MIT World Peace University
Advanced Data Structures

Assignment 7

Naman Soni Roll No. 10

Contents

1 Problem Statement

Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure and Heap sort.

2 Objective

- To study the concept of heap
- To study different types of heap and their algorithms

3 Theory

3.1 *What is a heap?*

A heap is a specialized tree-based data structure that is used to efficiently manage and prioritize dynamically changing data. A heap is typically used to implement a priority queue, which is a collection of elements where each element has an associated priority. The heap allows for efficient insertion and removal of elements while maintaining a specific ordering based on their priorities.

There are two types of heaps: max heaps and min heaps. In a max heap, the element with the highest priority is always at the root of the tree, while in a min heap, the element with the lowest priority is at the root. Heaps can be implemented as arrays, where the elements are stored in a specific order, or as trees, where the elements are organized in a hierarchy.

The most common operations performed on a heap include insertion of a new element, removal of an element with the highest or lowest priority, and querying the element with the highest or lowest priority. These operations can be performed in logarithmic time, which makes heaps an efficient data structure for managing large amounts of data in real-time applications such as job scheduling, network routing, and resource allocation.

3.2 *Write different types of Heaps*

There are two main types of heaps:

- Max Heap - In a max heap, the maximum element is always at the root of the tree.
- Min Heap - In a min heap, the minimum element is always at the root of the tree.
- In addition to these, there are also two variations of heaps:
- Binary Heap - A binary heap is a type of heap data structure where each node has at most two child nodes. Binary heaps are often implemented using arrays and are commonly used to implement priority queues.
- Fibonacci Heap - A Fibonacci heap is a type of heap data structure that provides faster amortized time complexity than binary heaps for some operations. It consists of a collection of trees, and the heap's structure is more relaxed than that of a binary heap. Fibonacci heaps are commonly used in algorithms such as Dijkstra's algorithm and Prim's algorithm for graph traversal and minimum spanning tree problems.

3.3 *Construction of heap and Data Structure used for creation.*

A heap can be constructed using an array-based implementation or a tree-based implementation.

In an array-based implementation, the heap is stored in an array where the first element of the array represents the root of the tree. The children of the root node are located at indices 2i+1 and 2i+2, where i is the index of the parent node. This allows for efficient traversal and manipulation of the heap, as the tree structure is implicitly defined by the indices of the array.

In a tree-based implementation, the heap is represented using a binary tree where each node has at most two child nodes. The heap property is maintained by ensuring that the parent node always has a higher priority than its children. This allows for more efficient insertion and deletion operations, as the tree structure can be easily manipulated by swapping nodes and adjusting their priorities.

The construction of a heap involves building the heap from an unordered set of elements. This is typically done using a bottom-up approach called heapify, where the elements are recursively swapped with their children until the heap property is satisfied. The heapify operation has a time complexity of O(n) and is commonly used to build a heap from an array of elements.

To summarize, the data structure used for the creation of a heap depends on the implementation approach. In an array-based implementation, the heap is stored in an array, while in a tree-based implementation, the heap is represented using a binary tree. The construction of a heap involves using the heapify operation to build the heap from an unordered set of elements.

4 Implementation

4.1 *Platform*

- 64-bit Mac OS
- Open Source C++ Programming tool like Visual Studio Code

subsection*Test Conditions*

1. Input min 10 elements.
2. Display Max and Min Heap
3. Find Maximum and Minimum marks obtained in a particular subject.

5 Conclusion

Thus, we have implemented Min and Max Heap.

6 FAQ's

6.1 *Discuss with suitable example for heap sort?*

Heap sort is a sorting algorithm that uses a heap data structure to sort an array of elements. It works by building a max heap from the array and repeatedly extracting the maximum element and placing it at the end of the array until all elements are sorted. The algorithm has a time complexity of O(n log n) and is an in-place sorting algorithm, meaning it sorts the array in situ without needing extra memory.

Let's take an example to illustrate the heap sort algorithm. Consider an unsorted array of integers:

```
[10, 8, 3, 7, 1, 9, 4, 2, 6, 5]
```

Step 1: Building a Max Heap We start by building a max heap from the array. This involves rearranging the elements of the array to satisfy the heap property, where the parent node has a higher priority than its children. We can start from the middle of the array and work our way up to the root, performing the heapify operation on each node. After building the max heap, the array would look like:

```
[10, 8, 9, 7, 6, 3, 4, 2, 1, 5]
```

Step 2: Sorting the Array We repeatedly extract the maximum element from the heap and place it at the end of the array until all elements are sorted. This involves swapping the root node (which is the maximum element) with the last element of the heap, reducing the heap size by one, and performing the heapify operation on the root node to maintain the heap property. After each iteration, the sorted portion of the array grows from right to left. The array after sorting would look like:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Thus, the heap sort algorithm has successfully sorted the unsorted array in ascending order.

6.2 *Compute the time complexity of heap sort?*

The time complexity of heap sort is O(n log n), where n is the number of elements in the input array. This is because the algorithm involves two main operations - building a heap and repeatedly extracting the maximum element from the heap.

The building of the heap takes O(n) time because each node of the heap needs to be heapified. Since there are n nodes in a heap, the total time taken for building the heap is O(n).

The repeated extraction of the maximum element takes O(log n) time because it involves swapping the root element with the last element in the heap and then performing the heapify operation on the new root to restore the heap property. Since we perform this operation n times, the total time taken for sorting is O(n log n).

Therefore, the time complexity of heap sort is O(n log n) in the worst case. Heap sort has a good worst-case time complexity and is often used in situations where we need to sort large datasets in-place.