Operations of Heap Data Structure

Operations on Min Heap:

1) getMini(): It returns the root element of Min Heap. Time Complexity of this operation is O(1).

2) extractMin(): Removes the minimum element from MinHeap. Time Complexity of this Operation is O(Logn) as this operation needs to maintain the heap property (by calling heapify()) after removing root.

3) decreaseKey(): Decreases value of key. The time complexity of this operation is O(Logn). If the decreases key value of a node is greater than the parent of the node, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

4) insert(): Inserting a new key takes O(Logn) time. We add a new key at the end of the tree. IF new key is greater than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

5) delete(): Deleting a key also takes O(Logn) time. We replace the key to be deleted with minum infinite by calling decreaseKey(). After decreaseKey(), the minus infinite value must reach root, so we call extractMin() to remove the key.

Below is the implementation of basic heap operations.

C++

```
// A C++ program to demonstrate common Binary Heap Operations
#include<iostream>
#include<climits>
using namespace std;
```

```
// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of min heap
    int heap_size; // Current number of elements in min heap
public:
    // Constructor
    MinHeap(int capacity);

    // to heapify a subtree with the root at given index
    void MinHeapify(int );

    int parent(int i) { return (i-1)/2; }

    // to get index of left child of node at index i
    int left(int i) { return (2*i + 1); }

    // to get index of right child of node at index i
    int right(int i) { return (2*i + 2); }

    // to extract the root which is the minimum element
    int extractMin();

    // Decreases key value of key at index i to new_val
    void decreaseKey(int i, int new_val);

    // Returns the minimum key (key at root) from min heap
    int getMin() { return harr[0]; }

    // Deletes a key stored at index i
    void deleteKey(int i);

    // Inserts a new key 'k'
    void insertKey(int k);
};
```

```
// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(int cap)
{
    heap_size = 0;
    capacity = cap;
    harr = new int[cap];
}

// Inserts a new key 'k'
void MinHeap::insertKey(int k)
{
    if (heap_size == capacity)
    {
        cout << "\nOverflow: Could not insertKey\n";
        return;
    }

    // First insert the new key at the end
    heap_size++;
    int i = heap_size - 1;
    harr[i] = k;

    // Fix the min heap property if it is violated
    while (i != 0 && harr[parent(i)] > harr[i])
    {
    swap(&harr[i], &harr[parent(i)]);
    i = parent(i);
    }
}

// Decreases value of key at index 'i' to new_val. It is assumed that
// new_val is smaller than harr[i].
void MinHeap::decreaseKey(int i, int new_val)
{
    harr[i] = new_val;
    while (i != 0 && harr[parent(i)] > harr[i])
    {
    swap(&harr[i], &harr[parent(i)]);
    i = parent(i);
    }
}
```

```
// Method to remove minimum element (or root) from min heap
int MinHeap::extractMin()
{
    if (heap_size <= 0)
        return INT_MAX;
    if (heap_size == 1)
    {
        heap_size--;
        return harr[0];
    }

    // Store the minimum value, and remove it from heap
    int root = harr[0];
    harr[0] = harr[heap_size-1];
    heap_size--;
    MinHeapify(0);

    return root;
}


// This function deletes key at index i. It first reduced value to minus
// infinite, then calls extractMin()
void MinHeap::deleteKey(int i)
{
    decreaseKey(i, INT_MIN);
    extractMin();
}

// A recursive method to heapify a subtree with the root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
```

```
        if (smallest != i)
        {
            swap(&harr[i], &harr[smallest]);
            MinHeapify(smallest);
        }
    }
```



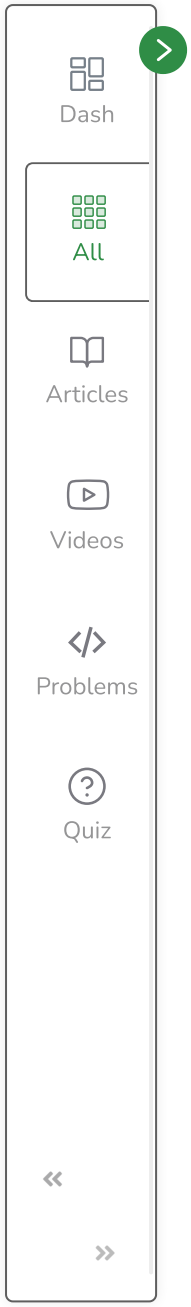
```
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// Driver program to test above functions
int main()
{
    MinHeap h(11);
    h.insertKey(3);
    h.insertKey(2);
    h.deleteKey(1);
    h.insertKey(15);
    h.insertKey(5);
    h.insertKey(4);
    h.insertKey(45);
    cout << h.extractMin() << " ";
    cout << h.getMin() << " ";
    h.decreaseKey(2, 1);
    cout << h.getMin();
    return 0;
}
```

Output:

2 4 1

Mark as Read



Report An Issue

If you are facing any issue on this page. Please let us know.