Boyer Moore String Matching

Notebook:	Algorithms		
Created:	28/03/2018 18:39	**Updated:**	28/03/2018 19:41
Author:	timtom		
Tags:	Boyer Moore, Java		

Boyer Moore String Matching algorithm

This algorithm is better than NAIVE Algorithm.

Why?

P: word

```
T: There would have been a time for such word
           word
           ---->
```

In naive algorithm the word is checked from left to right for matched patterns.

Since there's no **"u"** in the text word in **P.**

It will break out from the **inner loop** and check the next iteration from the **outer loop.**

Which means that it will skip the word **would** until it gets to the next **"w".**

This is effective but it has pointless steps, which results inefficacy.

Boyer Moore searches for strings from right to left (opposite).

Boyer Moore: Bad Character Rule:

Upon mismatch, two things will happen.

The mismatch becomes matched or the pattern P moves past along the mismatched character.

```
P: C C T T T T G C
          <-------
T: G C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
```

```
T:G C T T C T G C
          /
          /
        /
      V
P:C C T T T T G C
```

Now for the **C** to match again the **T** has to look for the next **C** in the next word in **P**.

It seems that the **P** has to be shifted in **three positions** in order to get a **match** with the **T**.

```
T:  T C T G C T A C
P :C C T T T T G C
       <------------
```

What happens is that the is has to be matched, for that to occur, we have to match it with a character before the two **C's** and move the rest to the right to get the match.