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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студент гр. 9382 _____ Савельев И.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

На практике ознакомиться с алгоритмом Ахо-Корасик.

Задание.

Вариант 5. Вычислить максимальное количество дуг, исходящих из одной вершины в боре; вы́резать из строки поиска все найденные образцы и вывести остаток строки поиска.

Stepik 1

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

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

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

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:

```
NTAG
3
TAGT
TAG
T
```

Sample Output:

2 2
2 3

Stepik 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 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.

Sample Input:

ACTANCA
A$$A$
$

Sample Output:

1

Описание алгоритма.

Считывается исходный текст и паттерны, по паттернам строится бор по следующему алгоритму: создается корень, после чего каждый из шаблонов добавляется в бор посимвольно, пока есть возможность перехода по дугам с соответствующими символами новые вершины не добавляются в бор, если из текущей вершины нет дуги с соответствующим символом, то из текущей вершины строиться дуга в новую вершину, если паттерн заканчивается на текущей вершине она отмечается как терминальная.

Затем алгоритм начинает посимвольно считывать исходный текст, переходя по дуге с символом в следующую вершину, если есть прямой переход или выполняя переход по суффиксной ссылке при его отсутствии. Если для заданной вершины суффиксная ссылка не была найдена она вычисляется по следующему алгоритму: проверяем является ли суффикс ссылка корнем или его сыном, если да то суффиксная ссылка = 0, если нет, то выполняется рекурсивный поиск, функция переходит в родителя текущей вершины выполняет переход по его суффиксной ссылке в вершину и из неё осуществляет переход по символу. После перехода в новую вершину инициализируется проверка текущей вершины и её суффиксных ссылок на терминальность, если попадаются терминальные вершины в массив ответов заносится индекс начала вхождения паттерна в исходный текст и его порядковый номер. Затем

считывается очередной символ текста, алгоритм прекратит работу, когда будут обработаны все символы исходного текста.

При поиске образца с джокером исходный паттерн разбивается на подпаттерны по символам джокера, которые аналогичным образом добавляются в бор. Затем происходит посимвольный анализ текста с помощью алгоритма описанного выше, с тем отличием, что при попадании в терминальную вершины, что соответствует нахождению в исходном тексте подпаттерна, инкрементируется значение массива res соответствующее началу основного слова относительно найденного подпаттерна. Если значение в какой то ячейке массива res станет равным количеству подпаттернов значит было найдено вхождение паттерна.

Описание структур данных и функций.

```
struct Vertex {
	map<char, int> next;
	map<char, int> go;
	char prev_char;
	int prev_vert;
	int suffix;
	int number;
	bool isTerminal = false;
};
```

struct Vertex - структура описывающая вершину бора.

next - контейнер содержащий дуги исходящие из вершины.

go - контейнер всех возможных переходов из вершины, в том числе и по

суф. ссылкам.

prev_vert - предыдущая вершина.

prev_char - символ по которому пришли в вершину.

suffix - суф. ссылка.

в step1 number - номер паттерна, в step2 number - массив содержащий номера под паттернов.

isTerminal - является ли вершина терминальной.

int next_state(int ind, char chr, vector<Vertex>& ver_mas) - функция переводит автомат в следующие состояние, ind - номер вершины в боре из которой ищем путь, chr - символ по по котором ищем переход в следующее состояние, ver_mas - массив вершин бора. Возвращает номер вершины в которую в которую нашли путь.

int get_suf(int ind, vector<Vertex>& ver_mas) - функция вычисляющая суф. ссылку для вершины ind. ver_mas - массив вершин бора. Возвращает номер вершины на которую указывает суф. ссылка.

step1

void search_pat(string& text, vector<Vertex>& ver_mas, vector<pair<int, int>>& res, vector<string>& pat_mas) - функция выполняющая поиск шаблонов и записывающая результаты в массив res. text - исходный текст, ver_mas - массив вершин бора, res - массив пар индексов вхождения и соответствующих шаблонов, pat_mas - массив паттернов.

step2

void search_pat(string& text, vector<Vertex>& ver_mas, vector<int>& res, vector<int>& pattern_offset_mas, int pat_length, vector<string>& pat_mas) -

функция выполняющая поиск подпаттернов и самого паттерна в исходном тексте.text - исходный текст, ver_mas - массив вершин бора, res - массив совпавших подпатернов, pat_mas - массив паттернов, pattern_offset_mas - массив индексов вхождения подпаттернов в исходном тексте, pat_length - длина паттерна .

void add_to_bor(string& pat_string, vector<Vertex>& ver_mas, int& count) - функция добавления строки шаблона в бор. pat_string - строка шаблона, ver_mas - массив вершин бора, count - счетчик паттернов.

step1

void read_pat(vector<Vertex>& ver_mas, vector<string>& pat_mas) - функция считывающая паттерны и вызывающая функцию add_to_bor для добавления их в бор. ver_mas - массив вершин бора, pat_mas - массив паттернов.

step2

void read_pat(vector<Vertex>& ver_mas, char& joker, vector<int>& pattern_offset_mas, int& pat_length, vector<string>& pat_mas) - функция считывающая паттерн и символ джокера. ver_mas - массив вершин бора, pat_mas - массив подпаттернов, joker - символ джокера, pattern_offset_mas - массив индексов вхождения подпаттернов в исходном тексте, pat_length - длина паттерна .

void split_pattern(string pattern, char joker, vector<string>& pat_mas, vector<int>& pattern_offset_mas) - функция разбивающая паттерн на подпаттерны по символу джокера. pattern - паттерн, joker - символ джокера, pattern_offset_mas - массив индексов вхождения подпаттернов в исходном тексте, pat_mas - массив подпаттернов

int max_edge(vector<Vertex> ver_mas) - функция для поиска максимального количества дуг исходящих из одной вершины. ver_mas - массив вершин.

step1

void print_result(vector<pair<int, int>>& res, vector<string>& pat_mas, string& text, string& cut_text) - функция печатающая результат и вырезающая паттерны из исходного текста. text - исходный текст, cut_text - обрезанный текст, res - массив пар индексов вхождения и соответствующих шаблонов, pat_mas - массив паттернов. Выводит индекс вхождения в исходном тексте и номер паттерна.

step2

void print_result(vector<int>& res, int pat_counter, string& cut_text, int pat_length, string& text) - функция печатающая результат и вырезающая паттерны из исходного текста. text - исходный текст, cut_text - обрезанный текст, res - массив совпадений подпаттернов, pat_counter - количество подпаттернов. Выводит индексы вхождения паттера в исходный текст.

void print_auto(vector <Vertex> ver_mas) - функция выводящая автомат. ver_mas - массив вершин.

Оценка сложности по памяти и времени.

Сложность по времени O((C+A)*log(B) + D), A - сумма длин всех паттернов, B - мощность алфавита, C - длинна исходного текста, D - число совпадений всех паттернов с текстом.

Сложность по памяти O(P + C), P - число вершин в боре, C - длина текста.

Тестирование.

Таблица 1 результаты работы программы step1.cpp.

Входные данные	Вывод программы
ACGACGBH 2 AC CG	1 1 2 2 4 1 5 2 7 3 Обрезанная строка: H Максимальное количество дуг исходящих из одной вершины: 3
cfthnikj 2 thn hn	3 1 4 2 Обрезанная строка: cfikj Максимальное количество дуг исходящих из одной вершины: 2
TTCTAG 3 TT C TAG	1 1 3 2 4 3 Обрезанная строка: 2 Максимальное количество дуг исходящих из одной вершины:

Таблица 2 результаты работы программы step1.cpp.

Входные данные	Вывод программы

ACACACACACCCA A$A $	1 3 5 7 Обрезанная строка: CCCA Максимальное количество дуг исходящих из одной вершины: 1
ACHBGJHACYBG AC#BG #	1 8 Обрезанная строка: JH Максимальное количество дуг исходящих из одной вершины: 2
VBNUIK B$$I $	2 Обрезанная строка: VK Максимальное количество дуг исходящих из одной вершины: 2
ABVGAJHHABN A** *	1 5 9 Обрезанная строка: GH Максимальное количество дуг исходящих из одной вершины: 1
BBBG !B!! !	1 Обрезанная строка: Максимальное количество дуг исходящих из одной вершины: 1

Выводы.

В ходе выполнения данной лабораторной работы был реализован алгоритм Ахо-Корасик на языке программирования C++.

Приложение А. Исходный код программы.

Stepik 1

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

using namespace std;

//#define OUTPUTCOM

// структура данных орисывающая вершину
struct Vertex {
	// контейнер прямых переходов
	map<char, int> next;
	// контейнер всех переходов включая суфф ссылки
	map<char, int> go;
	// предыдущая вершина
	char prev_char;
	// суф ссылка
	int prev_vert;
	// символ по которому пришли в вершину
	int suffix;
	// номер паттерна
	int number;
	// является ли терминальной
	bool isTerminal = false;
};

int next_state(int ind, char chr, vector<Vertex>& ver_mas);

// функция вычисляющая суффикс ссылку
int get_suf(int ind, vector<Vertex>& ver_mas) {
	// Если суф ссылка не была найдена
	if (ver_mas[ind].suffix == -1) {
	// Если корень или потомок корня
	if (ind == 0 || ver_mas[ind].prev_vert == 0) {
		ver_mas[ind].suffix = 0;
	}
	else {
		// Рекурсивный поиск суфф ссылки
		#ifdef OUTPUTCOM
		cout << "\tИщем суф. ссылку из суффикса родительской вершины "  <<  ver_mas[ind].prev_vert  <<  "  через дугу с символом "<< ver_mas[ind].prev_char << ".\n";
		#endif
```

```
          ver_mas[ind].suffix                                          =
next_state(get_suf(ver_mas[ind].prev_vert,                        ver_mas),
ver_mas[ind].prev_char, ver_mas);
     }
     }
     #ifdef OUTPUTCOM
     cout << "\tНайдена суф. ссылка из вершины " << ind << " равная "
<< ver_mas[ind].suffix << ".\n\n";
     #endif
     return ver_mas[ind].suffix;
}

// функция перехода в след состояние автомата
int next_state(int ind, char chr, vector<Vertex>& ver_mas) {
     #ifdef OUTPUTCOM
     cout << "\tИщем путь из вершины " << ind << " по дуге с символом "
<< chr << ".\n";
     #endif
     // Если для данного символа нет перехода в массиве go
     if (ver_mas[ind].go.find(chr) == ver_mas[ind].go.end()) {
     // Если найден прямой переход по символу
     if (ver_mas[ind].next.find(chr) != ver_mas[ind].next.end()) {
          ver_mas[ind].go[chr] = ver_mas[ind].next[chr];
     }
     else {
          if (ind == 0) {
               #ifdef OUTPUTCOM
               cout << "\tЭто корень!\n";
               #endif
               // суф. ссылке присваеваем значение 0
               ver_mas[ind].go[chr] = 0;
          }
          else {
               // переходим по суф ссылке
               #ifdef OUTPUTCOM
               cout << "\tНет прямого пути, поиск пути от суф. ссылки
этой вершины через дугу с символом " << chr << ".\n";
               #endif
               ver_mas[ind].go[chr] = next_state(get_suf(ind, ver_mas),
chr, ver_mas);
          }
     }
     }
     #ifdef OUTPUTCOM
     cout << "\tНайден путь из вершины "<< ind << " в вершину " <<
ver_mas[ind].go[chr] <<".\n";
     #endif
     return ver_mas[ind].go[chr];
}

// функция для поиска шаблонов
```

```
void search_pat(string& text, vector<Vertex>& ver_mas, vector<pair<int,
int>>& res, vector<string>& pat_mas) {
	#ifdef OUTPUTCOM
	cout << "Начинаем поиск !\n";
	#endif
	int curr = 0;
	// перебираем все символы в тексте
	for (int i = 0; i < text.size(); i++) {
	#ifdef OUTPUTCOM
	cout << "Текущий символ текста: " << text[i] <<  "\n";
	cout << "Текущая вершина: " << curr << "\n";
	#endif
	// ищим следующие состояние автомата
	curr = next_state(curr, text[i], ver_mas);
	#ifdef OUTPUTCOM
	cout << "Перешли в вершину: " << curr << "\n";
	cout << "Проверяем является ли она терминальной или ссылается на
терминальную:\n";
	#endif
	// заодно получаем суф ссылки
	for (int tmp = curr; tmp != 0; tmp = get_suf(tmp, ver_mas)) {
		#ifdef OUTPUTCOM
		cout << "\tПроверяем вершину : " << tmp <<"\n";
		#endif
		// если вершина терминальная
		if (ver_mas[tmp].isTerminal) {
			// добавляем ответ
			res.push_back(make_pair(i + 2 -
pat_mas[ver_mas[tmp].number - 1].size(), ver_mas[tmp].number));
			#ifdef OUTPUTCOM
			cout << "\tТерминальная вершина, позиия в тексте = " <<
i + 2 -  pat_mas[ver_mas[tmp].number - 1].size() << " паттерн = " <<
pat_mas[ver_mas[tmp].number - 1] << "\n";
			#endif
		}
		else {
			#ifdef OUTPUTCOM
			cout << "\tНе является терминальной.\n\n";
			#endif
		}
	}
	#ifdef OUTPUTCOM
	cout << "Текущая суф. ссылка указывает на корень, считываем
следующий символ текста." << "\n\n";
	#endif
	}
}

// функция добалвения строки в бор
void add_to_bor(string& pat_string, vector<Vertex>& ver_mas, int& count)
{
```

```
	#ifdef OUTPUTCOM
	cout << "Добавляем строку " << pat_string << " в бор.\n";
	#endif
	int current = 0;
	// обходим все символы паттерна
	for (int i = 0; i < pat_string.size(); i++) {
	#ifdef OUTPUTCOM
	cout << "\tТекущий символ паттерна: " << pat_string[i] << ".\n";
	cout << "\tТекущая вершина: " << current << ".\n";
	#endif
	// если прямой переход по символу не возможен
	if (ver_mas[current].next.find(pat_string[i]) == ver_mas[current].next.end()) {
		#ifdef OUTPUTCOM
		cout << "\tПуть через дугу с символом " << pat_string[i] << " не был найден. Добавляем новую вершину под номером " << ver_mas.size() << ".\n\n";
		#endif
		Vertex ver;
		ver.suffix = -1;
		ver.prev_vert = current;
		ver.prev_char = pat_string[i];
		ver_mas.push_back(ver);
		ver_mas[current].next[pat_string[i]] = ver_mas.size() - 1;
	}
	// если переход по символу есть
	else{
		#ifdef OUTPUTCOM
		cout << "\tПуть с таким ребром уже существует.\n\n";
		#endif
	}
	// переходим в следующую вершину
	current = ver_mas[current].next[pat_string[i]];
	}
	// выводим номер паттерна
	#ifdef OUTPUTCOM
	cout << "Паттерн зарегистрированн под номером " << count + 1 << ".\n";
	// выводим терминальную вершину и ее глубину в боре
	cout << "Терминальная вершина паттерна " << current << ".\n\n";
	#endif
	// записываем номер паттерна
	ver_mas[current].number = ++count;
	ver_mas[current].isTerminal = true;
}

// функция считывающая паттерны
void read_pat(vector<Vertex>& ver_mas, vector<string>& pat_mas) {
	Vertex root;
	root.prev_vert = -1;
	root.suffix = -1;
```

```
	// добавляем корень
	ver_mas.push_back(root);
	int count = 0;
	// количество паттернов
	int pat_counter;
	#ifdef OUTPUTCOM
	cout << "Введите количество паттернов:\n";
	#endif
	cin >> pat_counter;
	// считываем паттерны
	for (int i = 0; i < pat_counter; i++) {
	#ifdef OUTPUTCOM
	cout << "Введите паттерн:\n";
	#endif
	string pattern;
	cin >> pattern;
	// добавляем в массив паттернов
	pat_mas.push_back(pattern);
	// долабляем строку в бор
	add_to_bor(pattern, ver_mas, count);
	}
}

// функция поиска максимального количества дуг исходящих из вершины
int max_edge(vector<Vertex> ver_mas) {
	int max = ver_mas[0].next.size();
	// перебираем все вершины в массиве
	for(auto i : ver_mas) {
	if (i.next.size() > max)
		max = i.next.size();
	}
	return max;
}


// функция печатающая результат и обрезающая исходный текст
void print_result(vector<pair<int, int>>& res, vector<string>& pat_mas,
string& text, string& cut_text) {
	vector<bool> what_cut(text.size());
	// перебираем массив ответов
	for(auto i: res ) {
	cout << i.first << " " << i.second << '\n';
	// перебираем строку паттерна
	for (int j = 0; j < pat_mas[i.second - 1].size(); j++) {
		what_cut[i.first - 1 + j] = true;
	}
	}
	// перебираем исходный текст
	for (int i = 0; i < text.size(); i++) {
	if (!what_cut[i]) {
		cut_text.push_back(text[i]);
```

```
	}
	}
}

// выводим автомат
void print_auto(vector <Vertex> ver_mas) {
	cout << "Автомат:\n";
	// перебираем все вершины по порядку
	for (int i = 0; i < ver_mas.size(); i++) {
	cout << "Дуги изходящие из вершины " << i << ":\n";
	for (auto a: ver_mas[i].next) {
		if(a.second > 0) {
			cout <<   " ---" << a.first << "--->  " << a.second <<
"\n";
		}
	}
	// если нет потомков
	if(size(ver_mas[i].next) == 0) {
		cout << "Лист бора!" << '\n';
	}
	}
}

bool comp( const pair<int, int>& p1, const pair<int, int>& p2 ) {
	return  (p1.first  <  p2.first)  ||  ((p1.first  ==  p2.first)  &&
(p1.second < p2.second));
}

int main() {
	string text;
	string cut_text;
	#ifdef OUTPUTCOM
	cout << "Введите текст:\n";
	#endif
	// текст в котором будет происходить поиск
	cin >> text;
	// массив вершин
	vector<Vertex> ver_mas;
	// массив паттернов
	vector<string> pat_mas;
	// массив пар позиция в тексте - номер шаблона
	vector<pair<int, int>> res;
	// считывем паттерны
	read_pat(ver_mas, pat_mas);
	// запускаем поиск
	search_pat(text, ver_mas, res, pat_mas);
	// сортируем для вывода по порядку
	sort(res.begin(), res.end(), comp);
	// выводим результат
	print_result(res, pat_mas, text, cut_text);
	// выводим обрезанную строку
```

```
	#ifdef OUTPUTCOM
	cout << "\nОбрезанная строка: " << cut_text << "\n";
	// выводим макс кол-во дуг
	cout << "Максимальное количество дуг исходящих из одной вершины: "
<< max_edge(ver_mas) << "\n\n";
	#endif
	// выводим автомат
	#ifdef OUTPUTCOM
	print_auto(ver_mas);
	#endif
	return 0;
}


Stepik 2

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

using namespace std;

#define OUTPUTCOM

// структура данных орисывающая вершину
struct Vertex {
	// контейнер прямых переходов
	map<char, int> next;
	// контейнер всех переходов включая суфф ссылки
	map<char, int> go;
	// символ по которому пришли в вершину
	char prev_char;
	// предыдущая вершина
	int prev_vert;
	// суф ссылка
	int suffix;
	// массив для хранения номеров паттернов
	vector<int> number;
	// является ли терминальной
	bool isTerminal = false;
};

int next_state(int index, char symb, vector<Vertex>& ver_mas);

// функция добалвения строки в бор
void add_to_bor(string& pat_string, vector<Vertex>& ver_mas, int& count)
{
	#ifdef OUTPUTCOM
	cout << "Добавляем строку " << pat_string << " в бор.\n";
	#endif
```

```
	int current = 0;
	// обходим все символы паттерна
	for (int i = 0; i < pat_string.size(); i++) {
	#ifdef OUTPUTCOM
	cout << "\tТекущий символ паттерна: " << pat_string[i] << ".\n";
	cout << "\tТекущая вершина: " << current << ".\n";
	#endif
	// если прямой переход по символу не возможен
	if (ver_mas[current].next.find(pat_string[i]) == ver_mas[current].next.end()) {
		#ifdef OUTPUTCOM
		cout << "\tПуть через дугу с символом " << pat_string[i] << " не был найден. Добавляем новую вершину под номером " << ver_mas.size() << ".\n\n";
		#endif
		Vertex ver;
		ver.suffix = -1;
		ver.prev_vert = current;
		ver.prev_char = pat_string[i];
		ver_mas.push_back(ver);
		ver_mas[current].next[pat_string[i]] = ver_mas.size() - 1;
	}
	// если переход по символу есть
	else{
		#ifdef OUTPUTCOM
		cout << "\tПуть с таким ребром уже существует.\n\n";
		#endif
	}
	// переходим в следующую вершину
	current = ver_mas[current].next[pat_string[i]];
	}
	// выводим номер паттерна
	#ifdef OUTPUTCOM
	cout << "Паттерн зарегистрированн под номером " << count + 1 << ".\n";
	// выводим терминальную вершину и ее глубину в боре
	cout << "Терминальная вершина паттерна " << current << ".\n\n";
	#endif
	// записываем номер паттерна
	ver_mas[current].number.push_back(++count);
	ver_mas[current].isTerminal = true;
}

// функция вычисляющая суффикс ссылку
int get_suf(int ind, vector<Vertex>& ver_mas) {
	// Если суф ссылка не была найдена
	if (ver_mas[ind].suffix == -1) {
	// Если корень или потомок корня
	if (ind == 0 || ver_mas[ind].prev_vert == 0) {
		ver_mas[ind].suffix = 0;
	}
```

```
      else {
            // Рекурсивный поиск суфф ссылки
            #ifdef OUTPUTCOM
            cout << "\tИщем суф. ссылку из суффикса родительской вершины
"   <<   ver_mas[ind].prev_vert   <<   "   через   дугу   с   символом   "<<
ver_mas[ind].prev_char << ".\n";
            #endif
            ver_mas[ind].suffix                                              =
next_state(get_suf(ver_mas[ind].prev_vert,                          ver_mas),
ver_mas[ind].prev_char, ver_mas);
      }
      }
      #ifdef OUTPUTCOM
      cout << "\tНайдена суф. ссылка из вершины " << ind << " равная "
<< ver_mas[ind].suffix << ".\n\n";
      #endif
      return ver_mas[ind].suffix;
}

// функция перехода в след состояние автомата
int next_state(int ind, char chr, vector<Vertex>& ver_mas) {
      #ifdef OUTPUTCOM
      cout << "\tИщем путь из вершины " << ind << " по дуге с символом "
<< chr << ".\n";
      #endif
      // Если для данного символа нет перехода в массиве go
      if (ver_mas[ind].go.find(chr) == ver_mas[ind].go.end()) {
      // Если найден прямой переход по символу
      if (ver_mas[ind].next.find(chr) != ver_mas[ind].next.end()) {
            ver_mas[ind].go[chr] = ver_mas[ind].next[chr];
      }
      else {
            if (ind == 0) {
                  #ifdef OUTPUTCOM
                  cout << "\tЭто корень!\n";
                  #endif
                  // суф. ссылке присваеваем значение 0
                  ver_mas[ind].go[chr] = 0;
            }
            else {
                  // переходим по суф ссылке
                  #ifdef OUTPUTCOM
                  cout << "\tНет прямого пути, поиск пути от суф. ссылки
этой вершины через дугу с символом " << chr << ".\n";
                  #endif
                  ver_mas[ind].go[chr] = next_state(get_suf(ind, ver_mas),
chr, ver_mas);
            }
      }
      }
      #ifdef OUTPUTCOM
```

```
	cout << "\tНайден путь из вершины "<< ind << " в вершину " << ver_mas[ind].go[chr] <<".\n";
	#endif
	return ver_mas[ind].go[chr];
}

void search_pat(string& text, vector<Vertex>& ver_mas, vector<int>& res, vector<int>& pattern_offset_mas, int pat_length, vector<string>& pat_mas)
{
	#ifdef OUTPUTCOM
	cout << "Начинаем поиск !\n";
	#endif
	int curr = 0;
	// обходим весь текст
	for (int i = 0; i < text.size(); i++) {
	#ifdef OUTPUTCOM
	cout << "Текущий символ текста: " << text[i] <<  "\n";
	cout << "Текущая вершина: " << curr << "\n";
	#endif
	// выполняем переход в автомате
	curr = next_state(curr, text[i], ver_mas);
	#ifdef OUTPUTCOM
	cout << "Перешли в вершину: " << curr << "\n";
	cout << "Проверяем является ли она терминальной или ссылается на терминальную:\n";
	#endif
	// заодно получаем суф ссылки
	for (int tmp = curr; tmp != 0; tmp = get_suf(tmp, ver_mas)) {
		#ifdef OUTPUTCOM
		cout << "\tПроверяем вершину : " << tmp <<"\n";
		#endif
		// если вершина терминальная
		if (ver_mas[tmp].isTerminal) {
			// перебираем все паттерны
			for (int j = 0; j < ver_mas[tmp].number.size(); j++) {
			// если находится в промежутке от 0 до размера текста - размер паттерна
			if ((i + 1 - pattern_offset_mas[ver_mas[tmp].number[j] - 1] - pat_mas[ver_mas[tmp].number[j] - 1].size() >= 0) && (i + 1 - pattern_offset_mas[ver_mas[tmp].number[j] - 1] - pat_mas[ver_mas[tmp].number[j] - 1].size() <= text.size() - pat_length)) {
				res[i + 1 - pattern_offset_mas[ver_mas[tmp].number[j] - 1] - pat_mas[ver_mas[tmp].number[j] - 1].size()]++;
				#ifdef OUTPUTCOM
				cout << "\tТерминальная вершина," << " подпаттерн = " << pat_mas[ver_mas[tmp].number[j] - 1] << " Количество совпавщих подпаттернов " << res[i + 1 - pattern_offset_mas[ver_mas[tmp].number[j] - 1] - pat_mas[ver_mas[tmp].number[j] - 1].size()] <<
				" из " << pattern_offset_mas.size() << ".\n";
```

```
                        #endif
                        if          (res[i          +          1          -
pattern_offset_mas[ver_mas[tmp].number[j]          -          1]          -
pat_mas[ver_mas[tmp].number[j] - 1].size()] == pattern_offset_mas.size())
{
                              #ifdef OUTPUTCOM
                              cout << "\tНайдено вхождение паттерна !" <<
'\n';
                              #endif
                        }
                  }
                  }
            }
            // если вершина не терминалная
            else {
                  #ifdef OUTPUTCOM
                  cout << "\tНе является терминальной.\n\n";
                  #endif
            }
      }
      #ifdef OUTPUTCOM
      cout << "Текущая суф. ссылка указывает на корень, считываем
следующий символ текста." << "\n\n";
      #endif
      }
}

// функция печатающая результат и обрезающая исходный текст
void print_result(vector<int>& res, int pat_counter, string& cut_text,
int pat_length, string& text) {
      #ifdef OUTPUTCOM
      cout << "Индексы вхождения шаблона в текст\n";
      #endif
      vector<bool> what_cut(text.size());
      // перебираем массив ответов
      for (int i = 0; i < res.size(); i++) {
      // если совпали все подпатерны
      if (res[i] == pat_counter) {
            cout << i + 1 << "\n";
            for (int j = 0; j < pat_length; j++) {
                  what_cut[i + j] = true;
            }
      }
      }
      // перебираем исходный текст
      for (int i = 0; i < what_cut.size(); i++) {
      if (!what_cut[i])
            cut_text.push_back(text[i]);
      }
}
```

```
// функция разбивающая паттерн на подпаттерны
void split_pattern(string pattern, char joker, vector<string>& pat_mas,
vector<int>& pattern_offset_mas){
	#ifdef OUTPUTCOM
	cout << "Разобьем исходный паттерн по джокерам, на подпатерны.\n";
	#endif
	string buf = "";
	// перебираем все символы паттерна
	for (int i = 0; i < pattern.size(); i++) {
	// если встретили символ джокера
	if (pattern[i] == joker){
		// в буфере есть символы
		if (buf.size() > 0) {
			// добавляем в массив паттернов
			pat_mas.push_back(buf);
			#ifdef OUTPUTCOM
			cout << "\tНайден новый подпаттерн: " << buf << "\n";
			#endif
			// добавляем индекс вхождения
			pattern_offset_mas.push_back(i - buf.size());
			#ifdef OUTPUTCOM
			cout << "\tИндекс начала в исходном паттерне: " << i -
buf.size() << "\n";
			#endif
			buf = "";
		}
	}
	else {
		// добавляем символ в буфер
		buf.push_back(pattern[i]);
		if (i == pattern.size() - 1) {
			// добавляем в массив паттернов
			pat_mas.push_back(buf);
			#ifdef OUTPUTCOM
			cout << "\tНайден новый подпаттерн: " << buf << "\n";
			#endif
			// добавляем индекс вхождения
			pattern_offset_mas.push_back(i - buf.size() + 1);
			#ifdef OUTPUTCOM
			cout << "\tИндекс начала в исходном паттерне: " << i -
buf.size() + 1 << "\n";
			#endif
		}
	}
	}
}

// функция считывающая паттерн
void read_pat(vector<Vertex>& ver_mas, char& joker, vector<int>&
pattern_offset_mas, int& pat_length, vector<string>& pat_mas) {
	Vertex root;
```

```
	root.prev_vert = -1;
	root.suffix = -1;
	// добавляем корень
	ver_mas.push_back(root);
	int count = 0;
	#ifdef OUTPUTCOM
	cout << "Введите паттерн:\n";
	#endif
	string pattern;
	cin >> pattern;
	#ifdef OUTPUTCOM
	cout << "Введите символ джокера:\n";
	#endif
	cin >> joker;
	pat_length = pattern.size();
	// разбиваем паттерн на подпаттерны
	split_pattern(pattern, joker, pat_mas, pattern_offset_mas);
	// заносим в бор
	for (auto pattern : pat_mas) {
	add_to_bor(pattern, ver_mas, count);
	}

}

// функция поиска максимального количества дуг исходящих из вершины
int max_edge(vector<Vertex> ver_mas) {
	int max = ver_mas[0].next.size();
	// перебираем все вершины в массиве
	for(auto i : ver_mas) {
	if (i.next.size() > max)
		max = i.next.size();
	}
	return max;
}

// выводим автомат
void print_auto(vector <Vertex> ver_mas) {
	cout << "Автомат:\n";
	// перебираем все вершины по порядку
	for (int i = 0; i < ver_mas.size(); i++) {
	cout << "Дуги изходящие из вершины " << i << ":\n";
	for (auto a: ver_mas[i].next) {
		if(a.second > 0) {
			cout <<   " ---" << a.first << "--->  " << a.second <<
"\n";
		}
	}
	// если нет потомков
	if(ver_mas[i].next.size() == 0) {
		cout << "Лист бора!" << '\n';
	}
```

```
	}
}

int main() {
	string text;
	string cut_text;
	char joker;
	// массив вершин
	vector<Vertex> ver_mas;
	// массив паттернов
	vector<string> pat_mas;
	vector<int> res(110000);
	vector<int> pattern_offset_mas;
	int pat_length;

	#ifdef OUTPUTCOM
	cout << "Введите текст:\n";
	#endif
	cin >> text;

	for (int i = 0; i < 110000; i++){
	res[i] = 0;
	}
	// считываем паттерн
	read_pat(ver_mas, joker, pattern_offset_mas, pat_length, pat_mas);
	search_pat(text,  ver_mas,  res,  pattern_offset_mas,  pat_length,
pat_mas);
	// выводим результат
	print_result(res, pat_mas.size(), cut_text, pat_length, text);
	#ifdef OUTPUTCOM
	cout << "\nОбрезанная строка: " << cut_text << "\n";
	// выводим макс кол-во дуг
	cout << "Максимальное количество дуг исходящих из одной вершины: "
<< max_edge(ver_mas) << "\n\n";
	#endif
	#ifdef OUTPUTCOM
	// выводим автомат
	print_auto(ver_mas);
	#endif

	return 0;
}
```