LINEAR SEARCH IN DETAIL

Linear search or Sequential search is a method for finding a particular value in a linear list of values (Array or Linked List). It is done by checking each of its elements, one at a time in list, until the desired one is found or the list is exhausted.

Problem definition (for array)

Given an Array of numbers a_0, a_1, a_2, …, a_n. and a number x. Write a function which will return i, such that (a_i == x). If x is not in the Array, then return -1 (which means "Not Found").

For Example: If Array is as given below and x = 5 (element to be searched)

16	7	10	5	1	8	3	4	7	2

Then the function should return 3 (because 5 is at index 3)

The signature of function is:

```
1    int search(int *arr, int n, int x);
```

Algorithm:

Search for each element in the array starting from 0, 1, 2, … and continue till either arr[i] == x or the array is exhausted.

Code:

```
1   /** Function searches data in an array arr of size n.
2    * If data is present in arr, it returns index of first occurrence in array.
3    * Else returns -1 to indicate that data is not found in array.
4    */
5   int linearSearch(int * arr, int n, int data)
6   {
7       for(int i=0; i<n; i++)
8       {
9           if(arr[i] == data)
10              return i;
11      }
12      return -1;
13  }
```

Time Complexity:

Best case: O(1) – When element is found at the first position (it requires only one check)

Worst Case: O(n) – If element is not fount in the list. Then function will check all the n elements.

Average Case: O(n) – requires n/2 check.

Linear Search in Linked List

We cannot search in the linked list using Binary search because element at index i (arr[i]) cannot be found using constant time. Hence, linear search is the only option to search an element in a linked list.

```
1    int search(Node* head, int x)
2    {
3        int nodeNum = 0;
4        while(head != NULL)
5        {
6            if(head->data == x)
7                return nodeNum;
8            nodeNum++;
9            head = head->next;
10       }
11       return -1;
12   }
```

Variations:

Following are the variations of linear search on arrays and linked list.

---

1. Search for first occurrence of element from end of the list

For array this is easy, we just need to modify the for loop to check from the last element till the first element. The loop

```
1    for(int i=0; i<n; i++)
```

Will be replaced with

```
1    for(int i=n-1; i>=0; i--)
```

and rest everything will remain the same.

For linked list, it will be difficult because there is no way to back track in a singly linked list (can't move in the back ward direction). One way to do this is to

- reverse the list
- search in the reversed list
- reverse the list against

Another way is by using tail-recursion.

```
1    int search(Node* head, int x, int index=0)
2    {
3        if(head == NULL)
4            return -1;
5
6        int retVal = search(head->next, x, index+1);
7
```

```
8           if( retVal == -1)
9           {
10              if(head->data == x)
11                  return index;
12          }
13          return retVal;
14      }
```

---

2. Printing indexes of all occurrences of data

Print all the positions where x is present in the array. For example, if we are searching for 5 in the below array:

```
2 4 5 1 5 6 7 5 9 0 1
```

Then the output should be all the positions where 5 is present (2 is at position 0)

```
2 4 7
```

There is no best case for this variation, because we have to traverse the entire array (or linked list) in any case.

```
1    // Searching for all occurrences in forward direction:
2    for(int i=0; i<n; i++)
3    {
4        if(arr[i] == x)
5            cout<<"Element Found at position: "<<i;
6    }
```

Searching in backward direction for all occurrences:

```
1    // Searching for all occurrences in forward direction:
2    for(int i=n-1; i>=0; i++)
3    {
4        if(arr[i] == x)
5            cout<<"Element Found at position: "<<i;
6    }
```

Similarly it can be done for linked list also (don't return the index, just print it and move forward).

Time complexity of all the functions above is O(n).

Source: http://www.ritambhara.in/linear-search-in-detail/