# Project 15

## Group Members

Abhishek Gupta (axg137230) and Manuj Singh (mxs135630)

## Objective

To implement four string matching algorithm namely naïve, Rabin Karp, KMP, and Boyer Moore and compare their performances.

## Problem Statement

Given a pattern (comprising of only 0/1) and an input file containing long strings and new lines (comprising of only 0/1) we need to find the number of valid shifts for the given pattern in the input file and display the running time of each algorithm. We also need to display the shifts if the number of shifts is greater than equal to 20.

## Implementation

We are reading the input file line by line and appending the string by ignoring the new lines. Then using four algorithms searching the pattern in the string constructed from the input file. Below are the pseudo code for the four algorithms:

### Naïve Algorithm

Naïve Matching (String pattern [1…m], String text [1…n])

For s <- 1 to n-m do

If pattern = text.Substring (s, m) then

Report (s); // valid shift found

### Rabin Karp Algorithm

Rabin Karp Matching (String pattern [1…m], String text [1…n])

q <- 101;

mult <- pow (2, m-1) mod q;

hp <- hash(pattern, q);

hs <- hash(text.Substring(1,m),q);

For s <- 1 to n-m do

If hp = hs then

If pattern = text.Substring (s, m) then

Report (s); // valid shift found

If s < n – m then

hs = (2\*(hs-text[s]\*mult)+text[s+m])

If hs < 0 then

hs += q;

hash (String str[1..m], int q)

h <- 0;

For I <- 1 to m do

h = (h \* 2 + str[i]) mod q;

return h;

### KMP Algorithm

Kmp-Matcher (T, P):

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] do

q ← π[q]

if P[q+1] = T[i] then q ← q+1

if q = m then

print pattern occurs with shift i-m

q ← π[q]

Prefix function π:

//π[q] = Length of the largest prefix P[1 … i] of P[1 … q] that is //also a suffix of P[1 … q] (not including P[1 … q] itself)

//P[1 … q] is the current match

Compute-prefix-function (P):

π[1] ← 0

k ← 1

for q ← 2 to m do

if P[q] = P[k] then

π[q] = k++

else if k > 1

k =π[k-1] + 1; continue

else π[q] = 0

return π

### Boyer Moore Algorithm

Boyer Moore Matching (String pattern [1…m], String text [1…n])

bad <- new int[CHAR\_SIZE]; // here CHAR\_SIZE is 2

for k <- 1 to CHAR\_SIZE do

bad[k] <- -1;

badCharProcessing(bad)

for i <- 1 to n-m do

j <- m;

while j > 0 && pattern[j] = text[i+j] do j—-

if j<1

report i // pattern found here

i += ((i + m < n)? m – bad[text[i+m] : 1);

else

i += max (1, j – bad[text[i+j]]);

badCharProcessing (int[] bad)

for i <- 1 to m do

bad[pattern[i]] = i;

## Results and Conclusion

We tried various pattern and input file with various text size. Below are the some of the results.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input String Length (at least) | Pattern | # of Shifts | Running time (in milliseconds) | | | |
| Naïve | Rabin Karp | KMP | Boyer Moore |
| 1000 | 01010010 | 66 | 1 | 0 | 0 | 1 |
| 100000 | 01010010101 | 5116 | 32 | 15 | 17 | 17 |
| 10000000 | 0101001010110101 | 400013 | 476 | 147 | 120 | 172 |

The above table shows how Rabin-Karp, KMP, and Boyer Moore matching algorithms are a significant improvement over the naïve algorithm.