© 2018 IJSRST | Volume 4 | Issue6 |Print ISSN: 2395-6011 | Online ISSN: 2395-602X

National Conference on Smart Computation and Technology in Conjunction with The Smart City Convergence 2018

Searching A Node In Linked List Using Divide And Conquer Approach

Puneet Mathur*, Rahul Gupta, Pranav Pandey

Department of Computer Engineering, Poornima Inst. of Engg. & Technology, Jaipur, India

ABSTRACT

The general idea of this paper is to implement binary search algorithm based on divide and conquer approach on the linked list. To implement the binary search algorithm the nodes of the linked list must be sorted in any order. In case of arrays it is very easy to find out the middle element because of the static and contiguous arrangement of its elements but because of the dynamic nature and random allocation of the nodes it is time consuming task to find out the middle node in sorted linked list. In this paper we have implemented two distinct pointers named, single Step pointer and double Step pointer to find out the middle node.

Keywords: Divide and conquer, Binary Search, middle element, single Step pointer, double Step pointer

I. INTRODUCTION

Binary search is a divide and conquer algorithm and is very widely used. It is specially used when the size of data is very large. If we have very large amount of data and we apply any linear searching algorithm on it, then the process will be everlasting, and as the complexity of linear search is O(n), so the complexity will be very high as the number of input data increases. So, we need to use divide and conquer approach, in which the whole work is divided and then conquered separately. Binary search does the same, it determines the middle element first and then compares it with the key element, and on the basis of that comparison the operation either stops or the algorithm operates on right half or the left half and this process is done recursively until the middle element matches with key. Implementation of this algorithm in arrays is quite easy. But if we talk about linked lists, it is not so straight forward. Determining the middle element is the first step, and in linked lists the nodes are at random locations in memory, so the middle node cannot be directly found. So, here we have used the special approach of dual pointers, in which we have used two pointers, a single Step pointer and a double Step pointer, to determine the middle node.

II. STEPS TO DETERMINE THE MIDDLE ELEMENT

1. First of all, we take two pointers, a singleStep pointer and a doubleStep pointer.
2. Then both the pointers are initialized with head of the linked list.
3. Then the linked list is traversed by both of these pointers.
4. For each step the doubleSteppointer will move twice the doubleStep pointer.
5. In this manner, when the doubleStep pointer will reach the end of list, the singleStep pointer will be pointing to the middle element of the list.
6. And we get the address of middle node through singleStep pointer.

III. ALGORITHM TO FIND OUT MIDDLE ELEMENT

MIDDLE_NODE(startNode, endNode)

1. IF startNode = NULL, THEN
2. RETURN NULL;

(END OF STEP 1 IF CLUASE)

3. singleStep = startNode
4. doubleStep = startNode
5. REPEAT STEPS 6 TO 9 WHILE singleStep != doubleStep,

6. singleStep = doubleStep -> NEXT
7. IF singleStep != endNode, THEN
8. singleStep = singleStep -> NEXT
9. doubleStep= doubleStep -> NEXT
   (END OF STEP 7 IF CLUASE)
   (END OF STEP 5 LOOP)
10. RETURN singleStep

IV. STEPS FOR IMPLEMENTING BINARY SEARCH ON LINKED LIST

1. Firstly, we take two pointers startNode and endNode, and initialize startNode with HEAD and then find last node of list and store that in endNode.
2. Then, we pass these startNode and endNode to the function MIDDLE_NODE and the return value is stored in middleNode.
3. If the info of middleNode matches with the key element, then the process is stopped, else proceed to step 4.
4. If info of middleNode> key, then we need to repeat the above steps for upper half, else proceed to step 5.
5. If info of middleNode< key, then we need to repeat the above steps for lower half.

If the key element is found then the loop will terminate, but if the key element does not exist in the list, then we will get into indefinite loop. If we are handling arrays, then we simply put the condition LOW <= HIGH, but in case of linked list the nodes exists at random locations in the memory unlike in arrays, so loop termination condition will be different. In this process, we continuously divide the array, goes onto the left or right side of middle element on the basis of comparison with key, so if during this process, we get onto a single node parition, whose info does not match with key, then we have to terminate. So, when we have a single node partition, the condition would be, startNode=endNode=middleNode.

V. ALGORITHM FOR IMPLEMENTING THE BINARY SEARCH

BINARY_SEARCH(HEAD, KEY)

1. startNode =HEAD
2. REPEAT STEP 3 WHILE endNode -> NEXT != NULL,
3. endNode = endNode -> NEXT
   (END OF STEP 2 LOOP)
4. REPEAT STEPS 5 TO 17,
5. middleNode = MIDDLE_NODE(startNode, endNode)
6. IF middleNode -> INFO = KEY, THEN
7. PRINT “KEY FOUND”
8. BREAK
9. ELSE IF middleNode -> INFO > KEY, THEN
10. endNode = startNode
11. REPEAT STEP 12 WHILE endNode -> NEXT != middleNode,
12. endNode = endNode -> NEXT
    (END OF STEP 11 LOOP)
13. ELSE
14. startNode = startNode -> NEXT
15. (END OF STEP 6 IF CLAUSE)
16. IF startNode == middleNode AND endNode == middleNode, THEN
17. BREAK;
    (END OF STEP 4 LOOP)
18. RETURN

VI. CONCLUSION

Binary search implementation on Arrays is more easy as compared to Linked List, due to following reasons:

a. The array elements are directly accessible by their locations or indices, while this is not the case in linked list (in which any node is referred by the next pointer of its previous node).
b. The array elements are stored at contiguous memory locations, while linked list elements are stored at random locations in memory.
c. The arrays are static in nature, as its size(elements holding capacity) has to be decided at the time of its definition, while linked list is dynamic, so the nodes can be added or deleted at run time.

Complexity of Binary Search algorithm in case of linked list is high as compared to arrays, as there is an additional algorithm to find out the middle node.

VII. REFERENCES

[1] http://en.wikipedia.org/wiki/Binary_search_algorithm
[2] http://www.getappninja.com/blog/implementing- a-binary-search-in-ios