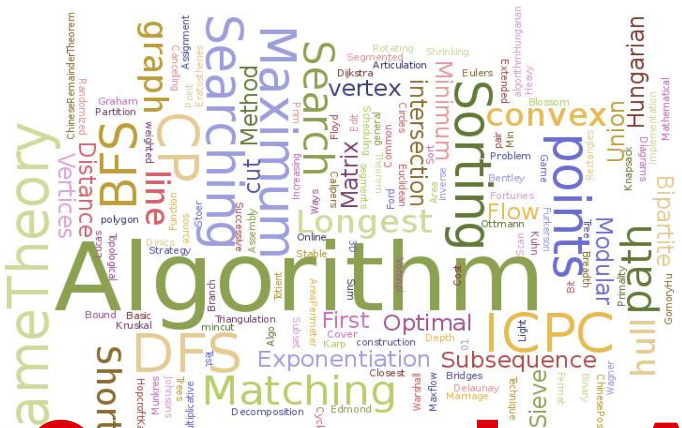
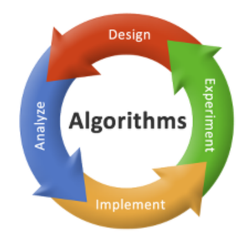


Search Algorithms



Course: Algorithms

Faculty: Dr. Rajendra Prasath

Autumn 2018

Search Algorithms

This lecture covers different search algorithms for searching an element in a given collection of elements. We provide illustrations and the complexity analysis of several search algorithms. We also discuss the criteria for choosing a better search algorithm for the problem in hand

Recap: Sorting Algorithms

- Suggest a simple algorithm for Sorting n elements
  - **Correctness:** First Test whether will the algorithm work for a small set of the input? Then apply on a bigger set
  - **Choice of the Data Structures:**
    - Choose a suitable data structure
  - **Perform complexity analysis**
    - How much space and running time required in the worst case?
  - **Adaptability:**
    - Is the solution adaptable with the growing size of the input n?
- How to get the tight bound of the sorting algorithm in terms of the running time?

Recap: Sorting Algorithms

- choosing a suitable sorting algorithm for a given problem??
  - How do you choose a specific sorting algorithm for any given problem?
  - Why do you choose that specific algorithm for that specific problem

- Examples:
  - Only a few items - Insertion Sort
  - Items are mostly sorted already - Insertion Sort
  - Concerned about worst-case scenarios - Heap Sort
  - Interested in a good average-case - Quicksort
  - Items are drawn from a dense universe - Bucket Sort

Choosing Right Sorting Algo

- **Size (Volume) of the Data:**
  - How much data are you expecting to sort?
  - look for an algorithm with a very low time complexity
  - Nature of the Data: Partly sorted or Random ?
    - Can affect the time complexity
    - Time complexity is not the same as running time
- **Running Time** of the algorithm:
  - Time complexity is not the same as running time
  - time complexity describes how the performance of an algorithm varies as the size of the data set increases
  - An algorithm does one pass ➔ O(n) - linear in n
  - An algorithm does two passes ➔ O(n) - still linear in n
- So the **asymptotic time complexity** and the **running time** of an algorithm are different

Searching for Info

- **Look at the following table:**

1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
Q1	3R	3R	4R	4R	W	SF	W	W	SF	F	W	SF	SF	SF	SF	3R	SF	W	W
1R	4R	QF	1R	1R	3R	SF	F	F	F	W	QF	F	SF	QF	4R	QF	A	A	A
1R	1R	QF	1R	W	W	W	W	W	F	W	QF	QF	W	2R	F	F	SF	W	QF
Q2	3R	4R	4R	4R	W	W	W	W	W	F	SF	SF	QF	4R	SF	F	A	QF	4R

- **Any more Details??**

Tournament	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	SR	W–L	Win %
Australian Open	Q1	3R	3R	4R	4R	W	SF	W	W	SF	F	W	SF	SF	SF	SF	3R	SF	W	W	6 / 19	94–13	87.9
French Open	1R	4R	QF	1R	1R	3R	SF	F	F	F	W	QF	F	SF	QF	4R	QF	A	A	A	1 / 17	65–16	80.2
Wimbledon	1R	1R	QF	1R	W	W	W	W	W	F	W	QF	QF	W	2R	F	F	SF	W	QF	8 / 20	95–12	88.7
US Open	Q2	3R	4R	4R	4R	W	W	W	W	W	F	SF	SF	QF	4R	SF	F	A	QF	4R	5 / 18	85–13	86.7
Win–Loss	0–2	7–4	13–4	6–4	13–3	22–1	24–2	27–1	26–1	24–3	26–2	20–3	20–4	19–3	13–4	19–4	18–4	10–2	18–1	14–2	20 / 74	339–54	86.3

- **What these tables are taking about??**
  - **Any Guessing or Search for information**

Grand Slam tournament finals: 30 (20 titles, 10 runner-ups)

Result	Year	Tournament	Surface	Opponent	Score
Win	2003	Wimbledon	Grass	Mark Philippoussis	7–6$^{(7–5)}$, 6–2, 7–6$^{(7–3)}$
Win	2004	Australian Open	Hard	Marat Safin	7–6$^{(7–3)}$, 6–4, 6–2
Win	2004	Wimbledon (2)	Grass	Andy Roddick	4–6, 7–5, 7–6$^{(7–3)}$, 6–4
Win	2004	US Open	Hard	Lleyton Hewitt	6–0, 7–6$^{(7–3)}$, 6–0
Win	2005	Wimbledon (3)	Grass	Andy Roddick	6–2, 7–6$^{(7–2)}$, 6–4
Win	2005	US Open (2)	Hard	Andre Agassi	6–3, 2–6, 7–6$^{(7–1)}$, 6–1
Win	2006	Australian Open (2)	Hard	Marcos Baghdatis	5–7, 7–5, 6–0, 6–2
Loss	2006	French Open	Clay	Rafael Nadal	6–1, 1–6, 4–6, 6–7$^{(4–7)}$
Win	2006	Wimbledon (4)	Grass	Rafael Nadal	6–0, 7–6$^{(7–5)}$, 6–7$^{(2–7)}$, 6–3
Win	2006	US Open (3)	Hard	Andy Roddick	6–2, 4–6, 7–5, 6–1
Win	2007	Australian Open (3)	Hard	Fernando González	7–6$^{(7–2)}$, 6–4, 6–4
Loss	2007	French Open	Clay	Rafael Nadal	3–6, 6–4, 3–6, 4–6
Win	2007	Wimbledon (5)	Grass	Rafael Nadal	7–6$^{(9–7)}$, 4–6, 7–6$^{(7–3)}$, 2–6, 6–2
Win	2007	US Open (4)	Hard	Novak Djokovic	7–6$^{(7–4)}$, 7–6$^{(7–2)}$, 6–4
Loss	2008	French Open	Clay	Rafael Nadal	1–6, 3–6, 0–6
Loss	2008	Wimbledon	Grass	Rafael Nadal	4–6, 4–6, 7–6$^{(7–5)}$, 7–6$^{(10–8)}$, 7–9
Win	2008	US Open (5)	Hard	Andy Murray	6–2, 7–5, 6–2
Loss	2009	Australian Open	Hard	Rafael Nadal	5–7, 6–3, 6–7$^{(3–7)}$, 6–3, 2–6
Win	2009	French Open	Clay	Robin Söderling	6–1, 7–6$^{(7–1)}$, 6–4
Win	2009	Wimbledon (6)	Grass	Andy Roddick	5–7, 7–6$^{(8–6)}$, 7–6$^{(7–5)}$, 3–6, 16–14
Loss	2009	US Open	Hard	Juan Martín del Potro	6–3, 6–7$^{(5–7)}$, 6–4, 6–7$^{(4–7)}$, 2–6
Win	2010	Australian Open (4)	Hard	Andy Murray	6–3, 6–4, 7–6$^{(13–11)}$
Loss	2011	French Open	Clay	Rafael Nadal	5–7, 6–7$^{(3–7)}$, 7–5, 1–6
Win	2012	Wimbledon (7)	Grass	Andy Murray	4–6, 7–5, 6–3, 6–4
Loss	2014	Wimbledon	Grass	Novak Djokovic	7–6$^{(9–7)}$, 4–6, 6–7$^{(4–7)}$, 7–5, 4–6
Loss	2015	Wimbledon	Grass	Novak Djokovic	6–7$^{(1–7)}$, 7–6$^{(12–10)}$, 4–6, 3–6
Loss	2015	US Open	Hard	Novak Djokovic	4–6, 7–5, 4–6, 4–6
Win	2017	Australian Open (5)	Hard	Rafael Nadal	6–4, 3–6, 6–1, 3–6, 6–3
Win	2017	Wimbledon (8)	Grass	Marin Čilić	6–3, 6–1, 6–4
Win	2018	Australian Open (6)	Hard	Marin Čilić	6–2, 6–7$^{(5–7)}$, 6–3, 3–6, 6–1

Information

- Swiss Tennis Star:
  Roger Federer
- How many Grand Slams won by him?
  ANS: 20
- What are they? Can you Guess?
  - 4 different grand slams (AO: 6; W: 8; RG: 1; US: 5)
  - Completed Career Grand Slam?
    - Yes / No
- How many ATP Masters won by him?
  - What are they?
- Let us search for information

M3: Searching Algorithms

- In this module, we focus on developing efficient search algorithms:
  - Overview of Search algorithms
  - Sequential Search
  - Binary Search
  - Hash based Search
  - Binary Tree Search
  - Scalable Searching Algorithms
  - Efficient approaches in Searching
  - Take Home assignments

M3: Searching Algorithms

- Given: A collection C of elements
- Existence:
  - Does C contain a target element x?
- Response:
  - Yes (exists in the collection) / No (does not exist)
- Need for Search Algorithms??
- How to devise algorithms that could give the answer with less response time??

Searching – An Overview

- Given: A collection C of elements
- Task: Search for the element x

- **Existence:**
  - Does C contain a target element x?
  - Response: Yes (exists) / No (does not exist)
- **Retrieval:**
  - Give your roll number and get all details?
  - Search anything in Google – Scalable searches
- **Associate Look up:**
  - Information Associated with the key x

Two Variations

- In this lecture, we will focus on two search algorithms:
- Sequential Search
  - Linear Search algorithm
- Binary Search Algorithm
  - Tree based search algorithm
- Complexity analysis

Algorithm - 1

Procedure **Search**(C, x)

Input: Collection C and an element x
Output: true (exists) or false (does not exist)

```
begin
    for i = 0 to n-1 do
        if C[i] = x then
            return true
    endfor
    return false
end
```

Algorithm - 2

Procedure **Search**(C, x)

Input: Collection C and an element x
Output: true (exists) or false (does not exist)

```
begin
    iter = c.start()
    while (iter != C.end()) do
        curr = iter.next
        if  curr = x then
            return true
    endwhile
    return false
end
```

Algorithm - 2

Procedure **Search**(C, x)

Input: Collection C and an element x
Output: true (exists) or false (does not exist)

```
begin
    iter = c.start()
    while (iter != C.end()) do
        curr = iter.next
        if  curr = x then
            return true
    endwhile
    return false
end
```

Linear Search - Facts

- For smaller collection, it is simple and efficient
- Can apply simple variations to traverse the collection organized in lists.

- Type of elements in the collection C
  - Does the collection require to follow any ordering?
  - If the given collection is sorted then what additional benefits you could get?
  - Can you reduce the worst case complexity?
    - At least in terms of the running time of the algorithm
    - Can you do any refinement during the implementation?

Linear Search - Analysis

- Input size: n elements in the collection C and x has to be search for
- Best Case:
  - O(1)
- Average Case:
  - O(n)
- Worst Case:
  - O(n)
- Data Structures:
  - Arrays, Linked Lists
  - Priority Queues ??

Binary Search

- Elements are arranged in a tree like structure
  - Based on Divide and conquer approach
  - Elements have to be organized according to the given partial order
  - Example:
    - Look at Phone Contacts
    - Yellow Pages (Telephone Directory)

Binary Search

```
Procedure binarySearch(C, x)

begin
    low = 0
    high = n-1;
    while (low ≤ high) do
        ivalue = (low + high) / 2
        if (x = C[ivalue]) then
            return true
        else
            if (x < C[ivalue]) then
                high = ivalue – 1
            else low = ivalue + 1
    endwhile
    return false
end
```

Binary Search – An Example

1	2	3	4	7	8	9	15	17	23	24
low					ivalue					High

Search(C, 15)

1	2	3	4	7	8	9	15	17	23	24
						low		ivalue		High

. . .

							ivalue			
1	2	3	4	7	8	9	15	17	23	24
							low High			

Binary Search - Facts

- Small amount of complexity for large gains
- Complexity can increase when collection is not stored in memory
- If the collection is kept in the secondary storage like disk or external drives, it may worsen the running time
  - Why??
    - File I/O operations
    - Cost needed to search an element will be dominated by the cost to access the storage.

Binary Search - Analysis

- Input size: n elements in the collection C and x has to be search for
- Best Case:
  - O(1)
- Average Case:
  - O(log n)
- Worst Case:
  - O(log n)
- Data Structures:
  - Arrays

Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

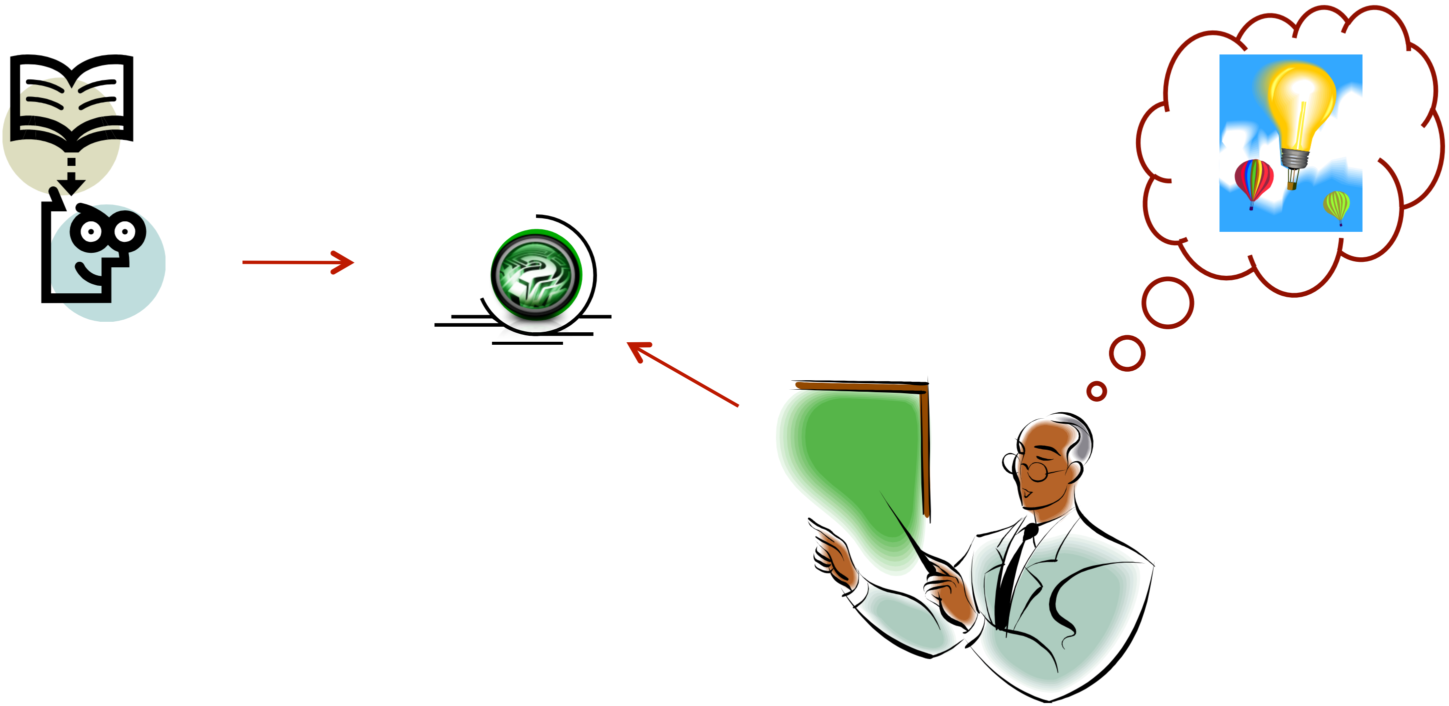
- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Assistance

- You may post your questions to me at any time
- You may meet me in person on available time or with an appointment
- TA s would assist you to clear your doubts.
- You may leave me an email any time (email is the best way to reach me faster)

Thanks …



… Questions ???