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. 8304 _____ Бутко А.М.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2020

Цель работы.

Реализовать алгоритм Ахо-Корасик, решить задачу точного поиска.

Постановка задачи.

(Вариант 2)

Используя реализацию точного множественного поиска, решить задачу точного поиска для одного образца с джокером.

Подсчитать количество вершин в автомате. Вывести список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска.

Реализация алгоритма.

В самом начале мы делим паттерн с джокерами на множество подстрок, заключенных между джокерами, в последствии мы им воспользуемся. Как и во всех задачах, сначала идет построение бора (Trie) — структуры данных, которая представляет собой дерево, на ребрах между вершинами которого написаны символы из алфавита, а конечные вершины — концы строк, которые можно составить при переходе из одной вершины в другую. Важное свойство бора — единственность, т.е. каждой конечной вершине соответствует ровно одна строка. Каждую подстроку поделенного паттерну мы добавляем в бор.

После того, как бор построен, строится конечный детерминированный автомат, по которому мы можем передвигаться с помощью функции getMove. Текст, в котором нужно найти паттерн с джокером обрабатывается посимвольно, для каждого символа мы передвигаемся по автомату, иначе возвращаемся на исходное положение. Если при передвижении мы дошли до конечной, терминальной вершины, то в векторе-счетчике шаблонов увеличиваем соответствующее значение. Если после очередного увеличения значения, оно стало равно количеству подстрок в изначальном шаблоне, то именно с этого значения начинается искомый паттерн.

Оценка сложности алгоритма.

Время выполнения алгоритма O(m + n + s + t), где m — длина текста, n — длина паттерна, s — размер вектора подстрок, t — размер вектора-счетчика шаблонов.

Описание структур данных и функций.

1)`const std::map<char, int> Alphabet` – словарь для хранения алфавита входных данных.

2)`std::vector<Vertex> Trie(1)` – вектор вершин бора; бор.

3)`std::vector<std::pair<std::string, int>> substring` – вектор подстрок с индексами вхождения в паттерне.

4)`struct Vertex` — структура для хранения информации о вершине и возможных переходах.

5)`void addString(std::string const& string, int strIndex, std::vector<Vertex>& Trie)` — функция добавления строки в бор.

6)`int getLink(int vertex, std::vector<Vertex>& Trie)` — функция получения суффиксальной ссылки.

7)`int getMove(int vertex, char symbol, std::vector<Vertex>& Trie)` — функция перехода из вершины в вершину.

8)`void searchIntersections(int indexString, const std::string& text, const std::string& string, int size, std::vector<int>& tempCounter)` — функция поиска и вывода пересечений строк, удовлетворяющих паттерну.

9)`void check(int vertex, int index, std::string& text, std::string& string,std::vector<Vertex>& Trie, std::vector<std::pair<std::string, int>>& substring, std::vector<int>& tempCounter)` — фунция проверки вершины на конечность, подсчета подстрок.

10)`void findAllPositions(std::string& text, std::string& string, std::vector<std::pair<std::string, int>>& substring, std::vector<Vertex>& Trie)` — функция перебора текста и поиска всех вхождений паттерна.

11)`void preparing(std::string& string, std::vector<Vertex>& Trie, std::vector<std::pair<std::string, int>>& substring, char joker)` — функция инициализации строк, вектора подстрок и пр.

Тестирование.

Ввод	Вывод
ACGCTCNCACGGCAA CC$ $	Quantity of vertexes in trie: 2 Pattern has been found at the 3 position. Pattern has been found at the 7 position. Intersection with string at the position 3 - |NCACG#GCTCN|;
AAACCCAAGACCAAACGTN AA$ $	Quantity of vertexes in trie: 3 Pattern has been found at the 1 position. Pattern has been found at the 2 position. Intersection with string at the position 1 - |AAC#AAA|; Pattern has been found at the 7 position. Pattern has been found at the 13 position. Pattern has been found at the 14 position. Intersection with string at the position 13 - |AAC#AAA|;
AAAAAAAAAAAAA CCA $	Quantity of vertexes in trie: 3

Вывод.

Была реализована структура данных бор, а так же был реализован алгоритм Ахо-Корасик.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

```
#include <iostream>
#include <vector>
#include <iterator>
#include <map>
#include <fstream>

/* Алфавит, используемый в автомате. */
const std::map<char, int> Alphabet = {{'A', 0}, {'C', 1}, {'G', 2}, {'T', 3}, {'N',
4}};

/* Переменные для логгирования и ввода/вывода. */
char in, out;
std::string path;
std::ofstream fout;

/* Структура, описывающий каждую вершину бора. */
struct Vertex
{
    std::vector<int> next = {-1, -1, -1, -1, -1};
    std::vector<int> move = {-1, -1, -1, -1, -1};
    bool isTerminate = false;
    int parent = -1;
    int link = -1;
    std::vector<int> sIndexes;
    char symbol;
    explicit Vertex(int p = -1, char ch = 0) : parent(p), symbol(ch){}
};

/* Функция добавления строки в бор. */
void addString(std::string const& string, int strIndex, std::vector<Vertex>& Trie)
{
    int vertex = 0;
    for (char symbol : string)
    {
        int index = Alphabet.at(symbol);
        if (Trie[vertex].next[index] == -1)
        {
            Trie[vertex].next[index] = Trie.size();
            Trie.emplace_back(vertex, symbol);
        }
        vertex = Trie[vertex].next[index];
    }
    Trie[vertex].isTerminate = true;
    Trie[vertex].sIndexes.push_back(strIndex);
}

/* Объявление функции для перехода по состояниям автомата. */
int getMove(int vertex, char symbol, std::vector<Vertex>& Trie);

/* Функция получения суффиксной ссылки. */
int getLink(int vertex, std::vector<Vertex>& Trie)
{
    if (Trie[vertex].link == -1)
    {
        if (vertex == 0 || Trie[vertex].parent == 0) Trie[vertex].link = 0;
        else Trie[vertex].link = getMove(getLink(Trie[vertex].parent, Trie),
Trie[vertex].symbol, Trie);
    }
    return Trie[vertex].link;
}
```

```
/* Функция перехода по состояниям автомата. */
int getMove(int vertex, char symbol, std::vector<Vertex>& Trie)
{
    int index = Alphabet.at(symbol);
    if (Trie[vertex].move[index] == -1)
    {
        if (Trie[vertex].next[index] != -1) Trie[vertex].move[index] =
Trie[vertex].next[index];
        else Trie[vertex].move[index] = vertex == 0 ? 0 : getMove(getLink(vertex,
Trie), symbol, Trie);
    }
    return Trie[vertex].move[index];
}

/* Функция вывода размера бора. */
void outputTrieSize(std::vector<Vertex>& Trie)
{
    if (out == 'c') std::cout << "Quantity of vertexes in trie: " << Trie.size() <<
std::endl;
    else
    {
        fout.open(path);
        fout << "Quantity of vertexes in trie: " << Trie.size() << std::endl;
    }
}

/* Функция вывода позиции паттерна в тексте. */
void outputTempIndex(int index)
{
    if (out == 'c') std::cout << "Pattern has been found at the " << index + 1 << "
position." << std::endl;
    else
    {
        fout.open(path);
        fout << "Pattern has been found at the " << index + 1 << " position." <<
std::endl;
    }
}
void outputUnderline()
{
    if (out == 'c') std::cout <<
"_____________________________________________________" << std::endl;
    else
    {
        fout.open(path);
        fout << "_____________________________________________________"  <<
std::endl;
    }
}

/* Функция поиска и вывода пересечения подстрок с паттерном в тексте. */
void searchIntersections(int indexString, const std::string& text, const std::string&
string, int size, std::vector<int>& tempCounter)
{
    for(int index = indexString + 1; index < indexString + string.size() - 1; ++index)
        if(tempCounter[index - string.size() + 1] == size)
        {
            if(out == 'c')
                std::cout << "Intersection with string at the position " << index -
string.size() + 2 << " - |"
                          << std::string(text.begin() + indexString, text.begin() +
indexString + string.size() - 1) << "#"
                          << std::string(text.begin() + index - string.size() + 1,
text.begin() + index) << "|;" << std::endl;
            else
```

```
            {
                fout.open(path);
                fout << "Intersection with string at the position " << index -
string.size() + 2 << " - |"
                     << std::string(text.begin() + indexString, text.begin() +
indexString + string.size() - 1) << "#"
                     << std::string(text.begin() + index - string.size() + 1,
text.begin() + index) << "|;" << std::endl;
            }
        }
}

/* Функция проверки вершины на конечность, подсчет индексов в счетчик. */
void check(int vertex, int index, std::string& text, std::string&
string,std::vector<Vertex>& Trie, std::vector<std::pair<std::string, int>>& substring,
std::vector<int>& tempCounter)
{
    int indexString = 0;
    for(int current = vertex; current != 0; current = getLink(current, Trie))
    {
        if(!Trie[current].isTerminate) continue;
        for (auto sIndex : Trie[current].sIndexes)
        {
            indexString += index + 2;
            indexString -= substring[sIndex - 1].first.length() + substring[sIndex -
1].second;
            if (indexString > -1) ++tempCounter[indexString];
            if (tempCounter[indexString] == substring.size())
            {
                outputTempIndex(indexString);
                searchIntersections(indexString, text, string,  substring.size(),
tempCounter);
                outputUnderline();
            }
        }
    }

}

/* Функция поиска всех вхождений. */
void findAllPositions(std::string& text, std::string& string,
std::vector<std::pair<std::string, int>>& substring, std::vector<Vertex>& Trie)
{
    int vertex = 0;
    std::vector<int> tempCounter(text.size(), 0);
    for(int index = 0; index < text.size(); ++index)
    {
        /* Получение номера вершины в боре. */
        vertex = getMove(vertex, text[index], Trie);
        check(vertex, index, text, string, Trie, substring, tempCounter);
    }
}

/* Подготовка вектора подстрок и строк для выполнения поиска.*/
void preparing(std::string& string, std::vector<Vertex>& Trie,
std::vector<std::pair<std::string, int>>& substring, char joker)
{
    string += joker;
    std::string temp;
    for (int i = 0; i < string.size(); ++i)
    {
        if (string[i] == joker)
        {
            if(!temp.empty())
            {
```

```
                substring.emplace_back(temp, i + 1 - temp.size());
                temp.clear();
            }
        }
        else temp += string[i];
    }
    for (int index = 0; index < substring.size(); ++index)
        addString(substring[index].first, index + 1, Trie);
}

/* Функция ввода данных в программу. */
bool input()
{
    std::string text;
    std::string string;
    char joker = 0;

    std::cout << "CHOOSE WHERE W/R DATA:" << std::endl;
    std::cout << "c - console;" << std::endl;
    std::cout << "f - file." << std::endl;
    std::cout << "READ DATA FROM" << std::endl;
    std::cin >> in;
    std::cout << "WRITE DATA TO" << std::endl;
    std::cin >> out;

    if((in != 'c' && in != 'f') || (out != 'c' && out != 'f'))
    {
        std::cout << "ERROR: WRONG INPUT";
        return false;
    }

    if(in == 'c') std::cin >> text >> string >> joker;
    else if(in == 'f')
    {
        getline(std::cin, path);
        std::ifstream fin(path);
        fin >> text >> string >> joker;
    }

    /* Бор, представленный как вектор вершин бора. */
    std::vector<Vertex> Trie(1);
    /* Вектор подстрок. */
    std::vector<std::pair<std::string, int>> substring;
    /* Инициализация вектора подстрок и подготовка остальных строк. */
    preparing(string, Trie, substring, joker);
    /* Вывод размера бора. */
    outputTrieSize(Trie);
    outputUnderline();
    /* Поиск всех вхождений. */
    findAllPositions(text, string, substring, Trie);
    return true;
}

int main()
{
    if(!input()) return 1;
}
```