For A Given Input Array A:3,2,1,6,4,8,5,9,7, What Is The Sequence Of Numbers In A After Calling Build-Max-Heap

For A Given Input Array A:3,2,1,6,4,8,5,9,7, What Is The Sequence Of Numbers In A After Calling Build-Max-Heap

Understanding how a binary heap transforms an array is fundamental to many efficient algorithms, particularly heapsort and priority queues. The process of building a max-heap from an arbitrary array involves reorganizing the elements so that each parent node is greater than or equal to its children. This process ensures that the largest element resides at the root of the heap, which simplifies maximum retrieval operations. In this article, we explore how the array [3, 2, 1, 6, 4, 8, 5, 9, 7] evolves after applying the build-max-heap procedure, detailing each step to provide a clear understanding of the heap construction process.

Understanding the Build-Max-Heap Algorithm

Before diving into the specific array transformation, it’s crucial to understand what build-max-heap does and how it operates.

What is a Max-Heap?

A max-heap is a complete binary tree where each parent node is greater than or equal to its children. This property ensures that the maximum element is always at the root of the tree.

Heap Representation in Arrays

In array representation, a binary heap is stored in a contiguous array where:
- The element at index 0 is the root.
- For any element at index i:
- Its left child is at index 2i + 1.
- Its right child is at index 2i + 2.
- Its parent is at index ⌊(i - 1) / 2⌋.

The Build-Max-Heap Procedure

The algorithm:
1. Starts from the last non-leaf node, moving upward to the root.
2. Applies a "heapify" process at each node, ensuring subtrees satisfy the max-heap

property.
3. After processing all nodes, the entire array becomes a max-heap.

The key insight is that leaf nodes are already heaps, so the process begins from the lowest internal nodes.

Initial Array and Indexing

Given the array:
- Index 0: 3
- Index 1: 2
- Index 2: 1
- Index 3: 6
- Index 4: 4
- Index 5: 8
- Index 6: 5
- Index 7: 9
- Index 8: 7

The total number of elements is 9. The last internal node is at index ⌊(n/2) - 1⌋ = ⌊(9/2) - 1⌋ = 3. So, the process begins from index 3 and moves up to index 0.

Step-by-Step Transformation of the Array

Let's walk through each step, applying "heapify" from the last internal node up to the root.

Heapify at Index 3

- Current element: 6
- Left child (index 7): 9
- Right child (index 8): 7

Compare with children:
- Largest among 6, 9, 7 is 9 at index 7.

Action:
- Swap 6 and 9.

Array after swap:
[3, 2, 1, 9, 4, 8, 5, 6, 7]

Now, heapify at index 7 (child swapped from index 3) — but since index 7 is a leaf, no further action is needed.

Heapify at Index 2

- Current element: 1
- Left child (index 5): 8
- Right child (index 6): 5

Compare:
- Largest among 1, 8, 5 is 8 at index 5.

Action:
- Swap 1 and 8.

Array after swap:
[3, 2, 8, 9, 4, 1, 5, 6, 7]

Heapify at index 5:
- Index 5 is a leaf, so no further action.

Heapify at Index 1

- Current element: 2
- Left child (index 3): 9
- Right child (index 4): 4

Compare:
- Largest among 2, 9, 4 is 9 at index 3.

Action:
- Swap 2 and 9.

Array after swap:
[3, 9, 8, 2, 4, 1, 5, 6, 7]

Heapify at index 3:
- Current element: 2
- Left child (index 7): 6
- Right child (index 8): 7

Compare:
- Largest among 2, 6, 7 is 7 at index 8.

Action:
- Swap 2 and 7.

Array after swap:
[3, 9, 8, 7, 4, 1, 5, 6, 2]

Heapify at index 8:
- Leaf node, no further action.

Heapify at Index 0

- Current element: 3
- Left child (index 1): 9
- Right child (index 2): 8

Compare:
- Largest among 3, 9, 8 is 9 at index 1.

Action:
- Swap 3 and 9.

Array after swap:
[9, 3, 8, 7, 4, 1, 5, 6, 2]

Heapify at index 1:
- Current element: 3
- Left child (index 3): 7
- Right child (index 4): 4

Compare:
- Largest among 3, 7, 4 is 7 at index 3.

Action:
- Swap 3 and 7.

Array after swap:
[9, 7, 8, 3, 4, 1, 5, 6, 2]

Heapify at index 3:
- Current element: 3
- Left child (index 7): 6
- Right child (index 8): 2

Compare:
- Largest among 3, 6, 2 is 6 at index 7.

Action:
- Swap 3 and 6.

Array after swap:
[9, 7, 8, 6, 4, 1, 5, 3, 2]

Heapify at index 7:
- Leaf node, no action needed.

Final Max-Heap Array:
After completing all heapify steps, the array representing the max-heap is:

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

This array now satisfies the max-heap property:
- The root (index 0): 9, greater than its children 7 and 8.
- Each parent node is larger than or equal to its children.

Summary of the Heapify Process

- The build-max-heap process involves starting from the last internal node and moving up to the root.
- At each step, the subtree rooted at the current node is adjusted to maintain the max-heap property.
- Swaps are made when a child exceeds the parent, and heapify is recursively applied to affected subtrees.
- The process ensures that, after completion, the entire array forms a valid max-heap.

Importance of Build-Max-Heap in Sorting Algorithms

The build-max-heap procedure is a crucial step in heapsort, an efficient comparison-based sorting algorithm. It allows the array to be transformed into a heap structure in linear time, enabling the extraction of maximum elements repeatedly to achieve sorted order. Understanding how the array evolves during build-max-heap provides insight into the overall heapsort process and highlights the importance of heap data structures in algorithm design.

Conclusion

Transforming the array [3, 2, 1, 6, 4, 8, 5, 9, 7] into a max-heap via the build-max-heap process results in the sequence [9, 7, 8, 6, 4, 1, 5, 3, 2]. This process involves systematic heapify operations starting from the last internal node and progressing upward to the root, ensuring the max-heap property is maintained throughout. Mastery of this procedure not only aids in understanding heapsort but also in grasping foundational data structures used in efficient algorithm implementations.

Frequently Asked Questions

What is the initial array provided for building the max heap?

The initial array is [3, 2, 1, 6, 4, 8, 5, 9, 7].

What is the purpose of calling Build-Max-Heap on an array?

Build-Max-Heap transforms the array into a max heap, where each parent node is greater than or equal to its children, which is essential for heap sort and priority queue operations.

At which index does the heap construction process start in the array?

The process starts from the last non-leaf node, which is at index (n/2) - 1; for this array, starting from index 3.

What is the sequence of steps involved in building the max heap from the array?

The process involves heapifying the subtrees starting from the last non-leaf node up to the root, ensuring each subtree satisfies the max heap property.

What is the resulting array after calling Build-Max-Heap on [3, 2, 1, 6, 4, 8, 5, 9, 7]?

The array becomes [9, 7, 8, 6, 4, 1, 5, 3, 2] after building the max heap.

How do the parent and child indices relate in the array representation of a heap?

For a node at index i, its left child is at 2i + 1, and its right child is at 2i + 2.

Why is the last non-leaf node important in the heap construction process?

Because all leaf nodes are already heaps, starting from the last non-leaf node ensures the entire array satisfies the max heap property after heapify operations.

Is the resulting sequence after Build-Max-Heap unique for the same input array?

Yes, given the same input array and implementation, the resulting max heap array will be consistent, although the exact arrangement might vary slightly depending on the heapify method used.

Additional Resources

Build-Max-Heap is a fundamental operation in the realm of heap data structures and is crucial for implementing efficient algorithms like Heap Sort, Priority Queues, and more.

When given a specific input array, such as A: 3, 2, 1, 6, 4, 8, 5, 9, 7, understanding how the Build-Max-Heap procedure transforms the array into a valid max-heap is essential for grasping the underlying mechanics of heap-based algorithms. In this comprehensive review, we will explore the step-by-step process of building a max-heap from the given array, analyze the resulting sequence, and discuss the implications, features, pros, and cons of the process.

Understanding the Input Array and Max-Heap Properties

Before diving into the process, it is vital to understand the initial array and what defines a max-heap.

Initial Array

The array provided is:
```
A: 3, 2, 1, 6, 4, 8, 5, 9, 7
```
Indices (assuming 0-based indexing):
- A[0] = 3
- A[1] = 2
- A[2] = 1
- A[3] = 6
- A[4] = 4
- A[5] = 8
- A[6] = 5
- A[7] = 9
- A[8] = 7

Max-Heap Properties

A max-heap is a complete binary tree where:
- The value of each node is greater than or equal to its children.
- The tree is complete, meaning all levels are fully filled except possibly the last, which is filled from left to right.

In array representation:
- For a node at index `i`, its children are at indices `2i + 1` (left) and `2i + 2` (right).
- The parent of node at index `i` is at `(i - 1) // 2`.

Step-by-Step Process of Build-Max-Heap

The Build-Max-Heap algorithm involves "heapifying" each non-leaf node from the bottom up. For an array of size `n`, the process starts from the last internal node at index `(n//2) - 1` and moves upward to the root at index 0.

Step 1: Identify the last non-leaf node

Given `n = 9`, last internal node:
```
(n//2) - 1 = (9//2) - 1 = 4 - 1 = 3
```
So, we start heapifying from index 3 and move up to index 0.

Initial Array (before heapification):
```
Index: 0 1 2 3 4 5 6 7 8
Value: 3, 2, 1, 6, 4, 8, 5, 9, 7
```

Heapify at index 3

- Current node: 6 (index 3)
- Children:
- Left child (index 7): 9
- Right child (index 8): 7

Comparison:
- Largest among 6, 9, 7 is 9 at index 7.

Action:
- Swap 6 and 9.

Updated array:
```
Index: 0 1 2 3 4 5 6 7 8
Value: 3, 2, 1, 9, 4, 8, 5, 6, 7
```

Heapify at index 7:
- Children of index 7:
- Left: 15 (non-existent, as 27+1=15 > 8)

- Right: 16 (non-existent)
- No children, so heapify ends here.

Heapify at index 2

- Current node: 1 (index 2)
- Children:
- Left (index 5): 8
- Right (index 6): 5

Comparison:
- Largest among 1, 8, 5 is 8 at index 5.

Action:
- Swap 1 and 8.

Updated array:
```
Index: 0 1 2 3 4 5 6 7 8
Value: 3, 2, 8, 9, 4, 1, 5, 6, 7
```

Heapify at index 5:
- Children:
- Left (11): out of bounds
- Right (12): out of bounds
- No children, heapify ends.

Heapify at index 1

- Current node: 2 (index 1)
- Children:
- Left (3): 9
- Right (4): 4

Comparison:
- Largest among 2, 9, 4 is 9 at index 3.

Action:
- Swap 2 and 9.

Updated array:

```
Index: 0 1 2 3 4 5 6 7 8
Value: 3, 9, 8, 2, 4, 1, 5, 6, 7
```

Heapify at index 3:
- Children:
- Left (7): 6
- Right (8): 7

Comparison:
- Largest among 2, 6, 7 is 7 at index 8.

Action:
- Swap 2 and 7.

Updated array:
```
Index: 0 1 2 3 4 5 6 7 8
Value: 3, 9, 8, 7, 4, 1, 5, 6, 2
```

Heapify at index 8:
- No children, process ends here.

Heapify at index 0 (root)

- Current node: 3
- Children:
- Left (1): 9
- Right (2): 8

Comparison:
- Largest among 3, 9, 8 is 9 at index 1.

Action:
- Swap 3 and 9.

Updated array:
```
Index: 0 1 2 3 4 5 6 7 8
Value: 9, 3, 8, 7, 4, 1, 5, 6, 2
```

Heapify at index 1:
- Children:
- Left (3): 7

- Right (4): 4

Comparison:
- Largest among 3, 7, 4 is 7 at index 3.

Action:
- Swap 3 and 7.

Updated array:
```
Index: 0 1 2 3 4 5 6 7 8
Value: 9, 7, 8, 3, 4, 1, 5, 6, 2
```

Heapify at index 3:
- Children:
- Left (7): 6
- Right (8): 2

Comparison:
- Largest among 3, 6, 2 is 6 at index 7.

Action:
- Swap 3 and 6.

Final array:
```
Index: 0 1 2 3 4 5 6 7 8
Value: 9, 7, 8, 6, 4, 1, 5, 3, 2
```

At this point, the heapification process is complete, and the array now satisfies the max-heap property.

Resulting Sequence After Build-Max-Heap

The sequence of numbers in the array after calling Build-Max-Heap on the input array is:

```
9, 7, 8, 6, 4, 1, 5, 3, 2
```

This array represents a valid max-heap, with each parent node greater than or equal to its children, and the structure maintains the complete binary tree property.

Features, Pros, and Cons of Build-Max-Heap

Features
- Converts an unordered array into a max-heap efficiently.
- Runs in O(n) time complexity, which is optimal for heap construction.
- Is a foundational step in Heap Sort and other priority queue implementations.

Pros
- Efficient: The bottom-up approach minimizes unnecessary swaps.
- In-place: Does not require extra space, as it modifies the array directly.

For A Given Input Array A 3 2 1 6 4 8 5 9 7 What Is The Sequence Of Numbers In A After Calling Build Max Heap

For A Given Input Array A 3 2 1 6 4 8 5 9 7 What Is The Sequence Of Numbers In A After Calling Build Max Heap

Related Articles

- Drag And Drop The Economic Traits To The Correct Country. Among The World's Leading Producer Of Coffee.Fishing
- Consider The Following Vectors In R5: A = (0,1,2,2,0), A2 = (1,1,4,0,0), A3 = (1,2,6,2,1), A4 = (-1,2,2,6,-1),
- If You Dissolve 20g Of NaOH In A Total Final Volume Of1Liter Of Deionized Water,what Is The Molar Concentration

Back to Home