122COM: Searching

David Croft

Coventry University

david.croft@coventry.ac.uk

2017

Overview

Introduction

Searching is used everywhere in computing.

- Obvious applications.
  - Text files.
  - Databases.
  - File systems.
  - Search engines.
- Hidden applications.
  - Computer games.
    - Field Of View (FOV) search for objects in view.
    - Path finding `https://www.youtube.com/watch?v=19h1g22hby8`.
  - Network routing.
  - Sat Nav.
  - Recommender systems.
    - Netflix What-to-watch.
    - Amazon recommended items.

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
Z

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
Z

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑

Z

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑

R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.
- $O(n)$
  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.

$O(n)$

  - Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Linear search

C

Simplest searching algorithm.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.
  - Worst case if the value isn't in the sequence at all.

$O(n)$

- Will discuss $O()$ notation in a later week.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	Z	Q	K	L	G	H	U	A	P	L	F	N	R

↑
R

Binary search

A Divide & conquer algorithm.

- Pro: Muuuuuuch faster than linear search.
- Con: Only works on sorted sequences.
- The algorithm:
  1. Find middle value of the sequence.
  2. If search value == middle value then success.
  3. If search value is $<$ middle value then forget about the top half of the sequence.
  4. If search value is $>$ middle value then forget about the bottom half of the sequence.
  5. Repeat from step 1 until `len(sequence)==0`.

Binary search II

I

Find E.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Binary search II

I

Find E.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Binary search II

I

Find E.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

↑

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Binary search II

I

Find E.

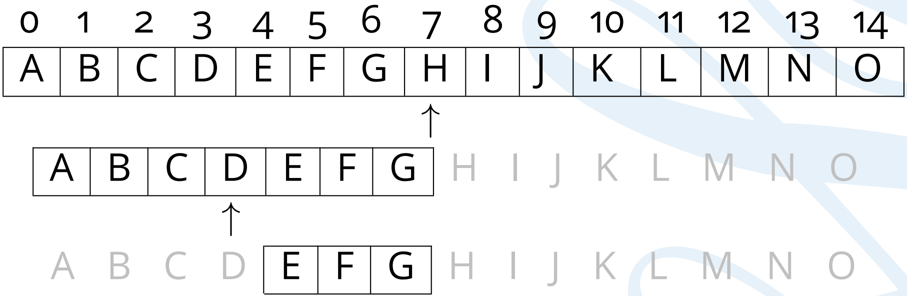
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Binary search II

I

Find E.



Binary search II

I

Find E.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
							↑							
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
			↑											
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
					↑									

Binary search II

I

Find E.

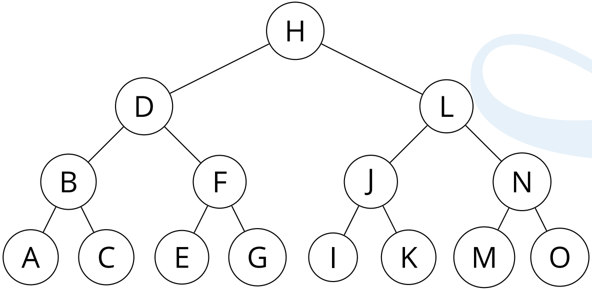
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
							↑							
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
			↑											
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
					↑									
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Complexity

A

Maximum number of comparisons needed? Binary Search Trees.

- How many times can we divide our sequence in half?

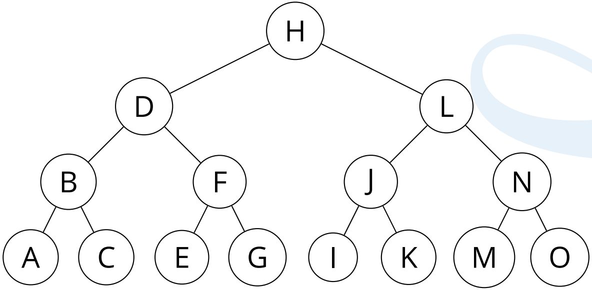


Complexity

A

Maximum number of comparisons needed? Binary Search Trees.

- How many times can we divide our sequence in half?
- Ideal depth of the tree is $\log_2(n)$
  - $n = 15$ in this example.
  - $\log_2(15) = 3.9 \Rightarrow 3$

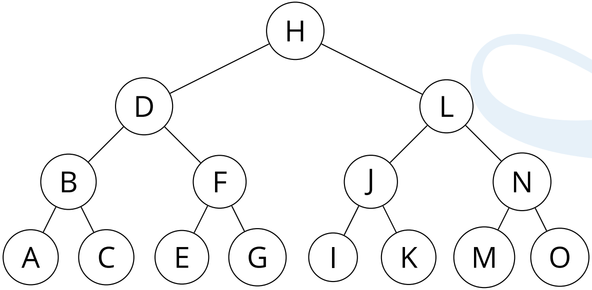


Complexity

A

Maximum number of comparisons needed? Binary Search Trees.

- How many times can we divide our sequence in half?
- Ideal depth of the tree is $\log_2(n)$
  - $n = 15$ in this example.
  - $\log_2(15) = 3.9 \Rightarrow 3$
- Binary search has a complexity of $O(\log n)$.
  - Will cover $O()$ complexity in later week.

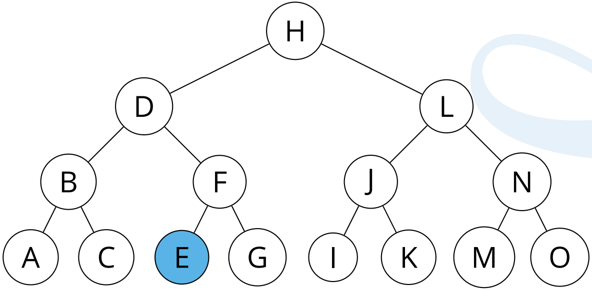


Complexity

A

Maximum number of comparisons needed? Binary Search Trees.

- How many times can we divide our sequence in half?
- Ideal depth of the tree is $\log_2(n)$
  - $n = 15$ in this example.
  - $\log_2(15) = 3.9 \Rightarrow 3$
- Binary search has a complexity of $O(\log n)$.
  - Will cover $O()$ complexity in later week.
- Find E.

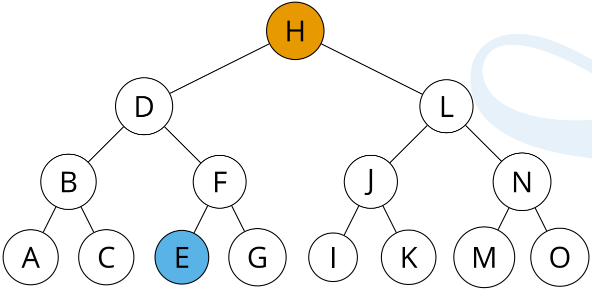


Complexity

A

Maximum number of comparisons needed? Binary Search Trees.

- How many times can we divide our sequence in half?
- Ideal depth of the tree is $\log_2(n)$
  - $n = 15$ in this example.
  - $\log_2(15) = 3.9 \Rightarrow 3$
- Binary search has a complexity of $O(\log n)$.
  - Will cover $O()$ complexity in later week.
- Find E.

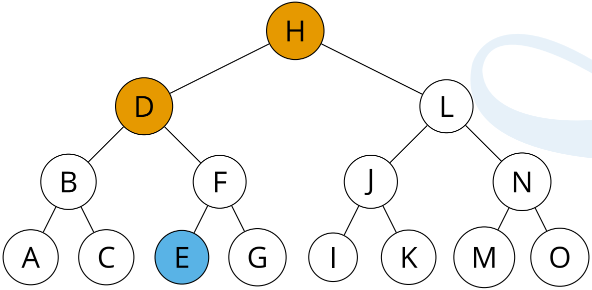


Complexity

A

Maximum number of comparisons needed? Binary Search Trees.

- How many times can we divide our sequence in half?
- Ideal depth of the tree is $\log_2(n)$
  - $n = 15$ in this example.
  - $\log_2(15) = 3.9 \Rightarrow 3$
- Binary search has a complexity of $O(\log n)$.
  - Will cover $O()$ complexity in later week.
- Find E.

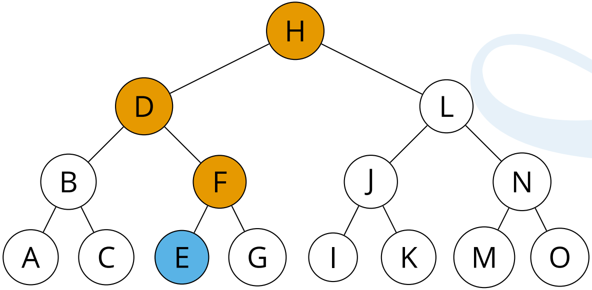


Complexity

A

Maximum number of comparisons needed? Binary Search Trees.

- How many times can we divide our sequence in half?
- Ideal depth of the tree is $\log_2(n)$
  - $n = 15$ in this example.
  - $\log_2(15) = 3.9 \Rightarrow 3$
- Binary search has a complexity of $O(\log n)$.
  - Will cover $O()$ complexity in later week.
- Find E.



It’s HOW much faster?!?!!

I

Clearly much faster than linear search.

- To search a trillion elements linearly could mean a trillion comparisons.
- Binary search does it in 40.

But...

- Have to sort the list first.
- Sorting lists can be expensive.
  - Will cover sorting in a later week.
- Can’t always sort sequences.
- Ordering can be important.
  - E.g. Words in text documents.
  - E.g. Genes in genetic chromosomes.

Break

String searching

C

I.e. Text searching.

- Finding one sequence in another sequence.
- Naive search.
  - Like linear search but with multiple values to compare.
  - Is very slow.

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										

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								

etc, etc, etc.

String searching better

I

- 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

I

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

I

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.

Boyer-Moore algorithm

A

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

$$\text{example} \Rightarrow \begin{array}{ccccccc} a & e & l & m & p & x & * \\ \hline 4 & 6 & 1 & 3 & 2 & 5 & 7 \end{array}$$

2. Same at 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.

Boyer-Moore II

A

example ⇒

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											

Boyer-Moore II

A

example ⇒

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											

Boyer-Moore II

A

example ⇒

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

7
↓

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

Boyer-Moore II

A

example ⇒

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

7
↓

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

Boyer-Moore II

A

example ⇒

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

7 ↓ (over the 7th character of text, "s")

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 ↓ (over the 14th character of text, "a")

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

Boyer-Moore II

A

example ⇒

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

Boyer-Moore II

A

example ⇒

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

Coventry University

Boyer-Moore III

A

Creating the bad character table.

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

$$\text{example} \Rightarrow \frac{\text{a} \quad \text{e} \quad \text{l} \quad \text{m} \quad \text{p} \quad \text{x} \quad *}{}$$

Boyer-Moore III

A

Creating the bad character table.

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

$$\text{example} \Rightarrow \begin{array}{ccccccc} a & e & l & m & p & x & * \\ \hline 4 & & & & & & \end{array}$$

Boyer-Moore III

A

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					

Boyer-Moore III

A

Creating the bad character table.

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

example ⇒

a	e	l	m	p	x	*
4	6	1				

Boyer-Moore III

A

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			

Boyer-Moore III

A

Creating the bad character table.

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

example ⇒

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

Boyer-Moore III

A

Creating the bad character table.

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

$$\text{example} \Rightarrow \begin{array}{ccccccc} a & e & l & m & p & x & * \\ \hline 4 & 6 & 1 & 3 & 2 & 5 & \end{array}$$

Boyer-Moore III

A

Creating the bad character table.

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

$$\text{example} \Rightarrow \frac{\begin{array}{ccccccc} a & e & l & m & p & x & * \end{array}}{\begin{array}{ccccccc} 4 & 6 & 1 & 3 & 2 & 5 & 7 \end{array}}$$

Boyer-Moore IV

A

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 string searching algorithm.

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

Quiz

Q1

Which search is faster?

1. Linear.
2. Binary.

Q1

Which search is faster?

1. Linear.
2. Binary.

Q2

By what other name is linear search known?

1. Divide & Conquer.
2. Binary search.
3. Sequential search.
4. Path finding.

Q2

By what other name is linear search known?

1. Divide & Conquer.
2. Binary search.
3. Sequential search.
4. Path finding.

Q3

What is the downside of binary search compared to linear?

1. Can only search sequences.
2. Can only search numbers.
3. Can only search sorted sequences.
4. Can only search an even number of things.

Q3

What is the downside of binary search compared to linear?

1. Can only search sequences.
2. Can only search numbers.
3. Can only search sorted sequences.
4. Can only search an even number of things.

Q4

Booyer-Moore is faster for searching text than a Naive search because ____.

1. No it isn't.
2. It skips over portions of the text.
3. It uses binary search.
4. It's a divide & conquer algorithm.

Q4

Booyer-Moore is faster for searching text than a Naive search because _____.

1. No it isn't.
2. It skips over portions of the text.
3. It uses binary search.
4. It's a divide & conquer algorithm.

Q5

The $O()$ complexity of binary search is _____.

1. $O(n)$
2. It depends on how many elements are being searched.
3. $O(\log n)$
4. $O(n!)$

Q5

The $O()$ complexity of binary search is _____.

1. $O(n)$
2. It depends on how many elements are being searched.
3. $O(\log n)$
4. $O(n!)$

Why do I care?

Everyone

- Searching algorithms are key to understanding many data type.
  - I.e. sets and maps/dicts.
- Key to writing efficent code.
- Key to understanding memory/processor trade offs.

Recap

- Searching
  - Applications everywhere.
- Linear search.
  - Simple.
  - Slow.
- Binary search.
  - Ordered sequence.
  - Very fast.
  - Divide & Conquer.
- String searching.
  - Finding subsequence in sequence.
  - Boyers-Moore.
  - Preprocessing.
  - Skipping sections.

The End