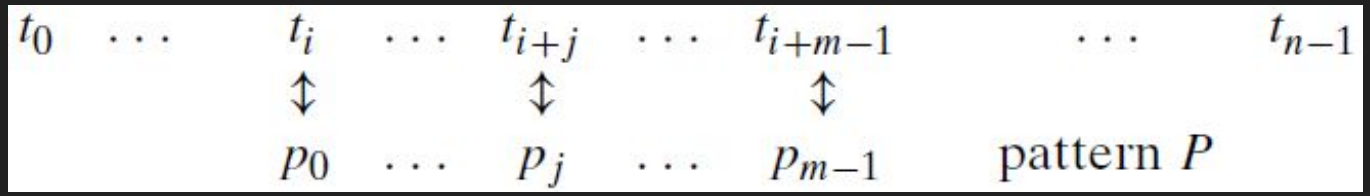
Design and Analysis of Algorithms

String Matching

Mr. Channa Bankapur

Input Enhancement in String Matching:

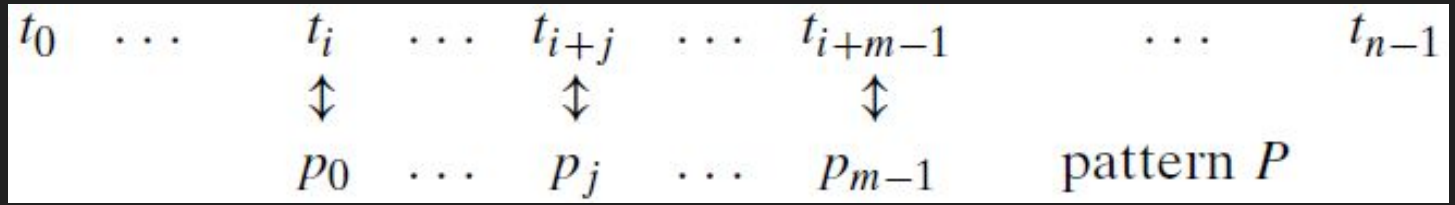


Input Enhancement to improve the average-case/amortized efficiency.

1. Horspool's algorithm
2. Boyer-Moore algorithm
3. Robin-Karp algorithm
4. Finite State Automaton
5. Knuth-Morris-Pratt algorithm

Consider the problem of searching for genes in DNA sequences.
A DNA sequence is represented by a text of the alphabet {A, C, G, T} and the gene segment is the pattern.

Eg: Text with n = 18 and pattern with m = 5.
DNA Sequence: **A C G T T A G C A G C G C A G C G C**
Gene segment: **A G C G C**



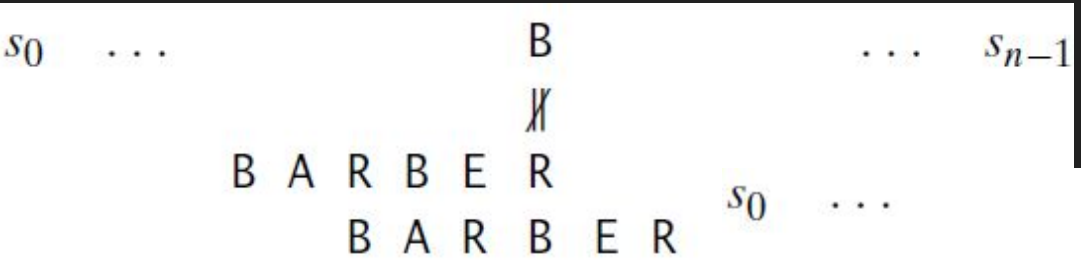
ALGORITHM *BruteForceStringMatch*$(T[0..n-1], P[0..m-1])$

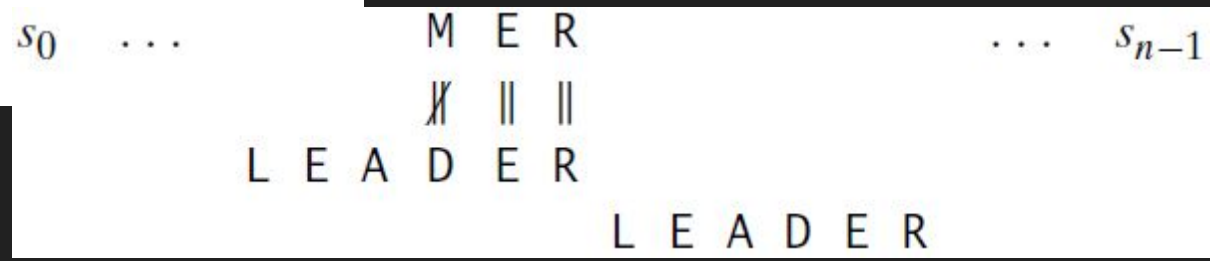
```
//Implements brute-force string matching
//Input: An array T[0..n − 1] of n characters representing a text and
//       an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//        matching substring or −1 if the search is unsuccessful
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

```
NaiveStringMatch(T[0..n-1], P[0..m-1])
    for i ← 0 to n-m
        j ← 0
        while (j < m and T[i+j] = P[j])
            j ← j + 1
        if(j = m) return i
    return -1
-------------------------------------
NaiveStringMatch2(T[0..n-1], P[0..m-1])
    for i ← m-1 to n-1
        j ← 0
        while (j < m and T[i-j] = P[m-1-j])
            j ← j + 1
        if(j = m) return i-(m-1)
    return -1
```

```
NaiveStringMatch2(T[0..n-1], P[0..m-1])
    for i ← m-1 to n-1
        j ← 0
        while (j < m and T[i-j] = P[m-1-j])
            j ← j + 1
        if(j = m) return i-(m-1)
    return -1
-----------------------------------
NaiveStringMatch3(T[0..n-1], P[0..m-1])
    i ← m-1
    while (i < n)
        j ← 0
        while (j < m and T[i-j] = P[m-1-j])
            j ← j + 1
        if(j = m) return i-(m-1)
        i ← i+1
    return -1
```

Horspool's Algorithm





Eg: Pattern **AGCGC**

Shift table:

A	C	G	T
4	2	1	5

Eg: Gene segment in DNA Sequence using Horspool's algo.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
A C G T T A G C A G C G C A G C G C
A G C G C
          A G C G C
            A G C G C
              A G C G C
```

Tbl[T] = 5

Tbl[G] = 1

Tbl[C] = 2

Horspool's Algorithm

$$t(c) = \begin{cases} \text{the pattern's length } m, \\ \text{if } c \text{ is not among the first } m-1 \text{ characters of the pattern;} \\ \\ \text{the distance from the rightmost } c \text{ among the first } m-1 \text{ characters} \\ \text{of the pattern to its last character, otherwise.} \end{cases}$$

ALGORITHM *ShiftTable*$(P[0..m-1])$

//Fills the shift table used by Horspool's and Boyer-Moore algorithms
//Input: Pattern $P[0..m-1]$ and an alphabet of possible characters
//Output: $Table[0..size-1]$ indexed by the alphabet's characters and
// filled with shift sizes computed by formula (7.1)
for $i \leftarrow 0$ **to** $size-1$ **do** $Table[i] \leftarrow m$
for $j \leftarrow 0$ **to** $m-2$ **do** $Table[P[j]] \leftarrow m-1-j$
return *Table*

```
NaiveStringMatch3(T[0..n-1], P[0..m-1])
    i ← m-1
    while (i < n)
        j ← 0
        while (j < m and T[i-j] = P[m-1-j])
            j ← j + 1
        if(j = m) return i-(m-1)
        i ← i+1
    return -1
```

```
HorspoolMatching(T[0..n-1], P[0..m-1])
    STbl[alphabet size] ← ShiftTbl(P[0..m-1])
    i ← m-1
    while (i < n)
        j ← 0
        while (j < m and T[i-j] = P[m-1-j])
            j ← j + 1
        if(j = m) return i-(m-1)
        i ← i + STbl[ T[i] ]
    return -1
```

ALGORITHM *HorspoolMatching*$(P[0..m-1], T[0..n-1])$

```
//Implements Horspool's algorithm for string matching
//Input: Pattern P[0..m − 1] and text T[0..n − 1]
//Output: The index of the left end of the first matching substring
//         or −1 if there are no matches
ShiftTable(P[0..m − 1])          //generate Table of shifts
i ← m − 1                        //position of the pattern's right end
while i ≤ n − 1 do
    k ← 0                        //number of matched characters
    while k ≤ m − 1 and P[m − 1 − k] = T[i − k] do
        k ← k + 1
    if k = m
        return i − m + 1
    else i ← i + Table[T[i]]
return −1
```

character c	A	B	C	D	E	F	. . .	R	. . .	Z	_
shift $t(c)$	4	2	6	6	1	6	6	3	6	6	6

```
J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
B A R B E R             B A R B E R
        B A R B E R             B A R B E R
            B A R B E R                 B A R B E R
```

Consider the problem of searching for genes in DNA sequences using Horspool's algorithm.
A DNA sequence is represented by a text on the alphabet {A, C, G, T}, and the gene or gene segment is the pattern.

Construct the shift table for the following gene segment of your chromosome 10: TCCTATTCTT

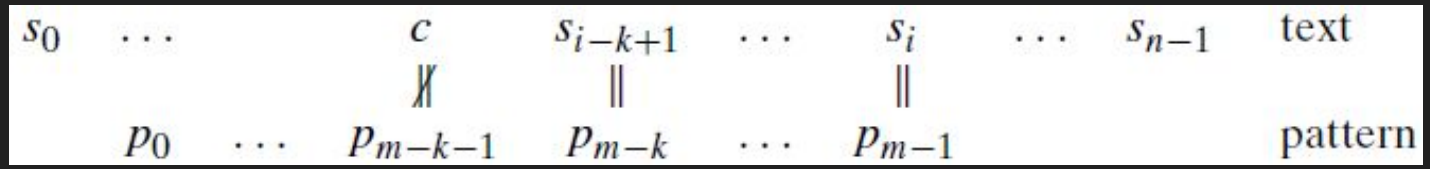
Apply Horspool's algorithm to locate the above pattern in the following DNA sequence:

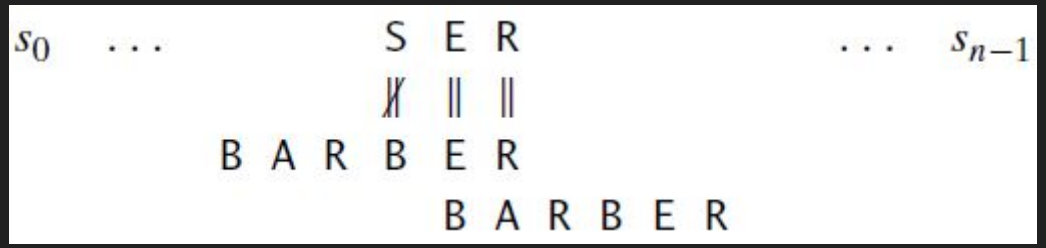
TTATAGATCTCGTATTCTTTTATAGATCTCCTATTCTT

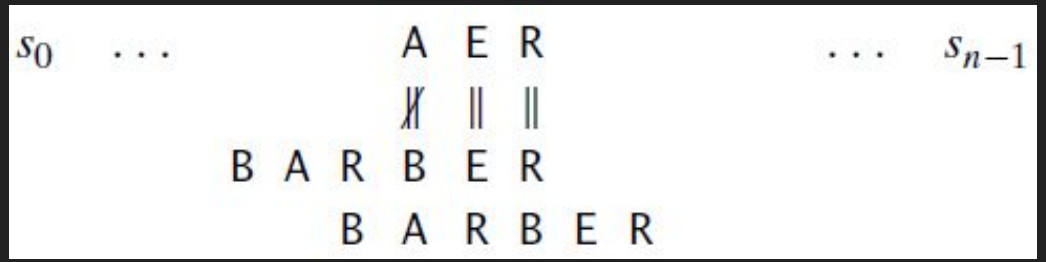
How many character comparisons will be made by Horspool's algorithm in searching for each of the following patterns in the binary text of 1000 zeros?

a. 00001 **b.** 10000 **c.** 01010

Boyer-Moore Algorithm: Bad-symbol shift

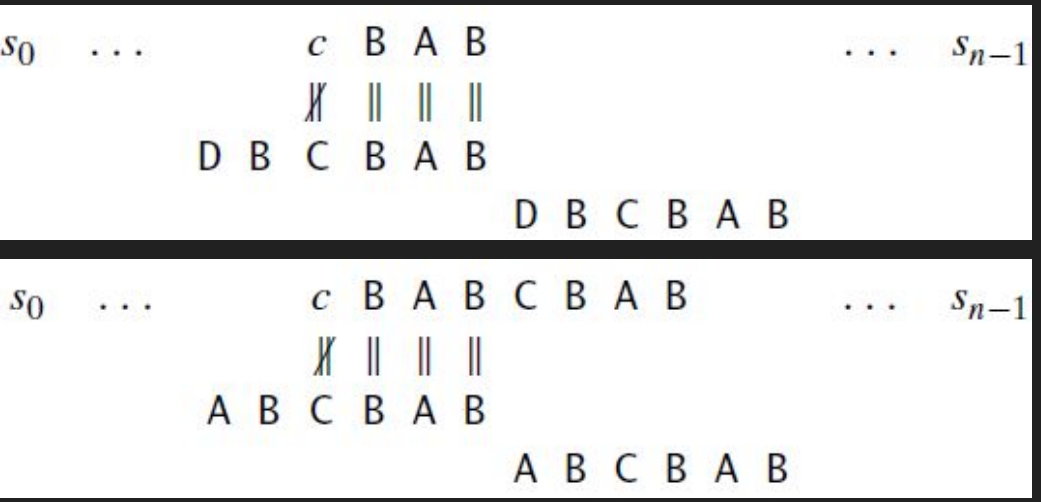






$$d_1 = \max\{t_1(c) - k, 1\}$$

Boyer-Moore Algorithm: Good-suffix shift



k	pattern	d_2
1	ABC$\overline{\text{B}}$A$\underline{\text{B}}$	2
2	$\overline{\text{AB}}$CB$\underline{\text{AB}}$	4
3	$\overline{\text{AB}}$C$\underline{\text{BAB}}$	4
4	$\overline{\text{AB}}$$\underline{\text{CBAB}}$	4
5	$\overline{\text{AB}}$$\underline{\text{CBAB}}$	4

$$d = \begin{cases} d_1 & \text{if } k = 0, \\ \max\{d_1, d_2\} & \text{if } k > 0, \end{cases}$$

where $d_1 = \max\{t_1(c) - k, 1\}$

Boyer-Moore Algorithm:

Step 1: For a given pattern and the alphabet used in both the pattern and the text, construct the bad-symbol shift table as described earlier.

Step 2: Using the pattern, construct the good-suffix shift table as described earlier.

Step 3: Align the pattern against the beginning of the text.

Step 4: Repeat the following step until either a matching substring is found or the pattern reaches beyond the last character of the text.

…

Boyer-Moore Algorithm:

Step 4: Repeat the following step until either a matching substring is found or the pattern reaches beyond the last character of the text. Starting with the last character in the pattern, compare the corresponding characters in the pattern and the text until either all m character pairs are matched (then stop) or a mismatching pair is encountered after k ≥ 0 character pairs are matched successfully. In the latter case, retrieve the entry t1(c) from the c's column of the bad-symbol table where c is the text's mismatched character. If k > 0, also retrieve the corresponding d_2 entry from the good-suffix table. **Shift the pattern to the right by the number of positions computed by the formula**

$$d = \begin{cases} d_1 & \text{if } k = 0, \\ \max\{d_1, d_2\} & \text{if } k > 0, \end{cases}$$

where $d_1 = \max\{t_1(c) - k, 1\}$

```
BoyerMooreMatching(T[0..n-1], P[0..m-1])
    STbl[alphabet size] ← ShiftTbl(P[0..m-1])
    GTbl[1..m-1] ← GoodSuffixTbl(P[0..m-1])
    i ← m-1
    while (i < n)
        j ← 0
        while (j < m and T[i-j] = P[m-1-j])
            j ← j + 1
        if(j = m) return i-(m-1)
        d ← max{STbl[T[i-j]] - j, 1}
        if(j>0)
            d2 ← GTbl[j]
            if(d2 > d) d ← d2
        i ← i + d
    return -1
```

BAOBAB

c	A	B	C	D	. . .	O	. . .	Z	_
$t_1(c)$	1	2	6	6	6	3	6	6	6

k	pattern	d_2
1	BAOB̄AB	2
2	B̄AOBAB	5
3	B̄AOBAB	5
4	B̄AOBAB	5
5	B̄AOBAB	5

B E S S _ K N E W _ A B O U T _ B A O B A B S

B A O B A B

$d_1 = t_1(K) - 0 = 6$

B A O B A B

$d_1 = t_1(\_) - 2 = 4$

$d_2 = 5$

$d = \max\{4, 5\} = 5$

B A O B A B

$d_1 = t_1(\_) - 1 = 5$

$d_2 = 2$

$d = \max\{5, 2\} = 5$

B A O B A B

Eg: Pattern **AGCGC**

Shift table:

A	C	G	T
4	2	1	5

Good-suffix table[1..4]: 5, 2, 5, 5

Gene segment in DNA Sequence using Boyer-Moore's algo.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
A C G T T A G C A G C G C A G C G C
A G C G C
          A G C G C
            A G C G C
              A G C G C
```

STbl[T] = 5

STbl[G] = 1

GTbl[2] = 2

Boyer-Moore algorithm:

Create bad-symbol shift table and good-suffix shift table for the pattern: BAOBABAB

A	B	O	Others
1	2	5	8

k: d_2

1: 4

2: 7

3: 2

4: 7

5,6,7: 7

Thank You :-)

Mr. Channa Bankapur