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

Студентка гр. 8303 _____ Потураева М.Ю.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

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

Постановка задачи.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

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

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

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

Выход:

Все вхождения образцов из P в T.

Каждое вхождение образца в текст представить в виде двух чисел - i p

Где i - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p (нумерация образцов начинается с 1).

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

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с джокером.

В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу P необходимо найти все вхождения P в текст T.

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

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

длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы.

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

Вход:

Текст (T, $1 \leq |T| \leq 100000$)

Шаблон (P, $1 \leq |P| \leq 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер). Номера должны выводиться в порядке возрастания.

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

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

Описание алгоритма Ахо-Корасик.

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

1. Создание бора.

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

Строить бор будем последовательным добавление исходных строк. Для этого добавляется начальная вершина, она становится текущей (корень) и для каждой буквы шаблона:

• Если переход по букве существует, он совершается.

•Иначе в боре создается новая вершина, добавляется и совершается переход в нее

2. Поиск шаблонов в тексте.

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

Правило перехода в боре:

Пусть мы находимся в состоянии p, которому соответствует строка t, и хотим выполнить переход по символу c.

• Если в боре уже есть переход по букве c, этот переход совершается и мы попадаем в вершину, соответствующую строке tc.

• Если же такого ребра нет, то мы должны найти состояние, соответствующее наидлиннейшему собственному суффиксу строки t (наидлиннейшему из имеющихся в боре), и попытаться выполнить переход по букве c из него. То есть задача сводится к поиску суффиксных ссылок для вершин.

Если мы хотим узнать суффиксную ссылку для некоторой вершины v, то мы можем перейти в предка p текущей вершины (пусть c — буква, по которой из p есть переход в v), затем перейти по его суффиксной ссылке, а затем из неё выполнить переход в автомате по букве c.

Сам поиск шаблонов в строке:

1. Текущая вершина – корень бора.
2. Пока есть символы в строке:

   • Совершается переход по следующей букве строки (по правилам, указанным выше).

• Из текущей вершины по суффикс ссылкам проходим до корня бора, проверяя встречу вхождений (если встречается лист, вхождение найдено).

Описание алгоритма работы программы с «джокером».

Для того, чтобы найти все вхождение в текст заданного шаблона необходимо обнаружить в тексте все подстроки шаблона, разбитых джокером. Создадим для этого массив строк Q, в котором хранятся все подстроки, разбитые джокером и массив l, в котором хранятся индекс вхождение этих подстрок в шаблон.

Создадим массив C, в котором на позиции i будет хранится количество встретившихся безмасочных подстрок шаблона, которые начинаются на позиции i. Тогда появление Q[i] в тексте на позиции j будет означать возможное появление шаблона на позиции j – l[i]+ 1.

Построим бор по массиву строк Q. Теперь с помощью алгоритма Ахо-Корасик находим все безмасочные подстроки, когда алгоритм находит подстроку Q[i] он увеличивает на 1 значение C[j – l[i] + 1]

Теперь каждое i, для которого C[i] равно количеству подстрок разбитых джокером является стартовой позицией появления шаблона в тексте.

Чтобы реализовать индивидуализацию во время вывода, джокер сравнивается с недопустимым символом и при совпадении вхождение шаблона не добавляется .

Описание структур и основных функций.

```
class Vertex {
public: std::map sons;
std::map go;
std::pair prev;
int suffLink;
bool isLeaf;
int number;
int deep;
```

```
void printInfo(int i);
};
```

Класс представления вершины бора и автомата.

sons – контейнер прямых переходов по символу char в вершину с номером int.

go – массив переходов (запоминаем переходы в ленивой рекурсии), используемый для вычисления суффиксных ссылок.

prev – пара, хранящая индекс предыдущей вершины и символ ребра, по которому был совершен переход в вершину.

suffLink – индекс элемента, на который указывает суффиксная ссылка.

isLeaf – true, если веришна терминальная (является послденим символом в строке), false в обратном случае.

number – номер строки по порядку в вводе (для красивого вывода на степике).

deep – глубина листа, равная длине строки-образца.

void printInfo(int i) – метод для печати информации о вершине с индексом i, принимает номер вышины, ничего не возвращает, выводит основную инфорацию вершины.

```
void createBohr(int size)
```

Функция предназначена для заполнения бора строками-образцами. Принимает количетво строк-образцов size, ничего не возвращает. Внутри функции происходит считывание строк-образцов и последовательное добавление элементов строки в бор.

```
int getSuffLink(int ind)
```

Функция поиска суффиксной ссылки для вершины. Принимает ind – номер вершины в боре, для которой ищет ссылку, возвращает номер вершины, на которую указывает суффиксная ссылка.

```
int getLink(int ind, char c)
```

Функция перехода из вершины по символу. Принимает ind – номер вершины в массиве вершин, из которой происходит переход, c – символ, по

которому происходит переход. Возвращает номер вершины, в которую будет совершен переход.

```
void ahoKorasik(std::string& text)
```

Функция поиска вхождений строк-образцов. Принимает по ссылке строку text (в ней происходит поиск). Осуществляет алгоритм Ахо-Корасик множественного поиска индексов вхождений строк-образцов в строку-текст, описанный выше, используя вышеописанные функции.

```
int compare(std::pair a, std::pair b)
```

Функция сравнения двух пар. Существует для корректной работы std::sort.

Сложность алгоритма по времени.

Сложность алгоритма Ахо-Корасик по времени можно оценить, как

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

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

Сложность алгоритма по памяти.

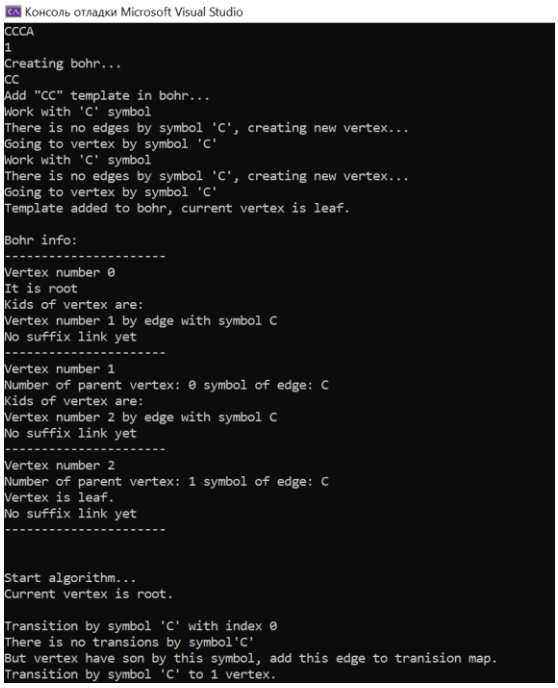
Сложность алгоритма Ахо-Корасик по памяти можно оценить, как

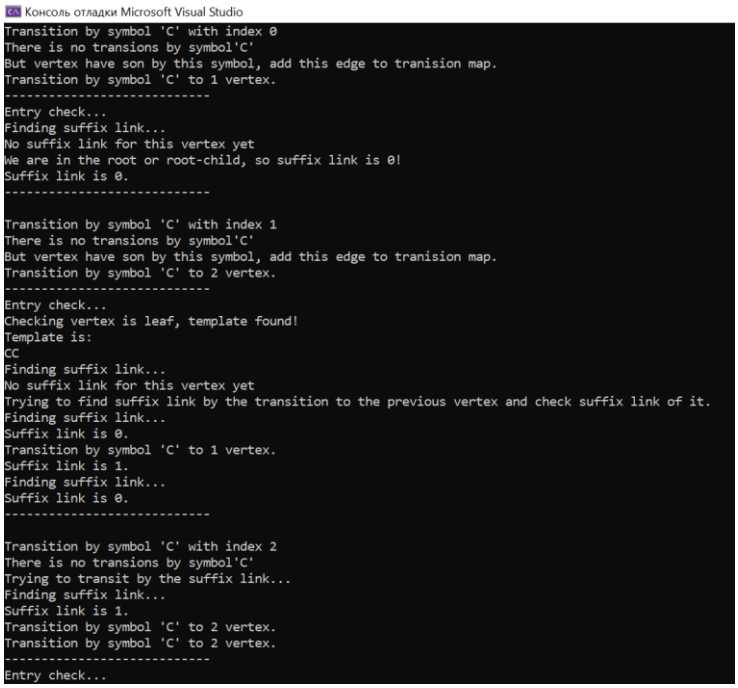
$$O(M * A).$$

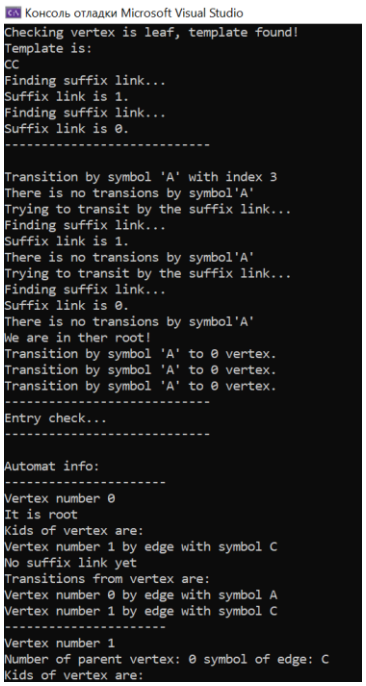
Так как на каждую вершину приходится $O(A)$ памяти.

Тестирование 1-й программы.

Пример вывода результата для 1-го теста.







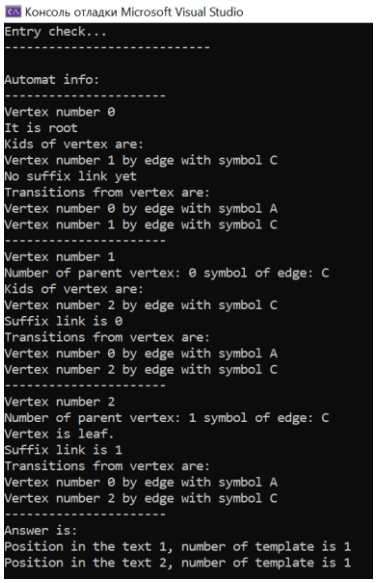


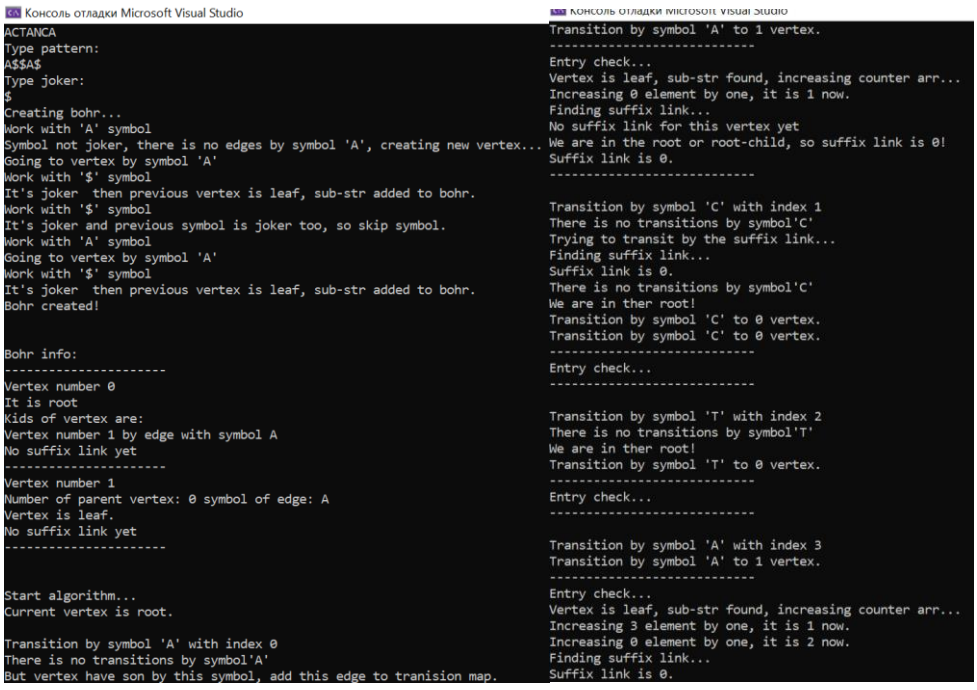
Таблица 1 – тестирование программы

№	Input	Output
1	CCCA 1 CC	1 1 2 1
2	NANNNANNANAN 2 NAN A	1 1 2 2 5 1 6 2 8 1 9 2 10 1 11 2
3	ACACGTNNNNAGT 3 G AG NNN	5 1 7 3 8 3 11 2 12 1

4	CCCA 2 CC CA	1 1 2 1 2 2
5	ACAACA 2 AT CG	

Тестирование 2-й программы.

Пример вывода результата для 1-го теста.



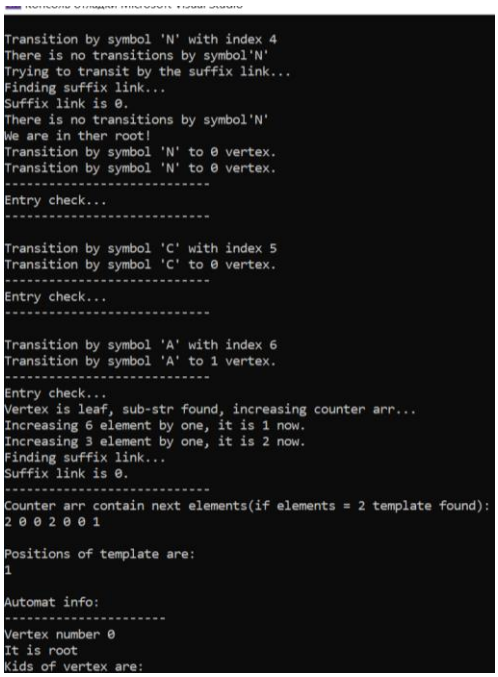


Таблица 2 – тестирование программы

№	Input	Output
1	ACTANCA A$$A$ $	1
2	ACACAGGAGA A%A %	1 3 8
3	ACG A&& &	1
4	ACTANCA A$$A $	1 4
5	ATGTNGT AC!GN !	

Тестирование 3-й программы.

Пример вывода результата для 1-го теста.

```
ACG
Type pattern:
A&&
Type joker:
&
Type non joker:
C
Creating bohr...
Work with 'A' symbol
Symbol not joker, there is no edges by symbol 'A', creating new vertex...
Going to vertex by symbol 'A'
Work with '&' symbol
It's joker  then previous vertex is leaf, sub-str added to bohr, push position of joker to non-joker vertex.
Work with '&' symbol
It's joker and previous symbol is joker too, so skip symbol, push position of joker to non-joker vertex.
Bohr created!


Bohr info:
----------------------
Vertex number 0
It is root
Kids of vertex are:
Vertex number 1 by edge with symbol A
No suffix link yet
----------------------
Vertex number 1
Number of parent vertex: 0 symbol of edge: A
Vertex is leaf.
No suffix link yet
----------------------


Start algorithm...
Current vertex is root.

Transition by symbol 'A' with index 0
There is no transitions by symbol'A'
But vertex have son by this symbol, add this edge to tranision map.
Transition by symbol 'A' to 1 vertex.
---------------------------
```

```
Консоль отладки Microsoft Visual Studio
Entry check...
Vertex is leaf, sub-str found, Increasing elements in counter arr...
        Increasing 0 element by one, it is 1 now.
Finding suffix link...
No suffix link for this vertex yet
We are in the root or root-child, so suffix link is 0!
Suffix link is 0.
---------------------------

Transition by symbol 'C' with index 1
There is no transitions by symbol'C'
Trying to transit by the suffix link...
Finding suffix link...
Suffix link is 0.
There is no transitions by symbol'C'
We are in ther root!
Transition by symbol 'C' to 0 vertex.
Transition by symbol 'C' to 0 vertex.
---------------------------
Entry check...
Found non jocker, decreasing elements in counter arr...
        Decreasing 0 element by one, it is 2 now.
---------------------------

Transition by symbol 'G' with index 2
There is no transitions by symbol'G'
We are in ther root!
Transition by symbol 'G' to 0 vertex.
---------------------------
Entry check...
---------------------------
Counter arr contain next elements(if elements = 1 template found):
0 0 0

Positions of template are:

Automat info:
----------------------
Vertex number 0
It is root
Kids of vertex are:
```

Таблица 3 – тестирование программы

№	Input	Output
1	ACG A&& & C	
2	NANNNANNANAN NXN X N	1 5 8 10
3	ACACAGGAGA A%A % G	1 3
4	ACTAGCANCAAAAAANA AXXA X N	1 4 10 11 12
5	NANANANANAN NANAXAN X A	1 3 5
6	ACCCAAAXCCA ACCCA C X	1
7	NAGABNAGNNABNNNB	

	NCG C B	1 6
8	ACNAGCANCAAAAAANA AXXA X T	1 4 7 10 11 12 14

Выводы.

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

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ 1

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


class Vertex {
public:
    std::map<char, int> sons; //map of sons
    std::map<char, int> go; //map of transition
    std::pair<char, int> prev; //information about parent vertex
    int suffLink; //suffix link
    bool isLeaf; //obvious
    int number; //number of str
    int deep; //deep in bohr

    Vertex() {
        prev.second = -1;
        prev.first = '}';
        isLeaf = false;
        suffLink = -1;
        deep = 0;
        number = 0;
    }
    Vertex(int prevInd, char prevChar) {
        isLeaf = false;
        suffLink = -1;
        prev.first = prevChar;
        prev.second = prevInd;
        deep = 0;
        number = 0;
    }
    void printInfo(int i) {
        std::cout << "Vertex number " << i << "\n";
        if (prev.second != -1) {
            std::cout << "Number of parent vertex: " << prev.second << "
symbol of edge: " << prev.first << "\n";
        }
        else {
            std::cout << "It is root\n";
        }
        if (isLeaf) {
            std::cout << "Vertex is leaf.\n";
        }
        else {
            std::cout << "Kids of vertex are:\n";
```

```
            for (auto it = sons.begin(); it != sons.end(); it++) {
                std::cout << "Vertex number " << it->second << " by edge
with symbol " << it->first << "\n";
            }
        }
        if (suffLink != -1) {
            std::cout << "Suffix link is " << suffLink << "\n";
        }
        else {
            std::cout << "No suffix link yet\n";
        }
        if (!go.empty()) {
            std::cout << "Transitions from vertex are:\n";
            for (auto it = go.begin(); it != go.end(); it++) {
                std::cout << "Vertex number " << it->second << " by edge
with symbol " << it->first << "\n";
            }
        }
    }
};


std::vector<std::pair<int, int>> answer; //first int - number, 2 -
position
std::vector<Vertex> bohr;
std::vector<std::string> forOut;
void createBohr(int size);
int getSuffLink(int ind);
int getLink(int ind, char c);
void ahoKorasik(std::string& text);
int compare(std::pair<int, int> a, std::pair<int, int>  b);


int main() {
    std::string text;
    int size = 0;
    std::cin >> text;
    std::cin >> size;
    createBohr(size);
    std::cout << "\nBohr info:\n----------------------\n";
    for (unsigned i = 0; i < bohr.size(); i++) {
        bohr[i].printInfo(i);
        std::cout << "----------------------\n";
    }

    std::cout << "\n\n";
    ahoKorasik(text);
    std::cout << "\nAutomat info:\n----------------------\n";
    for (unsigned i = 0; i < bohr.size(); i++) {
        bohr[i].printInfo(i);
        std::cout << "----------------------\n";
```

```
    }
    sort(answer.begin(), answer.end(), compare);
    std::cout << "Answer is:\n";
    for (unsigned i = 0; i < answer.size(); i++) {
        std::cout << "Position in the text " << answer[i].second << ",
number of template is " << answer[i].first + 1 << "\n";
    }

}


void createBohr(int size) {
    std::cout << "Creating bohr...\n";
    Vertex root;
    bohr.push_back(root);
    for (int i = 0; i < size; i++) { //for each template
        std::string buff;
        std::cin >> buff;
        forOut.push_back(buff);
        int curr = 0;
        std::cout << "Add \"" << buff << "\" template in bohr...\n";
        // system("pause");
        for (unsigned int j = 0; j < buff.length(); j++) { //for each
symbol in template
            std::cout << "Work with \'" << buff[j] << "\' symbol\n";
            if (bohr[curr].sons.find(buff[j]) == bohr[curr].sons.end()) {
//if can't go
                std::cout << "There is no edges by symbol \'" << buff[j]
<< "\', creating new vertex...\n";
                Vertex vert(curr, buff[j]); //create new vertex
                bohr.push_back(vert);
                bohr[curr].sons[buff[j]] = bohr.size() - 1; //add
information about child vertex
            }
            curr = bohr[curr].sons[buff[j]]; // move to vertex
            std::cout << "Going to vertex by symbol \'" << buff[j] <<
"\'\n";
        }
        std::cout << "Template added to bohr, current vertex is leaf.\n";
        bohr[curr].isLeaf = true;
        bohr[curr].number = i;
        bohr[curr].deep = buff.length();
    }
}


int getSuffLink(int ind) {
    std::cout << "Finding suffix link...\n";
    if (bohr[ind].suffLink == -1) {
        std::cout << "No suffix link for this vertex yet \n";
        if (ind == 0 || bohr[ind].prev.second == 0) {
```

```
            std::cout << "We are in the root or root-child, so suffix
link is 0!\n";
            bohr[ind].suffLink = 0;
        }
        else {
            std::cout << "Trying to find suffix link by the transition to
the previous vertex and check suffix link of it.\n";
            bohr[ind].suffLink =
getLink(getSuffLink(bohr[ind].prev.second), bohr[ind].prev.first);
        }
    }
    std::cout << "Suffix link is " << bohr[ind].suffLink << ".\n";
    return bohr[ind].suffLink;
}


int getLink(int ind, char c) {
    if (bohr[ind].go.find(c) == bohr[ind].go.end()) {
        std::cout << "There is no transions by symbol\'" << c << "\'\n";
        if (bohr[ind].sons.find(c) != bohr[ind].sons.end()) {
            std::cout << "But vertex have son by this symbol, add this
edge to tranision map.\n";
            bohr[ind].go[c] = bohr[ind].sons[c];
        }
        else {
            if (ind == 0) {
                std::cout << "We are in ther root!\n";
                bohr[ind].go[c] = 0;
            }
            else {
                std::cout << "Trying to transit by the suffix link...\n";
                bohr[ind].go[c] = getLink(getSuffLink(ind), c);
            }
        }
    }
    std::cout << "Transition by symbol \'" << c << "\' to " <<
bohr[ind].go[c] << " vertex.\n";
    return bohr[ind].go[c];
}


void ahoKorasik(std::string& text) {
    int curr = 0;
    std::cout << "Start algorithm...\nCurrent vertex is root.\n";
    for (unsigned int i = 0; i < text.length(); i++) { //for each symbol
of text
        std::cout << "\nTransition by symbol \'" << text[i] << "\' with
index " << i << "\n";
        curr = getLink(curr, text[i]); //make transition
        std::cout << "----------------------------\nEntry check...\n";
        for (int j = curr; j != 0; j = getSuffLink(j)) { //entry check
```

```
            if (bohr[j].isLeaf) {
                std::cout << "Checking vertex is leaf, template
found!\n";
                std::pair<int, int> res(bohr[j].number, i + 2 -
bohr[j].deep);
                answer.push_back(res);
                std::cout << "Template is: \n";
                std::cout << forOut[res.first] << "\n";
            }
        }
        std::cout << "----------------------------\n";
    }
}


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

ПРИЛОЖЕНИЕ Б

КОД ПРОГРАММЫ 2

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



class Vertex {
public:
    std::map<char, int> sons; //для бора
    std::map<char, int> go; //для автомата
    std::pair<char, int> prev; //предок
    int suffLink; //суф ссылка
    bool isLeaf; //конец слова
    int deep; //глубина
    std::vector<int> pos; //индекс

    Vertex() {//конструктор
        prev.second = -1;
        prev.first = '}';
        isLeaf = false;
        suffLink = -1;
```

```
        deep = 0;
    }
    Vertex(int prevInd, char prevChar) {//конструктор с вершиной
        isLeaf = false;
        suffLink = -1;
        prev.first = prevChar;
        prev.second = prevInd;
        deep = 0;
    }
    void printInfo(int i) {//вывод информации
        std::cout << "Vertex number " << i << "\n";
        if (prev.second != -1) {
            std::cout << "Number of parent vertex: " << prev.second << "
symbol of edge: " << prev.first << "\n";
        }
        else {
            std::cout << "It is root\n";
        }
        if (isLeaf) {
            std::cout << "Vertex is leaf.\n";
        }
        else {
            std::cout << "Kids of vertex are:\n";
            for (auto it = sons.begin(); it != sons.end(); it++) {
                std::cout << "Vertex number " << it->second << " by edge
with symbol " << it->first << "\n";
            }
        }
        if (suffLink != -1) {
            std::cout << "Suffix link is " << suffLink << "\n";
        }
        else {
            std::cout << "No suffix link yet\n";
        }
        if (!go.empty()) {
            std::cout << "Transitions from vertex are:\n";
            for (auto it = go.begin(); it != go.end(); it++) {
                std::cout << "Vertex number " << it->second << " by edge
with symbol " << it->first << "\n";
            }
        }
    }
};


std::vector<int> answer;//ответ
int strSize = 0;
char joker;//джокер
std::vector<Vertex> bohr;//бор
void createBorh();
int getSuffLink(int ind);
```

```
int getLink(int ind, char c);
void ahoKorasik(std::string& text, std::vector<int>& counterArr);


int main() {
    std::string text;
    int size = 0;
    std::cin >> text;//текст
    std::vector<int> counterArr(text.length());
    createBorh();//создание бора
    std::cout << "\nBohr info:\n----------------------\n";
    for (unsigned i = 0; i < bohr.size(); i++) {
        bohr[i].printInfo(i);
        std::cout << "----------------------\n";
    }
    std::cout << "\n\n";
    ahoKorasik(text, counterArr);//запуск алгоритма
    std::cout << "\nAutomat info:\n----------------------\n";
    for (unsigned i = 0; i < bohr.size(); i++) {
        bohr[i].printInfo(i);
        std::cout << "----------------------\n";
    }
}


void createBorh() {//создание бора
    Vertex root;
    bool isPrevJoker = false;//был джокер
    int counter = 0;
    bohr.push_back(root);//корень в бор
    std::string buff;//шаблон
    std::cout << "Type pattern:\n";
    std::cin >> buff;
    strSize = buff.length();//длина шаблона
    std::cout << "Type joker:\n";
    std::cin >> joker;//джокер
    std::cout << "Creating bohr...\n";
    int curr = 0;
    for (unsigned int j = 0; j < buff.length(); j++) { //разделяем на
подстроки
        std::cout << "Work with \'" << buff[j] << "\' symbol\n";
        if (buff[j] == joker) {//если джокер
            std::cout << "It's joker ";
            if (j == 0) {//первый символ
                std::cout << "and first symbol, skip symbol.\n";
                counter = 0;
                isPrevJoker = true;
                continue;
            }
            if (isPrevJoker) {//если был джокер
```

```
                std::cout << "and previous symbol is joker too, so skip
symbol.\n";
                curr = 0;
                counter = 0;
                continue;
            }
            std::cout << " then previous vertex is leaf, sub-str added to
bohr.\n";
            isPrevJoker = true;
            bohr[curr].isLeaf = true;
            bohr[curr].pos.push_back(j - counter);//позиция в шаблоне
            if (bohr[curr].deep == 0) {
                bohr[curr].deep = counter;
            }
            counter = 0;
            curr = 0;
            continue;
        }
        isPrevJoker = false;//не джокер
        counter++;//увеличиваем длину подстроки
        if (bohr[curr].sons.find(buff[j]) == bohr[curr].sons.end()) {
//если нет вершины в боре
            std::cout << "Symbol not joker, there is no edges by symbol
\'" << buff[j] << "\', creating new vertex...\n";
            Vertex vert(curr, buff[j]);//создаем вершину
            vert.deep = counter; ///////
            bohr.push_back(vert);
            bohr[curr].sons[buff[j]] = bohr.size() - 1;//куда можем
перейти
        }
        curr = bohr[curr].sons[buff[j]];//переход в след вершину
        std::cout << "Going to vertex by symbol \'" << buff[j] << "\'\n";
    }
    if (!isPrevJoker) {//если не джокер
        if (bohr[curr].deep == 0) {
            bohr[curr].deep = counter;
        }
        bohr[curr].isLeaf = true;
        bohr[curr].pos.push_back(buff.length() - counter);//индекс
последнего символа
        std::cout << "Template over, then current vertex is leaf too.\n";
    }
    std::cout << "Bohr created!\n\n";
}


int getLink(int ind, char c) {//получение ссылки
    if (bohr[ind].go.find(c) == bohr[ind].go.end()) {//если переходили
уже
        std::cout << "There is no transitions by symbol\'" << c <<
"\'\n";
```

```
        if (bohr[ind].sons.find(c) != bohr[ind].sons.end()) {//если имеет
сына по этому символу
            std::cout << "But vertex have son by this symbol, add this
edge to tranision map.\n";
            bohr[ind].go[c] = bohr[ind].sons[c];
        }
        else {
            if (ind == 0) {//если мы в корне
                std::cout << "We are in ther root!\n";
                bohr[ind].go[c] = 0;//кольцо
            }
            else {//если еще не переходили
                std::cout << "Trying to transit by the suffix link...\n";
                bohr[ind].go[c] = getLink(getSuffLink(ind), c);//пытаемся
перейти по суф ссылке
            }
        }
    }
    std::cout << "Transition by symbol \'" << c << "\' to " <<
bohr[ind].go[c] << " vertex.\n";
    return bohr[ind].go[c];//возвр ссылку
}


int getSuffLink(int ind) {//получение суф ссылки
    std::cout << "Finding suffix link...\n";
    if (bohr[ind].suffLink == -1) {//еще не определяли ссылку
        std::cout << "No suffix link for this vertex yet \n";
        if (ind == 0 || bohr[ind].prev.second == 0) {//корень или сын
корня
            std::cout << "We are in the root or root-child, so suffix
link is 0!\n";
            bohr[ind].suffLink = 0;
        }
        else {
            std::cout << "Trying to find suffix link by the transition to
the previous vertex and check suffix link of it.\n";
            bohr[ind].suffLink =
getLink(getSuffLink(bohr[ind].prev.second),
bohr[ind].prev.first);//рекурсивно переходим к вершине предку и пытаемся
перейти по ссылке
        }
    }
    std::cout << "Suffix link is " << bohr[ind].suffLink << ".\n";
    return bohr[ind].suffLink;//возвр суффиксальную ссылку
}


void ahoKorasik(std::string& text, std::vector<int>& counterArr)
{//алгоритм
    int curr = 0;
```

```
    int numOfSubStr = 0;//номер подстроки
    for (unsigned int i = 0; i < bohr.size(); i++) {//подсчет подстрок
        if (bohr[i].isLeaf) {
            for (int j = 0; j < bohr[i].pos.size(); j++) {
                numOfSubStr++;
            }
        }
    }
    std::cout << "Start algorithm...\nCurrent vertex is root.\n";
    for (unsigned int i = 0; i < text.length(); i++) { //проходимся по
тексту
        std::cout << "\nTransition by symbol \'" << text[i] << "\' with
index " << i << "\n";
        curr = getLink(curr, text[i]); //куда можем перейти по символу
        std::cout << "----------------------------\nEntry check...\n";
        for (int j = curr; j != 0; j = getSuffLink(j)) { //пока не дойдем
до корня
            if (bohr[j].isLeaf) {//если в конце шаблона
                int indInText = i + 1 - bohr[j].deep;//индекс в тексте
                std::cout << "Vertex is leaf, sub-str found, increasing
counter arr...\n";
                for (unsigned int k = 0; k < bohr[j].pos.size(); k++) {
                    if (indInText - bohr[j].pos[k] >= 0) {//если в
пределах текста
                        std::cout << "Increasing " << indInText -
bohr[j].pos[k] << " element by one, it is " << counterArr[indInText -
bohr[j].pos[k]] + 1 << " now." << "\n";
                        counterArr[indInText - bohr[j].pos[k]] +=
1;//увеличиваем значение вхождений в этом индексе
                    }
                }
            }
        }
        std::cout << "----------------------------\n";
    }
    std::cout << "Counter arr contain next elements(if elements = " <<
numOfSubStr << " template found):\n";
    for (unsigned int i = 0; i < counterArr.size(); i++) {
        std::cout << counterArr[i] << " ";
    }
    std::cout << "\n\nPositions of template are:\n";
    for (unsigned int i = 0; i < counterArr.size() - strSize + 1; i++) {
        if (counterArr[i] == numOfSubStr) {
            std::cout << i + 1 << "\n";
        }
    }

     }
```

ПРИЛОЖЕНИЕ В

КОД ПРОГРАММЫ 3

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



class Vertex {
public:
    std::map<char, int> sons;
    std::map<char, int> go;
    std::pair<char, int> prev;
    int suffLink;
    bool isLeaf;
    bool isNextJoker;
    int deep;
    std::vector<int> pos;

    Vertex() {
        prev.second = -1;
        prev.first = '}';
        isLeaf = false;
        suffLink = -1;
        deep = 0;
        isNextJoker = false;
    }
    Vertex(int prevInd, char prevChar) {
        isLeaf = false;
        suffLink = -1;
        prev.first = prevChar;
        prev.second = prevInd;
        deep = 0;
        isNextJoker = false;
    }
    void printInfo(int i) {
        std::cout << "Vertex number " << i << "\n";
        if (prev.second != -1) {
            std::cout << "Number of parent vertex: " << prev.second << "
symbol of edge: " << prev.first << "\n";
        }
        else {
            std::cout << "It is root\n";
        }
        if (isLeaf) {
            std::cout << "Vertex is leaf.\n";
        }
```

```
        else {
            std::cout << "Kids of vertex are:\n";
            for (auto it = sons.begin(); it != sons.end(); it++) {
                std::cout << "Vertex number " << it->second << " by edge
with symbol " << it->first << "\n";
            }
        }
        if (suffLink != -1) {
            std::cout << "Suffix link is " << suffLink << "\n";
        }
        else {
            std::cout << "No suffix link yet\n";
        }
        if (!go.empty()) {
            std::cout << "Transitions from vertex are:\n";
            for (auto it = go.begin(); it != go.end(); it++) {
                std::cout << "Vertex number " << it->second << " by edge
with symbol " << it->first << "\n";
            }
        }
    }
};


char notJoker;//не джокер
int strSize = 0;
std::vector<int> answer;
std::vector<int> jockersPos;//позиции джокера
char joker;
std::vector<Vertex> bohr;
std::string buff;
void createBorh();
int getSuffLink(int ind);
int getLink(int ind, char c);
void increase(std::vector<int>& counterArr, int indInText, int
j);//увеличить значения в индексах
void decrease(std::vector<int>& counterArr, int indInText);//уменьшить
значения в индексах
void ahoKorasik(std::string& text, std::vector<int>& c);


int main() {
    std::string text;
    int size = 0;
    std::cin >> text;
    std::vector<int> c(text.length());
    createBorh();
    std::cout << "\nBohr info:\n----------------------\n";
    for (unsigned i = 0; i < bohr.size(); i++) {
        bohr[i].printInfo(i);
        std::cout << "----------------------\n";
```

```
    }
    std::cout << "\n\n";
    ahoKorasik(text, c);
    std::cout << "\nAutomat info:\n----------------------\n";
    for (unsigned i = 0; i < bohr.size(); i++) {
        bohr[i].printInfo(i);
        std::cout << "----------------------\n";
    }
}


void createBorh() {
    Vertex root;
    int deepCounter = 0;
    bool flag = false;
    int counter = 0;
    bohr.push_back(root);
    std::cout << "Type pattern:\n";
    std::cin >> buff;
    strSize = buff.length();
    std::cout << "Type joker:\n";
    std::cin >> joker;
    std::cout << "Type non joker:\n";
    std::cin >> notJoker;
    int curr = 0;
    std::cout << "Creating bohr...\n";
    for (unsigned int j = 0; j < buff.length(); j++) { //делим на
подстроки
        std::cout << "Work with \'" << buff[j] << "\' symbol\n";
        if (buff[j] == joker) {//если джокер
            std::cout << "It's joker ";
            if (j == 0) {//первый символ
                std::cout << "and first symbol, skip symbol, push
position of joker to non-joker arr.\n";
                jockersPos.push_back(j);//записываем индекс джокера
                counter = 0;
                deepCounter = 0;
                flag = true;
                continue;
            }
            if (flag) {//если пред джокер
                std::cout << "and previous symbol is joker too, so skip
symbol, push position of joker to non-joker vertex.\n";
                jockersPos.push_back(j);
                deepCounter = 0;
                curr = 0;
                counter = 0;
                continue;
            }
            std::cout << " then previous vertex is leaf, sub-str added to
bohr, push position of joker to non-joker vertex.\n";
```

```
            jockersPos.push_back(j);
            flag = true;
            bohr[curr].isLeaf = true;
            bohr[curr].pos.push_back(j - counter);
            if (bohr[curr].deep == 0) {
                bohr[curr].deep = counter;
            }
            counter = 0;
            curr = 0;
            continue;
        }
        flag = false;
        counter++;
        if (bohr[curr].sons.find(buff[j]) == bohr[curr].sons.end()) {
//если не создавали еще ребро
            std::cout << "Symbol not joker, there is no edges by symbol
\'" << buff[j] << "\', creating new vertex...\n";
            Vertex vert(curr, buff[j]);
            vert.deep = counter;
            bohr.push_back(vert);
            bohr[curr].sons[buff[j]] = bohr.size() - 1;

        }
        curr = bohr[curr].sons[buff[j]];
        std::cout << "Going to vertex by symbol \'" << buff[j] << "\'\n";
    }
    if (!flag) {//если послед не джокер
        if (bohr[curr].deep == 0) {
            bohr[curr].deep = counter;
        }
        bohr[curr].isLeaf = true;
        bohr[curr].pos.push_back(buff.length() - counter);
        std::cout << "Template over, then current vertex is leaf too.\n";
    }
    std::cout << "Bohr created!\n\n";
}


int getLink(int ind, char c) {//получение ссылки
    if (bohr[ind].go.find(c) == bohr[ind].go.end()) {//если есть переход
        std::cout << "There is no transitions by symbol\'" << c <<
"\'\n";
        if (bohr[ind].sons.find(c) != bohr[ind].sons.end()) {//если есть
переход в боре от родителя
            std::cout << "But vertex have son by this symbol, add this
edge to tranision map.\n";
            bohr[ind].go[c] = bohr[ind].sons[c];
        }
        else {
            if (ind == 0) {//если первый символ
                std::cout << "We are in ther root!\n";
```

```
                bohr[ind].go[c] = 0;
            }
            else {
                std::cout << "Trying to transit by the suffix link...\n";
                bohr[ind].go[c] = getLink(getSuffLink(ind), c);//пыьаемся
получить суф ссылку
            }
        }
    }
    std::cout << "Transition by symbol \'" << c << "\' to " <<
bohr[ind].go[c] << " vertex.\n";
    return bohr[ind].go[c];//возвр состояние
}


int getSuffLink(int ind) {//получение суф ссылки
    std::cout << "Finding suffix link...\n";
    if (bohr[ind].suffLink == -1) {//если еще не определяли
        std::cout << "No suffix link for this vertex yet \n";
        if (ind == 0 || bohr[ind].prev.second == 0) {//корень или сын
корня
            std::cout << "We are in the root or root-child, so suffix
link is 0!\n";
            bohr[ind].suffLink = 0;
        }
        else {
            std::cout << "Trying to find suffix link by the transition to
the previous vertex and check suffix link of it.\n";
            bohr[ind].suffLink =
getLink(getSuffLink(bohr[ind].prev.second),
bohr[ind].prev.first);//пытаемся рекурсивно перейти по ссылке
        }
    }
    std::cout << "Suffix link is " << bohr[ind].suffLink << ".\n";
    return bohr[ind].suffLink;//возвр суф ссылку
}


void ahoKorasik(std::string& text, std::vector<int>& counterArr)
{//алгоритм
    int curr = 0;
    int tmp = 0;//количество подстрок
    for (unsigned int i = 0; i < bohr.size(); i++) {
        if (bohr[i].isLeaf) {
            for (int j = 0; j < bohr[i].pos.size(); j++) {
                tmp++;
            }
        }
    }
    std::cout << "Start algorithm...\nCurrent vertex is root.\n";
```

```
    for (unsigned int i = 0; i < text.length(); i++) { //проходимся по
тексту
        std::cout << "\nTransition by symbol \'" << text[i] << "\' with
index " << i << "\n";
        curr = getLink(curr, text[i]); //куда можем перейти по символу
        std::cout << "----------------------------\nEntry check...\n";
        for (int j = curr; j != 0; j = getSuffLink(j)) { //пока не дойдем
до корня
            if (bohr[j].isLeaf) {//если в конце шаблона
                int indInText = i + 1 - bohr[j].deep;//индекс в тексте
                std::cout << "Vertex is leaf, sub-str found, increasing
counter arr...\n";
                for (unsigned int k = 0; k < bohr[j].pos.size(); k++) {
                    if (indInText - bohr[j].pos[k] >= 0) {//если в
пределах текста
                        std::cout << "Increasing " << indInText -
bohr[j].pos[k] << " element by one, it is " << counterArr[indInText -
bohr[j].pos[k]] + 1 << " now." << "\n";
                        counterArr[indInText - bohr[j].pos[k]] +=
1;//увеличиваем значение вхождений в этом индексе
                    }
                }
            }
        }
        std::cout << "----------------------------\n";
    }
    std::cout << "Counter arr contain next elements(if elements = " <<
tmp << " template found):\n";
    for (unsigned int i = 0; i < counterArr.size(); i++) {
        std::cout << counterArr[i] << " ";
    }
    bool is_correct = true;
    std::cout << "\n\nPositions of template are:\n";
    for (unsigned int i = 0; i < counterArr.size(); i++) {
        if (counterArr[i] == tmp) {
            for (int k = i; k < i + strSize; k++) {
                if (buff[k - i] && text[k] == notJoker && buff[k - i] ==
joker) {
                    is_correct = false;
                    std::cout << "Element i= " << i << " has notJoker" <<
std::endl;
                    break;
                }
            }
            if(is_correct)
            std::cout << i + 1 << "\n";
            is_correct = true;
        }
    }

}
```