416. Partition Equal Subset Sum

Medium 3847 87 Add to List Share

Given a **non-empty** array `nums` containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Example 1:

```
Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: nums = [1,2,3,5]
Output: false
Explanation: The array cannot be partitioned into equal sum subsets.
```

Constraints:

- `1 <= nums.length <= 200`
- `1 <= nums[i] <= 100`

Use knapsack problem:
State: Capacity of knapsack, items
Choices: include an item or not.

Given a fixed-size knapsack and N items, each of them weighted nums[i], could we fill the knapsack exactly.

Definition:
dp[i][j]=x: for the first i items, when the capacity of knapsack is j, if x is true, we can fill it exactly.

Recurrence relation:
dp[i][j] = dp[i-1][j] or dp[i-1][j-nums[i]]

Time complexity: O(n * sum)
Space complexity: O(sum)

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

        int sum = 0;
        for (auto num : nums) {
            sum += num;
        }

        if (sum % 2 == 1) {
            return false;
        }

        // dp[i][0] = true, dp[i][j] = false
        vector<bool> dp(sum / 2 + 1, false);
        dp[0] = true;

        // dp[i][j] = dp[i-1][j] or dp[i-1][j-nums[i]]
        for (auto num: nums) {
            // traverse j reversely to avoid effect on other value
            for (int j = sum / 2 + 1; j >= 0; j--) {
                if (j >= num) {
                    dp[j] = dp[j] || dp[j - num];
                }
            }
        }

        return dp[sum / 2];
    }
};
```