String Matching algorithms

Naive String Matching Algorithm

- **Algorithm**

```
NAIVE_STRING_MATCHING(T, P)
n ← length [T]
m ← length [P]
for i ← 0 to n – m
do
if P[1… m] == T[i+1…i+m]
Then
print "Pattern occurs with shift" s
end
end
```

Complexity analysis

Time complexity

- In the naive string matching algorithm, the time complexity of the algorithm comes out to be **O(n-m+1)**, where n is the size of the input string and m is the size of the input pattern string.

Space complexity

- In the naive string matching algorithm, the space complexity of the algorithm comes out to be **O(1)**.

Rabin Karp algorithm

```
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
```

Complexity

Time Complexity:

- The average case and best case complexity of Rabin-Karp algorithm is O(m + n) and the worst case complexity is O(mn).
- The worst-case complexity occurs when spurious hits occur a number for all the windows.

Space Complexity: O(1)

- It uses constant space. So, the space complexity is O(1).

Limitations of 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 hit, we use modulus. It greatly reduces the spurious hit.

KMP Algorithm

```
findPrefix(pattern, m, prefArray)
Begin
   length := 0
   prefArray[0] := 0
   for all character index 'i' of pattern, do
      if pattern[i] = pattern[length], then
         increase length by 1
         prefArray[i] := length
      else
         if length ≠ 0 then
            length := prefArray[length - 1]
            decrease i by 1
         else
            prefArray[i] := 0
   done
End
```

```
kmpAlgorithm(text, pattern)
Begin
   n := size of text
   m := size of pattern
   call findPrefix(pattern, m, prefArray)
   while i < n, do
      if text[i] = pattern[j], then
         increase i and j by 1
      if j = m, then
         print "The location (i-j) as there is the
                pattern"
         j := prefArray[j-1]
      else if i < n AND pattern[j] ≠ text[i] then
         if j ≠ 0 then
            j := prefArray[j - 1]
         else
            increase i by 1
   done
End
```

Complexity

- **Time Complexity**
- Time complexity of the search algorithm is $O(n)$.
- These complexities are the same, no matter how many repetitive patterns are in.
- **Space Complexity**

It has a space complexity of O (m) because there's some pre-processing involved.