**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.**

UML диаграмма структуры данных для хранения вершины бора представлена на рис. 1.

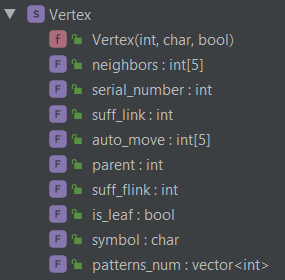


Рисунок 1. UML диаграмма структуры данных для хранения вершины бора.

UML - диаграмма класса бора представлена на рис. 2.

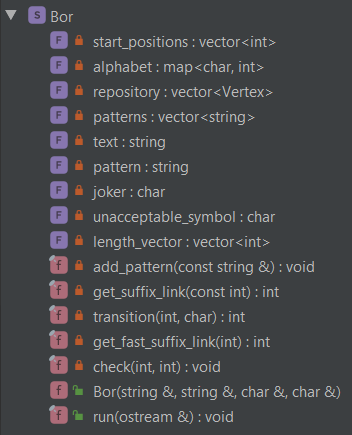


Рисунок 2. UML - диаграмма класса бора.

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

Метод, позволяющий добавить переданную в качестве аргумента строку в бор.

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

Метод, возвращающий номер вершины, на которую указывает суффиксная ссылка.

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

Метод, возвращающий номер вершины, следующий за текущей.

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

Метод, возвращающий номер вершины, на которую указывает «быстрая» суффиксная ссылка.

1. void check(int vertex, int position\_in\_text, std::ostream&) noexcept;

Метод проверки вершины бора на лист.

1. void run(std::ostream& output) noexcept;

Основной метод, производящий поиск шаблона с джокером в тексте.

**Тестирование**

Таблица 1 – результаты тестирования

|  |  |
| --- | --- |
| **Input** | **Output** |
| NACGNTTACGGTCACNN  AC$$T$AC$$  $  C | From 0 vertex to 0  path: 0  From 0 vertex to 1  path: 1 -> 0  From 1 vertex to 2  path: 2 -> 0  From 2 vertex to 0  path: 0  From 0 vertex to 0  path: 0  From 0 vertex to 3  path: 3 -> 0  From 3 vertex to 3  path: 3 -> 0  From 3 vertex to 1  path: 1 -> 0  From 1 vertex to 2  path: 2 -> 0  From 2 vertex to 0  path: 0  From 0 vertex to 0  path: 0  From 0 vertex to 3  path: 3 -> 0  From 3 vertex to 0  path: 0  From 0 vertex to 1  path: 1 -> 0  From 1 vertex to 2  path: 2 -> 0  From 2 vertex to 0  path: 0  From 0 vertex to 0  path: 0  Find match at 2 position.  Find unacceptable symbol! |
| NACGNTTACGGTCACNN  AC$$T$AC$$  $  A | From 0 vertex to 0  path: 0  From 0 vertex to 1  path: 1 -> 0  From 1 vertex to 2  path: 2 -> 0  From 2 vertex to 0  path: 0  From 0 vertex to 0  path: 0  From 0 vertex to 3  path: 3 -> 0  From 3 vertex to 3  path: 3 -> 0  From 3 vertex to 1  path: 1 -> 0  From 1 vertex to 2  path: 2 -> 0  From 2 vertex to 0  path: 0  From 0 vertex to 0  path: 0  From 0 vertex to 3  path: 3 -> 0  From 3 vertex to 0  path: 0  From 0 vertex to 1  path: 1 -> 0  From 1 vertex to 2  path: 2 -> 0  From 2 vertex to 0  path: 0  From 0 vertex to 0  path: 0  Find match at 2 position.  Find match at 8 position. |
| ACTANCA  A$$A$  $  G | From 0 vertex to 1  path: 1 -> 0  From 1 vertex to 0  path: 0  From 0 vertex to 0  path: 0  From 0 vertex to 1  path: 1 -> 0  From 1 vertex to 0  path: 0  From 0 vertex to 0  path: 0  From 0 vertex to 1  path: 1 -> 0  Find match at 1 position. |

**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.

**ПРИЛОЖЕНИЯ А.**

**ИСХОДНЫЙ КОД.**

Имя файла: main.cpp

#include "Bor.h"

int main() {

int input\_mode = 0;

int output\_mode = 0;

std::cout << "0 for terminal input, 1 for file input:";

std::cin >> input\_mode;

std::cout << "0 for terminal output, 1 for file output:";

std::cin >> output\_mode;

std::string text;

std::string pattern;

char joker;

char unacceptable\_symbol;

if (input\_mode == 1) {

std::ifstream input("input.txt");

if (input.is\_open()) {

input >> text;

input >> pattern;

input >> joker;

input >> unacceptable\_symbol;

} else {

std::cout << "Can't open input file!" << std::endl;

}

} else {

std::cin >> text;

std::cin >> pattern;

std::cin >> joker;

std::cin >> unacceptable\_symbol;

}

Bor bor(text, pattern, joker, unacceptable\_symbol);

if (output\_mode == 0) {

bor.run(std::cout);

} else if (output\_mode == 1) {

std::ofstream output("output.txt", std::ios::out | std::ios::trunc);

bor.run(output);

} else {

std::cout << "Wrong prameters!" << std::endl;

}

return 0;

}

Имя файла: Bor.h

#ifndef LAB5\_BOR\_H

#define LAB5\_BOR\_H

#include <iostream>

#include <map>

#include <vector>

#include <fstream>

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{};

};

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, std::ostream&) noexcept;

public:

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

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

};

#endif

Имя файла: Bor.cpp

#include "Bor.h"

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

int another\_vertex = 0;

for (char symbol : inserting\_string) {

char ch = alphabet[symbol];

if (repository[another\_vertex].neighbors[ch] == -1) {

repository.emplace\_back(Vertex(another\_vertex, ch));

repository[another\_vertex].neighbors[ch] = repository.size() - 1;

}

another\_vertex = repository[another\_vertex].neighbors[ch];

}

repository[another\_vertex].is\_leaf = true;

patterns.emplace\_back(inserting\_string);

repository[another\_vertex].patterns\_num.emplace\_back(patterns.size() - 1);

}

[[nodiscard]] int Bor::get\_suffix\_link(const int vertex) noexcept {

if (repository[vertex].suff\_link == -1) {

if (vertex == 0 || repository[vertex].parent == 0) {

repository[vertex].suff\_link = 0;

} else {

repository[vertex].suff\_link = transition(get\_suffix\_link(repository[vertex].parent), repository[vertex].symbol);

}

}

return repository[vertex].suff\_link;

}

[[nodiscard]] int Bor::transition(int vertex, char symbol) noexcept {

if (repository[vertex].auto\_move[symbol] == -1) {

if (repository[vertex].neighbors[symbol] != -1) {

repository[vertex].auto\_move[symbol] = repository[vertex].neighbors[symbol];

} else {

if (vertex == 0) {

repository[vertex].auto\_move[symbol] = 0;

} else {

repository[vertex].auto\_move[symbol] = transition(get\_suffix\_link(vertex), symbol);

}

}

}

return repository[vertex].auto\_move[symbol];

}

[[nodiscard]] int Bor::get\_fast\_suffix\_link(int vertex) noexcept {

if (repository[vertex].suff\_flink == -1) {

int tmp = get\_suffix\_link(vertex);

if (tmp == 0) {

repository[vertex].suff\_flink = 0;

} else {

repository[vertex].suff\_flink = repository[vertex].is\_leaf ? tmp : get\_fast\_suffix\_link(tmp);

}

}

return repository[vertex].suff\_flink;

}

void Bor::check(int vertex, int position\_in\_text, std::ostream& output) noexcept {

for (int u = vertex; u != 0; u = get\_fast\_suffix\_link(u)) {

output << u << " -> ";

if (repository[u].is\_leaf) {

for (int k : repository[u].patterns\_num) {

size\_t j = position\_in\_text - patterns[k].size() + 1;

if (j - start\_positions[k] + 1 > 0 && j - start\_positions[k] + 1 < length\_vector.size()) {

++length\_vector[j - start\_positions[k] + 1];

}

}

}

}

output << 0 << std::endl << std::endl;

}

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

repository.emplace\_back(Vertex(0, '#'));

alphabet['A'] = 0;

alphabet['C'] = 1;

alphabet['G'] = 2;

alphabet['T'] = 3;

alphabet['N'] = 4;

length\_vector = std::vector<int>(text.size());

}

void Bor::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

output << "From " << another\_vertex << " vertex ";

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

output << "to " << another\_vertex << std::endl;

output << "path: ";

check(another\_vertex, i, output);

}

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

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

bool is\_joker = false;

// output << "at " << k << " position" << std::endl;

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;

}

}

}