ссылке родителя " << parent << " по
СТАТИЧНОМУ символу '" << symbol << "'\n";
#endif
                    vertexes[vertex].suffixLink = getTransition(getSuffixLink(par-
ent), symbol);
                }
            }
        }

#ifdef DEBUG
        std::cout << "■ Суффиксная ссылка узла " << vertex << " - " << ver-
texes[vertex].suffixLink << "\n";
#endif

        return vertexes[vertex].suffixLink;
    }
}

int Bohr::getTransition(int vertex, char symbol) {
    char ordinal = ALPHABET[symbol];

#ifdef DEBUG
        std::cout << "\nВыполняется переход по символу '" << symbol << "'" << "
из узла " << vertex << "\n";
#endif

#ifdef DEBUG
        std::cout << "Для данного символа из данного узла не вычислен пере-
ход\n";
#endif
}

```

```

        if (vertexes[vertex].nextVertices[ordinal] != -1) {
            int nextVertex = vertexes[vertex].nextVertices[ordinal];
#ifdef DEBUG
                std::cout << "Из данного узла по данному символу возможно совершить
переход в узел " << nextVertex << "\n";
#endif
            vertexes[vertex].transitions[ordinal] = nextVertex;
        }

        else if (vertexes[vertex].nextVertices[5] != -1) {
            int nextVertex = vertexes[vertex].nextVertices[5];
#ifdef DEBUG
                std::cout << "Из данного узла возможно совершить переход по джокеру в
узел" << nextVertex << "\n";
#endif
            vertexes[vertex].transitions[ordinal] = nextVertex;
        }

        else {
            if (vertex == 0) {
#ifdef DEBUG
                std::cout << "Данный узел является корнем, и невозможно совершить
переход по заданному символу, поэтому переход будет в узел 0\n";
#endif
                vertexes[vertex].transitions[ordinal] = 0;
            }
            else {
#ifdef DEBUG
                std::cout << "Невозможно совершить переход по заданному символу
из заданного узла, поэтому необходимо вычислить переход по суффиксной
ссылке\n";
#endif
                vertexes[vertex].transitions[ordinal] = getTransition(getSuf-
fixLink(vertex), symbol);
            }
        }

#ifdef DEBUG
        std::cout << "Переход по символу '" << symbol << "' из узла " << vertex
<< " будет выполнен в узел " << vertexes[vertex].transitions[ordinal] <<
"\n";
#endif

        return vertexes[vertex].transitions[ordinal];
    }

void Bohr::doAlgorithm() {
    addStringToBohr(pattern);

    int vertex = 0;
    auto results = std::map<int, std::string>();

    for (int i = 0; i < text.size(); ++i) {
        char symbol = text[i];

#ifdef DEBUG
            std::cout << "-----ПЕРЕХОД-----\n";
            std::cout << "Текущее состояние автомата: " << vertex << "\n";
            std::cout << "Рассматривается символ текста '" << symbol << "' " <<
"по индексу " << i << "\n";
#endif

```

```

#endif

    vertex = getTransition(vertex, symbol);

    if (vertexes[vertex].symbol == joker) {
#ifdef DEBUG
        std::cout << "Был совершен переход по Джокеру, поэтому рассматри-
ваемый символ"
                    " был добавлен в список символов, которые находились
'под джокером'"
                    ", чтобы была возможность вычислить суффиксную
ссылку\n";
#endif
        symbolsUnderJoker.push_back(symbol);
    }
    numOfSymbolUnderJoker = (int) symbolsUnderJoker.size() - 1;

#ifdef DEBUG
    std::cout << "-----КОНЕЦ ПЕРЕХОДА-----\n";
#endif

    if (vertexes[vertex].isTerminal) {
        int indexOfPattern = i - (int) pattern.size() + 2;

#ifdef DEBUG
        std::cout << "Текущий узел является терминальным, найден шаблон
по индексу " << indexOfPattern << "\n";
#endif

        if (!isThereCrossing && indexOfPattern <
indexOfLastFoundedPattern + pattern.size()) {

#ifdef DEBUG
            std::cout << "Текущий шаблон пересекается с прошлым найденным
шаблоном в строке поиска\n";
#endif

            isThereCrossing = true;
        }
        indexOfLastFoundedPattern = indexOfPattern;

        results[indexOfPattern] = getPatternFromText(indexOfPattern - 1,
pattern.size());
    }
}

#ifdef DEBUG
    std::cout << "Индексы найденных образцов:\n";
#endif

    for (const auto& result : results)
        std::cout << result.first << std::endl;

#ifdef DEBUG
    if (isThereCrossing)
        std::cout << "Имеются экземпляры образца " << pattern << ", которые
пересекаются между собой" << std::endl;
    else
        std::cout << "Нет экземпляров образца, которые пересекаются между со-
бой" << std::endl;

```

```

        std::cout << "\nКоличество узлов в автомате " << vertexes.size() << "\n";
    #endif

}

std::string Bohr::getPatternFromText(int i, int sizeOfPattern) {
    auto foundPattern = std::string();
    int endOfPattern = i + sizeOfPattern;
    for (; i < endOfPattern; ++i) foundPattern += text[i];
    return foundPattern;
}

int main() {
    char joker;
    std::string text, pattern;
    std::cin >> text >> pattern >> joker;
    auto bohr = Bohr(joker, text, pattern);
    ALPHABET[joker] = 5;
    bohr.doAlgorithm();
    return 0;
}

```