CS 303, Paul Cao,
Topic: String matching algorithms (Horspool, BM) – extra topic

1. Horspool's string searching algorithm

Review of brute force algorithm
pattern: a string of m characters to search for
text: a (long) string of n characters to search in

Brute force algorithm

Step 1: Align pattern at beginning of text
Step 2: Moving from left to right, compare each character of pattern to the corresponding character in text until either all characters are found to match (successful search) or a mismatch is detected
Step 3:While a mismatch is detected and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2

Efficiency: O(mn) in worst case scenario

Idea of better algorithms: Since we know everything about the pattern, maybe we can use the input enhancement idea of preprocessing the pattern.

- Horspool's algorithm simplifies the Boyer-Moore algorithm by using just one table
- Boyer-Moore algorithm preprocesses pattern right to left and store information into two tables
- Knuth-Morris-Pratt (KMP) algorithm preprocesses pattern left to right to get useful information for later searching (not covered)

Horspool's idea: Since we know everything about the short pattern, can we generate a table to tell us how much we can shift under different conditions?

It is a simplified version of Boyer-Moore algorithm:

- preprocesses pattern to generate a **shift table** that determines how much to shift the pattern when a mismatch occurs
- always makes a shift based on the text's character *c* aligned with the last character in the pattern according to the shift table's entry for *c*

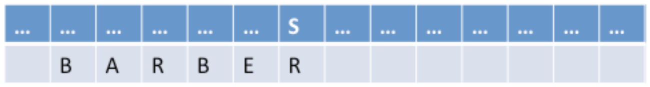
Details
Let $p_1..p_m$ be the m characters in the pattern.

...	...	...	...	...	...	c	...	...	...	...	...	...	...
	p_1	p_2	p_3	...	p_{m-1}	p_m							

Look at first (rightmost) character in text that was compared. They may be several scenarios

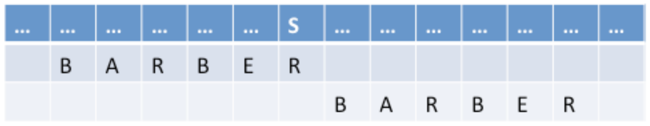
CS 303, Paul Cao,

1. The character is not in the pattern (c is not in the pattern)



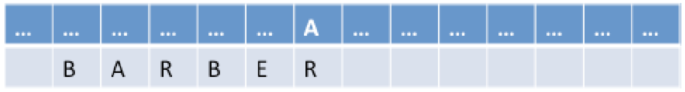
...	...	...	...	...	...	S	...	...	...	...	...	...	...
	B	A	R	B	E	R							

e.g.
Q:/ What should be our action?
A:/ shift the pattern to the right by m positions



...	...	...	...	...	...	S	...	...	...	...	...	...	...
	B	A	R	B	E	R							
							B	A	R	B	E	R	

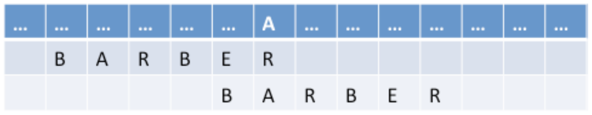
2. The character is in the pattern (but not the rightmost)



...	...	...	...	...	...	A	...	...	...	...	...	...	...
	B	A	R	B	E	R							

e.g.

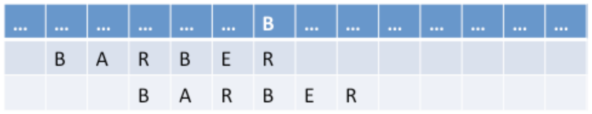
Q:/ What should we do?
A:/ shift the pattern such that the rightmost occurrence of c in the pattern with the c in the text



...	...	...	...	...	...	A	...	...	...	...	...	...	...
	B	A	R	B	E	R							
					B	A	R	B	E	R			

shift: 4

Another example



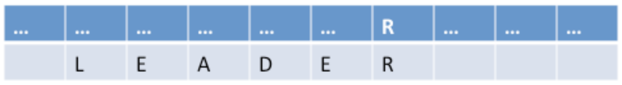
...	...	...	...	...	...	B	...	...	...	...	...	...	...
	B	A	R	B	E	R							
			B	A	R	B	E	R					

shift: 2

Q:/ From the previous two examples, can you decide the number of shifts we should do in scenario 2?

A:/ shift the pattern to the right by the distance of the right most c among the first m-1 characters of the pattern to its last character.

3. The rightmost characters do match



...	...	...	...	...	...	R	...	...	...
	L	E	A	D	E	R			

e.g.
Q:/ Action?
A:/ Keep comparing to the left.

Q:/ However, if we see a mismatch on the left, what should we do?

scenario 3.1: there are no c's among the first m-1 characters ⇒ shift m positions to the right
e.g.

...	...	...	...	M	E	R	...	...	...	...	...	...	...
	L	E	A	D	E	R							
							L	E	A	D	E	R	

scenario 3.2: there are other c's in the first m-1 characters. ⇒ shift the pattern to the right by the distance of the right most c among the first m-1 characters of the pattern to its last character.
e.g.

...	...	...	...	M	E	R	...	...	...	...	...	...	...
	B	A	R	B	E	R							
				B	A	R	B	E	R				

Overall, the shift table can be computed using the following formula. Note the c is the character in the text, not in the pattern.

t(c)= { distance from *c*'s rightmost occurrence in pattern among its first *m*-1 characters to its right end
pattern's length *m*, otherwise

We scan the pattern before search begins and store the result in a table called **shift table**
e.g. pattern is BAOBAB
Shift table can be built as

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																	
						B	A	O	B	A	B											
								B	A	O	B	A	B									
														B	A	O	B	A	B			
																B	A	O	B	A	B	

SUCCESSFUL SEARCH!

Efficiency: Worst case scenario: still O(mn)

- text: all n zeros: 000000000000000
- pattern: 1 followed by m-1 zeros

Shift table:

0	1
1	m-1

But Horspool's performance is faster on random text compared with brute force.

BM algorithm improves from Horspool's algorithm

- Horspool: only compare the characters of pattern and text one by one ⇨ one character pattern where basically every character is considered to be independent from each other.
- Boyer-Moore: can we find more patterns? ⇨ k-character patterns
- Use two tables
    - Bad-symbol table
    - Good-suffix table

Similar to Horspool, BM tries to compare the pattern from the right to the left but uses a more complicated way to decide how much to shift.

1. Bad symbol table

Purpose of this table: decide how much to shift based upon the **character of the text** that caused the mismatch. (not the character in the text aligned with the last char of the pattern like in Horspool)

Scenario 1:

If the last character is a mismatch ⇨ same as Horsloop (using the shift table)

e.g.

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	_
4	2	6	6	1	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6

…	…	…	…	…	…	S	…	…	…	…	…	…	…
	B	A	R	B	E	R							
							B	A	R	B	E	R	

…	…	…	…	…	…	A	…	…	…	…	…	…	…
	B	A	R	B	E	R							
					B	A	R	B	E	R			

Scenario 2:

If there are some k (0<=k<m) of the pattern's characters are matched successfully before a mismatch is found.

CS 303, Paul Cao,

$$\begin{array}{cccccccccc} s_1 & \ldots & & c & s_{i-k} & \ldots & s_i & \ldots & s_n & \textbf{text} \\ & & & \neq & = & & = & & & \\ & p_1 & \ldots & p_{m-k-1} & p_{m-k} & \ldots & p_m & & & \textbf{pattern} \end{array}$$

We can compare the effect of Horspool and BM

Example1

BM: since S is not in the pattern, we can shift 4 positions directly

Shift=4=t(S)-2=t(S)-k

…	…	…	…	S	E	R	…	…	…	…	…	…	…
	B	A	R	B	E	R							
					B	A	R	B	E	R			

Example 2

BM: since A is in the pattern, we will shift the next rightmost A to align with the current mismatched A. ▭

2 positions Shift=2=t(A)-2=t(A)-k

…	…	…	…	A	E	R	…	…	…	…	…	…	…
	B	A	R	B	E	R							
			B	A	R	B	E	R					

Example 3

BM: since there is no other E to the left in the pattern, we can shift one.
Shift=1 (because t(E)-2≤0)

Q:/ Can you summarize the way BM shifts with a bad symbol?
A:/ Shift to match the "bad symbol". ▭bad-symbol table

- bad-symbol table indicates how much to shift based on text's character causing a mismatch.
- bad-symbol shift $d_1 = \max\{t_1(c) - k, 1\}$ ▭ same idea as horspool.