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. 9383 _____ Камзолов Н.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Познакомиться на практике с алгоритмом Ахо-Корасика, реализовать данный алгоритм на языке программирования C++.

Задание.

1 задача:

Разработайте программу, решающую задачу точного поиска набора образцов.

Входные данные:

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 $P=\{p_1, ,\ldots,p_n\}$ $1 \leq |p_i| \leq 75$ Все строки содержат символы из алфавита {A,C,G,T,N}.

Выходные данные:

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..

2 задача:

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 PP 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}

Входные данные:

Текст (T, $1 \leq |T| \leq 100000$).

Шаблон (P, $1 \leq |P| \leq 40$).

Символ джокера

Выходные данные:

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

Номера должны выводиться в порядке возрастания.

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-Корасика — 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.

Описание алгоритма.

- ***Построение суффиксных ссылок:***

Для построения суффиксных ссылок следует применять правило:

- Суффиксная ссылка из корня или из сына корня ведет в корень.
- Для вычисления суффиксной ссылки вершины x нужно:
  1. Перейти к вершине-родителю.
  2. Пройти по суффиксным ссылками минимальное число раз, но не менее 1 раза, чтобы появился путь по ребру x, или до попадания в корень.

3. Пройти по ребру x. Если мы в корне и ребра x из него нет, то остаться в корне.

- **Построение конечных ссылок:**

Для построения конечной ссылки нужно переходить по суффиксным ссылкам до тех пор, пока не попадем в терминальную вершину. Если попали в корень, то конечная ссылка пуста.

- **Алгоритм Ахо-Корасик:**
  1. Построить бор из образцов.
  2. Построить автомат из бора.
  3. Перейти в корень бора.
  4. Посимвольная обработка текста. Для каждого символа:
     - Совершить переход в автомате из текущей вершины по рассматриваемому симолу. Если есть соответствующее ребро, то перейти по нему.
     - Если нет, то: если находимся в корне, то ничего не делать, если не в корне, то перейти по суффиксной ссылке и перейти к первому пункту.
     - Добавить в результат вхождение образца, если попали в конечную вершину.
     - Обойти цепочку конечных ссылок до конца, сохраняя результаты.
- **Алгоритм Ахо-Корасик с «джокером»:**
  1. Построить автомат Ахо-Корасик из образцов, полученных выделением максимальных бездженкерных подстрок из шаблонной подстроки.
  2. Для каждого образца записать смещение, по которому образец находится в шаблонной строке.
  3. Инициализровать массив, заполненный нулями, длинной совпадающий с текстом.
  4. Выполнить поиск по тексту с использованием автомата. При обнаружении образца инкрементировать ячейку массива по адресу, образованному разностью номера начального символа образца в тексте

и смещения образца. Если у образца несколько смещений, то инкрементировать все соотвествуюшие ячейки массива.

В результате щаблонная подстрока будет начинаться в тех местах текста, для которых соотвествующая ячейка массива содержит количество образцов с учетом кратности.

Оценка сложности по памяти и времени.

Пусть T – длина текста, в которой выполняется поиск, n – суммарная длина всех образцов, k – общая длина всех вхождений образцов в текст.

Сложность по времени O(T+k+nlogn), сложность по памяти O(n).

Описание функций и структур данных.

Node – структура данных, которая отвечает за вершину в автомате. Для первой задачи: хранит информацию о соседних вершинах, суффиксной и конченой ссылке, символе вершины, а также информация о том, является ли вершина терминальной, во второй задаче: хранит информацию о соседних вершинах, суффиксной ссылке, символе вершины и о смещении.

void input() – функция для считывания входных данных(разнится в зависимости от задачи).

void wordToTrie() – функция для преобразования строки в бор(одинакова для обеих задач).

void findSuffixLink() – функция, которая для конкретной вершины, строит суффиксные и конечные ссылки(в зависимости от задачи здесь инициализируются некоторые поля Node. Для первой задачи: информация о суффиксной и конечной ссылке, а также является ли вершина терминальной. Для второй: информация о суффиксный ссылке и смещении).

void suffixFunc() – функция, которая обходит бор с помощью алгоритма BFS и запускает для каждой вершины функцию findSuffixLink(одинакова для обеих задач).

void splitString() – функция, которая нужна только для второй задачи. Разделяет подстроку с джокерами на максимальные подстроки без джокеров.

void printAnswer() – функция, которая выводит ответ на задачу(разнится в зависимости от задачи).

std::vector<int> AhoCorasick() – функция, реализующая алгоритм Ахо-Корасик(разнится в зависимости от задачи, реализует алгоритмы, написанные в блоке «Описание алгоритма»).

Тестирование(Задание 1).

Входные данные	Выходные данные
NTAG 3 TAGT TAG T	2 2 2 3
NTCACGCAC 3 TAG TC AC	2 2 4 3 8 3
AAA 2 AA A	1 1 1 2 2 1 2 2 3 2
AAAAAAAN 1 N	8 1

Тестирование(Задание 2).

Входные данные	Выходные данные
ACTANKA A??A? ?	1
ANNGCTTCG A******** *	1
ACATA A?A ?	1 3
ACT * *	1 2 3

Выводы.

На практике были применены теоретические знания об алгоритме Ахо-Корасик. Была успешно написана программа, реализующая данный алгоритм. Данный алгоритм отлично подходит для нахождения нескольких подстрок в строке, а также хорошо работает для поиска вхождения подстроки с «джокером» (универсальный символ, который заменяет собой любую букву алфавита) в строку.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл lab5_1/main.cpp:

```
#include "aho_corasick.h"

int main() {
    std::string text;
    Node* root = new Node;
    root->suffixLink = root;
    input(text, root, std::cin);
    suffixFunc(root);

    std::vector<std::pair<int, int>> ans = ahoCorasik(text, root);
    printAnswer(ans);
}
```

Файл lab5_2/main.cpp:

```
#include "aho_corasick.h"

int main() {
    std::string text, pattern;
    char joker;
    int count = 0;
    Node* root = new Node;
    root->suffixLink = root;
    input(text, pattern, joker, std::cin);
    splitString(root, pattern, joker, count);
    suffixFunc(root);
    std::vector<int> ans = ahoCorasik(text, root, count, pattern.size());
    printAnswer(ans);
}
```

Файл lab5_1/aho_corasick.cpp:

```
#include "aho_corasick.h"

bool comparator(std::pair<int, int> first, std::pair<int, int> second) {
    if (first.first != second.first) {
        return first.first < second.first;
    } else {
        return first.second < second.second;
    }
}



void findSuffixLink(Node* node, Node* root, char symbol) {
    if(node->suffixLink == root) {
        return;
    }
```

```
    Node* tempSuffix = node->suffixLink->suffixLink;
    bool isChanged = false;
    while(true) {
        for(auto& vertex: tempSuffix->next) {
            if(vertex.second->symbol == symbol) {
                if(!isChanged) {
                    node->suffixLink = vertex.second;
                    isChanged = true;
                }
                if(vertex.second->isTerminal) {
                    node->endLink = vertex.second;
                    return;
                }

            }
        }
        if(tempSuffix == root)
            break;
        tempSuffix = tempSuffix->suffixLink;
    }
    if(!isChanged) {
        node->suffixLink = root;
    }
}


void suffixFunc(Node* root) {
    std::queue<Node*> tempQueue;
    Node* curNode = root;

    for(auto& node : curNode->next) {
        tempQueue.push(node.second);
    }

    while(!tempQueue.empty()) {
        curNode = tempQueue.front();
        tempQueue.pop();
        findSuffixLink(curNode, root, curNode->symbol);
        for (auto &node : curNode->next) {
            tempQueue.push(node.second);
        }
    }
}

void wordToTrie(Node* root, std::string& word, int number) {

    Node* tempNode = root;
    for(int i = 0; i < word.size(); i++) {
        if(tempNode->next.count(word[i]) == 0)
            tempNode->next[word[i]] = new Node;
        tempNode->next[word[i]]->suffixLink = tempNode;
        tempNode = tempNode->next[word[i]];
        tempNode->symbol = word[i];
        if(i == word.size() - 1) {
            tempNode->isTerminal = true;
            tempNode->wordSize = word.size();
            tempNode->stringNumbers.push_back(number);
```

```
        }
    }
}



void input(std::string& text, Node* root, std::istream& in) {

    in >> text;

    int n;
    in >> n;

    std::string tempString;
    for(int i = 0; i < n; i++) {
        in >> tempString;
        wordToTrie(root, tempString, i + 1);
    }

}



std::vector<std::pair<int, int>> ahoCorasik(std::string& text, Node*
root) {
    std::vector<std::pair<int, int>> answer;
    Node* tempNode = root;
    for(size_t i = 0; i < text.size(); i++) {
        if(tempNode->next.count(text[i]) == 0) {
            while(tempNode != root && tempNode->next.count(text[i]) == 0)
{
                tempNode = tempNode->suffixLink;
                if(tempNode->next.count(text[i]) != 0) {
                    tempNode = tempNode->next[text[i]];
                    break;
                }
            }
            if (tempNode == root) continue;
        }
        else {
            tempNode = tempNode->next[text[i]];
        }


        if(tempNode->isTerminal == true) {
           for(int j = 0; j < tempNode->stringNumbers.size(); j++)
                answer.push_back(std::make_pair(i - tempNode->wordSize +
2, tempNode->stringNumbers[j]));
        }

        Node* tempEndNode = tempNode->endLink;
        while(tempEndNode) {
            for(int j = 0; j < tempEndNode->stringNumbers.size(); j++)
                answer.push_back(std::make_pair(i - tempEndNode->wordSize
+ 2, tempEndNode->stringNumbers[j]));
            tempEndNode = tempEndNode->endLink;
        }
```

```
    }
    return answer;
}


void printAnswer(std::vector<std::pair<int, int>>& ans) {

    std::sort(ans.begin(), ans.end(), comparator);

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

Файл lab5_2/aho_corasick.cpp:

```
#include "aho_corasick.h"


void findSuffixLink(Node* node, Node* root, char symbol) {
    if(node->suffixLink == root) {
        return;
    }
    Node* tempSuffix = node->suffixLink->suffixLink;
    bool isChanged = false;
    while(true) {
        for(auto& vertex: tempSuffix->next) {
            if(vertex.second->symbol == symbol) {
                if(!isChanged) {
                    node->suffixLink = vertex.second;
                    isChanged = true;
                }
            }
        }
        if(tempSuffix == root)
            break;
        tempSuffix = tempSuffix->suffixLink;
    }
    if(!isChanged) {
        node->suffixLink = root;
    }
}


void suffixFunc(Node* root) {
    std::queue<Node*> tempQueue;
    Node* curNode = root;

    for(auto& node : curNode->next) {
        tempQueue.push(node.second);
    }

    while(!tempQueue.empty()) {
        curNode = tempQueue.front();
        tempQueue.pop();
        findSuffixLink(curNode, root, curNode->symbol);
        for (auto &node : curNode->next) {
```

```
            tempQueue.push(node.second);
        }
    }
}

void wordToTrie(Node* root, std::string& word, int offset) {
    Node* tempNode = root;
    for(size_t i = 0; i < word.size(); i++) {
        if(tempNode->next.count(word[i]) == 0)
            tempNode->next[word[i]] = new Node;
        tempNode->next[word[i]]->suffixLink = tempNode;
        tempNode = tempNode->next[word[i]];
        tempNode->symbol = word[i];
    }
    tempNode->offsets.push_back(offset);
}



void input(std::string& text, std::string& pattern, char& joker,
std::istream& in) {

    in >> text;
    in >> pattern;
    in >> joker;

}



std::vector<int> ahoCorasik(std::string& text, Node* root, int count, int
patternSize) {
    std::vector<int> answer;
    std::vector<int> matchCounter(text.size());
    Node* tempNode = root;

    for(size_t i = 0; i < text.size(); i++) {
        if(tempNode->next.count(text[i]) == 0) {
            while(tempNode != root && tempNode->next.count(text[i]) == 0)
{
                tempNode = tempNode->suffixLink;
                if(tempNode->next.count(text[i]) != 0) {
                    tempNode = tempNode->next[text[i]];
                    break;
                }
            }
            if (tempNode == root) continue;
        }
        else {
            tempNode = tempNode->next[text[i]];
        }

        Node* tempSufNode = tempNode;
        while(tempSufNode != root) {
            for (auto offset : tempSufNode->offsets) {
                if(i - offset >= 0 && i - offset < matchCounter.size()) {
                    matchCounter[i - offset]++;
```

```
                }
            }
            tempSufNode = tempSufNode->suffixLink;
        }
    }

    for(size_t i = 0; i < matchCounter.size(); i++) {
        if(matchCounter[i] == count && i + patternSize <=
matchCounter.size()) {
            answer.push_back(i + 1);
        }
    }

    return answer;
}


void printAnswer(std::vector<int>& ans) {
    std::sort(ans.begin(), ans.end());
    for(auto& elem : ans) {
        std::cout << elem << '\n';
    }
}


void splitString(Node* root, std::string& pattern, char joker, int&
count) {
    std::string tempString;
    int offset = 0;
    for(size_t i = 0; i < pattern.size(); i++) {
        if(pattern[i] == joker) {
            if(!tempString.empty()){
                wordToTrie(root, tempString, offset - 1);
                count++;
            }
            tempString.clear();
        }
        else {
            tempString.push_back(pattern[i]);
        }
        offset++;
    }

    if(!tempString.empty()){
        wordToTrie(root, tempString, offset - 1);
        count++;
    }
}
```

Файл lab5_1/aho_corasick.h:

```
#ifndef AHO_KORASICK_H
#define AHO_KORASICK_H

#include <iostream>
#include <string.h>
#include <map>
```

```
#include <vector>
#include <algorithm>
#include <queue>


struct Node {
    std::map<char, Node*> next;
    Node* suffixLink = nullptr;
    Node* endLink = nullptr;
    bool isTerminal = false;
    char symbol = '\0';
    int stringNumber = -1;
    std::vector<int> stringNumbers;
    int wordSize = 0;
};

bool comparator(std::pair<int, int> first, std::pair<int, int> second);
void findSuffixLink(Node* node, Node* root, char symbol);
void suffixFunc(Node* root);
void wordToTrie(Node* root, std::string& word, int number);
void input(std::string& text, Node* root, std::istream& in);
std::vector<std::pair<int, int>> ahoCorasik(std::string& text, Node*
root);
void printAnswer(std::vector<std::pair<int, int>>& ans);

#endif
```

Файл lab5_2/aho_corasick.h:

```
#ifndef AHO_KORASICK_H
#define AHO_KORASICK_H

#include <iostream>
#include <string.h>
#include <map>
#include <vector>
#include <algorithm>
#include <queue>
#include <array>

struct Node {
    std::map<char, Node*> next;
    Node* suffixLink = nullptr;
    char symbol = '\0';
    std::vector<int> offsets;
};

void findSuffixLink(Node* node, Node* root, char symbol);
void suffixFunc(Node* root);
void wordToTrie(Node* root, std::string& word, int offset);
void input(std::string& text, std::string& pattern, char& joker,
std::istream& in);
std::vector<int> ahoCorasik(std::string& text, Node* root, int count, int
patternSize);
void printAnswer(std::vector<int>& ans);
```

```
void splitString(Node* root, std::string& pattern, char joker, int&
count);

#endif
```