The Boyer-Moore Algorithm

Robert Boyer and J Strother Moore established it in 1977. The B-M String search algorithm is a particularly efficient algorithm and has served as a standard benchmark for string search algorithm ever since.

The B-M algorithm takes a 'backward' approach: the pattern string (P) is aligned with the start of the text string (T), and then compares the characters of a pattern from right to left, beginning with rightmost character.

If a character is compared that is not within the pattern, no match can be found by analyzing any further aspects at this position so the pattern can be changed entirely past the mismatching character.

For deciding the possible shifts, B-M algorithm uses two preprocessing strategies simultaneously. Whenever a mismatch occurs, the algorithm calculates a variation using both approaches and selects the more significant shift thus, if make use of the most effective strategy for each case.

The two strategies are called heuristics of B - M as they are used to reduce the search. They are:

1. Bad Character Heuristics
2. Good Suffix Heuristics

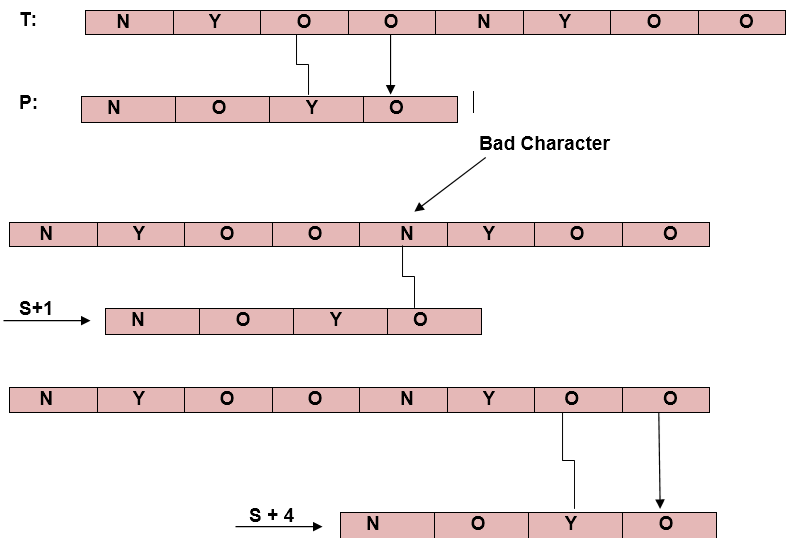
## **1. Bad Character Heuristics**

This Heuristics has two implications:

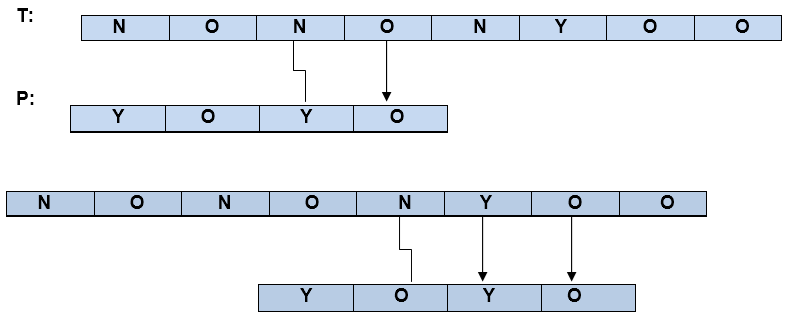
* Suppose there is a character in a text in which does not occur in a pattern at all. When a mismatch happens at this character (called as bad character), the whole pattern can be changed, begin matching form substring next to this 'bad character.'
* On the other hand, it might be that a bad character is present in the pattern, in this case, align the nature of the pattern with a bad character in the text.

Thus in any case shift may be higher than one.

**Example1:** Let Text T = <nyoo nyoo> and pattern P = <noyo>



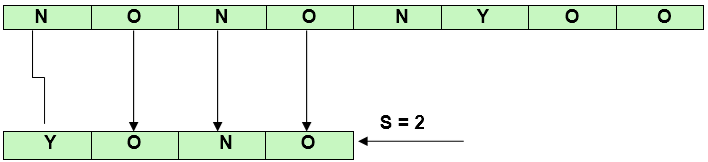
**Example2:** If a bad character doesn't exist the pattern then.



### **Problem in Bad-Character Heuristics:**

In some cases, Bad-Character Heuristics produces some negative shifts.

### **For Example:**



This means that we need some extra information to produce a shift on encountering a bad character. This information is about the last position of every aspect in the pattern and also the set of characters used in a pattern (often called the alphabet ∑of a pattern).

**COMPUTE-LAST-OCCURRENCE-FUNCTION (P, m, ∑ )**

1. for each character a ∈ ∑

2. do λ [a] = 0

3. for j ← 1 to m

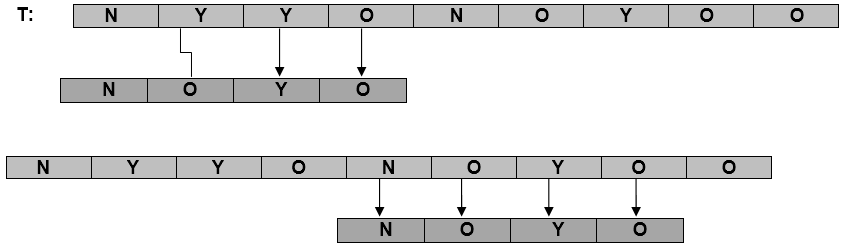
4. do λ [P [j]] ← j

5. Return λ

## **2. Good Suffix Heuristics:**

A good suffix is a suffix that has matched successfully. After a mismatch which has a negative shift in bad character heuristics, look if a substring of pattern matched till bad character has a good suffix in it, if it is so then we have an onward jump equal to the length of suffix found.

### **Example:**



**COMPUTE-GOOD-SUFFIX-FUNCTION (P, m)**

1. Π ← COMPUTE-PREFIX-FUNCTION (P)

2. P'← reverse (P)

3. Π'← COMPUTE-PREFIX-FUNCTION (P')

4. for j ← 0 to m

5. do ɣ [j] ← m - Π [m]

6. for l ← 1 to m

7. do j ← m - Π' [L]

8. If ɣ [j] > l - Π' [L]

9. then ɣ [j] ← 1 - Π'[L]

10. Return ɣ

**BOYER-MOORE-MATCHER (T, P, ∑)**

1. n ←length [T]

2. m ←length [P]

3. λ← COMPUTE-LAST-OCCURRENCE-FUNCTION (P, m, ∑ )

4. ɣ← COMPUTE-GOOD-SUFFIX-FUNCTION (P, m)

5. s ←0

6. While s ≤ n - m

7. do j ← m

8. While j > 0 and P [j] = T [s + j]

9. do j ←j-1

10. If j = 0

11. then print "Pattern occurs at shift" s

12. s ← s + ɣ[0]

13. else s ← s + max (ɣ [j], j - λ[T[s+j]])

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. <https://www.javatpoint.com/daa-boyer-moore-algorithm>

**Lecture Video:**

1. https://youtu.be/4Xyhb72LCX4

**Online Notes:**

1. <http://vssut.ac.in/lecture_notes/lecture1428551222.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.

**In addition: PPT can be also be given.**