

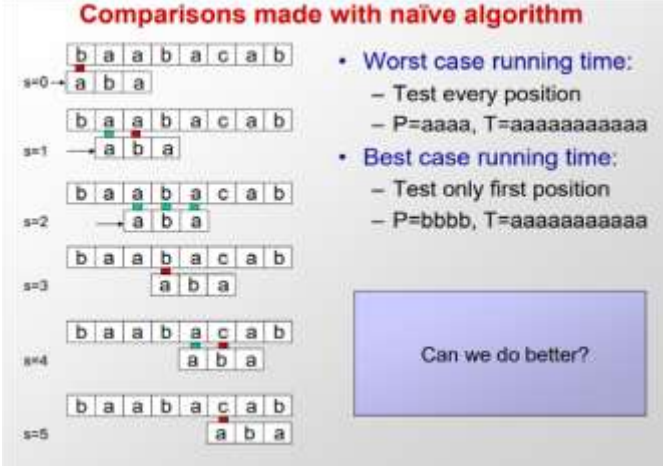


Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

Name	Pratik Pujari		
UID no.	2020300054	Class:	Comps C Batch
Experiment No.	10		

AIM:	Given a universe U of n elements, a collection of subsets of U say $S = \{S_1, S_2, \dots, S_m\}$ where every subset S_i has an associated cost. Find a minimum cost subcollection of S that covers all elements of U.
THEORY:	<p>NAÏVE STRING MATCHING</p> <p>The naïve approach tests all the possible placement of Pattern P [1.....m] relative to text T [1.....n]. We try shift $s = 0, 1, \dots, n-m$, successively and for each shift s. Compare T [s+1.....s+m] to P [1.....m]. The naïve algorithm finds all valid shifts using a loop that checks the condition $P [1.....m] = T [s+1.....s+m]$ for each of the $n - m + 1$ possible value of s.</p> <p>NAIVE-STRING-MATCHER (T, P)</p> <ol style="list-style-type: none"> 1. $n \leftarrow \text{length} [T]$ 2. $m \leftarrow \text{length} [P]$ 3. for $s \leftarrow 0$ to $n - m$ 4. do if $P [1.....m] = T [s + 1.....s + m]$ 5. then print "Pattern occurs with shift" s <div style="text-align: center;">  <p>Comparisons made with naïve algorithm</p> <ul style="list-style-type: none"> • Worst case running time: <ul style="list-style-type: none"> – Test every position – $P = \text{aaaa}$, $T = \text{aaaaaaaaaaaa}$ • Best case running time: <ul style="list-style-type: none"> – Test only first position – $P = \text{bbbb}$, $T = \text{aaaaaaaaaaaa}$ <p>Can we do better?</p> </div>



**Computer Engineering Department &
Information Technology Engineering Department**

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

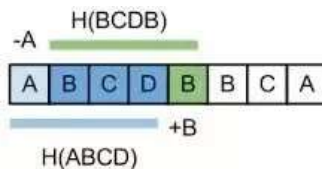
	<p>CALCULATION OF VALUE OF H IN TC $O(\log(M))$. HINT MODULAR EXPONENTIATION</p> <p>Modular exponentiation is exponentiation performed over a modulus.</p> <p>Modular exponentiation is the remainder when an integer b (the base) is raised to the power e (the exponent), and divided by a positive integer m (the modulus); i.e., $c = b^e \text{ mod } m$.</p> <p>It is $O(\log(m))$ because the value is calculated as the sum of geometric progression depending on the value of d and m along with using the modulus with q.</p> <p>ROLLING HASH:</p> <p>Consider this example. Let's say you have a string "abcdeacdoe" & you want to find the pattern "bcd" in this string.</p> <p>Now with the naive way, you will try to compare each character of pattern ("bcd") with the string characters ("abcdbbca") forming three-character strings. This might be really inefficient especially as the input gets larger. If you use rolling hash, you can do this intelligently.</p> <p>First you calculate the hash of first three letter substring (abc) in the string. To keep matters simple, let's say we use base 7 for this calculation (in the actual scenario we should mod it with a prime to avoid overflow):</p> <p>substring1 = $a*7^0 + b*7^1 + c*7^2 = 97*1 + 98*7 + 99*49 = 5634$ pattern = $b*7^0 + c*7^1 + d*7^2 = 98*1 + 99*7 + 100*49 = 5691$</p> <p>So you compare two hash values and since they are different, you move forward. Now you reach to second substring "bcd" in string. Here is where the fun begins. We can calculate the hash of this string without rehashing everything. As you can see, the window has moved forward only by dropping one character and adding another: a<-bc<-d</p> <p>Below diagrams explains it better (though it's working on a four-letter string rather than three):</p>
--	---



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA



** credits: quora*

From hash value we drop the first character value, divide the left out value with the base number and add the new char to the rightmost position with appropriate power of base: Here are the steps:

drop a $\Rightarrow (a \cdot 7^0 + b \cdot 7^1 + c \cdot 7^2) - a \cdot 7^0$

divide everything by 7 $\Rightarrow (b \cdot 7^1 + c \cdot 7^2) / 7 \Rightarrow b \cdot 7^0 + c \cdot 7^1$

add d $\Rightarrow b \cdot 7^0 + c \cdot 7^1 + d \cdot 7^2$ [the power of base for d is (pattern-length-1)]

Thus new hash for "bcd" in input string would be $\Rightarrow (5634 - 97) / 7 + 100 \cdot 49 = 5691$ which matches pattern hash value. Now we can go ahead and compare the strings to verify if they are in fact same.

Visualize doing this for a large string which comprising of lets say, 30 lines. You will be able to get new hash value in constant time without much effort by just dropping and adding new character(s).

In real use cases, to avoid overflow because of large power, we will have to do modulo with large prime. This technique is often used for multi-pattern string search and forms the core of algorithms like the Karp-Rabin algorithm. This algorithm is an excellent choice to detect plagiarism.

WHY ARE PRIME NUMBERS USED FOR CONSTRUCTING HASH FUNCTIONS?

The reason why prime numbers are used is to minimize collisions when the data exhibits some particular patterns. First things first: If the data is random then there's no need for a prime number, you can do a mod operation against any number and you will have the same number of collisions for each possible value of the modulus. But when data is not random then strange things happen. For example, consider numeric data that is always a multiple of 10.



**Computer Engineering Department &
Information Technology Engineering Department**

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

	<p>If we use mod 4 we find: $10 \bmod 4 = 2$ $20 \bmod 4 = 0$ $30 \bmod 4 = 2$ $40 \bmod 4 = 0$ $50 \bmod 4 = 2$ <i>So from the 3 possible values of the modulus (0, 1, 2, 3), only 0 and 2 will have collisions, that is bad.</i> If we use a prime number like 7: $10 \bmod 7 = 3$ $20 \bmod 7 = 6$ $30 \bmod 7 = 2$ $40 \bmod 7 = 4$ $50 \bmod 7 = 1$ We also note that 5 is not a good choice but 5 is prime the reason is that all our keys are a multiple of 5. This means we have to choose a prime number that doesn't divide our keys, choosing a large prime number is usually enough. So erring on the side of being repetitive the reason prime numbers are used is to neutralize the effect of patterns in the keys in the distribution of collisions of a hash function.</p> <p>THE RABIN-KARP ALGORITHM Like the Naive Algorithm, the Rabin-Karp algorithm also slides the pattern one by one. But unlike the Naive algorithm, the Rabin Karp algorithm matches the hash value of the pattern with the hash value of the current substring of text, and if the hash values match then only it starts matching individual characters. So Rabin Karp algorithm needs to calculate hash values for the following strings.</p> <ol style="list-style-type: none">1. Pattern itself.2. All the substrings of the text of length m. <p>Since we need to efficiently calculate hash values for all the substrings of size m of text, we must have a hash function that has the following property. Hash at the next shift must be efficiently computable from the current hash value and next character in text or we can say $\text{hash}(\text{txt}[s+1 \dots s+m])$ must be efficiently computable from $\text{hash}(\text{txt}[s \dots s+m-1])$ and $\text{txt}[s+m]$ i.e., $\text{hash}(\text{txt}[s+1$</p>
--	--



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

.. s+m])= rehash(txt[s+m], hash(txt[s .. s+m-1])) and rehash must be O(1) operation.
The hash function suggested by Rabin and Karp calculates an integer value. The integer value for a string is the numeric value of a string.

```
Input: txt[] = "THIS IS A TEST TEXT"
       pat[] = "TEST"
```

```
Output: Pattern found at index 10
```

```
Input: txt[] = "AABAACAADAABAABA"
       pat[] = "AABA"
```

```
Output: Pattern found at index 0
        Pattern found at index 9
        Pattern found at index 12
```

Text : A A B A A C A A D A A B A A B A

Pattern : A A B A

A A B A	A A B A
A A B A A C A A D A A B A A B A	
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
	A A B A

Pattern Found at 0, 9 and 12

*credits: documentation

Limitations of the Rabin-Karp Algorithm

Spurious Hit

When the hash value of the pattern matches with the hash value of a window of the text but the window is not the actual pattern then it is called a spurious hit. Spurious hit increases the time complexity of the algorithm. In order to minimize spurious hits, we use modulus. It greatly reduces the spurious hit.



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

PSEUDOCODE:	<pre>n = t.length m = p.length h = dm-1 mod q p = 0 t0 = 0 for i = 1 to m p = (dp + p[i]) mod q t0 = (dt0 + t[i]) mod q for s = 0 to n - m if p = ts if p[1.....m] = t[s + 1..... s + m] print "pattern found at position" s If s < n-m ts + 1 = (d (ts - t[s + 1]h) + t[s + m + 1]) mod q</pre>
EXPERIMENT 10	
CODE:	<pre>import java.util.ArrayList; import java.util.Scanner; public class rabinKarp { static ArrayList<Integer> locations = new ArrayList<Integer>(); // d is the number of characters in the input alphabet public final static int static void search(String pat, String txt, int q) { int d = 256; int M = pat.length(); int N = txt.length(); int i, j; int p = 0; // hash value for pattern int t = 0; // hash value for txt int h = 1; // The value of h would be "pow(d, M-1)%q" for (i = 0; i < M - 1; i++) h = (h * d) % q; // Calculate the hash value of pattern and first</pre>



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

	<pre>// window of text for (i = 0; i < M; i++) { p = (d * p + pat.charAt(i)) % q; t = (d * t + txt.charAt(i)) % q; } // Slide the pattern over text one by one for (i = 0; i <= N - M; i++) { // Check the hash values of current window of text // and pattern. If the hash values match then only // check for characters on by one if (p == t) { /* Check for characters one by one */ for (j = 0; j < M; j++) { if (txt.charAt(i + j) != pat.charAt(j)) break; } // if p == t and pat[0...M-1] = txt[i, i+1,...i+M-1] if (j == M) { locations.add(i); System.out.println("Pattern found at index " + i); } } // Calculate hash value for next window of text: Remove // leading digit, add trailing digit if (i < N - M) { t = (d * (t - txt.charAt(i) * h) + txt.charAt(i + M)) % q; // We might get negative value of t, converting it // to positive if (t < 0) t = (t + q); } } /* Driver program to test above function */ public static void main(String[] args) { Scanner sc = new Scanner(System.in); System.out.print("\nEnter the text: ");</pre>
--	---



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

	<pre>String txt = sc.next(); System.out.print("\nEnter the pattern: "); String pat = sc.next(); int q = 101; // A prime number search(pat, txt, q); System.out.print("\nPattern found at following locations: \n"); for (int i = 0; i < txt.length(); i++) { System.out.print(txt.charAt(i)); } System.out.print("\n"); for (int i = 0; i < txt.length() && locations.size() != 0; i++) { if (i == locations.get(0)) { System.out.print("^"); locations.remove(0); } else System.out.print(" "); } sc.close(); }</pre>
OUTPUT:	<pre>Enter the text: aabadjwdaaab Enter the pattern: aab Pattern found at index 0 Pattern found at index 9 Pattern found at following locations: aabadjwdaaab ^ ^</pre>



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

```
Enter the text: helloworldworlhelloworld
```

```
Enter the pattern: ellow
```

```
Pattern found at index 1
```

```
Pattern found at index 15
```

```
Pattern found at following locations:
```

```
helloworldworlhelloworld
```

```
^                ^
```

Rabin-Karp Algorithm Pratik Pujari

<p>Text 1 2 3 4 5 6 7 8 9 10 11</p> <p style="margin-left: 40px;">c c a c c a a e d b a</p> <p style="margin-left: 100px;">1 2 3</p> <p>Pattern - 'd b a'</p> <p style="margin-left: 40px;">c c a</p> $3 \times 10^2 + 3 \times 10^1 + 10^0$ $= 331$ <p style="margin-left: 40px;">c c a</p> $= 313 - 300 = 13 \times 10 + 3$ $= 133$ <p>∴ Continuing with further pairs matching at index 9</p> <p>No spurious hits</p> <p style="text-align: center;">$O(m+n+1) \rightarrow$ Average</p> <p>In case of spurious hit - $O(m \times n)$</p> <p>if value is exceeding the datatype using modulo.</p>	<p style="margin-left: 40px;">d b a</p> $4 \times 10^2 + 2 \times 10^1 + 10^0$ $= 421$ <p style="margin-left: 40px;">a a c c</p> $= (3 \times 10^1 + 1 \times 10 + 3 \times 10^0) - 3 \times 10^2 \times 10$ $= 313$ <p style="margin-left: 40px;">d d a</p>
---	--



**Computer Engineering Department &
Information Technology Engineering Department**

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

**TIME
COMPLEXITY:**

Time complexity Analysis Pratik Pujari

Hence call hash $[s[1 \dots m]]$
for index from 1 to $n-m+1$.
// code to check if substring
// matches
call hash $[index+1 \dots index+m]$

Compared to $O(m \cdot n)$, additive $O(m)$
is largely ignored
Giving $O(m) + O(n) + O(m) = O(mn)$

Time complexity is $O(m+n)$ for
cases when:-

hash $(s[1 \dots m])$
index from 1 to $n-m+1$
// code to check if substring matches
call hash $(s[index+1 \dots index+m])$
// which takes $O(1)$, applies rolling hash

Giving $O(m) + O(n) + O(1)$
 $= O(m) + O(n)$
 $= O(m+n)$

Assume the text is length n and the length of the word is m . The **best- and average-case** running time of Rabin-Karp is **$O(m+n)$** because the rolling hash step takes $O(n)$ time and once the algorithm finds a potential match, it must verify each letter to make sure that the match is true



Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech Sem.: 4 Course: DAA

	The worst-case running time of Rabin-Karp is $O(nm)$. This would occur with an extremely awful hash function that resulted in a false positive at each step.
<p>CONCLUSION: Things learnt during the procedural programming of the problem</p> <ul style="list-style-type: none">• Learnt the concept related to most of the String matching algorithms.• Implemented Rabin Karp algorithm where the hash value of the text and the pattern is calculated and found out all the answers to additional questions assigned.• Learnt how to find out the best and worst case time complexity.• Also learnt how to solve Rabin Karp problems on paper.	