Combinations Problem Documentation

Problem Statement

Given two integers n and k, return all possible combinations of k numbers chosen from the range [1, n].

You may return the answer in any order.

Examples

Example 1:

- *Input:* n = 4, k = 2
- *Output:* [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
- *Explanation:* There are 4 choose 2 = 6 total combinations. Note that combinations are unordered, i.e., [1,2] and [2,1] are considered to be the same combination.

Example 2:

- *Input:* n = 1, k = 1
- *Output:* [[1]]
- *Explanation:* There is 1 choose 1 = 1 total combination.

Constraints

- (1 leq n leq 20)
- (1 leq k leq n)

Approach

The solution uses a backtracking approach to generate all possible combinations.

Detailed Steps

1. Initialization:

- Define a class Solution.
- Create a method combine within the class, which takes two parameters n and k and returns a list of lists of integers.

2. Backtracking Function:

- Define an inner function backtrack that takes two parameters: start and path.
- The function backtrack performs the following steps:
- Base Case: If the length of the current path (path) is equal to k, append a copy of path to the result list res.
- Recursive Case: Iterate from the current start value to n. For each value i:
- Append i to path.
- Recursively call backtrack with i + 1 and the updated path.
- Remove the last element from path to backtrack.

3. Execution:

- Initialize an empty list res to store the final combinations.
- Call the backtrack function starting from 1 and with an empty list path.
- Return the list res containing all possible combinations.

Example Usage

```
sol = Solution()

print(sol.combine(4, 2))  # Output: [[1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]

print(sol.combine(1, 1))  # Output: [[1]]
```

Explanation of the Code

- **Class Definition:** A class Solution is defined.
- **Method combine:** This method takes two integers n and k and returns a list of lists, where each list represents a combination of k numbers chosen from the range [1, n].
- **Backtracking Function:** The backtrack function is used to explore all possible combinations.
- **Base Case:** When the path's length equals k, a copy of the path is added to the result list res.
- **Recursive Case:** The function iterates from the current start value to n, appending each value to the path, calling itself recursively, and then backtracking by removing the last element.
- **Initialization and Execution:** The result list res is initialized, and backtrack is called with starting values. The result list is then returned.

This approach efficiently generates all combinations using recursion and backtracking, ensuring that the solution meets the constraints and requirements of the problem statement.