877. Stone Game

Medium 2090 2094 Add to List Share

Alice and Bob play a game with piles of stones. There are an **even** number of piles arranged in a row, and each pile has a **positive** integer number of stones `piles[i]`.

The objective of the game is to end with the most stones. The **total** number of stones across all the piles is **odd**, so there are no ties.

Alice and Bob take turns, with **Alice starting first**. Each turn, a player takes the entire pile of stones either from the **beginning** or from the **end** of the row. This continues until there are no more piles left, at which point the person with the **most stones wins**.

Assuming Alice and Bob play optimally, return `true` *if Alice wins the game, or* `false` *if Bob wins*.

Example 1:

```
Input: piles = [5,3,4,5]
Output: true
Explanation:
Alice starts first, and can only take the first 5 or the last 5.
Say she takes the first 5, so that the row becomes [3, 4, 5].
If Bob takes 3, then the board is [4, 5], and Alice takes 5 to win with 10 points.
If Bob takes the last 5, then the board is [3, 4], and Alice takes 4 to win with 9
points.
This demonstrated that taking the first 5 was a winning move for Alice, so we return
true.
```

```
class Solution {
public:
    bool stoneGame(vector<int>& piles) {
        int n = piles.size();
        int dp[n][n][2];
        memset(dp, 0, sizeof(dp));
        for (int i = 0; i < n; i++) {
            dp[i][i][0] = piles[i];
        }
        for (int len = 2; len <= n; len++) {
            for (int i = 0; i < n - len; i++) {
                int j = i + len - 1;
                dp[i][j][0] = max(dp[i + 1][j][1] + piles[i], dp[i][j - 1][1] +
```

```
piles[j]);
                dp[i][j][1] = max(dp[i + 1][j][0], dp[i][j - 1][0]);
            }
        }
        return dp[0][n - 1];
    }
};
```

Proof by loop invariant:

The loop invariant is: `dp[i][j][k]` is always the max scores Alice can get. `i` is the start index of the piles, j is the end index of the piles (inclusive) and `k == 0` is Alice's turn and `k==1` means its Bob's turn

Initialization:
when there's only one pile, i.e., `i == j`, if it is currently Alice's turn, the best stratgy for Alice is just get that pile, so `dp[i][i][0] = piles[i]`, if it is currently Bob's turn, Alice can earn only zero score, the loop invariant holds

Maintainence:

for at loop `i, j = i + len - 1` it has two cases:

1. its Alice's turn. We need to update `dp[i][j][0]`. Alice may pick either pile `i` or pile `j`, and because `dp[i - 1][j][1]` and `dp[i][j - 1][1]` are the maximum score for next turn on each choice, then `dp[i][j][0] = max(dp[i + 1][j][1] + piles[i], dp[i][j - 1][1] + piles[j])` is the maximum score for current turn. The invairant holds

2. It's Bob's turn. We need to update `dp[i][j][1]`. Bob may also pick either pile `i` or pile `j`, and because `dp[i - 1][j][0]` and `dp[i][j - 1][0]` are the maximum score Alice can get for next turn on each Bob's choice, then `dp[i][j][1] = max(dp[i + 1][j][0], dp[i][j - 1][0]` is the maximum score for current turn. The invairant holds

Termination:

When i = 0 and j = i - 1, then loop ends. The invairant still holds

Time complexity:

O(n^2) where n is the length of piles