



**MARMARA UNIVERSITY
FACULTY OF ENGINEERING**

CSE-2246 Analysis Of Algorithms

Homework - 1

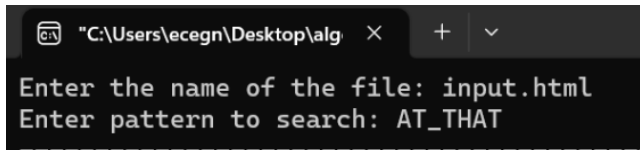
Due Date: 17.05.2023

	Department	Student Id	Name Surname
1	CSE	150120027	Mert Muslu
2	CSE	150120051	Erkut Dönmez
3	CSE	150121539	Gülsüm Ece Günay

1. PURPOSE

This homework aims to demonstrate and compare the different kinds of string-matching algorithms: the Brute Force Algorithm, Horspool's Algorithm, and the Boyer Moore Algorithm.

Firstly, our code starts with getting the name of the file from the user.

A screenshot of a terminal window with a dark background. The title bar shows the file path "C:\Users\ecegn\Desktop\alg" and standard window controls. The terminal displays two lines of text: "Enter the name of the file: input.html" and "Enter pattern to search: AT_THAT". Below the second line, there is a dashed line.

```
"C:\Users\ecegn\Desktop\alg" × + ∨  
Enter the name of the file: input.html  
Enter pattern to search: AT_THAT  
=====
```

2. OBJECTIVES

The Brute Force Algorithm checks every possible option starting from the left-most pattern until finds the correct pattern. So, this algorithm is pretty exhaustive.

The Horspool Algorithm creates a bad character table according to the occurrences of mismatch between pattern and text. In case of mismatching, shifting occurs according to the table.

The Boyer-Moore Algorithm creates a bad character table according to the mismatch and a good suffix table according to the substrings in the pattern. Comparison of the number of shifts between good and bad suffix tables determines the shifts.

According to this information, theoretically, we expect that in terms of the higher comparison and time, the list goes with Brute Force, The Horspool, and The Boyer Moore.

1. BRUTE FORCE ALGORITHM EXAMPLES:

- example 1:

INPUT: WHICH_FINALLY_HALTS. __ AT_THAT_THAT POINT
PATTERN:AT_THAT

As the working mechanism of code, it takes the file name as an input after that ask to user which pattern will be searched through the text. For the given input and the pattern which is given above, our program print out to the console wished values, which are execution time of brute force algorithm and number of comparisons to find the pattern on given input.

```
Enter the name of the file: input.html
Enter pattern to search: AT_THAT
-> Execution Time For Brute Force Algorithm = 0.004600 milliseconds
-> Comparison Number of Brute Force Algorithm = 54
Process returned 0 (0x0)   execution time : 14.641 s
Press any key to continue.
```

In this project also it is wished to mark the pattern which occurs on the input. On the homework fine note 5, we have a input and a pattern to check whether our program works properly or not. To check if our program satisfies different condition we changed input a little bit.

Original input was something like that:

WHICH_FINALLY_HALTS. __ AT_THAT POINT

But to control intertwined patterns we wrote AT_THAT_THAT instead of AT_THAT.

On the AT_THAT_THAT we should mark all of it because pattern AT_THAT occurs twice. So, our code on brute force does work with intertwined patterns.

Screenshot which is attached below shows that.

WHICH_FINALLY_HALTS. __ AT_THAT_THAT POINT

- **example 2:**

Example input which is shared on homework file!

INPUT:WHICH_FINALY_HALTS. __ AT_THAT POINT

PATTERN:AT_THAT

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: AT_THAT
-> Execution Time For Brute Force Algorithm   = 0.007300 milliseconds
-> Comparison Number of Brute Force Algorithm = 42
Process returned 0 (0x0)   execution time : 9.417 s
Press any key to continue.
```

Output file:

WHICH_FINALY_HALTS. __ AT_THATPOINT

- **example 3:**

PATTERN: 10010101101001100101111010

TEXT: 001011

10010101101001100101111010

```
-----
-> Execution Time For Brute Force Algorithm   = 0.004500 milliseconds
-> Comparison Number of Brute Force Algoritihm = 41
```

- **example 4:**

We've observed that with the larger inputs brute force algorithm works as it wished.

INPUT: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

PATTERN: Lorem

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: Lorem
-> Execution Time For Brute Force Algorithm   = 0.025900 milliseconds
-> Comparison Number of Brute Force Algorithm = 583
Process returned 0 (0x0)   execution time : 8.032 s
Press any key to continue.
```

Output file:

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

2. HORSPOOL'S ALGORITHM EXAMPLES:

- example 1:

INPUT: WHICH_FINALY_HALTS. __ AT_THAT_THAT POINT

PATTERN:AT_THAT

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: AT_THAT
-----
|          BAD SYMBOL TABLE          |
|-----|
| Char          Shift                  |
|-----|
| A              1                    |
| H              2                    |
| T              3                    |
| _              4                    |
| Others         7                    |
|-----|
-> Execution Time For Horspool Algortihm      = 0.003800 milliseconds
-> Comparison Number of Brute Force Algorithm = 0 -> Comparison Number of HorsPool Algorithm = 22
```

Output file:

| WHICH_FINALY_HALTS. __ AT_THAT_THAT POINT

- **example 2:**

Example input which is shared on homework file!

INPUT:WHICH_FINALY_HALTS. __ AT_THAT POINT

PATTERN:AT_THAT

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: AT_THAT
-----
|          BAD SYMBOL TABLE          |
|-----|
| Char          Shift                  |
|-----|
| A              1                    |
| H              2                    |
| T              3                    |
| _              4                    |
| Others         7                    |
|-----|

-> Execution Time For Horspool Algoritihm      = 0.003400 milliseconds
-> Comparison Number of Brute Force Algorithm = 0 -> Comparison Number of HorsPool Algorithm = 14
```

Output file:

WHICH_FINALY_HALTS. __ **AT_THAT**POINT

- **example 3:**

PATTERN: 10010101101001100101111010

TEXT: 001011

Output file:

10010101101001100101111010

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: 001011
-----
|          BAD SYMBOL TABLE          |
-----
|          Char          Shift        |
-----
|          1             1            |
|          0             2            |
|          Others        6            |
-----

-> Execution Time For Horspool Algortihm      = 0.002800 milliseconds
-> Comparison Number of HorsPool Algortihm = 35
```


- **example 4:**

Same inputs we tried on the brute force algorithm also works for the horspool algorithm.

INPUT: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

PATTERN: Lorem

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: Lorem
-----
|          BAD SYMBOL TABLE          |
-----
|          Char          Shift        |
-----
|          e             1            |
|          r             2            |
|          o             3            |
|          L             4            |
|          Others        5            |
-----

-> Execution Time For Horspool Algoritihm      = 0.014300 milliseconds
-> Comparison Number of HorsPool Algoritihm = 150
```

Output file:

LoremIpsum is simply dummy text of the printing and typesetting industry.
LoremIpsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing LoremIpsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of LoremIpsum.

3. BOYER-MOORE ALGORITHM EXAMPLES:

example 1:

INPUT: WHICH_FINALLY_HALTS. __ AT_THAT_THAT POINT

PATTERN: AT_THAT

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: AT_THAT

-----
|          BAD SYMBOL TABLE          |
-----
|          Char          Shift        |
-----
|          A             1            |
|          H             2            |
|          T             3            |
|          -             4            |
|          Others        7            |
-----

-----
|          GOOD SUFFIX TABLE          |
-----
|          k          Shift           |
-----
|          k = 1             3        |
|          k = 2             5        |
|          k = 3             5        |
|          k = 4             5        |
|          k = 5             5        |
|          k = 6             5        |
-----

-> Execution Time For Bayer-Moore Algortihm = 0.003400 milliseconds
-> Comparison Number of Boyer-Moore Algortihm = 22
```

Output file:

| WHICH_FINALLY_HALTS. __ AT_THAT_THAT POINT

example 2:

Example input which is shared on homework file!

INPUT: WHICH_FINALY_HALTS. __ AT_THAT POINT

PATTERN: AT_THAT

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: AT_THAT

-----
|          BAD SYMBOL TABLE          |
-----
|          Char          Shift        |
-----
|          A             1            |
|          H             2            |
|          T             3            |
|          -             4            |
|          Others        7            |
-----

-----
|          GOOD SUFFIX TABLE          |
-----
|          k          Shift           |
-----
|          k = 1             3        |
|          k = 2             5        |
|          k = 3             5        |
|          k = 4             5        |
|          k = 5             5        |
|          k = 6             5        |
-----

-> Execution Time For Bayer-Moore Algortihm = 0.002700 milliseconds
-> Comparison Number of Boyer-Moore Algortihm = 14
```

Output file:

WHICH_FINALY_HALTS. __ **AT_THAT**POINT

example 3:

PATTERN: 10010101101001100101111010

TEXT: 001011

Output file:

10010101101001100101111010

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: 001011
-----
|          BAD SYMBOL TABLE          |
-----
|          Char          Shift        |
-----
|          1             1            |
|          0             2            |
|          Others        6            |
-----

-----
|          GOOD SUFFIX TABLE          |
-----
|          k             Shift        |
-----
|          k = 1         1            |
|          k = 2         6            |
|          k = 3         6            |
|          k = 4         6            |
|          k = 5         6            |
-----

-> Execution Time For Bayer-Moore Algortihm = 0.003100 milliseconds
-> Comparison Number of Boyer-Moore Algortihm = 22
```

example 4:

Same inputs we tried on the brute force algorithm also works for the horspool algorithm.

INPUT: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

PATTERN:Lorem

Output file:

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Console output:

```
Enter the name of the file: input.html
Enter pattern to search: Lorem
-----
|          BAD SYMBOL TABLE          |
-----
|          Char          Shift        |
-----
|          e             1            |
|          r             2            |
|          o             3            |
|          L             4            |
|          Others        5            |
-----

-----
|          GOOD SUFFIX TABLE          |
-----
|          k          Shift          |
-----
|          k = 1          5          |
|          k = 2          5          |
|          k = 3          5          |
|          k = 4          5          |
-----

-> Execution Time For Bayer-Moore Algortihm = 0.025400 milliseconds
-> Comparison Number of Boyer-Moore Algortihm = 157
```

OBSERVATIONS

In this experiment we observed that Horspool's and Boyer Moore's algorithms are more efficient in terms of number of comparisons and execution time. This information leads us to instructions that we learned on the lectures. In the lectures we've seen that brute force string matching algorithm is less efficient than others, experiments we've done and the data that we collected confirms that information. However it is still hard to compare Horspool's algorithm and Boyer Moore's algorithm with each other.

On the other hand, in our code, Horspool's algorithm is more efficient in terms of time compared to Boyer-Moore's Algorithm although the number of comparisons of Horspool's Algorithm is much more than Boyer-Moore's Algorithm's number of comparisons. Execution time and number of comparisons are not efficient for just one of these algorithms as desired, probably because of our algorithms.

PROBLEMS WE ENCOUNTERED

A) FIRST PROBLEM

For our program to work properly, our input file should be one line length. Our input file may contain thousands of characters but it should be written on only one line. Otherwise, only the first line of the input will be processed.

As an example:

I scream, you scream, we all scream for ice cream.

The input given above will work since its line length is only one.

I scream,you scream,

We all scream for ice cream.

However, this input won't work for the second line and it will only process the first line.

We tried to get the input line by line but we failed. Our attempt is below:

```
/*
char input[MAX_LEN][MAX_LEN];
char str[100];

FILE input_file;
printf("Enter the name of the file:\n");
gets(str);
input_file = fopen(str, "r");
if (input_file == NULL) {
    printf("Dosya açilamadı.\n");
    return 1;
}
```

```

int row = 0;
int col = 0;

while (row < MAX_LEN && fgets(input[row], MAX_LEN, input_file) != NULL) {

    int len = strlen(input[row]);
    if (len > 0 && input[row][len - 1] == '\n') {
        input[row][len - 1] = '\0';
    }

    row++;
}

fclose(input_file);
char output[MAX_LEN];
char pattern[MAX_PAT];
printf("Enter pattern to search: ");
scanf("%s", pattern);
badSuffixTable(pattern);
goodSuffixTable(pattern);

FILE output_file;
char output_file_name[] = "output.html";
int ComparisonsForBruteForce = 0;
int ComparisonsForHorsPool = 0;
int ComparisonsForBoyerMoore = 0;
for(int i = 0; i<=row;i++) {
    bruteForce(data[i],output,pattern,&ComparisonsForBruteForce);
}
for(int i = 0; i<=row;i++) {
    int length = strlen(output);
    HorsPool(pattern,output,data[i],length,&ComparisonsForHorsPool);
}
for(int i = 0; i<=row;i++) {
    int length2 = strlen(output);
    BoyerMoore(pattern,output,data[i],length2,&ComparisonsForBoyerMoore);
}

printf("-> Comparison Number of Brute Force Algortihm = %d \n -> Comparison Number of
HorsPool Algortihm = %d \n -> Comparison Number of Boyer-Moore Algortihm =
%d",ComparisonsForBruteForce,ComparisonsForHorsPool,ComparisonsForBoyerMoore);
output_file = fopen(output_file_name, "w");
fputs(output, output_file);
fclose(output_file);
return 0;
*/

```

b) SECOND PROBLEM

For larger inputs which include more than 1 million characters, Boyer-Moore algorithm's number of comparisons are much more than Horspool's Algorithm. But we don't know how to find the correct one which algorithm gives us the true comparison number. One of them(Boyer-Moore and Horspool) is correct. But, we do not know which one is correct. Because of that we couldn't handle it. Probably, it is because of the high intensity of characters in memory(in terms of bits). When we run our code without brute force algorithm function called number of comparisons on Horspool's and Boyer-Moore algorithm has changed.

Except these no problems were encountered! 😊👍

DIVISION OF LABOR

- Algorithms(Brute Force–Boyer Moore-Horspool):
-Mert-Ece
- Checking nested pattern matches in text file
(controlAgain-controlAgainForHorspool-controlAgainForBoyerMoore):
-Mert-Ece-Erkut
- Auxiliary functions for finding nested pattern matches
(shiftingToBad -> finds shift number according to the Bad-Symbol Table
find_shift_value-> finds shift number according to the Good-Suffix Table):
-Mert-Erkut
- Table's algorithms (goodSuffixTable-badSymbolTable):
-Mert-Ece-Erkut
- Main Function (main):
-Mert-Ece-Erkut
- max:
-Erkut

REFERENCES

1. StackOverFlow
2. [Space and Time Tradeoffs \(mtu.edu\)](#)
3. Lecture Slides
4. [The Boyer-Moore-Horspool Algorithm | by Francisco Saldaña | Medium](#)