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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студент гр. 8304 Кирьянов Д.И.

Преподаватель Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

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

Задание.

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?ab??c? с джокером ?? встречается дважды в тексте xabvccbababcaxxabvccbababcax.

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}

Вариант 2.

Подсчитать количество вершин в автомате; вывести список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска.

Описание алгоритма.

По полученным образцам строим бор. Построение выполняется за время O(m), где m — суммарная длина строк.

Узлы бора можно понимать как состояния автомата, а корень как начальное состояние. Узлы бора, в которых заканчиваются строки, становятся терминальными. Мы можем понимать рёбра бора как переходы в автомате по соответствующей букве. Однако одними только рёбрами бора нельзя ограничиваться. Если мы пытаемся выполнить переход по какой-либо букве, а соответствующего ребра в боре нет, то мы тем не менее должны перейти в

какое-то состояние. Для этого нам нужны суффиксные ссылки.

При построении автомата может возникнуть такая ситуация, что ветвление есть не на каждом символе. Тогда можно использовать сжатые суффиксные ссылки, т.е. ближайшее допускающее состояние (терминал) перехода по суффиксным ссылкам.

По очереди просматриваем символы текста. Для очередного символа c переходим из текущего состояния u в состояние, которое вернёт функция δ(u,c). Оказавшись в новом состоянии, отмечаем по сжатым суффиксным ссылкам строки, которые нам встретились и их позицию (если требуется). Если новое состояние является терминалом, то соответствующие ему строки тоже отмечаем.

Для того чтобы найти все вхождения в текст заданного шаблона с масками Q, необходимо обнаружить вхождения в текст всех его безмасочных кусков. Пусть {Q1,…,Qk} — набор подстрок Q, разделенных масками, и пусть {l1,…,lk} — их стартовые позиции в Q. Например, шаблон abφφcφ содержит две подстроки без масок ab и c и их стартовые позиции соответственно 1 и 5. Для алгоритма понадобится массив C. C[i] — количество встретившихся в тексте безмасочных подстрок шаблона, который начинается в тексте на позиции i. Тогда появление подстроки Qi в тексте на позиции j будет означать возможное появление шаблона на позиции j−li+1.

1. Используя алгоритм Ахо-Корасик, находим безмасочные подстроки шаблона Q: когда находим Qi в тексте T на позиции j, увеличиваем на единицу C[j−li+1].
2. Каждое i, для которого C[i]=k, является стартовой позицией появления шаблона Q в тексте.

Рассмотрим подстроку текста T[i…i+n−1]. Равенство C[i]=k будет означать, что подстроки текста T[i+l1…i+l1+|Q1|−1],T[i+l2…i+l2+|Q2|−1] и так далее будут равны соответственно безмасочным подстрокам шаблона {Q1,…,Qk}. Остальная часть шаблона является масками, поэтому шаблон входит в текст на позиции i.

Поиск подстрок заданного шаблона с помощью алгоритма Ахо-Корасик выполняется за время O(m+n+a), где

n — суммарная длина подстрок, то есть длина шаблона,

m — длина текста,

a — количество появлений подстрок шаблона.

Далее просто надо пробежаться по массиву C и просмотреть значения ячеек за время O(m).

Описание основных структур данных и функций.

```
struct bohr_vrtx {
	bohr_vrtx(int p, char c) {
		this->par = p;
		this->symb = c;
	}
	int next_vrtx[k] = { -1,-1,-1,-1,-1 };
	std::vector<int> pat_num;
	int suff_link = -1;
	int auto_move[k] = { -1,-1,-1,-1,-1 };
	int par = -1;
	int suff_flink=-1;
	bool flag = false;
	char symb = 0;
};
```

- структура для хранения вершины бора, где

next_vrtx[i] — номер вершины, в которую мы придем по символу с номером i в алфавите.

flag — флаг, проверяющий, является ли наша вершина исходной строкой.

pat_num — номера строк-образцов, обозначаемых этой вершиной.

suff_link — суффиксная ссылка.

auto_move — запоминание перехода автомата.

par — вершина-отец в дереве.

symb — символ на ребре от par к этой вершине.

suff_flink — сжатая суффиксная ссылка.

```
void add_str_to_bohr(std::string& s);
```
- функция добавления строки в бор. Добавляет переданный образец.

```
void find_all_pos(std::istream& input, std::ostream& output);
```
- функция поиска введенных образцов в веденном тексте.

```
void check(int v, int i, std::vector<int>& C, std::ostream& output);
```
- функция хождения по сжатым суффиксным ссылкам.

```
int get_suff_flink(int v);
```
- функция вычисления сжатой суффиксной ссылки.

```
int get_auto_move(int v, char ch);
```
- функция перехода по состояниям автомата. Принимает текущее состояние и символ, по которому осуществляется переход. Возвращает новое состояние.

```
int get_suff_link(int v);
```
- функция вычисления суффиксной ссылки.

Тестирование.

Таблица 1 – Результат работы.

Ввод	**Вывод**
ACTANCA A$$A$ $	1
AAAAAAAAAAAAAA A%AA%A %	1 2 3 4 5 6 7 8 9
ACGACTACNACG AC*AC *	1 4 7

Вывод.

В ходе выполнения работы, была написана программа, находящая точное вхождение образца с джокером и получены знания о такой структуре данных как бор.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

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

std::string input_way = "C:/Users/Danielka/source/repos/paa_lr5/input";//пути до файлов
std::string output_way = "C:/Users/Danielka/source/repos/paa_lr5/output";

std::map<char, int> alphabet;//алфавит
const int k = 5;//размер алфавита
std::vector<int> pos;//хранение позиций

struct bohr_vrtx {
	bohr_vrtx(int p, char c) {
		this->par = p;
		this->symb = c;
	}
	int next_vrtx[k] = { -1,-1,-1,-1,-1 };
	std::vector<int> pat_num;
	int suff_link = -1;
	int auto_move[k] = { -1,-1,-1,-1,-1 };
	int par = -1;
	int suff_flink=-1;
	bool flag = false;
	char symb = 0;
};

std::vector<bohr_vrtx> bohr = { bohr_vrtx(-1, 0) };
std::vector<std::string> pattern;

int get_auto_move(int v, char ch);

int get_suff_link(int v) {//вычисление суфф ссылки
	if (bohr[v].suff_link == -1)
		if (v == 0 || bohr[v].par == 0)
			bohr[v].suff_link = 0;
		else
			bohr[v].suff_link = get_auto_move(get_suff_link(bohr[v].par),
bohr[v].symb);
	return bohr[v].suff_link;
}

int get_auto_move(int v, char ch) {//функция перехода по состояниям автомата
	if (bohr[v].auto_move[ch] == -1)
		if (bohr[v].next_vrtx[ch] != -1)//если есть ребро
			bohr[v].auto_move[ch] = bohr[v].next_vrtx[ch];//то пройдем по нему
		else
			if (v == 0)
				bohr[v].auto_move[ch] = 0;
			else//иначе идем по суфф ссылке
				bohr[v].auto_move[ch] = get_auto_move(get_suff_link(v), ch);
	return bohr[v].auto_move[ch];
}

int get_suff_flink(int v) {//вычисление сжатой суфф ссылки
	if (bohr[v].suff_flink == -1) {
		int u = get_suff_link(v);
		if (u == 0)
			bohr[v].suff_flink = 0;
```

```
		else
			bohr[v].suff_flink = (bohr[u].flag) ? u : get_suff_flink(u);
	}
	return bohr[v].suff_flink;
}

void check(int v, int i, std::vector<int>& C, std::ostream& output) {//хождение по сжатым
суфф ссылкам
	for (int u = v; u != 0; u = get_suff_flink(u)) {
		if (bohr[u].flag) {
			for (int k : bohr[u].pat_num) {
				int j = i - pattern[k].size() + 1;
				int che = j - pos[k] + 1;
				if (che > 0 && che < C.size()) { //проверка выхода за рамки
					++C[che];//увеличиваем значение
				}
			}
		}
	}
}

void add_str_to_bohr(std::string& s) {//добавление строки в бор
	int num = 0;
	for (char i : s) {//проходим по всей строке
		char ch = alphabet[i];//получаем номер элемента в алфавите
		if (bohr[num].next_vrtx[ch] == -1) {//если нет ребра
			bohr.push_back(bohr_vrtx(num, ch));//добавляем его
			bohr[num].next_vrtx[ch] = bohr.size() - 1;
		}
		num = bohr[num].next_vrtx[ch];
	}
	bohr[num].flag = true;
	pattern.push_back(s);
	bohr[num].pat_num.push_back(pattern.size() - 1);
}

void find_all_pos(std::istream& input, std::ostream& output) {//функция поиска
	std::string text;
	std::string temp;
	std::string current;
	char joker = 0;
	input >> text >> temp >> joker;
	for (int i = 0; i < temp.size(); ++i) {//разделение образца на куски
		if (temp[i] == joker) {//если очередной элемент джокер
			if (current.empty())//и текущий кусок пустой
				continue;//то продолжаем цикл
			else {//если текущий кусок не пустой
				output << "Found substring: " << current << std::endl;
				add_str_to_bohr(current);//то добавляем его в бор
				pos.push_back(i - current.size());//и записываем индекс
				current = "";//обнуляем текущий кусок
				continue;
			}
		}
		else if (i == temp.size() - 1) {//если строка заканчивается не джокером
			current += temp[i];//то добавляем в бор
			output << "Found substring: " << current << std::endl;
			add_str_to_bohr(current);//весь оставшийся кусок
			pos.push_back(i - current.size() + 1);
		}
		current += temp[i];//если элемент не джокер, то увеличиваем текущий кусок
	}
	output << std::endl;
	int u = 0;
```

```
	std::vector<int> C(text.size());//вектор для хранения количества подстрок без
джокера
	output << "State changes:" << std::endl;
	for (int i = 0; i < text.length(); i++) {
		output << "From " << u;
		u = get_auto_move(u, alphabet[text[i]]);
		output << " to " << u << std::endl;
		check(u, i, C, output);
	}
	std::vector<int> inter;//вектор для кранения ответов
	for (int k = 0; k < C.size(); ++k)
		if (C[k] == pos.size()) {
			if (k + temp.size() - 1 <= text.size()) {
				output << std::endl;
				output << "Found result at " << k << " position" << std::endl
					<< "Result is \"" << text.substr(k - 1,
temp.size())<<"\"" << std::endl;
				for (int h : inter) {//если разница между индексами
					if ((k - h) < temp.size()) {//меньше чем длина шаблона
						output <<"\"" << text.substr(h - 1, temp.size())
<< "\""
							<< " from pos = " << h << " has
intersection with " << "\""
							<< text.substr(k - 1, temp.size()) << "\""
<< " from pos = "
							<< k << std::endl;//то значит есть
пересечение
					}
				}
				inter.push_back(k);//заносим ответ в вектор
			}
		}
	output << std::endl;
	output << "Coincidences was found at: ";//вывод индексов
	for (int res : inter)//совпадений
		output << res << " ";
	output << std::endl << std::endl;
	output << "Number of vertices: " << bohr.size() << std::endl;//количество вершин
}

int main(){
	alphabet['A'] = 0;//инициализация алфавита
	alphabet['C'] = 1;
	alphabet['G'] = 2;
	alphabet['T'] = 3;
	alphabet['N'] = 4;
	std::cout << "How do you want to input?" << std::endl << std::endl
		<< "Press 1 to input from console." << std::endl//выбор как считать
		<< "Press 2 to input from file." << std::endl;
	int menu = 0;
	while (menu != 1 && menu != 2) {//пока не введется нужная цифра
		std::cin >> menu;
		if (menu == 1) {//считывание с консоли
			int menu = 0;
			std::cout << std::endl << "How do you want to output?" << std::endl <<
std::endl
				<< "Press 1 to output by console." << std::endl//выбор как
вывести
				<< "Press 2 to output into file." << std::endl;
			while (menu != 1 && menu != 2) {//пока не введется нужная цифра
				std::cin >> menu;
				if (menu == 1) {//вывод на консоль
					find_all_pos(std::cin, std::cout);
				}
```

```
                        else if (menu == 2) {//вывод в файл
                                std::ofstream fileout;
                                fileout.open(output_way);
                                if (!fileout.is_open()) {
                                        std::cout << "Can't open file!\n";
                                }
                                find_all_pos(std::cin, fileout);
                        }
                        else {//если неверно введена цифра, выводится сообщение
                                std::cout << std::endl << "Wrong choice! Try again!" <<
std::endl;
                        }//а цикл продолжается
                }
        }
        else if (menu == 2) {//считывание из файла
                std::ifstream filein;
                filein.open(input_way);
                if (!filein.is_open()) {
                        std::cout << "Can't open file!" << std::endl;
                        return 0;
                }
                int menu = 0;
                std::cout << std::endl << "How do you want to output?" << std::endl <<
std::endl
                        << "Press 1 to output by console." << std::endl//выбор как
вывести
                        << "Press 2 to output into file." << std::endl;
                while (menu != 1 && menu != 2) {//пока не введется нужная цифра
                        std::cin >> menu;
                        if (menu == 1) {//вывод на консоль
                                find_all_pos(filein, std::cout);
                        }
                        else if (menu == 2) {//вывод в файл
                                std::ofstream fileout;
                                fileout.open(output_way);
                                if (!fileout.is_open()) {
                                        std::cout << "Can't open file!\n";
                                }
                                find_all_pos(filein, fileout);
                        }
                        else {//если неверно введена цифра, выводится сообщение
                                std::cout << std::endl << "Wrong choice! Try again!" <<
std::endl;
                        }//а цикл продолжается
                }
        }
        else {//если неверно введена цифра, выводится сообщение
                std::cout << std::endl << "Wrong choice! Try again!" << std::endl;
        }//а цикл продолжается
    }
    return 0;
}
```