Editorial

Subset Sums

This is a classical dynamic programming problem. But a brute force approach can also partially solve the problem up to n=100.

- **Brute force approach**

This approach makes every possible subset and checks whether its sum equals to S and counts them. First, let's find how many subsets in total can be made from the n numbers.

For each number, we have two options, either to add the number to the subset or to not to add the number. So there are altogether $2 * 2 * 2 * 2 \ldots\ldots\ldots\ldots * 2 = 2^n$ number of subsets that can be made from n numbers. If we check the time complexity of the algorithm it is $O(2^n)$. A normal computer can run a program up to $O(10^8)$ within a second. Therefore this solution will only work up to n~100 $O(2^n \sim 10^8)$.(If you are not familiar with the big O notation, please read the reference text book.)

```
#include <stdio.h>

using namespace std;

int nums[1010];
int N,S,ans;

void DFS(int n,int s){ // depth first search to make all subsets

    if(n==N){ // terminating condition - if iterated through all the numbers n=N
        if(s==S) // check whether this subset satisfies our condition
          ans++;    // add to the list
        return; // stop running the below codes <-- finished
    }

    DFS(n+1,s); // without adding the nth number
    DFS(n+1,s+nums[n]); // adding the nth number
}
int main(){

    scanf ("%d %d",&N,&S);

    for(int i=0;i<N;i++)
        scanf("%d",&nums[i]);

    DFS(0,0);

    printf("%d\n",ans);
    return 0;
}
```

- **Dynamic Programming approach**

Dynamic programming is a more efficient way to solve this problem. This method *stores* the results of smaller/lower levels and uses the result to solve large/higher levels.

Let's define a dynamic programming array as follows,

dp[0] =1

dp[s] = how many ways it is possible to make the sum s after inserting i th number

Then we can iterate through the number list. If number list was [2,3,5] and S=10 after adding each number in the list we need to update the dp array as follows,

1st iteration [2] - dp = [1,0,1,0,0,0,0,0,0,0,0]

2nd iteration [2,3]- dp = [1,0,1,1,0,1,0,0,0,0,0]

3rd iteration [2,3,5]- dp = [1,0,1,1,0,2,0,1,1,0,1]

This is not possible to implement using a single dp array and at least two dp arrays are needed. For simplicity of our explanation, we will use a 2- dimensional dp array defined as follows.

dp[i][0] =2 for all i

dp[i][s] = how many ways it is possible to make the sum s after inserting i th number

Since both i,s<=1000 it is possible to make this 2D array in the given memory space. When we have calculated dp[i](upto i^{th} number) it is possible to make dp[i+1] eaily using the result already created in dp[i]. The equation will look as follows,

dp[i][s] = dp[i-1][s] + dp[i-1][s-nums[i]]

number of ways to make sum s = number of ways to make sum s without adding ith number +

number of ways to make sum s after adding ith number

The code will look as follows.

```cpp
#include <iostream>
#include <stdio.h>

using namespace std;

int dp[1010][1010];
int nums[1010];

int main(){

    int N,S;
    scanf ("%d %d",&N,&S);

    for(int i=0;i<N;i++)
        scanf("%d",&nums[i]);

    dp[0][0]=1;
    dp[0][nums[0]]=1;

    for(int i=1;i<N;i++){ // i is the number
        dp[i][0]=1; // manually add for the zero(# of ways =1)
        for(int s=1;s<=S;s++){ // s is the sum
            if(s-nums[i]>=0) // avoid negative indexing
                dp[i][s]=dp[i-1][s]+dp[i-1][s-nums[i]];
            else    dp[i][s]=dp[i-1][s];
        }
    }
    printf("%d\n",dp[N-1][S]);
    return 0;
}
```