January 2024 | CSE 106

Assignment 6: Heaps

Objective

In this assignment, you will implement a **Max Heap** data structure using an array representation that supports various heap operations, such as insertion and deletion, while maintaining the heap property efficiently.

Introduction to Max Heap

A **Max Heap** is a complete binary tree where each parent node is greater than or equal to its child nodes. The root node contains the largest value.

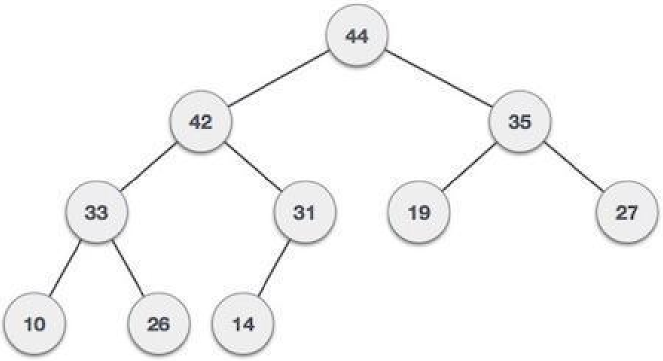

Figure: An example of max heap

Heaps are commonly used for various purposes, including sorting, searching, and organizing data.

Implementation of Max-Heap

A max heap can be represented as an array (**`Arr`**).

- The root element will be at **`Arr[0]`**
- For any node **`Arr[i]`,**
    - o Left child is stored at index **`2i+1`**
    - o Right child is stored at index **`2i+2`**
    - o Parent is stored at index ⌞**`(i-1)/2`**⌟

Max Heap Operations

Basic Operations

Basic operations are the core operations that users perform on a Max Heap.

1. **`insert(x):`**
   Inserts a new element x into the heap. The heap property is restored using **sift-up**.
   Example: If the heap is [50, 30, 20] and insert(60) is called, the heap will become [60, 50, 20, 30].

2. **`find-max:`**
   Returns the maximum element (root) of the heap without removing it.
   Example: If the heap is [50, 30, 20, 15], calling find-max() will return 50.

3. **`extract-max():`**
   Removes and returns the maximum value (the root) from the heap. After the removal, **sift-down** is used to restore the heap property.
   Example: For the heap [50, 30, 20], extract-max() will return 50 and modify the heap to [30, 20].

4. **`increase-key(index, new_value):`**
   Increases the value of the element at index to new_value and restores the heap property using **sift-up**.
   Example: If the heap is [50, 30, 20] and increase-key(2, 35) is called, the heap will become [50, 35, 30].

5. **`delete-key(index):`**
   Deletes the element at the index.
   Example: For the heap [50, 30, 20], delete-key(1) will remove the element at index 1 and rebalance the heap.

Inspection Operations

Inspection operations allow users to query the state of the heap.

6. **`get-size():`**
   Returns the number of elements in the heap.
   Example: For the heap [50, 30, 20], get-size() will return 3.

7. **`is-empty():`**
   Returns true if the heap is empty, false otherwise.
   Example: For an empty heap, is-empty() will return true.

8. **`print-heap():`**
   Prints all the elements of the heap.
   Example: For the heap [50, 30, 20], calling print-heap() will output 50 30 20.

9. **`is-valid-max-heap():`**
   Determines whether the heap property is preserved at any given time. It returns true if the Max Heap property is valid and returns false otherwise.
   Example: If the heap is [50, 30, 20], calling is-valid-max-heap() will return true. The function will return false if the heap is incorrectly ordered, like [30, 50, 20].

Internal Operations

Internal operations are used internally for efficiently maintaining the heap property (after modifications).

10. **`sift-up(index):`**
    Moves the element at index upwards in the heap to restore the heap property after insertion.

11. **`sift-down(index):`**
    Moves the element at index downwards in the heap to restore the heap property after deletion or replacement.

Input Format

Each line of input will contain a command representing one of the heap operations. Depending on the command, an additional value may follow. Here is the command mapping table:

Command	**Operation**	**Description**
1 x	insert(x)	Insert the value x into the heap
2	extract-max()	Extract and return the maximum element
3	find-max()	Return the maximum value without removing it
4	get-size()	Return the size of the heap
5	is-empty()	Return true if the heap is empty, false otherwise
6 i	delete-key(i)	Delete the element at index i
7 i x	increase-key(i, x)	Increase the value at index i to x
8	print-heap()	Print the current state of the heap (array representation)
9	is-valid-max-heap()	Check if the heap satisfies the Max Heap property

Instructions

- You have been given a header file named `MaxHeap.h`. You only have to edit this file in the mentioned places: `/**Write your code here**/`. Implement all the operations.

- You have also been given a main function in the `main.cpp` file to evaluate your implementations. You do not have to edit anything in this file.
- Input and output will be handled through files. You can enter your inputs in the `input.txt`, and the outputs will be generated in `output.txt`. The sample input is already given in `input.txt`.

Sample I/O

Sample Input (input.txt)	Sample Output (output.txt)
1 50	**Inserted 50 into the heap.**
8	**Heap: 50**
9	**Max Heap property is preserved.**
1 30	Inserted 30 into the heap.
8	Heap: 50 30
9	Max Heap property is preserved.
1 20	**Inserted 20 into the heap.**
8	**Heap: 50 30 20**
9	**Max Heap property is preserved.**
3	Max: 50
2	Extracted Max: 50
8	Heap: 30 20
9	Max Heap property is preserved.
5	**Is heap empty? No**
4	**Heap size: 2**
6 0	Deleted element at index 0
8	Heap: 20
9	Max Heap property is preserved.

Submission Guidelines

- Create a directory with your 7-digit student ID as its name.
- Place all source files (`.c`, `.h`) into that directory.
- Zip the directory in the `.zip` format (any other format like .rar, .7z, etc. is not acceptable).
- Upload the `.zip` file to Moodle.

Submission Deadline

November 17th, 2024 11:59 PM

Tentative Marks Distribution

Task	**Tentative Mark**
insert(x)	5
extract-max()	10
find-max()	5
get-size()	5
is-empty()	5
delete-key(i)	10
increase-key(i, x)	5
print-heap()	5
is-valid-max-heap()	10
sift-up(index)	20
sift-down(index)	20
Total	**100**