

# MA/CSSE 473

## Day 27

(No class Day 26)  
String Search

Horspool

Boyer-Moore



# MA/CSSE 473 Day 27

- HW 11 due tomorrow
- HW 12 due Oct 30 (available tomorrow)
- Exam 2 is Friday, Oct 31
- **Student Questions**
- Horspool String search algorithm
- Boyer-Moore



# Brute Force String Search Example

What makes brute force so slow?

When we find a mismatch, we can shift the pattern by only one character position in the text.

**Text:**        abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra

**Pattern:** **abracadabra**

abracadabra

abracadabra

a**br**acadabra

abracadabra

abracadabra



# Recap: Horspool's Algorithm

- A simplified version of the Boyer-Moore algorithm
- A good bridge to understanding Boyer-Moore
- Like Boyer-Moore, Horspool does the comparisons in a counter-intuitive order (moves right-to-left through the pattern)



# Horspool's Main Question

- If there is a character mismatch, how far can we shift the pattern, with no possibility of missing a match within the text?
- What if the last character in the pattern is compared with a character in the text that does not occur in the pattern at all?
- Text :            . . . ABCDEFG . . .  
Pattern:            BOUTELL



# How Far to Shift?

- Look at first (rightmost) character in text that was compared:

- The character is not in the pattern

.....C.....C not in pattern)

AOBAB

- The character is in the pattern (but not the rightmost)

.....O.....(O occurs once in pattern)  
BAOBAB

.....A.....(A occurs twice in pattern)

BAOBAB

- The rightmost characters do match

.....B.....

BAOBAB



# Harpool Shift Table

- Shift sizes can be precomputed by the formula

$$t(c) = \begin{cases} \text{distance from } c\text{'s rightmost occurrence in} \\ \text{pattern among its first } m-1 \text{ characters from its} \\ \text{right end} \\ \text{pattern's length } m, \text{ otherwise} \end{cases}$$

by scanning the pattern before the search begins, and stored in a table called the *shift table*



# Shift Table Example

- Shift table is indexed by text and pattern alphabet

E.g., for BAOBAB :

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 1 | 2 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 3 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |



# Example of Horspool's Algorithm

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | - |   |
| 1 | 2 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 3 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |

BARD LOVED BANANAS

BAOBAB

BAOBAB

BAOBAB

BAOBAB (unsuccessful search)



# Horspool Code

```
def populateShiftTable(table, pattern, mMinusOne):  
    for i in range(mMinusOne):  
        table[ord(pattern[i])] = mMinusOne - i
```

```
def search(pattern, text):  
    n, m = len(text), len(pattern)  
    shiftTable = [m]*128  
    populateShiftTable(shiftTable, pattern, m-1)  
  
    i = m - 1  
    while i < n:  
        k = 0  
        while k < m and pattern[m-1-k]==text[i-k]:  
            k += 1;  
        if k==m:  
            return i - m + 1  
        i = i + shiftTable[ord(text[i])]  
    return -1
```

# Horspool Example

pattern = abracadabra

text =

abracadabtabradabracadabcbadaxbrabbracadabraxxxxxxabracadabracadabra

shiftTable: a3 b2 r1 a3 c6 a3 d4 a3 b2 r1 a3 x11

abracadabtabradabracadabcbadaxbrabbracadabraxxxxxxabracadabracadabra  
abracadabra

abracadabtabradabracadabcbadaxbrabbracadabraxxxxxxabracadabracadabra  
abracadabra

abracadabtabradabracadabcbadaxbrabbracadabraxxxxxxabracadabracadabra  
abracadabra

abracadabtabradabracadabcbadaxbrabbracadabraxxxxxxabracadabracadabra  
abracadabra

abracadabtabradabracadabcbadaxbrabbracadabraxxxxxxabracadabracadabra  
abracadabra

abracadabtabradabracadabcbadaxbrabbracadabraxxxxxxabracadabracadabra  
abracadabra

abracadabtabradabracadabcbadaxbrabbracadabraxxxxxxabracadabracadabra  
abracadabra



# Horspool Example Continued

```
pattern = abracadabra
```

```
text =
```

```
abracadabtabradabracadabcadaxbrabbracadabraxxxxxabracadabracadabra
```

```
shiftTable:  a3 b2 r1 a3 c6 a3 d4 a3 b2 r1 a3 x11
```

```
abracadabtabradabracadabcadaxbrabbracadabraxxxxxabracadabracadabra  
                abracadabra
```

```
abracadabtabradabracadabcadaxbrabbracadabraxxxxxabracadabracadabra  
                abracadabra
```

```
abracadabtabradabracadabcadaxbrabbracadabraxxxxxabracadabracadabra  
                abracadabra
```

```
abracadabtabradabracadabcadaxbrabbracadabraxxxxxabracadabracadabra  
                abracadabra
```

```
abracadabtabradabracadabcadaxbrabbracadabraxxxxxabracadabracadabra  
                abracadabra
```

```
abracadabtabradabracadabcadaxbrabbracadabraxxxxxabracadabracadabra  
                abracadabra
```

49

With Brute force, we would have to compare the pattern to 50 different positions in the text before we find it; with Horspool, only 13 positions are tried.



# Boyer Moore Intro

- When determining how far to shift after a mismatch, Horspool only uses the text character corresponding to the rightmost pattern character
- Often there is a partial match (from the right) before a mismatch occurs
- Boyer-Moore takes into account  $k$ , the number of matched characters (from the right) before a mismatch occurs.
- If  $k=0$ , we simply do the same shift as Horspool's algorithm.



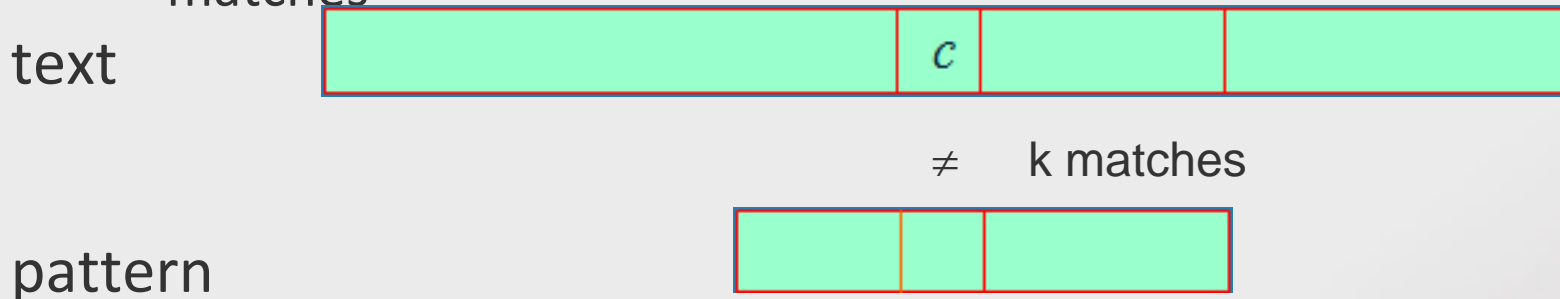
# Boyer-Moore Algorithm

- Based on same two main ideas:
- compare pattern characters to text characters from right to left
- precompute the shift amounts in **two** tables
  - **bad-symbol table** indicates how much to shift based on the text's character that causes a mismatch
  - **good-suffix table** indicates how much to shift based on matched part (suffix) of the pattern



# Bad-symbol shift in Boyer-Moore

- If the rightmost character of the pattern does not match, Boyer-Moore algorithm acts much like Horspool's
- If the rightmost character of the pattern does match, BM compares preceding characters right to left until either
  - all pattern's characters match, or
  - a mismatch on text's character  $c$  is encountered after  $k > 0$  matches



bad-symbol shift: How much should we shift by?

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

where  $t_1(c)$  is the value from the Horspool shift table.



# Good-suffix Shift in Boyer-Moore

- Good-suffix shift  $d_2$  is applied after the  $k$  last characters of the pattern are matched
  - $0 < k < m$
- How can we take advantage of this?
- As in the bad suffix table, we want to precompute some information based on the characters in the suffix.
- We want to create a **good suffix table** whose indices are  $k = 1 \dots m-1$ , and whose values are how far we can shift after matching a  $k$ -character suffix (from the right).
- Spend 5 minutes talking with one or two other students. Try to come up with criteria for how far we can shift.
- Example patterns: CABABA      AWOWWOW  
                         WOWWOW    ABRACADABRA



# Boyer-Moore Algorithm

After matching successfully  $0 < k < m$  characters, the algorithm shifts the pattern right by

$$d = \max \{d_1, d_2\}$$

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

$d_2(k)$  is good-suffix shift



# Boyer-Moore Example

- On Moore's home page
- <http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/fstrpos-example.html>

