MIN-MAX HEAP AND DEAP DATA STRUCTURE A Research Report on MIN MAX HEAP AND DEAP DATA STRUCUTRE

A Research Report on

MIN MAX HEAP AND DEAP DATA STRUCUTRE

Jiwan Pokharel

Department of Computer Science

Loyola University Chicago

Chicago, Illinois, USA

Abstract

This is a research report undertaken as the Assignment-3, requirement for the completion of COMP 460, Algorithms and Complexity, Spring 2014. This report is a miniature excerpt of the resources available through various publications, books and websites, intended to produce a concise understanding of the Heap and Deap Data Structures. I would like to thank Dr. Chandra Sekharan, for making me undertake this work as a requirement in the course.

1. HEAP:

A heap is a special kind of data structure, which is based on tree data structure and fulfills the property of heap .i.e. the key associated with the parent node, is either larger than the key of the child nodes or smaller than the key of the child nodes. Thus, the heap in which the key of the parent node is larger than the key of the child nodes, throughout the heap structure is known as the *max heap*, and if the key of the parent node is smaller than the key of the child nodes, throughout the heap structure, it is known as *min heap*. Heap data structure has wide usage like in *heapsort* technique, in various graph algorithms, as well as to efficiently represent dynamic priority lists. A good aspect of heap structure is its capability to be efficiently represented in array without making the using of explicit pointers. The following example depicts an example of a heap structure:

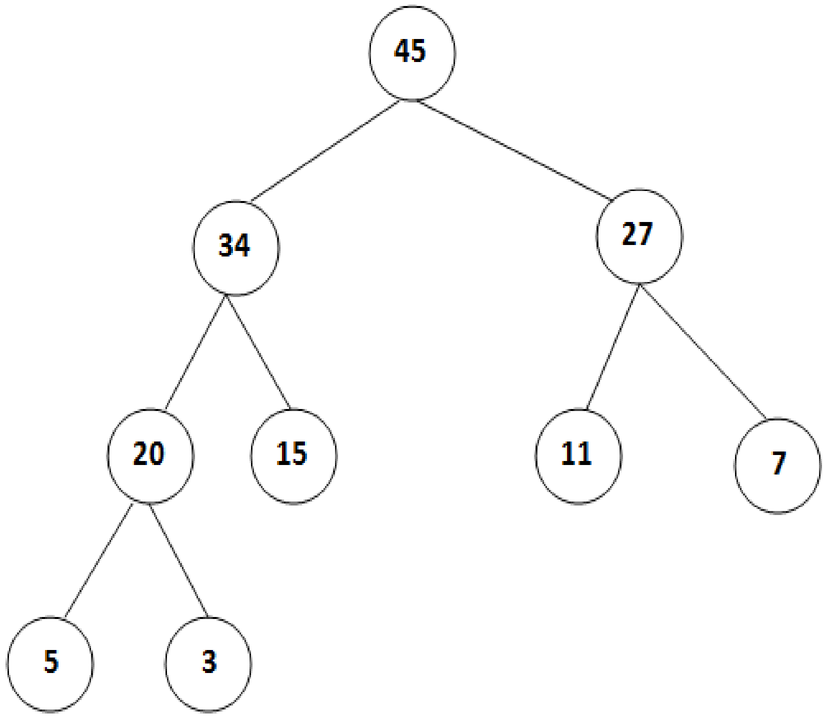

Fig 1: Example of a heap data structure

2. MIN-MAX HEAP:

Heap data structure has various types viz. min heap, max heap, min-max heap, etc. A Min-Max heap is a complete binary tree structure where the root of the tree is set to the minimum level. This implies that any node in the minimum level has smaller key value than its child or descending nodes. In this data structure, if it is not empty, all its elements are known as keys. Moreover, it employs alternating min and max levels in its structure, with root being at the min level, and subsequent child nodes alternating as max level and min level respectively.

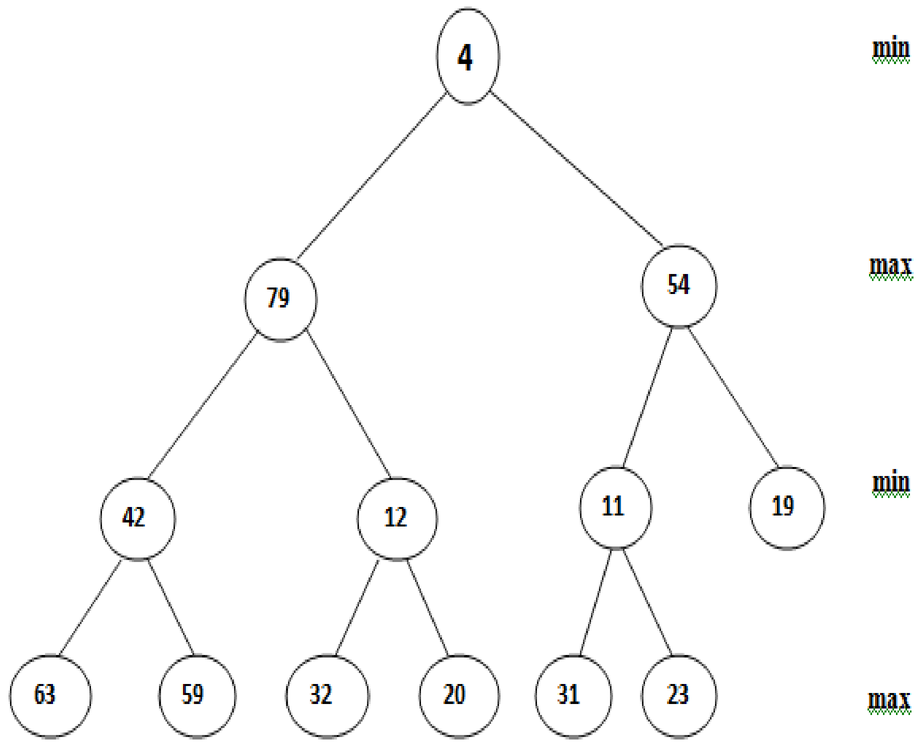

Fig. 2: Min-Max Heap

1. ***Min level:***

   Let us suppose that any **node A** belongs to the Min level. This implies, node A has the smallest key from all of the elements in its subtree or child nodes. Thereby, this node A is called as a *min node.*

2. ***Max level:***

   Similarly, if we take any **node B** in Max level, then it has the largest key as compared to the elements available in its subtree or subsequent child nodes. Thus, we call this node B as a *max node.*

Fig.2. depicts a min-max heap structure. The root node containing key value **'4'** is the minimum node in the heap. Similarly, alternating min-max levels are created. Thus, each level helps to identify either minimum or maximum values as per their level.

2.1.IMPLEMENTATION OF MIN-MAX HEAP:

We can implement a min max heap using an array. Let us consider a min max heap with ***'n'*** elements, which can be stored in an array **a[n]**. As we have min and max levels in this heap structure, the nodes lying at preceding levels contain key values either smaller or larger than the nodes in the subsequent levels in an alternating fashion. This helps us to conclude that the root node has the smallest key value and the largest value gets stored in one of the children of the root. In the above example, the root node stores the value **4** , which is the smallest value and its child node at the left-hand side stores value **79**, which is the largest value. Now, we can proceed to implement the structure using the array. As the heap structure employs a tree structure, it is identifiable that the node at **i**th location corresponds to the $\log_2 i$ level.

2.2.INSERTION IN MIN-MAX HEAP:

We can perform insertion operation in min-max heap using the similar operation as in conventional heap structure. We can employ the following pseudocode to perform insertion into the min-max heap structure:

```
Procedure insertion(a[],n,x):
    n=n+1;
    parent=n/2;
    if (parent==0)      //This implies heap is empty
        a[1]=x;
    else{
        if(level==min_heap ){
            if(x<a[parent]){
                construct_min_heap(a[],parent,x);
            }
            else
```

```
                construct_max_heap(a[],n,x);
        }
        else if(level==max_heap){
            if(x>a[parent]){
                A[n]=a[parent];
                construct_max_heap(a[],parent,x);
            }
        else
            construct_min_heap(a[],n,x);
        }
    }
```

2.3.DELETION OF MINIMUM ELEMENT:

The deleteMinimum() procedure depicts the process of deleting the minimum element from the min-max heap in **O(log n)** time complexity. In this process, the minimal element is identified and eventually, the next minimal element is over-written over it. This procedure is repeated until the minimum level is balanced completely.

```
Procedure deleteMinimum(a[],n):
    x=n--;
    for i=1 to n/2{
        mc=minimum_node(i,n);        //To find the minimum value node
        if(x.Keyvalue<=a[mc].Keyvalue)
            break;
        a[i]=a[mc];
        if(mc<=2*i+1){
            i=mc;
            break;
        }
        if(x.KeyValue>a[mc/2].KeyValue)
            Swap(a[mc/2],x);
        i=mc;
    }
    a[i]=x;
```

2.4.DELETION OF MAXIMUM ELEMENT:

This is the inverse of the deletion of minimum element from the min-max heap. To perform this operation, we shall employ deteteMaximum() procedure. This procedure takes the time in the order of **O(log n)**, where n represents the number of nodes and $\log_2 n$ implies the height of the tree. Here, we shall identify the maximum element and the eventually the next maximum element from among its children and grand-children. Then, we shall overwrite the next maximum value over the intial maximum value. This process is repeated until the heap is completely maximum level balanced.

```
Procedure deleteMaximum(a[],n):
    x=n--;
    for i=1 to n/2{
        mc=maximum_node(i,n);      //To find the maximum value node
        if(x.Keyvalue>=a[mc].Keyvalue)   break;
        a[i]=a[mc];
        if(mc<=2*i+1){
            i=mc;
            break;
        }
        if(x.KeyValue<a[mc/2].KeyValue)
            Swap(a[mc/2],x);
        i=mc;
    }
    a[i]=x;
```

3. DEAP:

Deap is a data structure which has no element or key value at the root node. It is constructed by using the following principles:

1. Root node shall not have any element or root node is empty.
2. Min heap shall be the left subtree of the deap.
3. Max heap shall be the right subtree of deap.

Thus, a deap structure can mathematically provide correctness to the following statement:

If the left sub tree and right sub tree of certain nodes are non-empty, and their corresponding nodes can be represented by ***'i'*** and ***'j'*** respectively, then:

$$\boxed{\textbf{i.KeyValue <= j.KeyValue}}$$

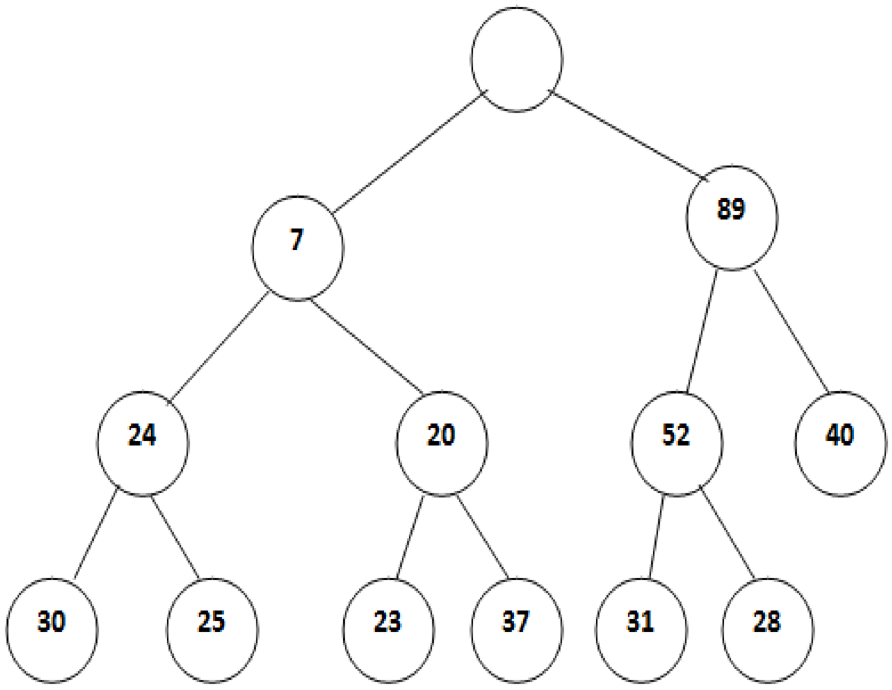

Fig. 3: Deap Data Structure

3.1.INSERTION INTO DEAP

To insert into deap data structure, we might need the procedures to find the minimum and maximum values as depicted below:

```
Procedure min_value(n):      //To determine the minimum value in deap.
        return $n-2^{\log_2 n-1}$;
Procedure max_value(n):      //To determine the maximum value in deap.
        return $n+2^{\log_2 n-1}$;
```

The insertion operation in deap data structure can be done as follows:

1. For any heap a[], we need to check if n is a position within the maximum-heap of deap.
2. We shall then compute the minimum and maximum values in deap.
3. Now, the key values at left sub- tree and right sub-tree are compared.
4. Finally, we perform the insertion operation.

```
Procedure deap_insertion(a[],x,n):
        if (n==1)
                a[2]=x;
        else{
                if ( n is in maximum subtree){
                         index=min_value(n);
                        if(x<a[index]){
                                a[n]=a[index];
                                insert x in minimum subtree;
                        }
                else
                        insert in maximum subtree;
        }
        else{
```

```
    index=max_value(n);
    if(x>a[index]){
        a[n]=a[index];
        insert x into maximum subtree;
    }
    else
        insert x into minimum subtree;
}
```

3.2.DELETION FROM DEAP

We can deletete an element from deap in O(log n) time, where, ‘n’ represents the number of elements in the deap data structure. During deletion, we shall try to delete from the minimal value. As the height of the tree is **log n**, it takes time of order of **log n**. We can depict deletion operation as follows:

```
Procedure deap_deletion(a[],n):
    if(n<2)
      return; //There are no elements.
    min=a[2]; //Saving the minimum value
    for (i=2;2*i<=n;a[i]=a[k],i=k){
        k=i*2;
        If(k+1<=n&&a[k]>a[k+1)
            k++;
    }
    k=max_value(i);
    if(x>a[k]){
        a[i]=a[k];
        insert x into maximum subtree;
    }
    else{
        insert x into minimum subtree;
    }
```

4. RUN TIME COMPLEXITIES

We can illustrate the run-time complexities for various data structures as follows:

	Insertion	**Deletion**	**DeleteMinimum**	**DeleteMaximum**
Heap	O(log n)	O(log n)		
Min-Max Heap	O(log n)		O(log n)	O(log n)
Deap	O(log n)	O(log n)		

Thus, in terms of the worst case complexities, all these data structures tend to take the similar time complexities, but it is notable that the various variables in the computing system, internal operational structures, pattern of data as well as constants associated with each of these operations impact the time consumption for each of these data structures. Also, these are the worst case complexities, indicating these to be the maximum possible worst times, but there can be abundant scenario where the time consumption can fall well below these indicated orders of time complexities.

REFERENCES

[1] "Min Max Heaps and Generalized Priority Queues", M.D. Atkinson et. Al., 1986, Communications of the ACM

[2] http://en.wikipedia.org/wiki/Heap_%28data_structure%29

[3] "Data Structure and File Management", 2000, Computer Science Dept, VA Tech.

[4]"Symmetric Min-Max Heap: A simpler data structure for double ended priority queue", A. Arvind et. Al., 1999, Information Processing Letters

[5] "The Bounds of Min-Max Pair Heap Construction", R. A. Chowdhury et. Al., 2002, Computers and Mathematics with Applications

[6] "Introduction to Algorithms", 3rd edition, Thomas H. Cormen et. Al.

[7] "15-121 Introduction to Data Structures", Carnegie Mellon University-CORTINA

[8] "A Note on the Construction of The Data Structure Deap", S. Carlsson, J. Chen,1989, Information Processing Letters

[9] http://en.wikipedia.org/wiki/Min-max_heap

[10] http://www.diku.dk/forskning/performance-engineering/Jes~/heaplab/heapsurvey_html/node11.html