Who did what:

150121073 Yigit Tuncer -> Wrote the code for brute force and bilmem ne project rapor falan

---------------Hasan Pekedis -> wrote project report yaz biseler buraya

---------------Ardacan Ozener -> Wrote the code for the Boyer Moore and Horspools algorithms

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 it should not cause a problem as python is very good at handling strings. Since the program works line by line, it would be expected that more lines would result in worse performance, as these lines are worked on separately.

**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 it is very important.

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

The comments made for these patterns are mostly about the horspool and boyer-moore algorithms. Brute force should not change too significantly for different patterns.

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.

Now all that’s left to do is to test our hypothesis by running our program.