Q. Use the horsepool’s algorithm to search **for the pattern GORE in the string ALGORITHM.**

Solution :

Initially we would create the bad value table.

Comparison is made from the right to left.

The value is given to be as :

Value = length – index – 1

Whenever a value reoccurs in the pattern, its value is updated in the bad value table.

The bad value table for GORE is given as :

G O R E

**0 1 2 3 -> index positions**

**Length = 4**

|  |  |
| --- | --- |
| **Letter** | **Value = Length – index -1** |
| **G** | **4-0-1 = 3** |
| **O** | **4-1-1 = 2** |
| **R** | **4-2-1 = 1** |
| **E** | **4** |
| **\*** | **4** |
|  |  |

**The last character E would take the number of shifts corresponding to the length of the pattern which is 4 and all of the other characters which is not same as the pattern in the text would take the number of shifts corresponding to the length of the pattern which is 4.**

**Algorithm in Action:**

**A L G O R I T H M**

**G O R E**

**Mismatch between E and O. So the value corresponding to O is 2 and then 2 shifts would be made from the position of E in the pattern so that E will be under I.**

**A L G O R I T H M**

**G O R E**

**Mismatch here between E and I. The value corresponding to I is 4 and thus 4 shifts would be made as I is any other character which is not part of the pattern. So shifts would be made from the position of E in the pattern to the right. This would take beyond the end of the string so the algorithm would halt.**

**A L G O R I T H M**

**G O R E**

Algorithm halts as it is taken out of the String.

Q2. How many character comparisons does it take the Horspool’s algorithm to decide that CAB is not found in ABRACADABRA? How many to find that DRAC is not there?

Answer:

C A B

0 1 2 <- index positions.

Length = 3

|  |  |
| --- | --- |
| Letter | Value = Length – index – 1 |
| C | 3-0-1 = 2 |
| A | 3-1-1 = 1 |
| B | 3 (The last letter and hence the patterns length ) |
| \* | 3 |

A B R A C A D A B R A

C A B (mismatch here, 3 positions will have to be shifted.)

A B R A C A D A B R A

C A B (mismatch here, 1 position would be shifted corresponding to value of A in the text. )

A B R A C A D A B R A

C A B (mismatch here between D and B, 3 positions will be shifted corresponding to the value of 3.)

A B R A C A D A B R A

C A B (mismatch here between R and B, shifted 3 positions corresponding to the value of R in the text.)

A B R A C A D A B R A

C A B

No more comparisons would be made as the pattern would go out of the text, and the algorithm would halt.

So 4 comparisons are made before the algorithm came to a halt.

The number of comparisons made to find out DRAC is not there

D R A C

0 1 2 3 <- index

Length = 4

Value = Length – index – 1

|  |  |
| --- | --- |
| Letter | Value |
| D | 4-0-1 = 3 |
| R | 4-1-1 = 2 |
| A | 4 – 2 – 1 = 1 |
| C | 4 (Last position will have shift equal to the length of the pattern ) |
| \* | 4 |

Algorithm in action:

A B R A C A D A B R A

D R A C (mismatch here, so shift by 1 position corresponding to the value of A. )

A B R A C A D A B R A

D R A C (mismatch here between D and B so shift value corresponding to B’s value which is 4 (Any other character. Shift from C’s position always))

**3 successful comparisons are made**

A B R A C A D A B R A

D R A C

One unsuccessful comparison between B and C and this would shift the pattern by 4 more positions to the right which would reach the end of the text.

As a result of this, no more comparisons will be made.

So a total of 6 comparisons are done (Including both successful and unsuccessful).