**Analysis of Algorithm Project1**

Melik Özdemir 150120004

Ahmet Abdullah Gültekin 150121025

Ömer Deligöz 150120035

Emre Gürkan 150121823

In this project, we are expected to design an experiment to compare following three string matching algorithms:

1. Brute-force string matching,

2. Horspool’s algorithm,

3. Boyer-Moore algorithm.

Firstly, we made discussions for choosing the sensible html files which has English words. As a conclusion of division of labor, we decided to handle 3 different subjects that assist us for observing from different ways.

1-Books

2-Movies

3-Articles

**Codes And Descriptions: (INPUT/OUTPUT):**

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

In this part, we have a class which has called main. In this class, there is printResults method that creates a html file and print the results into the that file.

**metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Because of markText method, the matches that are obtained from the html file, is marked and written to the output file. Firstly, we have Collection.reverse in the first line, it reverses the order of elements. Additionally, we have variables that are called startIndex and endIndex.

In the first for loop, the elements are checked one by one. And endIndex assigned to the current element that is in list. Then, first if condition checks these:

1-Is the current element last element?

2-Is not the difference between the current element and the next element is less than the size parameter(Overlapping)?

If the answer of these questions are ‘NO’, it means the condition is true and inner for loop executed. In the inner for loop, the startIndex assigned to the value of the current element in the list. If ‘j’ is equal to ‘list.size()-1’, it means it is the last element in the list. And result will be updated. Otherwise, there will be overlapping.

**metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Firstly, we give the source of html file to the program. Additionally, we chose a pattern for testing. We have a list which has int type. The list holds the first character of matches. And we created 3 variables to calculate total time. Moreover, we have try catch command for inspecting the errors.

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

In this part, we have some variables such as startTime, endTime, occurences. According to the homework, we should find total time. These variables are used for calculating the total time. Additionall, this part is used for BruteForceAlgortihm. We have calling from the BruteForceAlgorithm class. And results are printed to console.

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

In this part, we have some variables and class calling same as above. That part is used for Horspool Algorithm. In the same way, results are printed to the console.

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Unlike the others, in this part, Bad symbol and Good suffix are also written to the console.

**RandomBitStrings Class:**

**metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

As desired we have a random bit strings method to provide us a text maker. This function creates a new HTML file and fills it with zeros-ones. We need this method because we will use a text exist of zeros-ones for the comparison experiment. Eventually, unless you encounter error random bit strings will be generated and be ready for the usage. Also, you can determine the size of HTML file.

**Horspool Algorithm Class:**

**metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

In this parti we have a Map with Character keys and Integer values. This map is used for storing the bad symbol table for the Horspool algorithm. Additionally, we have ‘findAllOccurrences’ method that initializes a counter to keep track of the number of comparisons. There is an empty array that is called matches. The array holds indices of all occurrences of the pattern in the html files.

Additionally, we have 2 variables that equal to length of text and pattern.’badSymbol’ map is initialized as a new HashMap.

If the program finds a match, it will add that match to the array that is called match. Finally, until we cannot reach the end of the file, the program will handle comparisons.

**BruteForceAlgorithm Class:**

**metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

In the same way, the program will check the matches. If ıt find a match, the program will add it to the array that called occurences. Then, because of comparisonCounter variable, number of comparisons will be printed to the console for BruteForce Algorithm.

**BoyerMooreAlgorithm Class:**

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

In this part, we have an int list that is called goodsuffix and we have a map with character keys and integer values. First of all, the variables n (length of text), m(length of pattern), matches and comparisonsCounter are initialized. And BadSymbol and goodSuffix tables are created according to rules of BoyerMoore algorithm.

**metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**

We have variables that are called ‘i’ and ‘j’. ‘i’ variable navigates the text, ‘j’ variable navigates the pattern. If a match can be found, the match is added to ‘matches array’. If the match equals 0, we only check the badSymbol table. If the match doesn’t equal 0, we should check both tables. Finally, the number of comparisons is printed, and matches are returned.

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Unlike the others, in this part we have a method that is called createBadSymbol for creating bad symbol tables.

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

The method takes 1 parameter, the pattern string. The outer loop starts from the first element of the table and carry on until the last element. The suffix variable is assigned the substring of the ‘pattern’. ‘previousCh’ variable is assigned the character preceding te suffix in the ‘pattern’.

In the inner for loop, the suffix is checked whether it is in pattern or not. If i is 0 it means the suffix occurs at the beginning of the pattern and the table value is updated. If previousCh is not equal to the character preceding the current suffix the table valjue is updated

**Analyzing of Results and Inferences:**

When we tried all the possibilities that came to our mind, we got the following as a result:

1-We chose a pattern that is in the text. For example**, ‘end’** is in the text. When we compared the performance of all three algorithms, we noticed that the **Brute Force algorithm is fastest algorithm for this pattern. Boyer Moore algorithm and Horspool algorithm worked for almost same time**.

2-We chose a pattern that is **long and is in the text**. For example, ‘**at the end of’** is long and is in the text. When we compared the performance of all three algorithms, we noticed that the **Brute Force algorithm is the slowest algorithm for this pattern.** And **it runs twice as slow.** And **as the text gets shorter**, we see that the **performance of all three algorithms run in almost equal time. And if the length of the pattern is much shorter Brute Force algorithm runs fastest.**

3-We chose a pattern that is long and is not in the text. For example, ‘**Congratulations’** is the pattern. When we compared the performance of all three algorithms, we noticed that the **Brute Force algorithm is slowest algorithm for this pattern**. As a result of that test, we noticed that **there is no difference between whether there is a match or not**.

4-We chose a pattern that has punctuation. For example, ‘**etc.**’ is the pattern. When we compared the performance of all three algorithms, we noticed that **all three algorithms have almost the same execution time**. As a result of that test, **we can inference that punctuation affects execution time.**

5- If it is necessary that we choose a pattern as a bit-string. For example, ‘**100100100**’ is pattern. When we compared the performance of all three algorithms, we noticed that if the pattern is **repetitive**, the **Boyer Moore algorithm performs significantly fewer comparisons**. As a result of, if we choose a pattern that is repetitive bit string, the order from **fast to slow** goes like this:

**Boyer Moore > Horspool > Brute Force.**

6-If we choose a pattern as a one bit, the conclusion is different from item 5. **For example, ‘1’ or ‘0’ is pattern**. When we compared the performance of all three algorithms, we noticed that **if the pattern is one bit, The fastest algorithm is Brute Force algorithm**. Additionally, **the number of comparisons is same**. As a result of that test, we can say that when we select a short bit for the pattern, **we see that the Brute Force algorithm runs the fastest, as when we select a short English word for the pattern**.

7-If we choose a pattern as a normal bit string. For example**, ‘101’** is pattern. When we compared the performance of all three algorithms, we noticed that **the slowest algorithm, Horspool algorithm**. We expect that the total times of Horspool and Boyer Moore should be te same. But in this case that is not True. Because **Boyer Moore algorithm uses the tables that were calculated**. As a result of that test, **if we choose the pattern as a non-repetitive, the slowest algorithm is Horspool algorithm.**

**TRYING OF SAMPLE INPUT AND CONCLUSION:**

**When we try the sample input, the results like that:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

Firstly, when we try the sample input, we saw that the total times of Horspool and Boyer Moore was 0 milliseconds. After some executions, we saw that the total times of Horspool and Boyer Moore was 1 or 2 or 3 milliseconds. **We inference that reason of the 0 milliseconds is fraction part**. **On average, Horspool and Boyer Moore have an operating speed of between 0 millisecond and 1 millisecond**.

Moreover, we notice **that the slowest algorithm is Brute Force Algorithm**. Unlike previous results, the pattern is short, and **we expected that the fastest algorithm should be Brute Force.** **But this didn’t happen as expected**. And we inference that **if the text is very short, Horspool and Boyer Moore algorithms runs faster than Brute Force Algorithm**

**Division of Labor:**

**Melik Özdemir: Finding and Analyzing HTML Files (Books), Code and Description (Report)**

**Ahmet Abdullah Gültekin: Finding and Analyzing HTML Files (Articles), Code and Description(report)**

**Ömer Deligöz: Implementation of Program Classes, Finding HTML files (Bit-String)**

**Emre Gürkan: Finding and Analyzing HTML files (Movies), Analyzing and Inferences(Report)**