Documentation

Intuition

Generating numbers in lexicographical order involves arranging numbers as they appear in a dictionary. Instead of generating all numbers from 1 to n and sorting them, we can directly construct them in the correct order. This avoids the overhead of sorting and leverages a systematic traversal technique using Depth-First Search (DFS). The key observation here is that we can treat numbers as nodes in a tree, where each number leads to its next valid lexicographical number by appending digits (0 to 9).

Problem Understanding

The problem aims to return all integers from 1 to n in lexicographical order while ensuring a time complexity of **O(n)** and minimal space usage. Sorting the numbers explicitly would take **O(n log n)** time and use extra memory. Instead, DFS allows us to traverse and generate numbers on the fly in the desired order. By starting from smaller digits and exploring all possibilities systematically, we ensure the result is accurate without needing additional sorting logic.

Approach

The process begins with numbers 1 to 9 as roots. For each root number, we perform DFS to generate subsequent numbers by appending digits (0 to 9). For example, starting from 1, we generate 10, 11, 12... until the numbers exceed n. This recursive exploration naturally produces numbers in lexicographical order. When the current number exceeds n, the recursion stops, ensuring we only consider valid numbers.

Efficiency and Optimization

The DFS approach is efficient because it visits each number in the range [1, n] exactly once, making the time complexity **O(n)**. Additionally, since we do not explicitly store or sort the numbers, the space complexity is limited to the depth of the recursion, which corresponds to the number of digits in n. This makes the algorithm space-efficient with **O(log n)** complexity. No auxiliary data structures like arrays for sorting are required.

Key Observations

One critical insight is the tree-like structure of numbers in lexicographical order. Each number x can have children x0, x1, x2,..., x9, provided they remain within the range. This hierarchical structure ensures that DFS traversal produces numbers in the correct order without backtracking or extra computations. Starting from 1 to 9 as independent roots guarantees full coverage of the range [1, n].

Handling Edge Cases

The algorithm naturally handles edge cases. For example, if n = 1, only 1 is added to the result. For larger values of n, the algorithm ensures that numbers exceeding n are excluded, avoiding unnecessary computations. Additionally, starting from 1 to 9 ensures that all valid numbers are explored, even for non-continuous ranges (e.g., skipping 20 when n = 19).

Practical Applications

This approach is beneficial for generating lexicographical sequences in constrained environments where memory usage and time efficiency are critical. It avoids the computational cost of sorting and is ideal for scenarios like lexicographical pagination, hierarchical number generation, or scenarios requiring strict memory constraints. This method elegantly combines mathematical intuition with computational efficiency to solve the problem effectively.