MA/CSSE 473 Day 25

Student questions

String search

Horspool

Boyer Moore intro

Brute Force, Horspool, Boyer-Moore

STRING SEARCH

Brute Force String Search Example

The problem: Search for the first occurrence of a **pattern** of length m in a **text** of length n.
Usually, m is much smaller than n.

- 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
            abracadabra
             abracadabra
              abracadabra
```

Faster String Searching

- Brute force: **worst case m(n-m+1)** **Was a HW problem**
- A little better: **but still Θ(mn) on average**
  - **Short-circuit the inner loop**

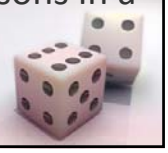
```
def search(pattern, text):
    n, m = len(text),len(pattern)
    for i in range(n-m+1):
        j = 0
        while j < m and text[i+j] == pattern[j]:
            j +=1
        if j==m:
            return i
    return False
```

What we want to do

- When we find a character mismatch
  - Shift the pattern as far right as we can
  - With no possibility of skipping over a match.

Horspool's Algorithm

- A simplified version of the Boyer-Moore algorithm
- A good bridge to understanding Boyer-Moore
- Published in 1980
- Recall: What makes brute force so slow?
  - When we find a mismatch, we can only shift the pattern to the right by one character position in the text.
  -
```
Text:    abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
Pattern: abracadabra
          abracadabra
           abracadabra
            abracadabra
```
- Can we sometimes shift farther?
  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 to a character in the text that does not occur anywhere in the pattern?
- Text: ... ABCDEFG ...
  Pattern: CSSE473

How Far to Shift?

- Look at first (rightmost) character in the part of the text that is compared to the pattern:
- The character is not in the pattern

  .....**C**..... {*C* not in pattern)

  BAOBAB
- 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

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

- **EXERCISE:** Create the shift table for COCACOLA (on your handout)

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

BARD LOVED BANANAS (this is the text)
BAOBAB (this is the pattern)
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):
    """ return index of first occurrence of pattern in text;
        return -1 if no match """
    n, m = len(text), len(pattern)
    shiftTable = [m]*128 # if char not in pattern, shift by full amount
    populateShiftTable(shiftTable, pattern, m-1)

    i = m - 1 # i is position in text that corresponds to end of pattern

    while i < n:  # while not past end of text
        k = 0 # k is number of pattern characters compared so far

        while k < m and pattern[m-1-k]==text[i-k]:
            k += 1;  # loop stops if mismatch or complete match

        if k==m:  # found a match
            return i - m + 1

        i = i + shiftTable[ord(text[i])] # ready to begin next comparison
    return -1
```

Horspool Example

```
pattern = abracadabra
text =
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
shiftTable:  a3 b2 r1 a3 c6 a3 d4 a3 b2 r1 a3 x11
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
   abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
      abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
        abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
           abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
              abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                 abracadabra
```

Continued on next slide

Horspool Example Continued

```
pattern = abracadabra
text =
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
shiftTable:  a3 b2 r1 a3 c6 a3 d4 a3 b2 r1 a3 x11
```

```
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                  abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                            abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                               abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                                  abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                                                 abracadabra
abracadabtabradabracadabcadaxbrabbracadabraxxxxxxabracadabracadabra
                                                    abracadabra
49
```

Using 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

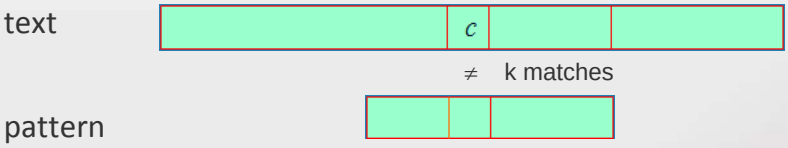
- When determining how far to shift after a mismatch
  - Horspool only uses the text character corresponding to the rightmost pattern character
  - Can we do better?
- Often there is a partial match (on the right end of the pattern) before a mismatch occurs
- Boyer-Moore takes into account k, the number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

Boyer-Moore Algorithm

- Based on 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
- 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.

Boyer-Moore Algorithm

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

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

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

$d_2(k)$ is the good-suffix shift

Remaining question:
How to compute good-suffix shift table?

$d_2[k]$ = ???

Good-suffix Shift in Boyer-Moore

- Good-suffix shift d_2 is applied after the k last characters of the pattern are successfully matched
  - $0 < k < m$
- How can we take advantage of this?
- As in the bad suffix table, we want to pre-compute some information based on the characters in the suffix.
- We create a **good suffix table** whose indices are k = 1...m-1, and whose values are how far we can shift after matching a k-character suffix (from the right).
- **Spend some time 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

Can you figure these out?

4. For each given string, fill in the good-suffix table from the Boyer-Moore algorithm.

1. banana

k	shift
1	
2	
3	
4	
5	

2. wowwow

k	shift
1	
2	
3	
4	
5	

3. abcdcbcabcabc

k	shift
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Boyer-Moore example (Levitin)

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

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

$d_1 = t_1(K) = 6$

B A O B A B

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

$\underline{d_2(2) = 5}$

B A O B A B

$\underline{d_1 = t_1(\_)-1 = 5}$

$d_2(1) = 2$

B A O B A B (success)

k	pattern	d_2
1	BAOBAB	2
2	BAOBAB	5
3	BAOBAB	5
4	BAOBAB	5
5	BAOBAB	5