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: Алгоритм Ахо-Корасик

Студент гр. 3343 _____ Пименов П.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

Реализовать алгоритм Ахо-Корасик для поиска набора подстрок в строке. Решить задачу поиска шаблона с символами-джокерами в строке.

Задание.

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 $P=\{p1,\ldots,pn\}$ ($1 \leq |pi| \leq 75$).

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 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.

Sample Input:

NTAG

3

TAGT

TAG

T

Sample Output:

2 2

2 3

Задание 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 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}

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.

Sample Input:

ACTANCA

A$$A$

$

Sample Output:

1

Индивидуализация (Вариант 7)

Вывод графического представления автомата.

Выполнение работы.

Алгоритм успешно реализован. Описание членов класса-решения:

- struct Node – структура, описывающая вершину бора. Хранит информацию о дочерних вершинах, суффиксную и конечную ссылки, номер вершины родителя и ребро перехода, а также номера шаблонов, оканчивающихся в этой вершине.
- vector<Node> trie – вектор вершин, представляющий бор
- string buildLabel(int idx) – строит строку, которую представляет вершина с определенным номером. Необходимо для визуализации
- void checkMatches(int pos, int node, const vector<string>& patterns, vector<pair<int, int>>& result) – поиск всех шаблонов, которые заканчиваются в текущем состоянии автомата. Поднимается по конечным ссылкам и сохраняет найденные шаблоны. Необходим для поиска точного набора образцов.
- void addPattern(const string& pattern, int index) – добавляет шаблон pattern с номером index в бор. Достраивает необходимые вершины и добавляет их в бор.
- void buildAutomaton() – строит автомат. Для каждой вершины инициализирует суффиксную и конечную ссылку (если такие есть) по правилам алгоритма.

- void generateGraph() – генерирует графическое представление автомата.
- void printAutomaton() – выводит текстовое представление автомата с описанием каждой вершины. Для визуализации используется библиотека Graphviz.
- vector<pair<int, int>> search(const string& text, const vector<string>& patterns) – осуществляет поиск точного набора образцов в строке, используя построенный автомат. Идет по строке поиска, переводит автомат в соответствующее состояние. При ненахождении вершины в автомате, переходит по суффиксной ссылке. При нахождении образца, вызывается checkMatches, который обрабатывает конечные ссылки.
- vector<int> searchWithJoker(const string& text, const string& pattern, vector<pair<string, int>>& parts) – осуществляет шаблона с джокерами в строке. Для этого необходимо построить автомат по частям шаблона без символов-джокеров, найти их точные позиции, совместить и отфильтровать результаты (с учетом выравнивания).

Сложность алгоритма

- Поиск точного набора образцов
  - По времени: O(n + mS + t), где n – длина строки поиска, m – общая длина всех шаблонов, S – количество символов в алфавите, t – общая длина всех совпадений
  - По памяти: O(m), из-за построения бора и конечных ссылок
- Поиск с джокером
  - По времени: O(nq + t + mS), где q – количество частей шаблона без джокеров
  - По памяти: O(m)

Тестирование

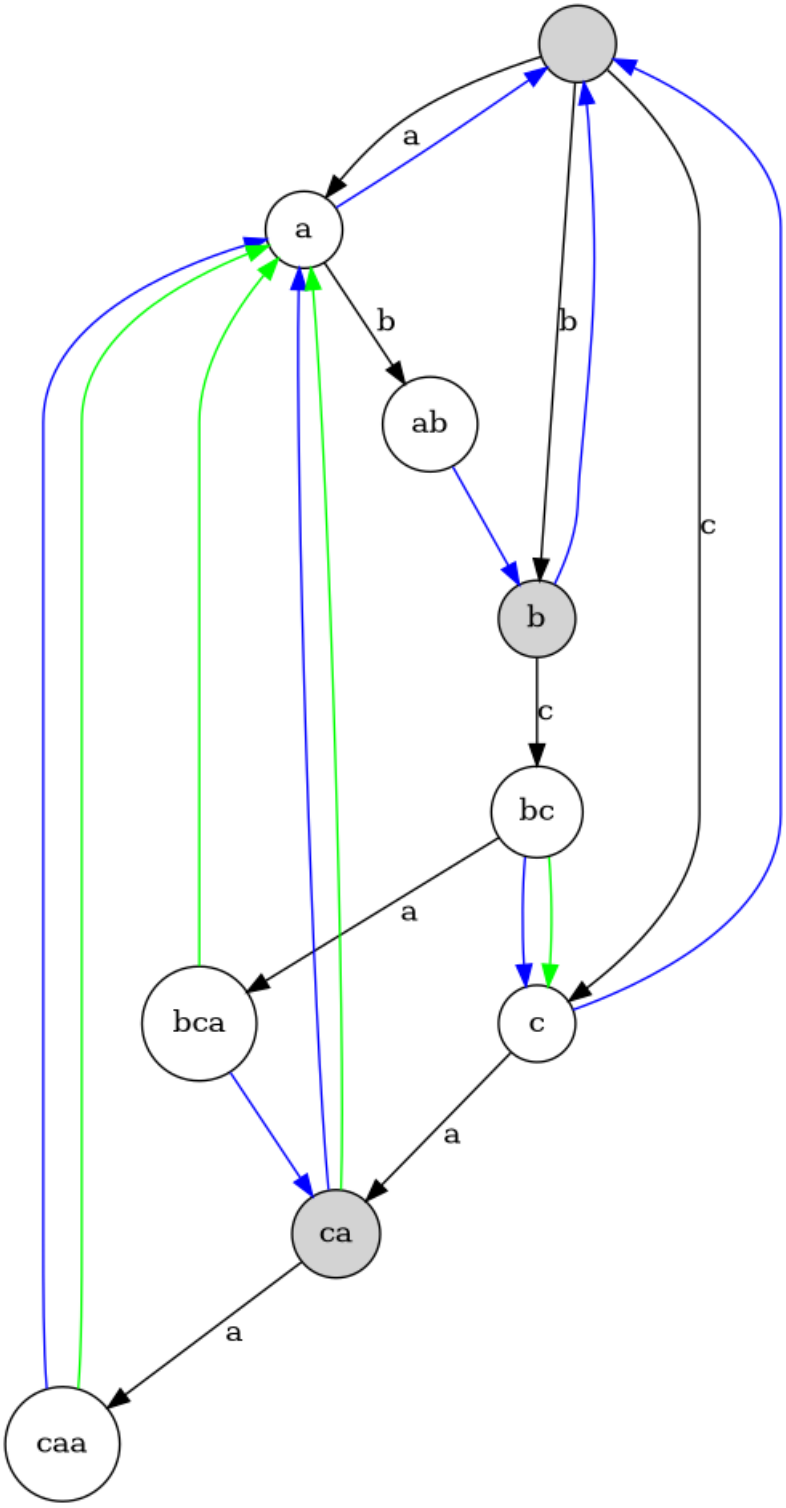

Рисунок 1 – Пример визуализации автомата

В приведенном примере зеленые стрелки – конечные ссылки, синие – суффиксные, черные – переходы. Серым выделены промежуточные вершины, белым – терминальные.

Входные данные	Выходные данные	Комментарий
abcabc 6 a ab bc bca c caa	1 1 1 2 2 3 2 4 3 5 4 1 4 2 5 3 6 5	Поиск набора шаблонов, успех
NTAG 3 TAGT TAG T	2 2 2 3	Поиск набора шаблонов, успех
ababcbababa 1 ab	1 1 3 1 7 1 9 1	Поиск набора шаблонов, успех
ACTANCA A$$A$ $	1	Поиск с джокером, успех
xabvccbababcax ab??c? ?	2 8	Поиск с джокером, успех
aaaaaa a? ?	1 2 3 4 5	Поиск с джокером, успех

Выводы.

Реализован алгоритм Ахо-Корасик для поиска набора подстрок в строке. Решена задача поиска шаблона с символами-джокерами в строке.

ПРИЛОЖЕНИЕ А

Файл main.cpp:

```
#include <algorithm>
#include <graphviz/gvc.h>
#include <iostream>
#include <map>
#include <queue>
#include <string>
#include <vector>

using namespace std;

struct Node {
    map<char, int> children;
    int suffixLink = -1;
    int outputLink = -1;
    int parent = -1;
    char parentChar = 0;
    vector<int> patternIndices;
};

class AhoCorasick {
private:
    vector<Node> trie;

    string buildLabel(int idx) {
        string label;
        while (idx > 0) {
            label = trie[idx].parentChar + label;
            idx = trie[idx].parent;
        }
        return label.empty() ? "" : label;
    }

    void checkMatches(int pos, int node, const vector<string>& patterns,
                      vector<pair<int, int>>& result) {
        cout << "Проверка совпадений в позиции " << pos << ", вершина " << node
             << endl;
        for (int temp = node; temp != -1; temp = trie[temp].outputLink) {
            cout << "  Переход к вершине " << temp << " (конечная ссылка)"
                 << endl;
            for (int pid : trie[temp].patternIndices) {
                int patternLength = patterns[pid].size();
                cout << "    Найден шаблон " << patterns[pid]
                     << " (индекс: " << pid + 1 << ") на позиции "
                     << pos - patternLength + 1 << endl;
                result.emplace_back(pos - patternLength + 1, pid + 1);
            }
        }
    }

public:
    AhoCorasick() {
        trie.emplace_back();  // root
    }

    void addPattern(const string& pattern, int index) {
        int node = 0;
        cout << "Добавление шаблона: " << pattern << endl;
        for (char ch : pattern) {
            if (!trie[node].children.count(ch)) {
                trie[node].children[ch] = trie.size();
```

```
            trie.emplace_back();
            trie.back().parent = node;
            trie.back().parentChar = ch;
            cout << "  Создан новый узел для символа '" << ch << "'"
                 << ", parent: " << node << endl;
        }
        node = trie[node].children[ch];
    }
    trie[node].patternIndices.push_back(index);
    cout << "  Шаблон " << pattern << " добавлен в вершину " << node
         << endl;
}

void buildAutomaton() {
    queue<int> q;
    trie[0].suffixLink = 0;
    trie[0].outputLink = -1;
    cout << "Строим автомат..." << endl;

    for (auto& it : trie[0].children) {
        int child = it.second;
        trie[child].suffixLink = 0;
        trie[child].outputLink = -1;
        q.push(child);
        cout << "  Инициализация ссылок для вершины " << child
             << " - suffix: " << trie[child].suffixLink
             << ", output: " << trie[child].outputLink << endl;
    }

    cout << "  Начинаем обход в ширину..." << endl;
    while (!q.empty()) {
        int current = q.front();
        cout << "  Обрабатываем вершину " << current << endl;
        q.pop();

        cout << "  Дети вершины " << current << ": ";
        for (auto& it : trie[current].children) {
            char ch = it.first;
            int child = it.second;

            cout << "   " << ch << "->" << child << " ";

            int fallback = trie[current].suffixLink;
            while (fallback != 0 && !trie[fallback].children.count(ch)) {
                fallback = trie[fallback].suffixLink;
            }
            if (trie[fallback].children.count(ch)) {
                fallback = trie[fallback].children[ch];
            }
            trie[child].suffixLink = fallback;

            int ol = (!trie[fallback].patternIndices.empty())
                         ? fallback
                         : trie[fallback].outputLink;
            trie[child].outputLink = (ol == child) ? -1 : ol;

            cout << "  Суффиксная ссылка для вершины " << child
                 << " установлена на вершину " << fallback
                 << ", конечная ссылка: " << trie[child].outputLink << endl;

            q.push(child);
        }
        cout << endl;
```

```
    }
}

void generateGraph() {
    GVC_t* gvc = gvContext();
    Agraph_t* g = agopen(const_cast<char*>("Aho-Corasick Automaton"),
                         Agdirected, nullptr);

    map<int, Agnode_t*> nodes;
    for (int i = 0; i < trie.size(); ++i) {
        string label = buildLabel(i);
        string color =
            trie[i].patternIndices.empty() ? "lightgray" : "white";
        Agnode_t* node = agnode(g, const_cast<char*>(label.c_str()), TRUE);
        agsafeset(node, const_cast<char*>("fillcolor"),
                  const_cast<char*>(color.c_str()), const_cast<char*>(""));
        agsafeset(node, const_cast<char*>("style"),
                  const_cast<char*>("filled"), const_cast<char*>(""));
        agsafeset(node, const_cast<char*>("shape"),
                  const_cast<char*>("circle"), const_cast<char*>(""));
        agsafeset(node, const_cast<char*>("label"),
                  const_cast<char*>(label.c_str()), const_cast<char*>(""));
        nodes[i] = node;
    }

    for (int i = 0; i < trie.size(); ++i) {
        for (auto it = trie[i].children.begin();
             it != trie[i].children.end(); ++it) {
            int child = it->second;
            string ch = string(1, it->first);
            Agedge_t* edge =
                agedge(g, nodes[i], nodes[child], nullptr, TRUE);
            agsafeset(edge, const_cast<char*>("label"),
                      const_cast<char*>(ch.c_str()), const_cast<char*>(""));
            agsafeset(edge, const_cast<char*>("color"),
                      const_cast<char*>("black"), const_cast<char*>(""));
        }
    }

    for (int i = 1; i < trie.size(); ++i) {
        if (trie[i].suffixLink != -1) {
            Agedge_t* edge = agedge(g, nodes[i], nodes[trie[i].suffixLink],
                                    nullptr, TRUE);
            agsafeset(edge, const_cast<char*>("color"),
                      const_cast<char*>("blue"), const_cast<char*>(""));
        }
    }

    for (int i = 1; i < trie.size(); ++i) {
        if (trie[i].outputLink != -1) {
            Agedge_t* edge = agedge(g, nodes[i], nodes[trie[i].outputLink],
                                    nullptr, TRUE);
            agsafeset(edge, const_cast<char*>("color"),
                      const_cast<char*>("green"), const_cast<char*>(""));
        }
    }

    gvLayout(gvc, g, "dot");
    gvRenderFilename(gvc, g, "png", "picture.png");
    gvFreeLayout(gvc, g);
    agclose(g);
    gvFreeContext(gvc);
}
```

```
void printAutomaton() {
    cout << "Состояния автомата:" << endl;
    for (int i = 0; i < trie.size(); ++i) {
        cout << "Вершина " << i << " (" << buildLabel(i) << ") ";
        cout << ", Родитель: " << trie[i].parent
             << ", Символ: " << trie[i].parentChar
             << ", Ссылка: " << trie[i].suffixLink
             << ", Конечная ссылка: " << trie[i].outputLink << ", Дети: ";
        for (auto& p : trie[i].children) {
            cout << p.first << "->" << p.second << " ";
        }
        if (!trie[i].patternIndices.empty()) {
            cout << ", Шаблоны: ";
            for (int pid : trie[i].patternIndices)
                cout << pid << " ";
        }
        cout << endl;
    }
}

vector<pair<int, int>> search(const string& text,
                              const vector<string>& patterns) {
    vector<pair<int, int>> result;
    int node = 0;

    cout << "Начинаем поиск в тексте..." << endl;
    for (int i = 0; i < text.size(); ++i) {
        char ch = text[i];
        cout << "  Прочитан символ '" << ch << "', текущий узел: " << node
             << endl;
        while (node != 0 && !trie[node].children.count(ch)) {
            node = trie[node].suffixLink;
            cout << "  Символ '" << ch
                 << "' не найден в текущей вершине, идем по ссылке "
                    "suffixLink к вершине "
                 << node << endl;
        }
        if (trie[node].children.count(ch)) {
            node = trie[node].children[ch];
            cout << "  Символ '" << ch << "' найден в вершине " << node
                 << endl;
        } else {
            cout << "  Символ '" << ch << "' не найден в вершине " << node
                 << ", остаемся на текущей вершине" << endl;
        }
        cout << "  Текущая вершина: " << node << endl;
        checkMatches(i + 1, node, patterns, result);
    }

    sort(result.begin(), result.end());
    return result;
}

vector<int> searchWithJoker(const string& text, const string& pattern,
                            vector<pair<string, int>>& parts) {
    vector<vector<int>> positions(parts.size());
    cout << "Поиск с джокерами в тексте: " << text
         << " для шаблона: " << pattern << endl;
    int node = 0;
    for (int i = 0; i < text.size(); ++i) {
        char ch = text[i];
        cout << "Обрабатываем символ текста '" << ch << "' в позиции " << i
```

```
            << endl;
        while (node != 0 && !trie[node].children.count(ch)) {
            node = trie[node].suffixLink;
            cout << "  Символ '" << ch
                 << "' не найден в текущей вершине, идем по ссылке "
                    "suffixLink к вершине "
                 << node << endl;
        }
        if (trie[node].children.count(ch)) {
            node = trie[node].children[ch];
            cout << "  Символ '" << ch << "' найден в вершине " << node
                 << endl;
        } else {
            cout << "  Символ '" << ch << "' не найден в вершине " << node
                 << ", остаемся на текущей вершине" << endl;
        }

        for (int temp = node; temp != -1; temp = trie[temp].outputLink) {
            cout << "  Проверяем выходную ссылку на вершину " << temp
                 << endl;
            for (int pid : trie[temp].patternIndices) {
                cout << "    Для шаблона " << parts[pid].first
                     << " (индекс: " << pid << "): " << endl;
                int plen = parts[pid].first.size();
                int pos = i - plen + 1;
                if (pos >= 0) {
                    cout << "  Найдено совпадение для части шаблона "
                         << parts[pid].first << " в позиции " << pos
                         << endl;
                    positions[pid].push_back(pos - parts[pid].second);
                } else {
                    cout << "    Совпадение невозможно (позиция < 0)"
                         << endl;
                }
            }
        }
    }

    cout << "Подсчет совпадений для каждой части шаблона..." << endl;
    map<int, int> matchCount;
    for (int i = 0; i < parts.size(); ++i) {
        for (int p : positions[i]) {
            matchCount[p]++;
            cout << "  Совпадение для части шаблона " << parts[i].first
                 << " в позиции " << p << endl;
        }
    }

    cout
        << "Фильтрация совпадений по количеству найденных частей шаблона..."
        << endl;
    vector<int> result;
    for (auto& match : matchCount) {
        int pos = match.first;
        int count = match.second;
        cout << "  Совпадение в позиции " << pos << ": найдено " << count
             << " частей шаблона" << endl;
        if (count == parts.size() && pos >= 0 &&
            pos + pattern.size() <= text.size()) {
            cout << "  Совпадение для позиции: " << pos + 1 << endl;
            result.push_back(pos + 1);
        }
    }
```

```
        return result;
    }
};

int main() {
    // Вариант 1: Обычный поиск
    string text;
    int n;
    cin >> text >> n;

    vector<string> patterns(n);
    AhoCorasick ac;

    for (int i = 0; i < n; ++i) {
        cin >> patterns[i];
        ac.addPattern(patterns[i], i);
    }

    cout << "Построение автомата..." << endl;
    ac.buildAutomaton();
    ac.printAutomaton();
    ac.generateGraph();

    cout << "Поиск в тексте..." << endl;
    auto matches = ac.search(text, patterns);

    cout << "Результаты:" << endl;
    for (auto& match : matches) {
        int pos = match.first;
        int pat = match.second;
        cout << pos << " " << pat << endl;
    }

    return 0;

    // Вариант 2: Поиск с джокерами
    // string text, pattern;
    // char joker;
    // cin >> text >> pattern >> joker;

    // vector<pair<string, int>> parts;
    // int start = -1;
    // for (int i = 0; i <= pattern.size(); ++i) {
    //     if (i < pattern.size() && pattern[i] != joker) {
    //         if (start == -1)
    //             start = i;
    //     } else {
    //         if (start != -1) {
    //             parts.emplace_back(pattern.substr(start, i - start), start);
    //             start = -1;
    //         }
    //     }
    // }

    // AhoCorasick ac;

    // if (parts.empty()) {
    //     return 0;
    // }

    // for (int i = 0; i < parts.size(); ++i) {
    //     ac.addPattern(parts[i].first, i);
    // }
```

```
    // ac.buildAutomaton();
    // ac.printAutomaton();
    // ac.generateGraph();

    // auto positions = ac.searchWithJoker(text, pattern, parts);
    // for (int position : positions) {
    //     cout << position << endl;
    // }

    // return 0;
}
```