DATA STRUCTURE

All Notes Are Verified by NPTL and Published By

WWW.ASKTOHOW.COM

IIT Guwahati

NPTL

C.E.O of ASKTOHOW

Deepak

Retrieval of information from a database by any user is very trivial aspect of all the databases. To retrieve any data, first we have to locate it. This operation which finds the location of a given element in a list is called searching.

If the element to be searched is found, then the search is *successful* otherwise it is *unsuccessful* .

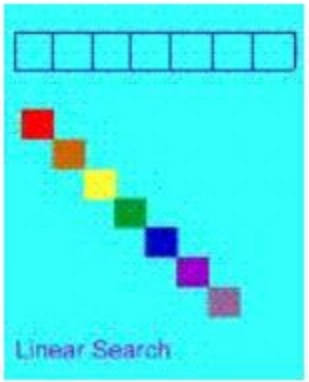
We will discuss different the searching methods. The two standard ones are :

1. Linear Search
2. Binary search

Linear search is the most simple of all searching techniques. It is also called sequential search. To find an element with key value='key',every element of the list is checked for key value='k' sequentially one by one.If such an element with key=k is found out, then the search is stopped. But if we eventually reach the end of the list & still the required element is not found then also we terminate the search as an unsuccessful one.

The linear search can be applied for both unsorted & sorted list.

- Linear search for Unsorted list
- Linear search for sorted list



In case of unsorted list, we have to search the entire list every time i.e.we have to keep on searching the list till we find the required element or we reach the end of the list.this is because as elements are not in any order,so any element can be found just anywhere.

Algorithm:

```
linear search(int x[],int n,int key)
{
  int i,flag = 0;
  for(i=0;i < n ; i++)
   {
   if(x[i]==key)
    {
    flag=1;
    break;
    }
   }
   if(flag==0)
   return(-1);
   else
  return(1);
}
```

Complexity:

The number of comparisons in this case is `n-1`.So it is of `o(n)`. The implementation af algo is simple but the efficiency is not good. Everytime we have to search the whole array(if the element with required value is not found out).

The efficiency of linear search can be increased if we take a previously sorted array say in ascending order. Now in this case, the basic algorithm remains the same as we have done in case of an unsorted array but the only difference is we do not have to search the entire array everytime. Whenever we encounter an element say y greater than the key to be searched, we conclude that there is no such element which is equal to the key, in the list. This is because all the elements in the list are in ascending order and all elements to the right of y will be greater or equal to y, ie greater than the key. So there is no point in continuing the search even if the end of the list has not been reached and the required element has not been found.

```
Linear search( int x[], int n, int key)
{
int i, flag=0;
for(i=0; i < n && x[i] <= key; i++)
{
if(x[i]==key)
{
flag=1;
break;
}
}
if(flag==1) /* Unsuccessful Search*/
return(-1);
else return(1); /*Successful search*/
}
```

Illustrative Explanation:

The array to be sorted is as follows:

21 35 41 65 72
It is sorted in ascending order. Now let key = 40. At first 21 is checked as x[0]=21.
It is smaller than 40. So next element is checked which is 35 that is also smaller than 40. So now 41 is checked.But 41 > 40 & all elements to the right of 41 are also greater than 40.So we terminate the search as an unsuccessful one and we may not have to search the entire list.

Complexity:

Searching is NOT more efficient when key is in present in the list in case when the search key value lies between the minimum and the maximum element in the list. The Complexity of linear search both in case of sorted and unsorted list is the same. The average complexity for linear search for sorted list is better than that in unsorted list since the search need not continue beyond an element with higher value than the search value.

The most efficient method of searching a sequential file is binary search. This method is applicable to elements of a sorted list only. In this method, to search an element we compare it with the center element of the list. If it matches, then the search is successful and it is terminated. But if it does not match, the list is divided into two halves. The first half consists of 0th element to the center element whereas the second list consists of the element next to the center element to the last element. Now It is obvious that all elements in first half will be < or = to the center element and all element elements in the second half will be > than the center element. If the element to be searched is greater than the center element then searching will be done in the second half, otherwise in the first half.

Same process of comparing the element to be searched with the center element & if not found then dividing the elements into two halves is repeated for the first or second half. This process is repeated till the required element is found or the division of half parts gives a single element.

Algorithm for Binary Search.

Illustrative Explanation :
Let the array to be sorted is as follows:

```
11 23 31 33 65 68 71 89 100
```

Now let the element to be searched ie `key = 31` At first `hi=8 low=0` so `mid=4` and `x[mid]= 65` is the center element but `65 > 31`. So now `hi = 4-1=3`. Now `mid= (0 + 3)/2 = 1`, so `x[mid]= 23 < 31`.So again `low= 1 + 1 = 2`. Now `mid = (3 + 2)/2 = 2` & `x[mid]= 31 = key`.So the search is successful. Similarly had the key been `32` it would have been an unsuccessful search.

Complexity:

This is highly efficient than linear search. Each comparision in the binary search reduces the no. of possible candidates by a factor of 2. So the maximum no. of key comparisions is equal to log(2,n) approx. So the complexity of binary search is O(log n).

Limitations:

Binary search algorithm can only be used if the list to be searched is in array form and not linked list. This is because the algorithm uses the fact that the indices of the array elements are consecutive integers. This makes this algorithm useless for lists with many insertions and deletions which can be implemented only when the list is in the form of a linked list.

But this can be overcome using *padded list* .

To use binary search in the presence of large no. of insertions & deletions we use a data structure known as padded list. Two arrays , an element array & a parallel flag array are used. The element array consists of the sorted keys in the table with empty slots initially evenly interspersed among the keys of the table to allow for growth. The empty slot is indicated by a 0 value in the corresponding flag array element and a full slot is indicated by the value 1. Each empty slot in the element array contains a key value > or = to the key value in the previous full slot & < than the key value in the following full slot. Thus the element array is sorted. So binary search can be done on it.

Search:

To search for an element, perform a binary search on the element array. If the argument key is not found, the element does not exist in the list. If it is found & corresponding flag is 1 , then the search is successful. If the flag is 0, check if the previous full slot contains the argument key. If yes then search is successful otherwise its an unsuccessful search.

Insertion:

To insert an element locate the position. If its empty insert the element there, reset the flag to 1 & adjust the contents of all previous contagious empty positions to equal the contents of the previous full element & of all the following contagious empty positions to the inserted element, leaving their flags at 0. If the position is full shift forward by 1 position all the following elements upto the first empty position to make place for new element.

Deletion:

Deleting simply involves locating a key and changing its associated flag to 0.

1. Write a program to find out a number in a given unsorted array. Input will be a random number; Output should locate its position.
2. Write a program to perform linear search in a given sorted array.
3. The maximum number of comparisons in binary search is limited to …….
4. Write down the case(s) when binary search can't be implemented.
5. What is the difference between internal & external sorting?
6. Arrange following sorting methods in the order of their running time, number of comparisons made & worst case complexity……

   Selection sort, Bubble sort, Quick sort, Insertion sort.

7. Implement quick sort.
8. What would be the worst case scenario for bubble sort program?
9. Write a program that sorts the elements of a two dimensional array
   - Row wise
   - Column wise
10. Suppose an array contains N elements. Given a number X that may occur several times in the array. Find
    - The number of occurrence of X in the array.
    - The position of first occurrence of X in the array.
11. WAP that determines the number of spaces " " in a given input & removes it. Input should be like STCGSS TGCCGT GTCCCTTSGTT GGSST GSGSSGGG output -> 4 STCGSSTGCCGTGTCCCTTSGTTGGSST

12. WAP that replaces all S's in the last program with H.
13. WAP that makes a dictionary type of listing from a given set of words. (Hint: Use 2-D array).
14. Take any five dates in dd\mm\yyyy format, and arrange them in ascending order. Give the outline of the work required.
15. Write a program that accepts a set of 5 records for students. Ask user to enter name, age and height of a student. Sort these records in ascending order of their names. If the names are alike then sort according to their age.