Tuesday, March 18, 2025 4:42 PM

Partition Equal Subset Sum

Partition Equal Subset Sum

Difficulty: **Medium** Accuracy: **30.24%** Submissions: **255K+** Points: **4**

Given an array **arr[],** determine if it can be partitioned into two subsets such that the sum of elements in both parts is the **same.**

Note: Each element must be in exactly one subset.

Examples:

Input: arr = [1, 5, 11, 5]
Output: true
Explanation: The two parts are [1, 5, 5] and [11].

Input: arr = [1, 3, 5]
Output: false
Explanation: This array can never be partitioned into two such parts.

Constraints:
1 ≤ arr.size ≤ 100
1 ≤ arr[i] ≤ 200

i/p : arr = [1, 5, 11, 5]
o/p : true

[1, 5, 5] [11]

2 subsets of equal sum :-

$\sum S_1 = \sum S_2 = S$ (let)

S_1 S_2

∴ Sum of the array = $S_1 + S_2 = S + S = 2S$.

⇒ Total = 2 × subset sum → for this to be viable

arr[] = [1, 5, 11, 5] → Total = 22

$S = 22/2 = 11$

we need to check if a subset is there with sum = 11.

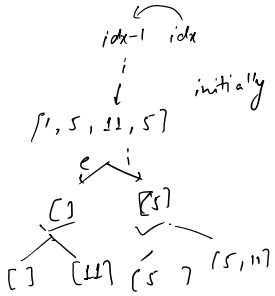
If yes, → true
no → false

How can we check if a subset with sum is present?

Brute force :- Generate all subset → check for each of them

[1, 5, 11, 5]

at each index → decide → either to include or exclude from current subset



generate Subset (vector<int> &a) {

}

help (vector<int> &a, int i, int sum) {
if(sum == total/2) return true;
if(i<0) return false;
return help(a, i-1, sum + a[i]) || help(a, i-1, sum);
}

Brute force

```
class Solution {
  public:
    bool equalPartition(vector<int>& arr) {
        // code here
        int n=arr.size();
        int total=0;
        for(int i:arr)total+=i;
        return help(arr, n-1, 0, total);
    }
    int help(vector<int> &arr, int i, int sum, int total){
```

T.C. $O(2^1 \times 2^1 \times 2^1 \cdots 2^{n\text{ times}}) = O(2^n)$

S.C. $O(n)$ ← recursion stack

Brute force

```
class Solution {
  public:
    bool equalPartition(vector<int>& arr) {
        // code here
        int n=arr.size();
        int total=0;
        for(int i:arr)total+=i;
        return help(arr, n-1, 0, total);
    }
    int help(vector<int> &arr, int i, int sum, int total){
        if(2*sum==total)return 1;
        if(i<0)return 0;
        return help(arr, i-1, sum+arr[i], total)||help(arr, i-1, sum, total);
    }
};
```

T.C. $O(2^1 \times 2^1 \times 2^1 \cdots 2^1)$ ↑

S.C. $O(n)$ ← recursion stack

Optimized :- DP + Memoization :-

```
class Solution {
  public:
    bool equalPartition(vector<int>& arr) {
        // code here
        int n=arr.size();
        int total=0;
        for(int i:arr)total+=i;
        vector<vector<int>> dp(n, vector<int>(total, -1));
        return help(arr, n-1, 0, total, dp);
    }
    int help(vector<int> &arr, int i, int sum, int total,  vector<vector<int>> &dp){
        if(2*sum==total)return 1;
        if(i<0)return 0;
        if(dp[i][sum]!=-1)return dp[i][sum];
        return dp[i][sum]=help(arr, i-1, sum+arr[i], total, dp)||help(arr, i-1, sum, total, dp);
    }
};
```

state

dp[i][sum]

$0 \rightarrow n-1$, $0 \rightarrow$ ~~total~~

T.C $O(n \times$ ~~total~~$)$

S.C. $O(n \times$ ~~total~~$)$