**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. 7383 |  | Лосев М.Л. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2018

## Постановка задачи.

**Цель работы:** реализация и исследование алгоритма Ахо-Корасик, получение опыта и знания по его использованию.

**Формулировка задачи:** реализовать алгоритм Ахо-Корасик и с его помощью для **n (1 ≤ n ≤ 3000)** заданных шаблонов **Pi (|Pi | ≤ 75)** и текста **T** **(1 ≤ |T |≤ 100000)** найти все вхождения **Pi** в **T**.

**Входные данные:**

Первая строка содержит текст **(T, 1 ≤ |T| ≤ 100000T, 1 ≤ |T| ≤ 100000)**. Вторая - число **n (1 ≤ n ≤ 3000)**, каждая следующая из **n** строк содержит шаблон из набора **P = {p1, …, pn}, 1 ≤ |pi| ≤ 75, P = {p1, …, pn}, 1 ≤ |pi| ≤ 75**.

**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.

**Описание алгоритма:**

Сначала для набора шаблонов строится префиксное дерево(бор). Далее в дерево добавляются суффиксные ссылки проходом по суффиксным ссылкам родителя и поиска ребра, помеченного тем же символом, что и ребро из родителя в ребенка. Таким образом строится конечный автомат. Автомату передается текст для поиска всех шаблонов. Начиная с корня обходим дерево, переходя по ребрам, помеченным считанным из строки символом. При переходе проверяется, является ли данное состояние конечным и ищутся конечные вершины по суффиксным ссылкам. Для конечных вершин высчитывается положение начала вхождения шаблона в строку и записывается в ответ.

**Реализация**

Был использован следующие классы:

Node – узел бора;

Tire – бор;

AhoCorasick – public-класс, который инициализируется набором шаблонов и ищет их вхождения в текст с помощью метода run.

## Исследование

Пусть **m** – суммарная длина шаблонов (общая длина всех слов в словаре), **n** – длина текста. Если таблицу переходов автомата хранить как [индексный массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) – расход памяти **O(m σ)**{\displaystyle O(n\sigma )}Щ, потому что в боре может быть до **m** узлов, и каждый хранит таблицу переходов автомата размера **σ**. Вычислительная сложность {\displaystyle O(n\sigma +H+k)} **O(mσ + n + k)**, где{\displaystyle \sigma } **σ** – размер алфавита, **{\displaystyle k}k** – общая длина всех совпадений. **O(mσ)** – сложность построения бора, **O(n)** – сложность всех переходов по автомату, **O(k)** – сложность добавления найденных вхождений в решение.

# **Тестирование.**

Тестирование проводилось в ОС Ubuntu 18.04 с помощью IDE intellij idea. Кроме того, программа прошла тесты на Stepic. Результаты тестирования показали, что программа работает корректно.

## Вывод.

При выполнении работы был реализован и изучен Ахо-Корасика. Его вычислительная сложность в худшем случае есть **O(mσ + n + k)**,а затраты памяти **O(mσ).**

**ПРИЛОЖЕНИЕ А  
Код программы**

import java.util.Scanner;

import javafx.util.Pair;

import java.util.StringTokenizer;

import java.util.Vector;

import java.util.Set;

import java.util.HashSet;

class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String haystack = scanner.nextLine();

int n = Integer.parseInt(scanner.nextLine()); // better than nextInt(), skips the rest of the line

String[] needles = new String[n];

for (int i = 0; i < n; i++) {

needles[i] = scanner.nextLine();

}

//AhoCorasick test #1:

AhoCorasick ahoCorasick = new AhoCorasick(needles, haystack);

while (ahoCorasick.hasMoreElements()) {

Pair <Integer, Integer> occurrence = ahoCorasick.nextOccurrence();

System.out.println((occurrence.getKey() + 1) + " " + (occurrence.getValue() + 1));

}

//AhoCorasick test #2:

/\*

AhoCorasick ahoCorasick = new AhoCorasick(needles);

ahoCorasick.run(haystack);

while (ahoCorasick.hasMoreElements()) {

Pair <Integer, Integer> occurrence = ahoCorasick.nextOccurrence();

System.out.println((occurrence.getKey() + 1) + " " + (occurrence.getValue() + 1));

}

\*/

//JOKER TEST:

/\*

String haystack = scanner.nextLine();

String needle = scanner.nextLine();

char joker = scanner.nextLine().toCharArray()[0]; // input the joker

searchWithJoker(needle, haystack, joker);

\*/

}

private static void searchWithJoker(String needle, String haystack, char joker) {

StringTokenizer st = new StringTokenizer(needle, "" + joker);

String[] needles = new String[st.countTokens()];

int[] startPositions = new int[st.countTokens()];

int j = 0, behind = 0;

while (st.hasMoreTokens()) {

needles[j] = st.nextToken();

startPositions[j] = needle.indexOf(needles[j], behind);

behind = startPositions[j] + needles[j].length();

j++;

}

AhoCorasick ahoCorasick = new AhoCorasick(needles, haystack);

int[] countOccurances = new int[haystack.length()];

while (ahoCorasick.hasMoreElements()) {

Pair <Integer, Integer> occurrence = ahoCorasick.nextOccurrence();

int pos = occurrence.getKey();

int currWordNum = occurrence.getValue();

if (pos - startPositions[currWordNum] < 0) continue;

countOccurances[pos- startPositions[currWordNum]]++;

if (countOccurances[pos - startPositions[currWordNum]] == needles.length &&

pos - startPositions[currWordNum] <= haystack.length() - needle.length())

System.out.println(pos - startPositions[currWordNum] + 1);

}

}

}

class Node {

private int[] edges; // i'th edge edges[i] is a son with the edge i from the node (i is a character)

private Node[] transits;

private Node link; // suffix link

private boolean isFinal;

private Set<Integer> wordNum = new HashSet<>();

private Node parent;

private char incomingEdge;

Node() { // root

isFinal = false;

edges = new int[Character.MAX\_VALUE];

transits = new Node[Character.MAX\_VALUE];

parent = this;

incomingEdge = (char)0;

link = this;

}

Node(Node parent, char edge, Node root) {

isFinal = false;

this.parent = parent;

incomingEdge = edge;

edges = new int[Character.MAX\_VALUE];

transits = new Node[Character.MAX\_VALUE];

link = root;

}

boolean isFinal() {

return isFinal;

}

Node getParent() {

return parent;

}

int getChildIndex(char letter) {

return edges[letter];

}

void addEdge(char letter, int childIndex) {

edges[letter] = childIndex;

}

Node getTransitIndex(char letter) {

return transits[letter];

}

void setTransitIndex(char letter, Node node) {

this.transits[letter] = node;

}

void setLink(Node link) {

this.link = link;

}

Node getLink () {

return link;

}

char getIncomingEdge() {

return incomingEdge;

}

void makeFinal() {

isFinal = true;

}

void setWordNum(int wordNum) {

this.wordNum.add(wordNum);

}

Set<Integer> getWordNum() {

return wordNum;

}

}

class Trie {

Vector<Node> nodes; // there can up to m = |words| of them

Trie(String[] words) {

nodes = new Vector<>();

nodes.add(new Node()); // add root

int i = 0;

for (String word: words) {

addWord(word);

nodes.get(nodes.size() - 1).makeFinal();

nodes.get(nodes.size() - 1).setWordNum(i);

i++;

}

}

private void addWord(String word) {

Node currNode = nodes.get(Character.MIN\_VALUE); // root

Node nextNode;

for (int i = 0; i < word.length(); i++) {

char currEdge = word.charAt(i);

//System.out.println("Current edge: " + currEdge);

if (currNode.getChildIndex(currEdge) < nodes.size()) {

nextNode = nodes.get(currNode.getChildIndex(currEdge));

} else {

nextNode = nodes.get(0);

}

if (nextNode == nodes.get(0)) { // if next is root (that happens only when there is no edge currEdge)

nextNode = new Node(currNode, currEdge, nodes.get(0));

currNode.addEdge(currEdge, nodes.size());

nodes.add(nextNode);

}

currNode = nextNode;

}

}

Node getSuffixLink(Node node) {

if (node.getLink() == nodes.get(0)) { // not defined yet or node is the root or its son

if (node == nodes.get(0) || node.getParent() == nodes.get(0)) { // node is the root or its son

node.setLink(nodes.get(0)); // do nothing

} else { // node is not the root or its son

node.setLink(transitFunction(node.getParent().getLink(), node.getIncomingEdge()));

}

}

return node.getLink();

}

Node transitFunction(Node node, char letter) {

//if (node == null) return nodes.get(0);

if (node.getTransitIndex(letter) == null) node.setTransitIndex(letter, nodes.get(0));

if (node.getTransitIndex(letter) == nodes.get(0)) {

if (node.getChildIndex(letter) != 0) {

node.setTransitIndex(letter, nodes.get(node.getChildIndex(letter))); // sigma(node, letter) = v, if v is a son of node by letter

}

else if (node == nodes.get(0)) { // if node == root

node.setTransitIndex(letter, nodes.get(0)); // sigma(node, letter) = 0, if node is the root

}

else {

node.setTransitIndex(letter, transitFunction(getSuffixLink(node), letter));

}

}

return node.getTransitIndex(letter);

}

}

class AhoCorasick {

private String[] needles;

private Trie tire;

private Vector<Pair<Integer, Integer>> solution;

private int elementCounter;

public AhoCorasick(String[] needles) { // no running

this.needles = needles;

tire = new Trie(needles);

//tire.print(); // for debugging

}

public AhoCorasick(String[] needles, String haystack) {

this.needles = needles;

tire = new Trie(needles);

run(haystack);

//tire.print(); // for debugging

}

private void check(Node v, int i) {

for (Node u = v; u != tire.nodes.get(0); u = tire.getSuffixLink(u)) {

if (u.isFinal()) {

for (int currWordNum: u.getWordNum()) {

solution.add(new Pair<>(i - needles[currWordNum].length(), currWordNum));

}

}

}

}

public void run(String haystack){

solution = new Vector<>();

Node u = tire.nodes.get(0);

for(int i = 0; i < haystack.length(); i++) {

u = tire.transitFunction(u, haystack.charAt(i));

check(u,i + 1);

}

}

public int countOccurrences() {

return solution == null ? 0 : solution.size();

}

public boolean hasMoreElements() {

return solution != null && elementCounter < solution.size();

}

public Pair<Integer, Integer> nextOccurrence() {

if (hasMoreElements())

return solution.get(elementCounter++);

return null;

}

public Object nextElement() {

if (hasMoreElements())

return solution.get(elementCounter++);

return null;

}

}