Design an algorithm to find the location of kth largest element in a max heap

Harsh Meena - IIT20190005,
Asha Jyothi Donga - IIT2019006,
Sampada Kathar - IIT2019007

Date: 13-02-2021

Abstract

In this paper we have devised an algorithm which finds the location of the kth greatest element in a max heap. We have determined both the time, and the space complexity of our algorithm by illustrating through some graphs.

1 Problem

Find the index of the kth largest element in given max heap.

2 Keywords

Dictionary, Max Heap,Priority Queue, Array.

3 Introduction

A heap is a tree-based data structure in which all the nodes of the tree are in a specific order.
Max Heap: In this type of heap, the value of parent node will always be greater than or equal to the value of child node across the tree and the node with highest value will be the root node of the tree. Heap is taken in the form of array where index of the root element is 0. and If i is the index of the node in the array Left child is (2*i)+1, Right child is (2*i)+2 and Parent is :(i-1)/2.

4 Algorithm Design

To find the index of the kth greatest element for the given Binary Max heap Array :

1. Create the max heap (priority_queue in stl) pq.
2. Push the root of given max heap array (i.e first element of the Heap array).
3. Repeat the steps for k-1 steps.
   - Pop from the max heap pq (greater element in the max heap pq)
   - Insert the left and right of the popped element if the children exist.
4. The greatest element in the pq is the kth greatest element in the given heap.

To get the left child and right child of the node we need index, so the pair of value and its index is pushed into the priority_queue (max heap).

5 Algorithm Analysis

Let the Max heap in form of array.

```
int kthGreatestInMaxHeap(Array heap,Int k)

    priority_queue<Pair(value,index) > pq
    pq.push(Pair(heap[0], 0)
```

```
for( i=0 to k-1)
    poppedIndex= indexOf(pq.top())
    pq.pop()
    leftChild=2*poppedIndex+1;
    rightChild=2*poppedIndex+2;
    if(leftChild exists)
        pq.push(Pair(heap
        [leftChild],leftChild)
    if(rightChild exists)
        pq.push(Pair(heap
        [rightChild],rightChild)

return indexOf(pq.top())
```

5.1 Time Complexity Analysis

As we can see from the above algorithm, we need to go through the max heap, k-1 times, so that finally our priority queue will have the kth largest element.

The initial size of the priority queue is one, and it increases by at most one at each of the k – 1 steps. Therefore, there are maximum k elements in the priority queue and the time complexity of the pop and insert operations is $O(logk)$.

So the total Time Complexity of this Algorithm becomes of order of $k * log(k)$.

The Best case is when, k=1 and hence TC for best case is $\Omega(1)$.

The Worst case will depend upon value of N. Hence, Worst case TC will be $O(k * log(k))$.

5.2 Space Complexity:

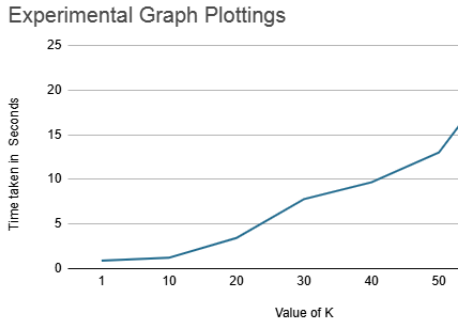
In this algorithm we have used priority queue for storing k max elements, hence Space Complexity (Extra Space used) is O(k).

6 Experimental Analysis

In the following table some cases are plotted,

k	Time Taken
2	0.0003
4	0.0005
8	0.0009
16	0.0011
20	0.0010
24	0.0019
30	0.02
40	0.034
50	0.04

Here k represents the kth greatest element, and on y-axis the time taken is in seconds



The above graph shows the variance of Time with respect to size of heap taken(N), and follows our theoretically calculated TC of $O(k * log(k))$.

7 Conclusion

From this paper we can conclude that the algorithm devised by us is an efficient way of solving our given problem.
It takes a priority queue to get max kth element from gic=ven heap which makes it efficient memory wise, and with a worst case complexity of only $O(k * log(k))$.

8 References

[1]vaibhav29498, 'K-th Greatest Element in a Max-Heap', GeeksforGeeks, 2018. [Online]. Available here [Accessed: 1-Feb-2021]
[2]'Heap Data Structures', tutorialspoint. [Online]. Available here [Accessed: 1-Feb-2021]

[3]GeeksforGeeks, 'Priority Queue — Set 1 (Introduction)', GeeksforGeeks, 2018. [Online]. Available here [Accessed: 1-Feb-2021]