

# 122COM: Boyer-Moore Searching

Coventry University

## String searching better

A

- Naive search works but is inefficient.
- Obvious solution is not always the best one.
- Think about the problem and what is being searched.
  - Can you be smarter?

Boyer-Moore string searching algorithm.

- 1977.
- Not going to talk about the whole algorithm here.
  - Gets complex.
- Right to left comparison.
- Can skip sections of the text.
  - Don't need to test every position.
  - How?

Boyer-Moore string searching algorithm.

- 1977.
- Not going to talk about the whole algorithm here.
  - Gets complex.
- Right to left comparison.
- Can skip sections of the text.
  - Don't need to test every position.
  - How?
- Pre-processes the search string.
  - Produce bad character rule table.
  - Will explain how in a minute.

- 1 Preprocess the search string to create the “bad character table”.
  - Will explain how in a minute.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

- 2 Same as the naive search, position the search string at the start of the main text.
- 3 Compare the strings.
- 4 If strings don't match then in the bad character table lookup the character positioned at the end of the search string.
- 5 Move the search string by the number of positions specified in the table.
- 6 Repeat from step 3.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

text =	t	h	i	s	_	i	s	_	a	n	_	e	x	a	m	p	l	e
search =	e	x	a	m	p	l	e											

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

text = 

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

search = 

e	x	a	m	p	l	e
---	---	---	---	---	---	---

↓

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

7  
↓

text =	t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
search =	e	x	a	m	p	l	e											



example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

7  
↓  
text = 

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  
search = 

e	x	a	m	p	l	e
---	---	---	---	---	---	---

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  

	e	x	a	m	p	l	e
--	---	---	---	---	---	---	---

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

text = 

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  
search = 

e	x	a	m	p	l	e
---	---	---	---	---	---	---

Diagram illustrating the first step of the Boyer-Moore search algorithm. The text "this is an example" is shown. The search string "example" is aligned under the text, starting at index 0. A mismatch is found at index 7 (the character 's' in the text vs 'e' in the search string). The value 7 is shown above the mismatch, with a downward arrow indicating the shift.

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  

					e	x	a	m	p	l	e
--	--	--	--	--	---	---	---	---	---	---	---

Diagram illustrating the second step of the Boyer-Moore search algorithm. The search string "example" is shifted to the right by 4 positions, as indicated by the value 4 and a downward arrow. The search string is now aligned under the text, starting at index 4. A mismatch is found at index 10 (the character 'n' in the text vs 'a' in the search string).

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

7  
↓  
text = 

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  
search = 

e	x	a	m	p	l	e
---	---	---	---	---	---	---

4  
↓  

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  

e	x	a	m	p	l	e
---	---	---	---	---	---	---

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  

e	x	a	m	p	l	e
---	---	---	---	---	---	---

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

7  
↓  
text = 

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  
search = 

e	x	a	m	p	l	e
---	---	---	---	---	---	---

4  
↓  

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  

e	x	a	m	p	l	e
---	---	---	---	---	---	---

6  
↓  

t	h	i	s		i	s		a	n		e	x	a	m	p	l	e
---	---	---	---	--	---	---	--	---	---	--	---	---	---	---	---	---	---

  

e	x	a	m	p	l	e
---	---	---	---	---	---	---

Creating the bad character table.

- For each character.
  - Except the last.
- Just count number of places between it and end of search string.

example  $\Rightarrow$  a e l m p x \*

Creating the bad character table.

- For each character.
  - Except the last.
- Just count number of places between it and end of search string.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
						4

Creating the bad character table.

- For each character.
  - Except the last.
- Just count number of places between it and end of search string.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6					

Creating the bad character table.

- For each character.
  - Except the last.
- Just count number of places between it and end of search string.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1				



Creating the bad character table.

- For each character.
  - Except the last.
- Just count number of places between it and end of search string.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3			

Creating the bad character table.

- For each character.
  - Except the last.
- Just count number of places between it and end of search string.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2		

Creating the bad character table.

- For each character.
  - Except the last.
- Just count number of places between it and end of search string.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	

Creating the bad character table.

- For each character.
  - Except the last.
- Just count number of places between it and end of search string.

example  $\Rightarrow$ 

a	e	l	m	p	x	*
4	6	1	3	2	5	7

Doesn't need to sort or modify the sequence being searched.

- Small amount of pre-processing on the search value.

Worst case.

- Linear time.

Average case

- Sub-linear.

Not the only improved string searching algorithm.

- Knuth-Morris-Pratt.
- Finite State Machine (FSM).
- Rabin-Karp.

# The End