Consider The Array A=30,10,15,9,7,50,8,22,5,3. 1) (5 Points) Write A After Calling The Function BUILD-MAX-HEAP(A)

Consider The Array A=30,10,15,9,7,50,8,22,5,3. 1) (5 Points) Write A After Calling The Function BUILD-MAX-HEAP(A)

Introduction to BUILD-MAX-HEAP and Its Significance

Building a max-heap from an unsorted array is a fundamental operation in the realm of data structures and algorithms, especially in the context of heap sort. The process transforms an arbitrary array into a binary heap where each parent node is greater than or equal to its child nodes. This property ensures efficient retrieval of the maximum element, which is pivotal in priority queues, sorting algorithms, and several other computational applications. Understanding how to convert an array into a max-heap using the BUILD-MAX-HEAP function provides insight into the internal mechanics of heaps and their practical utility.

Initial Array and Its Structure

Let's examine the initial array:

- A = [30, 10, 15, 9, 7, 50, 8, 22, 5, 3]

Assuming the array is 1-based indexed (common in algorithm pseudocode), the elements correspond to nodes in a binary tree representation:

```
Index:  1   2   3   4   5   6   7   8   9   10
Value: 30  10  15   9   7  50   8  22   5    3
```

This array can be visualized as a binary tree:

- The root is at index 1 (value 30).
- Left child of node at index i: 2i

- Right child of node at index i: 2i + 1

The initial tree structure:

```
30
/ \
10 15
/ \ / \
9 7 50 8
/ \
22 5
/
3
```

In practice, the array is stored in contiguous memory, but conceptual visualization aids understanding the heapification process.

Understanding BUILD-MAX-HEAP Algorithm

The BUILD-MAX-HEAP function works by applying the MAX-HEAPIFY procedure starting from the lowest non-leaf nodes up to the root. The process ensures that each subtree satisfies the max-heap property:

- For each node, compare it with its children.
- If a child's value is greater than the parent's, swap them.
- Recursively ensure subtrees are max-heaps after swaps.

The algorithm proceeds in a bottom-up manner, starting from the last parent node:

- The last parent node is at index $\lfloor n/2 \rfloor$, where n is the number of elements.
- For our array with 10 elements, last parent index is at $\lfloor 10/2 \rfloor = 5$.

The process involves:

1. Applying MAX-HEAPIFY at index 5
2. Moving upward to index 4
3. Then index 3
4. Then index 2
5. Finally, index 1

This ensures all subtrees become max-heaps, culminating in the entire array satisfying the max-heap property.

Step-by-Step Heapification Process

Let's detail the process at each relevant node:

Heapify at Index 5 (Value 7)

- Children: left at 10 (value 3), right at 11 (none, since array length is 10)
- Compare 7 with 3
- 7 > 3, so no swap needed; subtree rooted at 5 is already a max-heap

Heapify at Index 4 (Value 9)

- Children: 8 (index 8, value 22), 9 (index 9, value 5)
- Compare 9 with children: 22 and 5
- 22 > 9, swap 9 and 22
- Array after swap:

`[30, 10, 15, 22, 7, 50, 8, 9, 5, 3]`

- Now, at index 8 (value 9), check children: index 16 and 17 — out of bounds, so stop

Heapify at Index 3 (Value 15)

- Children: 6 (value 50), 7 (value 8)
- Compare 15 with children: 50 and 8
- 50 > 15, swap 15 and 50

- Array after swap:

`[30, 10, 50, 22, 7, 15, 8, 9, 5, 3]`

- Now, at index 6 (value 15), check children: 12 (value 22), 13 (none)
- 22 > 15, swap 15 and 22

- Array:

`[30, 10, 50, 22, 7, 22, 8, 9, 5, 3]`

- Wait, but note that after swapping, the subtree rooted at index 6 (value 15) now has 22, which is greater than 15. Since 22 is now at index 6, check its children:

- Children: 12 (value 22), 13 (none).

- But after swap, index 6 has 22, and its children are:

- left: 12 (value 22)

- right: 13 (none)

- Since 22 at index 6 is greater than its children, no further swaps needed.

Heapify at Index 2 (Value 10)

- Children: 4 (value 22), 5 (value 7)
- Compare 10 with children: 22 and 7
- 22 > 10, swap 10 and 22

- Array:

`[30, 22, 50, 10, 7, 15, 8, 9, 5, 3]`

- Now, at index 4 (value 10), check children: 8 (value 9), 9 (value 5)

- 10 > 9 and 10 > 5, no further swaps.

Heapify at Index 1 (Value 30)

- Children: 2 (value 22), 3 (value 50)
- Compare 30 with children: 22 and 50
- 50 > 30, swap 30 and 50

- Array:

`[50, 22, 30, 10, 7, 15, 8, 9, 5, 3]`

- Now, at index 3 (value 30), check children: 6 (value 15), 7 (value 8)

- 30 > 15 and 30 > 8, no swaps needed

- At index 2 (value 22), check children: 4 (value 10), 5 (value 7)

- 22 > 10 and 22 > 7, no swaps needed

- At index 1 (value 50), check children: 2 (value 22), 3 (value 30)

- 50 > 22 and 50 > 30, no swaps needed

The heapification process completes here.

Final Max-Heap Array

After applying the BUILD-MAX-HEAP procedure, the array transforms into:

- **[50, 22, 30, 10, 7, 15, 8, 9, 5, 3]**

This array satisfies the max-heap property:

- For every parent node, the value is greater than or equal to its children.

Visualizing the heap:

```
50
/ \
22 30
/ \ / \
10 7 15 8
/ \
9 5
/
3
```

Implications and Applications of the Final Max-Heap

Transforming the array into a max-heap has significant computational benefits:

- **Efficient Extraction:** The maximum element (root) can be retrieved in O(1) time.
- **Heap Sort:** Repeatedly extracting the maximum and rebuilding the heap sorts the array in O(n log n) time.
- **Priority Queues:** Max-heaps underpin priority queue implementations, allowing for quick priority updates and retrievals.

Understanding the internal steps of BUILD-MAX-HEAP helps in optimizing algorithms and grasping the underlying data structure principles.

Conclusion

The process of converting an arbitrary array into a max-heap using the BUILD-MAX-HEAP procedure involves systematic heapification starting from the last non-leaf node up to the root. For the array A=30,10,15,9,7,50,8,22

Frequently Asked Questions

What is the resulting array after building a max-heap from the array A = [30, 10, 15, 9, 7, 50, 8, 22, 5, 3] using the BUILD-MAX-HEAP function?

[50, 30, 15, 22, 10, 8, 5, 9, 7, 3]

How does the BUILD-MAX-HEAP procedure transform the input array A = [30, 10, 15, 9, 7, 50, 8, 22, 5, 3]?

It rearranges the array elements to satisfy the max-heap property, resulting in the array [50, 30, 15, 22, 10, 8, 5, 9, 7, 3], where each parent is greater than or equal to its children.

Which element becomes the root of the max-heap after calling BUILD-MAX-HEAP on array A?

The element 50 becomes the root of the max-heap.

What is the key process performed during BUILD-MAX-HEAP to ensure the array satisfies the max-heap property?

The process involves 'heapifying' subtrees starting from the lowest non-leaf nodes up to the root, swapping elements as needed to maintain the max-heap condition.

Is the array A = [30, 10, 15, 9, 7, 50, 8, 22, 5, 3] already a max-heap before calling BUILD-MAX-HEAP?

No, it is not a max-heap initially; the BUILD-MAX-HEAP function rearranges it to satisfy the max-heap property.

What is the significance of building a max-heap in

algorithms like Heapsort?

Building a max-heap allows efficient extraction of the maximum element and is fundamental to the heapsort algorithm, enabling sorting in O(n log n) time.

Additional Resources

Consider The Array A=30,10,15,9,7,50,8,22,5,3. 1) (5 Points) Write A After Calling The Function BUILD-MAX-HEAP(A)

Introduction

The process of transforming an unordered array into a max-heap is fundamental in the field of computer science, particularly within sorting algorithms such as heapsort. The array under consideration, A = [30, 10, 15, 9, 7, 50, 8, 22, 5, 3], presents an intriguing case for understanding the mechanics and implications of the BUILD-MAX-HEAP procedure. This article explores the step-by-step transformation, elucidating the underlying logic, potential pitfalls, and the significance of the resulting max-heap in algorithm design.

Understanding Max-Heap and BUILD-MAX-HEAP

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 largest element resides at the root of the tree, facilitating efficient retrieval of maximum values.

Key Properties:
- Complete binary tree structure
- Parent node ≥ child nodes
- Efficient for priority queue implementation

The BUILD-MAX-HEAP Algorithm

BUILD-MAX-HEAP transforms an unordered array into a max-heap by applying the max-heapify procedure starting from the lowest non-leaf nodes up to the root.

High-Level Steps:
1. Identify the last non-leaf node (at index ⌊n/2⌋).
2. For each node from this index down to the root:
- Apply max-heapify to ensure the subtree rooted at that node satisfies the max-heap property.

This bottom-up approach guarantees the entire array satisfies heap properties

efficiently, typically in O(n) time.

Initial Array and Indexing

Given array A:

Index	Element
1	30
2	10
3	15
4	9
5	7
6	50
7	8
8	22
9	5
10	3

Note: Arrays in typical heap implementations are 1-indexed for clarity.

The total number of elements, n = 10.

Step-by-Step Transformation

Step 1: Identify the last non-leaf node

- Last non-leaf node is at index ⌊n/2⌋ = ⌊10/2⌋ = 5.

Nodes from index 5 down to 1 will be heapified.

Step 2: Heapify nodes from index 5 to 1

Heapify at index 5

- Element: 7
- Children at indices 10 and 11 (since 2i and 2i + 1): 10 and 11. But index 11 doesn't exist, so only left child at index 10 (3).

- Compare 7 with its child 3:
- 7 ≥ 3, so max-heap property holds.
- No swap needed.

Heap remains unchanged at index 5

Heapify at index 4

- Element: 9
- Children at indices 8 and 9: 22 and 5

- Compare 9 with children:
- Largest among 9, 22, 5 is 22 at index 8.
- Swap 9 and 22:

Array after swap:

A = [30, 10, 15, 22, 7, 50, 8, 9, 5, 3]

- Now, heapify at index 8 (child):

- Element: 9
- Children: index 16 and 17 (none), so no further action.

Result after heapify at index 4:

A = [30, 10, 15, 22, 7, 50, 8, 9, 5, 3]

Heapify at index 3

- Element: 15
- Children at indices 6 and 7: 50 and 8

- Largest among 15, 50, 8 is 50 at index 6.
- Swap 15 and 50:

A = [30, 10, 50, 22, 7, 15, 8, 9, 5, 3]

- Heapify at index 6:

- Element: 15
- Children: index 12 and 13 (none), so no further action.

Result after heapify at index 3:

A = [30, 10, 50, 22, 7, 15, 8, 9, 5, 3]

Heapify at index 2

- Element: 10
- Children: indices 4 and 5: 22 and 7

- Largest among 10, 22, 7 is 22 at index 4.
- Swap 10 and 22:

A = [30, 22, 50, 10, 7, 15, 8, 9, 5, 3]

- Heapify at index 4:

- Element: 10
- Children: indices 8 and 9: 9 and 5

- Largest among 10, 9, 5 is 10.

- No further swaps.

Result after heapify at index 2:

A = [30, 22, 50, 10, 7, 15, 8, 9, 5, 3]

Heapify at index 1

- Element: 30
- Children at indices 2 and 3: 22 and 50

- Largest among 30, 22, 50 is 50 at index 3.
- Swap 30 and 50:

A = [50, 22, 30, 10, 7, 15, 8, 9, 5, 3]

- Heapify at index 3:

- Element: 30
- Children at indices 6 and 7: 15 and 8

- Largest among 30, 15, 8 is 30; no swap needed.

Final Max-Heap Array

After completing all heapify steps, the array A becomes:

A = [50, 22, 30, 10, 7, 15, 8, 9, 5, 3]

Deep Dive: The Underlying Mechanics

Visualizing the Heap Structure

Represented as a binary tree, the max-heap looks like:

```
50
/ \
22 30
/ \ / \
10 7 15 8
/ \
9 5
/
3
```

This structure conforms to the complete binary tree property and satisfies the max-heap property at every node.

The Significance of Bottom-Up Heapification

- The process ensures that subtrees are heapified before moving upward.
- This bottom-up approach guarantees linear time complexity, O(n), making it efficient even for large datasets.
- It prevents unnecessary swaps by fixing smaller subtrees first.

Practical Implications and Applications

BUILD-MAX-HEAP is not just an academic exercise but a cornerstone in various algorithms:

- Heapsort: Sorting array in-place with guaranteed O(n log n) performance.
- Priority Queues: Efficiently managing dynamic datasets where quick maximum retrieval is necessary.
- Graph Algorithms: Dijkstra's algorithm uses heaps for shortest path computations.

Understanding how BUILD-MAX-HEAP transforms an array provides foundational insight into these applications' efficiency and reliability.

Concluding Remarks

Transforming the array A = [30, 10, 15, 9, 7, 50, 8, 22, 5, 3] into a max-heap via BUILD-MAX-HEAP illustrates the elegance of bottom-up heapification. It demonstrates how local adjustments at individual nodes propagate upward to produce a globally valid heap structure. This process exemplifies the power of recursive, hierarchical algorithms in organizing data efficiently, which underpins some of the most optimized sorting and priority management techniques in computer science.

By meticulously analyzing each step, from initial heapify operations to the final maximally ordered array, practitioners gain a deeper appreciation for the algorithm's robustness, versatility, and critical role in advanced computational tasks.

References:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
- Knuth, D. E. (1998). The Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley.
- Sedgewick, R., & Wayne, K. (2011). Algorithms (4th Edition). Addison-Wesley.

Consider The Array A 30 10 15 9 7 50 8 22 5 3 1 5 Points Write A After Calling The Function Build Max Heap A

Consider The Array A 30 10 15 9 7 50 8 22 5 3 1 5 Points Write A After Calling The Function Build Max Heap A

Related Articles

- The Process Of Allocating Electoral Seats To Geographical Areas Is Known As Group Of Answer Choices Gerrymandering.
- Consider The Analog Signal Xa(t) = 6cos(600t)1.) Determine The Minimum Sampling Rate Required To Avoidaliasing.2.)
- What Is The Maximum Time For Last Known Normal When Endovascular Therapy Can Be Performed?a.3hrsb.12hrsc.6hrsd.24hrs

Back to Home