Theoretical and Empirical Comparisons of String Matching Algorithms

Faisal Albaiz 436170294, and Nahar Alrasheedi 435100367

Introduction:

In this project we are going to implement and test each of the following string-matching algorithms:

1. **brute -force algorithm**: a straightforward code, easy, and simple.
2. **Boyer-Moore algorithm:** an algorithm that skips through the text looking for the pattern
3. **Knuth-Morris-Pratt algorithm**: an algorithm that matches letter by letter from both the pattern and the text.

To see which of the previous algorithm is faster and more efficient we need to experiment and try to get an outcome of each of the three algorithms, we will try to get repeat our code input five to ten times to get the average of the output, thus assuring that we have a reliable answer.

Algorithms:

The brute-force algorithm:

Brute-force is good mainly for small sized problems, it is simple to implement, compile, and run. All problems faced with brute-force is handled easily and quickly. Brute-force is also better for general use.

The Boyer-Moore algorithm:

The Boyer-Moore(BM) uses data acquired by preprocessing the pattern to skip as many alignments as possible.
for example:

The *text* is: “KSUCSC” and the *pattern* is: “UCS”

K	S	U	C	S	C
U	C	S	-	-	-
-	U	C	S	-	-
-	-	U	C	S	-

reference[5]

The Knuth-Morris-Pratt algorithm:

With Knuth-Morris-Pratt(KMP) we keep in mind four attributes, and the example will make it clear.

1. The *text* given which we will label “**txt**”
2. The *position* of the text which we will call “**P**”
3. The *pattern* “**pat**”
4. The *index* of the *pattern* “**I**”

What we will do is see if the letter on the position of the text “P’’ is matching its counterpart the index of the pattern “I”, if the answer is yes then move P and I one step and repeat.
reference[7]

(I)

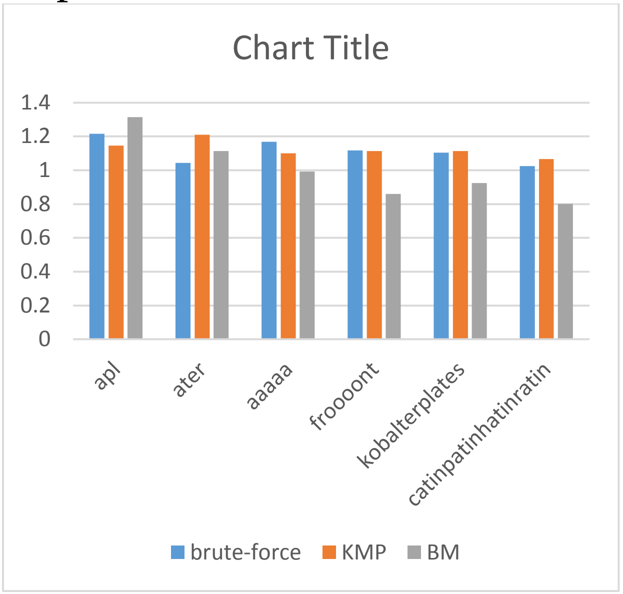
P:	0	1	2	3	4
Txt:	A	B	C	D	E
I:	0	1	2		
Pat:	B	C	D		

(II)

P:	0	1	2	3	4
Txt:	A	B	C	D	E
I:		0	1	2	
Pat:		B	C	D	

reference[6]

Experiment:



In this experiment, we found that the boyer-moore is the best string-matching algorithm when putting it next to KMP and brute-force.
We can clearly see that the boyer-moore decreases its time when putting a large input. And if the KMP has no repetitive letters it is then very close in the performance to Brute-force.
With KMP and brute-force we can see similarity in the majority of the large input. While boyer-moore is faster.

pseudoCode:

KMP:

```
KMP-MATCHING (T, P)
n ← length(T)
m ← length(P)
π ← COMPUTE-PREFIX-FUNCTION (P)
q ← 0
for i ← 1 to n
(*) while (q > 0 ∧ P[q + 1] 6= T[i]) q ←
π[q]
(*) if (P[q + 1] = T[i]) q ← q + 1
if q = m
print “pattern occurs with shift i − m”
(*) q ← π[q]
```

Boyer-Moore:

```
BoyerMooreMatch(T, P, Σ)
L ← lastOccurenceFunction(P, Σ )
i ← m − 1
j ← m − 1
repeat
if T[i] = P[j]
if j = 0
return i { match at i }
else
i ← i − 1
j ← j − 1
else
{ character-jump }
l ← L[T[i]]
i ← i + m min(j, 1 + l)
j ← m − 1
until i > n − 1
return −1 { no match }
```

Brute-Force:

```
Searching for a pattern, P[0...m-1], in text,
T[0...n-1] BruteForceStringMatch(T[0...n-
1],P[0...m-1])
for i ← 0 to n-m do
j ← 0
while j < m and P[j] = T[i+j] do
j++
if j = m then return i
return -1
```

References:

[1]http://cs.indstate.edu/~kmandumula/presentation.pdf
[2]https://www.cs.ubc.ca/~hoos/cpsc445/Handouts/kmp.pdf
[3]https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/
[4]https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/
[5]https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm
[6]https://www.youtube.com/watch?v=V5-7GzOfADQ
[7]https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm
[8]https://www.cs.auckland.ac.nz/compsci369s1c/lectures/GG-notes/CS369-StringAlgs.pdf