**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 |  | Сергеев А.Д. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2020

**Цель работы.**

Изучить алгоритм Ахо-Корасик для оптимального поиска всех вхождений данных подстрок в строку.

**Задание.**

1. Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст .

Вторая — число , каждая следующая из строк содержит шаблон из набора .

Все строки содержат символы из алфавита .

Выход:

Все вхождения образцов из в .

Каждое вхождение образца в текст представить в виде двух чисел - .

Где - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером (нумерация образцов начинается с 1).

Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

1. 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 необходимо найти все вхождения в .

Например, образец с джокером встречается дважды в тексте.

Символ джокер не входит в алфавит, символы которого используются в . 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 .

Вход:

Текст .

Шаблон .

Символ джокера

Выход:

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

Номера должны выводиться в порядке возрастания.

Дополнительное задание (вариант 4): Реализовать режим поиска, при котором все найденные образцы не пересекаются в строке поиска (т.е. некоторые вхождения не будут найдены; решение задачи неоднозначно).

**Порядок выполнения работы.**

Написание работы производилось на базе операционной системы Windows 10 на языке программирования java в среде программирования IntelliJ IDEA.

Обе программы состоят из трёх классов похожей структуры.

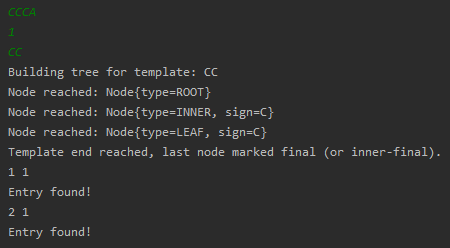
Класс Node содержит описание узла бора и методы работы с ним. Узел содержит указатель на родительский узел, указатель на узел, на который указывает суффиксная ссылка из этого узла (но сама ссылка строится во время выполнения программы), массив узлов-наследников и тип узла. Типов узла всего 4 — корневой, внутренний, конечный и внутренний-конечный (то есть, у узла есть наследники, но один из данных шаблонов на нём заканчивается). Также класс Node в первой программе содержит номер шаблона, для которого он является последним (он устанавливается, только когда узел становится конечным или внутренним-конечным), а во второй — статическую переменную типа char, в которой хранится символ джокера.

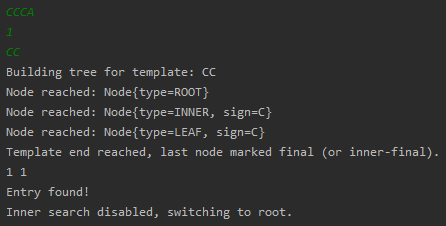
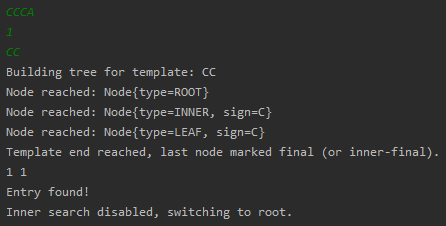
Класс AhoCorasick представляет из себя сам алгоритм в виде функций для работы с бором. Метод build строит бор, принимая в качестве параметров в первой программе массив шаблонов, а во второй — единственный шаблон и символ джокер. Метод solve запускает и выполняет сам алгоритм. Он принимает строку поиска, а также логическую переменную, указывающую, следует ли отсекать пересекающиеся вхождения. Метод printNode выводит на экран информацию о найденной подстроке и её позицию.

Описание классов в UML-виде приложено к отчеты в файлах UML.F.png и UML.S.png.

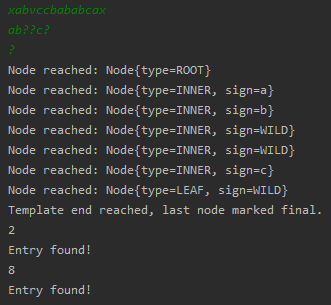
**Тестирование.**

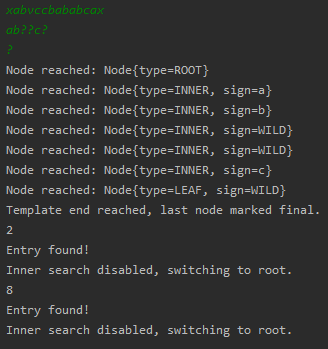
Тестирование первой программы:

Рисунок 1 - Вывод первой программы с учётом пересекающихся вхождений

Рисунок 2 - Вывод первой программы без учёта пересекающихся вхождений

Тестирование второй программы:

Рисунок 3 - Вывод второй программы с учётом пересекающихся вхождений

Рисунок 4 - Вывод второй программы без учёта пересекающихся вхождений

Из рисунков видно, что вхождения не пересекаются.

**Вывод.**

В результате лабораторной работы были изучены алгоритм Ахо-Корасик для поиска вхождений нескольких подстрок в строку.

**Приложение А**

**Исходный код программы, файл first/Main.java**

import java.util.ArrayList;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String source = sc.nextLine();

int number = Integer.parseInt(sc.nextLine());

ArrayList<String> marks = new ArrayList<>();

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

marks.add(sc.nextLine());

}

AhoCorasick ac = new AhoCorasick(5);

ac.build(marks);

ac.solve(source, false);

}

}

**Приложение Б**

**Исходный код программы, файл first/Node.java**

import java.util.ArrayList;

class Node {

enum NodeType {ROOT, INNER, INNER\_LEAF, LEAF}

NodeType type;

Integer leafNumber;

Character sign;

Node parent;

ArrayList<Node> children;

Node def\_link;

public Node(char sign, Node parent, int alphbetLength) {

this.sign = sign;

this.parent = parent;

if (parent.type == NodeType.ROOT) def\_link = parent;

this.type = NodeType.INNER;

this.children = new ArrayList<>(alphbetLength);

}

public Node(int alphbetLength) {

this.type = NodeType.ROOT;

this.children = new ArrayList<>(alphbetLength);

}

public void setLeaf(int leafNumber) {

this.leafNumber = leafNumber;

if (this.type == NodeType.INNER)

if (this.children.isEmpty()) this.type = NodeType.LEAF;

else this.type = NodeType.INNER\_LEAF;

}

public void add(Node child) {

this.children.add(child);

if (this.type == NodeType.LEAF) this.type = NodeType.INNER\_LEAF;

}

public Node getDefLink() {

if (def\_link == null) {

def\_link = parent.getDefLink().pass(sign);

}

return def\_link;

}

public Node get(char letter) {

for (Node child : children) {

if (child.sign == letter) return child;

}

return null;

}

private Node pass(char letter) {

for (Node child : children) {

if (child.sign == letter) return child;

}

if (type == Node.NodeType.ROOT) return this;

else return getDefLink().pass(letter);

}

@Override

public String toString() {

return "Node{" +

"type=" + type +

((type == NodeType.ROOT) ? "" : ", sign=" + sign) +

'}';

}

}

**Приложение В**

**Исходный код программы, файл first/AhoCorassick.java**

import java.util.List;

public class AhoCorasick {

int alphabetSize;

List<String> marks;

Node root;

public AhoCorasick(int size) {

this.alphabetSize = size;

}

public void build(List<String> marks) {

this.marks = marks;

root = new Node(alphabetSize);

for (int i = 0; i < marks.size(); i++) {

System.out.println("Building tree for template: " + marks.get(i));

Node current = root;

for (char ch : marks.get(i).toCharArray()) {

System.out.println("Node reached: " + current.toString());

Node next = current.get(ch);

if (next == null) {

next = new Node(ch, current, alphabetSize);

current.add(next);

}

current = next;

}

current.setLeaf(i);

System.out.println("Node reached: " + current.toString());

System.out.println("Template end reached, last node marked final (or inner-final).");

}

}

public void solve(String src, boolean searchForInner) {

char [] source = src.toCharArray();

Node current = root;

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

Node next = current.get(source[i]);

int counter = 0;

while ((next == null) && (current != root)) {

counter--;

current = current.getDefLink();

next = current.get(source[i]);

if (printNode(current, i + counter)) {

System.out.println("Entry found!");

if (!searchForInner) {

System.out.println("Inner search disabled, switching to root.");

current = root;

}

}

}

if (next != null) current = next;

if (printNode(current, i)) {

System.out.println("Entry found!");

if (!searchForInner) {

System.out.println("Inner search disabled, switching to root.");

current = root;

}

}

}

}

private boolean printNode(Node node, int pos) {

if ((node.type == Node.NodeType.LEAF) || (node.type == Node.NodeType.INNER\_LEAF)) {

System.out.println((pos+2 - marks.get(node.leafNumber).length()) + " " + (node.leafNumber+1));

return true;

}

return false;

}

}

**Приложение Г**

**Исходный код программы, файл** **second/Main.java**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String source = sc.nextLine();

String proto = sc.nextLine();

char wild = sc.nextLine().charAt(0);

AhoCorasick ac = new AhoCorasick(5);

ac.build(proto, wild);

ac.solve(source, true);

}

}

**Приложение Д**

**Исходный код программы, файл second/Node.java**

import java.util.ArrayList;

class Node {

public static char wild;

enum NodeType {ROOT, INNER, INNER\_LEAF, LEAF}

NodeType type;

Character sign;

Node parent;

ArrayList<Node> children;

Node def\_link;

public Node(char sign, Node parent, int alphbetLength) {

this.sign = sign;

this.parent = parent;

if (parent.type == NodeType.ROOT) def\_link = parent;

this.type = NodeType.INNER;

this.children = new ArrayList<>(alphbetLength);

}

public Node(int alphbetLength) {

this.type = NodeType.ROOT;

this.children = new ArrayList<>(alphbetLength);

}

public void setLeaf() {

if (this.type == NodeType.INNER)

if (this.children.isEmpty()) this.type = NodeType.LEAF;

else this.type = NodeType.INNER\_LEAF;

}

public void add(Node child) {

this.children.add(child);

if (this.type == NodeType.LEAF) this.type = NodeType.INNER\_LEAF;

}

public Node getDefLink() {

if (def\_link == null) {

def\_link = parent.getDefLink().pass(sign);

}

return def\_link;

}

public Node get(char letter) {

for (Node child : children) {

if ((child.sign == letter) || (child.sign == wild)) return child;

}

return null;

}

private Node pass(char letter) {

for (Node child : children) {

if ((child.sign == letter) || (child.sign == wild)) return child;

}

if (type == Node.NodeType.ROOT) return this;

else return getDefLink().pass(letter);

}

@Override

public String toString() {

return "Node{" +

"type=" + type +

((type == NodeType.ROOT) ? "" : ", sign=" + ((sign == wild) ? "WILD" : sign)) +

'}';

}

}

**Приложение Е**

**Исходный код программы, файл second/AhoCorassick.java**

public class AhoCorasick {

int alphabetSize;

String mark;

Node root;

public AhoCorasick(int size) {

this.alphabetSize = size;

}

public void build(String mark, char wild) {

Node.wild = wild;

this.mark = mark;

root = new Node(alphabetSize);

Node current = root;

for (char ch : mark.toCharArray()) {

System.out.println("Node reached: " + current.toString());

Node next = current.get(ch);

if (next == null) {

next = new Node(ch, current, alphabetSize);

current.add(next);

}

current = next;

}

current.setLeaf();

System.out.println("Node reached: " + current.toString());

System.out.println("Template end reached, last node marked final.");

}

public void solve(String src, boolean searchForInner) {

char [] source = src.toCharArray();

Node current = root;

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

Node next = current.get(source[i]);

int counter = 0;

while ((next == null) && (current != root)) {

counter--;

current = current.getDefLink();

next = current.get(source[i]);

if (printNode(current, i + counter)) {

System.out.println("Entry found!");

if (!searchForInner) {

System.out.println("Inner search disabled, switching to root.");

current = root;

}

}

}

if (next != null) current = next;

if (printNode(current, i)) {

System.out.println("Entry found!");

if (!searchForInner) {

System.out.println("Inner search disabled, switching to root.");

current = root;

}

}

}

}

private boolean printNode(Node node, int pos) {

if ((node.type == Node.NodeType.LEAF) || (node.type == Node.NodeType.INNER\_LEAF)) {

System.out.println((pos+2 - mark.length()));

return true;

}

return false;

}

}