Mitkumar Pandya,
CSC 505 : Assignment 5

Problem Statement : KMP and Performance Evaluation

Report : KMP Algorithm implements the idea of String-matching automata, using prefix and suffixes to match a pattern. KMP computes the prefix of the pattern to be matched such that it remembers the knowledge of shifts required to match a string when the matching fails the first time.

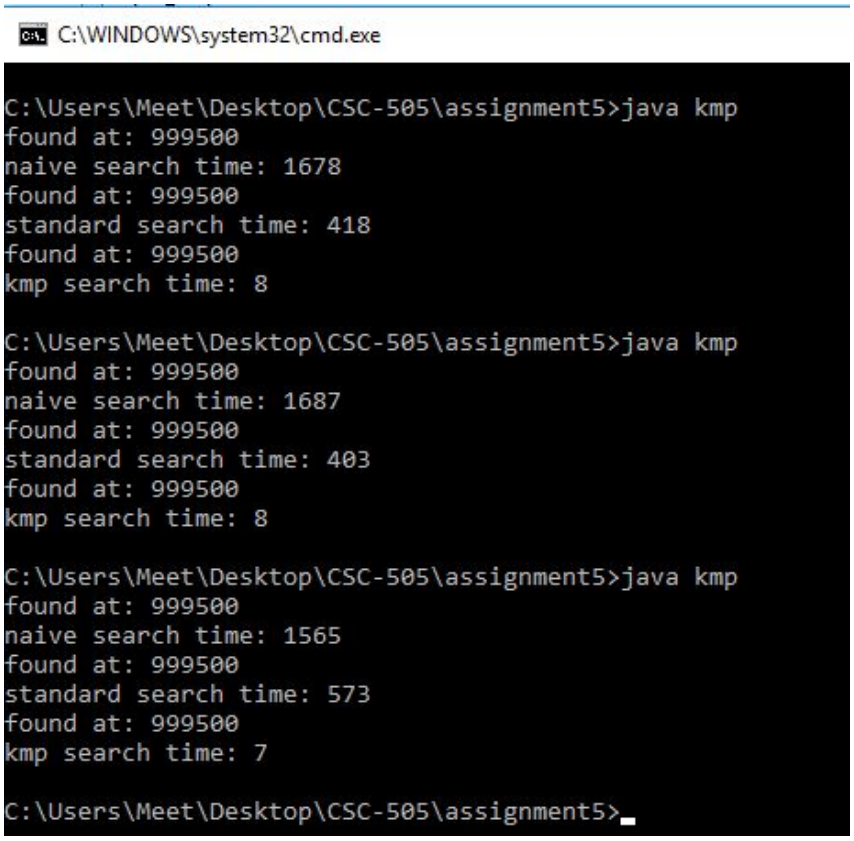
According to this functionality I have designed a simple String with 1 million '0' s as the prefix and a suffix '1' . The pattern to be matched contains five hundred '0' s followed by a '1'.

Input String : "00000000000000000000000000000000000000000000……...00000001"

Pattern : "0000000000000000000000 …… 0000000000001"

The algorithm first computes the prefix of the pattern to be matched. The prefix function returns the computed array as [0,1,2,3,4 …., 499, 0]. Now once it starts matching the pattern with original string, at character 499 (counting from 0) the match fails as 0 != 1. But here we have already matched first 499 characters successfully hence for the next character we will only match the last character of the pattern as computed by the prefix function. This concludes that at a certain point KMP matcher will only compare from the occurrence of the pattern where character matching failed. This saves the computing time of already matched characters. Hence the algorithm runs in the linear time of O (n + m) where n = size of the String to be matched and p = size of the pattern and it requires additional space of O(m) to store the prefix function value.

The output of the above inputs is shown below.


C:\WINDOWS\system32\cmd.exe
C:\Users\Meet\Desktop\CSC-505\assignment5>java kmp
found at: 999500
naive search time: 1678
found at: 999500
standard search time: 418
found at: 999500
kmp search time: 8
C:\Users\Meet\Desktop\CSC-505\assignment5>java kmp
found at: 999500
naive search time: 1687
found at: 999500
standard search time: 403
found at: 999500
kmp search time: 8
C:\Users\Meet\Desktop\CSC-505\assignment5>java kmp
found at: 999500
naive search time: 1565
found at: 999500
standard search time: 573
found at: 999500
kmp search time: 7
C:\Users\Meet\Desktop\CSC-505\assignment5>

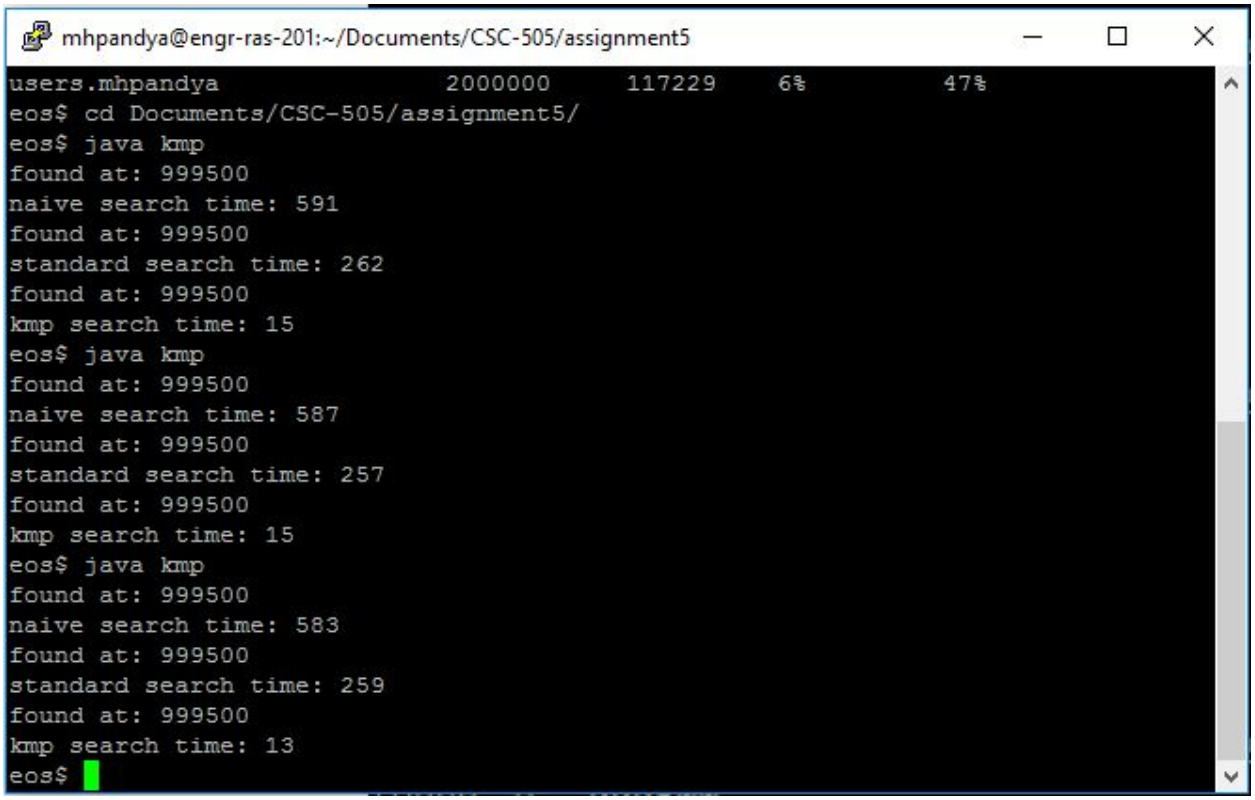

mhpandya@engr-ras-201:~/Documents/CSC-505/assignment5
users.mhpandya 2000000 117229 6% 47%
eos$ cd Documents/CSC-505/assignment5/
eos$ java kmp
found at: 999500
naive search time: 591
found at: 999500
standard search time: 262
found at: 999500
kmp search time: 15
eos$ java kmp
found at: 999500
naive search time: 587
found at: 999500
standard search time: 257
found at: 999500
kmp search time: 15
eos$ java kmp
found at: 999500
naive search time: 583
found at: 999500
standard search time: 259
found at: 999500
kmp search time: 13
eos$

Fig : output on linux machine

In this case KMP works faster compared to both Naive approach and Java's inbuilt indexOf functionality.

KMP vs Naive : The advantage of KMP over Naive's O (n * m) approach is that Naive string matching algorithm does not utilize the already computed information. The Naive algorithms will compare every character of String to that of pattern and when the character mismatch occurs it starts from the first character of the pattern.

KMP vs indexOf : Java's indexOf method is the better version of Naive algorithm and it performs O (n + m) in average case and O (n * m) in the worst case as shown in the example above.

From the above experiments we can conclude that KMP algorithms works faster using the idea of suffix and prefix. When the pattern contains many repeated characters then KMP is better approach as compared to naive algorithm and indexOf method, but it requires extra space to store the prefix function values. Hence when the pattern to be matched is long enough then KMP algorithm is not an optimal approach considering the memory overhead.