metin, küçük resim içeren bir resim

Açıklama otomatik olarak oluşturuldu

**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

Analysis of Algorithms

**Project-1**

**Brute-Horspool’s-Boyer Moore’s**

**String Matching Algorithms Comparison**

**Due Date:** 14.05.2023

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Dept** | **Student Id** | **Name Surname** |
| **1** | CSE | 150120026 | Ardacan Özener |
| **2** | CSE | 150120068 | Hasan Pekedis |
| **3** | CSE | 150121073 | Yiğit Tuncer |

**Table of Contents**

1. [**EXPERİMENT DESİGN**](about:blank)

**2.** [**RESULT OF RUNNING THE CODE**](about:blank)

[**3. RESULTS**](about:blank) **AND CONCLUSION**

[**4. GIVEN INPUT RESULT**](about:blank)

**WHO DID WHAT**

**150121073 Yigit Tuncer** -> Wrote the code for brute force algorithm and scanning, worked on “Experiment Design”, “Results and Conclusion” and “Result of Sample Text” parts of the report.

**150120068 Hasan Pekedis** -> Worked “Experiment Design” and “Results of Running the Code” parts of the project, ran all of the patterns and samples with the code.

**150120026 Ardacan Ozener** -> Wrote the code for the Boyer Moore and Horspools algorithms, Good and Bad table creation, and other parts of the code.

**1) EXPERIMENT DESIGN**

**DECIDING ON SAMPLE TEXTS**

For the HTML text files, we decided to use 3 books from the internet written in modern English, as these texts will be long enough and also, they are written in modern English so we can predict the outcome for certain patterns. For example, we can assume that there are not many words that contain the letter ‘x’ or ‘q’, but we can presume that many words contain ‘e’ or ‘a’.

The three files are like follows:

* Text Sample 1 is 24,422 lines long, contains 1,592,632 characters and it is 1.51mb. It is a book from the internet.
* Text Sample 2 is 10,285 lines long, contains 2,791,651 characters and it is 2.68mb. It is also a book from the internet.
* Text Sample 3 is 3,994 lines long, contains 1,050,729 characters and it is 1mb. It is also a book from the internet.

As for the binary HTML text files we decided to use 2 randomly generated strings using generators. These strings are created by adding a 1 or 0 with equal 50% chance. Also, we used a file containing only 0’s. The 2 random strings are to test the algorithms’ performances at the general level, and the file containing all 0’s is to easily test the best and worst cases of patterns for these algorithms.

The three files are as follows:

* Binary Random Sample 1 contains 757 lines and 2,251,550 characters. It is a completely random bit string.
* Binary Random Sample 2 contains 1505 lines and 6,009,038 characters. It is also a completely random bit string.
* Binary Zero Sample contains 740 lines and 2,221,503 characters. It is comprised of all zeros.

**DECIDING ON PATTERNS**

We chose patterns off of 4 criteria:

-How long is the pattern

-Does the pattern repeat itself

-Does the pattern contain common English letters or uncommon English letters

-What character does the pattern start with, but most importantly what does It end with.

How It ends is really important for Horspools and Boyer-Moore’s algorithms, because if it is very uncommon, not many comparisons will be done; because it is done from the end of the word to the start of the word. But for the brute force method, the word is read from the start to the end, so the start of the word Is more important and the end is irrelevant.

**The patterns we chose for the English part are like so**:

The comments made for these patterns are mostly about the Horspool’s and boyer-moore algorithms. Brute force should not change as significantly for different patterns, since it will compare each time either way.

**likelike** -> This is a short repeating pattern. It is expected that many comparisons are made because it contains a common English ending as well.

**repeatrepeatrepeatrepeatrepeat** -> This is a long repeating pattern, so it is expected that there will be some long jumps and some short jumps. It is hard to predict how this pattern will act relative to the other patterns.

**gigahertz** -> This is a short word with an uncommon ending. It is expected that this will have a short runtime and not too many comparisons as it will not match a lot.

**gigahertz is frequency equal to one billion hertz** -> This is a long pattern with an uncommon ending. We could predict that this will have a very short runtime in comparison to the others, as it will shift a lot.

**because** -> This is a short common word with common letters and a common ending. It is expected that this will have a relatively longer runtime and more comparisons.

**centuries are long times** -> This is a longer pattern, so it is expected that longer shifts are going to happen. Thus, it would be logical to expect shorter runtime and less comparisons.

**baobab** -> This is a short random word. It is difficult to make a prediction on how it will act.

**qqqqqqqqqqqqqqqqqqqq** -> This will probably be the best case for most samples, because it will shift by 20 almost each time, because there won’t be many q’s in the text.

**The patterns we chose the patterns for the binary samples:**

**100100100** -> This is a repeating pattern, so it is expected that this will have many comparisons and a long running time

**1000000** -> This is the worst case for the all 0’s sample, because it will compare with 0’s a lot. On the other random samples, it will probably result in a long runtime as well.

**10101010101010101010101010101010** -> This is a long repeating pattern and will probably have a longer runtime because it will jump with smaller values compared to if it did not repeat.

**101001100** -> This is a random short string that will also probably have a relatively long execution time.

**111110100101110011011101001000000100000100000** -> This is a random long string a will probably have a relatively short execution time because of the number of comparisons.

So, in general, our predictions can be summarized as follows. We expect longer English patterns to have a shorter runtime, because they will jump further ahead in the text, because they are longer. The more uncommon letters the pattern has the shorter its runtime will be, because the number of comparisons will drop. This effect will increase when the uncommon character is in the start of the pattern for brute force, and for boyer-moore and Horspools this effect will increase when the uncommon character is at the end of the pattern. This is because brute force scans from the beginning to the end of the pattern while boyer-moore scans from the end to the start. The repeating texts will increase the runtime, because this will shorten the distance the pattern will jump in the text for matches. Patterns starting with common characters will have longer runtimes for brute force because mor comparisons will be done. Similarly for Horspools and boyer-moore patterns ending with common characters will have longer runtimes as well.

As for the binary patterns, the longer patterns are expected to have longer runtimes for the brute force algorithm because this will just increase the number of comparisons. As for the Horspools and boyer-moore algorithms, this will probably shorten the execution time with longer jumps. Repetition in the pattern will probably cause longer runtimes compared to non-repeating patterns, because they will jump further ahead.

**RESULT OF RUNNING THE CODE**

**RESULTS AND CONCLUSION**

**TIME COMPLEXITY**

While making comments on the results, its important to keep in mind what exactly we are analyzing. The number of iterations for these iterative algorithms grow or shrink with the size of the text we are searching in, if we assume the same pattern. The main thing the number of iterations change Is the number of comparisons. So, the way the algorithms running time changes according to the input and patterns is dependent on these 2 variables. The notation for this concept is big Theta.

**Average time complexities for string matching algorithms**

Brute force -> Θ (n \* m)

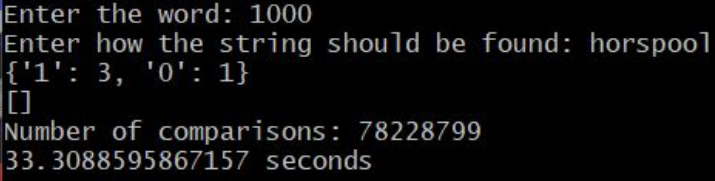
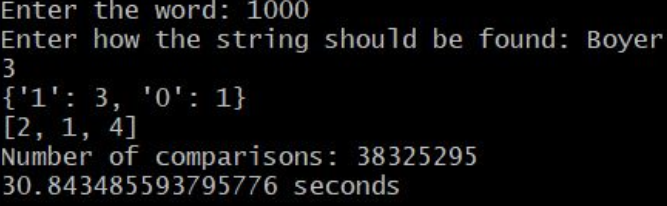
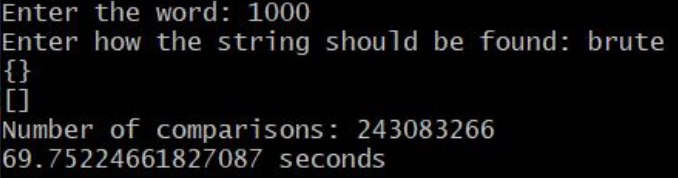
Horspool’s algorithm -> O (n + m)

Boyer Moore’s algorithm -> O (n + m)

The time complexity for brute force is worse than the others. For almost all inputs, Horspool’s and Boyer Moore’s algorithms will have a lot a smaller number of comparisons than Brute force. After a certain input size, brute force will also have longer run time for all inputs as well. But only after a certain size in inputs. So, it can be expected that for smaller inputs, the running time may not differ that much, and in fact brute force may be even better in terms of run time, though this would be rare. This is based on a few reasons:

1. All algorithms have the same worst case time complexity.
2. Creating the good and bad tables are sometimes costly. This difference is much more visible in smaller inputs, but is not visible at all in larger ones. This is illustrated in the following example:

We executed the code for a 157MB binary text file, to illustrate that the Horspool’s and Boyer Moore algorithms are more efficient. We used the pattern 1000. Here are the results



As is clearly visible, Horspools and Boyer Moore’s algorithms are much more efficient. The number of comparisons is huge for brute force, While Horspool’s and Boyer Moore’s are a lot more reasonable, with Boyer Moore being the best out of them all. This is a very expected result and in all big files this result can be assumed to happen. As said previously this is may not be the case for smaller inputs, since the time to create the tables becomes more blatant in the comparison. But the number of character comparisons will always be better, as is shown in our results.

Now that we have gone over the time complexity created by comparisons, creation of the good comparison table and the Bad comparison table, we can talk about other factors as well. One of these is the cost of loading the text into the array. The program takes the entire text into an array line by line. This happens no matter which algorithm is chosen. After this, it searches for the string in each line using the chosen algorithm, and then loads the line back into the array, edited. This also takes time but it is done for each algorithm as well so should not affect the comparison.

The sample texts that have more lines should have a worse run time, since the program works line by line. The binary pattern with all 0’s will have a very biased outcome, since strings containing 1’s will have much better run times than strings containing 0’s.

It is also important to consider that normally the worst case for these algorithms would be the case where no match is found. But here It is the opposite, since we are searching for all occurrences. When a match is found there is an operation to add the <MARK> tags. This is costly, so it is expected that patterns that match less might have better complexities.

The running time is also affected a lot by the system hardware. For example, if there are a lot of background processes running while executing the code, the run time will be affected. This will not affect the number of comparisons though.

**SPACE COMPLEXITY**

Our program takes the entire text into the program memory in the form of an array, then applies the selected algorithm for the current line and repeats this process until no lines are left. In this sense the program is not very memory efficient. But this is better for the run time, since the scanning operation is done only once.

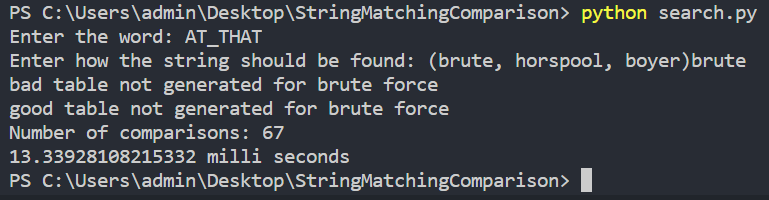
**COMMENTS ON RESULTS**

**GIVEN INPUT RESULT**

As stated in the homework pdf, here is the result of all 3 algorithms when it is run with the pattern AT\_THAT and text WHICH\_FINALLY\_HALTS. \_ \_ AT\_THAT POINT

1. BRUTE FORCE:

Console output:

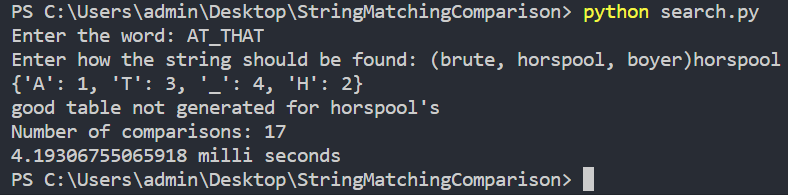


HTML output



1. HORSPOOL’S:

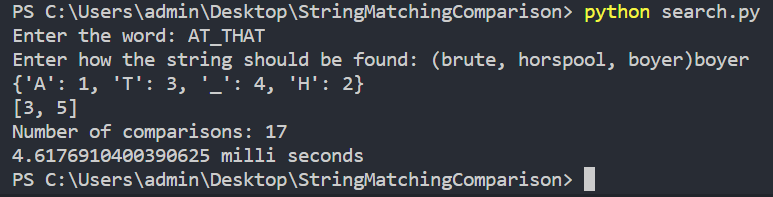
Console output:



HTML output:

1. BOYER MOORE’s:

Console Output:



HTML Output:



All of the source code for the HTML outputs are the same, which is:

<HTML><BODY>WHICH\_FINALLY\_HALTS. \_ \_ <mark>AT\_THAT</mark> POINT </BODY></HTML>