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 P в текст T.

Например, образец a*b*??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*}.

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 $\{Q_1, \ldots, Q_k\}$ — набор подстрок Q, разделенных масками, и пусть

$\{l_1,\ldots,l_k\}$ — 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 Q_i в тексте на позиции j будет означать возможное появление шаблона на позиции $j-l_i+1$.

1. Используя алгоритм Ахо-Корасик, находим безмасочные подстроки шаблона Q: когда находим Q_i в тексте T на позиции j, увеличиваем на единицу $C[j-l_i+1]$.
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.

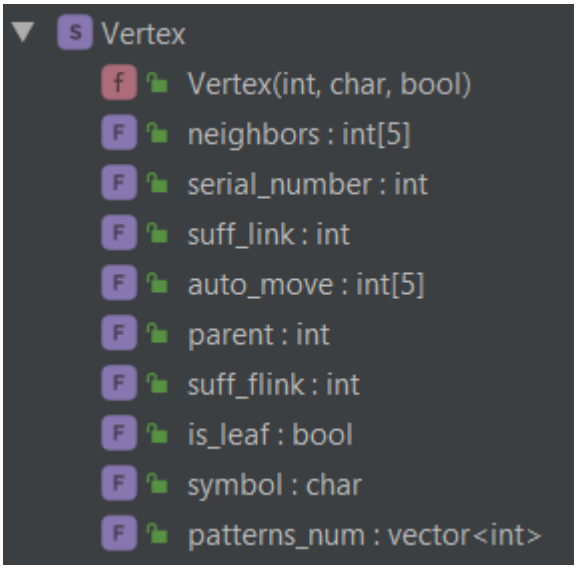

Рисунок 1. UML диаграмма структуры данных для хранения вершины бора.

UML - диаграмма класса бора представлена на рис. 2.

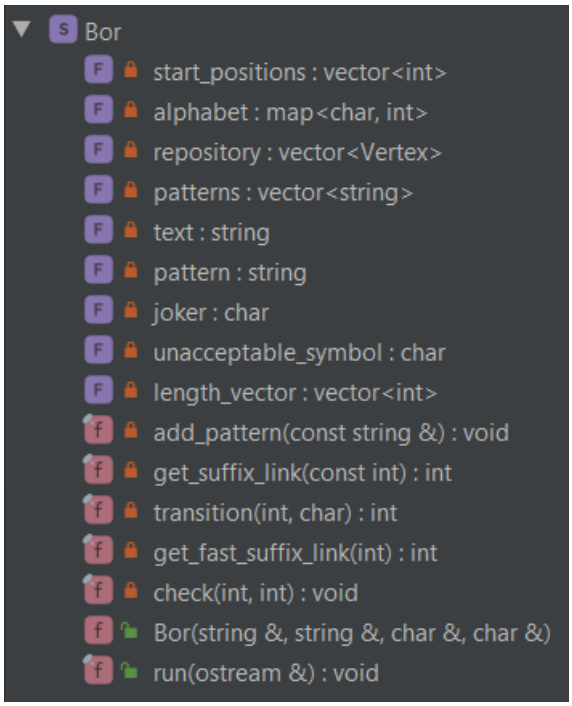

Рисунок 2. UML - диаграмма класса бора.

1) `void add_pattern(const std::string& inserting_string) noexcept;`

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

2) `[[nodiscard]] int get_suffix_link(const int vertex) noexcept;`

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

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

   Метод, возвращающий номер вершины, следующий за текущей.

4) `[[nodiscard]] int get_fast_suffix_link(int vertex) noexcept;`

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

5) `void check(int vertex, int position_in_text, std::ostream&) noexcept;`

Метод проверки вершины бора на лист.

6) `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;
            }
        }
    }
```