486. Predict the Winner

Add to List

Description Submission Solutions

- Total Accepted: **5110**
- Total Submissions: **11629**
- Difficulty: **Medium**
- Contributors: **sameer13**

Given an array of scores that are non-negative integers. Player 1 picks one of the numbers from either end of the array followed by the player 2 and then player 1 and so on. Each time a player picks a number, that number will not be available for the next player. This continues until all the scores have been chosen. The player with the maximum score wins.

Given an array of scores, predict whether player 1 is the winner. You can assume each player plays to maximize his score.

Example 1:

```
Input: [1, 5, 2]
Output: False
Explanation: Initially, player 1 can choose between 1 and 2.
If he chooses 2 (or 1), then player 2 can choose from 1 (or 2) and 5. If player 2 chooses 5, then player 1 will be left with 1 (or 2).
So, final score of player 1 is 1 + 2 = 3, and player 2 is 5.
Hence, player 1 will never be the winner and you need to return False.
```

Example 2:

```
Input: [1, 5, 233, 7]
Output: True
Explanation: Player 1 first chooses 1. Then player 2 have to choose between 5 and 7. No matter which number player 2 choose, player 1 can choose 233.
Finally, player 1 has more score (234) than player 2 (12), so you need to return True representing player1 can win.
```

We can use dynamic programming. Suppose after several rounds, the remaining array is nums[i], nums[i+1],... nums[j]

- dp[i][j] = the maximum score of player1 on subarray nums[i..j]

Player1 can choose either nums[i] or nums[j]. If nums[i] is chosen, player2 will also make best effort to get dp[i+1][j]. So for the subarray nums[i+1] ... nums[j], player1 can get:

- nums[i + 1] + nums[i + 2] + ... + nums[j] - dp[i+1][j], which is
- sum(nums[i+1] to nums[j]) - dp[i+1][j]

So we need another array sum to do range sum query, I set sum[0] to 0, sum[i] is the sum of all elements in nums before index i. so finally:

- dp[i][j] = max { sum[j+1] - sum[i+1] - dp[i+1][j] + nums[i],
  sum[j] - sum[i] - dp[i][j-1] + nums[j]}

```
class Solution {
public:
    bool PredictTheWinner(vector<int>& nums) {
```

```
        if(nums.size()<2) return true;
        int n=nums.size();
        vector<int> sum(n+1,0);
        sum[0]=0;
        for(int i=1;i<=n;i++) sum[i] = sum[i-1]+nums[i-1];
        vector<vector<int>> dp(n,vector<int>(n,0));
        for(int len=1;len<n;len++)
        {
            for(int i=0;i+len<n;i++)
            {
                int j=i+len;
                if(len==1) dp[i][j]=max(nums[i],nums[j]);
                else
                {
                    int can1 = sum[j+1]-sum[i+1] - dp[i+1][j] + nums[i]; // nums[i + 1]
+ nums[i + 2] + ... + nums[j] = sum[j+1]-sum[i+1]
                    int can2 = sum[j]-sum[i] - dp[i][j-1] + nums[j];// nums[i] + nums[i
+ 2] + ... + nums[j-1] = sum[j] - sum[i]
                    dp[i][j] = max(can1, can2);
                }
            }
        }
        return 2*dp[0][n-1]>=sum[n];
    }
};
```