IT251 – DATA STRUCTURES & ALGORITHMS II

ASSIGNMENT 6

Name: **Sachin Prasanna**
Roll No.: **211IT058**

Problem Statements:

• Lab 6 – Study Boyer Moore Algorithm and bring out the following aspects of Boyer Moore Method in your code
• Bad Character Heuristic •
Good Suffix Heuristic
Mark Distribution: (Total – 10 Marks)
A One page (max 4 paragraphs containing 5 statements each) on Boyer Moore Algorithm (2 Marks)
A running and working implementation of Boyer-Moore in language of your choice (4 Marks) Time Analysis of Boyer-Moore with respect to string matching (2 Marks)
Your perspective on Boyer-Moore strengths & weaknesses (2 Marks)

Answers

1)

Please refer to the code file attached with the folder. I have found if the pattern is present or not in the given text, and give the indices of occurrences, if present.

Firstly, I have found it using only the bad character heuristic of the Boyer Moore algorithm. Secondly, only the good suffix heuristic of the Boyer Moore algorithm is used to find the patterns.

Lastly, both the heuristics are taken advantage of and combinedly written in one function.

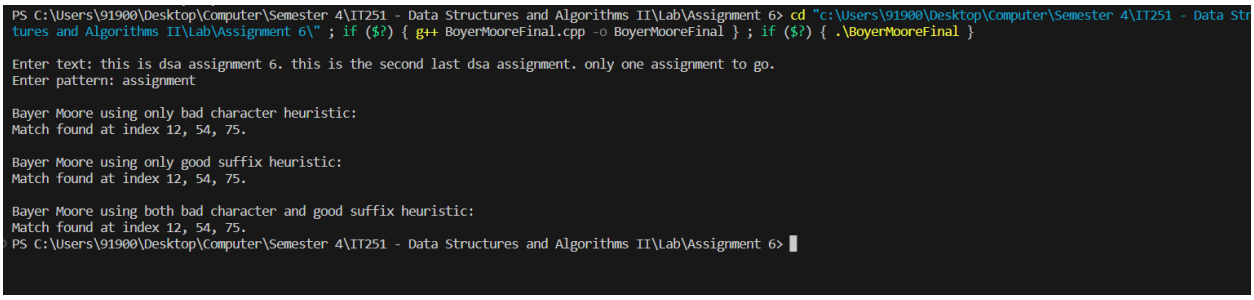
2)

One Pager on Boyer Moore Algorithm is attached in a different file in the same assignment folder.

3)

Outputs (4 different test cases tried):

Text: this is dsa assignment 6. this is the second last dsa assignment. only one assignment to go.
Pattern: assignment



Text: abababab
Pattern: aba

```
PS C:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Structures and Algorithms II\Lab\Assignment 6> cd "c:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Struc
tures and Algorithms II\Lab\Assignment 6\" ; if ($?) { g++ BoyerMooreFinal.cpp -o BoyerMooreFinal } ; if ($?) { .\BoyerMooreFinal }

Enter text: abababab
Enter pattern: aba

Bayer Moore using only bad character heuristic:
Match found at index 0, 2, 4.

Bayer Moore using only good suffix heuristic:
Match found at index 0, 2, 4.

Bayer Moore using both bad character and good suffix heuristic:
Match found at index 0, 2, 4.
PS C:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Structures and Algorithms II\Lab\Assignment 6>
```

Text: you can ask questions, discuss what you've learned from this problem, or show off how many days of streak you've made!
Pattern: you

```
PS C:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Structures and Algorithms II\Lab\Assignment 6> cd "c:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Struc
tures and Algorithms II\Lab\Assignment 6\" ; if ($?) { g++ BoyerMooreFinal.cpp -o BoyerMooreFinal } ; if ($?) { .\BoyerMooreFinal }

Enter text: you can ask questions, discuss what you've learned from this problem, or show off how many days of streak you've made!
Enter pattern: you

Bayer Moore using only bad character heuristic:
Match found at index 0, 36, 106.

Bayer Moore using only good suffix heuristic:
Match found at index 0, 36, 106.

Bayer Moore using both bad character and good suffix heuristic:
Match found at index 0, 36, 106.
PS C:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Structures and Algorithms II\Lab\Assignment 6>
```

Text: hi john, smith and roger. how are you all today?
Pattern: peter

```
PS C:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Structures and Algorithms II\Lab\Assignment 6> cd "c:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Struc
tures and Algorithms II\Lab\Assignment 6\" ; if ($?) { g++ BoyerMooreFinal.cpp -o BoyerMooreFinal } ; if ($?) { .\BoyerMooreFinal }

Enter text: hi john, smith and roger. how are you all today?
Enter pattern: peter

Bayer Moore using only bad character heuristic:
No matches found.

Bayer Moore using only good suffix heuristic:
No matches found.

Bayer Moore using both bad character and good suffix heuristic:
No matches found.
PS C:\Users\91900\Desktop\Computer\Semester 4\IT251 - Data Structures and Algorithms II\Lab\Assignment 6>
```

4) Time Analysis of the Boyer Moore Algorithm

The Bayer Moore algorithm is much better than the Naïve String-matching algorithm and achieves this by utilizing two heuristics: the "bad character rule" and the "good suffix rule." The time complexity of the Boyer-Moore algorithm can be analysed in the best-case, worst-case, and average-case scenarios.

Say the text is: abracadabra

Best Case:

In the best-case scenario, when the pattern to be matched does not occur in the text, the Boyer-Moore algorithm has a time complexity of O(n/m), where n is the length of the text and m is the length of the pattern. This is because the algorithm only performs a single comparison per character, and there are no shifts required and we can directly get to the end of string as fast as possible. For example, in the best-case scenario, the pattern will be “pqr”.

Worst Case:

In the worst-case scenario, the time complexity of the Boyer-Moore algorithm is O(n*m). This occurs when the pattern occurs multiple times in the text, and each occurrence aligns with the last character of the pattern. In such cases, the algorithm performs a comparison for every character in the text and shifts the pattern by one position until it finds a mismatch. For example, if the pattern is “abra”. Now, the worst case occurs when all occurrences of “abra” align with the last character “a” in the text.

Average Case:

The average case time complexity of the Boyer-Moore algorithm is O(n), where n is the length of the text string. This is because the algorithm uses a table to store the last occurrence of each character in the pattern, which allows it to skip over characters that do not match the pattern. This can significantly reduce the number of comparisons that need to be made. In practice, the Boyer-Moore algorithm often demonstrates excellent average-

case performance and has shown that it is better than various other algorithms.

The following table summarises the pre-processing time and matching time of different algorithms in comparison to the Bayer Moore algorithm.

Algorithm	Pre-processing Time	Matching Time
Naive Approach	O	(O (n - m + 1) m)
Rabin-Karp	O(m)	(O (n - m + 1) m)
Knuth-Morris-Pratt	O(m)	O (n)
Boyer-Moore	**O (|*|)**	**(O ((n - m + 1) + |*|))**

Where, m is the length of the pattern, n is the length of the text and * is the length of the alphabet or the number of unique characters in the input string.

5)

Advantages

1. Speed: The on the face of advantage of the Bayer Moore algorithm would be its speed. Even on average cases, the algorithm is much faster compared to other algorithms. This can be attributed to the pre-processing steps involved in the Bayer Moore algorithm.

2. Simple: Although not as simple to implement as the Naïve string-matching algorithm, the algorithm was relatively simpler to implement.

3. Long Patterns: The algorithm becomes much more efficient whilst trying to match longer patterns or more unique characters because of its pre-processing steps.

Disadvantages

1. Space: One disadvantage I felt that the algorithm might have is the extra space needed to create and store the badCharacter and goodSuffix tables.

2. Worst Case: Although a significant improvement over the other algorithms, it still has cases where the complexity can spill to O(mn), which is bad.

3. Pre-processing: As the pre-processing overhead is directly proportional to the size of the alphabet, it can be significantly high if the size of the alphabet is high.

All in all, the algorithm has great potential and comes in very handy in a lot of situations despite of its certain disadvantages. It also be said that the advantages by far outweigh the disadvantages making it highly efficient.

****