Boyer-Moore Algorithm

STRING MATCHING ALGORITHM

Boyer-Moore Algorithm

- It was developed by **Robert S. Boyern** and **J Strother Moore** in 1977.
- The Boyer-Moore algorithm is consider the most efficient string-matching algorithm.
    - **For Example:** text editors and commands substitutions.

- **WHY WE USE THIS ALGORITHM??**

    - Problem for Brute Force Search:
        - We keep considering too many comparisons, So the time complexity increase O(*mn*).That's why Boyer-Moore Algorithm. It works the fastest when the alphabet is moderately sized and the pattern is relatively long.

Boyer-Moore Algorithm

- The algorithm scans the characters of the pattern from **right to left** **i.e beginning with the rightmost character.**
- If the text symbol that is compared with the rightmost pattern symbol does not occur in the pattern at all, then the pattern can be shifted by m positions behind this text symbol.
- **Example:**

“Hello to the world.” is a string and if we want to search “world” for that string that’s a Pattern.

Boyer-Moore Algorithm

- Boyer Moore is a combination of following two approaches.
  1) Bad Character Approach
  2) Good Suffix Approach
- Both of the above Approach can also be used independently to search a pattern in a text.
- But here we will discusses about Bad-Match Approach.
- Boyer Moore algorithm does preprocessing.
- **Why Preprocessing??**

  To shift the pattern by more than one character.

Bad Character Approach

- The character of the text which doesn't match with the current character of pattern is called the **Bad Character**.
- Upon mismatch we shift the pattern until –
  1) The mismatch become a match.
     - If the mismatch occur then we see the Bad-Match table for shifting the pattern.
  2) Pattern P move past the mismatch character.
     - If the mismatch occur and the mismatch character not available in the Bad-Match Table then we shift the whole pattern accordingly.

Good Suffix Approach

- Just like bad character heuristic, a preprocessing table is generated for good suffix Approach.
- Let **t** be substring of text **T** which is matched with substring of pattern **P**.
  Now we shift pattern until :
  1) Another occurrence of t in P matched with t in T.
  2) A prefix of P, which matches with suffix of t
  3) P moves past t.

Boyer-Moore Algorithm

Here we use Bad-Match Approach for Searching

Boyer-Moore Algorithm

Steps to find the pattern :

Step 1: Construct the bad-symbol shift table.

Step 2: Align the pattern against the beginning of the text.

Step 3: Repeat the following step until either a matching substring is found or the pattern reaches beyond the last character of the text.

1.Construct Bad Match Table:

Formula for constructing Bad Match Table:

- **Formula:**

  Values =Length of pattern-index-1

Construct Bad Match Table:

Example:

Text: ” WELCOMETOTEAMMAST”

Pattern: ’TEAMMAST’

Pattern	T E A M M A S T	Length =8
Index #	0 1 2 3 4 5 6 7	

Letter	T	E	A	M	S	*
Values						

Bad Match Table

Construct Bad Match Table:

Pattern T E A M M A S T

Index # 0 1 2 3 4 5 6 7

Length =8

Letter	T	E	A	M	S	*
Values	7					

Values =Max(1,Length of string-index-1)

T = max(1,8-0-1)=7

Construct Bad Match Table:

Pattern T E A M M A S T

Index # 0 1 2 3 4 5 6 7

Length =8

Letter	T	E	A	M	S	*
Values	7	6				

Values =max(1,Length of string-index-1)

E = max(1,8-1-1)=6

Construct Bad Match Table:

Pattern T E A M M A S T

Index # 0 1 2 3 4 5 6 7

Length =8

Letter	T	E	A	M	S	*
Values	7	6	5			

Values =max(1,Length of string-index-1)

A = max(1,8-2-1)=5

Construct Bad Match Table:

Pattern T E A M M A S T

Index # 0 1 2 3 4 5 6 7

Length =8

Letter	T	E	A	M	S	*
Values	7	6	5	4		

Values =max(1,Length of string-index-1)

M = max(1,8-3-1)=4

Construct Bad Match Table:

Pattern T E A M M A S T

Index # 0 1 2 3 4 5 6 7

Length =8

Letter	T	E	A	M	S	*
Values	7	6	5	3		

Values =max(1,Length of string-index-1)

M = max(1,8-4-1)=3

Construct Bad Match Table:

Pattern T E A M M A S T

Index # 0 1 2 3 4 5 6 7

Length =8

Letter	T	E	A	M	S	*
Values	7	6	2	3		

Values =max(1,Length of string-index-1)

A = max(1,8-5-1)=2

Construct Bad Match Table:

Pattern T E A M M A S T

Index # 0 1 2 3 4 5 6 7

Length =8

Letter	T	E	A	M	S	*
Values	7	6	2	3	1	

Values =Length of string-index-1

S = max(1,8-6-1)=1

Construct Bad Match Table:

Pattern T E A M M A S T

Index # 0 1 2 3 4 5 6 7

Letter	T	E	A	M	S	*
Values	1	6	2	3	1	

Values =max(1,Length of string-index-1)

T = max(1,8-7-1)=1

ShiftTable Algorithm:

```
public void shiftTable(){
int lengthofpattern=this.pattern.length();
for(int index=0;index<lengthofpattern;index++)
      {
       char actualCharacter=this.Pattern.charAt(index);
       int maxshift=Max.max(1,lengthOfPattern-index-1);
       this.mismatchShiftstable.put(actualCharater,maxshift);
       }

}
```

Construct Bad Match Table:

Pattern T E A M M A S T

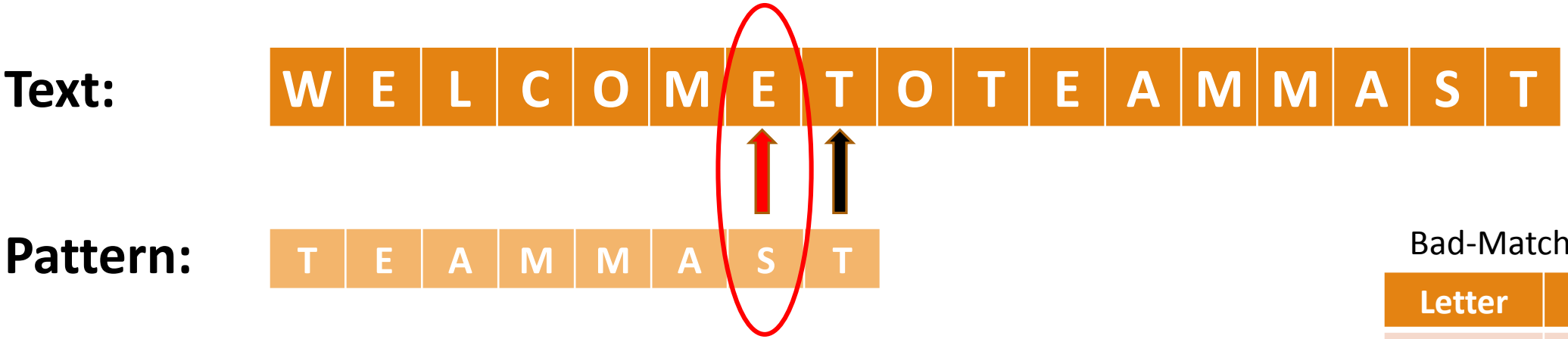
Index # 0 1 2 3 4 5 6 7

Length =8

Letter	T	E	A	M	S	*
Values	8	6	2	3	1	8

Any other letter is presented by ‘*’ is taken equal value to length of string i.e 8 here.

2. Align the pattern



Bad-Match Table

Letter	Values
T	8
E	6
A	2
M	3
S	1
*	8

Move 6 spaces toward right

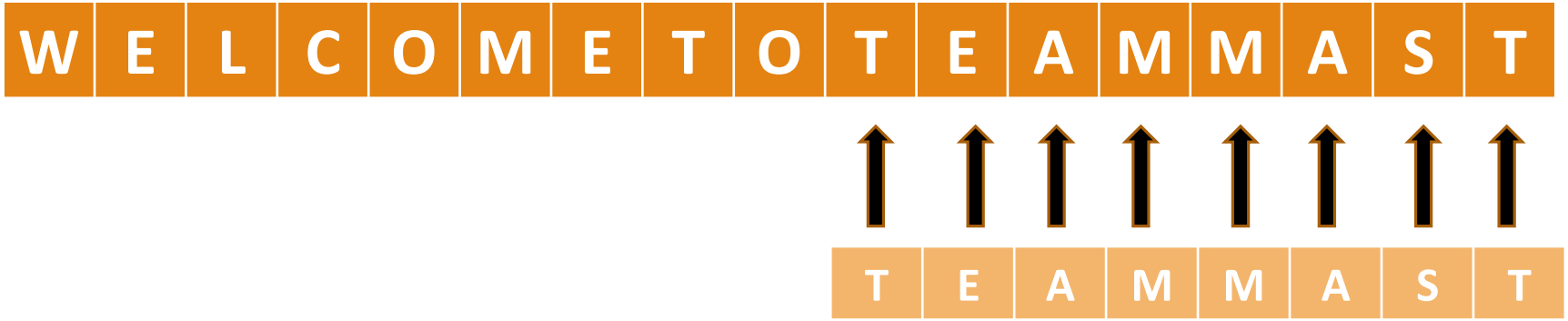
Matching……..

Now move 3 spaces toward right.

Bad-Match Table

Letter	Values
T	8
E	6
A	2
M	3
S	1
*	8

Matching……..



Hence the pattern match……

Bad-Match Table

Letter	Values
T	8
E	6
A	2
M	3
S	1
*	8

Boyer-Moore Algorithm

```
for(int i=0;i=<lengthofText-lengthOfPattern;i+=numOfSkips)
     {
      numOfSkips=0;

      for(int j=lengthOfPattern-1;j>=0;j--){
            if(pattern.charAt(j)!=text.charAt(i+j)){
                  if(this.mismatchShiftTable.get(text.charAt(i+j))!=NULL)
                {
                numOfSkips=this.mismatchShiftTable.get(text.charAt(i+j));
                break;
                }
else{
                 numOfSkips=lengthOfPattern;
                 break;
               }  }  }
      if(numOfSkips==0)
      return i;
      }
```

Time Complexity

- The preprocessing phase in O(m+Σ) time and space complexity and searching phase in ***O***(mn) time complexity.

- It was proved that this algorithm has $O(m)$ comparisons when P is not in T. However, this algorithm has $O(mn)$ comparisons when P is in T.

THANK YOU