МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритм Ахо-Корасик

Студент гр. 8383 _____ Бессуднов Г. И.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2020

Цель работы.

Научиться использовать алгоритм Ахо-Корасика множественного поиска индексов вхождений строк-образцов в строку-текст.

Алгоритм Ахо-Корасик.

Задание.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

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

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

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

Выход:

Все вхождения образцов из P в T.

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

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

(нумерация образцов начинается с 1).

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

Sample Input:

CCCA

1

CC

Sample Output:

1 1

2 1

Вариант 3. Вычислить длину самой длинной цепочки из суффиксных ссылок и самой длинной цепочки из конечных ссылок в автомате.

Описание алгоритма.

Вначале необходимо построить бор из строк-образцов. Бор - это префиксное дерево, которое строится следующим образом: создается корневая вершина дерева, которая является нулевым символом. Вершина становится текущей. Далее поочередно считываются все строки-образцы и посимвольно обрабатываются. Если из текущей вершины нет перехода по текущему символу, то создается новая вершина, которой присваивается данный символ, и она становится текущей. Если же преход есть, то он осуществляется. Если символ - конечный в строке, то текущая вершина помечается и становится терминальной.

Текущая вершина устанавливается в корень. Затем алгоритм обрабатывает каждый символ текста. Выполняется переход из текущей вершины в вершину по символу. Если есть прямой переход, то он выполняется, если его нет, то происходит переход по суффиксной ссылке.

Суффиксная ссылка для каждой вершины - это вершина, в которой оканчивается самый длинный суффикс строки. Для корня бора суффиксная ссылка - это ссылка на себя же. Суффиксная сслыка находится следующим образом: происходит переход в вершину по суффиксной ссылке родителя, затем из этой вершины делается переход по символу.

После перехода, происходит проверка. Из текущей вершины происходит переход по суффиксным ссылкам, пока не дойдем до корня. Если вершина - терминальная, то вхождение найдено.

Алгоритм завершает работу, когда каждый символ строки-текста был обработан.

Исходный код программы, решающей задачу поиска множества подстрок в строке при помощи алгоритма Ахо-Корасик, представлен в приложении А.

Для выполнения требований задания индивидуализации был реализован следующий алгоритм. Когда происходит проверка символа после сделанного шага, проивзодится подсчет количества ссылок (из того условия, что при проверке делаются переходы по суффиксным ссылкам до корня), при каждой итерации увеличивается счетчик, запоминается большее число. Для подсчета

наибольшего числа конечных ссылок счетчик увеличивается не каждую итераицию, а только в ту, которая ведет в терминальный символ.

Описание функций и структур данных.

1.

```
struct Node{
char symbol; //символ
int patternEnd; //номер образца
int patternOffset; //длина образца
int sufLink; //суффикс ссфылка
int parent; //родитель
std::map<char, int> moves; //ходы из вершины
std::map<char, int> children; //дети вершины

Node() : symbol('0'), patternEnd(0), patternOffset(0), sufLink(-1),
parent(-1) {
}
Node(char symb, int link, int parent) : symbol(symb), patternEnd(0),
patternOffset(0), sufLink(link), parent(parent) {
}
```

Структура для хранения информации о вершине бора.

2.

```
class Bor {
private:
    std::vector<Node> patternsBor; //бор
    std::vector<std::pair<int, int>> answers; //ответы
    int borStepPos; //текущая вершина

    int maxSufLink; //макимальная длина суффикс-ссылки
    int maxEndLink; //максимальная длина конечной ссылки

    int step(int node, char symb); //функция перехода
```

```
        int getLink(int node); //функция получения ссылки
        void nodeCheck(int index); //функция проверки вершины
        void printAnswers(); //функция печати ответов
    public:
        Bor();
        ~Bor()=default;
        void printBor(); //печать бора
        void addStringToBor(const std::string &str, int
index);//добавить строку в бор
        void findPatterns(const std::string &text);//найти все вхождения
    };
```

Класс для хранения информации о боре и работе с ним. Содержит следующие функции:

`int Bor::step(int node, char symb);` - функция перехода из вершины `node` по символу `symb`. Возвращает индекс движения, который можно сделать.

`int Bor::getLink(int node);` - получение суффикс ссылки из вершины `node`. Возвращает суффикс ссылку.

`void Bor::nodeCheck(int index);` - проверка символа по индексу `index` в боре.

`void Bor::addStringToBor(const std::string &str, int index);` - функция добавления строки `str` с номером `index` в бор.

`void Bor::findPatterns(const std::string &text);` - функция нахождения всех вхождений ранее инициализированных подстрок в тексте `text`.

Оценка сложности алгоритма по памяти и по времени

Алгоритм имеет сложность по памяти O(m), где m – длина всех строк-образцов т.к. в худшем случае в боре хранится каждая буква строк-образцов.

Алгоритм строит бор за O(m * log(k)), где k – количество символов алфавита, m – длина всех строк-образцов т.к. в худшем случае в бор добавляется каждая вершина, так как для хранения информации о детях вершины

используется std::map (вставка в ней имеет временную сложность O(log(k))). Алгоритм пройдется по всей длинне текста t, получая переходы из словаря за log(k), после каждого перехода будут проверены все суффиксные ссылки до корня, которых максимально m штук, итого сложность алгоритма составит O((t+2m) * log(k)).

Тестирование первой программы

Входные данные:

aca

1

a

```
aca
1
a
<----------------->
Making step by symbol: a
No move from 0 to a
...Finding...
Got child with symbol a
...Move found...
From 0 to a is 1


Checking (0, 0)
In finish state at(a, 10
Link from (a, 1) is not set
In the root, or parent is root
Link from (a, 1) is 0

<----------------->

<----------------->
Making step by symbol: c
No move from a to c
...Finding...
...No child, go by link...
Link from (a, 1) is 0

No move from 0 to c
...Finding...
...No child, go by link...
...In the root...
...Move found...
From 0 to c is 0

...Move found...
From a to c is 0


Checking (a, 1)
<----------------->
```

```
<----------------->
Making step by symbol: a
...Move found...
From 0 to a is 1


Checking (T, 2)
In finish state at(a, 10
Link from (a, 1) is 0

<----------------->

-------ANSWERS-------
   Pos  PNum
     1     1
     3     1
Max suf link: 1
Max end link: 1
--------Bor--------
  Symbol  Parent    Link    Sons
       0       0       0   (a, 1)
       a       0       0
```

Ввод	Вывод
abcabcb 5 abcabcb abcb cabcb cb b	1 1 2 5 3 3 4 2 5 5 6 4 7 5
abbcbcgaha 2 qpqpoiqi tryeiw	
abcabcabc 3 cab bca abc	1 3 2 2 3 1 4 3 5 2 6 1 7 3
thequickbrownfoxjumpsoverthelazydog 3 quick fox over	4 1 14 2 22 3

Алгоритм Ахо-Корасик с джокером

Задание

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

В шаблоне встречается специальный символ, именуемого 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 неопределенной длины. В шаблоне входит хотя бы один символ не джокер, те шаблоны вида ??? недопустимы.

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

Вход:

Текст (T, $1 \leq |T| \leq 100000$)

Шаблон (P,$1 \leq |P| \leq 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания.

Sample Input:

ACT

A$

$

Sample Output:

1

Описание алгоритма

Для поиска шаблона, содержащего джокер, строка разбивается на части по символу джокера. Для каждой такой части сохраняется ее место в первоначальном шаблоне. Получившиеся подстроки ищутся в тексте с помощью

алгоритма Ахо-Корасика. При каждом совпадении вычисляется место в массиве позиций по формуле $j-l_i+1$, где j - позиция в тексте, l_i - длина подстроки и эта позиция увеличивается на 1. Если в массиве позиций есть место, значение которого равно количеству получившихся подстрок и строка-образец может встать на это место, то совпадение найдено.

Исходный код программы, рещающей задачу поиска подстроки с джокерами в строке при помощи алгоритма Ахо-Корасик, представлен в приложении Б.

Описание функций и структур данных

1.

```
struct Node{
	char symbol; //символ
	int sufLink; //суффикс ссылка
	int parent; //родитель
	std::vector<int> pos; //позиции в строке-образце
	std::map<char, int> moves; //ходы из вершины
	std::map<char, int> children; //дети вершины

	Node() : symbol('0'), sufLink(-1), parent(-1) {
	}
	Node(char symb, int link, int parent) : symbol(symb),
	sufLink(link), parent(parent) {
	}
};
```

Структура для хранения информации о вершине бора.

2.

```
class Bor {
private:
    std::vector<Node> patternsBor; //бор
    std::vector<int> positions; //массив позиций
```

```
        int initialPatternLength; //исходный размер строки-образца
        int localPatternNumber; //количество подстрок без джокеров в строке-образце
        int borStepPos; //текущая вершина
        char joker; //символ-джокер

        int maxSufLink; //макимальная длина суффикс-ссылки
        int maxEndLink; //максимальная длина конечной ссылки

        int step(int node, char symb); //функция перехода
        int getLink(int node); //цункция получения ссылки
        void addStringToBor(const std::string &str, int index); //функция добавлени строки к бору
        void nodeCheck(int index); //функция проверки вершины
        void printAnswers(); //функция печати ответов
    public:
        Bor(); //конструктор
        ~Bor()=default; //деструктор

        void printBor();//печать бора
        void initBor(char joker, int textLength, const std::string pattern); //функция инициализации бора
        void findPatterns(const std::string &text); //функция нахождения вхождений
    };
```

Класс для хранения информации о боре и работе с ним. Содержит следующие функции:

`int Bor::step(int node, char symb);` - функция перехода из вершины `node` по символу `symb`. Возвращает индекс движения, который можно сделать.

`int Bor::getLink(int node);` - получение суффикс ссылки из вершины `node`. Возвращает суффикс ссылку.

`void Bor::nodeCheck(int index);` - проверка символа по индексу `index` в боре.

`void Bor::addStringToBor(const std::string &str, int index);` - функция добавления строки `str` с номером `index` в бор.

`void Bor::findPatterns(const std::string &text);` - функция нахождения всех вхождений ранее инициализированных подстрок в тексте `text`.

`void Bor::initBor(char joker, int textLength, const std::string pattern);` - функция для инициализации бора при помощи длины текста `textLength`, символа джокера `joker` и строки-образца `pattern`.

Оценка сложности алгоритма по памяти и по времени

Так как в программе используется алоритм Ахо-Корасик, то временная сложность составит O((t+2m)*log(k)), где t – длина текста, m – длина шаблона, k – количество символов алфавита.

Сложность по памяти составит O(m + t), так как в алгоритме используется массив-счетчик позиций длины t, где t – длина текста, а O(m) памяти требуется для хранения бора.

Тестирование второй программы

Входные данные:

aacc

axx

x

```
aacc
axx
x
<----------------->
Making step by symbol: a
No move from 0 to a
...Finding...
Got child with symbol a
...Move found...
From 0 to a is 1


Checking (0, 0)
In finish state at(a, 10
Link from (a, 1) is 0

<----------------->

<----------------->
Making step by symbol: a
No move from a to a
...Finding...
...No child, go by link...
Link from (a, 1) is 0

...Move found...
From 0 to a is 1

...Move found...
From a to a is 1


Checking (a, 1)
In finish state at(a, 10
Link from (a, 1) is 0

<----------------->
```

```
<----------------->
Making step by symbol: c
No move from a to c
...Finding...
...No child, go by link...
Link from (a, 1) is 0

No move from 0 to c
...Finding...
...No child, go by link...
...In the root...
...Move found...
From 0 to c is 0

...Move found...
From a to c is 0


Checking (╥, 2)
<----------------->

<----------------->
Making step by symbol: c
...Move found...
From 0 to c is 0


Checking (A, 3)
<----------------->

-------ANSWERS-------
  Position
         1
         2
Max suf link: 1
Max end link: 1
--------Bor---------
  Symbol  Parent    Link    Sons
       0       0       0   (a, 1)
       a       0       0
```

Ввод	Вывод
ACACGGG XCXXG X	1 3
abcbabcbbbc axcxxbc x	1 5
axxaxxabcabcx a??a??abc? ?	1 4
acbcvcb vbxx x	

Вывод

В ходе выполнения лабораторной работы был изучен алоритм Ахо-Корасика для нахождения подстроки в строке.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД АЛГОРИТМА АХО-КОРАСИК

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



struct Node{
    char symbol; //символ
    int patternEnd; //номер образца
    int patternOffset; //длина образца
    int sufLink; //суффикс ссфылка
    int parent; //родитель
    std::map<char, int> moves; //ходы из вершины
    std::map<char, int> children; //дети вершины

    Node() : symbol('0'), patternEnd(0), patternOffset(0), sufLink(-1),
parent(-1) {
    }
    Node(char symb, int link, int parent) : symbol(symb), patternEnd(0),
    patternOffset(0), sufLink(link), parent(parent) {
    }
};

class Bor {
private:
    std::vector<Node> patternsBor; //бор
    std::vector<std::pair<int, int>> answers; //ответы
    int borStepPos; //текущая вершина

    int maxSufLink; //макимальная длина суффикс-ссылки
    int maxEndLink; //максимальная длина конечной ссылки

    int step(int node, char symb); //функция перехода
    int getLink(int node); //функция получения ссылки
    void nodeCheck(int index); //функция проверки вершины
    void printAnswers(); //функция печати ответов
public:
    Bor();
    ~Bor()=default;
    void printBor(); //печать бора
```

```
    void addStringToBor(const std::string &str, int index);//добавить строку в
бор
    void findPatterns(const std::string &text);//найти все вхождения
};

//компаратор для ответов
bool answerComp(const std::pair<int, int> &p1, const std::pair<int, int> &p2) {
    if (p1.first == p2.first) {
        return p1.second < p2.second;
    }

    return p1.first < p2.first;
}

int main() {
    Bor bor;
    std::string text;
    std::string currentPattern;
    int patternsNumber;

    std::cin >> text;
    std::cin >> patternsNumber;
    for(int i = 0; i < patternsNumber; i++) {
        std::cin >> currentPattern;
        bor.addStringToBor(currentPattern, i + 1);
    }

    bor.findPatterns(text);
    bor.printBor();

    return 0;
}



Bor::Bor() : borStepPos(0), maxEndLink(0), maxSufLink(0) {
    patternsBor.push_back(Node('0', 0, 0));
}

//печать бора
void Bor::printBor() {
    std::cout << "--------Bor---------" << std::endl;
    std::cout << std::setw(8) << "Symbol";
    std::cout << std::setw(8) << "Parent";
    std::cout << std::setw(8) << "Link";
    std::cout << std::setw(8) << "Sons" << std::endl;
    for (auto &node : patternsBor) {
        std::cout << std::setw(8) << node.symbol;
```

```
        std::cout << std::setw(8) << node.parent;
        std::cout << std::setw(8) << node.sufLink;
        std::cout << std::setw(4);

        std::map<char, int>::iterator mapIt;

        for(mapIt = node.children.begin(); mapIt != node.children.end();
mapIt++) {
            std::cout << "(" << mapIt->first << ", " << mapIt->second << ") ";
        }

        std::cout << std::endl;
    }
}

//печать ответов
void Bor::printAnswers() {
    std::sort(answers.begin(), answers.end(), answerComp);

    std::cout << "-------ANSWERS-------" << std::endl;
    std::cout << std::setw(6) << "Pos" << std::setw(6) << "PNum" << std::endl;
    for (auto &answer : answers) {
        std::cout << std::setw(6) << answer.first << std::setw(6) <<
answer.second << std::endl;
    }
    std::cout << "Max suf link: " << maxSufLink << std::endl;
    std::cout << "Max end link: " << maxEndLink << std::endl;
}


void Bor::addStringToBor(const std::string &str, int index) {
    int curPos;
    curPos = 0;
    for(int i = 0; i < str.length(); i++) {
        if(patternsBor[curPos].children[str[i]] != 0) {
            //если переход по символу возможен
            curPos = patternsBor[curPos].children[str[i]];
        } else {
            //если переход невозможен, создаем новую вершину
            patternsBor.push_back(Node(str[i], -1, curPos));
            patternsBor[curPos].children[str[i]] = patternsBor.size() - 1;
            curPos = patternsBor.size() - 1;
        }
    }
    //установить номер образца
    patternsBor[curPos].patternEnd = index;
    //установить длину образца
    patternsBor[curPos].patternOffset = str.length();
```

```
}

void Bor::findPatterns(const std::string &text) {
    borStepPos = 0;

    for (int i = 0; i < text.length(); i++) {
        std::cout << "<----------------->" << std::endl;
        std::cout << "Making step by symbol: " << text[i] << std::endl;
        //делаем шаг
        borStepPos = step(borStepPos, text[i]);
        std::cout << std::endl;
        //делаем проверку
        nodeCheck(i);

        std::cout << "<----------------->" << std::endl;
        std::cout << std::endl;
    }
    //печатаем ответы
    printAnswers();
}

int Bor::getLink(int node) {
    //если ссылки нет
    if (patternsBor[node].sufLink == -1) {
        std::cout << "Link from (" << patternsBor[node].symbol << ", " << node
<< ") is not set" << std::endl;
        //если находимся в корне, или корень - это родитель
        if (node == 0 || patternsBor[node].parent == 0) {
            std::cout << "In the root, or parent is root" << std::endl;
            patternsBor[node].sufLink = 0;
        //иначе делаем шаг из суффикс ссылки родителя
        } else {
            patternsBor[node].sufLink = step(getLink(patternsBor[node].parent),
patternsBor[node].symbol);
        }
    }

    std::cout << "Link from (" << patternsBor[node].symbol << ", " << node <<
") is "
    << patternsBor[node].sufLink << std::endl << std::endl;
    //возвращаем суффикс ссылку
    return patternsBor[node].sufLink;
}

void Bor::nodeCheck(int index) {
    //для подсчета текущей длины суффиксных и конечных ссылок
    int currentSufLinks = 0;
    int currentEndLinks = 0;
```

```
    std::cout << "Checking (" << patternsBor[index].symbol << ", " << index <<
")" << std::endl;

    for (int pos = borStepPos; pos != 0; pos = getLink(pos), ++currentSufLinks)
{
        //если символ терминальный
        if (patternsBor[pos].patternEnd) {
            std::cout << "In finish state at(" << patternsBor[pos].symbol << ",
" << pos << "0" << std::endl;

            answers.push_back(std::pair<int, int>(index -
patternsBor[pos].patternOffset + 2
            , patternsBor[pos].patternEnd));

            ++currentEndLinks;
        }
    }

    //если текущая длина суффикс или конечной ссылки больше, то обновим
значение
    maxEndLink = (currentEndLinks > maxEndLink) ? currentEndLinks : maxEndLink;
    maxSufLink = (currentSufLinks > maxSufLink) ? currentSufLinks : maxSufLink;
}

int Bor::step(int node, char symb) {
    std::map<char, int>::iterator mapIt;
    mapIt = patternsBor[node].moves.find(symb);

    //если нет шага по текущему символу
    if (mapIt == patternsBor[node].moves.end()) {
        std::cout << "No move from " << patternsBor[node].symbol << " to " <<
symb << std::endl;
        std::cout << "...Finding..." << std::endl;

        mapIt = patternsBor[node].children.find(symb);
        // если был найден прямой переход по символу
        if (mapIt != patternsBor[node].children.end()) {
            std::cout << "Got child with symbol " << symb << std::endl;

            //прямой переход
            patternsBor[node].moves[symb] = patternsBor[node].children[symb];
        //прямого преходанет, идем по суффикс ссылке
        } else {
            std::cout << "...No child, go by link..." << std::endl;

            //если в корне, то движемся в корень
            if (node == 0) {
                std::cout << "...In the root..." << std::endl;
```

```
                patternsBor[node].moves[symb] = 0;
            //иначе движемся по суффикс сылке символа
            } else {
                patternsBor[node].moves[symb] = step(getLink(node), symb);
            }
        }
    }

    std::cout << "...Move found..." << std::endl;
    std::cout << "From " << patternsBor[node].symbol << " to " << symb << " is
"
    << patternsBor[node].moves[symb] << std::endl << std::endl;

    //возвращаем преход
    return patternsBor[node].moves[symb];
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД АЛГОРИТМА АХО-КОРАСИК С ДЖОКЕРАМИ

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


struct Node{
    char symbol; //символ
    int sufLink; //суффикс ссылка
    int parent; //родитель
    std::vector<int> pos; //позиции в строке-образце
    std::map<char, int> moves; //ходы из вершины
    std::map<char, int> children; //дети вершины

    Node() : symbol('0'), sufLink(-1), parent(-1) {
    }
    Node(char symb, int link, int parent) : symbol(symb),
    sufLink(link), parent(parent) {
    }
};

class Bor {
private:
    std::vector<Node> patternsBor; //бор
    std::vector<int> positions; //массив позиций
    int initialPatternLength; //исходный размер строки-образца
    int localPatternNumber; //количество подстрок без джокеров в строке-образце
    int borStepPos; //текущая вершина
    char joker; //символ-джокер

    int maxSufLink; //макимальная длина суффикс-ссылки
    int maxEndLink; //максимальная длина конечной ссылки

    int step(int node, char symb); //функция перехода
    int getLink(int node); //цункция получения ссылки
    void addStringToBor(const std::string &str, int index); //функция добавлени
строки к бору
    void nodeCheck(int index); //функция проверки вершины
    void printAnswers(); //функция печати ответов
public:
    Bor(); //конструктор
```

```
    ~Bor()=default; //деструктор

    void printBor();//печать бора
    void initBor(char joker, int textLength, const std::string pattern);
//функция инициализации бора
    void findPatterns(const std::string &text); //функция нахождения вхождений
};


int main() {
    Bor bor;
    std::string text;
    std::string pattern;
    char joker;

    std::cin >> text;
    std::cin >> pattern;
    std::cin >> joker;
    bor.initBor(joker, text.length(), pattern);

    bor.findPatterns(text);
    bor.printBor();

    return 0;
}



Bor::Bor() : borStepPos(0), maxEndLink(0), maxSufLink(0) {
    patternsBor.push_back(Node('0', 0, 0));
}

void Bor::initBor(char joker, int textLength, const std::string pattern) {
    //накопитель строки
    std::string patternTemp;

    //зануление веткора позиций
    positions.clear();
    positions.resize(textLength, 0);


    localPatternNumber = 0;
    initialPatternLength = pattern.length();

    int i = 0;
    //проход по образцу
    for(; i < pattern.length(); i++) {
        //если символ в паттерне не джокер, то сохраним его
```

```
        if (pattern[i] != joker) {
            patternTemp.push_back(pattern[i]);
        } else {
        //если символ джокер, то добавить получившуюся строку в бор и очистить
накопитель
            addStringToBor(patternTemp, i);
            patternTemp.clear();
        }
    }
    //добавляем строку, которая получилась в конце
    addStringToBor(patternTemp, i);
}

//печать бора
void Bor::printBor() {
    std::cout << "--------Bor---------" << std::endl;
    std::cout << std::setw(8) << "Symbol";
    std::cout << std::setw(8) << "Parent";
    std::cout << std::setw(8) << "Link";
    std::cout << std::setw(8) << "Sons" << std::endl;
    for (auto &node : patternsBor) {
        std::cout << std::setw(8) << node.symbol;
        std::cout << std::setw(8) << node.parent;
        std::cout << std::setw(8) << node.sufLink;
        std::cout << std::setw(4);

        std::map<char, int>::iterator mapIt;

        for(mapIt = node.children.begin(); mapIt != node.children.end();
mapIt++) {
            std::cout << "(" << mapIt->first << ", " << mapIt->second << ") ";
        }

        std::cout << std::endl;
    }
}

//печать ответов
void Bor::printAnswers() {
    std::cout << "-------ANSWERS-------" << std::endl;
    std::cout << std::setw(10) << "Position" << std::endl;
    for(int i = 0; i < positions.size(); i++) {
        if (positions[i] == localPatternNumber) {
            std::cout << std::setw(10) << i + 1 << std::endl;
        }
    }
    std::cout << "Max suf link: " << maxSufLink << std::endl;
    std::cout << "Max end link: " << maxEndLink << std::endl;
```

```
}

//добавляем строку в бор
void Bor::addStringToBor(const std::string &str, int index) {
    //если длина строки 0, то ничего не делаем
    if (str.length() == 0) {
        return;
    }
    int curPos;
    curPos = 0;
    for(int i = 0; i < str.length(); i++) {
        //если символ уже есть, то проходим по нему
        if(patternsBor[curPos].children[str[i]] != 0) {
            curPos = patternsBor[curPos].children[str[i]];
        } else {
            //если символа нет, то добавляем
            patternsBor.push_back(Node(str[i], -1, curPos));
            patternsBor[curPos].children[str[i]] = patternsBor.size() - 1;
            curPos = patternsBor.size() - 1;
        }
    }
    localPatternNumber++;
    //сохраняем номер позиции для строки
    patternsBor[curPos].pos.push_back(index);
}

void Bor::findPatterns(const std::string &text) {
    borStepPos = 0;

    for (int i = 0; i < text.length(); i++) {
        std::cout << "<----------------->" << std::endl;
        std::cout << "Making step by symbol: " << text[i] << std::endl;

        //делаем шаг
        borStepPos = step(borStepPos, text[i]);
        std::cout << std::endl;

        //делаем проверку
        nodeCheck(i);

        std::cout << "<----------------->" << std::endl;
        std::cout << std::endl;
    }

    //печатаем ответы
    printAnswers();

}
```

```
int Bor::getLink(int node) {
    //если суффикс ссылки еще нет
    if (patternsBor[node].sufLink == -1) {
        std::cout << "Link from (" << patternsBor[node].symbol << ", " << node
<< ") is not set" << std::endl;
        //если находимся в корне, или корень - это родитель
        if (node == 0 || patternsBor[node].parent == 0) {
            std::cout << "In the root, or parent is root" << std::endl;
            patternsBor[node].sufLink = 0;
        //иначе делаем шаг из суффикс ссылки родителя
        } else {
            patternsBor[node].sufLink = step(getLink(patternsBor[node].parent),
patternsBor[node].symbol);
        }
    }

    std::cout << "Link from (" << patternsBor[node].symbol << ", " << node <<
") is "
    << patternsBor[node].sufLink << std::endl << std::endl;
    //возвращаем суффикс ссылку
    return patternsBor[node].sufLink;
}

void Bor::nodeCheck(int index) {
    //для подсчета текущей длины суффиксных и конечных ссылок
    int currentSufLinks = 0;
    int currentEndLinks = 0;
    std::cout << "Checking (" << patternsBor[index].symbol << ", " << index <<
")" << std::endl;

    for (int pos = borStepPos; pos != 0; pos = getLink(pos), ++currentSufLinks)
{
        //если символ терминальный
        if (!patternsBor[pos].pos.empty()) {
            std::cout << "In finish state at(" << patternsBor[pos].symbol << ",
" << pos << "0" << std::endl;
            ++currentEndLinks;
            //для каждой терминальной позиции увеличить значение в векторе
            for(auto &startPos : patternsBor[pos].pos) {
                int posIndex;
                posIndex = index - startPos + 1;
                if(posIndex >= 0 && posIndex + initialPatternLength - 1 <
positions.size()) {
                    positions[posIndex]++;
                }
            }
        }
```

```
    }

    //если текущая длина суффикс или конечной ссылки больше, то обновим
значение
    maxEndLink = (currentEndLinks > maxEndLink) ? currentEndLinks : maxEndLink;
    maxSufLink = (currentSufLinks > maxSufLink) ? currentSufLinks : maxSufLink;
}

int Bor::step(int node, char symb) {
    std::map<char, int>::iterator mapIt;
    mapIt = patternsBor[node].moves.find(symb);

    //если нет шага по текущему символу
    if (mapIt == patternsBor[node].moves.end()) {
        std::cout << "No move from " << patternsBor[node].symbol << " to " <<
symb << std::endl;
        std::cout << "...Finding..." << std::endl;

        mapIt = patternsBor[node].children.find(symb);
        // если был найден прямой переход по символу
        if (mapIt != patternsBor[node].children.end()) {
            std::cout << "Got child with symbol " << symb << std::endl;
            //прямой переход
            patternsBor[node].moves[symb] = patternsBor[node].children[symb];
        //прямого преходанет, идем по суффикс ссылке
        } else {
            std::cout << "...No child, go by link..." << std::endl;
            //если в корне, то движемся в корень
            if (node == 0) {
                std::cout << "...In the root..." << std::endl;

                patternsBor[node].moves[symb] = 0;
            //иначе движемся по суффикс сылке символа
            } else {
                patternsBor[node].moves[symb] = step(getLink(node), symb);
            }
        }
    }

    std::cout << "...Move found..." << std::endl;
    std::cout << "From " << patternsBor[node].symbol << " to " << symb << " is
"
    << patternsBor[node].moves[symb] << std::endl << std::endl;

    //возвращаем преход
    return patternsBor[node].moves[symb];
}
```