CSE-213
(Data Structure)

Lecture on

Linear and Binary Searching

Md. Jalal Uddin

Lecturer

Department of CSE

City University

Email: jalalruice@gmail.com

No: 01717011128 (Emergency Call)

Searching

- ✓ Searching refers to the operation of finding the location LOC of ITEM in Data, or printing some message that ITEM does not appear there.
- ✓ The search is said to be *successful* if ITEM does appear in Data and *unsuccessful* otherwise.
- ✓ There are many different searching algorithms. The algorithm that one chooses generally depends on the way the information is DATA is organized.
- ✓ A simple searching algorithm: Linear Search Algorithm
- ✓ The well known algorithm: Binary search Algorithm

Linear Search Algorithm

- ✓ Suppose DATA is a linear array with n elements. Given no other information about DATA.
- ✓ Simple way to search for a given ITEM in DATA is to compare ITEM with each element of DATA one by one.
- ✓ Suppose we want to know whether Jhon appears in the array or not.
- ✓ Again, Suppose, we want to know wheter Moon appears in the array or not.

Adams
Charlie
Rasha
Moon
Rock
Smith

Adams
Charlie
Rasha
Moon
Rock
Smith
Jhon

Adams
Charlie
Rasha
Moon
Rock
Smith
Moon

Linear Search Algorithm: Example

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

search element **12**

Step 1:

search element (12) is compared with first element (65)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

12

Both are not matching. So move to next element

Step 2:

search element (12) is compared with next element (20)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

12

Both are not matching. So move to next element

Step 3:

search element (12) is compared with next element (10)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

12

Both are not matching. So move to next element

Step 4:

search element (12) is compared with next element (55)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

12

Both are not matching. So move to next element

Step 5:

search element (12) is compared with next element (32)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

12

Both are not matching. So move to next element

Step 6:

search element (12) is compared with next element (12)

	0	1	2	3	4	5	6	7
list	65	20	10	55	32	12	50	99

12

Both are matching. So we stop comparing and display element found at index 5.

Searching: Linear Search Algorithm

LINEAR (DATA, N, ITEN, LOC)

Step 1. [Insert ITEM at the end of DATA] Set DATA[N+1]:=ITEM

Step 2. [Initialize counter] Set LOC:=1.

Step 3. [Search for ITEM.]

Repeat while Data [LOC]$\neq ITEM$:

Set LOC:=LOC+1.

[End of loop.]

Step 4. [Successful?] IF LOC=N+1, then: Set LOC:=0;

Step 5. Exit.

Linear Search Algorithm: Complexity

- ✓ The complexity of this algorithm is measured by the number f(n) of comparison required to find ITEM where DATA contains n elements.
- ✓ Two important cases to consider are:
  - ✓ The average case, and
  - ✓ The worst case

The worst case occurs when one must search through the entire array DATA, when ITEM does not appear in DATA.

Algorithm Requires f(n)= $n+1$ comparison.

Thus, in the worst case, the running time is proportional to n

Linear Search Algorithm: Complexity

- ✓ The complexity of this algorithm is measured by the number f(n) of comparison required to find ITEM where DATA contains n elements.
- ✓ Two important cases to consider are:
  - ✓ The average case, and
  - ✓ The worst case

The running time of the average case uses the probabilistic notation of expectation.

- ✓ Suppose, p_k is the probability that ITEM appears in DATA[K], and
- ✓ suppose, q is the probability that ITEM does not appear in DATA.
- ✓ Since, the algorithm uses k comparisons when ITEM appears in DATA [K], the average number of comparisons is given by

$$f(n)= 1.p_1+2.p_2+\ldots\ldots.n.p_n+(n+1)q$$

- ✓ In particular, q is very small, and ITEM appears with equl probability in each element of DATA. Then $q \approx 0$ and each $p_i=1/n$.
- ✓ Accordingly $f(n) = 1.\frac{1}{n} + 2.\frac{1}{n} + 3.\frac{1}{n} + \ldots\ldots \text{n}.\frac{1}{n} + (n+1).0$
- ✓ $= (1+2+\ldots+\text{n}).\frac{1}{n} = \frac{n(n+1)}{2}.\frac{1}{n} = \frac{n+1}{2}$

Binary Search:

Suppose DATA is an array which is sorted in ***INCREASING ORDER***, or equivalently, alphabetically.

Then, *Binary Search algorithm* is an extremely efficient searching algorithm to find the location LOC of a given ITEM of information in DATA.

Binary search algorithm applied to array DATA works as follows:

DATA[BEG], DATA[BEG+1], DATA[BEG+2],........, DATA[END]

This algorithm compares ITEM with the middle element DATA [MID] of the segment, where MID is obtained by

MID=INT((BEG+END)/2)

- ✓ If DATA[MID]=ITEM, then the search is **SUCCESSFUL**.

  We set LOC:=MID

  ....Otherwise a new segment of DATA is obtained.

Binary Search:

(a) If ITEM < DATA[MID], then ITEM can appear only in the left half of the segment:

DATA[BEG], DATA[BEG+1], ….DATA[MID-1]

So, we resent END:=MID-1 and begin searching again.

(b) If ITEM> DATA[MID], then ITEM appear only in the right half of the segment:

DATA[MID+1], DATA[MID+2],……….DATA[END]

So, we reset BEG:=MID+1 and begin searching.

✓ Initially, we begin with entire array DATA, i.e. We begin wit BEG=1 and END=n, or more generally, with BEG=LB and END=UB.

✓ If ITEM is not in DATA, then eventually we obtain

END< BEG

Which means the search is **Unsuccessful**

So, SET LOC:=NULL (OUT side of DATA indices)

Binary Search: : Example

Searching the array below for the value **42**:

											10						
	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

min

mid

max

Binary Search: Algorithm

BINARY (DATA, LB, UB, ITEM, LOC)

(This algorithm finds the location LOC of item in DATA or sets LOC:=NULL)

1. [Initialize segment variables.]
   Set BEG:=LB, END:=UB, and MID=INT((BEG+END)/2).
2. Repeat Steps 3 and 4 while BEG<=END and DATA[MID] $\neq$ ITEM
3. If ITEM<DATA[MID], then
   Set END:=MID-1.
   Else:
   Set BEG:=MID+1 [End of If structure.]
4. Set MID:=INT((BEG+END)/2
   [End of Step 2. loop]
5. If DATA[MID]=ITEM, then:
   Set LOC:=MID
   Else:
   Set LOC:=NULL. [End of If structure]
6. Exit

LINEAR SEARCH
VERSUS
BINARY SEARCH

LINEAR SEARCH	BINARY SEARCH
An algorithm to find an element in a list by sequentially checking the elements of the list until finding the matching element	An algorithm that finds the position of a target value within a sorted array
Also called sequential search	Also called half-interval search and logarithmic search
Time complexity is O(N)	Time complexity is O(log2N)
Best case is to find the element in the first position	Best case is to find the element in the middle position
It is not required to sort the array before searching the element	It is necessary to sort the array before searching the element
Less efficient	More efficient
Less complex	More complex

Visit www.PEDIAA.com

Binary Search: Limitations

Your Task