698. Partition to K Equal Sum Subsets

Solved

Medium | Topics | Companies | Hint

Given an integer array `nums` and an integer `k`, return `true` if it is possible to divide this array into `k` non-empty subsets whose sums are all equal.

Example 1:

> **Input:** nums = [4,3,2,3,5,2,1], k = 4
> **Output:** true
> **Explanation:** It is possible to divide it into 4 subsets (5), (1, 4), (2,3), (2,3) with equal sums.

Example 2:

> **Input:** nums = [1,2,3,4], k = 3
> **Output:** false

Constraints:

- `1 <= k <= nums.length <= 16`
- `1 <= nums[i] <= 10^4`
- The frequency of each element is in the range `[1, 4]`.

Key Observations

We are given:

- nums[] with n ≤ 16
- We must partition into k non-empty subsets
- Each subset must have the same sum

Step 1: Total sum check (mandatory)

Let:

```
total = sum(nums)
```

If:

```
total % k != 0
```

Impossible immediately → return false

Let:

```
target = total / k
```

Every subset must sum to target.

We need to track:

- Which elements are already used
- How full the current subset is

Since n ≤ 16, we can represent usage by a bitmask of length n:

- Bit i = 1 → nums[i] already used
- Total states = 2^n ≤ 65536 → totally manageable

So each DP state represents:

"Which elements have already been taken into some completed or current subset."

DP State

We use:

```
dp[mask] = current sum of the ongoing subset
```

Meaning:

- mask → which elements are already used
- dp[mask] → how much sum we have accumulated in the current group
- All previous groups are already completed with sum = target

Special meaning:

- If dp[mask] == 0, it means:
  - Either we are at the start
  - Or we just completed one subset and moved on to the next

Base case:

```
dp[0] = 0
```

No elements used, current subset sum = 0.

Final goal:

```
mask == (1<<n)-1  AND  dp[mask]==0
```

All elements used and we just finished a group exactly.

Transitions

From a given mask, we try to add one unused element i:

Conditions:

1. i not used → (mask & (1<<i)) == 0
2. Adding nums[i] does not overflow:

```
dp[mask] + nums[i] <= target
```

Then:

```
newMask = mask | (1<<i)
newSum  = (dp[mask] + nums[i]) % target
dp[newMask] = newSum
```

Why % target?

If a subset reaches exactly target, we reset to 0 and start filling the next group.

Code

```cpp
class Solution {
public:
    bool canPartitionKSubsets(vector<int>& nums, int k) {
        int n = nums.size();
        int total = 0;
        for (int x : nums) total += x;

        if (total % k != 0) return false;
        int target = total / k;

        // Optimization: sort descending (prunes faster)
        sort(nums.rbegin(), nums.rend());
        if (nums[0] > target) return false;

        int N = 1 << n;
        vector<int> dp(N, -1);
        dp[0] = 0;

        for (int mask = 0; mask < N; mask++) {
            if (dp[mask] == -1) continue;

            for (int i = 0; i < n; i++) {
                if (mask & (1 << i)) continue;

                if (dp[mask] + nums[i] <= target) {
                    int newMask = mask | (1 << i);
                    int newSum = (dp[mask] + nums[i]) % target;

                    dp[newMask] = newSum;
                }
            }
        }

        return dp[N - 1] == 0;
    }
};
```

At any moment:

- mask decides which numbers are already fixed permanently
- dp[mask] tracks how full the current bucket is
- When dp == 0, we just finished one bucket and move to the next
- We never exceed target
- Since we use exactly all numbers, we necessarily form k equal-sum subsets