1.A String Operations

public class StringManipulation {

public static String concatenateAndExtract(String str1, String str2, int length) {

String concatenated = str1.concat(str2)

String reversed = "";

for (int i = concatenated.length() - 1; i >= 0; i--) {

char currentChar = concatenated.charAt(i);

reversed = reversed.concat(Character.toString(currentChar));

}

int middleIndex = reversed.length() / 2;

int startIndex = middleIndex - length / 2;

int endIndex = startIndex + length;

return reversed.substring(startIndex, endIndex);

}

public static void main(String[] args) {

String str1 = "Hello";

String str2 = "World";

int desiredLength = 4;

String result = concatenateAndExtract(str1, str2, desiredLength);

System.out.println("Result: " + result);

}

}

2.A Naïve Search Pattern

public class NaiveSearch {

public static void search(String pat, String txt) {

int l1 = pat.length();

int l2 = txt.length();

for (int i = 0; i <= l2 - l1; i++) {

int j;

for (j = 0; j < l1; j++) {

if (txt.charAt(i + j) != pat.charAt(j)) {

break;

}

}

if (j == l1) {

System.out.println("Pattern found at index " + i);

}

}

}

public static void main(String[] args) {

String txt1 = "AABAACAADAABAABA";

String pat1 = "AABA";

System.out.println("Example 1:");

search(pat1, txt1);

String txt2 = "agd";

String pat2 = "g";

System.out.println("\nExample 2:");

search(pat2, txt2);

}

}

3.A KMP Algorithm

public class KMPAlgorithm {

private static int[] computeLPS(String pat) {

int m = pat.length();

int[] lps = new int[m];

int len = 0; // Length of the previous longest prefix suffix

for (int i = 1; i < m; ) {

if (pat.charAt(i) == pat.charAt(len)) {

len++;

lps[i] = len;

i++;

} else {

if (len != 0) {

len = lps[len - 1];

} else {

lps[i] = 0;

i++;

}

}

}

return lps;

}

public static void KMPSearch(String pat, String txt) {

int m = pat.length();

int n = txt.length();

int[] lps = computeLPS(pat);

int i = 0; // Index for txt[]

int j = 0; // Index for pat[]

while (i < n) {

if (pat.charAt(j) == txt.charAt(i)) {

i++;

j++;

}

if (j == m) {

System.out.println("Pattern found at index " + (i - j));

j = lps[j - 1];

} else if (i < n && pat.charAt(j) != txt.charAt(i)) {

if (j != 0) {

j = lps[j - 1];

} else {

i++;

}

}

}

}

public static void main(String[] args) {

String txt = "THIS IS A TEST TEXT";

String pat = "TEST";

KMPSearch(pat, txt);

}

}

4.A Rabin-Karp Sub String Search

import java.util.ArrayList;

import java.util.List;

public class RabinKarpSubstringSearch {

private static final int PRIME\_MOD = 101;

private static final int BASE = 256;

public static List<Integer> search(String text, String pattern) {

List<Integer> occurrences = new ArrayList<>();

int patternHash = hash(pattern);

for (int i = 0; i <= text.length() - pattern.length(); i++) {

if (patternHash == hash(text.substring(i, i + pattern.length()))) {

if (text.substring(i, i + pattern.length()).equals(pattern)) {

occurrences.add(i);

}

}

}

return occurrences;

}

private static int hash(String str) {

int hash = 0;

for (char c : str.toCharArray()) {

hash = (hash \* BASE + c) % PRIME\_MOD;

}

return hash;

}

public static void main(String[] args) {

String text = "THIS IS A TEST TEXT";

String pattern = "TEST";

List<Integer> occurrences = search(text, pattern);

if (!occurrences.isEmpty()) {

System.out.println("Pattern found at indices: " + occurrences);

} else {

System.out.println("Pattern not found in the text.");

}

}

}

5.A Boyer Moore Algorithm

public class BoyerMooreSubstringSearch {

public static int findLastOccurrence(String text, String pattern) {

int n = text.length();

int m = pattern.length();

int[] badChar = new int[256];

for (int i = 0; i < 256; i++) {

badChar[i] = -1; // Initialize all characters as not found

}

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

badChar[pattern.charAt(i)] = i;

}

int i = m - 1; // Start from the end of the pattern

while (i < n) {

int j = m - 1; // Compare characters from right to left

while (j >= 0 && text.charAt(i) == pattern.charAt(j)) {

i--;

j--;

}

if (j == -1) {

return i + 1; // Return the index of the last occurrence

} else {

int shift = Math.max(1, j - badChar[text.charAt(i)]);

i += shift;

}

}

return -1; // Pattern not found

}

public static void main(String[] args) {

String text = "THIS IS A TEST TEXT";

String pattern = "TEST";

int lastIndex = findLastOccurrence(text, pattern);

if (lastIndex != -1) {

System.out.println("Last occurrence found at index: " + lastIndex);

} else {

System.out.println("Pattern not found in the text.");

}

}

}