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

Студентка гр. 9382 _____ Пя С.

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

На вход подаются строка, количество шаблонов и шаблоны. Алгоритм начинается с построения бора. Шаблоны заносятся в бор в качестве узлов, которым дают свой уникальный номер в боре. В качестве имени выступает символ, нумерация идет по порядку. Построение начинается с корня, поэтому при совпадении начальных символов у шаблонов, они не вносятся в бор повторно. Также последние узлы помечаются в качестве конечных.

Затем начинается работа алгоритма. Проходимся по строке, начиная с первого символа. Каждый раз пытаемся совершить переход к следующему узлу по дереву, начиная с текущей позиции. Если переход можно осуществить (имя следующего узла совпадает с текущим символом строки), то переходим, если нельзя, то нам нужно перейти по суффиксной ссылке родителя текущего узла и перейти на следующий узел с совпадающим именем с текущим символом строки. Если это невозможно, то мы снова переходим по суффиксной ссылке родителя, пока не получится либо пока суффиксная ссылка не будет указывать на корень.

Суффиксная ссылка – номер узла, на котором заканчивается наибольший собственный суффикс, если построить строку из имен узлов, начиная с корня.

Суффиксная ссылка для узла находится таким образом:

Если узел или родитель узла являются корнем, то суффиксная ссылка указывает на корень, иначе мы переходим по суффиксной ссылке родителя узла и пытаемся совершить переход к следующему узлу.

Как только для текущего символа строки мы осуществили переход, пытаемся проверить, является ли текущий суффикс шаблоном, входящим в строку.

Введем понятие хорошей суффиксной ссылки. Так как чтобы проверить нахождение суффикса, являющегося шаблоном в строке, мы должны перебрать все суффиксы, начиная с наибольшего, заканчивая наименьшим в текущем положении в автомате. Мы знаем, что бессмысленно проверять суффиксы, которые не являются шаблонами, поэтому мы можем с помощью хороших суффиксных ссылок попадать только в те, которые совпадают с шаблонами. То есть хорошая суффиксная ссылка – номер узла, который был бы конечным, построив путь от корня до которого, можно получить ближайший суффикс, имеющийся в боре.

Двигаясь только по хорошим суффиксным ссылкам до корня найдем искомые шаблоны в строке для текущей позиции в строке.

Для второй программы использовался шаблон с джокером, который «совпадает» с любым символом. Также построим бор, в котором будет узел/узлы с джокером. При переходе по любому символу, можем заменить его на джокера. Если будет совершаться переход по символу джокера, будем запоминать символы строки, которые стояли на позиции джокера. Если мы будем искать суффикс, который оканчивается джокером, то вместо него будем совершать переходы от указанного узла суффиксной ссылки родителя к символу, который мы ранее сохраняли вместо джокера.

Оценка сложности по времени.

Алгоритм проходит по всей длине строки (длина строки N), он проверяет только те вершины, суффиксы которых хорошие (T – количество возможных вхождений всех шаблонов в строку). Если еще учесть вычисления автомата с суффиксными ссылками(L – размер бора, K – размер алфавита(константа), так

как у нас будет два массива размера K, мы должны это учитывать), то общая сложность будет O(N + T + L*K*2).

Оценка сложности по памяти.

Бор хранится как дерево с узлами. (K – длина всех шаблонов, N – размер алфавита(константа), так как у нас будет два массива размера K, мы должны это учитывать), общая сложность будет O(K*N*2).

Тестирование.

Тестирование первой программы.

Нумерация	Входные данные	Выходные данные
1	NTAG 3 TAGT TAG T	Хотите ввести данные из файла или с помощью клавиатуры?(1/2) 1 Начинаем построение бора Рассматриваем 1 подстроку: TAGT Рассматриваем T символ Узел с соответствующим очередной букве символом не найден. Добавим в бор новый. Рассматриваем A символ Узел с соответствующим очередной букве символом не найден. Добавим в бор новый. Рассматриваем G символ Узел с соответствующим очередной букве символом не найден. Добавим в бор новый. Рассматриваем T символ Узел с соответствующим очередной букве символом не найден. Добавим в бор новый.

		Рассматриваем 2 подстроку: TAG Рассматриваем T символ Узел с соответствующим очередной букве символом найден. Рассматриваем A символ Узел с соответствующим очередной букве символом найден. Рассматриваем G символ Узел с соответствующим очередной букве символом найден. Рассматриваем 3 подстроку: T Рассматриваем T символ Узел с соответствующим очередной букве символом найден. Рассматриваем строку с текущей позицией: "N"TAG Следующий узел с соответствующим символом не найден, перейдем в узел корня Рассматриваем строку с текущей позицией: N"T"AG Совершим переход к следующему узлу с соответствующим символом Следующий рассматриваемый суффикс: T Найден шаблон в тексте 2 3 ---------- Для хорошей суффиксной ссылки получим узел по суффиксной ссылке текущего узла Суффиксная ссылка указывает на корень Текущий суффикс: $ Хорошая суффиксная ссылка указывает на корень ---------- Рассматриваем строку с текущей позицией: NT"A"G Совершим переход к следующему узлу с соответствующим символом Следующий рассматриваемый суффикс: TA

		Для хорошей суффиксной ссылки получим узел по суффиксной ссылке текущего узла
		Совершим переход по суффиксной ссылке родителя
		Текущий суффикс:
		$
		Следующий узел с соответствующим символом не найден, перейдем в узел корня
		Хорошая суффиксная ссылка указывает на корень

		Рассматриваем строку с текущей позицией: NTA"G"
		Совершим переход к следующему узлу с соответствующим символом
		Следующий рассматриваемый суффикс:
		TAG
		Найден шаблон в тексте
		2 2

		Для хорошей суффиксной ссылки получим узел по суффиксной ссылке текущего узла
		Совершим переход по суффикс��ой ссылке родителя
		Текущий суффикс:
		$
		Следующий узел с соответствующим символом не найден, перейдем в узел корня
		Хорошая суффиксная ссылка указывает на корень

		Позиции шаблонов в тексте и номер образца:
		2 2
		2 3
		Количество вершин в автомате: 5
		Список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска:

		T TAG
2	ACCACC CACACC AAC 5 ACCAC CACA AACC ACAAC CCCAA	1 1 7 2 Количество вершин в автомате: 20 Список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска, пуст
3	ACGTNC TNCTNA 3 TNC NCTN NC	4 1 5 2 5 3 7 1 8 2 8 3 Количество вершин в автомате: 8 Список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска: TNC NC NCTN

Тестирование второй программы.

Нумерация	Входные данные	Выходные данные
1	ACTANCA A$$A$ $	Хотите ввести данные из файла или с помощью клавиатуры?(1/2) 1 Начинаем построение бора Рассматриваем A символ

		Узел с соответствующим очередной букве символом не найден. Добавим в бор новый. Рассматриваем $ символ Узел с соответствующим очередной букве символом не найден. Добавим в бор новый. Рассматриваем $ символ Узел с соответствующим очередной букве символом не найден. Добавим в бор новый. Рассматриваем A символ Узел с соответствующим очередной букве символом не найден. Добавим в бор новый. Рассматриваем $ символ Узел с соответствующим очередной букве символом не найден. Добавим в бор новый. Рассматриваем строку с текущей позицией: "A"CTANCA Совершим переход к следующему узлу с соответствующим символом Следующий рассматриваемый суффикс: A --- Для хорошей суффиксной ссылки получим узел по суффиксной ссылке текущего узла Суффиксная ссылка указывает на корень Текущий суффикс: . Хорошая суффиксная ссылка указывает на корень --- Рассматриваем строку с текущей позицией: A"C"TANCA Совершим переход к следующему узлу с джокером Следующий рассматриваемый суффикс: A$ ---

		Для хорошей суффиксной ссылки получим узел по суффиксной ссылке текущего узла Возьмем вместо джокера символ на той же позиции в тексте: Текущий суффикс: . Следующий узел с соответствующим символом не найден, перейдем в узел корня Текущий суффикс: . Хорошая суффиксная ссылка указывает на корень --- Рассматриваем строку с текущей позицией: AC"T"ANCA Совершим переход к следующему узлу с джокером Следующий рассматриваемый суффикс: A$$ --- Для хорошей суффиксной ссылки получим узел по суффиксной ссылке текущего узла Возьмем вместо джокера символ на той же позиции в тексте: Возьмем вместо джокера символ на той же позиции в тексте: Текущий суффикс: . Следующий узел с соответствующим символом не найден, перейдем в узел корня Текущий суффикс: . Текущий суффикс: . Хорошая суффиксная ссылка указывает на корень ---

		Рассматриваем строку с текущей позицией: ACT"A"NCA Совершим переход к следующему узлу с соответствующим символом Следующий рассматриваемый суффикс: A$$A --- Для хорошей суффиксной ссылки получим узел по суффиксной ссылке текущего узла Совершим переход по суффиксной ссылке родителя Возьмем вместо джокера символ на той же позиции в тексте: Возьмем вместо джокера символ на той же позиции в тексте: Текущий суффикс: . Текущий суффикс: . Текущий суффикс: . Хорошая суффиксная ссылка указывает на хорошую суффиксную ссылку узла --- --- --- Рассматриваем строку с текущей позицией: ACTA"N"CA Совершим переход к следующему узлу с джокером Следующий рассматриваемый суффикс: A$$A$ Найден шаблон в тексте 1 --- Для хорошей суффиксной ссылки получим узел по суффиксной ссылке текущего узла

		Возьмем вместо джокера символ на той же позиции в тексте: Текущий суффикс: A Совершим переход к следующему узлу с джокером Текущий суффикс: A$ Текущий суффикс: A$ Хорошая суффиксная ссылка указывает на хорошую суффиксную ссылку узла --- --- --- Рассматриваем строку с текущей позицией: ACTAN"C"A Следующий узел с соответствующим символом не найден, перейдем по суффиксной ссылке Возьмем вместо джокера символ на той же позиции в тексте: Следующий рассматриваемый суффикс: A$$A Текущий суффикс: A$ Совершим переход к следующему узлу с джокером Текущий суффикс: A$$ --- --- Рассматриваем строку с текущей позицией: ACTANC"A" Совершим переход к следующему узлу с соответствующим символом Следующий рассматриваемый суффикс: A$$A

		---------- ---------- Позиции шаблонов в тексте и номер образца: 1 Количество вершин в автомате: 6 Список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска, пуст
2	ACNTCNACG N% %	3 6 Количество вершин в автомате: 3 Список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска, пуст
3	TANGTANGTAN TAN% %	1 5 Количество вершин в автомате: 5 Список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска, пуст
4	NATNATT NAT% %	1 4 Количество вершин в автомате: 5 Список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска: NAT%

Выводы.

Была разработана программа, занимающаяся поиском шаблонов в строке и находящая все ее вхождения, и был изучен алгоритм Ахо-Корасика. Также была реализована программа, ищущая шаблон в тексте, в котором содержится джокер.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5_1.cpp

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

//#define DEBUG

std::map<char, int> alphabet{
        { '$', 0 },
        { 'A', 1 },
        { 'C', 2 },
        { 'G', 3 },
        { 'T', 4 },
        { 'N', 5 } };

#define countOfAlph 6

class Node {
public:
    int name;
    int nextVer[countOfAlph];
    int numOfPattern;
    int suffixRef;
    int autoMove[countOfAlph];
    int suffixFRef;
    int parent;//узел родителя
    bool isEnd;
    Node(int parent, int c) {
        std::fill_n(nextVer, countOfAlph, -1);
        std::fill_n(autoMove, countOfAlph, -1);
        suffixRef = -1;
        suffixFRef = -1;
        this->parent = parent;
        name = c;
        isEnd = false;
    }
};

class  Aho_Corasick {
private:
    std::vector<Node> bohr;
    std::vector<std::string> P;
```

```
    std::string T;
    std::vector<std::string> crossPatterns;
    std::vector<std::pair<int, int>> otvet;
    bool isF;
    int crossCount, crossP;

public:
    Aho_Corasick() {
        bohr.emplace_back(0, alphabet['$']);
        isF = true;
        crossCount = 0, crossP = -1;
    }

    void readData(std::istream& fin) {
        int n;
        fin >> T >> n;
        for (int i = 0; i < n; i++) {
            std::string cur_P;
            fin >> cur_P;
            P.push_back(cur_P);
        }
    }

    void addPatternToBohr() {
#ifdef DEBUG
        std::cout << "Начинаем построение бора\n";
#endif
        for (int str = 0; str < P.size();str++) {
            int n = 0;
#ifdef DEBUG
            std::cout << "Рассматриваем " << str + 1 << " подстроку: " << P[str]
<< "\n";
#endif
            for (auto &i : P[str]) {
#ifdef DEBUG
                std::cout << "Рассматриваем " << i << " символ\n";
#endif
                int sym;
                if (alphabet.find(i) != alphabet.end()) {
                    sym = alphabet[i];
                } else {
                    //error
                    std::cout << "error\n";
                    sym = 0;
                }
                if (bohr[n].nextVer[sym] == -1) {
                    bohr.emplace_back(n, sym);
                    bohr[n].nextVer[sym] = bohr.size() - 1;
#ifdef DEBUG
```

```
                    std::cout << "Узел с соответствующим очередной букве символом
не найден. Добавим в бор новый.\n";
#endif
                } else {
#ifdef DEBUG
                    std::cout << "Узел с соответствующим очередной букве символом
найден.\n";
#endif
                }
                n = bohr[n].nextVer[sym];
            }
            bohr[n].isEnd = true;
            bohr[n].numOfPattern = str;
        }
    }

    int getSuffixRef(int ver){
        if (bohr[ver].suffixRef == -1) {
            if (ver == 0 || bohr[ver].parent == 0) {
#ifdef DEBUG
                std::cout << "Суффиксная ссылка указывает на корень\n";
#endif
                bohr[ver].suffixRef = 0;
            }
            else {
#ifdef DEBUG
                std::cout << "Совершим переход по суффиксной ссылке родителя\n";
#endif
                bohr[ver].suffixRef = getAutoMove(getSuffixRef(bohr[ver].parent),
bohr[ver].name);
                return bohr[ver].suffixRef;
            }
        }
#ifdef DEBUG
        if (isF)
            printSuffix(ver);
        else printSuffix(bohr[ver].suffixRef);
#endif
        return bohr[ver].suffixRef;
    }

    int getAutoMove(int ver, int sym) {
        if (bohr[ver].autoMove[sym] == -1) {
            if (bohr[ver].nextVer[sym] != -1) {
#ifdef DEBUG
                std::cout << "Совершим переход к следующему узлу с
соответствующим символом\n";
                printSuffix(bohr[ver].nextVer[sym]);
#endif
```

```
                bohr[ver].autoMove[sym] = bohr[ver].nextVer[sym];
            } else if (ver == 0) {
#ifdef DEBUG
                std::cout << "Следующий узел с соответствующим символом не
найден, перейдем в узел корня\n";
#endif
                bohr[ver].autoMove[sym] = 0;
            } else {
#ifdef DEBUG
                std::cout << "Следующий узел с соответствующим символом не
найден, перейдем по суффиксной ссылке\n";
#endif
                bohr[ver].autoMove[sym] = getAutoMove(getSuffixRef(ver), sym);
            }
        }
        return bohr[ver].autoMove[sym];
    }

    int getSuffixFRef(int ver) {
#ifdef DEBUG
        std::cout << "-------------------------------------------------\n";
#endif
        if (bohr[ver].suffixFRef == -1) {
#ifdef DEBUG
            std::cout << "Для хорошей суффиксной ссылки получим узел по
суффиксной ссылке текущего узла\n";
#endif
            int u = getSuffixRef(ver);
            if (u == 0) {
#ifdef DEBUG
                std::cout << "Хорошая суффиксная ссылка указывает на корень\n";
#endif
                bohr[ver].suffixFRef = 0;
            }
            else {
                if (bohr[u].isEnd) {
#ifdef DEBUG
                    std::cout << "Хорошая суффиксная ссылка указывает на конец
шаблона\n";
#endif
                    bohr[ver].suffixFRef = u;
                } else {
#ifdef DEBUG
                    std::cout << "Хорошая суффиксная ссылка указывает на хорошую
суффиксную ссылку узла\n";
#endif
                    bohr[ver].suffixFRef = getSuffixFRef(u);
                }
            }
```

```
        }
#ifdef DEBUG
        std::cout << "-----------------------------------------------\n";
#endif
        return bohr[ver].suffixFRef;
    }

    void checkResearching(int ver, int i) {
        for (int y = ver; y != 0; y = getSuffixFRef(y)) {
            if (bohr[y].isEnd) {
#ifdef DEBUG
                std::cout << "Найден шаблон в тексте\n";
                std::cout << i - P[bohr[y].numOfPattern].length() + 1 << " " <<
bohr[y].numOfPattern + 1 << "\n";
#endif
                if (crossP != -1 &&  i - crossCount < P[bohr[y].numOfPat-
tern].length() + P[crossP].length()) {
                    if (std::find(crossPatterns.begin(), crossPatterns.end(),
P[crossP]) == crossPatterns.end())
                        crossPatterns.push_back(P[crossP]);
                    if (std::find(crossPatterns.begin(), crossPatterns.end(),
P[bohr[y].numOfPattern]) == crossPatterns.end())
                        crossPatterns.push_back(P[bohr[y].numOfPattern]);
                }
                crossP = bohr[y].numOfPattern;
                crossCount = i - P[bohr[y].numOfPattern].length();
                otvet.push_back(std::pair<int,int>(i - P[bohr[y].numOfPat-
tern].length() + 1, bohr[y].numOfPattern + 1));
            }
        }
    }

    void printSuffix(int suf) {
        if (isF) {
            std::cout << "Следующий рассматриваемый суффикс:\n";
            isF = false;
        }
        else
            std::cout << "Текущий суффикс:\n";
        std::string suffix;
        int k = suf;
        while (bohr[k].parent != 0) {
            suffix += (char) (bohr[k].name + 48);
            k = bohr[k].parent;
        }
        suffix += (char) (bohr[k].name + 48);
        for (auto &i : alphabet) {
            for (std::string::size_type n = 0; (n = suffix.find((char) (i.second
+ 48), n)) != std::string::npos; ++n) {
```

```
                suffix.replace(n, 1, 1, i.first);
            }
        }
        for (int i = suffix.length() - 1; i > -1; i--)
            std::cout << suffix[i];
        std::cout << "\n";
    }

    void doAlgorithm() {
        int u = 0;
        for (int i = 0; i < T.length(); i++) {
#ifdef DEBUG
            std::cout << "Рассматриваем строку с текущей позицией: ";
            for (int j = 0; j < T.length(); j++)
                if (j == i)
                    std::cout << "\"" << T[j] << "\"";
                else
                    std::cout << T[j];
            std::cout << "\n";
            isF = true;
#endif
            u = getAutoMove(u, alphabet[T[i]]);
            checkResearching(u, i + 1);
        }
#ifdef DEBUG
        std::cout << "Позиции шаблонов в тексте и номер образца:\n";
#endif
        sort (otvet.begin(), otvet.end());
        for (auto& i : otvet)
            std::cout << i.first << " " << i.second << "\n";
        std::cout << "Количество вершин в автомате: " << bohr.size() << "\n";
        std::cout << "Список найденных образцов, имеющих пересечения с другими
найденными образцами в строке поиска";
        if (crossPatterns.empty())
            std::cout << ", пуст\n";
        else {
            std::cout << ":\n";
            for (auto &i : crossPatterns) {
                std::cout << i << " ";
            }
        }
    }

};

void startProgram() {
    Aho_Corasick* algorithm = new Aho_Corasick();
    int answer = 2;
#ifdef DEBUG
```

```
    std::cout << "Хотите ввести данные из файла или с помощью
клавиатуры?(1/2)\n";
    std::cin >> answer;
#endif
    if (answer == 1) {
        std::ifstream fin("test1_2.txt");
        algorithm->readData(fin);
        fin.close();
    } else {
#ifdef DEBUG
        std::cout << "Введите текст, количество шаблонов и шаблоны, используя
символы алфавита {A,C, G, T, N}:\n";
#endif
        algorithm->readData(std::cin);
    }
    algorithm->addPatternToBohr();
    algorithm->doAlgorithm();
}

int main() {
    startProgram();
    return 0;
}
```

Название файла: lab5_2.cpp

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

//#define DEBUG

std::map<char, int> alphabet {
        { '.', 0 },
        { 'A', 1 },
        { 'C', 2 },
        { 'G', 3 },
        { 'T', 4 },
        { 'N', 5 } };

#define countOfAlph 7

class Node {
public:
    int name;
    int nextVer[countOfAlph];
    int suffixRef;
    int autoMove[countOfAlph];
    int suffixFRef;
    int parent;//узел родителя
    bool isEnd;
    Node(int parent, int c) {
        std::fill_n(nextVer, countOfAlph, -1);
        std::fill_n(autoMove, countOfAlph, -1);
        suffixRef = -1;
        suffixFRef = -1;
        this->parent = parent;
        name = c;
        isEnd = false;
    }
};

class  Aho_Corasick {
private:
    std::vector<Node> bohr;
    std::string P;
    std::string T;
    bool isCross;
    char joker;
    bool isF;
    std::vector<int> key;
```

```
    int num;
    std::vector<int> otvet;
    int crossCount;

public:
    Aho_Corasick() {
        bohr.emplace_back(0, alphabet['.']);
        isF = true;
        num = 0;
        isCross = false;
        crossCount = -1;
    }

    void readData(std::istream &fin) {
        fin >> T >> P >> joker;
    }

    void addPatternToBohr() {
#ifdef DEBUG
        std::cout << "Начинаем построение бора\n";
#endif
        int n = 0;
        for (auto &i : P) {
#ifdef DEBUG
            std::cout << "Рассматриваем " << i << " символ\n";
#endif
            int sym;
            if (alphabet.find(i) != alphabet.end()) {
                sym = alphabet[i];
            } else {
                sym = 6;
            }
            if (bohr[n].nextVer[sym] == -1) {
                bohr.emplace_back(n, sym);
                bohr[n].nextVer[sym] = bohr.size() - 1;
#ifdef DEBUG
                std::cout << "Узел с соответствующим очередной букве символом не
найден. Добавим в бор новый.\n";
#endif
            } else {
#ifdef DEBUG
                std::cout << "Узел с соответствующим очередной букве символом
найден.\n";
#endif
            }
            n = bohr[n].nextVer[sym];
        }
        bohr[n].isEnd = true;
    }
```

```
    int getSuffixRef(int ver) {
        if (bohr[ver].suffixRef == -1 || bohr[ver].name == 6) {
            if (ver == 0 || bohr[ver].parent == 0) {
#ifdef DEBUG
                std::cout << "Суффиксная ссылка указывает на корень\n";
#endif
                bohr[ver].suffixRef = 0;
            } else if (bohr[ver].name == 6) {
#ifdef DEBUG
                std::cout << "Возьмем вместо джокера символ на той же позиции в
тексте:\n";
#endif
                bohr[ver].suffixRef = getAutoMove(getSuffixRef(bohr[ver].parent),
key[num--]);
            }  else {
#ifdef DEBUG
                std::cout << "Совершим переход по суффиксной ссылке родителя\n";
#endif
                bohr[ver].suffixRef = getAutoMove(getSuffixRef(bohr[ver].parent),
bohr[ver].name);
                return bohr[ver].suffixRef;
            }
        }
#ifdef DEBUG
        if (isF)
            printSuffix(ver);
        else printSuffix(bohr[ver].suffixRef);
#endif
        return bohr[ver].suffixRef;
    }

    int getAutoMove(int ver, int sym) {
        if (bohr[ver].autoMove[sym] == -1 || bohr[ver].name == 6) {
            if (bohr[ver].nextVer[sym] != -1) {
#ifdef DEBUG
                std::cout << "Совершим переход к следующему узлу с
соответствующим символом\n";
                printSuffix(bohr[ver].nextVer[sym]);
#endif
                bohr[ver].autoMove[sym] = bohr[ver].nextVer[sym];
            } else if (bohr[ver].nextVer[6] != -1) {
                bohr[ver].autoMove[sym] = bohr[ver].nextVer[6];
#ifdef DEBUG
                std::cout << "Совершим переход к следующему узлу с джокером\n";
                printSuffix(bohr[ver].autoMove[sym]);
#endif
            } else if (ver == 0) {
#ifdef DEBUG
```

```
                std::cout << "Следующий узел с соответствующим символом не
найден, перейдем в узел корня\n";
#endif
                bohr[ver].autoMove[sym] = 0;
            } else {
#ifdef DEBUG
                std::cout << "Следующий узел с соответствующим символом не
найден, перейдем по суффиксной ссылке\n";
#endif
                bohr[ver].autoMove[sym] = getAutoMove(getSuffixRef(ver), sym);
            }
        }
        return bohr[ver].autoMove[sym];
    }

    int getSuffixFRef(int ver) {
#ifdef DEBUG
        std::cout << "-------------------------------------------------\n";
#endif
        if (bohr[ver].suffixFRef == -1) {
#ifdef DEBUG
            std::cout << "Для хорошей суффиксной ссылки получим узел по
суффиксной ссылке текущего узла\n";
#endif
            int u = getSuffixRef(ver);
            if (u == 0) {
#ifdef DEBUG
                std::cout << "Хорошая суффиксная ссылка указывает на корень\n";
#endif
                bohr[ver].suffixFRef = 0;
            } else {
                if (bohr[u].isEnd) {
#ifdef DEBUG
                    std::cout << "Хорошая суффиксная ссылка указывает на конец
шаблона\n";
#endif
                    bohr[ver].suffixFRef = u;
                } else {
#ifdef DEBUG
                    std::cout << "Хорошая суффиксная ссылка указывает на хорошую
суффиксную ссылку узла\n";
#endif
                    bohr[ver].suffixFRef = getSuffixFRef(u);
                }
            }
        }
#ifdef DEBUG
        std::cout << "-------------------------------------------------\n";
#endif
```

```
        return bohr[ver].suffixFRef;
    }

    void checkResearching(int& ver, int i) {
        for (int y = ver; y != 0; y = getSuffixFRef(y)) {
            if (bohr[y].isEnd) {
#ifdef DEBUG
                std::cout << "Найден шаблон в тексте\n";
#endif
                if (crossCount != -1 &&  i - crossCount < P.length() +
P.length()) {
                    isCross = true;
                }
                crossCount = i - P.length();
                std::cout << i - P.length() + 1 << "\n";
                otvet.push_back(i - P.length() + 1);
                }
            if (bohr[y].name == 6) {
                key.push_back(alphabet[T[i -1]]);
            }
            num = key.size() - 1;
        }
    }


    void printSuffix(int suf) {
        if (isF) {
            std::cout << "Следующий рассматриваемый суффикс:\n";
            isF = false;
        } else
            std::cout << "Текущий суффикс:\n";
        std::string suffix;
        int k = suf;
        while (bohr[k].parent != 0) {
            suffix += (char) (bohr[k].name + 48);
            k = bohr[k].parent;
        }
        suffix += (char) (bohr[k].name + 48);
        for (auto &i : alphabet) {
            for (std::string::size_type n = 0; (n = suffix.find((char) (i.second
+ 48), n)) != std::string::npos; ++n) {
                suffix.replace(n, 1, 1, i.first);
            }
        }
        for (std::string::size_type n = 0; (n = suffix.find((char) (48 + 6),
n)) != std::string::npos; ++n) {
            suffix.replace(n, 1, 1, joker);
        }
        for (int i = suffix.length() - 1; i > -1; i--)
```

```
            std::cout << suffix[i];
        std::cout << "\n";
    }

    void doAlgorithm() {
        int u = 0;
        for (int i = 0; i < T.length(); i++) {
#ifdef DEBUG
            std::cout << "Рассматриваем строку с текущей позицией: ";
            for (int j = 0; j < T.length(); j++)
                if (j == i)
                    std::cout << "\"" << T[j] << "\"";
                else
                    std::cout << T[j];
            std::cout << "\n";
            isF = true;
#endif
            num = key.size() - 1;
            u = getAutoMove(u, alphabet[T[i]]);
            num = key.size() - 1;
            checkResearching(u, i + 1);
        }
#ifdef DEBUG
        std::cout << "Позиции шаблонов в тексте и номер образца:\n";
        sort (otvet.begin(), otvet.end());
        for (auto& i : otvet)
            std::cout << i << "\n";
#endif
//#ifdef DEBUG
        std::cout << "Количество вершин в автомате: " << bohr.size() << "\n";
        std::cout << "Список найденных образцов, имеющих пересечения с другими
найденными образцами в строке поиска";
        if (!isCross)
            std::cout << ", пуст\n";
        else {
            std::cout << ":\n";
            std::cout << P << "\n";
        }
//#endif
    }

};

void startProgram() {
    Aho_Corasick* algorithm = new Aho_Corasick();
    int answer = 2;
#ifdef DEBUG
    std::cout << "Хотите ввести данные из файла или с помощью
клавиатуры?(1/2)\n";
```

```
    std::cin >> answer;
#endif
    if (answer == 1) {
        std::ifstream fin("test2_1.txt");
        algorithm->readData(fin);
        fin.close();
    } else {
#ifdef DEBUG
        std::cout << "Введите текст, используя символы алфавита {A,C, G, T, N},
шаблон и джокер:\n";
#endif
        algorithm->readData(std::cin);
    }
    algorithm->addPatternToBohr();
    algorithm->doAlgorithm();
}

int main() {
    startProgram();
    return 0;
}
```