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-Корасика

Студент гр. 9382 _____ Русинов Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Разработать программу, занимающуюся поиском шаблонов в строке, используя алгоритм Ахо-Корасика. Также разработать программу, которая ищет шаблон в строке, который содержит джокера.

Задание 1

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

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

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

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

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 P*P* необходимо найти все вхождения P в текст T.

Например, образец a*b*??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
```

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 ‘a’ 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”.

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	NTAG 3 TAGT ---------- 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 Рассматривается символ 'T'

		Для данного символа был добавлен узел в бор, его номер: 4 ---------- TAG ---------- Добавление шаблона TAG в бор Рассматривается символ 'T' Узел для данного символа уже есть в боре, его номер: 1 Рассматривается символ 'A' Узел для данного символа уже есть в боре, его номер: 2 Рассматривается символ 'G' Узел для данного символа уже есть в боре, его номер: 3 ---------- T ---------- Добавление шаблона T в бор Рассматривается символ 'T' Узел для данного символа уже есть в боре, его номер: 1 ---------- ----------ПЕРЕХОД---------- Текущее состояние автомата: 0 Рассматривается символ текста 'N' по индексу 0 Выполняется переход по символу 'N' из узла 0 Для данного символа из данного узла не вычислен переход Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0

		Переход по символу 'N' из узла 0 будет выполнен в узел 0 --------КОНЕЦ ПЕРЕХОДА-------- ----------ПЕРЕХОД---------- Текущее состояние автомата: 0 Рассматривается символ текста 'T' по индексу 1 Выполняется переход по символу 'T' из узла 0 Для данного символа из данного узла не вычислен переход Из данного узла по данному символу возможно совершить переход в узел 1 Переход по символу 'T' из узла 0 будет выполнен в узел 1 --------КОНЕЦ ПЕРЕХОДА-------- --------ПОИСК ХОРОШИХ ССЫЛОК-------- Так как новый узел терминальный, то необходимо вычислить хорошие суффиксные ссылки, поскольку в образец могут входить другие образцы ■ По индексу 2 располагается шаблон T Вычисляется хорошая суффиксная ссылка для узла: 1 Вычисляется суффиксная ссылка для узла: 1 Узел корневой или первый после корня, поэтому суффиксная ссылка - 0 ■ Суффиксная ссылка узла 1 - 0 Суффиксная ссылка указывает на 0, поэтому хорошая суффиксная ссылка тоже 0 ■ Хорошая суффиксная ссылка узла 1 - 0 -----КОНЕЦ ПОИСКА ХОРОШИХ ССЫЛОК----- ----------ПЕРЕХОД---------- Текущее состояние автомата: 1 Рассматривается символ текста 'A' по индексу 2 Выполняется переход по символу 'A' из узла 1

		Для данного символа из данного узла не вычислен переход Из данного узла по данному символу возможно совершить переход в узел 2 Переход по символу 'A' из узла 1 будет выполнен в узел 2 --------КОНЕЦ ПЕРЕХОДА-------- ----------ПЕРЕХОД---------- Текущее состояние автомата: 2 Рассматривается символ текста 'G' по индексу 3 Выполняется переход по символу 'G' из узла 2 Для данного символа из данного узла не вычислен переход Из данного узла по данному символу возможно совершить переход в узел 3 Переход по символу 'G' из узла 2 будет выполнен в узел 3 --------КОНЕЦ ПЕРЕХОДА-------- Индекс прошлого найденного образца: 1 Размер прошлого найденного образца: 3 Индекс текущего найденного образца: 1 Прошлый и текущий образцы пересекаются, так как индекс текущего образца попал в область прошлого образца --------ПОИСК ХОРОШИХ ССЫЛОК-------- Так как новый узел терминальный, то необходимо вычислить хорошие суффиксные ссылки, поскольку в образец могут входить другие образцы ■ По индексу 2 располагается шаблон TAG Вычисляется хорошая суффиксная ссылка для узла: 3 Вычисляется суффиксная ссылка для узла: 3 Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 2 по символу 'G' Вычисляется суффиксная ссылка для узла: 2

		Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 1 по символу 'A' ■ Суффиксная ссылка узла 1 - 0 Выполняется переход по символу 'A' из узла 0 Для данного символа из данного узла не вычислен переход Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0 Переход по символу 'A' из узла 0 будет выполнен в узел 0 ■ Суффиксная ссылка узла 2 - 0 Выполняется переход по символу 'G' из узла 0 Для данного символа из данного узла не вычислен переход Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0 Переход по символу 'G' из узла 0 будет выполнен в узел 0 ■ Суффиксная ссылка узла 3 - 0 Суффиксная ссылка указывает на 0, поэтому хорошая суффиксная ссылка тоже 0 ■ Хорошая суффиксная ссылка узла 3 - 0 -----КОНЕЦ ПОИСКА ХОРОШИХ ССЫЛОК----- 2 2 2 3 Пересекающиеся шаблоны: TAG, T, Количество узлов в автомате 5
2	AAAAA CCCCCC 3 A C AC	1 1 2 1 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$$A$ $	---------- Добавление шаблона A$$A$ в бор Рассматривается символ 'A' Для данного символа был добавлен узел в бор, его номер: 1 Рассматривается символ '$' Для данного символа был добавлен узел в бор, его номер: 2

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

		Для данного символа из данного узла не вычислен переход Из данного узла возможно совершить переход по джокеру в узел2 Переход по символу 'C' из узла 1 будет выполнен в узел 2 Был совершен переход по Джокеру, поэтому рассматриваемый символ был добавлен в список символов, которые находились 'под джокером', чтобы была возможность вычислить суффиксную ссылку --------КОНЕЦ ПЕРЕХОДА-------- ----------ПЕРЕХОД---------- Текущее состояние автомата: 2 Рассматривается символ текста 'T' по индексу 2 Выполняется переход по символу 'T' из узла 2 Для данного символа из данного узла не вычислен переход Из данного узла возможно совершить переход по джокеру в узел3 Переход по символу 'T' из узла 2 будет выполнен в узел 3 Был совершен переход по Джокеру, поэтому рассматриваемый символ был добавлен в список символов, которые находились 'под джокером', чтобы была возможность вычислить суффиксную ссылку --------КОНЕЦ ПЕРЕХОДА-------- ----------ПЕРЕХОД---------- Текущее состояние автомата: 3

		Рассматривается символ текста 'A' по индексу 3 Выполняется переход по символу 'A' из узла 3 Для данного символа из данного узла не вычислен переход Из данного узла по данному символу возможно совершить переход в узел 4 Переход по символу 'A' из узла 3 будет выполнен в узел 4 --------КОНЕЦ ПЕРЕХОДА-------- ----------ПЕРЕХОД---------- Текущее состояние автомата: 4 Рассматривается символ текста 'N' по индексу 4 Выполняется переход по символу 'N' из узла 4 Для данного символа из данного узла не вычислен переход Из данного узла возможно совершить переход по джокеру в узел5 Переход по символу 'N' из узла 4 будет выполнен в узел 5 Был совершен переход по Джокеру, поэтому рассматриваемый символ был добавлен в список символов, которые находились 'под джокером', чтобы была возможность вычислить суффиксную ссылку --------КОНЕЦ ПЕРЕХОДА-------- Текущий узел является терминальным, найден шаблон по индексу 1 ----------ПЕРЕХОД---------- Текущее состояние автомата: 5

		Рассматривается символ текста 'C' по индексу 5 Выполняется переход по символу 'C' из узла 5 Для данного символа из данного узла не вычислен переход Невозможно совершить переход по заданному символу из заданного узла, поэтому необходимо вычислить переход по суффиксной ссылке Вычисляется суффиксная ссылка для узла: 5 Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 4 по символу ПОД ДЖОКЕРОМ в тексте 'N' Вычисляется суффиксная ссылка для узла: 4 Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 3 по СТАТИЧНОМУ символу 'A' Вычисляется суффиксная ссылка для узла: 3 Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной ссылке родителя 2 по символу ПОД ДЖОКЕРОМ в тексте 'T' Вычисляется суффиксная ссылка для узла: 2 Для вычисления суффиксной ссылки узла нужно совершить переход по суффиксной

		ссылке родителя 1 по символу ПОД ДЖОКЕРОМ в тексте 'C' Вычисляется суффиксная ссылка для узла: 1 Узел корневой или первый после корня, поэтому суффиксная ссылка - 0 ■ Суффиксная ссылка узла 1 - 0 Выполняется переход по символу 'C' из узла 0 Для данного символа из данного узла не вычислен переход Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0 Переход по символу 'C' из узла 0 будет выполнен в узел 0 ■ Суффиксная ссылка узла 2 - 0 Выполняется переход по символу 'T' из узла 0 Для данного символа из данного узла не вычислен переход Данный узел является корнем, и невозможно совершить переход по заданному символу, поэтому переход будет в узел 0 Переход по символу 'T' из узла 0 будет выполнен в узел 0 ■ Суффиксная ссылка узла 3 - 0 Выполняется переход по символу 'A' из узла 0 Переход по символу 'A' из узла 0 будет выполнен в узел 1 ■ Суффиксная ссылка узла 4 - 1

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

		Из данного узла по данному символу возможно совершить переход в узел 4 Переход по символу 'A' из узла 3 будет выполнен в узел 4 --------КОНЕЦ ПЕРЕХОДА-------- Индексы найденных образцов: 1 Нет экземпляров образца, которые пересекаются между собой Количество узлов в автомате 6
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) ver-
texes.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, по-
этому вычисляется хорошая суффиксная ссылка узла" << suffixLink << "\n";
#endif
                vertexes[vertex].goodSuffixLink = getGoodSuffixLink(suf-
fixLink);
            }
        }
    }

#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
```

```
        if (vertexes[vertex].isTerminal) {

            int indexOfCurrentPattern = i - (int) patterns[vertexes[ver-
tex].patternNumber].size() + 1;
            if (numOfLastPattern != - 1 && indexOfCurrentPattern < in-
dexOfLastPattern + patterns[numOfLastPattern].size()) {

#ifdef DEBUG
                std::cout << "Индекс прошлого найденного образца: " << in-
dexOfLastPattern << "\n";
                std::cout << "Размер прошлого найденного образца: " << pat-
terns[vertexes[vertex].patternNumber].size() << "\n";
                std::cout << "Индекс текущего найденного образца: " << in-
dexOfCurrentPattern << "\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[vertex].patternNumber);
            }

            numOfLastPattern = vertexes[vertex].patternNumber;
            indexOfLastPattern = i - (int) patterns[numOfLastPattern].size()
+ 1;

#ifdef DEBUG
            std::cout << "--------ПОИСК ХОРОШИХ ССЫЛОК--------\n";
            std::cout << "Так как новый узел терминальный, то необходимо вы-
числить хорошие суффиксные ссылки,"
                         " поскольку в образец могут входить другие об-
разцы\n";
#endif

            auto innerPatterns = std::vector<int>();

            // Все внутренние шаблоны (хорошие суффиксные ссылки) точно пере-
секаются
            // Если в innerPatterns будет > 1 элемента, значит нужно добавить
все эти элементы в множество пересекающихся

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

                innerPatterns.push_back(patternNumber);

                int indexOfPattern = i - (int) patterns[patternNumber].size()
+ 2;
#ifdef DEBUG
                std::cout << "■ По индексу " << indexOfPattern << " распола-
гается шаблон " << patterns[patternNumber] << "\n";
#endif
                result.emplace_back(indexOfPattern, patternNumber + 1);
            }

            if (innerPatterns.size() > 1)
                for (auto p : innerPatterns) crosses.insert(p);
```

```
#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;
    }

    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";

}


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) {

    if (vertexes[vertex].suffixLink == -1 || vertexes[vertex].symbol ==
joker) {
```

```
#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;
            if (vertexes[vertex].symbol == joker) {
                char symbol = symbolsUnderJoker[numOfSymbolUnderJoker--];
#ifdef DEBUG
                std::cout << "Для вычисления суффиксной ссылки узла нужно со-
вершить переход"
                             " по суффиксной ссылке родителя " << parent << "
по символу ПОД ДЖОКЕРОМ в тексте '" << symbol << "'\n";
#endif
                vertexes[vertex].suffixLink = getTransition(getSuf-
fixLink(parent), symbol);
            }
            else {
                char symbol = vertexes[vertex].symbol;
#ifdef DEBUG
                std::cout << "Для вычисления суффиксной ссылки узла нужно со-
вершить переход"
                             " по суффиксной ссылке родителя " << parent << "
по СТАТИЧНОМУ символу '" << symbol << "'\n";
#endif
                vertexes[vertex].suffixLink = getTransition(getSuf-
fixLink(parent), 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 || vertexes[vertex].sym-
bol == joker) {
```

```
#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 (vertexes[vertex].nextVertices[5] != -1) {
            int nextVertex = vertexes[vertex].nextVertices[5];
#ifdef DEBUG
            std::cout << "Из данного узла возможно совершить переход по джо-
керу 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];
}


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

        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 < indexOfLastFoundedPat-
tern + 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;

    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";

}

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;
}
```