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 _____ Колмыков В.Д.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Реализовать алгоритм Ахо-Корасик, на его основе построить алгоритмы для поиска вхождений шаблонов с «джокерами» в строку.

Вариант 1.

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

Задание.

1. 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 $P = \{p_1, \dots, p_n\}$ $1 <= |p_i| <= 75$

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

2. 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 Ахо-Корасик.

1. Создание бора.

Для создания бора все шаблоны поиска поочередно добавляются в бор. 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 строке.

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

С учетом реализации хранения вершин автомата в массиве, а переходов в словаре сложность алгоритма составляет $O((n + h) * log(k) + s)$, потребление памяти – $O(n)$, где n – длина всех слов(кол-во вершин), h – длина текста, в котором происходит поиск, k – размер алфавита, s – общая длина всех совпадений.

Описание функций и структур данных.

Структура Vertex.

```
struct Vertex {
      std::map<char, int> next;
      bool isLeaf = false;
      int prev;
      char prevChar;
      int suffix;
      std::map<char, int> go;
      int number;
      int deep;
};
```

Структура для хранения автомата/бора.

next – словарь переходов по ребрам бора.

isLeaf – флаг для определения является ли вершина листом.

prev – индекс вершины предка в боре.

prevChar – символ ребра связывающего вершину с предком в боре.

suffix – индекс вершины, в которую можно попасть из текущей по суффиксной ссылке.

go – словарь переходов для автомата.

number – номер шаблона (для листа).

deep – глубина вершины в боре.

Вершины хранятся в массиве структур Vertex.

Структура Result.

```
struct Result {
      int pos;
      int number;
};
```

pos – позиция вхождения шаблона.

number – номер шаблона.

Функция `void createBor(std::vector<Vertex>& vertexArr, int count)`.

Функция создания бора.

vertexArr – вектор для хранения вершин бора.

count – число шаблонов.

Функция `void findTempl(std::string& str, std::vector<Vertex>& vertexArr, std::vector<Result>& res).`

Функция поиска шаблонов в строке.

str – строка для поиска.

vertexArr – массив с бором, созданным для шаблонов поиска.

res – вектор для хранения результата.

Функция `int go(int curr, char c, std::vector<Vertex>& vertexArr).`

Функция получения вершины для перехода.

curr – вершина из которой совершается переход.

c – символ для перехода.

vertexArr – массив с вершинами.

Возвращает индекс вершины, в которую можно попасть из curr по символу c.

Функция `int getSuffix(int curr, std::vector<Vertex>& vertexArr).`

Функция получения вершины, доступной по суффиксной ссылке.

curr – вершина, для которой нужно найти суффиксную ссылку.

vertexArr – массив вершин автомата.

Функция `bool comp(Result res1, Result res2).`

Компаратор для сортировки результатов.

Функция `void printAuto(std::vector<Vertex>& vertexArr).`

Печать автомата/бора.

Тестирование.

Тестирование первой программы:

Ввод	Вывод
vladislav 3 vlad la adi	1 1 2 2 3 3 7 2
vlvlv 1 vlv	1 1 3 1
helloworld	Не найдено

1 privetmir	

Тестирование второй программы:

Ввод	Вывод
vladislav vl#d #	1
vladislav vl#d $	Не найдено
vlvlv v#v #	1 3

Тестирование индивидуальной программы:

Ввод	Вывод
vladislav vl#d $	1
vladislav vl#d #	Не найдено
vladislav vlad #	1

Выводы.

Был реализован алгоритм Ахо-Корасик, на его основе построены алгоритмы для поиска вхождений шаблонов с «джокерами» в строку.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММ

Поиск вхождений при помощи алгоритма Ахо-Корасик.

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>
#include <Windows.h>


struct Result {
	int pos;
	int number;
};


struct Vertex {
	std::map<char, int> next;
	bool isLeaf = false;
	int prev;
	char prevChar;
	int suffix;
	std::map<char, int> go;
	int number;
	int deep;
};


void createBor(std::vector<Vertex>& vertexArr, int count);
int getSuffix(int curr,  std::vector<Vertex>& vertexArr);
int go(int curr, char c, std::vector<Vertex>& vertexArr);
void findTempl(std::string& str, std::vector<Vertex>& vertexArr,
std::vector<Result>& res);
bool comp(Result res1, Result res2);
void printAuto(std::vector<Vertex>& vertexArr);


int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	std::string str;
	int templCount;
	std::cin >> str;
	std::cin >> templCount;
	std::vector<Vertex> vertexArr;
	createBor(vertexArr, templCount);
	printAuto(vertexArr);
	std::vector<Result> res;
	findTempl(str, vertexArr, res);
	printAuto(vertexArr);
	sort(res.begin(), res.end(), comp);
	for (auto result : res) {
		std::cout << result.pos << " " << result.number << std::endl;
	}
	system("pause");
}
```

```
void createBor(std::vector<Vertex>& vertexArr, int count) {//Создание бора
	Vertex root;
	root.prev = root.suffix = -1;
	vertexArr.push_back(root);
	for (int j = 1; j <= count; j++) { // Для каждого шаблона
		std::string str;
		std::cin >> str;
		std::cout << "Добавление шаблона " << str << ":" << std::endl;
		int curr = 0; //Изначально текущая вершина - корень
		for (size_t i = 0; i < str.length(); i++) {//Для всех символов
шаблона
			std::cout << "Рассматривается символ " << str[i] <<
std::endl;
			if (vertexArr[curr].next.find(str[i]) ==
vertexArr[curr].next.end()) { //Если из текущей вершины нет ребра с текущим
символом, добавляются новые вершина и ребро
				std::cout << "Ребра бора по этому символу из текущей
вершины еще нет. Создана новая вершина и добавлена в словарь переходов\n";
				Vertex ver;
				ver.suffix = -1;
				ver.prev = curr;
				ver.prevChar = str[i];
				vertexArr.push_back(ver);
				vertexArr[curr].next[str[i]] = vertexArr.size() - 1;
			}
			curr = vertexArr[curr].next[str[i]]; //Текущая вершина -
вершина, доступная по ребру с текущим символом
			std::cout << "Совершен переход по символу " << str[i] <<
std::endl;
		}
		std::cout << "Текущая вершина - лист\n";
		vertexArr[curr].number = j; //Последняя добавленная при обработке
шаблона вершина - лист. На будущее запоминается глубина и номер шаблона,
которому этот лист принадлежит
		vertexArr[curr].isLeaf = true;
		vertexArr[curr].deep = str.length();
	}
}


int getSuffix(int curr, std::vector<Vertex>& vertexArr) {
	std::cout << "Определение вершины для перехода по суффиксной ссылке:\n";
	if (vertexArr[curr].suffix == -1) { //Если суффикс ссылка еще не
определена
		std::cout << "Суффикс ссылка еще не определена\n";
		if (curr == 0 || vertexArr[curr].prev == 0) { //Вершина начальная
или предок вершины - начальная, суффиксная ссылка ведена на начальную
			std::cout << "Суффиксная ссылка из вершины = 0 (вершина -
корень или предок вершины - корень)\n";
			vertexArr[curr].suffix = 0;
		}
		else {
			std::cout << "Поиск суффиксной ссылки путем попытки перехода
по символу из вершины предка:\n";
			vertexArr[curr].suffix = go(getSuffix(vertexArr[curr].prev,
vertexArr), vertexArr[curr].prevChar, vertexArr); //Иначе суффиксная ссылка
ведет в переход из вершины, полученной по суффиксная ссылке предка, по prev-
символу
		}
	}
	std::cout << "Вершина для перехода определена\n";
	return vertexArr[curr].suffix;
}
```

```
int go(int curr, char c, std::vector<Vertex>& vertexArr) {
	if (vertexArr[curr].go.find(c) == vertexArr[curr].go.end()) { //Если в словаре переходов нет текущего символа
		std::cout << "В словаре переходов пока нет перехода по символу " << c << std::endl;
		if (vertexArr[curr].next.find(c) != vertexArr[curr].next.end()) { //Если символ есть в словаре ребер бора, добавляем переход по ребру
			std::cout << "Существует ребро бора с символом " << c << ". Этот переход добавляется в словарь переходов\n";
			vertexArr[curr].go[c] = vertexArr[curr].next[c];
		}
		else {
			std::cout << "Не существует ребра бора с символом " << c << ". Попытка совершения перехода из вершины, доступной по суффиксной ссылке\n";
			vertexArr[curr].go[c] = curr == 0 ? 0 : go(getSuffix(curr, vertexArr), c, vertexArr); //Иначе переход по суфикс ссылке
		}
	}
	std::cout << "Совершается переход по символу " << c << std::endl;
	return vertexArr[curr].go[c];
}


void findTempl(std::string& str, std::vector<Vertex>& vertexArr, std::vector<Result>& res) { //Функция поиска вхождений шаблонов в строку
	std::cout << "Прогон текста по автомату:\n";
	int curr = 0; //Текущая вершина - корень бора
	std::cout << "Текущая вершина - корень бора\n";
	for (int i = 0; i < str.length(); i++) {//Для каждого символа в строке
		std::cout << "Совершение перехода по символу " << str[i] << ":\n";
		curr = go(curr, str[i], vertexArr); //Переход из текущей вершины по текущему символу
		std::cout << "Проверка на вхождение:\n";
		for (int tmp = curr; tmp != 0; tmp = getSuffix(tmp, vertexArr)) { // Обход автомата по суффикс ссылкам
			if (vertexArr[tmp].isLeaf) { //Если при обходе встретился лист, вхождение строки найдено
				std::cout << "Проверяемая вершина - лист. Вхождение найдено\n";
				Result result;
				result.number = vertexArr[tmp].number;
				result.pos = i + 2 - vertexArr[tmp].deep;
				res.push_back(result);
			}
		}
	}
}


bool comp(Result res1, Result res2) {
	if (res1.pos != res2.pos) {
		return res1.pos < res2.pos;
	}
	return res1.number < res2.number;
}


void printAuto(std::vector<Vertex>& vertexArr) {
	std::cout << "_____________________________________\n";
	for (int i = 0; i < vertexArr.size(); i++) {
		Vertex curr = vertexArr[i];
```

```
            std::cout << "Номер вершины " << i << ":\n";
            if (curr.isLeaf) {
                  std::cout << "Вершина - лист по шаблону номер " <<
curr.number << std::endl;
            }
            if (curr.prev != -1) {
                  std::cout << "Номер вершины предка " << curr.prev << ",
символ предка " << curr.prevChar << "\n";
            }
            if (curr.suffix != -1) {
                  std::cout << "Номер вершины, доступной по суффиксной ссылке
" << curr.suffix << std::endl;
            }
            if (!curr.isLeaf) {
                  std::cout << "Ребра бора, доступные из вершины:\n";
                  for (auto v : curr.next) {
                        std::cout << "Ребро " << v.first << ", номер вершины "
<< v.second << std::endl;
                  }
            }
            std::cout << "Переходы, доступные из вершины:\n";
            for (auto v : curr.go) {
                  std::cout << "Символ " << v.first << ", номер вершины " <<
v.second << std::endl;
            }

      }
      std::cout << "_____________________________________\n";
}
```

Поиск вхождения шаблона с джокерами:

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


struct Vertex {
      std::map<char, int> next;
      bool isLeaf = false;
      int prev;
      char prevChar;
      int suffix;
      std::map<char, int> go;
      int number;
      int deep;
      bool isNextJoker = false;
};


void createBor(std::vector<Vertex>& vertexArr, std::string str, char joker);
int getSuffix(int curr, std::vector<Vertex>& vertexArr, char joker);
int go(int curr, char c, std::vector<Vertex>& vertexArr, char joker);
void findTempl(std::string& str, std::vector<Vertex>& vertexArr, char joker);
void printAuto(std::vector<Vertex>& vertexArr);


int main() {
      std::string str;
      std::string templ;
      char joker;
```

```
	std::cin >> str >> templ >> joker;
	std::vector<Vertex> vertexArr;
	createBor(vertexArr, templ, joker);
	findTempl(str, vertexArr, joker);
	system("pause");
}


void createBor(std::vector<Vertex>& vertexArr, std::string str, char joker) {
	std::cout << "Добавление в бор шаблона " << str << ":" << std::endl;
	Vertex root;
	root.prev = root.suffix = -1;
	vertexArr.push_back(root);
	int curr = 0;
	std::cout << "Текущая вершина - корень бора\n";
	for (size_t i = 0; i < str.length(); i++) {
		std::cout << "Рассматривается символ " << str[i] << std::endl;
		if (vertexArr[curr].next.find(str[i]) ==
vertexArr[curr].next.end()) {
			std::cout << "Ребра бора по этому символу из текущей вершины
еще нет. Создана новая вершина и добавлена в словарь переходов\n";
			Vertex ver;
			ver.suffix = -1;
			ver.prev = curr;
			ver.prevChar = str[i];
			if (str[i] == joker) {
				vertexArr[curr].isNextJoker = true;
			}
			vertexArr.push_back(ver);
			vertexArr[curr].next[str[i]] = vertexArr.size() - 1;
		}
		curr = vertexArr[curr].next[str[i]];
		std::cout << "Совершен переход по символу " << str[i] <<
std::endl;
	}
	std::cout << "Текущая вершина - лист\n";
	vertexArr[curr].isLeaf = true;
	vertexArr[curr].deep = str.length();
}


int getSuffix(int curr, std::vector<Vertex>& vertexArr, char joker) {
	std::cout << "Определение вершины для перехода по суффиксной ссылке:\n";
	if (vertexArr[curr].suffix == -1) {
		std::cout << "Суффикс ссылка еще не определена\n";
		if (curr == 0 || vertexArr[curr].prev == 0) {
			std::cout << "Суффиксная ссылка из вершины = 0 (вершина -
корень или предок вершины - корень)\n";
			vertexArr[curr].suffix = 0;
		}
		else {
			std::cout << "Поиск суффиксной ссылки путем попытки перехода
по символу из вершины предка:\n";
			vertexArr[curr].suffix = go(getSuffix(vertexArr[curr].prev,
vertexArr, joker), vertexArr[curr].prevChar, vertexArr, joker);
		}
	}
	std::cout << "Вершина для перехода определена\n";
	return vertexArr[curr].suffix;
}


int go(int curr, char c, std::vector<Vertex>& vertexArr, char joker) {
```

```
	std::cout << "Определение вершины для перехода:\n";
	if (vertexArr[curr].go.find(c) == vertexArr[curr].go.end()) {
		std::cout << "Вершина для перехода еще не определена в словаре переходов\n";
		if (vertexArr[curr].isNextJoker) {
			std::cout << "Из вершины существует переход по символу джокеру\n";
			std::cout << "Вершина для перехода определена\n";
			return vertexArr[curr].next[joker];
		}
		if (vertexArr[curr].next.find(c) != vertexArr[curr].next.end()) {
			std::cout << "В боре существует переход по символу " << c << std::endl;
			vertexArr[curr].go[c] = vertexArr[curr].next[c];
		}
		else {
			std::cout << "Поиск перехода по с помощью суффиксной ссылки\n";
			vertexArr[curr].go[c] = curr == 0 ? 0 : go(getSuffix(curr, vertexArr, joker), c, vertexArr, joker);
		}
	}
	std::cout << "Вершина для перехода определена\n";
	return vertexArr[curr].go[c];
}


void findTempl(std::string& str, std::vector<Vertex>& vertexArr, char joker)
{
	std::cout << "Прогон текста по автомату:\n";
	int curr = 0;
	std::cout << "Текущая вершина - корень бора\n";
	for (int i = 0; i < str.length(); i++) {
		std::cout << "Совершение перехода по символу " << str[i] << ":\n";
		curr = go(curr, str[i], vertexArr, joker);
		if (vertexArr[curr].isLeaf) {
			std::cout << "Текущая вершина - лист, вхождение найдено\n";
			std::cout << i + 2 - vertexArr[curr].deep << std::endl;
		}
	}
}


void printAuto(std::vector<Vertex>& vertexArr) {
	std::cout << "_____\n";
	for (int i = 0; i < vertexArr.size(); i++) {
		Vertex curr = vertexArr[i];
		std::cout << "Номер вершины " << i << ":\n";
		if (curr.isLeaf) {
			std::cout << "Вершина - лист по шаблону номер " << curr.number << std::endl;
		}
		if (curr.prev != -1) {
			std::cout << "Номер вершины предка " << curr.prev << ", символ предка " << curr.prevChar << "\n";
		}
		if (curr.suffix != -1) {
			std::cout << "Номер вершины, доступной по суффиксной ссылке " << curr.suffix << std::endl;
		}
		if (!curr.isLeaf) {
			std::cout << "Ребра бора, доступные из вершины:\n";
			for (auto v : curr.next) {
```

```
                        std::cout << "Ребро " << v.first << ", номер вершины "
<< v.second << std::endl;
                  }
            }
            std::cout << "Переходы, доступные из вершины:\n";
            for (auto v : curr.go) {
                  std::cout << "Символ " << v.first << ", номер вершины " <<
v.second << std::endl;
            }

      }
      std::cout << "_____________________________________\n";
}
```

Программа для индивидуального варианта.

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>
#include <set>
#include <Windows.h>


struct Vertex {
      std::map<char, int> next;
      bool isLeaf = false;
      int prev;
      char prevChar;
      int suffix;
      std::map<char, int> go;
      int number;
      int deep;
      bool isNextJoker = false;
      char joker;
};


void createBor(std::vector<Vertex>& vertexArr, std::string str,
std::set<char> noJokers);
int getSuffix(int curr, std::vector<Vertex>& vertexArr);
int go(int curr, char c, std::vector<Vertex>& vertexArr);
void findTempl(std::string& str, std::vector<Vertex>& vertexArr);
void printAuto(std::vector<Vertex>& vertexArr);


int main() {
      SetConsoleCP(1251);
      SetConsoleOutputCP(1251);
      std::string str;
      std::string templ;
      char noJoker;
      std::cin >> str >> templ >> noJoker;
      std::vector<Vertex> vertexArr;
      std::set<char> noJokers;
      noJokers.insert(noJoker);
      for (auto c : str) {
            if (noJokers.find(c) == noJokers.end()) {
                  noJokers.insert(c);
            }
      }
      createBor(vertexArr, templ, noJokers);
```

```
	printAuto(vertexArr);
	findTempl(str, vertexArr);
	system("pause");
}


void createBor(std::vector<Vertex>& vertexArr, std::string str,
std::set<char> noJokers) {
	std::cout << "Добавление в бор шаблона " << str << ":" << std::endl;
	Vertex root;
	root.prev = root.suffix = -1;
	vertexArr.push_back(root);
	int curr = 0;
	std::cout << "Текущая вершина - корень бора\n";
	for (size_t i = 0; i < str.length(); i++) {
		std::cout << "Рассматривается символ " << str[i] << std::endl;
		if (vertexArr[curr].next.find(str[i]) ==
vertexArr[curr].next.end()) {
			std::cout << "Ребра бора по этому символу из текущей вершины
еще нет. Создана новая вершина и добавлена в словарь переходов\n";
			Vertex ver;
			ver.suffix = -1;
			ver.prev = curr;
			ver.prevChar = str[i];
			if (noJokers.find(str[i]) == noJokers.end()) {
				std::cout << "Данный переход может выполняться всегда,
так как " << str[i] << "может являться джокером\n";
				vertexArr[curr].isNextJoker = true;
				vertexArr[curr].joker = str[i];
			}
			vertexArr.push_back(ver);
			vertexArr[curr].next[str[i]] = vertexArr.size() - 1;
		}
		curr = vertexArr[curr].next[str[i]];
		std::cout << "Совершен переход по символу " << str[i] <<
std::endl;
	}
	std::cout << "Текущая вершина - лист\n";
	vertexArr[curr].isLeaf = true;
	vertexArr[curr].deep = str.length();
}


int getSuffix(int curr, std::vector<Vertex>& vertexArr) {
	std::cout << "Определение вершины для перехода по суффиксной ссылке:\n";
	if (vertexArr[curr].suffix == -1) {
		std::cout << "Суффикс ссылка еще не определена\n";
		if (curr == 0 || vertexArr[curr].prev == 0) {
			std::cout << "Суффиксная ссылка из вершины = 0 (вершина -
корень или предок вершины - корень)\n";
			vertexArr[curr].suffix = 0;
		}
		else {
			std::cout << "Поиск суффиксной ссылки путем попытки перехода
по символу из вершины предка:\n";
			vertexArr[curr].suffix = go(getSuffix(vertexArr[curr].prev,
vertexArr), vertexArr[curr].prevChar, vertexArr);
		}
	}
	std::cout << "Вершина для перехода определена\n";
	return vertexArr[curr].suffix;
}
```

```
int go(int curr, char c, std::vector<Vertex>& vertexArr) {
	if (vertexArr[curr].go.find(c) == vertexArr[curr].go.end()) {
		if (vertexArr[curr].isNextJoker) {
			return vertexArr[curr].next[vertexArr[curr].joker];
		}
		if (vertexArr[curr].next.find(c) != vertexArr[curr].next.end()) {
			vertexArr[curr].go[c] = vertexArr[curr].next[c];
		}
		else {
			vertexArr[curr].go[c] = curr == 0 ? 0 : go(getSuffix(curr,
vertexArr), c, vertexArr);
		}
	}
	return vertexArr[curr].go[c];
}


void findTempl(std::string& str, std::vector<Vertex>& vertexArr) {
	bool isFound = false;
	std::cout << "Прогон текста по автомату:\n";
	int curr = 0;
	std::cout << "Текущая вершина - корень бора\n";
	for (int i = 0; i < str.length(); i++) {
		std::cout << "Совершение перехода по символу " << str[i] << ":\n";
		curr = go(curr, str[i], vertexArr);
		if (vertexArr[curr].isLeaf) {
			std::cout << "Текущая вершина - лист, вхождение найдено\n";
			isFound = true;
			std::cout << i + 2 - vertexArr[curr].deep << std::endl;
		}
	}
	if (!isFound) {
		std::cout << "Вхождение не было найдено\n";
	}
}


void printAuto(std::vector<Vertex>& vertexArr) {
	std::cout << "_____________________________________\n";
	for (int i = 0; i < vertexArr.size(); i++) {
		Vertex curr = vertexArr[i];
		std::cout << "Номер вершины " << i << ":\n";
		if (curr.isLeaf) {
			std::cout << "Вершина - лист по шаблону номер " <<
curr.number << std::endl;
		}
		if (curr.prev != -1) {
			std::cout << "Номер вершины предка " << curr.prev << ",
символ предка " << curr.prevChar << "\n";
		}
		if (curr.suffix != -1) {
			std::cout << "Номер вершины, доступной по суффиксной ссылке
" << curr.suffix << std::endl;
		}
		if (!curr.isLeaf) {
			std::cout << "Ребра бора, доступные из вершины:\n";
			for (auto v : curr.next) {
				std::cout << "Ребро " << v.first << ", номер вершины "
<< v.second << std::endl;
			}
		}
		std::cout << "Переходы, доступные из вершины:\n";
```

```
            for (auto v : curr.go) {
                  std::cout << "Символ " << v.first << ", номер вершины " <<
v.second << std::endl;
            }

      }
      std::cout << "_____________________________________\n";
}
```