CSC263 Tutorial #2 Exercises

Analyzing the Max-Heap Data Structure

Alexander He Meng

Prepared for January 17, 2025

Question #1

Claim: The *minimum* element of a binary max-heap must be one of its leaf nodes (assuming distinct elements).

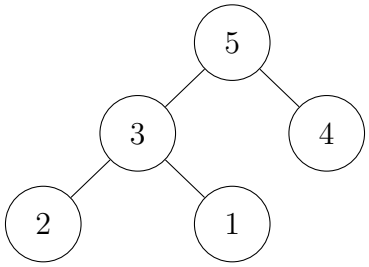
Proof.
Seeking a contradiction, assume the minimum element of a binary max-heap is not a leaf node. Then, it must have at least one child node.
Since this element is the minimum, that child node must contain a larger element, and this contradicts the property of a binary max-heap where parent nodes are larger than their children.
By this contradiction, the minimum element of a binary max-heap must be one of its leaf nodes.

□

Question #2

Claim: The *median* element of a binary max-heap is **not necessarily** one of its leaf nodes.

Proof.
Consider the binary max-heap `max_heap = [5, 3, 4, 2, 1]`, represented as:



Clearly, the median element is `3`, yet it is not a leaf node.

□

Question #3

A ternary max-heap is like a binary max-heap except that non-leaf nodes have three children instead of two children. (As with binary heaps, there can of course be one non-leaf node that has fewer than three children.) We refer to the children as the left child, middle child, and right child.

The values stored in the nodes are ordered according to the same principle as for binary max-heaps: the value at each node is greater than or equal to the values in the node's children. Answer the following questions regarding ternary max-heaps:

(a)

Claim: In a ternary max-heap, the indices of the left, middle, and right children are $3i + 1$, $3i + 2$, and $3i + 3$, respectively.

Proof.
Given the array representation of the ternary heap, for any node at index i:

- The left child is located at $3i + 1$.
- The middle child is located at $3i + 2$.
- The right child is located at $3i + 3$.

These formulas arise from the structure of the array representation of a complete ternary tree. For corner cases:

- If $3i + 1$, $3i + 2$, or $3i + 3$ exceed the array length, the node has fewer than three children.

□

(b)

Claim: In a ternary max-heap, `EXTRACT-MAX` works by removing the root node and replacing it with the last element of the heap, then "bubbling down" to restore the max-heap property. `INSERT` works by appending the new element to the end of the array and "bubbling up" to restore the max-heap property.

Proof.
The differences from binary max-heaps are primarily in handling three children instead of two during the bubbling operations. For `EXTRACT-MAX`, during bubbling down, the parent node is swapped with the largest of its three children if it violates the max-heap property. For `INSERT`, during bubbling up, the inserted node is swapped with its parent if it is larger than the parent. □

(c)

Recursive Implementation:

```python
IS_TERNARY_MAX_HEAP(A: list[int], i: int) -> bool:
    if 3i + 1 >= len(A):  # Node has no children
        return True

    left = 3i + 1
    middle = 3i + 2
    right = 3i + 3

    # Check max-heap property
    if (left < len(A) and A[i] < A[left]) or \
       (middle < len(A) and A[i] < A[middle]) or \
       (right < len(A) and A[i] < A[right]):
        return False

    return IS_TERNARY_MAX_HEAP(A, left) and \
           IS_TERNARY_MAX_HEAP(A, middle) and \
           IS_TERNARY_MAX_HEAP(A, right)
```

Explanation: The function checks the max-heap property at the current node and recursively verifies the children. The worst-case runtime is $O(n)$.

(d)

Iterative Implementation:

```python
IS_TERNARY_MAX_HEAP_ITER(A: list[int]) -> bool:
    for i in range(len(A)):
        left = 3i + 1
        middle = 3i + 2
        right = 3i + 3

        if (left < len(A) and A[i] < A[left]) or \
           (middle < len(A) and A[i] < A[middle]) or \
           (right < len(A) and A[i] < A[right]):
            return False

    return True
```

Explanation: The iterative approach checks each node in a loop. The worst-case runtime is $O(n)$.