Subject :- ADSAA **Semester :- V**

Module-6

String Matching

Introduction

String matching algorithms have greatly influenced computer science and play an essential role in various real-world problems. It helps in performing time-efficient tasks in multiple domains. These algorithms are useful in the case of searching a string within another string. String matching is also used in the Database schema, Network systems.
Let us look at a few string matching algorithms before proceeding to their applications in real world. String Matching Algorithms can broadly be classified into two types of algorithms –

1. Exact String Matching Algorithms
2. Approximate String Matching Algorithms

Exact String Matching Algorithms:
Exact string matching algorithms is to find one, several, or all occurrences of a defined string (pattern) in a large string (text or sequences) such that each matching is perfect. All alphabets of patterns must be matched to corresponding matched subsequence. These are further classified into four categories:

1. Algorithms based on character comparison:
    - Naive Algorithm: It slides the pattern over text one by one and check for a match. If a match is found, then slides by 1 again to check for subsequent matches.
    - KMP (Knuth Morris Pratt) Algorithm: The idea is whenever a mismatch is detected, we already know some of the characters in the text of the next window. So, we take advantage of this information to avoid matching the characters that we know will anyway match.

- Boyer Moore Algorithm: This algorithm uses best heurestics of Naive and KMP algorithm and starts matching from the last character of the pattern.
- Using the Trie data structure: It is used as an efficient information retrieval data structure. It stores the keys in form of a balanced BST.

2. Deterministic Finite Automaton (DFA) method:
   - Automaton Matcher Algorithm: It starts from the first state of the automata and the first character of the text. At every step, it considers next character of text, and look for the next state in the built finite automata and move to a new state.
3. Algorithms based on Bit (parallelism method):
   - Aho-Corasick Algorithm: It finds all words in O(n + m + z) time where n is the length of text and m be the total number characters in all words and z is total number of occurrences of words in text. This algorithm forms the basis of the original Unix command fgrep.
4. Hashing-string matching algorithms:
   - Rabin Karp Algorithm: It matches the hash value of the pattern with the hash value of current substring of text, and if the hash values match then only it starts matching individual characters.

Approximate String Matching Algorithms:

Approximate String Matching Algorithms (also known as Fuzzy String Searching) searches for substrings of the input string. More specifically, the approximate string matching approach is stated as follows: Suppose that we are given two strings, text T[1…n] and pattern P[1…m]. The task is to find all the occurrences of patterns in the text whose edit distance to the pattern is at most k. Some well known edit distances are – Levenshtein edit distance and Hamming edit distance.

The Naive String Matching Algorithm

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.....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.

NAIVE-STRING-MATCHER (T, P)

1. n ← length [T]
2. m ← length [P]
3. for s ← 0 to n -m
4. do if P [1.....m] = T [s + 1....s + m]
5. then print "Pattern occurs with shift" s

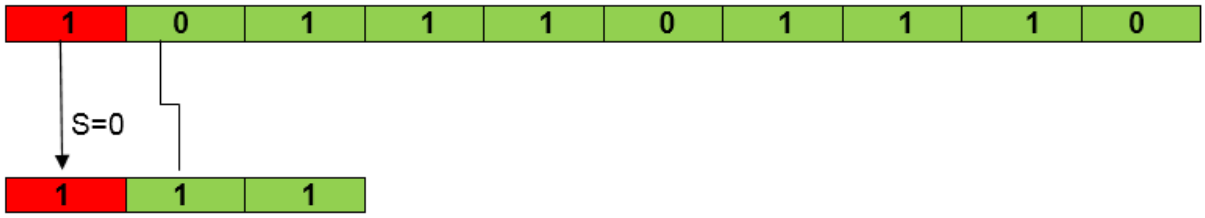
Example:

1. Suppose T = 1011101110
2. P = 111
3. Find all the Valid Shift

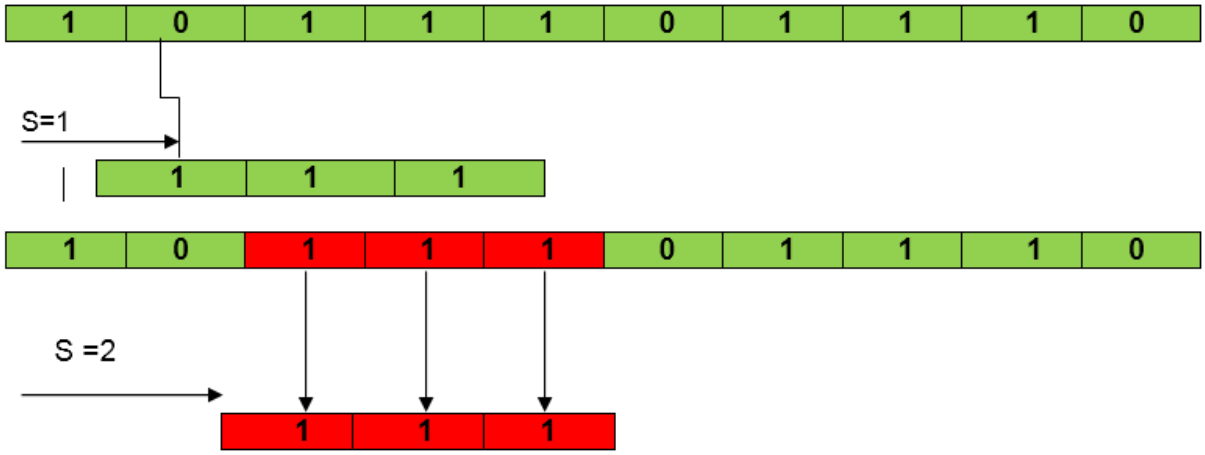
Solution:

Subject :- ADSAA **Semester :- V**

T = Text

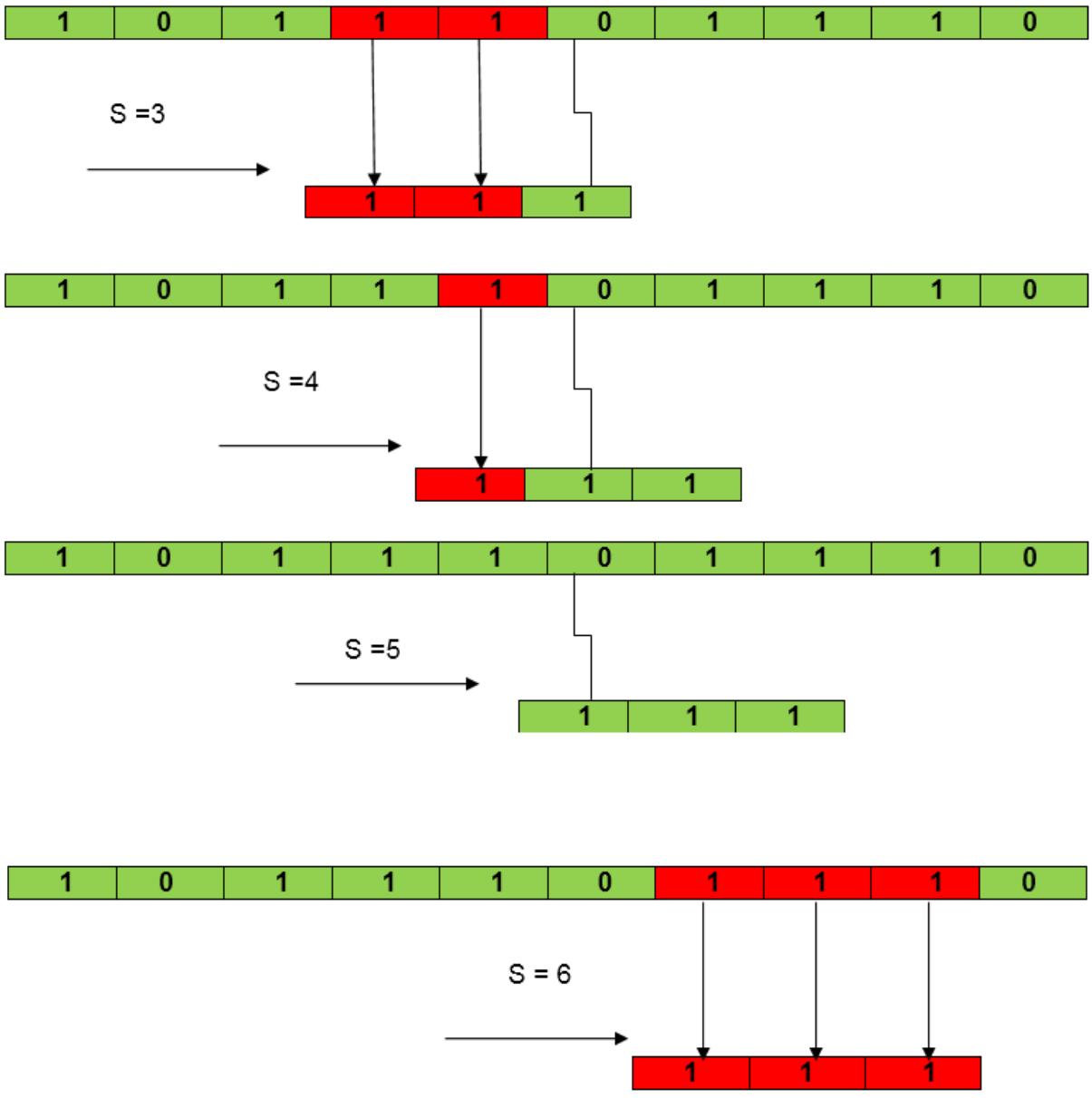


P = Pattern



Subject :- ADSAA **Semester :- V**

So, S=2 is a Valid Shift



Analysis: This for loop from 3 to 5 executes for n-m + 1(we need at least m characters at the end) times and in iteration we are doing m comparisons. So the total complexity is O (n-m+1).

The Rabin-Karp-Algorithm

The Rabin-Karp string matching algorithm calculates a hash value for the pattern, as well as for each M-character subsequences of text to be compared. If the hash values are unequal, the algorithm will determine the hash value for next M-character sequence. If the hash values are equal, the algorithm will analyze the pattern and the M-character sequence. In this way, there is only one comparison per text subsequence, and character matching is only required when the hash values match.

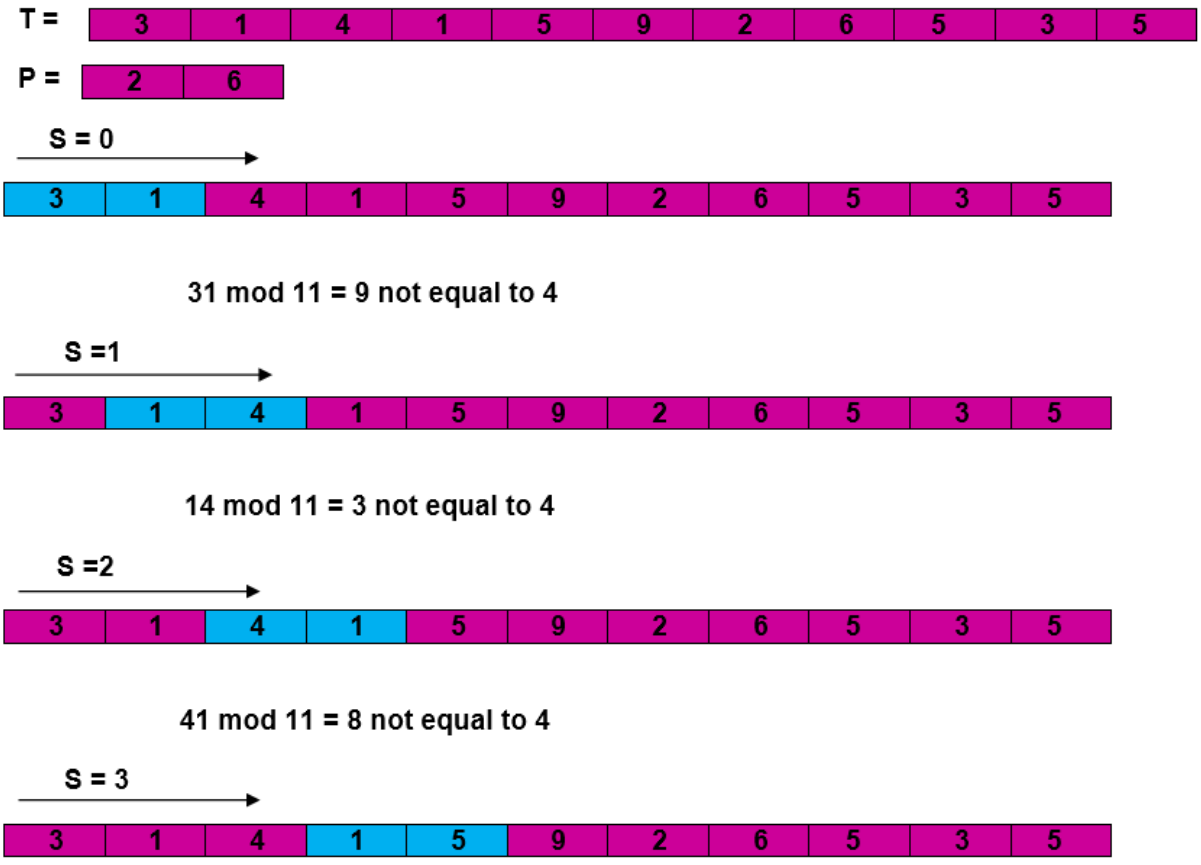
RABIN-KARP-MATCHER (T, P, d, q)

1. n ← length [T]
2. m ← length [P]
3. h ← d^{m-1} mod q
4. p ← 0
5. t_0 ← 0
6. for i ← 1 to m
7. do p ← (dp + P[i]) mod q
8. t_0 ← (dt_0+T [i]) mod q
9. for s ← 0 to n-m
10. do if p = t_s
11. then if P [1.....m] = T [s+1.....s + m]
12. then "Pattern occurs with shift" s
13. If s < n-m
14. then t_{s+1} ← (d (t_s-T [s+1]h)+T [s+m+1])mod q

Example: For string matching, working module q = 11, how many spurious hits does the Rabin-Karp matcher encounters in Text T = 31415926535.....

1. T = 31415926535.....
2. P = 26
3. Here T.Length =11 so Q = 11
4. And P mod Q = 26 mod 11 = 4
5. Now find the exact match of P mod Q...

Solution:



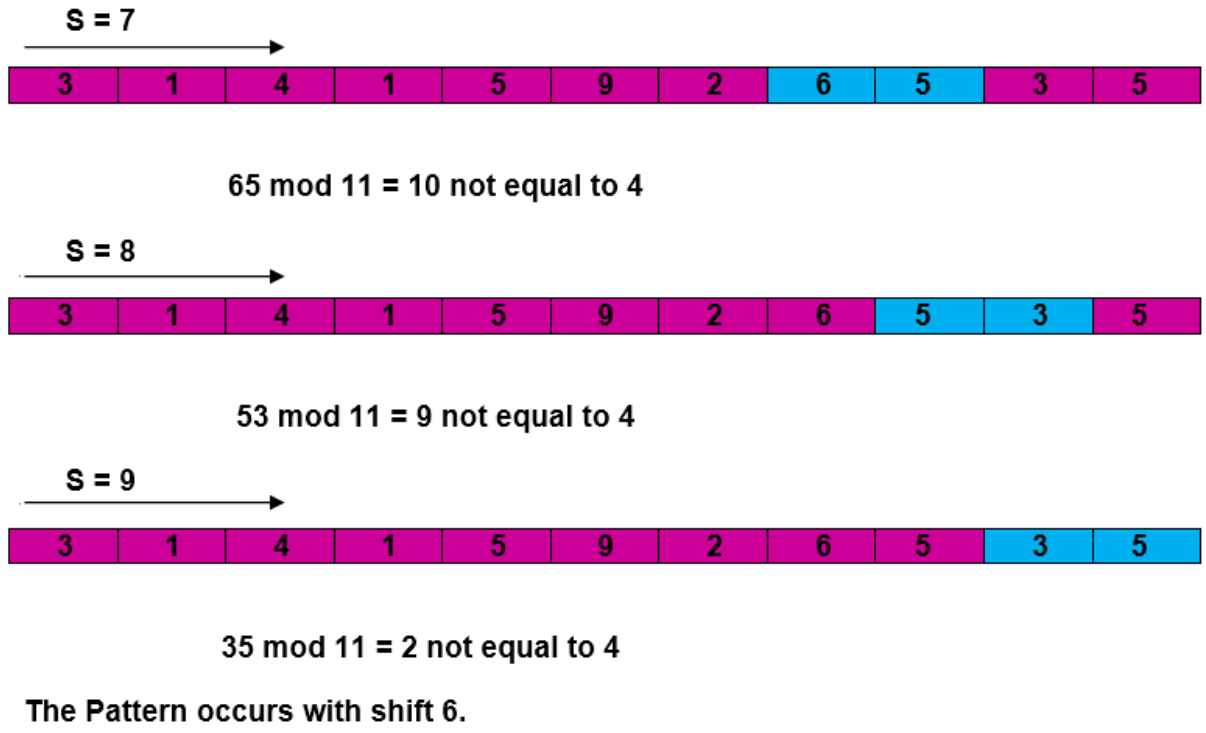
Subject :- ADSAA **Semester :- V**

15 mod 11 = 4 equal to 4 SPURIOUS HIT

59 mod 11 = 4 equal to 4 SPURIOUS HIT

92 mod 11 = 4 equal to 4 SPURIOUS HIT

26 mod 11 = 4 EXACT MATCH



Complexity:

The running time of **RABIN-KARP-MATCHER** in the worst case scenario **O ((n-m+1) m** but it has a good average case running time. If the expected number of strong shifts is small **O (1)** and prime q is chosen to be quite large, then the Rabin-Karp algorithm can be expected to run in time **O (n+m)** plus the time to require to process spurious hits.