



**Mansoura University**  
**Faculty of Computers and Information**  
**Department of Computer Science**  
**First Semester: 2020-2021**



**[MED121] Bioinformatics: Boyer Moore Algorithm**  
**Grade: Third Year (Medical Informatics Program)**

**Sara El-Metwally, Ph.D.**  
**Faculty of Computers and Information,**  
**Mansoura University,**  
**Egypt.**

# AGENDA

- Boyer Moore Algorithm (Bad Character Rule)
- Boyer Moore Algorithm (Good Suffix Rule)
- Boyer Moore Algorithm (Strong Good Suffix Rule)
- Boyer Moore Algorithm (All Together)
- Boyer Moore Algorithm (Algorithm Code and Trace)

# BOYER MOORE ALGORITHM

- Boyer Moore is a combination of two approaches:
  - Bad Character Rule
  - Good Suffix Rule
- Naïve algorithm slides the pattern **P** over the text **T** one by one.
- Boyer Moore does preprocessing over the pattern so that the pattern can be shifted by more than one.

# BOYER MOORE ALGORITHM (BAD CHARACTER RULE)

- The character of the text which doesn't match with the current character of pattern is **called the Bad Character**.

[illegible]

# BOYER MOORE ALGORITHM (BAD CHARACTER RULE)

- The character of the text which doesn't match with the current character of pattern is **called the Bad Character**.

[illegible]

# BOYER MOORE ALGORITHM (BAD CHARACTER RULE)

- The character of the text which doesn't match with the current character of pattern is **called the Bad Character**.

[illegible]

# BOYER MOORE ALGORITHM (BAD CHARACTER RULE)

- The character of the text which doesn't match with the current character of pattern is **called the Bad Character**.

[illegible]

# BOYER MOORE ALGORITHM (BAD CHARACTER RULE)

- Upon mismatch we shift the pattern until:
  - The mismatch become a match.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C
T	A	T	G	T	G										

- ✓ The pattern is shifted by **2** positions (n).
- ✓ We skip **1** alignment (n-1).





# BOYER MOORE ALGORITHM (BAD CHARACTER RULE)

- Upon mismatch we shift the pattern until:
  - Pattern **P** moves past the mismatched character.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C
		T	A	T	G	T	G									

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C
								T	A	T	G	T	G			

- ✓ The pattern is shifted by **6** positions (n).
- ✓ We skip **5** alignment (n-1).

# BOYER MOORE ALGORITHM (GOOD SUFFIX RULE)

- Let  $t$  be substring of text  $T$  which is matched with substring of pattern  $P$ .

i	0	1	2	3	4	5	6	7	8	9	10
T	A	B	A	A	B	A	B	A	C	B	A
P	C	A	B	A	B						

# BOYER MOORE ALGORITHM (GOOD SUFFIX RULE)

- Now we shift pattern until :
  - Another occurrence of **t** in **P** matched with **t** in **T**.

<b>i</b>	0	1	2	3	4	5	6	7	8	9	10
<b>T</b>	A	B	A	<b>A</b>	<b>B</b>	A	B	A	C	B	A
<b>P</b>	C	A	B	A	B						

<b>i</b>	0	1	2	3	4	5	6	7	8	9	10
<b>T</b>	A	B	A	A	B	A	B	A	C	B	A
<b>P</b>			C	A	B	A	B				

- ✓ The pattern is shifted by **2** positions (n).
- ✓ We skip **1** alignment (n-1).

# BOYER MOORE ALGORITHM (GOOD SUFFIX RULE)

- Now we shift pattern until :
  - A **prefix** of **P**, which matches with **suffix of t** in **T**

i	0	1	2	3	4	5	6	7	8	9	10
T	A	A	B	A	B	A	B	A	C	B	A
P	A	B	B	A	B						

i	0	1	2	3	4	5	6	7	8	9	10
T	A	A	B	A	B	A	B	A	C	B	A
P				A	B	B	A	B			

- ✓ The pattern is shifted by **3** positions (n).
- ✓ We skip **2** alignment (n-1).

# BOYER MOORE ALGORITHM (GOOD SUFFIX RULE)

- Now we shift pattern until :

- P moves past t

i	0	1	2	3	4	5	6	7	8	9	10
T	A	A	C	A	B	A	B	A	C	B	A
P	C	B	A	A	B						

i	0	1	2	3	4	5	6	7	8	9	10
T	A	B	A	A	B	A	B	A	C	B	A
P						C	B	A	A	B	

- ✓ The pattern is shifted by 5 positions (n).
- ✓ We skip 4 alignment (n-1).

# BOYER MOORE ALGORITHM (STRONG GOOD SUFFIX RULE)

- Suppose substring  $q = P[i \text{ to } n]$  got matched with  $t$  in  $T$  and  $c = P[i-1]$  is the mismatching character.
- Now unlike case 1 we will search for  $t$  in  $P$  which is not preceded by character  $c$ . The closest such occurrence is then aligned with  $t$  in  $T$  by shifting pattern  $P$ .

$i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$T$	A	A	B	A	B	A	B	A	C	B	A	C	A	B	B	C	A	B
$P$	A	A	C	C	A	C	C	A	C									

Diagram illustrating the Boyer Moore algorithm (Strong Good Suffix Rule) with a mismatch at index 6.

The text  $T$  (Target) is: A A B A B A B A C B A C A B B C A B. The text  $P$  (Pattern) is: A A C C A C C A C.

The mismatching character  $c$  is at index 6 of  $P$  (character 'C'). The substring  $q$  of  $P$  (characters 'A C') is matched with the substring of  $T$  (characters 'A C') starting at index 7. The character  $t$  is the character 'A' at index 7 of  $T$ .

# BOYER MOORE ALGORITHM (STRONG GOOD SUFFIX RULE)

- Now unlike case 1 we will search for **t** in **P** which is not preceded by character **c**. The closest such occurrence is then aligned with **t** in **T** by shifting pattern **P**.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T	A	A	B	A	B	A	B	A	C	B	A	C	A	B	B	C	A	B
P	A	A	C	C	A	C	C	A	C									

Diagram illustrating the Boyer Moore algorithm (Strong Good Suffix Rule) for finding the pattern **P** in the text **T**.

The text **T** is: A A B A B A B A C B A C A B B C A B. The pattern **P** is: A A C C A C C A C.

The initial alignment (top) shows the pattern **P** starting at index 0. The character **c** (red) at index 6 of **P** is compared with the character **B** (red) at index 6 of **T**. The character **t** (green) at index 7 of **P** is compared with the character **A** (green) at index 7 of **T**. The character **q** (green) at index 8 of **P** is compared with the character **C** (green) at index 8 of **T**. The mismatch at index 6 indicates that the pattern **P** is not aligned correctly.

The shifted alignment (bottom) shows the pattern **P** shifted by 6 positions (n). The character **c** (red) at index 6 of **P** is compared with the character **A** (green) at index 6 of **T**. The character **t** (green) at index 7 of **P** is compared with the character **C** (green) at index 7 of **T**. The character **q** (green) at index 8 of **P** is compared with the character **B** (green) at index 8 of **T**. The mismatch at index 6 indicates that the pattern **P** is not aligned correctly.

- ✓ The pattern is shifted by **6** positions (n).
- ✓ We skip **5** alignment (n-1).



# BOYER MOORE ALGORITHM (ALL TOGETHER)

**Text= GTTATAGCTGATCGCGGGCGTAGCGGGCGAA**

**Pattern= GTAGCGGGCG**

**GTTATAGCTGATCGCGGGCGTAGCGGGCGAA**  
**GTAGCGGGCG**

## BOYER MOORE ALGORITHM (ALL TOGETHER)

GTTATAGCT**T**GATCGCGGGCGTAGCGGGCGAA  
GTAGCGGGC**G**

← No matching chars, no good suffix rule is used, bad  
char rule = **7positions**

GTTATAGCT**T**GATCGCGGGCGTAGCGGGCGAA  
GT**T**AGCGGGCG

## BOYER MOORE ALGORITHM (ALL TOGETHER)

GTTATAGCTGATCGCGGCGTAGCGGCGAA  
GTAGCGGCG

GTTATAGCTGATCGC**G**GCGTAGCGGCGAA  
GTAGCGGCG**G**

GTTATAGCTGATC**GC**GCGTAGCGGCGAA  
GTAGCG**GCG**

# BOYER MOORE ALGORITHM (ALL TOGETHER)

GTTATAGCTGATCGCGGCGTAGCGGCGAA  
GTAGCGGCG

GTTATAGCTGATCGCGGCGTAGCGGCGAA  
GTAGCGGCG

Bad character rule=1

Good suffix rule=3

GTTATAGCTGATCGCGGCGTAGCGGCGAA  
GTAGCGGCG

## BOYER MOORE ALGORITHM (ALL TOGETHER)

GTTATAGCTGATCGCGGCGTAGCGGGCGAA  
GTAGCGGCG

GTTATAGCTGATCGCGGCGTAGCGGGCGAA  
GTAGCGGCG

GTTATAGCTGATCGCGGCGTAGCGGGCGAA  
GTAGCGGCG

## BOYER MOORE ALGORITHM (ALL TOGETHER)

GTTATAGCTGATCGCGGGCGTAGCGGGCGAA  
GTAGCGGGCG

GTTATAGCTGATCGCGGGCGTAGCGGGCGAA  
GTAGCGGGCG

GTTATAGCTGATCGCGGGCGTAGCGGGCGAA  
GTAGCGGGCG

# BOYER MOORE ALGORITHM (ALL TOGETHER)

GTTATAGCTGATCGCGGCGTAGCGGCGAA  
GTAGCGGCG

Bad character rule=3  
Good suffix rule=8

GTTATAGCTGATCGCGGCGTAGCGGCGAA  
GTAGCGGCG

GTTATAGCTGATCGCGGCGTAGCGGCGAA  
GTAGCGGCG

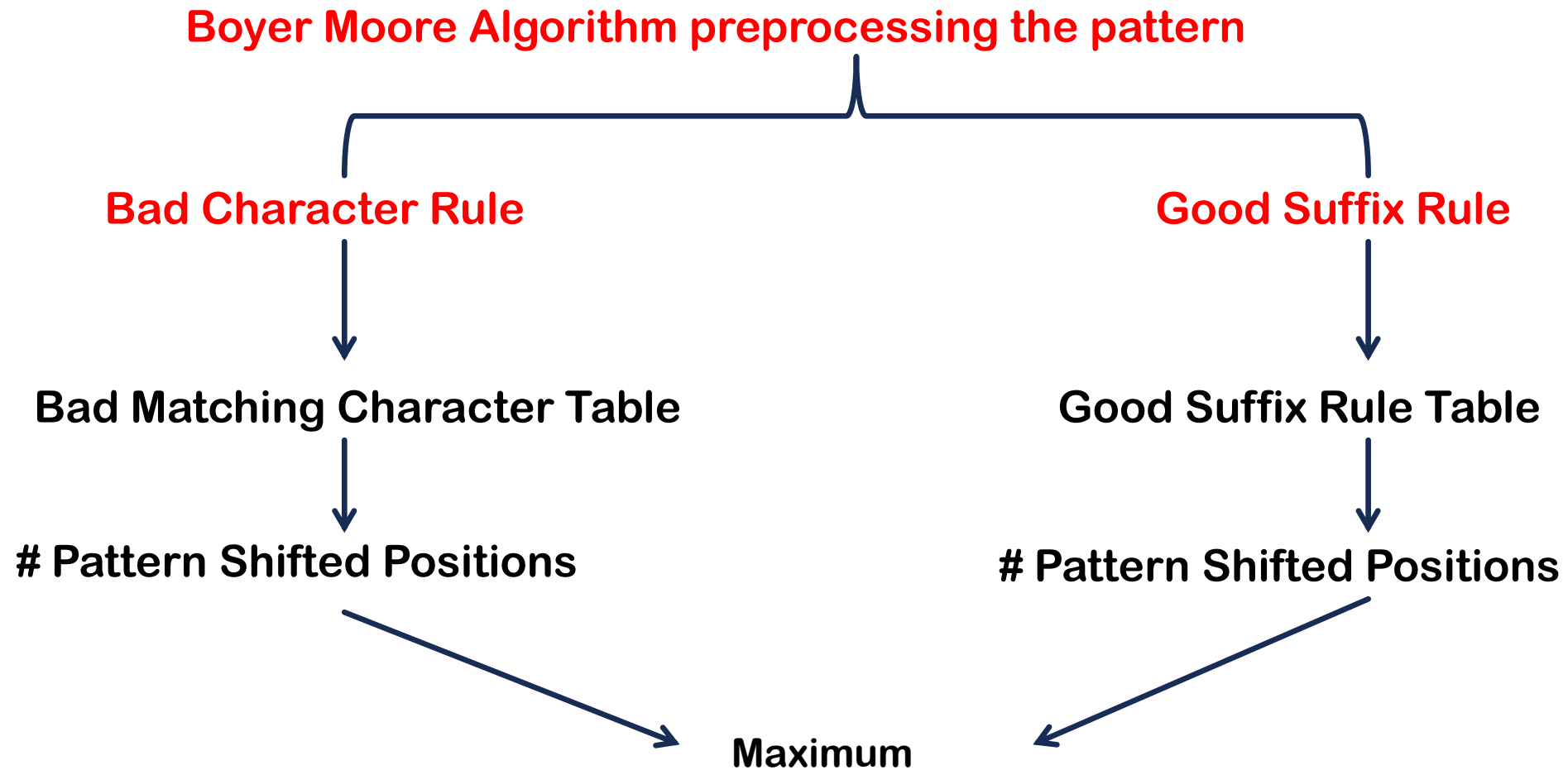
## BOYER MOORE ALGORITHM (ALL TOGETHER)

GTTATAGCTGATCGCGGGCGTAGCGGGCGAA  
GTAGCGGGCG

GTTATAGCTGATCGCGGGCGTAGCGGGCGAA  
GTAGCGGGCG



# BOYER MOORE ALGORITHM



# BOYER MOORE ALGORITHM

- How to Construct Bad Match Table?
  - The table columns corresponding to  $\Sigma$  in the pattern.
  - Value = last occurrence of this character in the pattern.
- Example 1: for pattern **ACGGA** construct the bad matching character table?

✓Determine  $\Sigma$

$$\Sigma \{A, C, G\}$$

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

0 1 2 3 4  
ACGGA

AACCGACGGAATGTTAACGGA

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**

**n = 5**

**m = 20**

s	j	s+j	P[j]	T[s+j]	match	new s
0						

```
def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            print("Pattern occur at index {}".format(s))
            s += (n - badChar[ord(txt[s+n])]) if s+n < m else 1
            print(s)
        else:
            s += max(1, j - badChar[ord(txt[s+j])])
            print(s)
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**  
 $j$

$n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
0	4	4	A	G	N	badChar [ txt[4] ] = badChar[G]=3  s = s + max (1, j -3) s = s + max (1, 4-3) s= 0+ 1=1

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            print("Pattern occur at index {}".format(s))
            s += (n-badChar[ord(txt[s+n])] if s+n < m else 1)
            print(s)
        else:
            s += max(1, j-badChar[ord(txt[s+j])])
            print(s)
    
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**  
 $j$

$n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
1	4	5	A	A	Y	

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while (s <= m - n):
        j = n - 1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            print("Pattern occur at index {}".format(s))
            s += (n - badChar[ord(txt[s+n])]) if s+n < m else 1
            print(s)
        else:
            s += max(1, j - badChar[ord(txt[s+j])])
            print(s)
    
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**  
 $j$

$n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
1	4	5	A	G	Y	
1	3	4	G	G	Y	

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            print("Pattern occur at index ".format(s))
            s += (n - badChar[ord(txt[s+n])] if s+n < m else 1)
            print(s)
        else:
            s += max(1, j - badChar[ord(txt[s+j])])
            print(s)
    
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**  
 $j$

$n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
1	4	5	A	G	Y	
1	3	4	G	G	Y	
1	2	3	G	C	N	badChar [ txt[3] ] = badChar[C]=1  s = s + max (1, j -1) s = s + max (1, 2-1) s= 1+ 1=2

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            print("Pattern occur at index {}".format(s))
            s += (n - badChar[ord(txt[s+n])]) if s+n < m else 1
            print(s)
        else:
            s += max(1, j - badChar[ord(txt[s+j])])
            print(s)
    
```



# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**  
 $j$

$n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
2	4	6	A	C	N	badChar [ txt[6]] = badChar[C]=1  $s = s + \max(1, j - 1)$ $s = s + \max(1, 4 - 1)$ $s = 2 + 3 = 5$

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m - n):
        j = n - 1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            print("Pattern occur at index {}".format(s))
            s += (n - badChar[ord(txt[s+n])]) if s+n < m else 1
            print(s)
        else:
            s += max(1, j - badChar[ord(txt[s+j])])
            print(s)
    
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**  
 $j$

$n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
5	4	9	A	A	Y	

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1

        if j < 0:
            print("Pattern occur at index ".format(s))
            s += (n-badChar[ord(txt[s+n])] if s+n < m else 1)
            print(s)
        else:
            s += max(1, j-badChar[ord(txt[s+j])])
            print(s)
    
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**  
 $j$

$n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
5	4	9	A	A	Y	
5	3	8	G	G	Y	
5	2	7	G	G	Y	
5	1	6	C	C	Y	
5	0	5	A	A	Y	
5	-1	-	-	-	-	Pattern at index s=5 $s+n = 5+5 = 10 < 20$ $\text{badChar}[T[s+n]] =$ $\text{badChar}[A] = 4$ $s = s + (n-4)$ $s = 5 + (5-4) = 6$

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while(j >= 0 and pat[j] == txt[s+j]):
            j -= 1
        if j < 0:
            print("Pattern occur at index ".format(s))
            s += (n - badChar[ord(txt[s+n])]) if s+n < m else 1
            print(s)
        else:
            s += max(1, j - badChar[ord(txt[s+j])])
            print(s)
    
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**

$j$

$n = 5$

$m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
6	4	10	A	A	Y	
6	3	9	G	A	N	badChar [ txt[9] ] = badChar[A]=4 s = s + max (1, j -4) s = s + max (1, 3-4) s= 6+ 1=7

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            print("Pattern occur at index ".format(s))
            s += (n-badChar[ord(txt[s+n])]) if s+n < m else 1
            print(s)
        else:
            s += max(1, j-badChar[ord(txt[s+j])])
            print(s)
    
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

badCharTable

$s+j$   
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
**AACCGACGGAATGTTACGGA**  
 0 1 2 3 4  
**ACGGA**

$j$        $n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
7	4	11	A	T	N	badChar [ txt[11]] = badChar[T]= -1 $s = s + \max(1, j - (-1))$ $s = s + \max(1, 4 + 1)$ $s = 7 + 5 = 12$

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            print("Pattern occur at index ".format(s))
            s += (n - badChar[ord(txt[s+n])]) if s+n < m else 1
            print(s)
        else:
            s += max(1, j - badChar[ord(txt[s+j])])
            print(s)
    
```

# BOYER MOORE ALGORITHM

A	C	G
4	1	3

## badCharTable

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

AACCGACGGAATGTTACGGA

0 1 2 3 4  
ACGGA

$n = 5$   
 $m = 20$

s	j	s+j	P[j]	T[s+j]	match	new s
12	4	16	A	C	N	badChar [ txt[16]] = badChar[C]= 1  s = s + max (1, j -1)) s = s + max (1, 4-1) s= 12+ 3=15

```
def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j>=0 and pat[j] == txt[s+j]:
            j -= 1

        if j<0:
            print("Pattern occur at index ".format(s))
            s += (n-badChar[ord(txt[s+n])] if s+n<m else 1)
            print(s)
        else:
            s += max(1, j-badChar[ord(txt[s+j])])
            print(s)
```

# BOYER MOORE ALGORITHM

A	C	G	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
4	1	3	AACCGACGGAATGTTACGGA																			
badCharTable			0 1 2 3 4				ACGGA															

$n = 5$   
 $m = 20$   
 $j$

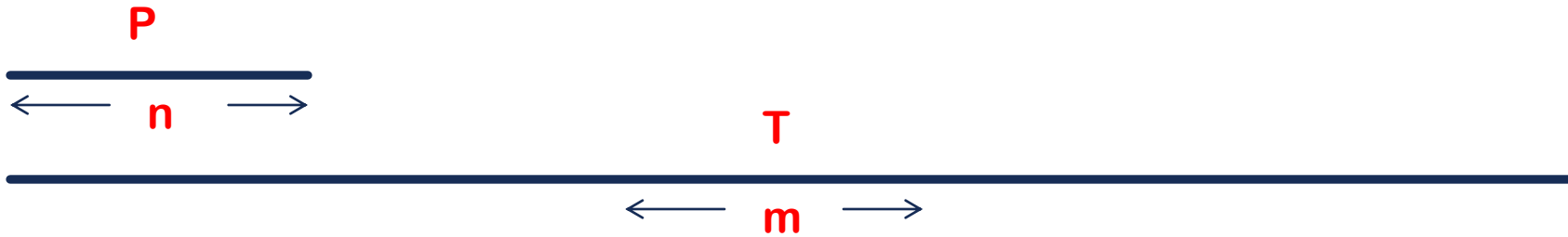
s	j	s+j	P[j]	T[s+j]	match	new s
15	4	19	A	A	Y	Pattern at index s=15 $s+n = 15+5 = 20 < 20$ $s=15+(1)=16$
						$16 \leq 15$

```

def search(txt, pat):
    n = len(pat)
    m = len(txt)
    badChar = badCharHeuristic(pat, n)
    s = 0
    while(s <= m-n):
        j = n-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1

        if j < 0:
            print("Pattern occur at index ".format(s))
            s += (n-badChar[ord(txt[s+n])] if s+n < m else 1)
            print(s)
        else:
            s += max(1, j-badChar[ord(txt[s+j])])
            print(s)
    
```

Q



- What is the worst case running time? Give me example?
- What is the best case running time? Give me example?



# BOYER MOORE ALGORITHM ANALYSIS

- What is the worst case running time? Give me example?

$1^m$  Input text, length  $m$   
 $01111$  Pattern, length  $n$

$O(mn)$

- What is the best case running time? Give me example?

$1^m$  Input text, length  $m$   
 $0^n$  Pattern, length  $n$

$O\left(\frac{m}{n}\right)$



**Thank you!**