Ayça Begüm Taşçıoğlu
21600907
CS481 - 1
20 October 2019
Asst. Prof. Can Alkan
Homework I - Report

Exact Pattern Matching

Between the algorithms, Brute Force, Knuth-Morris-Pratt, Boyer-Moore and Rabin Karp, the execution achieved as follows:

```
(base) Ayca-MacBook-Pro:CS481-HW1 Ayca$ make
python hw1.py program.dat
Type the name of text file (should be located in same file with program code):
text.fa
Type the name of pattern file (should be located in same file with program code):
pattern.fa
Welcome to exact string matching comperator bot!
If the pattern is in text, locations will be printed in 1-based coordinate

 For Brute Force type 1
 For KnuthMorrisPratt type 2
 For BoyerMoore type 3
 For RabinKarp type 4
 For exit type 0.
1
Given Pattern found in the text in position:  180 , 212
Number of comparisons:  277
Time : 0.09819099999930359

 For Brute Force type 1
 For KnuthMorrisPratt type 2
 For BoyerMoore type 3
 For RabinKarp type 4
 For exit type 0.
2
Given Pattern found in the text in position:  180 , 212
Number of comparisons:  354
Time : 0.17911999999853379

 For Brute Force type 1
 For KnuthMorrisPratt type 2
 For BoyerMoore type 3
 For RabinKarp type 4
 For exit type 0.
3
Time : 0.11769599999844615
Given Pattern found in the text in position:  (180, ',', 212)
Number of comparisons:  57

 For Brute Force type 1
 For KnuthMorrisPratt type 2
 For BoyerMoore type 3
 For RabinKarp type 4
 For exit type 0.
4
Time : 0.17053599999883318
Given Pattern found in the text in position:  (180, ',', 212)
Number of comparisons:  31

 For Brute Force type 1
 For KnuthMorrisPratt type 2
 For BoyerMoore type 3
 For RabinKarp type 4
 For exit type 0.
0
GoodBye!
```

Figure 1: Given pattern is matched in the given text

Figure 2: Given pattern is not found in the given text

Even though for small inputs, these algorithms run time functions are close, for bigger inputs of data we can observe differences. I tried the examples which are provided in lecture slides. Timer results were close. However, when I tried the example provided under the assignment, I got different results. This is because, for larger inputs since the Brute Force Algorithm shifts one whenever a mismatch occurs, it becomes slower. It is also the same for the Knuth-Morris-Pratt Algorithm since the Failure Function table gets larger for larger pattern sizes. For the Boyer-Moore Algorithm, since in my implementation I calculate and compare shifts that are provided by BadCharRule and GoodSuffixRules, it also becomes slower for larger inputs. Last, for Rabin Karp Algorithm, since the worst-case scenario for this algorithm is "fingerprints of pattern and pattern-size text portion is same yet, strings are not exactly matching", probability of encounter with this worst-case scenario gets also larger (the alphabet size in FASTA format {A, T, G, C} also affects this).

As a conclusion, Brute Force and Knuth-Morris-Pratt are not observed as effective as others. Brute Force Algorithm has the worst-case run time of O(mn) where m is pattern length and n is text length. Knuth-Morris-Pratt algorithm has a worst-case run time of O(m+n) since its failure function has performed in O(m). Furthermore, since we compared algorithms by the number of character comparisons, these algorithms have the biggest number of comparisons of characters. Boyer-Moore algorithm is more effective than these algorithms for bigger and more complex inputs (if the input is not in FASTA format, more character types). However, its number of character comparisons is bigger than Rabin Karp Algorithm. Hence, I think I would use Rabin Karp Algorithm. Because of the number of character comparison in this algorithm is computed if and only if the modulo of the fingerprints of text and pattern is matched. It leads to the algorithm to work faster, and perform more efficiently than the other algorithms mentioned above. Of course, the disadvantageous perspective is this algorithm is the having same fingerprints from pattens and pattern-size text yet not having exactly matching strings. If this case is faced, in my implementation I iterate pattern characters and pattern-size text characters one-by-one to calculate the number of character comparisons, hence it executes slower. If we can be certainly sure about this worst-case scenario will not be faced, I would use Rabin Karp Algorithm.