Problem Submissions Solution

Subset Sum Equal To K

42 Difficulty: MEDIUM

Avg. time to solve 30 min
Success Rate 65%

Problem Statement

Suggest Edit

You are given an array/list ‘ARR’ of ‘N’ positive integers and an integer ‘K’. Your task is to check if there exists a subset in ‘ARR’ with a sum equal to ‘K’.

Note: Return true if there exists a subset with sum equal to ‘K’. Otherwise, return false.

For Example :

```
If ‘ARR’ is {1,2,3,4} and ‘K’ = 4, then there exists 2
subsets with sum = 4. These are {1,3} and {4}. Hence,
return true.
```

Input Format :

```
The first line contains a single integer T representing
the number of test cases.

The first line of each test case contains two space-
separated integers ‘N’ and ‘K’ representing the size of
the input ‘ARR’ and the required sum as discussed above.

The next line of each test case contains ‘N’ single
space-separated integers that represent the elements of
the ‘ARR’.
```

Output Format :

```
For each test case, return true or false as discussed
above.
Output for each test case will be printed in a separate
line.
```

Note:

```
You don’t need to print anything, it has already been
taken care of. Just implement the given function.
```

Constraints:

```
1 <= T <= 5
1 <= N <= 10^3
0 <= ARR[i] <= 10^9
0 <= K <= 10^9

Time Limit: 1 sec
```

Sample Input 1:

```
2
4 5
4 3 2 1
5 4
2 5 1 6 7
```

Sample Output 1:

```
true
false
```

Explanation For Sample Input 1:

```
In example 1, ‘ARR’ is {4,3,2,1} and ‘K’ = 5. There exist
2 subsets with sum = 5. These are {4,1} and {3,2}. Hence,
return true.
In example 2, ‘ARR’ is {2,5,1,6,7} and ‘K’ = 4. There are
no subsets with sum = 4. Hence, return false.
```

Sample Input 2:

```
2
4 4
6 1 2 1
5 6
1 7 2 9 10
```

Sample Output 2:

```
.
```

C++ (g++ 5.4)

```cpp
/*
In both the approach differ in space complexity
    Approach 1: Space Complexity = O(n * sum of all elements)

    Approach 2: Space COmplexity = O(n * k)

*/
        //Approach 1: Vary or keep the count of Addition of
                //Elements in subset (dp array size = [n][sum of all elements])


bool solve(int n, int sumTillNow, int k, vector<vector<int>> &dp, vector<int> &arr){

    if(sumTillNow == k)
        return dp[n][sumTillNow] = true;
    if(n == 0)
        return dp[n][sumTillNow] = false;

    if(dp[n][sumTillNow] != -1)
        return dp[n][sumTillNow];
    bool include = solve(n - 1, sumTillNow + arr[n - 1], k, dp, arr);
    bool nInclude = solve(n - 1, sumTillNow, k, dp, arr);

    return dp[n][sumTillNow] = (include or nInclude);
}
bool subsetSumToK(int n, int k, vector<int> &arr) {

    // Write your code here.
    int sum = 0;
    for(int i = 0; i < n; i++)
        sum += arr[i];



    vector<vector<int>> dp(n + 1, vector<int>(sum + 1, - 1));
    bool ans = solve(n, 0, k, dp, arr);
    return ans;
}


        //Approach 2: Vary or keep the count of Subtraction from Target/K

/*

bool solve(int n, int k, vector<vector<int>> &dp, vector<int> &arr){
    if(k < 0)
        return false;
    if(k == 0)
        return dp[n][k] = true;
    if(n == 0)
        return dp[n][k] = false;

    if(dp[n][k] != -1)
        return dp[n][k];
    bool include = solve(n - 1, k - arr[n - 1], dp, arr);
    bool nInclude = solve(n - 1, k, dp, arr);

    return dp[n][k] = (include or nInclude);
}
bool subsetSumToK(int n, int k, vector<int> &arr) {

    // Write your code here.


    vector<vector<int>> dp(n + 1, vector<int>(k + 1, - 1));
    bool ans = solve(n, k, dp, arr);
    return ans;
}

*/
```

Console

Previous Next Show Hint Last saved on 8:42:24 PM Run Code Submit