Backtracking Cheatsheet

Topic Overview

Backtracking in Java explores all possibilities by building solutions incrementally. This cheatsheet covers backtracking techniques.

Prerequisites

Arrays, Recursion

List of Subtopics

- N-Queens Problem
- Sudoku Solver
- Permutations
- Combinations
- Subset Sum
- Hamiltonian Cycle
- Rat in a Maze
- Word Search
- Generate Parentheses
- Knight's Tour

Key Concepts Explained

- **N-Queens Problem**: Places queens on a chessboard with no attacks.
- **Backtracking**: Builds solution, backtracks on failure.
- **Generate Parentheses**: Creates valid parenthesis combinations.

Approaches to Solve Problems with Step-by-Step Algorithms

- **N-Queens Problem**:
  - **Algorithm**:
    1. Use a board, place queen in first row.
    2. Check for attacks, move to next row if safe.
    3. Backtrack if no position works.
  - **Context**: O(n!) time, O(nš) space.
- **Sudoku Solver**:
  - **Algorithm**:
    1. Find empty cell, try digits 1-9.
    2. Check row, column, 3x3 box for validity.
    3. Recurse, backtrack on failure.
- **Context**: O($9^n)time, O(n)space$.
- **Permutations**:
  - **Algorithm**:
    1. Use a set to track used elements.
    2. Recurse with current permutation.
    3. Backtrack by removing element.
  - **Context**: O(n!) time, O(n) space.
- **Combinations**:
  - **Algorithm**:
    1. Start with empty combination.
    2. Add element, recurse with remaining.
    3. Backtrack to try other elements.
  - **Context**: O(C(n,k)) time, O(k) space.
- **Subset Sum**:
  - **Algorithm**:
    1. Recurse with current sum, include/exclude element.
    2. Backtrack if sum exceeds target.
- **Context**: O($2^n)time, O(n)space$.
- **Hamiltonian Cycle**:
  - **Algorithm**:
    1. Start at node, visit all, check return.

        2. Backtrack if cycle not possible.

    - **Context**: O(n!) time, O(n) space.

- **Rat in a Maze**:

    - **Algorithm**:

        1. Use DFS from start, move up/down/left/right.
        2. Mark visited, backtrack if no path.

- **Context**: O($3^n)time, O(n)space.$

- **Word Search**:

    - **Algorithm**:

        1. DFS from each cell, match word chars.
        2. Backtrack if no match or out of bounds.

- **Context**: O(n*m*$4^l)time, O(l)space.$

- **Generate Parentheses**:

    - **Algorithm**:

        1. Use recursion with open/close counts.
        2. Add '(' if open < n, ')' if close < open.
        3. Backtrack when counts equal n.

- **Context**: O($4^n/n)time, O(n)space.$

- **Knight's Tour**:

    - **Algorithm**:

        1. Start at position, try all moves.
        2. Backtrack if no valid next move.

- **Context**: O($8^n)time, O(n)space.$

Common LeetCode Problems with Approaches

- **N-Queens (51)**: Use backtracking to place queens.
- **Word Search (79)**: DFS with backtracking for word match.
-
- **Generate Parentheses (22)**: Use recursive counting.

Time & Space Complexities

- Varies: O(n!) to O($2^n)Space : O(n) to O(n)$

Important Tips & Tricks

- Use recursion with backtracking state.
- Prune branches with early checks.
- Optimize with visited arrays.
- Handle base cases at start.
- Test with small grids for validation.