**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по лабораторной работе №5**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Алгоритм Ахо-Корасик**

Студент гр.8304 \_\_\_\_\_\_\_\_\_\_\_\_\_ Мухин А. М.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Размочаева Н.В.

Санкт-Петербург

2020

**Цель работы.**

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

**Задание.**

Вариант 1

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

В шаблоне встречается специальный символ, именуемый джокером (wildcard), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу *P* необходимо найти все вхождения Р в текст Т.

Например, образец а*b*??с? 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*}.

**Описание алгоритма.**

Алгоритм строит конечный автомат, которому затем передаёт строку поиска. Автомат получает по очереди все символы строки и переходит по соответствующим рёбрам. Если автомат пришёл в конечное состояние, соответствующая строка словаря присутствует в строке поиска.

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 cc 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.

Вычислительная сложность алгоритма: O(2m + n + a), где n – длинна шаблона, m – длинна текста, a – кол-во появлений подстрок шаблона.

**Описание функций и структур данных.**

Структура для хранения вершины бора.

struct Vertex{

Vertex(int parent, char symbol, bool is\_leaf = false) :

parent(parent), symbol(symbol), is\_leaf(is\_leaf), serial\_number(-1), suff\_link(-1), suff\_flink(-1) {}

int neighbors[5] = {-1, -1, -1, -1, -1};

int serial\_number;

int suff\_link;

int auto\_move[5] = {-1, -1, -1, -1, -1};

int parent;

int suff\_flink;

bool is\_leaf;

char symbol;

std::vector<int> patterns\_num{};

};

Для структуры бора реализован соответствующий класс.

class Bor {

private:

std::vector<int> start\_positions;

std::map<char, int> alphabet;

std::vector<Vertex> repository;

std::vector<std::string> patterns;

std::string text;

std::string pattern;

char joker;

char unacceptable\_symbol;

std::vector<int> length\_vector;

void add\_pattern(const std::string& inserting\_string) noexcept;

[[nodiscard]] int get\_suffix\_link(const int vertex) noexcept;

[[nodiscard]] int transition(int vertex, char symbol) noexcept;

[[nodiscard]] int get\_fast\_suffix\_link(int vertex) noexcept;

void check(int vertex, int position\_in\_text) noexcept;

public:

Bor(std::string& text, std::string& pattern, char& joker, char& unacceptable\_symbol);

void run(std::ostream& output) noexcept {

std::string tmp\_substring;

for (int i = 0; i < pattern.size(); ++i) { // построение бора из слов,

if (pattern[i] == joker) { // не содержащих wildcard

if (!tmp\_substring.empty()) {

add\_pattern(tmp\_substring);

start\_positions.push\_back(i - tmp\_substring.size());

tmp\_substring.clear();

}

continue;

}

tmp\_substring += pattern[i];

}

int another\_vertex = 0; // проход по тексту, с фиксированием листов и

for (int i = 0; i < text.size(); ++i) { // заполнением массива C

another\_vertex = transition(another\_vertex, alphabet[text[i]]);

check(another\_vertex, i);

}

for (int k = 0; k < length\_vector.size(); ++k) {

if (length\_vector[k] == start\_positions.size()) {

bool is\_joker = false;

for (size\_t i = k; i < k + pattern.size() - 1; ++i) { // проход по всему шаблону и проверка на

if (pattern[i - k] == joker && text[i - 1] == unacceptable\_symbol){ // то, чтобы джокер в шаблоне не стоял

is\_joker = true; // на месте запрещённого символа в тексте

output << "Find unacceptable symbol!" << std::endl;

break;

}

}

if (!is\_joker && k + pattern.size() - 1 <= text.size())

output << "Find match at " << k << " position." << std::endl;

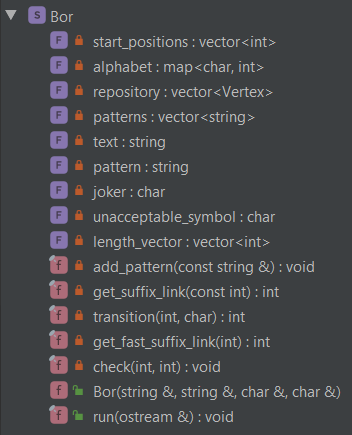
}

}

}

};

UML - диаграмма класса представлена ниже:



**Тестирование**

Таблица 1 – результаты тестирования

|  |  |
| --- | --- |
| **Input** | **Output** |
| NACGNTTACGGTCACNN  AC$$T$AC$$  $  C | 2 |
| NACGNTTACGGTCACNN  AC$$T$AC$$  $  A | 2  8 |
| ACTANCA  A$$A$  $  G | 1 |

**Выводы.**

В ходе выполнения работы, была написана программа, решающая задачу точного поиска для одного образца с джокером с индивидуализацией, чтобы месте джокера мог находится быть любой символ, за исключением заданного.