ISSN:0975-9646

Implementation of Binary Search on a Singly Linked List Using Dual Pointers

Sreemana Datta[1], Parichay Bhattacharjee[2]

[1] *School Of Education Technology (Master In Multimedia Development), Jadavpur University, India*

[2] *Department Of Computer Science & Engineering, Institute of Engineering & Management, India*

***Abstract*— To perform Binary Search based on Divide and Conquer Algorithm, determination middle element of a series of elements is necessary. But memory allocation for a singly linked list is dynamic and not contiguous. Hence, determination of middle element becomes difficult. This paper aims to provide an alternative approach using two different pointers to perform binary search on a singly linked list.**

***Keywords*— Binary Search, Fast Pointer, Slow Pointer**

I. INTRODUCTION

Binary Search is a widely used searching algorithm based on divide and conquer algorithm. Binary Search is particularly used in cases where the size of the input set of data is large and hence this approach provides better result in comparison with the linear search technique. However this technique requires the middle position of the sorted sequence to be identified so that the search may proceed from there. In case of conventional single dimensional arrays, determination of the middle element is easy and can be performed in O(1) or constant time using direct access. In case of linked list, as the memory allocation is non-contiguous, hence, determination of this middle element becomes difficult. We present a solution of the same using two pointers for determining the middle element of a sequence and hence perform binary search on that particular sequence.

II. RELATED WORK

Binary Search is usually fast and efficient for arrays and is difficult to implement for singly linked lists. This is because in an array, accessing the middle index between two given array indices is easy and fast. It usually involves finding out the average of the two indices and accessing that particular element. Thus the running time is constant .i.e. O (1).However in linked list, To access the middle node, we need to traverse the entire list, node by node and keep a count of the elements, and then traversing again up to half the counted value to go to the middle element. Thus the running time of finding middle node increases. We adopt an alternative algorithm to compute the middle element of a linked list.

III. ALGORITHM TO FIND OUT THE MIDDLE ELEMENT

We use two pointers named as Fast Pointer and Slow Pointer to compute the middle element of a linked list. We initialize both the pointers to the beginning of the linked list or head node at the first. Now we traverse through the list node by node. For every single step of the Slow Pointer, the Fast Pointer moves twice. Thus, when the Fast Pointer reaches the end of the list, the Slow Pointer has made half the number of movements as the Fast Pointer and hence points to the middle element as required.

IV. SOURCE CODE FOR DETERMINING THE MIDDLE ELEMENT

```
node * middleNode( node * startNode , node * endNode)
{
  if( startNode == NULL )
  {
     //  If the linked list is empty

   return NULL;
  }

  node * slowPtr = startNode;
  node * fastPtr = startNode -> next;

  while ( fastPtr != endNode )
  {
       fastPtr = fastPtr -> next ;

       if( fastPtr != endNode )
       {
          slowPtr = slowPtr -> next ;
          fastPtr = fastPtr -> next ;

/*

Note that for each loop iteration,
slowPtr moves just one location
while fastPtr moves two nodes at a time.

*/
       }
  }
  return slowPtr ;
/*
At the end , the slowPtr will be
pointing to the middle node
*/
}
```

V. Algorithm for Implementing Binary Search On The Linked List After Computing the Middle Element

After implementing the function to get middle node using the concept of dual pointers, we now need to focus on the algorithm itself. We proceed in a similar way as binary search is implemented in arrays.

- We have the starting node (set to Head of the list), and the ending node (set to NULL initially).
- Then the middle node between the startNode & endNode is calculated using the method described earlier.
- If middleNode matches the required value to search , we have found the required node searched for and we return it
- else, if middleNode's data < value , then we need to move to upper half (by setting startNode to middle's next)
- else we move to lower half (by setting endNode to middle)

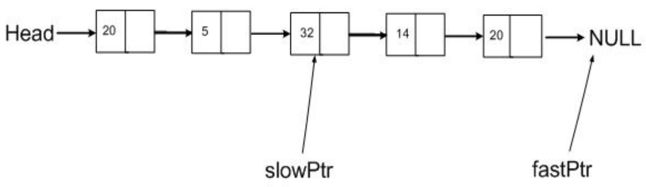

Fig 1. Determination of middle element of a linked list using two pointers.

Further to these, decision on the base case is necessary. It is important to decide when to terminate the loop as it cannot run infinitely. When dealing with arrays, the loop termination condition was index of start > index of end. But in linked list, we deal with nodes, so, the concept of indices, as in arrays, does not arise.A solution to this is keeping a check on values and terminate if : starting node's data > ending node's data. Howwever this check would fail if the list is comprised of all nodes with the same value. We then decide upon the base case as when the entire list gets traversed,we encounter the condition where endNode points directly before startNode i.e. endNode->next == startNode. This marks the end of iterations as the entire list has been traversed.

VI. Source Code for Implementing Binary Search On The Linked List After Computing the Middle Element using the concept of dual pointers

```
node * binarySearch( int valueToSearch )
{
   node * startNode = head;
   node * endNode = NULL;

   do
   {
      node * middle = middleNode( startNode , endNode );

      if( middle == NULL )
      {
         // Searched Element Not Present
         return NULL;
      }

      if( middle->data == valueToSearch )
      {
         return middle;
      }

      else if ( middle->data < valueToSearch )
      {
          // need to search in upper half
          startNode = middle->next;
      }
      else
      {
         // need to search in lower half
         endNode = middle;
      }

   }while(endNode == NULL ||
      endNode->next != startNode );

   // data not present
   return NULL;
}
```

VII. Conclusions

Implementation of Binary Search on single dimensional arrays is simpler than implementing it on linked lists due to the fact that linked list nodes are allocated memory non contiguously and also that there is no concept of index in a linked list as there is in arrays. The approach of Fast and Slow Pointers as described tries to implement binary search and any other divide and conquer algorithm on a singly linked list that relies on the determination of middle element of a sequence. Unlike arrays, where determination of middle element is done via direct access at O(1) time, computing middle element for a singly linked list may take upto O(n) time where n is the number of elements in the list.

References

[1] S. M. Metev and V. P. Veiko, *Laser Assisted Microtechnology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2] J. Breckling, Ed., *The Analysis of Directional Time Series: Applications to Wind Speed and Direction*, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.

[3] S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, “A novel ultrathin elevated channel low-temperature poly-Si TFT,” *IEEE Electron Device Lett.*, vol. 20, pp. 569–571, Nov. 1999.

[4] M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, “High resolution fiber distributed measurements with coherent OFDR,” in *Proc. ECOC’00*, 2000, paper 11.3.4, p. 109.

[5] R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, "High-speed digital-to-RF converter," U.S. Patent 5 668 842, Sept. 16, 1997.

[6] (2002) The IEEE website. [Online]. Available: http://www.ieee.org/

[7] M. Shell. (2002) IEEEtran homepage on CTAN. [Online]. Available: http://www.ctan.org/tex-archive/macros/latex/contrib/supported/IEEEtran/

[8] *FLEXChip Signal Processor (MC68175/D)*, Motorola, 1996.

[9] "PDCA12-70 data sheet," Opto Speed SA, Mezzovico, Switzerland.

[10] A. Karnik, "Performance of TCP congestion control with rate feedback: TCP/ABR and rate adaptive TCP/IP," M. Eng. thesis, Indian Institute of Science, Bangalore, India, Jan. 1999.

[11] J. Padhye, V. Firoiu, and D. Towsley, "A stochastic model of TCP Reno congestion avoidance and control," Univ. of Massachusetts, Amherst, MA, CMPSCI Tech. Rep. 99-02, 1999.

[12] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11, 1997.