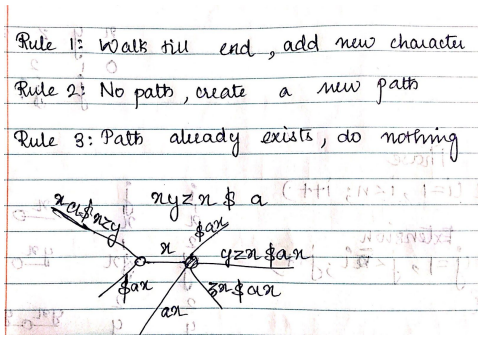
STRAIGHTFORWARD VS SUFFIX TREE TRAVERSAL IMPLEMENTATION ANALYSIS & COMPARISON

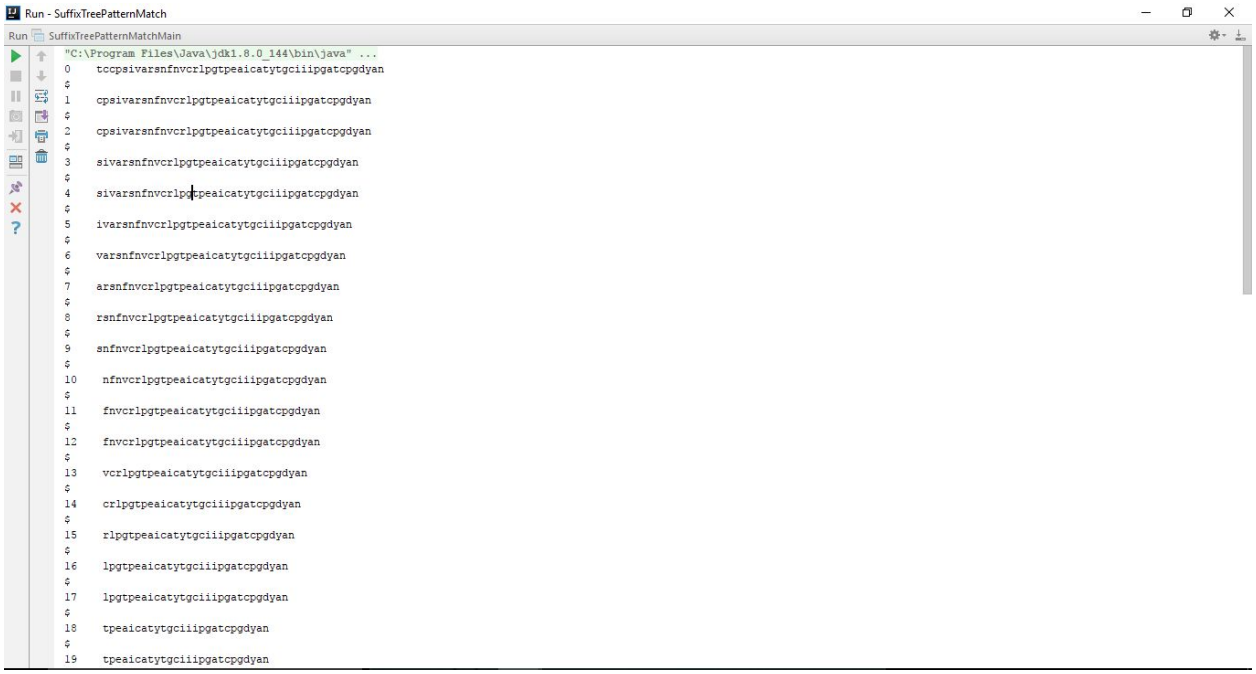
Based on Length of String and Length of Pattern.

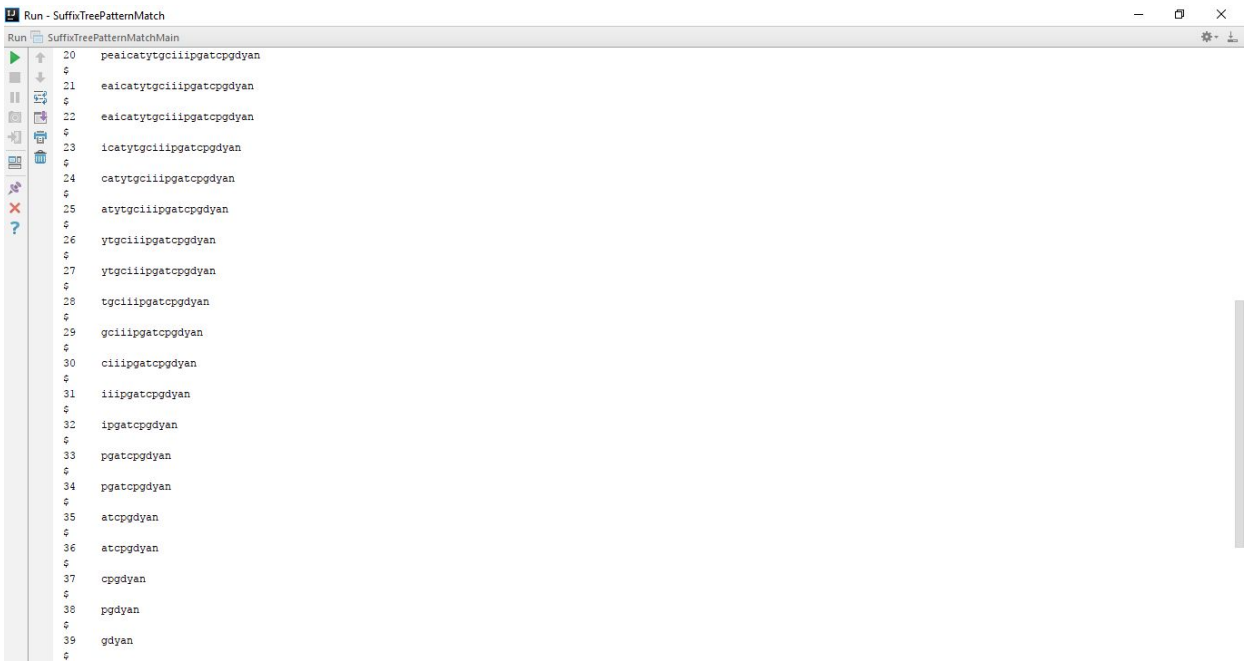
Construction of Suffix tree using Ukkonen's algorithm takes O(n) time where n is the length of the string and the traversal takes O(m) time for the length of the pattern.

```
//traverse for each character in the tree
public void traverseAfterBuild(LinkedHashMap patternHashMap,Node rootNode, List<Character> output) {
    //If root node is null, no traversal
    //empty string in tree
    if (rootNode == null) {
        return;
    }
    //if current position is not equal to -1
    if (rootNode.charPos != -1) {
        //StringBuilder builder = new StringBuilder(textToCheck.length);
        //from start to end
        for (int i = rootNode.start; i <= rootNode.finalPos.finalPos; i++) {
            output.add(textToCheck[i]);
        }
       // suffixHashMap.put(rootNode.charPos,builder.toString());

        for (int i = rootNode.start; i <= rootNode.finalPos.finalPos; i++) {
            output.remove( index: output.size() - 1);
        }
        return;
    }

    for (int i = rootNode.start; i <= rootNode.finalPos.finalPos; i++) {
        output.add(textToCheck[i]);
    }
    for (Node node : rootNode.child) {
        traverseAfterBuild(patternHashMap,node, output);
    }
    for (int i = rootNode.start; i <= rootNode.finalPos.finalPos; i++) {
        output.remove( index: output.size() - 1);
    }
}
```



Traversal of first test case in suffix tree-Step by step shown in below screenshots.





Straightforward implementation

```
//Check if the given pattern matches
public boolean checkPattern(BufferedWriter writer, FileUtils fileUtilsObj, char[] patternArr, char[] textArr) {
    for (int i = 0; i < textArr.length - patternArr.length + 1; i++) {
        for (int j = 0; j < patternArr.length; j++) {
            if (textArr[i + j] == patternArr[j]) {
                if (j == patternArr.length) {
                    return true;
                }
            }
        }
    }
    return false;
}
```

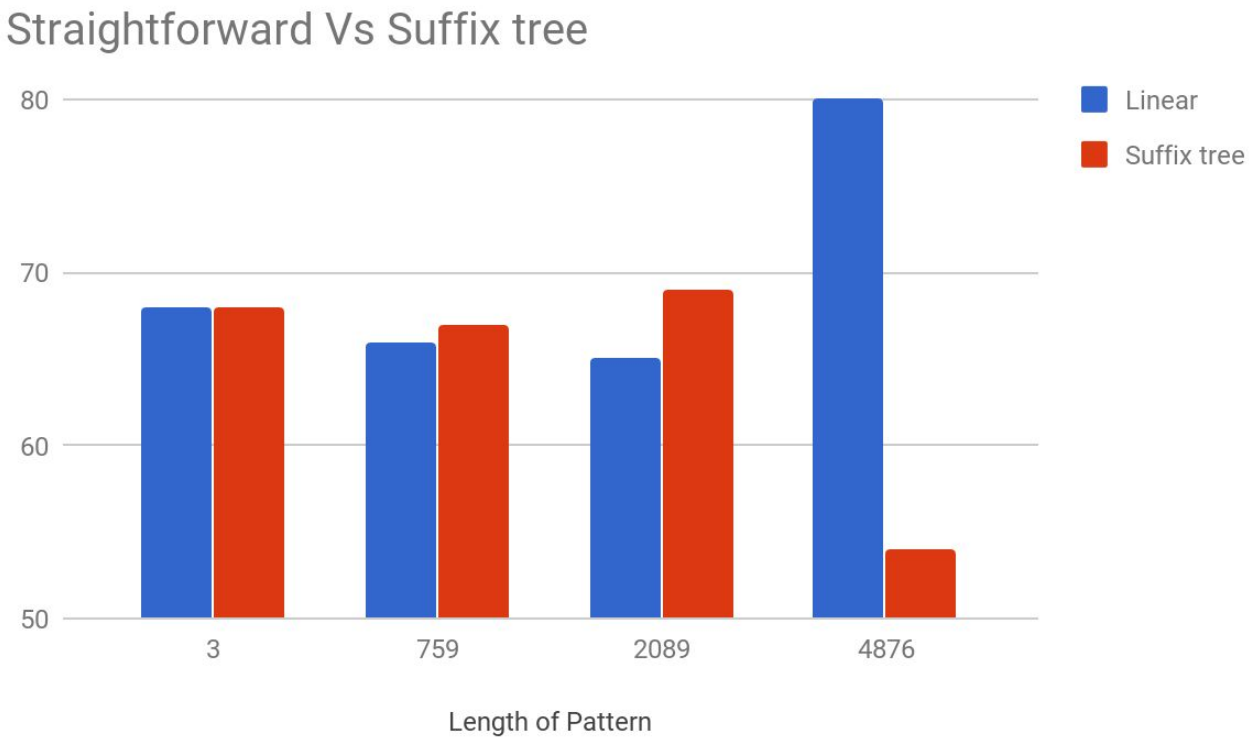
```
//Get the position of the substring
    public int getPos(String text, String pattern) {
        int j=0;
        if (pattern.length() >= 1) {
            for (int i = 0; i < text.length(); i++) {
                if (text.charAt(i) == pattern.toString().charAt(j)) {
                    j++;
                    if (j == pattern.length()) {
                        return i - pattern.length()+1;
                    }
                } else {
                    j = 0;
                }
            }
        }
        return -1;
    }
}
```

While the straightforward approach, takes O(n*m) time ,n being the length of the string and m being the length of the pattern, i.e., time taken is m*n time which isn't desirable as it takes too much time and space.

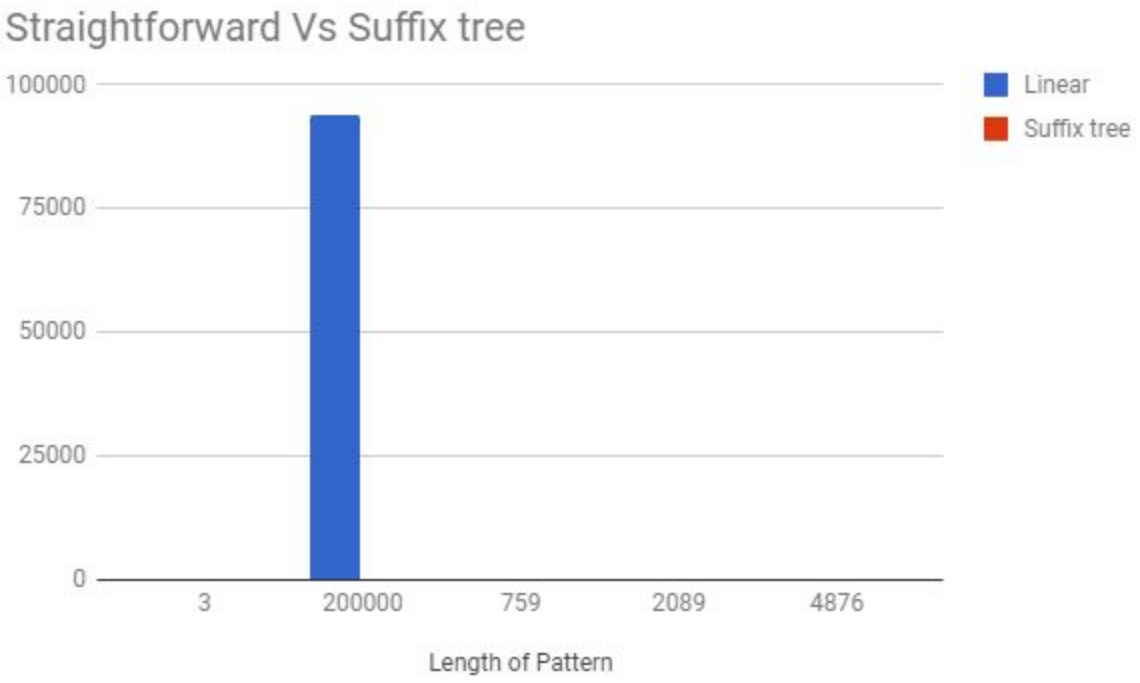
Comparison of Straight forward approach Vs Suffix tree approach without including the time for tree construction

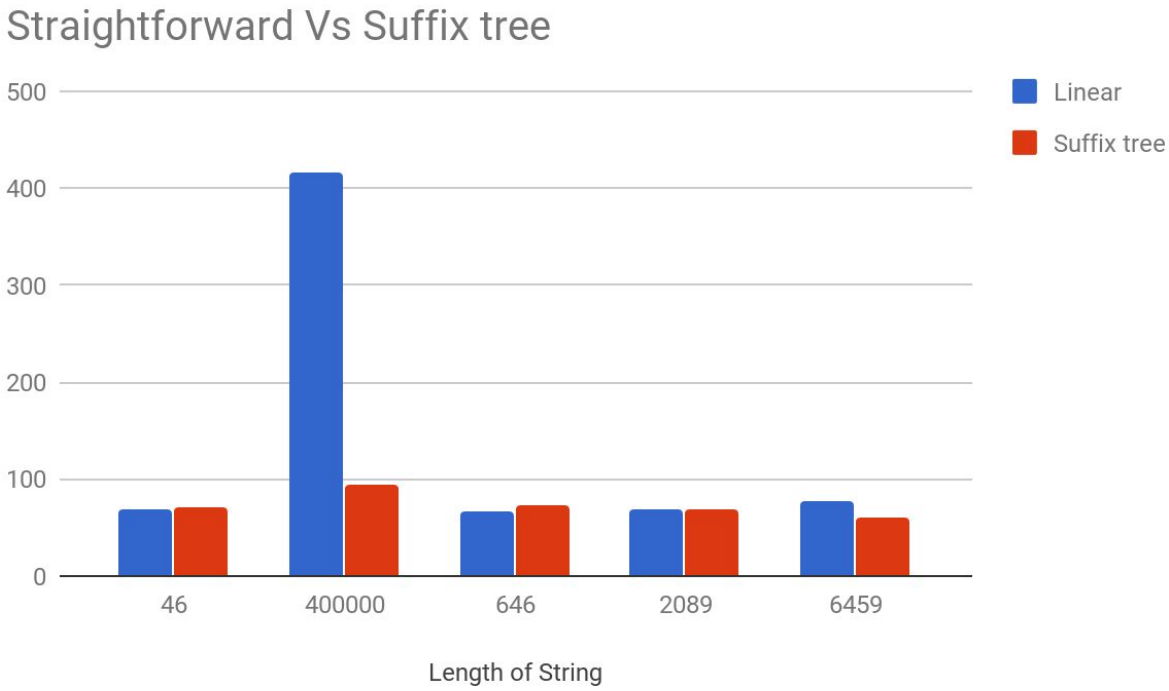
Based on 5 testcases(2 given and 3 self experimentation)

Number of characters in given string and pattern and the time in ms.



WHEN THE LENGTH OF PATTERN IS 2,00,000 AND LENGTH OF STRING IS 4,00,000 IT TAKES 94023ms,find the below graph for the same


Straightforward Vs Suffix tree
100000
75000
50000
25000
0
3
200000
759
2089
4876
Length of Pattern
Linear
Suffix tree


Straightforward Vs Suffix tree
500
400
300
200
100
0
46
400000
646
2089
6459
Length of String
Linear
Suffix tree

References:

1. http://www.geeksforgeeks.org/pattern-searching-set-8-suffix-tree-introduction/
2. http://www.geeksforgeeks.org/ukkonens-suffix-tree-construction-part-1/
3. http://www.geeksforgeeks.org/suffix-tree-application-1-substring-check/
4. https://www.youtube.com/watch?v=aPRqocoBsFQ
5. http://www.geeksforgeeks.org/suffix-tree-application-1-substring-check/
6. https://web.stanford.edu/~mjkay/gusfield.pdf