Burst Balloons (DP on Intervals)

Burst Balloons

Solution

You are given `n` balloons, indexed from `0` to `n - 1`. Each balloon is painted with a number on it represented by an array `nums`. You are asked to burst all the balloons.

If you burst the i^{th} balloon, you will get `nums[i - 1] * nums[i] * nums[i + 1]` coins. If `i - 1` or `i + 1` goes out of bounds of the array, then treat it as if there is a balloon with a `1` painted on it.

Return the maximum coins you can collect by bursting the balloons wisely.

Example 1:

```
Input: nums = [3,1,5,8]
Output: 167
Explanation:
nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []
coins =  3*1*5    +   3*5*8   +  1*3*8  + 1*8*1 = 167
```

Example 2:

```
Input: nums = [1,5]
Output: 10
```

Constraints:

- `n == nums.length`
- `1 <= n <= 300`
- `0 <= nums[i] <= 100`

eg: nums = [3, 1, 5, 8]

o/p : 167

[3, 1, 5, 8] [3, 5, 8] [3, 8] [8] []

coins[j][i] = coins earned on bursting [j...i] balloons

coins[0][n-1] = ans

coins[j][i+1] = nums(i+1)

Key Insight:-

Instead of bursting balloons in any order, we

- Think in reverse: which balloon to burst last in a subarray
- This way, the subproblem boundaries are easier to define, & the surrounding balloons are still intact.

Preprocessing:-

- Add a virtual balloon with value 1 to both of ends of nums.

eg: If nums = [3, 1, 5, 8] → we change it to nums = [1, 3, 1, 5, 8, 1]

- This simplifies edge cases when bursting first or last balloons.

DP Definition:-

Let dp[i][j] be the maximum coins you can get by bursting balloons between index i & j (exclusive) in the new padded array.

So
- Interval is (i, j) → balloons at i & j are not burst, but act as the boundaries

Recursive Strategy:
To compute dp[i][j]:
1. Loop over all possible balloons k such that i < k < j.
2. Imagine k is the last balloon to burst in interval (i, j).
3. The coins you get:
   - nums[i] * nums[k] * nums[j] → bursting k last, surrounded by i & j.
   - Plus coins from subproblems:
     - dp[i][k] → burst balloons in (i, k)
     - dp[k][j] → burst balloons in (k, j).

So:

```cpp
dp[i][j] = max(dp[i][j], dp[i][k] + nums[i]*nums[k]*nums[j] + dp[k][j])
```

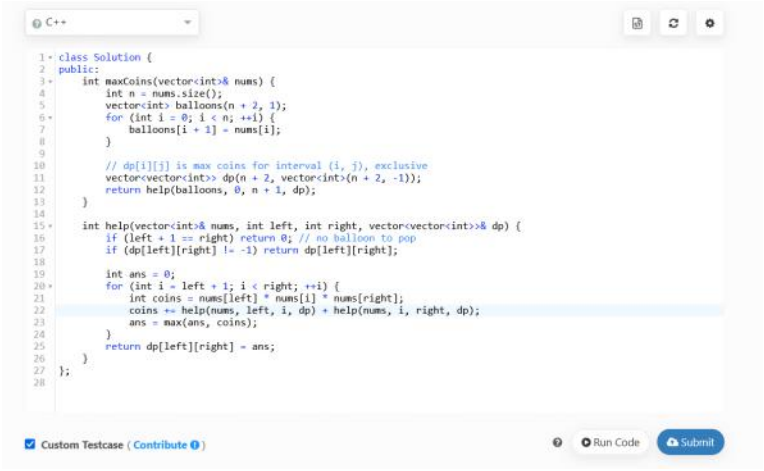
Base Case:
- when i+1 == j, there's no balloon to burst in between → return 0.

Memoization:
- Store results in a 2D table dp[i][j]
- Before computing dp[i][j], check if it's already computed (dp[i][j] != -1)

already computed (dp[i][j] != -1)

Final Answer:

- call help(nums, 0, n+1)



```cpp
class Solution {
public:
    int maxCoins(vector<int>& nums) {
        int n = nums.size();
        vector<int> balloons(n + 2, 1);
        for (int i = 0; i < n; ++i) {
            balloons[i + 1] = nums[i];
        }

        // dp[i][j] is max coins for interval (i, j), exclusive
        vector<vector<int>> dp(n + 2, vector<int>(n + 2, -1));
        return help(balloons, 0, n + 1, dp);
    }

    int help(vector<int>& nums, int left, int right, vector<vector<int>>& dp) {
        if (left + 1 == right) return 0; // no balloon to pop
        if (dp[left][right] != -1) return dp[left][right];

        int ans = 0;
        for (int i = left + 1; i < right; ++i) {
            int coins = nums[left] * nums[i] * nums[right];
            coins += help(nums, left, i, dp) + help(nums, i, right, dp);
            ans = max(ans, coins);
        }
        return dp[left][right] = ans;
    }
};
```

Custom Testcase (Contribute) Run Code Submit

T.C

T.C $O(n^2)$

S.C $O(n^2)$