UNIT 3: Space and Time Tradeoffs

Input Enhancement in String Matching: Horspool’s algorithm

String matching problem

Finding an occurrence of a given string of m characters (***pattern***) in a longer string of n characters (***text***).

Brute force approach: string matching

- simply matches corresponding pairs of characters in the pattern and the text left to right
- if a mismatch occurs, shifts the pattern one position to the right for the next trial.

Note:

maximum number of such trials is $n-m+1$ and, in the worst case, m comparisons need to be made on each of them, the worst-case number of character comparisons is $m(n - m + 1)$. i.e., **$\Theta(nm)$** .

String matching: Input enhancement

Idea:

preprocess the pattern to get some information about it, store this information in a table, and then use this information during an actual search for the pattern in a given text.

Best known example algorithms:

- Knuth-Marris-Pratt algorithm (KMP)
- Boyer-Moore algorithm

Horspool's algorithm

- published by Nigel Horspool in 1980
- simplified version of the Boyer-Moore algorithm
- uses input enhancement idea

Horspool’s algorithm: Strategy

- Match the pattern right to left
- On mismatch, shift the pattern: by +1 character(s)
- Preprocess string to determine shifting - Build a table for shifts for each valid character

Note:

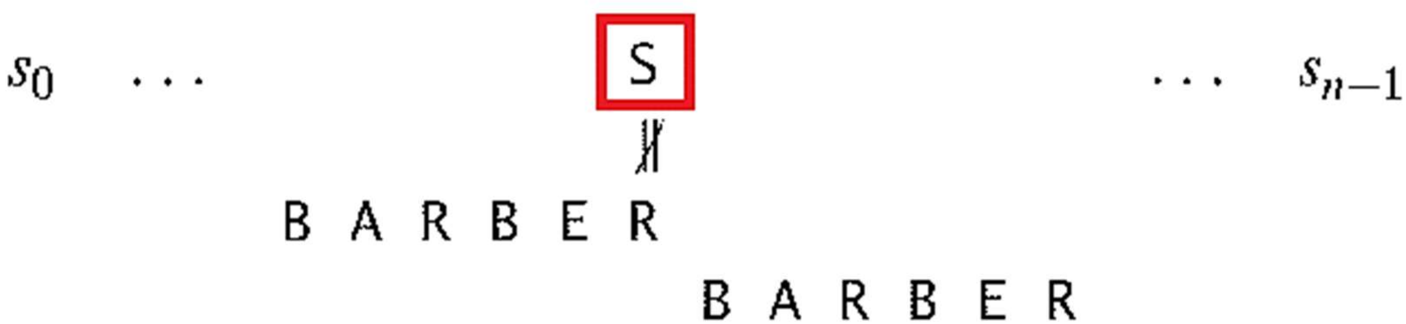
Horspool's algorithm determines the size of a shift by looking at the character c of the text that was aligned against the last character of the pattern.

Horspool’s algorithm: Case 1

- If there are no letter say 'S' in the pattern, we can safely **shift the pattern by its entire length**

Example:

string is $s_0s_1\ldots S\ldots s_{n-1}$ and pattern is “BARBER”

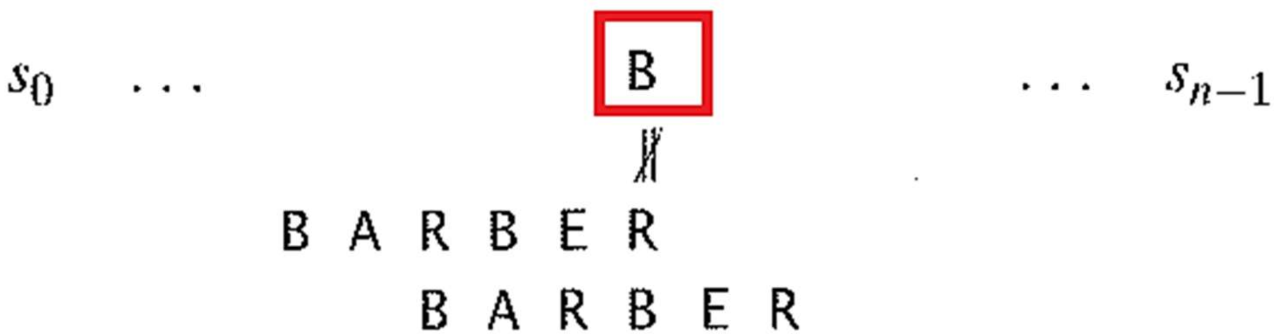


Horspool's algorithm: Case 2

- If there are occurrences of character c in the pattern but it is not the last one there. Then **shift should align the rightmost occurrence of c in the pattern with the c in the text**

Example:

string is $s_0s_1...B...s_{n-1}$ and pattern is "BARBER"



Horspool’s algorithm: Case 3

- If c happens to be the last character in the pattern but there are no c's among its other m-1 characters, the **shift should be similar to that of Case 1: the pattern should be shifted by the entire pattern's length m**

Example:

string is s_0s_1...MER...s_{n-1} and pattern is “LEADER”

Horspool’s algorithm: Case 4

- if c happens to be the last character in the pattern and there are other c's among its first m-1 characters, the **shift should be similar to that of Case 2: the rightmost occurrence of c among the first m - 1 characters in the pattern should be aligned with the text's c**

Example:

string is s_0s_1…OR…s_{n-1} and pattern is “REORDER”

Horspool's algorithm: Shift size computation

Shift table entries are computed by the formula

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

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 initialize all the elements of *Table* with m

for $j \leftarrow 0$ **to** $m-2$ **do** $Table[P[j]] \leftarrow m-1-j$
return *Table*

Shift table for the pattern "BARBER"

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

```
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
```

NOTE:
LOOK for FIRST SYMBOL(from right) and refer to its shift in the shift table in case of mismatch.

Pattern: "BARBER“
Text: “JIM_SAW_ME_IN_A_BARBERSHOP”

```
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
```

Let’s check our understanding

Text: BESS_KNEW_ABOUT_BAOBABS

Pattern: BAOBAB

- Generate the shift table
- Show the steps of the algorithm

Let's check our understanding

Text: JIMY_HAILED_THE_LEADER_TO_STOP

Pattern: LEADER

- Generate the shift table
- Show the steps of the algorithm

Let’s check our understanding

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:

T C C T A T T C T T

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

T T A T A G A T C T C G T A T T C T T T T A T A G A T C T C C T A T T C T T