Stacks Cheatsheet

Topic Overview

Stacks in Java are LIFO (Last In, First Out) structures, useful for managing function calls and backtracking. This cheatsheet details stack-based techniques.

Prerequisites

Arrays, Linked List

List of Subtopics

- Array-Based Stack
- Linked List-Based Stack
- Push Operation
- Pop Operation
- Peek Operation
- Check Empty
- Balanced Parentheses
- Next Greater Element
- Prefix to Infix Conversion
- Stock Span Problem

Key Concepts Explained

- **Array-Based Stack**: Uses a fixed-size array with a top pointer, efficient for small datasets.
- **Linked List-Based Stack**: Dynamic size using nodes, ideal for large or unpredictable data.
- **Balanced Parentheses**: Verifies matching pairs using stack operations.

Approaches to Solve Problems with Step-by-Step Algorithms

- **Array-Based Stack**:
  - **Algorithm**:
    1. Initialize an array and a top index (e.g., -1 for empty).
    2. Set a maximum size to prevent overflow.
    3. Use top to track the last element's position.
  - **Context**: O(1) time for operations, O(n) space.
- **Linked List-Based Stack**:
  - **Algorithm**:
    1. Create a node class with data and next pointer.
    2. Use a top pointer initialized to null.
    3. Operations modify the top node directly.
  - **Context**: O(1) time, O(n) space with dynamic growth.
- **Push Operation**:
  - **Algorithm**:
    1. Check if stack is full (array) or create new node (linked list).
    2. Increment top and add element at top+1 (array) or set new node as top (linked list).
  - **Context**: O(1) time.
- **Pop Operation**:
  - **Algorithm**:
    1. Check if stack is empty, return error if true.
    2. Retrieve element at top, decrement top (array) or update top to next (linked list).
    3. Free the removed node (linked list).
  - **Context**: O(1) time.
- **Peek Operation**:
  - **Algorithm**:
    1. Check if stack is empty, return error if true.
    2. Return the element at top without modifying it.
  - **Context**: O(1) time.
- **Check Empty**:
  - **Algorithm**:
    1. Return true if top is -1 (array) or null (linked list).

- **Context**: O(1) time.

- **Balanced Parentheses**:
  - **Algorithm**:
    1. Initialize an empty stack.
    2. For each character, push opening brackets, pop on closing if matching.
    3. If stack is empty at end, parentheses are balanced.
  - **Context**: O(n) time, O(n) space.

- **Next Greater Element**:
  - **Algorithm**:
    1. Use a stack to store indices with decreasing elements.
    2. For each element, pop from stack while top is smaller, set next greater.
    3. Push current index onto stack.
  - **Context**: O(n) time, O(n) space.

- **Prefix to Infix Conversion**:
  - **Algorithm**:
    1. Reverse the prefix expression.
    2. Use a stack to push operands, pop two on operators to form expressions.
    3. Reverse the final expression.
  - **Context**: O(n) time, O(n) space.

- **Stock Span Problem**:
  - **Algorithm**:
    1. Use a stack to store indices of decreasing prices.
    2. For each day, pop while stack top has lower price, span is distance to new top.
    3. Push current index.
  - **Context**: O(n) time, O(n) space.

Common LeetCode Problems with Approaches

- **Valid Parentheses (20)**: Use stack to match opening and closing brackets.
- **Next Greater Element I (496)**: Use stack to find next greater for each element.
- **Evaluate Reverse Polish Notation (150)**: Use stack to evaluate postfix expressions.
- **Min Stack (155)**: Maintain a secondary stack for minimums.
- **Largest Rectangle in Histogram (84)**: Use stack for height-based rectangle calculation.

Time & Space Complexities

- Push/Pop/Peek: O(1)
- Search: O(n)
- Space: O(n)

Important Tips & Tricks

- Use linked lists for dynamic stacks to avoid size limits.
- Handle stack overflow/underflow with checks.
- Optimize space with a single stack for multiple tasks.
- Use stacks for recursive problem simulation.
- Precompute where possible to reduce stack operations.