MARMARA UNIVERSITY
FACULTY OF ENGINEERING

Algorithm Analysis

Project-1

String Matching Algorithm

Due Date: 17.05.2023

	Name Surname	Student Id
1	Niyazi Ozan Ateş	150121991
2	Ahmet Arda Nalbant	150121004
3	Hasan Özeren	150121036
4	Umut Bayar	150120043

TASKS

There were no particular tasks to anyone. We decided to work as a group. Sitting around 1 big table and working together as a team on the same part of the project. Until we found multiple solutions for sub-problems. Then we compared our coding's to get the optimal algorithms and time efficiency.

While writing the report we decide this time to divide the tasks among the members of the team.

Niyazi Ozan Ateş and Ahmet Arda Nalbant worked on creating and testing the different inputs and patterns for the code. Also, they wrote the part of the creation of the code and results/conclusion.

Hasan Özeren and Umut Bayar worked on writing the purpose and combining the text that are given from the other 2 members work. Also, they worked on the layout of the report.

PURPOSE

This program is aimed to create 3 different algorithms that checks whenever a pattern occurrence in a text. We use the Brute Force, Horspool, and Boyer Moore algorithms. While the program starts with one of the algorithms, we keep track of the time, occurrence, and number of comparisons. This program works by asking the user to enter the input .html file and the pattern

that it will be searching. As an output we give the time, occurrence, and number of comparisons in the console. Also, we highlight the found patterns in the output file which is basically the input file but with highlighted patterns.

Algorithms Explained

Brute Force:

A Brute Force algorithm solves a problem through exhaustion. It goes through all possible choices until a solution is found. The time complexity of a brute force algorithm is often proportional to the input size. Brute Force algorithms are simple and consistent, but very slow. It starts from the most left character and checks until a mismatch occurrence. When a mismatch or a successful match is found we check the index only by 1.

Horspool:

Horspool is a pattern matching algorithm. It will compare from the right of the pattern with the text. The shift amount is based on the Bad Symbol Table. This shifts the amount of the first occurrence of the character. The character that it searches for is the character of the first iteration in the text.

Boyer Moore:

Boyer Moore is a pattern matching algorithm. It checks from the right of the pattern and shift based on the maximum value of the bad and good tables.

The Bad Symbol Table is used a little different than in the Horspool algorithm. The Table is the same but this time we check in the text what the bad character to find the jump. We then do jump amount minus the correct part. The Good Suffix Table is as explained before. At last we do Math.max(Math.max(badJump,1),goodJump). This will find the maximum we can jump to possible find a new match.

Tables Explained

Bad Symbol Table:

The Bad Symbol Table create a table based on whenever a character is inside the pattern or not. If it is not in the pattern, it shift the amount of the length of the pattern. Otherwise, it jumps until the first occurrence of the character from the right that match the text from the index.

Good Suffix Table:

The Good Suffix Table is created based on the correct found matches. It chooses the best possible shift based on the number of matches from the right.

EXAMPLE #1 (string1.html) (TEST CASE IN GIVEN HOMEWORK TIMES 5)

For the first .html file we have chosen the homework pdf file 5 times the length. We did this to get an enough long file (1 megabyte) so we can compare properly. The pattern input will be ‘AT_THAT’. By doing this we will also test the given homework task. We believe that this is a proper .html file because it contains English-only words.

```
What is the input file labeled as (example.html): string1.html
What is the pattern labeled as (BAOBAB): AT_THAT

Bad Symbol Table:
Char: _       Jump: 4
Char: T       Jump: 3
Char: H       Jump: 2
Char: A       Jump: 1
Char: others Jump: 7

Good Suffix Table:
Matching part (k = 1): T        = 3
Matching part (k = 2): AT       = 5
Matching part (k = 3): HAT      = 5
Matching part (k = 4): THAT     = 5
Matching part (k = 5): _THAT    = 5
Matching part (k = 6): T_THAT   = 5
Matching part (k = 7): AT_THAT  = 5

Brute Force:
Number of comparisons for Brute Force: 1136103
Number of occurence for Brute Force:   10
Brute Force time(in miliseconds):      158

Horspool:
Number of comparisons for Horspool:    168422
Number of occurence for Horspool:      10
Horspool time(in miliseconds):         112

Boyer Moore:
Number of comparisons for Boyer Moore: 168392
Number of occurence for Boyer Moore:   10
Boyer Moore time(in miliseconds):      162
```

Note 1: You may do this homework in groups of three or four.

Note 2: Some old browsers may not support <mark> tag. Please use a browser that supports this tag for displaying the output HTML file.

Note 3: Please submit your commented source codes, input files, and detailed report in a zip file via google classroom. The name of the zip file should include the names and the surnames of all the group members. It is enough if a single group member submits the homework.

Note 4: Describe the detailed division of labor among the group members.

Note 5: Other than the long HTML files and selected patterns, you SHOULD also run your code for the following text and pattern.

Text: **<HTML><BODY>WHICH_FINALLY_HALTS. _ _ AT_THAT POINT </BODY></HTML>**

Pattern: **AT_THAT**

If your code doesn't work correctly for this text and this pattern, or you do not provide results for this text and this pattern, then your code will be considered as NOT WORKING! Algorithms should work exactly as described in the lecture.

The reason that we get 10 occurrences is because we got the homework turned into a .html files 5 times the length. We can also see that Horspool is the best algorithm to choose for this .html file and pattern.

EXAMPLE #2 (string2.html)

For the second .html file we have chosen one of the previous homework pdf file 10 times the length. We did this to get an enough long file (1 megabyte) so we can compare properly. The pattern input will be 'BAOBAB'. In this example we will see what happens if the pattern is not found and the reaction of the execution time. We believe that this is a proper .html file because it contains English-only words.

```
What is the input file labeled as (example.html): string2.html
What is the pattern labeled as (BAOBAB): BAOBAB

Bad Symbol Table:
Char: O      Jump: 3
Char: B      Jump: 2
Char: A      Jump: 1
Char: others Jump: 6

Good Suffix Table:
Matching part (k = 1): B       = 2
Matching part (k = 2): AB      = 5
Matching part (k = 3): BAB     = 5
Matching part (k = 4): OBAB    = 5
Matching part (k = 5): AOBAB   = 5
Matching part (k = 6): BAOBAB  = 5

Brute Force:
Number of comparisons for Brute Force: 603599
Number of occurence for Brute Force:   0
Brute Force time(in miliseconds):      94

Horspool:
Number of comparisons for Horspool:    110616
Number of occurence for Horspool:      0
Horspool time(in miliseconds):         59

Boyer Moore:
Number of comparisons for Boyer Moore: 109953
Number of occurence for Boyer Moore:   0
Boyer Moore time(in miliseconds):      41
```

The reason that we get 0 occurrences is because the .html file does not contain the pattern. We have no highlighted pattern in the output .html file. We can also see that Boyer Moore is the best algorithm to choose for this .html file and pattern. This is because of the Good Suffix Table.

EXAMPLE #3 (string3.html)

For the third .html file we have chosen one of the previous practice questions pdf file 2 times the length. We did this to get an enough long file (1 megabyte) so we can compare properly. The pattern input will be ‘ollo’. In this example we will see what happens if the pattern is starting and ending with the same character and the effect on the execution time. We believe that this is a proper .html file because it contains English-only words.

```
What is the input file labeled as (example.html): string3.html
What is the pattern labeled as (BAOBAB): ollo

Bad Symbol Table:
Char: o      Jump: 3
Char: l      Jump: 1
Char: others Jump: 4

Good Suffix Table:
Matching part (k = 1): o     = 3
Matching part (k = 2): lo    = 3
Matching part (k = 3): llo   = 3
Matching part (k = 4): ollo  = 3

Brute Force:
Number of comparisons for Brute Force: 1256275
Number of occurence for Brute Force:   18
Brute Force time(in miliseconds):      223

Horspool:
Number of comparisons for Horspool:    316935
Number of occurence for Horspool:      18
Horspool time(in miliseconds):         113

Boyer Moore:
Number of comparisons for Boyer Moore: 316935
Number of occurence for Boyer Moore:   18
Boyer Moore time(in miliseconds):      116
```

1. Consider a text with one thousand zeros.
(a) How many character comparisons will the **Horspool's algorithm** make in searching the pattern 10010? (Show also the shift table.)
(b) How many character comparisons will the **Boyer-Moore algorithm** make in searching the pattern 10010? (Show also the bad symbol and good-suffix tables.)
(c) How many character comparisons will the **brute-force string matching** algorithm make in searching the pattern 10010?

2. Consider the following pattern:
12345678909876543211112223333444555666777888999000567483920934930293847745 89
(a) In the above pattern, search for last six digits of your student ID using **Horspool's algorithm**.
(b) Repeat (a) using **Boyer-Moore** algorithm.
(c) How many character comparisons are needed in each case. Comment on your results.
(d) How many character comparisons are needed when you apply **brute-force string matching** algorithm?

3. *Rod cutting problem:* Given a rod of length n, the problem is to find best way to cut the rod into integer-length pieces (to obtain maximum sale price) where the sale price of a piece i units long is p_i for i=1, 2, ..., n.
(a) Design a **dynamic programming algorithm** for the rod cutting problem. (Hint: It is similar to change making problem.) What is the time and space complexity of your algorithm?
(b) Apply your dynamic programming algorithm to the following instance: You have a rod of length 6. And sale prices of pieces are given as: p_1=1, p_2=5, p_3=7, p_4=11, p_5=14, p_6=15.
(c) Design an efficient greedy algorithm for the rod cutting problem. Apply your algorithm to the same instance as in (b). Do your algorithm give the optimal solution for this instance?

4. Consider the following weighted digraph.

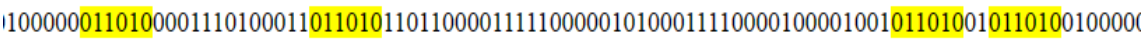
All algorithms find the same number of occurrences. We see the highlighted pattern in the output .html file. We can also see that Horspool is a

little bit better than Boyer Moore for this .html file and pattern. This is because of the Bad Symbol Table dominates the jump amount.

EXAMPLE #4 (binary1.html)

For the binary1.html file we have created a randomized bit text using a random bit generator. In the first example we compare it with a randomized pattern of bits. Which is '011010'. We believe this is a correct way because there is a 50% chance to get a 0 or 1 each.

```
What is the input file labeled as (example.html): binary1.html
What is the pattern labeled as (BAOBAB): 011010

Bad Symbol Table:
Char: 0      Jump: 2
Char: 1      Jump: 1
Char: others Jump: 6

Good Suffix Table:
Matching part (k = 1): 0       = 5
Matching part (k = 2): 10      = 2
Matching part (k = 3): 010     = 5
Matching part (k = 4): 1010    = 5
Matching part (k = 5): 11010   = 5
Matching part (k = 6): 011010  = 5

Brute Force:
Number of comparisons for Brute Force: 3896406
Number of occurence for Brute Force:   37532
Brute Force time(in miliseconds):      342

Horspool:
Number of comparisons for Horspool:    2939866
Number of occurence for Horspool:      37532
Horspool time(in miliseconds):         253

Boyer Moore:
Number of comparisons for Boyer Moore: 1885757
Number of occurence for Boyer Moore:   37532
Boyer Moore time(in miliseconds):      250
```



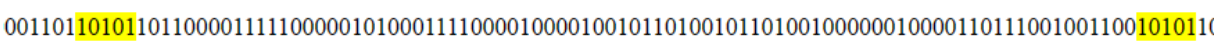
As we can see the best algorithm for this .html and pattern combination is the Boyer Moore algorithm. This is because of the Good Suffix Table. Also, we

can see from the number of comparisons that it is the best among all of the algorithms.

EXAMPLE #5 (binary2.html)

For the binary2.html file we have created a randomized bit text using a random bit generator. In the second example we compare it with a repetitive pattern of bits. Which is ‘10101’. We believe this is a correct way because there is a 50% chance to get a 0 or 1 each.

```
What is the input file labeled as (example.html): binary2.html
What is the pattern labeled as (BAOBAB): 10101

Bad Symbol Table:
Char: 1      Jump: 2
Char: 0      Jump: 1
Char: others Jump: 5

Good Suffix Table:
Matching part (k = 1): 1      = 4
Matching part (k = 2): 01     = 4
Matching part (k = 3): 101    = 2
Matching part (k = 4): 0101   = 2
Matching part (k = 5): 10101  = 2

Brute Force:
Number of comparisons for Brute Force: 2167227
Number of occurence for Brute Force:   28043
Brute Force time(in miliseconds):      219

Horspool:
Number of comparisons for Horspool:    1686793
Number of occurence for Horspool:      28043
Horspool time(in miliseconds):         192

Boyer Moore:
Number of comparisons for Boyer Moore: 1054302
Number of occurence for Boyer Moore:   28043
Boyer Moore time(in miliseconds):      192
```



As we can see the best algorithm for this .html and pattern combination are the Horspool and Boyer Moore algorithm. We can see from the number of

comparisons that Boyer Moore algorithm has a smaller number of comparisons. So, it's more efficient based on the comparison's numbers.

EXAMPLE #6 (binary3.html)

For the binary3.html file we have created a randomized bit text using a random bit generator. In the third example we compare it with a false pattern of input. Which is '10131'. We believe this is a correct way because there is a 50% chance to get a 0 or 1 each.

```
What is the input file labeled as (example.html): binary3.html
What is the pattern labeled as (BAOBAB): 10131

Bad Symbol Table:
Char: 0      Jump: 3
Char: 1      Jump: 2
Char: 3      Jump: 1
Char: others Jump: 5

Good Suffix Table:
Matching part (k = 1): 1      = 2
Matching part (k = 2): 31     = 4
Matching part (k = 3): 131    = 4
Matching part (k = 4): 0131   = 4
Matching part (k = 5): 10131  = 4

Brute Force:
Number of comparisons for Brute Force: 17183397
Number of occurence for Brute Force:   0
Brute Force time(in miliseconds):      1277

Horspool:
Number of comparisons for Horspool:    5071492
Number of occurence for Horspool:      0
Horspool time(in miliseconds):         1210

Boyer Moore:
Number of comparisons for Boyer Moore: 5071492
Number of occurence for Boyer Moore:   0
Boyer Moore time(in miliseconds):      1262
```

10110010101000001000101001000000110100001110100011011010110110000111110000010100011

There is no occurrence found as we see in our console and output file. As we can see the best algorithm for this .html and pattern combination are the Horspool and Boyer Moore algorithm. We can also see that Horspool is a little

bit better than Boyer Moore for this .html file and pattern. This is because of the Bad Symbol Table dominates the jump amount. We concluded to try also for a big .html file (8 megabytes). This is the reason that the time is much longer than the previous examples.

RESULTS/CONCLUSION

Comparison Number:

Input/Algorithm	Brute Force	Horspool	Boyer Moore
AT_THAT	1136103	168422	168392
BAOBAB	603599	110616	109953
ollo	1256275	316935	316935
011010	3896406	2939866	1885757
10101	2167227	1686793	1054302
10131	17183397	5071492	5071492

Time in Milliseconds:

Input/Algorithm	Brute Force	Horspool	Boyer Moore
AT_THAT	158	112	162
BAOBAB	94	59	41
ollo	223	113	116

011010	342	253	250
10101	219	192	192
10131	1277	1210	1262

n: length of the .html file m: length of the pattern

Brute Force: BigO(n*m)

Horspool: BigO(n*m)

Boyer Moore: BigO(n*m)

At first, we can easily see that the Brute Force algorithm's comparison and execution time is the worst among all. This is because it shifts only by 1 each time a bad match or a successful match is found. Which is very inefficiency for a string pattern matching algorithm.

We can see that the Horspool algorithm's comparison number is always equal or larger than the Boyer Moore algorithm. This is because they both use the Bad Symbol Table. But sometimes the Boyer Moore algorithms take usage of the Good Suffix Table causing it to have a smaller total comparison number. This is the reason why the execution time is nearly in all example of string pattern matchings are close to equal.