МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студент гр. 7383 _____ Александров Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Познакомиться с алгоритмом Ахо-Корасик и его реализацией.

Реализация задачи.

В ходе работы были написаны класс AK, содержащий всю реализацию, в нем находится структура Trie, представляющая собой бор.

Метод void run() используется для консольного ввода, заполнения бора и старта поиска.

Метод void addStringToTrie(string &str) заполняет бор.

Метод void startSearch(string &str) начинает поиск.

Метод Trie makeTrie(int parent, char symb) добавляет вершины в бор.

Метод void checkAllSuffLinks(int vertex, int symb) выводит ответ в виде *i p*, где *i* – позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером *p* (нумерация образцов начинается с 1).

Метод int getAutoMove(int vertex, char symb) выполняет переход конечного автомата.

Метод int getSuffLink(int vertex) для вершины возвращает суффиксальную ссылку.

Метод void initArrays(int *array) заполняет массив значением -1.

Исследование алгоритма.

Количество операций считалось по вызову метода подстановки квадратов разных размеров от N-1 до 1. Результаты пре вычислительная сложность O(μn + H + k) где H — длина текста, в котором производится поиск, n — общая длина всех слов в словаре, μ — размер алфавита, k — общая длина всех совпадений. Результаты работы алгоритма представлены в табл. 1.

Таблица 1 - Результаты работы алгоритма

Вход	Выход
CCCA 1 CC	1 1 2 1
ACT A$ $	1

Выводы.

В ходе лабораторной работы был изучен и реализован алгоритм Ахо-Корасик.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

```
#include <iostream>
#include <vector>
#include <string>
#include <cstring>

using namespace std;

class AK {
    static const int defaultLength = 64;

    struct Trie {
        int vertices[defaultLength];
        int patternsCount;
        bool ifStr;
        int suffLink;
        int autoMove[defaultLength];
        int parent;
        char symbol;
    };
    vector<Trie> trie;
    vector<string> pattern;

    void initArrays(int *array) {
        for (int i = 0; i < defaultLength; i++) {
            array[i] = -1;
        }
    }

    int getSuffLink(int vertex) {
        if (trie[vertex].suffLink == -1) {
            if (vertex == 0 || trie[vertex].parent == 0) {
                trie[vertex].suffLink = 0;
            } else {
                trie[vertex].suffLink                                    =
getAutoMove(getSuffLink(trie[vertex].parent), trie[vertex].symbol);
            }
        }

        return trie[vertex].suffLink;
    }
```

```
    int getAutoMove(int vertex, char symb) {
        if (trie[vertex].autoMove[symb] == -1) {
            if (trie[vertex].vertices[symb] != -1) {
                trie[vertex].autoMove[symb]                                    =
trie[vertex].vertices[symb];
            } else if (vertex == 0) {
                trie[vertex].autoMove[symb] = 0;
            } else {
                trie[vertex].autoMove[symb]                                    =
getAutoMove(getSuffLink(vertex), symb);
            }
        }

        return trie[vertex].autoMove[symb];
    }


    void checkAllSuffLinks(int vertex, int symb) {
        for (int i = vertex; i != 0; i = getSuffLink(i)) {
            if (trie[i].ifStr) {
                cout << symb - pattern[trie[i].patternsCount].length() +
1 << " " << trie[i].patternsCount + 1 << endl;
            }
        }
    }

    Trie makeTrie(int parent, char symb) {
        Trie vertex;
        vertex.ifStr = false;
        vertex.suffLink = -1;
        vertex.parent = parent;
        vertex.symbol = symb;
        initArrays(vertex.vertices);
        initArrays(vertex.autoMove);

        return vertex;
    }

    void addStringToTrie(string &str) {
        int num = 0;
        for (int i = 0; i < str.length(); i++) {
            char ch = str[i] - 'A';
            if (trie[num].vertices[ch] == -1) {
                trie.push_back(makeTrie(num, ch));
```

```
                trie[num].vertices[ch] = trie.size() - 1;
            }
            num = trie[num].vertices[ch];
        }

        trie[num].ifStr = true;
        pattern.push_back(str);
        trie[num].patternsCount = pattern.size() - 1;
    }


    void startSearch(string &str) {
        int v = 0;
        for (int i = 0; i < str.length(); i++) {
            v = getAutoMove(v, str[i] - 'A');
            checkAllSuffLinks(v, i + 1);
        }
    }

public:
    AK() {
        trie.push_back(makeTrie(0, -1));
    }

    void run() {
        string mainText;
        int totalPatterns;
        cin >> mainText;
        cin >> totalPatterns;
        string pattern;
        for (int i = 0; i < totalPatterns; i++) {
            cin >> pattern;
            addStringToTrie(pattern);
        }

        startSearch(mainText);
    }
};

int main() {
    AK k;
    k.run();

    return 0;
```

```
}
#include <iostream>
#include <vector>
#include <sstream>
#include <string>
#include <cstring>

using namespace std;

class AK {
    static const int defaultLength = 64;
    struct Trie {
        int vertices[defaultLength];
        vector<int> patternCount;
        bool ifStr;
        int suffLink;
        int autoMove[defaultLength];
        int parent;
        char symbol;
    };
    vector<Trie> bohr;

    Trie makeTrie(int parent, char symb) {
        Trie vertex;
        vertex.ifStr = false;
        vertex.suffLink = -1;
        vertex.parent = parent;
        vertex.symbol = symb;
        initArrays(vertex.vertices);
        initArrays(vertex.autoMove);

        return vertex;
    }

    void initArrays(int *array) {
        for (int i = 0; i < defaultLength; i++) {
            array[i] = -1;
        }
    }

    void addStringToTrie(const string &str, vector<string> &pattern) {
        int num = 0;
        for (int i = 0; i < str.length(); i++) {
            char symb = str[i] - 'A';
```

```
            if (bohr[num].vertices[symb] == -1) {
                bohr.push_back(makeTrie(num, symb));
                bohr[num].vertices[symb] = bohr.size() - 1;
            }
            num = bohr[num].vertices[symb];
        }
        bohr[num].ifStr = true;
        pattern.push_back(str);
        bohr[num].patternCount.push_back(pattern.size() - 1);
    }


    int getSuffLink(int vertex) {
        if (bohr[vertex].suffLink == -1) {
            if (vertex == 0 || bohr[vertex].parent == 0) {
                bohr[vertex].suffLink = 0;
            } else {
                bohr[vertex].suffLink                                    =
getAutoMove(getSuffLink(bohr[vertex].parent), bohr[vertex].symbol);
            }
        }

        return bohr[vertex].suffLink;
    }

    int getAutoMove(int vertex, char symb) {
        if (bohr[vertex].autoMove[symb] == -1)
            if (bohr[vertex].vertices[symb] != -1)
                bohr[vertex].autoMove[symb]                              =
bohr[vertex].vertices[symb];
            else if (vertex == 0)
                bohr[vertex].autoMove[symb] = 0;
            else
                bohr[vertex].autoMove[symb]                              =
getAutoMove(getSuffLink(vertex), symb);
        return bohr[vertex].autoMove[symb];
    }

    void check(int vertex, int index, vector<int> &count, vector<int>
&length) {
        for (int u = vertex; u != 0; u = getSuffLink(u)) {
            if (bohr[u].ifStr) {
                for (auto &b : bohr[u].patternCount) {
                    if (index - length[b] < count.size()) {
```

```
                    count[index - length[b]]++;
                }
            }

        }
    }

    void startSearch(const string &s, vector<int> &count, vector<int>
&length) {
        int u = 0;
        for (int i = 0; i < s.length(); i++) {
            u = getAutoMove(u, s[i] - 'A');
            check(u, i + 1, count, length);
        }
    }

    vector<int> dividePattern(stringstream &patterns, char j,
vector<string> &pattern) {
        vector<int> length;
        int len = 0;
        string tmp;
        while (getline(patterns, tmp, j)) {
            if (tmp.size() > 0) {
                len += tmp.size();
                length.push_back(len);
                addStringToTrie(tmp, pattern);
            }
            len++;
        }

        return length;
    }


    void printAnswer(vector<int> &count, string mainTextSize,
vector<string> patSize) {
        for (int i = 0; i < mainTextSize.size(); i++) {
            if (count[i] == patSize.size())
                cout << i + 1 << endl;
        }
    }

public:
```

```
    AK() {
        bohr.push_back(makeTrie(0, -1));
    }

    void run() {
        string mainText;
        string neededPattern;
        char j;
        cin >> mainText;
        cin >> neededPattern;
        cin >> j;
        stringstream ssPatern(neededPattern);
        vector<int> count(mainText.size());
        vector<string> patt;
        vector<int> length = dividePattern(ssPatern, j, patt);
        startSearch(mainText, count, length);
        printAnswer(count, mainText, patt);
    }
};

int main() {
    AK k;
    k.run();

    return 0;
}
```