Lecture 19

Searching

October 26, 2021

Tuesday

SEARCHING

- For each particular structure used to hold data, the functions that allow access to elements in the structure must be defined.
    - Such as only top element can be accessed.
        - LIFO ordered structure.
    - Only front or bottom element can be accessed.
        - FIFO ordered structure.
    - Sometimes when we already know the location of the elements, we can access directly.
        - 5th element of the array.

SEARCHING

- What if we don’t know the position of the element in the structure.
- What if we don’t know if the required element exist or not in the structure.
- In this particular situation we have to search the element in the structure.
    - We require a key, to identify the element in the structure.
    - We refer to the unique features of the element as keys, which distinguishes one element from another.

SEARCHING FUNCTION FEATURES

Function: Determine whether an item in the list has a key that matches element's key

Precondition: List has been initialized. Items keys have been initialized.

Postcondition: location = position of elements whose key matches item's key, if it exists; otherwise, location = NULL

SEARCHING

- These specifications applies to both array-based and linked-lists
  - In case of array, location would be the index
  - -1 in case, item does not exist in the array.
  - In case of linked-list, location would be the pointer to that node
  - NULL pointer in case, item does not exist in the linked-list.

SEARCHING TYPES

- There are different ways we can search our data
    - Linear Search
    - Binary Search
    - Jump Search
    - Interpolation Search
    - Exponential Search

Linear Search

Linear Search

- When the data is not sorted, or the storage medium does not allow direct access.
    - Linked list, magnetic tape, where data may or maynot be ordered.
- The simplest way to search.
- The element to be found is sequentially searched in the list
- The method will work with both, sorted & unsorted data
- The search continues until the target element is found or the list ends.

Linear Search

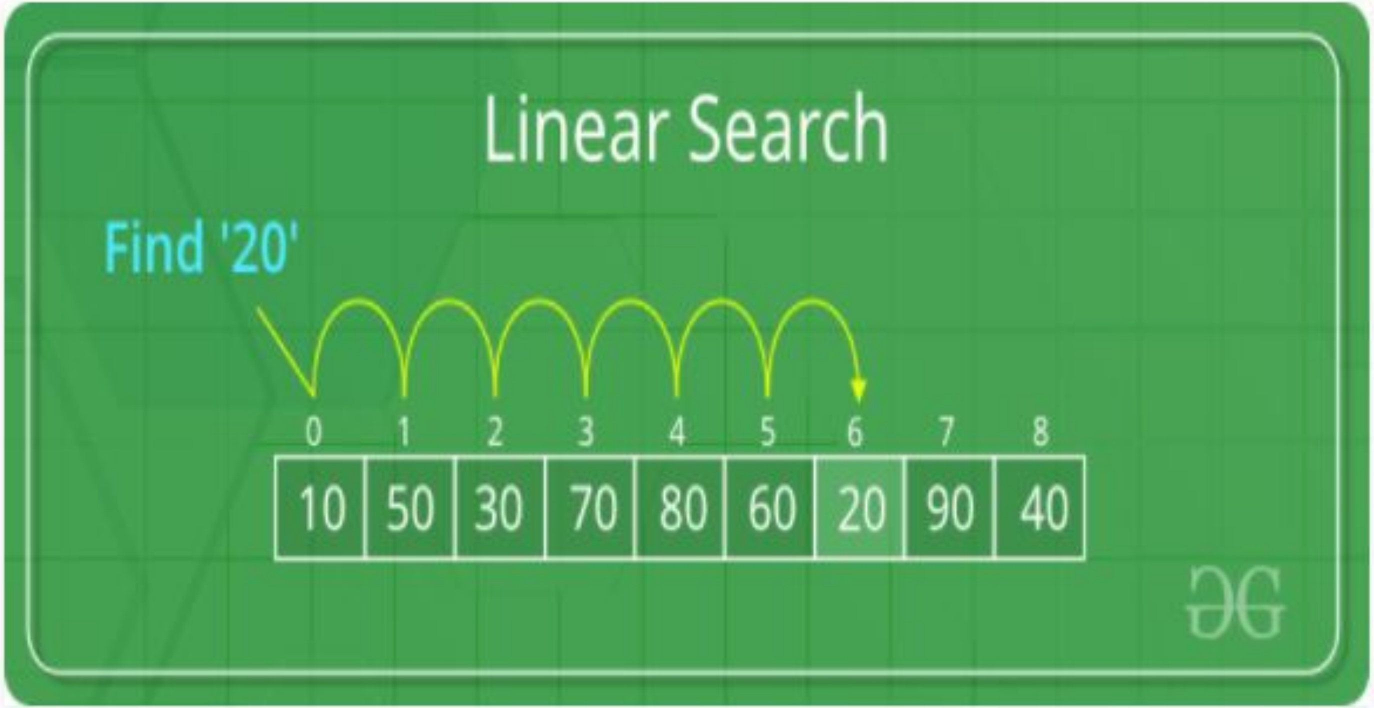

Image Taken from Geeks Of Geeks

PSEUDOCODE

```
LinearSearch ( unsorted data )

Initialize location to position of first item
Set found to false
Set moreToSearch to (have not examined Info ( last ) )

while moreToSearch AND NOT found
        if item equals Info ( location )
                Set found to true;
        else
                Set location to Next ( location )
                Set moreToSearch to (have not examined Info (last ) )

if NOT found
        Set location to NULL
```

Linear Search Time Complexity

- Based on the number of comparisons
  - Worst Case we compare key to n-elements resulting in *O (n).*
  - For average case we may compare half of the elements resulting in *O(n/2),* dropping the constant *O (n).*
  - In best case scenario the first item is the target and we only make one comparison, *O (1).*

Linear Search Class Activity

- Write a function which searches for a given item in linear time
    - In an Array [ ].
    - In a Linked List.

Jump Search

Jump Search

- What if NADRA was using Linear Search to find a given CNIC number.
  - Sequential search is extremely slow when massive data is search repeatedly.
- One way to reduce search time is to preprocess the data by sorting them.
- An obvious improvement over simple linear search is to check if the current element in data is greater than the key or not?
  - If it is, we know that the key cannot occur later in the data.
  - Does this improves over worst case cost of the algorithm.

Jump Search

- If we look at position 1 of Sorted array and find that key is greater.
    - Then we can rule out position 0 as well as position 1.
- Similarly, if we look at position 2 of Sorted array and find that key is greater.
    - Then we can rule out position 0, 1 and 2 with one comparison.
- What if we carry this to extreme?
    - Comparing the last element with the key?
    - If key is still greater than what does it mean?

Jump Search

- Shall we always start by looking at the last position.
    - What we learn a lot sometimes
    - Usually we learn a little bit (last element is not equal to key).
- Then we have to ask ourselves, what is the proper size of Jump?
    - This idea leads to the Jump Search
    - For some value j, we check every data [j], data [j + j] and so on.
    - As long as key is greater than K.
    - When we reach a value in data greater than key.
    - We perform a linear search on the piece of the length j - 1, with the guarantee that K must be in the interval $mj \leq n \leq (m + 1) j$

Jump Search

- Then the total cost of this algorithm is at most $m + j - 1$, *3* way comparison.
  - 3 way comparison, as we want to know
    - data [j] == key
    - data [j] > key
    - data [j] < key
- Solving for the minimum value T (n, j) = m + j - 1 = ⌊n / j⌋ + j - 1.
  - $j = \sqrt{n}$
- Follows Divide & Conquer strategy.

```
int JumpSearch ( int data [ ], int value, int n) {

        int step = sqrt ( n );
        int prev = 0;
        while (data [ min (step, n) - 1] < x ) {
                prev = step;
                step += sqrt ( n )
                if (prev >= n)
                        return -1;
        }
        while (data [prev ] < x) {
                prev ++;
                if (prev == min( step, n ) )
                        return -1;
        if (data [ prev ]  == x )
                return prev;
        return - 1;
}
```

TIME COMPLEXITY

- Time complexity
  - Best Case: $O(1)$
  - Average Case: $O(\sqrt{n})$
  - *Worst Case:* $O(\sqrt{n})$
  - Space Complexity $O(1)$

Binary Search

BINARY SEARCH

- What if NADRA was using Linear Search to find a given CNIC number.
  - Sequential search is extremely slow when massive data is search repeatedly.
- One way to reduce search time is to preprocess the data by sorting them.
- If the data elements are sorted and stored in an array sequentially.
  - We can use Binary Search to find a particular element
- The binary search improves the efficiency of search by limiting the search to the area where the element might be.
- Uses Divide & Conquer strategy.

BINARY SEARCH

- Looking for the name Dawood in the phone book.

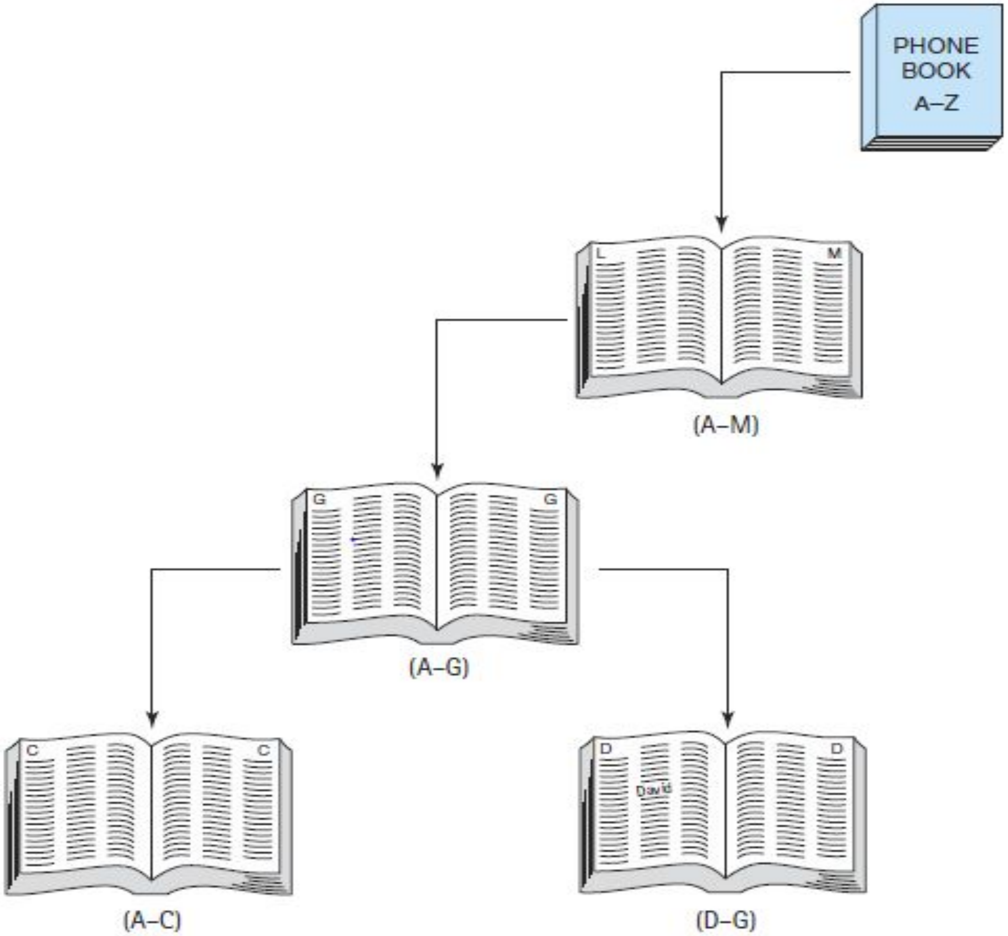

Figure 3.6 *A binary search of the phone book*

BINARY SEARCH

- We compare the key with the center element of the array.
  - If it matches, search is successful return the index.
- Otherwise, the list is divided into two halves
  - One half from 0 to the center, containing the elements less than the center value.
  - Second half from center to the last index, containing the elements greater than the center value.

BINARY SEARCH

- The searching will start in either of the two halves depending upon
  - If the key is less than the center element than search will continue in the left subarray.
  - else the search will continue in right subarray.
- Same process will be repeated in the subarray
  - Key will be compared with the center of the subarray
  - If not found, new subarrays will be created.

RECURSIVE IMPLEMENTATION

```
BinarySearch (Array [ ], int low, int high, int value)
    if ( low < high) {
        int mid = ( low + high ) / 2;
        if ( Array [ mid ] == value )
            return mid;
        if ( value < Array [ mid ] )
            return BinarySearch (Array, low, mid - 1, value);
        if ( value > Array [ mid ] )
            return BinarySearch (Array, mid + 1, high, value);
    }
    return -1;
}
```

BINARY SEARCH EXAMPLE

Binary Search Time Complexity

- Based on the number of comparisons
    - Worst Case we call divide problem in subarrays until we can no longer divide it into subarrays, single item is accessed.
        - *O (log n).*
    - For average case dropping some factors we will still have
        - *O (log n).*
    - In best case scenario the first center item is the target and we only make one comparison, *O (1).*

BINARY SEARCH LIMITATION

- The binary search is not guaranteed to be faster for searching very small arrays.
  - Even though the binary search generally requires fewer comparisons,
    - Each comparison requires more computation
- Binary search maynot be efficient on linked lists
  - How can you efficiently find the mid-node of the linked list