Heaps Cheatsheet

Topic Overview

Heaps in Java are tree-based structures maintaining the heap property, used for priority queues. This cheatsheet covers heap-based techniques.

Prerequisites

Trees

List of Subtopics

- Min Heap
- Max Heap
- Heapify
- Insert Operation
- Delete Min/Max
- Heap Sort
- Median in a Stream
- Top K Elements
- Merge K Sorted Arrays
- Kth Largest Element

Key Concepts Explained

- **Min Heap**: Parent is smaller than children, used for finding minimum.
- **Max Heap**: Parent is larger than children, used for maximum.
- **Heapify**: Converts an array into a heap.

Approaches to Solve Problems with Step-by-Step Algorithms

- **Min Heap**:
  - **Algorithm**:
    1. Use an array, parent at i, children at 2i+1, 2i+2.
    2. Ensure parent is smaller than children.
  - **Context**: O(1) access to min, O(log n) for modifications.
- **Max Heap**:
  - **Algorithm**:
    1. Use array with same indexing, ensure parent > children.
  - **Context**: O(1) access to max, O(log n) modifications.
- **Heapify**:
  - **Algorithm**:
    1. Start from last non-leaf node (n/2 - 1).
    2. Compare with children, swap with smaller/larger, recurse down.
    3. Repeat for all nodes.
  - **Context**: O(n) time to build heap.
- **Insert Operation**:
  - **Algorithm**:
    1. Add element at end, increase size.
    2. Heapify up by swapping with parent if smaller/larger.
  - **Context**: O(log n) time.
- **Delete Min/Max**:
  - **Algorithm**:
    1. Replace root with last element, decrease size.
    2. Heapify down by comparing with children.
  - **Context**: O(log n) time.
- **Heap Sort**:
  - **Algorithm**:
    1. Build max heap from array.
    2. Repeatedly delete max, place at end, reduce heap size.
  - **Context**: O(n log n) time, O(1) extra space.
- **Median in a Stream**:

  - **Algorithm**:
    1. Use two heaps: max heap for lower half, min heap for upper.
    2. Balance heaps, add to appropriate heap, adjust if needed.
  - **Context**: O(log n) per addition, O(1) median.

- **Top K Elements**:
  - **Algorithm**:
    1. Use min heap of size k.
    2. Add elements, remove smallest if size exceeds k.
  - **Context**: O(n log k) time.

- **Merge K Sorted Arrays**:
  - **Algorithm**:
    1. Use min heap to store one element per array.
    2. Extract min, add next from same array, repeat.
  - **Context**: O(n log k) time, n total elements, k arrays.

- **Kth Largest Element**:
  - **Algorithm**:
    1. Use min heap of size k.
    2. Add all elements, kth largest is heap minimum.
  - **Context**: O(n log k) time.

Common LeetCode Problems with Approaches

- **Kth Largest Element in an Array (215)**: Use min heap of size k.
- **Find Median from Data Stream (295)**: Use two heaps for median.
- **Merge k Sorted Lists (23)**: Use min heap for merging.
-
- **Top K Frequent Elements (347)**: Use heap with frequency.

Time & Space Complexities

- Insert/Delete: O(log n)
- Build Heap: O(n)
- Space: O(n)

Important Tips & Tricks

- Use arrays for heap implementation to save space.
- Balance heaps for median calculations.
- Optimize k-related problems with heap size limits.
- Handle edge cases like empty heaps.
- Use iterative heapify for better cache performance.