

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

Студент гр. 9383

Гладких А.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Применить на практике знания о построение алгоритма Ахо-Корасик. Реализовать алгоритм Ахо-Корасик для поиска всех подстрок по заданному шаблону.

Основные теоретические положения.

Бор — структура данных для хранения набора строк, представляющая из себя подвешенное дерево с символами на рёбрах. Строки получаются последовательной записью всех символов, хранящихся на рёбрах между корнем бора и терминальной вершиной. Размер бора линейно зависит от суммы длин всех строк, а поиск в бору занимает время, пропорциональное длине образца.

Алгоритм Ахо-Корасик — эффективный алгоритм, осуществляющий поиск подстрок в заданном тексте.

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.

Задание.

1) Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

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

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

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

Выход:

Все вхождения образцов из P в T .

Каждое вхождение образца в текст представить в виде двух чисел - i и 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.

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 $xabvccbababcsax$.

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 , $1 \leq |T| \leq 100000$)

Шаблон (P , $1 \leq |P| \leq 40$)

Символ джокера

Выход:

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

Номера должны выводиться в порядке возрастания.

Ход работы:

1. Произведён анализ задания.
2. Было реализовано построение бора:

1. Сперва создается бор из искомых подстрок. Заданные подстроки одна за другой посимвольно добавляются в бор. При необходимости программа выделяет дополнительную память под вершины бора.
2. После этого программа преобразует бор в автомат. Программа добавляет суффиксные ссылки с помощью обхода бора в ширину. Суффиксная ссылка для каждой вершины — это вершина, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине. У корня бора суффиксная ссылка указывает на самого себя. Если у вершины нулевой собственный суффикс, то ее суффиксная ссылка указывает на корень.
3. Был реализован алгоритм Ахо-Корасик:
 1. Программа постепенно считывает текст, в котором нужно искать заданные подстроки. Параллельно с этим осуществляется движение по бору:
 - Если в боре существует переход по очередному символу текста, то он совершается. Если же перехода нет, то программа переходит по суффиксной ссылке данной вершины.
 - Новая вершина может быть:
 - Конечной — это означает, что мы успешно нашли подстроку в тексте. Тогда отметка о нахождении образца отмечается в результатах, и программа обходит цепочку конечных ссылок, чтобы проверить, не были ли пропущены образцы, являющиеся подстроками найденного образца. При обходе все конечные вершины также заносятся в ответ.
 - Не конечной — тогда программа продолжает алгоритм с пункта 1.
 2. Алгоритм завершит свою работу, когда дойдет до конца текста.
4. Сложность алгоритма Ахо-Корасик по памяти — $O(|M|)$, где $|M|$ — суммарная длина искомых образцов. По времени же сложность можно оценить как $O(|T| + |M| * \log|M| + |k|)$, где $|T|$

- длина текста, в котором ведется поиск, $|M|$ – суммарная длина искомых образцов, а $|k|$ - общая длина всех совпадений. Фактически сложность состоит из построения бора для заданных образцов и дальнейшего прохода по всем символам текста.
5. Задача 1 была решена простым применением алгоритма Ахо-Корасик.
 6. Задача 2 была решена с помощью алгоритма Ахо-Корасик с некоторыми изменениями:
 1. Автомат строился из образцов, полученных выделением максимальных безджокерных подстрок из шаблонной подстроки, для каждого из которых было записано смещение (смещения).
 2. Был создан дополнительный заполненный нулями массив с длиной, совпадающей с текстом.
 3. При обнаружении образца инкрементировалась ячейка массива с адресом номер начального символа образца в тексте минус смещение образца. В случае же, если у образца было несколько смещений, то инкрементировать все соответствующие ячейки массива.
 4. В результате работы алгоритма Ахо-Корасик шаблонная подстрока будет начинаться в тех местах текста, для которых соответствующая ячейка массива содержит количество образцов с учетом кратности.
 7. Код разработанной программы расположен в Приложении А.

Описание функций и структур данных.

1. Класс AhoCorasick. Является реализацией алгоритма Ахо-Корасик. Имеет поля `root_node_` для хранения корня бора и структуру `vector` из стандартной библиотеки языка C++ `words_` для хранения заданных слов. Класс имеет следующие методы: `add_string()` для добавления подстроки в бор, `init()` для инициализации автомата, `create_refs()` для построения суффиксных и конечных ссылок автомата, публичную функцию `print()` и приватную функцию `print_trie()` для вывода бора на экран и метод `search()` для запуска алгоритма Ахо-Корасик.

Для задания 2 в классе AhoCorasick также присутствуют методы `add_pattern_with_jokers()` для добавления строки с джокерами в бор и метод `add_string_with_offset()`, который добавляет строку в бор и выставляет ей смещение. Также дополнительно в поле `patt_size` хранится длина подстроки до исключения джокеров.

2. Структура `Node`. Имеет следующие поля — `symbol` для хранения символа вершины, `word_index` для получения слова из списка, динамический массив `vector` для хранения смещений, указатель `parent` на вершину родителя, указатель `suffix_ref` на вершину, в которую приходит суффиксная ссылка, указатель `final_ref` на вершину, в которую приходит конечная ссылка, а также динамический массив `next_nodes` для хранения потомков текущей вершины. Также в структуре для удобства были определены следующие методы: `findNextNode()` для поиска вершины по символу среди потомков и метод `isFinal()` для проверки, является ли вершина конечной.
3. Функция `main()` - функция, в которой происходит считывание входных данных и запуск выполнения алгоритма.

Примеры работы программы.

Таблица 1 – Пример работы программы в задании №1

| № п/п | Входные данные | Выходные данные |
|-------|-------------------------------|-----------------|
| 1. | NTAG З TAGT TAG Т | 2 2 2 3 |
| 2. | ACCACC | 1 1 |

| | | |
|----|----------------------------|--------------------------|
| | 3 AC CA CC | 2 3 3 2 4 1 5 3 |
| 3. | GGGGGG 3 N A C | |

Таблица 2 – Пример работы программы в задании №2

| № п/п | Входные данные | Выходные данные |
|-------|----------------------------|-----------------------|
| 1. | ACTANCA A\$\$\$ \$ | 1 |
| 2. | GACAAC %%A%% % | 2 |
| 3. | GACAAAGACAAC %%A%% % | 2 3 4 6 8 |

Иллюстрация работы программы.

```
followjust@DESKTOP-0H9PDI5:
NTAG
3
TAGT
TAG
T
2 2
2 3
```

Рисунок 1 - Пример работы программы в задании №1 на входных данных №1

```
followjust@DESKTOP-0H9PDI5:
ACCACC
3
AC
CA
CC
1 1
2 3
3 2
4 1
5 3
```

Рисунок 2 - Пример работы программы в задании №1 на входных данных №2

```
followjust@DESKTOP-0H9PDI5:
GGGGGG
3
N
A
C
```

Рисунок 3 - Пример работы программы в задании №1 на входных данных №3


```
followjust@DESKTOP-0H9PDI5
ACTANCA
A$$$A$
$
1
```

Рисунок 4 - Пример работы программы в задании №2 на входных данных №1

```
followjust@DESKTOP-0H9PDI5:
GACAAC
$$A$$
$
2
```

Рисунок 5 - Пример работы программы в задании №2 на входных данных №2

```
followjust@DESKTOP-0H9PDI5:
GACAAAGACAAC
%%A%%
%
2
3
4
6
8
```

Рисунок 6 - Пример работы программы в задании №2 на входных данных №3

Выводы.

Были применены на практике знания о построение алгоритма Ахо-Корасик. Был реализован алгоритм Ахо-Корасик для поиска всех подстрок по заданным шаблонам. С помощью реализованного алгоритма была решена задаче о поиске всех подстрок в тексте, а также была решена задача множественного поиска в тексте подстрок со специальным символом-джокером, являющимся любым символом алфавита.

ПРИЛОЖЕНИЕ А

ФАЙЛ MAIN1.CPP

```
#INCLUDE <Iostream>

#include "AhoCorasick.hpp"

int main() {

    AhoCorasick aho_corasick;

    int patt_amount;
    std::string patt, temp;

    std::cin >> temp;
    std::cin >> patt_amount;

    for(int i = 0; i < patt_amount; ++i) {
        std::cin >> patt;
        aho_corasick.add_string(patt);
    }

    aho_corasick.init();

    auto ans = aho_corasick.search(temp);

    for(const auto& pair: ans) {
        std::cout << pair.first << " " << pair.second << "\n";
    }

    return 0;
}
```

ФАЙЛ AhoCorasick1.hpp

```
#pragma once

#include <Iostream>

#include <vector>
#include <string>
#include <queue>
#include <algorithm>

struct Node{
    char symbol;
    int word_index = -1;

    Node* parent;
    Node* suffix_ref;
    Node* final_ref;

    std::vector<Node*> next_nodes;
}
```

```

    NODE() {}
        NODE(CHAR    NEWSYMBOL,    NODE*    PARENT_NODE)    :    SYMBOL(NEWSYMBOL),
PARENT(PARENT_NODE) {}

    NODE* FIND_NEXT_NODE(CHAR SYMBOL);

    BOOL ISFINAL();
};

CLASS AHOCORASICK{

PUBLIC:

    AHOCORASICK();
    VOID ADD_STRING(CONST STD::STRING& PATT);

    STD::VECTOR<STD::PAIR<INT, INT>> SEARCH(CONST STD::STRING& TEMP);

    VOID INIT();

    VOID PRINT();

PRIVATE:

    NODE* ROOT_NODE_;
    STD::VECTOR<STD::STRING> WORDS_;

    VOID CREATE_REFS();
    VOID PRINT_TRIE(NODE* CUR_NODE, INT LEVEL);

};

```

Файл AHOCORASICK1.CPP

```

#include "AHOCORASICK.HPP"

AHOCORASICK::AHOCORASICK() {
    ROOT_NODE_ = NEW NODE();
    ROOT_NODE_ -> PARENT = ROOT_NODE_;
    ROOT_NODE_ -> SUFFIX_REF = ROOT_NODE_;
}

VOID AHOCORASICK::ADD_STRING(CONST STD::STRING& PATT) {

    NODE* CUR_NODE = ROOT_NODE_;

    NODE* CHILD_NODE;

    FOR(CONST AUTO& SYMBOL: PATT) {

        CHILD_NODE = CUR_NODE -> FIND_NEXT_NODE(SYMBOL);

        IF(!CHILD_NODE) {

```

```

        CHILD_NODE = NEW NODE ({SYMBOL, CUR_NODE});
        CUR_NODE->NEXT_NODES.PUSH_BACK(CHILD_NODE);
    }

    CUR_NODE = CHILD_NODE;
}

CUR_NODE->WORD_INDEX = WORDS_.SIZE();
WORDS_.PUSH_BACK(PATT);
}

VOID AhoCORASICK::INIT() {
    CREATE_REFS();
}

VOID AhoCORASICK::CREATE_REFS() {
    STD::QUEUE<NODE*> NODES_QUEUE;
    NODE* TEMP_NODE;
    NODE* FOUND_NODE;
    NODE* NODE_FOR_SUFFIX_REF;
    BOOL SUFFIX_REF_SET;

    FOR (CONST AUTO& NEXT_NODE: ROOT_NODE->NEXT_NODES) {
        NEXT_NODE->SUFFIX_REF = ROOT_NODE;
        NODES_QUEUE.PUSH(NEXT_NODE);
    }

    WHILE (!NODES_QUEUE.EMPTY()) {
        TEMP_NODE = NODES_QUEUE.FRONT();
        NODES_QUEUE.POP();

        FOR (CONST AUTO& NEXT_NODE: TEMP_NODE->NEXT_NODES) {
            SUFFIX_REF_SET = FALSE;
            NODE_FOR_SUFFIX_REF = TEMP_NODE;
            WHILE (NODE_FOR_SUFFIX_REF != ROOT_NODE) {
                NODE_FOR_SUFFIX_REF = NODE_FOR_SUFFIX_REF->SUFFIX_REF;
                FOUND_NODE = NODE_FOR_SUFFIX_REF->FIND_NEXT_NODE(NEXT_NODE->
>SYMBOL);

                IF (FOUND_NODE) {
                    IF (!SUFFIX_REF_SET) {
                        SUFFIX_REF_SET = TRUE;
                        NEXT_NODE->SUFFIX_REF = FOUND_NODE;
                    }

                    IF (FOUND_NODE->ISFINAL()) {
                        NEXT_NODE->FINAL_REF = FOUND_NODE;
                        BREAK;
                    }
                }
            }

            IF (!SUFFIX_REF_SET) {
                NEXT_NODE->SUFFIX_REF = ROOT_NODE;
            }

            NODES_QUEUE.PUSH(NEXT_NODE);
        }
    }
}

```

```

    }
}

STD::VECTOR<STD::PAIR<INT, INT>> AhoCorasick::SEARCH(CONST STD::STRING& TEMP) {

    STD::VECTOR<STD::PAIR<INT, INT>> ANS;

    NODE* CUR_NODE = ROOT_NODE_;
    NODE* NEXT_NODE;

    STD::STRING DEBUG;

    FOR(INT I = 0; (SIZE_T)I < TEMP.SIZE(); ++I) {
        CHAR TEXT_SYMBOL = TEMP[I];

        NEXT_NODE = CUR_NODE->FIND_NEXT_NODE(TEXT_SYMBOL);

        WHILE(!NEXT_NODE) {
            IF(CUR_NODE == ROOT_NODE_) {
                NEXT_NODE = ROOT_NODE_;
                BREAK;
            }
            CUR_NODE = CUR_NODE->SUFFIX_REF;
            NEXT_NODE = CUR_NODE->FIND_NEXT_NODE(TEXT_SYMBOL);
        }

        CUR_NODE = NEXT_NODE;

        WHILE(NEXT_NODE != ROOT_NODE_) {
            IF(NEXT_NODE->IS_FINAL()) ANS.PUSH_BACK(STD::MAKE_PAIR(I -
WORDS_[NEXT_NODE->WORD_INDEX].SIZE() + 2, NEXT_NODE->WORD_INDEX + 1));
            NEXT_NODE = NEXT_NODE->SUFFIX_REF;
        }
    }

    AUTO CMP_FOR_ANS = [] (CONST STD::PAIR<INT, INT>& A, CONST STD::PAIR<INT,
INT>& B) {
        IF(A.FIRST == B.FIRST) RETURN A.SECOND < B.SECOND;
        RETURN A.FIRST < B.FIRST;
    };

    STD::SORT(ANS.BEGIN(), ANS.END(), CMP_FOR_ANS);

    RETURN ANS;
}

VOID AhoCorasick::PRINT() {
    STD::COUT << "ROOT";
    PRINT_TRIE(ROOT_NODE_, 0);
}

VOID AhoCorasick::PRINT_TRIE(NODE* CUR_NODE, INT LEVEL) {
    IF(!CUR_NODE) RETURN;

    FOR(INT I = 0; I < LEVEL; ++I) {

```

```

        STD::COUT << '\t';
    }

    STD::COUT << CUR_NODE->SYMBOL << " " << CUR_NODE << " SUFF: " <<
CUR_NODE->SUFFIX_REF << " FINAL: " << CUR_NODE->FINAL_REF << "\n";

    FOR (CONST AUTO& NEXT_NODE: CUR_NODE->NEXT_NODES) {
        PRINT_TRIE (NEXT_NODE, LEVEL + 1);
    }
}

NODE* NODE::FIND_NEXT_NODE (CHAR SYMBOL_TO_FIND) {
    FOR (CONST AUTO& NEXT_NODE: NEXT_NODES) {
        IF (NEXT_NODE->SYMBOL == SYMBOL_TO_FIND) {
            RETURN NEXT_NODE;
        }
    }

    RETURN NULLPTR;
}

BOOL NODE::ISFINAL () {
    RETURN WORD_INDEX >= 0;
}

```

ФАЙЛ MAIN2.CPP

```

#include <Iostream>

#include "AHOCORASICK.HPP"

INT MAIN () {

    STD::STRING PATT, TEMP;
    CHAR JOKER;

    STD::CIN >> TEMP;
    STD::CIN >> PATT;
    STD::CIN >> JOKER;

    AHO_CORASICK AHO_CORASICK;

    AHO_CORASICK.ADD_PATTERN_WITH_JOKERS (PATT, JOKER);

    AHO_CORASICK.INIT ();

    AUTO ANS = AHO_CORASICK.SEARCH (TEMP);

    FOR (CONST AUTO& IT: ANS) {
        STD::COUT << IT << "\n";
    }

    RETURN 0;
}

```

```
}
```

ФАЙЛ AHO-CORASICK2.HPP

```
#PRAGMA ONCE

#include <iostream>

#include <vector>
#include <string>
#include <queue>
#include <algorithm>

STRUCT NODE{
    CHAR SYMBOL;
    INT WORD_INDEX = -1;
    STD::VECTOR<INT> OFFSETS;

    NODE* PARENT;
    NODE* SUFFIX_REF;
    NODE* FINAL_REF;

    STD::VECTOR<NODE*> NEXT_NODES;

    NODE() {}
    NODE(CHAR NEWSYMBOL, NODE* PARENT_NODE) : SYMBOL(NEWSYMBOL),
PARENT(PARENT_NODE) {}

    NODE* FIND_NEXT_NODE(CHAR SYMBOL);

    BOOL ISFINAL();
};

CLASS AHO-CORASICK{

PUBLIC:

    AHO-CORASICK();
    VOID ADD_PATTERN_WITH_JOKERS(STD::STRING PATT, CONST CHAR JOKER);

    STD::VECTOR<INT> SEARCH(CONST STD::STRING& TEMP);

    VOID INIT();

    VOID PRINT();

PRIVATE:

    NODE* ROOT_NODE_;
    STD::VECTOR<STD::STRING> WORDS_;
    INT PATT_SIZE;

    VOID CREATE_REFS();
    VOID PRINT_TRIE(NODE* CUR_NODE, INT LEVEL);
```



```
VOID ADD_STRING_WITH_OFFSET(CONST STD::STRING& PATT, INT OFFSET);
```

```
};
```

ФАЙЛ АНОCORASICK2.CPP

```
#INCLUDE "AHOCORASICK.HPP"
```

```
AHOCORASICK::AHOCORASICK() {
    ROOT_NODE_ = NEW NODE();
    ROOT_NODE_ -> PARENT = ROOT_NODE_;
    ROOT_NODE_ -> SUFFIX_REF = ROOT_NODE_;
}
```

```
VOID AHOCORASICK::INIT() {
    CREATE_REFS();
}
```

```
VOID AHOCORASICK::CREATE_REFS() {
    STD::QUEUE<NODE*> NODES_QUEUE;
    NODE* TEMP_NODE;
    NODE* FOUND_NODE;
    NODE* NODE_FOR_SUFFIX_REF;
    BOOL SUFFIX_REF_SET;

    FOR (CONST AUTO& NEXT_NODE: ROOT_NODE_ -> NEXT_NODES) {
        NEXT_NODE -> SUFFIX_REF = ROOT_NODE_;
        NODES_QUEUE.PUSH(NEXT_NODE);
    }

    WHILE (!NODES_QUEUE.EMPTY()) {
        TEMP_NODE = NODES_QUEUE.FRONT();
        NODES_QUEUE.POP();

        FOR (CONST AUTO& NEXT_NODE: TEMP_NODE -> NEXT_NODES) {
            SUFFIX_REF_SET = FALSE;
            NODE_FOR_SUFFIX_REF = TEMP_NODE;
            WHILE (NODE_FOR_SUFFIX_REF != ROOT_NODE_) {
                NODE_FOR_SUFFIX_REF = NODE_FOR_SUFFIX_REF -> SUFFIX_REF;
                FOUND_NODE = NODE_FOR_SUFFIX_REF -> FIND_NEXT_NODE(NEXT_NODE -
>SYMBOL);

                IF (FOUND_NODE) {
                    IF (!SUFFIX_REF_SET) {
                        SUFFIX_REF_SET = TRUE;
                        NEXT_NODE -> SUFFIX_REF = FOUND_NODE;
                    }

                    IF (FOUND_NODE -> IS_FINAL()) {
                        NEXT_NODE -> FINAL_REF = FOUND_NODE;
                        BREAK;
                    }
                }
            }

            IF (!SUFFIX_REF_SET) {
```

```

        NEXT_NODE->SUFFIX_REF = ROOT_NODE_;
    }

    NODES_QUEUE.PUSH(NEXT_NODE);
}

}

STD::VECTOR<INT> AhoCorasick::SEARCH(CONST STD::STRING& TEMP) {
    STD::VECTOR<INT> ANS;
    STD::VECTOR<INT> MATCHES_AMOUNT(TEMP.SIZE());

    NODE* CUR_NODE = ROOT_NODE_;
    NODE* NEXT_NODE;

    FOR(INT I = 0; (SIZE_T)I < TEMP.SIZE(); ++I) {
        CHAR TEXT_SYMBOL = TEMP[I];

        NEXT_NODE = CUR_NODE->FIND_NEXT_NODE(TEXT_SYMBOL);

        WHILE(!NEXT_NODE) {
            IF(CUR_NODE == ROOT_NODE_) {
                NEXT_NODE = ROOT_NODE_;
                BREAK;
            }
            CUR_NODE = CUR_NODE->SUFFIX_REF;
            NEXT_NODE = CUR_NODE->FIND_NEXT_NODE(TEXT_SYMBOL);
        }

        CUR_NODE = NEXT_NODE;

        WHILE(NEXT_NODE != ROOT_NODE_) {
            FOR(CONST AUTO& OFFSET : NEXT_NODE->OFFSETS) {
                IF(I - OFFSET >= 0) {
                    MATCHES_AMOUNT[I - OFFSET]++;
                }
            }
            NEXT_NODE = NEXT_NODE->SUFFIX_REF;
        }
    }

    FOR (INT I = 0; (SIZE_T)I < MATCHES_AMOUNT.SIZE(); I++) {
        IF (MATCHES_AMOUNT[I] == (INT)WORDS_.SIZE() && I + PATT_SIZE <=
(INT)MATCHES_AMOUNT.SIZE()) {
            ANS.PUSH_BACK(I + 1);
        }
    }

    RETURN ANS;
}

VOID AhoCorasick::PRINT() {
    STD::COUT << "ROOT";
    PRINT_TRIE(ROOT_NODE_, 0);
}

VOID AhoCorasick::PRINT_TRIE(NODE* CUR_NODE, INT LEVEL) {

```

```

    IF (!CUR_NODE) RETURN;

    FOR (INT I = 0; I < LEVEL; ++I) {
        STD::COUT << '\t';
    }

    STD::COUT << CUR_NODE->SYMBOL << " " << CUR_NODE << " SUFF: " <<
CUR_NODE->SUFFIX_REF << " FINAL: " << CUR_NODE->FINAL_REF << "\n";

    FOR (CONST AUTO& NEXT_NODE: CUR_NODE->NEXT_NODES) {
        PRINT_TRIE (NEXT_NODE, LEVEL + 1);
    }
}

VOID AhoCORASICK::ADD_PATTERN_WITH_JOKERS (STD::STRING PATT, CONST CHAR JOKER) {
    PATT_SIZE = PATT.SIZE();
    STD::STRING SUBPATT;
    INT OFFSET = 0;
    FOR (CONST AUTO& SYMBOL: PATT) {
        IF (SYMBOL == JOKER) {
            IF (!SUBPATT.EMPTY()) {
                ADD_STRING_WITH_OFFSET (SUBPATT, OFFSET - 1);
            }

            SUBPATT.CLEAR();
        }
        ELSE {
            SUBPATT += SYMBOL;
        }
        OFFSET++;
    }
    IF (!SUBPATT.EMPTY()) ADD_STRING_WITH_OFFSET (SUBPATT, OFFSET - 1);
}

VOID AhoCORASICK::ADD_STRING_WITH_OFFSET (CONST STD::STRING& PATT, INT OFFSET) {
    NODE* CUR_NODE = ROOT_NODE;

    NODE* CHILD_NODE;

    FOR (CONST AUTO& SYMBOL: PATT) {

        CHILD_NODE = CUR_NODE->FIND_NEXT_NODE (SYMBOL);

        IF (!CHILD_NODE) {
            CHILD_NODE = NEW NODE ({SYMBOL, CUR_NODE});
            CUR_NODE->NEXT_NODES.PUSH_BACK (CHILD_NODE);
        }

        CUR_NODE = CHILD_NODE;
    }

    CUR_NODE->WORD_INDEX = WORDS_.SIZE();
    CUR_NODE->OFFSETS.PUSH_BACK (OFFSET);
    WORDS_.PUSH_BACK (PATT);
}

```

```

NODE* NODE::FIND_NEXT_NODE(CHAR SYMBOL_TO_FIND) {
    FOR(CONST AUTO& NEXT_NODE: NEXT_NODES) {
        IF (NEXT_NODE->SYMBOL == SYMBOL_TO_FIND) {
            RETURN NEXT_NODE;
        }
    }

    RETURN nullptr;
}

BOOL NODE::ISFINAL() {
    RETURN WORD_INDEX >= 0;
}

```

ФАЙЛ MAKEFILE

```

FLAGS = -std=c++17 -Wall -Wextra
BUILD = BUILD
SOURCE_TASK1 = SOURCE/TASK1
SOURCE_TASK2 = SOURCE/TASK2
TEST = TEST

$(SHELL) MKDIR -p $(BUILD)

ALL: LAB5_TASK1 LAB5_TASK2

LAB5_TASK1: $(BUILD)/MAIN1.o $(BUILD)/AHOCORASICK1.o
    @ECHO "TO START ENTER ./LAB5_TASK1"
    @G++ $(BUILD)/MAIN1.o $(BUILD)/AHOCORASICK1.o -o LAB5_TASK1 $(FLAGS)

LAB5_TASK2: $(BUILD)/MAIN2.o $(BUILD)/AHOCORASICK2.o
    @ECHO "TO START ENTER ./LAB5_TASK2"
    @G++ $(BUILD)/MAIN2.o $(BUILD)/AHOCORASICK2.o -o LAB5_TASK2 $(FLAGS)

$(BUILD)/MAIN1.o: $(SOURCE_TASK1)/MAIN.CPP
    @G++ -c $(SOURCE_TASK1)/MAIN.CPP -o $(BUILD)/MAIN1.o $(FLAGS)

$(BUILD)/MAIN2.o: $(SOURCE_TASK2)/MAIN.CPP
    @G++ -c $(SOURCE_TASK2)/MAIN.CPP -o $(BUILD)/MAIN2.o $(FLAGS)

$(BUILD)/AHOCORASICK1.o: $(SOURCE_TASK1)/AHOCORASICK.CPP
$(SOURCE_TASK1)/AHOCORASICK.HPP
    @G++ -c $(SOURCE_TASK1)/AHOCORASICK.CPP -o $(BUILD)/AHOCORASICK1.o $(
    FLAGS)

$(BUILD)/AHOCORASICK2.o: $(SOURCE_TASK2)/AHOCORASICK.CPP
$(SOURCE_TASK2)/AHOCORASICK.HPP
    @G++ -c $(SOURCE_TASK2)/AHOCORASICK.CPP -o $(BUILD)/AHOCORASICK2.o $(
    FLAGS)

CLEAN:
    @RM -rf $(BUILD)/
    @RM -rf *.o LAB5_TASK1 LAB5_TASK2

```