The problem

* The brute force string matching involves matching a pattern with a text.
* Given pattern P(1..n) and a text T(1…m) defined over the alphabet ∑, find one or more occurrences of P in T.
* It is primarily used for Web Searches, Data base Queries and Detecting Plagiarism.

We would start with the **pattern (P) of the string: ‘ANT’**

The **Text (T) of the string would be: ‘I AM AN ANTELOPE’**

I A M A N A N T E L O P E

Mismatch between I and A. so we would not compare the remaining characters in the pattern.

**A**

I A M A N A N T E L O P E

So we would restart the comparison from the beginning of the pattern. Initially the A from the pattern and the string would match each other. So we would then compare, the remaining characters in the pattern and the next character in the pattern would be **N**, so we would compare N from the pattern with M in the text. It would be an unsuccessful comparison and M and N would not match each other. **So we would shift the pattern by one position to the right.**

**A N**

I A M A N A N T E L O P E

A N

* So again as we shift the pattern by one position to the right, we would do a comparison again. A and M would not be a match.

I A M A N A N T E L O P E

A N T **Mismatch here between A and T**

Again we would shift the pattern the pattern by one position and finally. The right match would be when.

I A M A N A N T E L O P E

A N T

**MATCH for the Brute Force String Matching**

**Brute force string matching algorithm analysis:**

* **The worst case is when the pattern is not present in the given text and we would do all comparisons between the text and pattern at all of the positions until n-m positions.**
* **Say if we have a text of length n and a pattern of length of m, then the worst case is when we do all of the searches.**
* **Mismatch is for m-n times.**
* **Success is at (n-m) +1.**
* **So total comparisons would be ((n-m)+1)x m),**
* **Ignoring constants we would have a time complexity of ө(nm).**

**Bayer Moore Horse pool Algorithm Approach**

* **Here we would have to construct a Bad Match Table.**
* **We would compare the pattern to the text, starting from the rightmost character in the pattern.**
* **If a mismatch occurs, move the pattern forward corresponding to the value in the** TABLE. +
* **First we have to construct the BAD MATCH TABLE.**
* **To construct the BAD MATCH TABLE, we would initially assign a Value to each character in the text which would occur in the pattern and also for every other character in the text, according to the equation.**
* **Value = length – index – 1.**
* **Say if we are looking for the pattern (P) ‘TOOTH’ in the text ‘TRUST HARD TOOTH BRUSHES’.**
* **The BAD MATCH TABLE FOR the above TEXT is given as :**
* **The index positions would be like :**

**T O O T H**

***Index positions with a length 5***

**0**

**1**

**2**

**3**

**4**

* **Whenever a particular character occurs twice, the latest value is updated in the table.**

|  |  |
| --- | --- |
| **LETTER** | **\*\* VALUE (length – index – 1 )** |
| **T** | **5-0-1=4** |
| **O** | **5-1-1=3** |
| **O** | **5-2-1=2(KEPT VALUE)** |
| **T** | **1 (KEPT VALUE)** |
| **H** | **5 (KEPT VALUE)** |
| **\*(Any other character in the text T.)** | 1. **(KEPT VALUE)** |

**\*\* Value would correspond to number of shifts that needs to be made.**



* **Initially computing the value of T**
* **Index = 0 , Value = 5 – 0 – 1 = 4**
* **Computing the value of O**
* **Index = 1 , Value = 5 – 1 – 1 = 3**
* **Computing the value of O**
* **Index = 2, Value = 5 – 2 – 1 = 2 (This value would become the updated value.)**
* **Computing value for T again.**
* **Index = 3, Value = 5-3-1 = 1 (This is updated value.)**
* **The last character would have the value of 5 or the length of the pattern by default.**
* **ALL OTHER CHARACTERS WHICH WOULD OCCUR IN THE TEXT AND NOT IN THE PATTERN WOULD HAVE A VALUE OF 5 OR LENGTH OF THE PATTERN.**
* **THESE VALUES WOULD CORRESPOND TO THE NUMBER OF SHIFTS IN POSITION THAT NEEDS TO MAKE IN THE PATTERN WHILE COMPARING THE PATTERN AND THE TEXT.**

**Algorithm in action:**

**Initially as we do the comparison.**

**T O O T H**

Mismatch here.

Shift 5 positions starting from here.

**T O O T H**

**Mismatch between S and O in the pattern and string, comparison is made from right to left. So we would shift the positions according to the value corresponding to H in the table above which is 5 and we would shift 5 positions.**

**T O O T H**

* **In the value table, the value for T would be 1, so the pattern would be shifted by ONE POSITION.**

* ****

****

**H does not match with O in the text, so we would shift 2 positions corresponding to the value as according to the text.**

****

**Mismatch between T and H.**

**So we would shift by 1 value.**

**Corresponding to the value in the table. (Corresponding to T.)**

(MISMATCH HERE).

T O O T H

**Match is found between the pattern and the String.**

**T O O T H**

T R U S T H A R D T O O T H B R U S H E S