Design and Analysis of Algorithms
CSE – 5311 – 010

Neel R Vora (1002078988)

String Matching Algorithms

Rabin-Karp algorithm:

The Rabin Karp algorithm makes use of hash function and rolling hash functions

Algorithm:

- Firstly, calculate the hash value of the pattern.
- Start traversing through the starting of the string:
    - Calculate the hash value of the current substring for that window in input text.
    - If the hash value and pattern is same as current substring, then store start index into answer.
    - Slide window to next substring.
- End

Pseudocode:

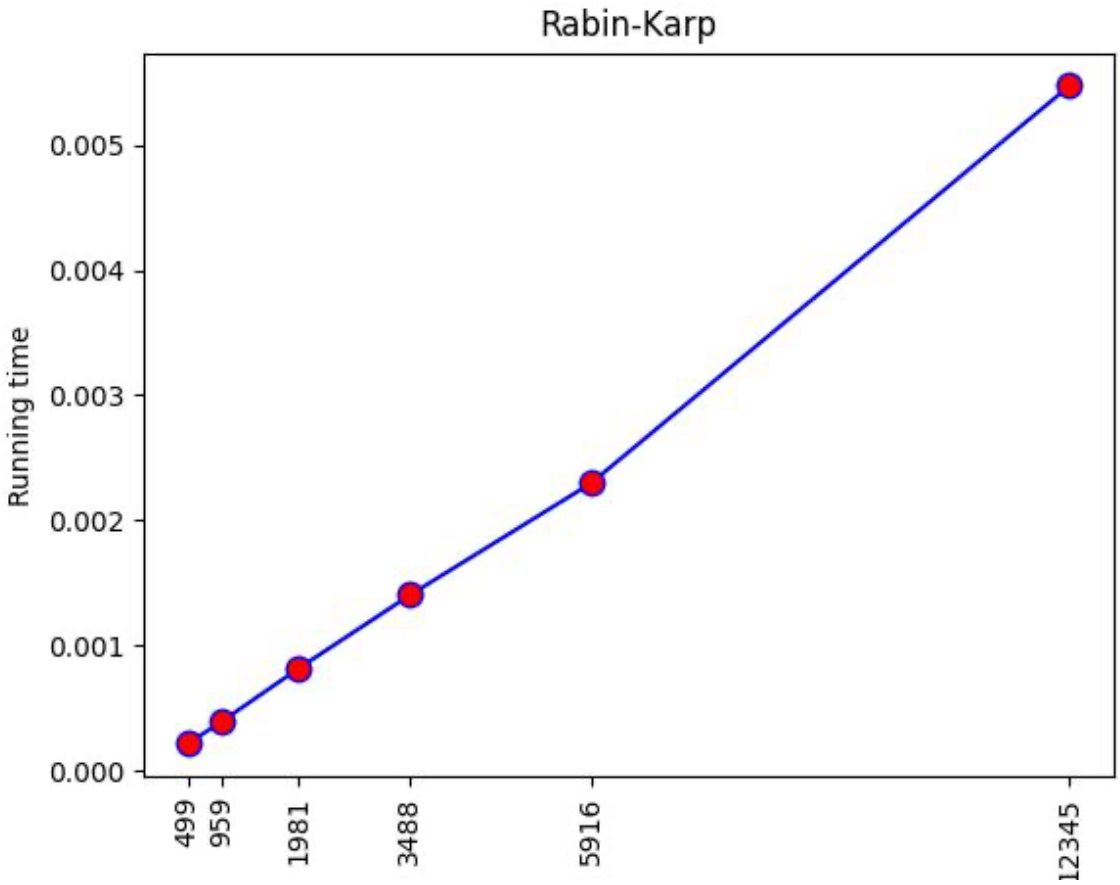
```
n=input.length
m=pattern.length
h=d-1 mod q
p=0
t=0
for i = 1 to m
p = (pat + p[i]) % q
t = (text + t[i]) % q
for s = 0 to n-m
        if p = ts
                if p[1...m] = t[s-1.....m]
                print ”hit”
                if s > n-m
                        ts + 1 = (d (ts-t[s+1]h) + t[s+m+1])%q
```

Time Complexity:

Best case – Average case: O(M+N)

Worst case: O(MN)

Output:


Rabin-Karp
Running time
0.000
0.001
0.002
0.003
0.004
0.005
499
959
1981
3488
5916
12345

Knuth – Morris – Pratt algorithm:

The KMP matching algorithm uses degenerating property of the pattern and improves the worst-case time complexity to linear. Foundation is not to compare the character that we know are already a match.

Algorithm:

- We first calculate **Π** function or longest proper prefix which is also a suffix.
- If pat[len] and pat[i] match, we increment len by 1 and assign the incremented value to lps[i]
- If pat[i] and pat[len] do not match, we update len to lps[len-1]
- We then start the comparison of pat[i] with i = 0 with characters of the current window of text
- We start matching character by character if it's a match
- If it's not matching, jump back to lps[j-1] because previous character is already compared once

Pseudocode:

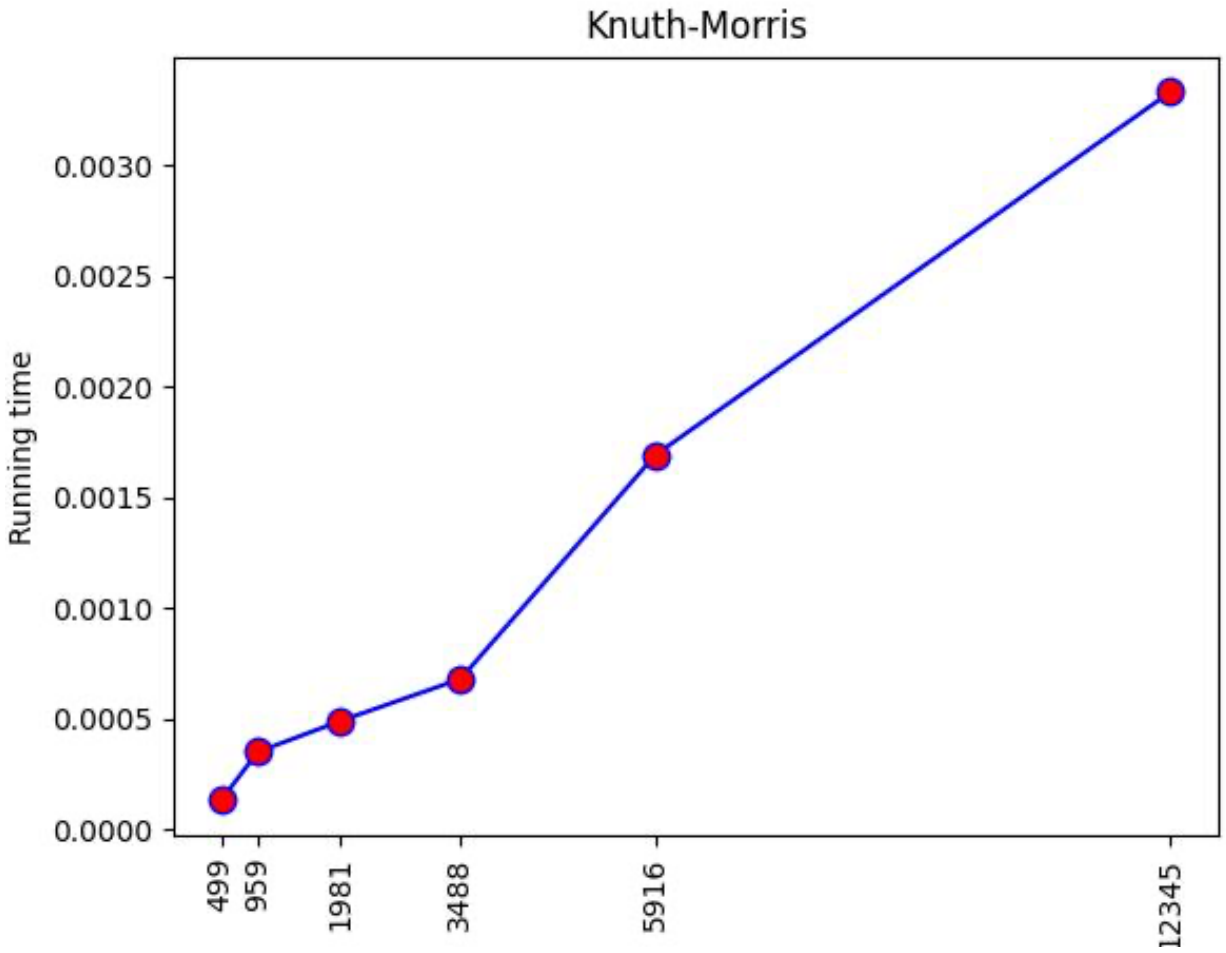
```
Compute Π function
m =length [P]
Π [1] = 0
k = 0
for q = 2 to m
        do while k > 0 and P [k + 1] ≠ P [q]
                k = Π [k]
        If P [k + 1] = P [q]
                K= k + 1
        Π [q] = k
Return Π

n = length [T]
m = length [P]
Π = COMPUTE-PREFIX-FUNCTION (P)
q = 0
for i = 1 to n
        do while q > 0 and P [q + 1] ≠ T [i]
                q = Π [q]
        If P [q + 1] = T [i]
                q = q + 1
        If q = m
                then print "Pattern occurs with shift" i - m
        q = Π [q]
```

Time Complexity:

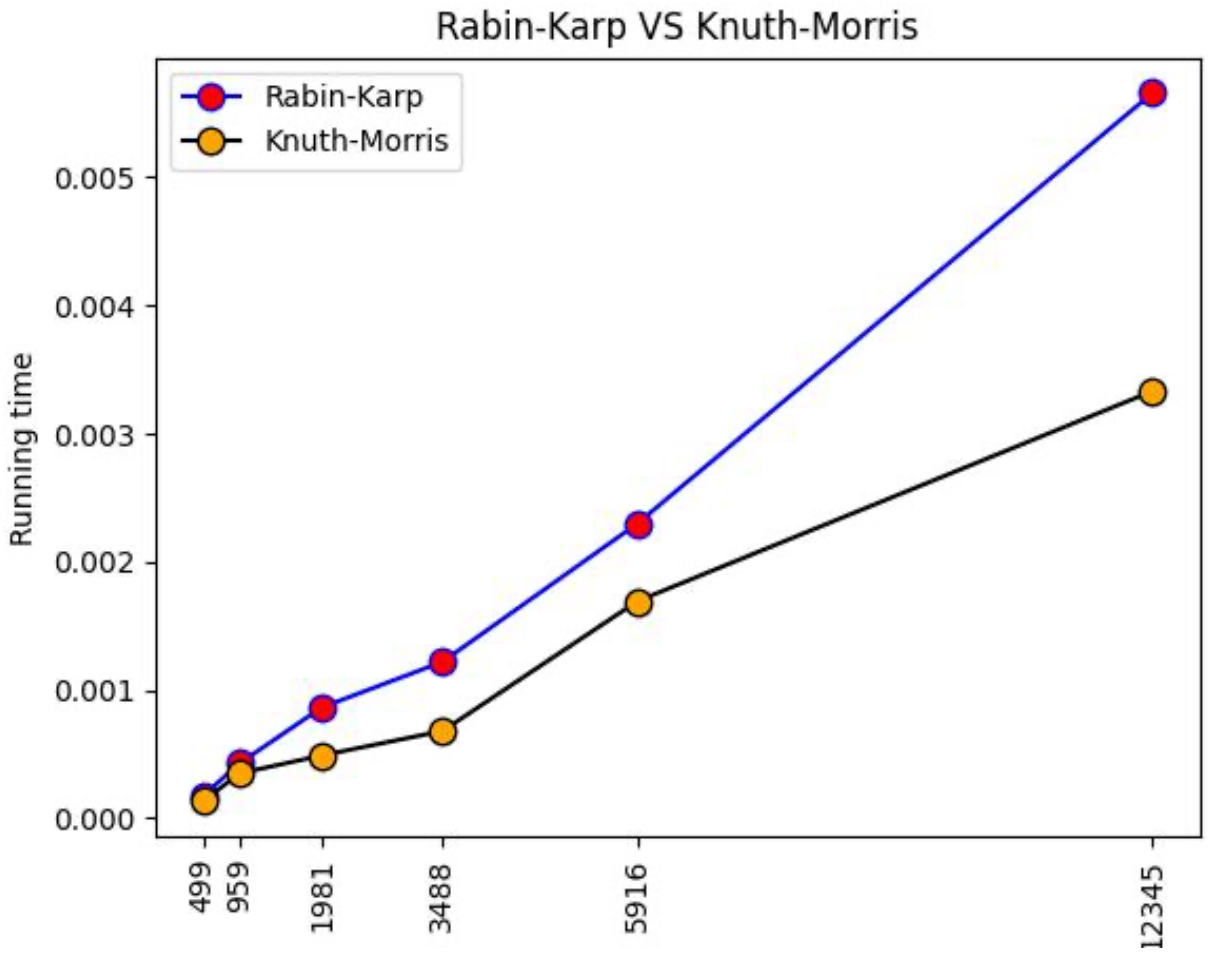
Worst case: O(M+N)

Output:



Performance Comparison.

Result:



Rabin-Karp	Knuth-Morris-Pratt
It is used to find one of the patterns from the input string in a text	KMP match the character from left to right suited for small variable
It can be modified to do fast searching	Constructs an Π from the patter for reduced redundancy
It uses hashing-based approach	It uses heuristic-based approach
Worst case runtime O(MN)	Worst case runtime O(M+N)