Advanced Topics Cheatsheet

Topic Overview

Advanced Topics in Java include complex algorithms and data structures. This cheatsheet covers advanced techniques.

Prerequisites

All previous topics

List of Subtopics

- Segment Tree
- Fenwick Tree (Binary Indexed Tree)
- Trie
- Suffix Array
- K-D Tree
- Disjoint Set Union (DSU)
- A* Search
- String Matching with Wildcards
- Rolling Hash
- Parallel Algorithms

Key Concepts Explained

- **Segment Tree**: Efficient range queries and updates.
- **Trie**: Tree for string storage and retrieval.
- **A* Search**: Heuristic-based pathfinding.

Approaches to Solve Problems with Step-by-Step Algorithms

- **Segment Tree**:
  - **Algorithm**:
    1. Build tree by recursively dividing array into halves.
    2. Store aggregates (sum, min) at each node.
    3. Update/query using range and node traversal.
  - **Context**: O(n) build, O(log n) query/update.
- **Fenwick Tree (Binary Indexed Tree)**:
  - **Algorithm**:
    1. Use array, update with least significant bit.
    2. Query prefix sum with bit manipulation.
  - **Context**: O(log n) update/query, O(n) space.
- **Trie**:
  - **Algorithm**:
    1. Create node with children array, end marker.
    2. Insert by adding characters as child nodes.
    3. Search by traversing character path.
  - **Context**: O(m) per operation, m is string length.
- **Suffix Array**:
  - **Algorithm**:
    1. Sort all suffixes, use comparison.
    2. Build with induced sorting or DC3.
  - **Context**: O(n log n) time, O(n) space.
- **K-D Tree**:
  - **Algorithm**:
    1. Build by splitting on alternate dimensions.
    2. Search/insert with nearest neighbor checks.
  - **Context**: O(log n) average, O(n) worst.
- **Disjoint Set Union (DSU)**:
  - **Algorithm**:
    1. Use array for parent, rank for optimization.
    2. Union by rank, find with path compression.
  - **Context**: O((n)) amortized time, O(n) space.

- **A* Search**:
  - **Algorithm**:
    1. Use priority queue with f(n) = g(n) + h(n).
    2. Expand node with minimum f, update paths.
- **Context**: O(b^d)$time, bisbranchingfactor$.
- **String Matching with Wildcards**:
  - **Algorithm**:
    1. Use DP or recursion with wildcard checks.
    2. Match '*' as any sequence, '?' as any char.
  - **Context**: O(mn) time, O(mn) space.
- **Rolling Hash**:
  - **Algorithm**:
    1. Use polynomial hashing with prime modulus.
    2. Update hash by removing left, adding right.
  - **Context**: O(n) preprocessing, O(1) per slide.
- **Parallel Algorithms**:
  - **Algorithm**:
    1. Divide task into independent subtasks.
    2. Use threads or fork-join for parallel execution.
  - **Context**: O(n/p) time with p processors.

Common LeetCode Problems with Approaches

- **Range Sum Query - Mutable (307)**: Use segment tree.
-
- **Word Break II (140)**: Use trie for prefix matching.
-
- **K Closest Points to Origin (973)**: Use K-D tree.

Time & Space Complexities

- Varies: O(log n) to O(n log n)
- Space: O(n) to O(nš)

Important Tips & Tricks

- Use segment trees for range queries.
- Optimize DSU with path compression.
- Choose good heuristics for A*.
- Handle collisions in rolling hash.
- Parallelize independent computations.