📓 Remove Boxes - Complete Documentation

1. Problem Statement

You are given a list of boxes, where each element is a positive integer representing a color. You may repeatedly remove groups of consecutive boxes with the same color. When you remove k consecutive boxes of the same color, you earn k × k points. Your goal is to find the maximum number of points you can earn by optimally removing all boxes.

- Input: boxes: List[int]
- Output: int (maximum points)
- Constraints:
  - 1 <= boxes.length <= 100
  - 1 <= boxes[i] <= 100

2. Intuition

The main idea is to remove larger groups of the same color to maximize the square bonus (k*k). Sometimes, removing a group later by merging distant, same-colored boxes yields more points. So we need to find an optimal order of removals that groups colors strategically.

3. Key Observations

- Removing more boxes of the same color at once gives disproportionately more points (k^2 grows quadratically).
- It may be beneficial to delay the removal of a group and try to merge it with a similar group later in the array.
- The problem has overlapping subproblems and optimal substructure — perfect for dynamic programming.

4. **Approach**

We use a top-down dynamic programming (DP) approach with memoization:

DP State:

Dp (l, r, k) = max points from subarray boxes[l...r] where there are k additional boxes adjacent to the right of r that have the same color as boxes[r].

Transition:

- Combine boxes[r-k] group and remove:
  dp(l, r-1, 0) + (k+1)^2
- Try merging same-colored boxes:
- for i in range(l, r):
- if boxes[i] == boxes[r]:
- res = max(res, dp(l, i, k+1) + dp(i+1, r-1, 0))

Memoization:

Cache the results using lru_cache for (l, r, k) states.

5. **Edge Cases**

- All boxes are the same → remove in one go.
- No repeating colors → remove one by one.
- Distant matching colors → need merging logic.

6. **Complexity Analysis**

🕒 Time Complexity:

- Worst case is $O(n^4)$, but optimized using memoization.
- Effective complexity: $O(n^3)$

💾 Space Complexity:

- DP cache uses $O(n^3)$ space due to three state variables l, r, and k.

7. Alternative Approaches

- Greedy: Removing the largest block first may work in some cases, but fails to find optimal solutions due to a lack of future look-ahead.
- Bottom-up DP: Possible but more complex due to 3D state; top-down with memoization is more intuitive and manageable.

8. Test Cases

✅ Example 1:

```
boxes = [1,3,2,2,2,3,4,3,1]
# Output: 23
```

✅ Example 2:

```
boxes = [1,1,1]
# Output: 9
```

✅ Example 3:

```
boxes = [1]
# Output: 1
```

✅ Custom Test Case:

```
boxes = [1,2,1,2,1,2,1]
# Output: 17
# Explanation: Merge all 1's into one group with careful planning.
```

9. Final Thoughts

This problem is a great example of using advanced DP with a 3D state. It teaches how delaying immediate rewards and planning can lead to optimal solutions. Key takeaways:

- Always consider non-greedy strategies when optimal substructure exists.
- Use memoization to manage complex recursive states efficiently.
- Mastering this pattern helps with interval DP, matrix chain multiplication, and merging problems.