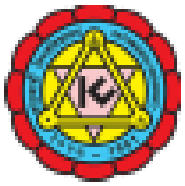
Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavrepalanchowk



COMP202

Mini Project-Report

Submitted By:

Rohan Dhakal (14)

CE II Year/I semester

Submitted to:

Dr. Rajani Chulyadyo

Assistant Professor,

Department of Computer Science and Engineering

Submission Date:

08/20/2021

HEAP DATA STRUCTURE:

A heap is a complete or almost complete binary tree, and the binary tree is a tree in which the node can have utmost two children.

Basically, there are two types of Heaps. They are:

1. **MAX HEAP**

   In max heap, every node contains a greater or equal value element than its child nodes. Thus, the root node contains the largest value element.

2. **MIN HEAP**

   In the min heap, every node contains a lesser or equal value element than its child nodes. Thus, the root node contains the smallest value element.

PRIORITY QUEUE:

A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue.

A priority queue is an extension of a queue that contains the following characteristics:

- Every element in a priority queue has some priority associated with it.
- An element with the higher priority will be deleted before the deletion of the lesser priority.
- If two elements in a priority queue have the same priority, they will be arranged using the FIFO principle.

There are two types of priority queue:

1. **ASCENDING PRIORITY QUEUE**

   In the ascending order priority queue, a lower priority number is given as a

higher priority. It can be implemented using min heap data structure.

2. **DESCENDING PRIORITY QUEUE**

   In the descending order priority queue, a lower priority number is given as a higher priority. It can be implemented using max heap data structure.

ABOUT THE PROJECT:

The project that I was assigned was the implementation of heap data structure. And, the implementation of the ascending priority queue using the heap data structure. So, min heap data structure is implemented for the implementation of the ascending priority queue.

Operations of Min Heap implementations with time complexities are:

1. **isEmpty():**

   Returns true if the Heap is empty.Else, false. The time complexity is O(1).

2. **isFull():**

   Returns true if the Heap is full.Else, false. The time complexity is O(1).

3. **insert(int key):**

   This function inserts/adds the new key at the end of the tree. If the key to be inserted is smaller than the parent index, then the min heap structure will be broken . So, we up-heapify the heap, in order to maintain heap structure. Thus, the time complexity for inserting a key to the min heap is O(log(n).

4. **extractMin();**

   This function removes and returns the minimum element i.e. root from the min heap. The time complexity of this operation is O(log(n)) as heap property should be maintained using heapifyDown property.

   But incase of a single element in the heap, which is the best case, the time complexity is O(1).

5. **heapifyUp(int index):**

   This function reorders the broken heap structure. The new node is floated up the tree until it reaches its correct location in the heap. The time complexity of this function is O(log(n)).

6. **heapifyDown(int index):**

   This function reorders the broken heap structure. The root is floated down the tree until it reaches its correct location in the heap. The time complexity of this function is O(log(n)).

7. **decreaseKey(int index, int key):**

   This operation decreases the value of the element's key to the new value at a node. This involves finding the node with the given value, changing the value and then heapifyUp to restore the min heap property. Thus, the time complexity is O(log(n)).

8. **removeKey(int index):**

   In this function, we delete the selected index key which takes O(log(n)) time.

9. **getMin():**

   This function returns the minimum value i.e, root, in the min heap data structure. Thus, its time complexity is O(1).

10. **display():**

    This function displays the min heap data structure visiting all the nodes. Thus, its time complexity is O(n).\

11. **swap(int &, int &)**

    This function exchanges the address of the parent index and the selected index

and swap is used in the heapify function. The time complexity of the given function is O(1).

Other helper functions of min heap implementations are:

1. **parent(int index):**
   This function returns the parent index of the min heap. The time complexity of the given function is O(1).

2. **getLeftChild(int index):**
   This function returns the left child index of the parent index in the min heap. The time complexity of the given function is O(1).

3. **getRightChild(int index):**
   This function returns the right child index of the parent index in the min heap. The time complexity of the given function is O(1).

So, the overall time complexity of the program becomes O(n).

Major Operations of ascending Priority Queue with time complexities are:

1. **isEmpty():**
   Checks if the queue is empty or not. Return true if the queue is empty. Otherwise, return false. Time Complexity is O(1).

2. **isFull():**
   Checks if the queue is full or not. Return true if the queue is full. Otherwise, return false. Time Complexity is O(1).

3. **enqueue(int):**
   This operation inserts the element to the priority queue with arbitrary priority.

Insert function of min heap is used. So,The time complexity to enqueue the element in the queue is O(log(n)).

4. **dequeue():**

   The dequeue operation removes the element with highest priority.In ascending priority queue, smallest element is given the highest priority. So, deletion of the smallest element takes place first. ExtractMin is used for dequeue. So,The time complexity is O(log(n)).

5. **getMinimum():**

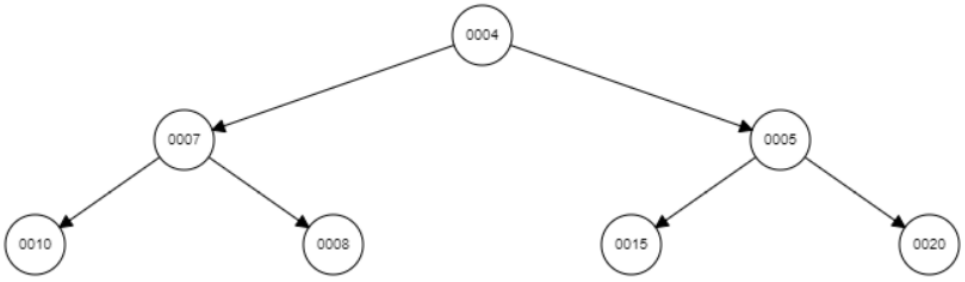
   The same getMin function is used as a min heap with time complexity O(1).

6. **displayQueue():**

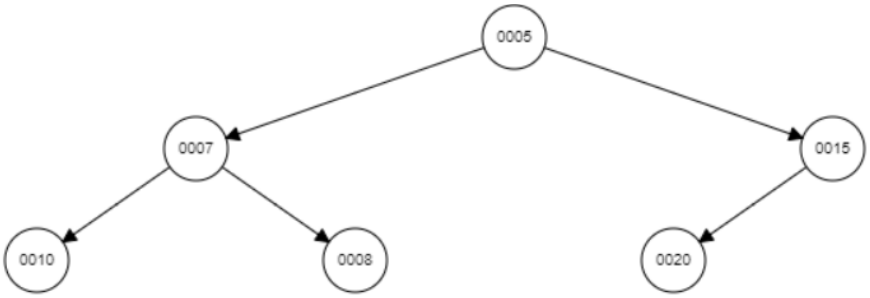
   The same display function as minHeap is used with time complexity O(n)

So, the overall time complexity of the program becomes O(n).

In my implementation, 10,5,15,8,7,4,20 are inserted. So, the heap will look as



Now, After the removal of Minimum, the heap will look as follows:



In this way, heap data structure functions. Execution of operations are already mentioned.

OUTPUT OF PROGRAM:

```
PS C:\Users\ROHAN\Desktop\CE2019_Roll_14__MiniProject> g++ src/main.cpp src/heap.cpp
PS C:\Users\ROHAN\Desktop\CE2019_Roll_14__MiniProject> ./a

---------------------------------------
         Min Heap Created
---------------------------------------

Is Empty? Yes(1)/ No(0): 1
Inserted Key: 10
Inserted Key: 5
Inserted Key: 15
Inserted Key: 8
Inserted Key: 7
Inserted Key: 4
Inserted Key: 20
Cannot insert. Size Full
Is Empty? Yes(1)/ No(0): 0
Is Full? Yes(1)/ No(0): 1
Minimum/root element of the given Heap is: 4
The elements are:
4       7       5       10      8       15      20
After extracting minimum, The elements are:
5       7       15      10      8       20
The elements are:
3       5       15      10      8       20
After removing element from index 3, The elements are:
3       5       15      20      8

-----------------------------------------------------------------
        Ascending Priority Queue Implementation
-----------------------------------------------------------------

Is Empty? Yes(1)/ No(0): 1
Inserted Key: 10
Inserted Key: 5
Inserted Key: 15
Inserted Key: 8
Inserted Key: 7
Inserted Key: 4
Inserted Key: 20
Cannot Enqueue 44 .Queue overflow.
The elements are:
4       7       5       10      8       15      20
Is Empty? Yes(1)/ No(0): 0
Is Full? Yes(1)/ No(0): 1
Minimum element of priority queue is: 4
Value Dequeued is: 4
The elements are:
5       7       15      10      8       20
Inserted Key: 3
The elements are:
3       7       5       10      8       20      15
PS C:\Users\ROHAN\Desktop\CE2019_Roll_14__MiniProject>
```

Github Link:
[fearalert/CE2019_Roll_14_MiniProject (github.com)](https://github.com/fearalert/CE2019_Roll_14_MiniProject)