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: Ахо-Корасик

Студентка гр. 9382 _____ Голубева В.П.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Ознакомиться с алгоритмом Ахо-Корасик, научиться применять его для поиска набора образцов в тексте, а также для поиска шаблонной подстроки.

Задание 1

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

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

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

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

Выход:

Все вхождения образцов из P

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

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

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

Sample Input:

```
NTAG
3
TAGT
TAG
T
```

Sample Output:

```
2 2
2 3
```

Задание 2

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

В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу *P* 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*, $1 \le |T| \le 100000$)

Шаблон (*P*, $1 \le |P| \le 40$)

Символ джокера

Выход:

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

Номера должны выводиться в порядке возрастания.

Sample Input:

```
ACTANCA
A$$A$
```

```
$
```

Sample Output:

```
1
```

Вариант 1.

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

Описание алгоритма

Бор - дерево, образованное последовательным добавлением всех образцов посимвольно. При добавлении символа создаётся вершина, если соответствующий подобразец ещё не добавлялся; иначе просто осуществляется переход к ранее созданной вершине. Изначально бор состоит из корня.

Автомат - это бор, дополненный суффиксными и конечными ссылками. Это дополнение может происходить заранее или во время обработки текста.

Суффиксная ссылка из вершины A - это ссылка на вершину, соответствующую максимально длинному (под)образцу в автомате, являющемуся несобственным суффиксом (под)образца A. Для корня и его сыновей суффиксная ссылка указывает на корень.

Конечная ссылка из вершины A - это ссылка на вершину, соответствующую максимально длинному образцу, который может быть получен при выполнении нескольких переходов по суфф. ссылкам, начиная с A. Если образцов получить нельзя, то конечная ссылка пуста.

Для построения **суффиксных ссылок** следует применять правило:

- Суффиксная ссылка из корня или из сына корня ведёт в корень.

- Для вычисления суффиксной ссылки вершины x нужно:

1. Перейти к вершине-родителю.

2. Пройти по суффиксным ссылкам минимальное число раз — но не менее 1 раза — чтобы появился путь по ребру x*, или до попадания в корень.

3. Пройти по ребру x. Если мы в корне и ребрах из него нет, то остаться в корне.

Полученная вершина и есть искомая суффиксная ссылка.

* Имя вершины совпадает с именем ребра бора, ведущего в эту вершину.

Для построения **конечной ссылки** (сжатой суффиксной ссылки) нужно переходить по суффиксным ссылкам до тех, пока не попадём в терминальную вершину (т. е. соответствующую образцу). Полученная вершина будет искомой конечной ссылкой. Если попали в корень, то конечная ссылка пуста.

Алгоритм Ахо-Корасик

1. Построить бор из образцов.

2. Построить автомат из бора (можно выполнять по ходу обработки текста).

3. Перейти в корень бора.

4. Посимвольная обработка текста. Для каждого символа:

4.1. Совершить переход в автомате из текущей вершины по рассматриваемому символу:

4.1а. Если есть соответствующее ребро, то перейти по нему;

4.16. Если нет, то:

4.1.6.а. Если находимся в корне, то ничего не делать.

4.1.6.6. Если находимся не в корне, то перейти по суффиксной ссылке и перейти кп. 4.1.

4.2. Добавить в результат вхождение образца, если попали в конечную вершину.

4.3. Обойти цепочку конечных ссылок до конца, сохраняя результаты.

Алгоритм поиска шаблонной подстроки

1. Построить автомат Ахо-Корасик из образцов, полученных выделением максимальных безджокерных подстрок из шаблонной подстроки.

2. Для каждого образца записать смещение (смещения), по которому (по которым) образец находится в шаблонной строке.

3. Инициализировать массив, заполненный нулями, длиной, совпадающей с текстом.

4. Выполнить поиск по тексту с использованием автомата. При обнаружении образца инкрементировать ячейку массива по адресу, образованному разностью номера начального символа образца в тексте и смещения образца. Если у образца несколько смещений, то инкрементировать все соответствующие ячейки массива.

В результате шаблонная подстрока будет начинаться в тех местах текста, для которых соответствующая ячейка массива содержит количество образцов с учётом кратности.

Функции и структуры данных

Для вершины бора был реализован класс class BohrVertex с полями int parent - вершина родитель, char value - символ на ребре от parent к этой вершине, int next_ver[ALPHABET_LEN] - массив, где next_ver[i] - номер вершины, в которую мы перейдём по символу с номером i в алфавите, int numPattern - номер строки-шаблона для вершины next_ver[i], bool flag = false - является ли шаблоном данная вершина, int suffLink - суффиксная ссылка от этой вершины, int move[ALPHABET_LEN] - запоминание перехода автомата, а также BohrVertex(int, char) — конструктор класса, принимает вершину-

родителя и значения ребра при переходе из родителя, BohrVertex() - конструктор по умолчанию, ~BohrVertex() - деструктор класса.

Для хранения вершин и передвижению по ним был реализован класс бора - class Bohr. В нём std::vector <BohrVertex> bohr — вектор, хранит вершины бора, int get_move(int v, int edge) — перемещаемся по бору из вершины v по ребру edge — возвращает вершину, в которую мы перейдём, int get_suffix_link(int v) — принимает вершину, возвращаем вершину, в которую ведёт суффиксная ссылка из v, void res(std::vector <std::string>& patterns, int v, int i, std::string text) — принимает строку шаблонов, вершину, номер символа в строке-тексте и сам текст, формирует результат поиска шаблонов в тексте, также в классе Bohr() - конструктор, ~Bohr() - деструктор, void add_in_bohr(std::map<char, int> m, int index, std::string s) — добавляет образец — строку s в бор, void find_matches(std::vector <std::string> patterns, std::string s, std::map<char, int> m) — осуществляет поиск совпадений строке, принимает вектор шаблонов, строку-текст и словарь для сопоставления символа алфавита и его значения.

Для поиска в текст строки с джокером в классе Bohr также добавлены методы: std::vector <int> patterns(std::map<char, int> m, std::stringstream& string_pattern, char joker) — принимает словарь для сопоставления символа алфавита и его значения, строку и символ джокера, для разделения строки с джокером на подстроки для бора, а также void print_res(const std::vector<int>& array, int t_size, int length) — принимает вспомогательный массив длины текста, его размер, а также длину строки-шаблона, проверяет совпадения и выводит результат.

Оценка сложности по памяти и времени

Пусть T - длина текста, в которой выполняется поиск, n - суммарная длина всех образцов, s - размер алфавита, k - общая длина всех вхождений образцов в текст.

Автомат хранится как индексный массив, сложность по времени - O(ns+T+k), по памяти - O(ns).

Тестирование

Результаты тестирования программ можно посмотреть в приложениях А и Б.

Выводы.

Был изучен алгоритм Ахо-Корасик, реализована программа, осуществляющая поиск набора образцов в указанном тексте.

ПРИЛОЖЕНИЕ А
ТЕСТИРОВАНИЕ ПРОГРАММЫ 1

Входные данные	Выходные данные
NTAG 3 TAGT TAG TG	2 2
NTCACGCAC 3 TAG TC AC	2 2 4 3 8 3
NTCACGCAC 2 NTCA C	1 1 3 2 5 2 7 2 9 2
NTCACGCAC 1 A	4 1 8 1
ACACACACN 1 N	9 1

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ ПРОГРАММЫ 2

Входные данные	Выходные данные
ACTANKA A$$A$ $	1
ANANACAACA #NACA# #	3
GTA @T@ @	1
TACA AC %	2
NAGAT ^ ^	1 2 3 4 5

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5_1.cpp

```
#include <iostream>
#include <vector>
#include <cstring>
#include <map>
#include <algorithm>
#define ALPHABET_LEN 5
std::vector <std::pair<int, int>> output;

class BohrVertex{
public:
	int parent;//вершина родитель
	char value;
	int next_ver[ALPHABET_LEN]; //массив, где next_ver[i] - номер вершины, в которую мы перейдём по символу с номером i в алфавите
	int numPattern = 0; //номер строки-шаблона для вершины next_ver[i]
	bool flag = false; // является ли шаблоном
	int suffLink = -1; // суффиксная ссылка от этой вершины
	int move[ALPHABET_LEN];
	BohrVertex(int, char);
	BohrVertex() = default;
	~BohrVertex() = default;
};

class Bohr{
	std::vector <BohrVertex> bohr;//вершины бора
	int get_move(int v, int edge);
	int get_suffix_link(int v);
	void res(std::vector <std::string>& patterns, int v, int i, std::string text);
```

```
public:

	Bohr();
	~Bohr() = default;
	void add_in_bohr(std::map<char, int> m, int index, std::string
s);
	void find_matches(std::vector <std::string> patterns, std::string
s, std::map<char, int> m);

};

Bohr::Bohr(){
	bohr.push_back(BohrVertex(0,0));
}

BohrVertex::BohrVertex(int parent, char symbol){
	for (int i = 0; i < ALPHABET_LEN; i++){
		next_ver[i] = -1;
		move[i] = -1;
	}
	this->parent = parent;
	this->value = symbol;
}

// добавляем образец в бор
void Bohr::add_in_bohr(std::map<char, int> m, int index, std::string
s){
	int n = 0;
	int edge = 0;
	int len =  s.length();
	for (int i = 0; i < len; i++){
		char symb = s[i];
		edge = m[symb];//вычисляем номер символа
		if (bohr[n].next_ver[edge] == -1){    // нет ребра
			bohr.push_back(BohrVertex(n, edge));//добавляем ребро
```

```
                bohr[n].next_ver[edge] = bohr.size() - 1;
            }
            n = bohr[n].next_ver[edge];
        }
        bohr[n].flag = true;
        bohr[n].numPattern = index;
}

// получаем суффиксную ссылку
int Bohr::get_suffix_link(int v){
        if (bohr[v].suffLink == -1)
            if (v == 0 || bohr[v].parent == 0) // корень указывает на
себя
                bohr[v].suffLink = 0;
        else
            bohr[v].suffLink = get_move(get_suffix_link(bohr[v].parent),
bohr[v].value);
        return bohr[v].suffLink;
}

// перемещаемся по бору по ребру edge
int Bohr::get_move(int v, int edge){
        if (bohr[v].move[edge] == -1)//если из текущей вершины нельзя
переместиться
            if (bohr[v].next_ver[edge] != -1)//пробуем переместиться из
следующей вершины
                bohr[v].move[edge] = bohr[v].next_ver[edge];
            else
                if (v == 0)//если корень
                    bohr[v].move[edge] = 0;
                else
                    bohr[v].move[edge] = get_move(get_suffix_link(v),
edge);//если не корень перемещаемся по суффиксной ссылке
        return bohr[v].move[edge];
}
```

```
void Bohr::res(std::vector <std::string>& patterns, int v, int i,
std::string text){
	for(int u = v; u != 0; u = get_suffix_link(u)){
		if (bohr[u].flag)
			output.push_back(std::make_pair(i -
patterns[bohr[u].numPattern].length() + 1, bohr[u].numPattern + 1));
	}
}

//находим шаблоны в тексте
void Bohr::find_matches(std::vector <std::string> patterns,
std::string s, std::map<char, int> m){
	int u = 0;
	int edge;
	int len = s.length();
	for (int i = 0; i < len; i++){
		char symb = s[i];
		edge = m[symb];
		u = get_move(u, edge);//перешли из u по ребру edge
		res(patterns, u, i + 1, s);
	}
}

int main(){
	std::map<char, int> m { {'A', 0}, {'C', 1}, {'G', 2}, {'T', 3},
{'N', 4}}; // буква и её значение
	Bohr bohr;
	std::vector <std::string> patterns;
	std::string text;
	int number = 0;
	std::string pattern;
	std::cin >> text >> number;
	for (int i = 0; i < number; i++){
		std::cin >> pattern;
		bohr.add_in_bohr(m, i, pattern);
		patterns.push_back(pattern);
	}
```

```
	bohr.find_matches(patterns, text, m);
	sort(output.begin(), output.end());
	for (std::vector <std::pair<int, int>>::iterator it =
output.begin(); it!=output.end(); ++it)
		if (true)
			std::cout<< (*it).first << ' ' << (*it).second << "\
n";
	return 0;
}
```

Название файла: lab5_2.cpp

```
#include <iostream>
#include <vector>
#include <map>
#include <cstring>
#include <sstream>
#include <algorithm>
#define ALPHABET_LEN 5

class BohrVertex{
public:
	int parent;//вершина - родитель
	char value;
	int next_ver[ALPHABET_LEN];//массив, где next_ver[i] - номер
вершины, в которую мы перейдём по символу с номером i в алфавите
	std::vector <int> num_pattern;
	bool flag = false; // является ли шаблоном
	int suffix_link = -1; // суффиксная ссылка от этой вершины
	int move[ALPHABET_LEN];
	BohrVertex(int, char);
	BohrVertex() = default;
	~BohrVertex() = default;
```

```
};

class Bohr{
	std::vector <BohrVertex> bohr;//вершины бора
	std::vector < std::string > pattern;
	int get_move(int v, int edge);
	int get_suffix_link(int v);

public:
	Bohr();
	~Bohr() = default;
	void add_in_bohr(std::map<char, int> m, std::string s);
	void find_matches(std::map<char, int> m, std::string &s,
std::vector <int> & array, const std::vector <int> & len);
	void res(int v, int i, std::vector <int> &array, std::vector
<int> len);
	std::vector <int> patterns(std::map<char, int> m,
std::stringstream& string_pattern, char joker);
	void print_res(const std::vector<int>& array, int t_size, int
length);
};

Bohr::Bohr(){
	bohr.push_back(BohrVertex(0,0));
}

BohrVertex::BohrVertex(int parent, char symbol){
	for (int i = 0; i < ALPHABET_LEN; i++){
		next_ver[i] = -1;
		move[i] = -1;
	}
	this->parent = parent;
	this->value = symbol;
	num_pattern.resize(0);
}
```

```
// Добавление образца в бор
void Bohr::add_in_bohr(std::map<char, int> m, std::string s){
	int n = 0;
	int edge = 0;
	int len =  s.length();
	for (int i = 0; i < len; i++){
		char symb = s[i];
		edge = m[symb];//вычисляем номер символа
		if (bohr[n].next_ver[edge] == -1){    // нет ребра
			bohr.push_back(BohrVertex(n, edge));//добавляем ребро
			bohr[n].next_ver[edge] = bohr.size() - 1;
		}
		n = bohr[n].next_ver[edge];
	}
	bohr[n].flag = true;
	pattern.push_back(s);
	bohr[n].num_pattern.push_back(pattern.size() - 1);

}

// Получаем суффиксную ссылку
int Bohr::get_suffix_link(int v){
	if (bohr[v].suffix_link == -1)
		if (v == 0 || bohr[v].parent == 0) // корень указывает на
себя
			bohr[v].suffix_link = 0;
	else
		bohr[v].suffix_link =
get_move(get_suffix_link(bohr[v].parent), bohr[v].value);
	return bohr[v].suffix_link;
}

// перемещаемся по бору по ребру edge
int Bohr::get_move(int v, int edge){
	if (bohr[v].move[edge] == -1)//если из текущей вершины нельзя
переместиться
```

```
          if (bohr[v].next_ver[edge] != -1)//пробуем переместиться из
следующей вершины
                bohr[v].move[edge] = bohr[v].next_ver[edge];
          else
                if (v == 0)//если корень
                      bohr[v].move[edge] = 0;
                else
                      bohr[v].move[edge] = get_move(get_suffix_link(v),
edge);//если не корень перемещаемся по суффиксной ссылке
     return bohr[v].move[edge];
}

void Bohr::res(int v, int i, std::vector <int>& array, std::vector
<int> len){
      for(int u = v; u != 0; u = get_suffix_link(u))
          if (bohr[u].flag){
                for (const auto &j : bohr[u].num_pattern)
                      if ((i - len[j] < array.size()))
                            array[i - len[j]]++;
          }
}

std::vector <int> Bohr::patterns(std::map<char, int> m,
std::stringstream& string_pattern, char joker){
     std::vector < int > len;
     int length = 0;
     std::string buffer;
     while (getline(string_pattern, buffer, joker)){//читаем из строки
подобразцы по символу разделителю - джокеру
          if (buffer.size() > 0){
                length += buffer.size();
                len.push_back(length);
                add_in_bohr(m, buffer);
          }
          length++;
     }
     return len;
```

```
}

void Bohr::print_res(const std::vector<int>& array, int t_size, int
length){
	for (int i = 0; i < t_size; i++)
		if ((array[i] == pattern.size()) && (i + length <= t_size))
{//проверяем условие того, что значение в массиве равно длине шаблона
			std::cout << i + 1 << "\n";
		}

}

void Bohr::find_matches(std::map<char, int> m, std::string &s,
std::vector <int> & array, const std::vector <int> & len){
	int edge;
	int u = 0;
	int lenght = s.length();
	for (int i = 0; i < lenght; i++){
		char symb = s[i];
		edge = m[symb];
		u = get_move(u, edge);//перешли из u по ребру edge
		res(u, i + 1, array, len);
	}
}

int main(){
	std::map<char, int> m { {'A', 0}, {'C', 1}, {'G', 2}, {'T', 3},
{'N', 4}}; // буква и её значение
	Bohr bohr;
	std::string text, pattern_text;
	char joker;
	std::cin >> text >> pattern_text >> joker;
	std::stringstream str_stream(pattern_text);
	std::vector <int> len = bohr.patterns(m, str_stream, joker);
	std::vector <int> array(text.length(), 0);
	bohr.find_matches(m, text, array, len);
```

```
        bohr.print_res(array, text.size(), pattern_text.length());
    }
```