

122COM: Searching

David Croft

Coventry University

david.croft@coventry.ac.uk

2015

Overview

- 1 Introduction
- 2 Linear search
- 3 Binary search
- 4 String searching
- 5 Recap

Introduction

Searching is used everywhere in computing.

- Obvious applications.
 - Text files.
 - Databases.
 - File systems.
- Hidden applications.
 - Computer games.
 - FOV search for objects in view.

- Path finding algorithms in games
 - <https://www.youtube.com/watch?v=19h1g22hby8>
- Brute force approaches that find the best/shortest/fastest problem are too slow (travelling salesman).
- Heuristic approaches are used instead.
 - Find "good enough" solutions.
 - Not always the best solution.
 - Dijkstra's algorithm.
 - A* algorithm.

Simplest search.

- Also called sequential search.
- Iterate over elements.
- Until found or until end of sequence.
- Potentially slow.

● $O(n)$

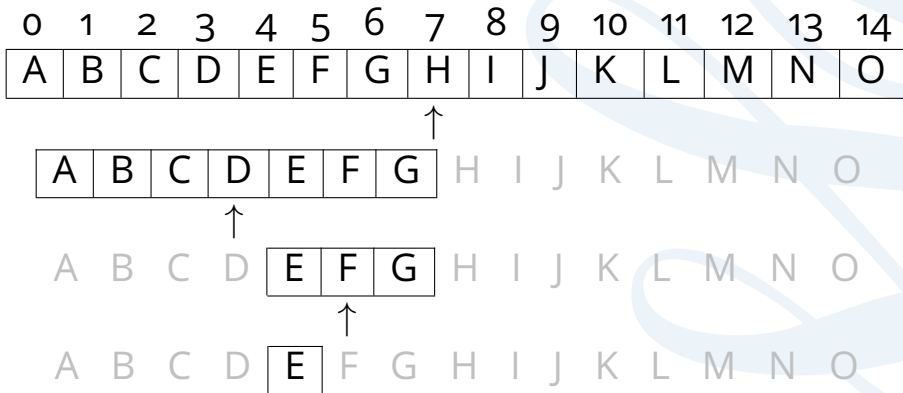
- Will discuss $O()$ notation in a later week.

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	Z	Z												
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Muuuuuuch faster than linear search.

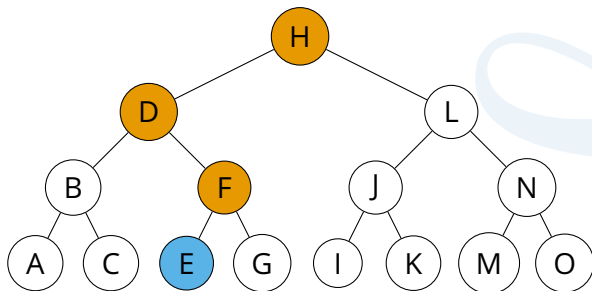
- Divide & conquer.
- Only works on sorted sequences.
- Algorithm is:
 - 1 Find middle value of sequence.
 - 2 If search value == middle value then success.
 - 3 If search value is < middle value then forget about the top half of the sequence.
 - 4 If search value is > middle value then forget about the bottom half of the sequence.
 - 5 Repeat from step 1 until `len(sequence)==0`.

Find E.



How many comparisons do we need to do for binary search?

- How many times can we divide our list by 2?
- No more than $1 + \log_2(n)$
 - $n = 14$.
 - $\log_2(14) = 3.9 \Rightarrow 3$
 - $1 + 3 = 4$
- Binary search has a complexity of $O(\log n)$.
 - Will cover $O()$ complexity in later week.
- Find E.



It's HOW much faster?!?!

Clearly much faster than linear search.

- To search a trillion elements linearly could mean a trillion comparisons.
- 40 with binary search.

But...

- Have to sort the list first.
- Sorting lists can be expensive.
- Can't always sort sequences.
- Ordering is important.
- Can't always search for sequences.
 - Text documents.
 - Genetic codes.

I.e. Text searching.

- Finding one sequence in another sequence.
- Naive search.
 - Like linear search.
 - 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.

Boyer-Moore string searching algorithm.

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

example \Rightarrow

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

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.
- 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 string searching algorithm.

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

Quiz

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

The End