Binary Search In Linked List

Kumar Sanu

Abstract: ***This paper is based on an approach to implement Binary Search in Linked List. Binary Search is divide and conquer approach to search an element from the list of sorted element. In Linked List we can do binary search but it has time complexity O(n) that is same as what we have for linear search which makes Binary Search inefficient to use in Linked List.***

The main problem that binary search takes O(n) time in Linked List due to fact that in linked list we are not able to do indexing which led traversing of each element in Linked list take O(n) time.In this paper a method is implemented through which binary search can be done with time complexity of O(log$_2$n). This is done with the help of auxiliary array. Auxiliary array helps in indexing of linked list through which one can traverse a node in O(1) complexity hence reducing the complexity of binary search to O(log$_2$n) hence increasing efficiency of binary search in linked List.

Keywords: Array of pointers, Binary Search, Indexing, Linked List.

I. INTRODUCTION

Binary Search is divide and conquer approach to search an element from the list of sorted element. Divide and Conquer is an algorithm design technique in which a problem is divided into sub problems then each sub problem is solved and solutions of each sub problem is combined to get the actual solution of the problem. In Binary Search there is list of elements in sorted order and the list is then divided from the middle into left and right sub parts to search the desired element. If desired element value is greater than middle value element, then we go to right sub part of list otherwise we move to left sub part of list and this process continues till we get the desired element.

Linked list is a data structure which is used to store the collection of data. It has following properties.

1)Successive elements are connected by pointers.

2)The last element points to NULL.

3)Can grow and shrink in size

4)Memory efficient.

Linked list has an advantage over array that they can be expanded in constant time. To create an array, we must allocate memory for certain number of elements. To add more elements to array when full, we must create a new array and copy the old array into new array. This can take a lot of time. With a linked list, we can start with space for just one element and add on new elements easily without the need to do any copying and reallocating.

However, there is also a disadvantage in linked list that if the list is sorted still we can't apply binary search as it is meaningless because time complexity will be O(n) due to fact that we cannot do indexing in linked list hence we have to go with linear search also having O(n) time complexity while in array this can be done using Binary search with O(log$_2$n).

This paper focus on implementing Binary search in Linked List. This will be done by doing indexing in Linked List. The time complexity will be O(log$_2$n) and space complexity will be O(n) of this implementation.

Revised Manuscript Received on October 05, 2019

Kumar Sanu,

II. PROBLEM IDENTIFICATION

Linked List is a data structure used to store collection of data. The main advantage of Linked list is that it has dynamic memory allocation hence only that much memory is allocated as much required. However, in Linked list we are not able perform Binary Search hence leaving only option of doing Linear Search in Linked List. This paper will give an approach to implement Binary Search In linked list with time complexity O(log$_2$n). In Binary search we have to find middle element to divide the list into 2 parts and do searching on this. To access middle node in array it is taken O (1) constant time but to find middle element in Binary search it is very much difficult as it takes O(n) time. So first I had provided a method to find middle in Linked list in O(1) complexity.

III. LITERATURE SURVEY

Literature review was carried out throughout whole research to gain knowledge and skills needed to do this research. In paper [1] authors implemented Binary Search using dual pointers. In paper [2] author analyses Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array, Dynamic Array and Linked List. In paper [3] author tries to explain Binary Search algorithm. In paper [4] author compare linear search and binary search algorithms.

IV. ALGORITHM TO FIND MIDDLE IN LINKED LIST

To do indexing in Linked List we will make an auxiliary pointer array having size same as linked list which will store address of each corresponding node of linked list. In this way using the pointer array we can go to any node with O(1) Time complexity in Linked List.

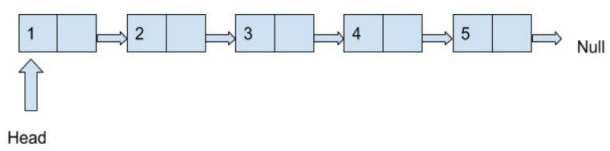

Fig 1-Linked List

Fig 2-Array of Pointers

Each entry in array will be corresponding address of each node of linked list.

V. SOURCE CODE TO ASSIGN ADDRESS OF NODES TO ARRAY

```
void assigning_address(struct node *head , int n)
{
   node *arr[n]; //node type array of pointer is declared.
   for(int i=0;i<n;i++)
   {
      arr[i]=head; //assigning each node address to array
      head=head->next;
   }
   for(int i=0;i<n;i++)
   {
      cout<<arr[i]<<endl;//display address of each node
   }
}
```

```
Enter number of nodes5

enter numbers1
2
3
4
5
0xd11258
0xd111b8
0xd11268
0xd11278
0xd11118

Process returned 0 (0x0)   execution time : 4.176 s
Press any key to continue.
```

Fig-3 Output showing address of each node Of linked list

VI. FINDING MIDDLE OF LINKED LIST

Middle of Linked list can be obtained easily using array of pointers. We only had to find middle of array and assign the middle value of array into a node type pointer.

```
//pseudo code to find middle of linked list
void binary_search(struct node *arr[],struct node *head,int n,int key)
{
 int beg=0, end=n-1;//First and last index of array
int mid= (beg+end)/2;//Finding middle of array
head=arr[mid];
cout<<head->data;
}
```

VII. BINARY SEARCH AFTER FINDING MIDDLE ELEMENT

```
While(beg<end)
 {
  if(head->data==key)//if current data is key value
  {
     status=true;
     break;
   }
else if(key>head->data)//if key value is large than current data
  {
     beg=beg+1;
   }
else if(key<head->data)//if key value is smaller than   current data
  {
    end=end-1;
  }
 mid=(beg+end)/2;//finding middle of array
 head=arr[mid];//finding middle of linked list
  }
 if(status)
  cout<<"Number found";
 else
  cout<<"Number not found";
```

VIII. DRY RUN OF CODE

Let initial linked list be like given in the fig-4. Data in linked list is in sorted order. Hence we can perform binary search in this linked list. In fig-5 array of pointers(arr) is shown in which each entry is address of each of corresponding nodes. Suppose key element is 2.

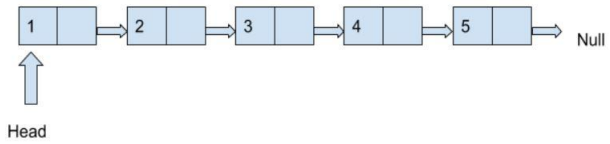

Fig-4 linked list with five nodes

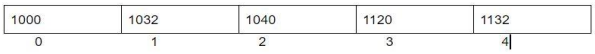

Fig-5 array of pointers

Initially beg=0, end=4.
Thus, mid=(0+4)/2=2
Thus, head=a[mid]
head=1040
head->data=3
This is shown in figure 6

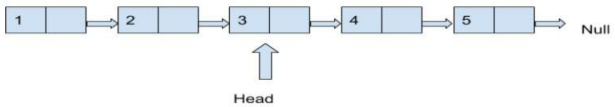

Fig-6 Head pointer pointing to third node of list.

Now, since head->data(3) is greater than key value(2).Hence there will be decrement in end index of array(arr).
Hence now beg=0, end=3
Thus mid=(0+3)/2=1
Thus, head=a[mid]
head=1032
head->data=2
This is shown in figure 7

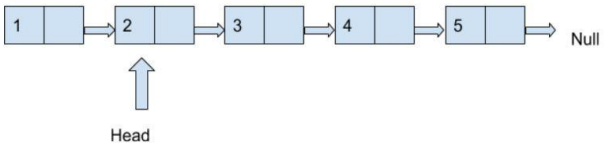

Fig-7 Head pointer pointing to second node of list.

Now, head->data=2 which is equal to key value which is 2. Hence key value is found in linked list.

IX. COMPLEXITY ANALYSIS

Time Complexity of this approach will be $O(\log_2 n)$ as linked list with reference of array is divided into two halves always till (beg<end) which is much better than O(n) complexity of binary search in linked list if we traverse the linked list one by one.

X. OUTPUT

```
Enter number of nodes5

enter numbers1
2
3
4
5
enter number to search4
Number found
Process returned 0 (0x0)   execution time : 8.298 s
Press any key to continue.
```

Fig-8 Binary search (Number found)

```
Enter number of nodes5

enter numbers1
2
3
4
5
enter number to search8
Number not found
Process returned 0 (0x0)   execution time : 7.650 s
Press any key to continue.
```

Fig-9 Binary search (Number not found)

XI. RESULT AND DISCUSSION

Since with this approach indexing get possible in linked list. Hence we are able to implement binary search with complexity $O(\log_2 n)$ which led searching of element to be fast when the linked list is sorted.

XII. CONCLUSION

The paper provides you an algorithm to do binary search in Linked list with time complexity of $O(\log_2 n)$ that is same as that of Binary Search time complexity in array. Hence with this approach one can do fast Binary search of an element if the elements are sorted in linked list. This will be much better approach if there is sorted large list as we do not have to do Linear Search which have complexity of O(n).

REFERENCES

1. Sreemana Datta, Parichay Bhattacharjee," Implementation of Binary Search on a Singly Linked List Using Dual Pointers" (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (2) , 2014.
2. Vimal P.Parmar, CK Kumbharana," Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array, Dynamic Array and Linked List" International Journal of Computer Applications (0975 – 8887) Volume 121 – No.3, July 2015
3. Ancy Oommen , Chanchal Pal," BINARY SEARCH ALGORITHM", IJIRT Volume 1 Issue 5| ISSN: 2349-6002
4. Morrice joseph,Palak keshwani,"Comparision between linear and binary search algorithms",IJARIIT (ISSN: 2454-132X)

AUTHORS PROFILE

Kumar Sanu is currently pursuing CSE with specialization in Internet of things from Chandigarh University. His area of interest is data structures and algorithms, machine leaning ,internet of things and software development .He is technically skilled in C,C++,Java, Python ,Machine learning, IOT and neural networks.