1 Introdução

Esse problema foi retirado de `https://cses.fi/problemset/task/1636`, da plataforma de programação competitiva CSES.

2 Código em C++

```cpp
#include<bits/stdc++.h>
using namespace std;

/*
Construct solutions from array:
* every ordered solution can be represented as a n-uple (k_1, ... k_n)
        where k_i is the amount of times c[i] appears in the sum
* however, code this solution BOTTOM UP
*/


void solve(){
        const int MOD = 1e9 + 7;
        int n, x;
        cin >> n >> x;
        int c[n];
        for (int i = 0; i < n; i++){
        cin >> c[i];
        }
        int dp[x + 1] = {0};
        dp[0] = 1;

        for (int i = 0; i < n; i++){
        for (int s = c[i]; s <= x; s++){
                dp[s] = (dp[s] + dp[s - c[i]]) % MOD;
        }
        }
        cout << dp[x];
}
```

3 Demonstração

Lema 3.1. *Cada solução ordenada $(s_1, s_2, \ldots, s_k)$ é unicamente representada por um vetor $(a_1, \ldots, a_n)$, em que a_i é a quantidade de vezes que a moeda $c[i]$ foi utilizada.*

Lema 3.2. *Denote por $C(i, j)$ a quantidade de combinações $(a_1, \ldots, a_n)$ com soma j que usam apenas as moedas $c[0], \ldots, c[i]$, isto é, com $a_{i+1} = \ldots = a_n = 0$, e (extendendo essa definição) deixe $C(-1, j)$ denotar a quantidade de combinações vazias com soma j.*

Então, após a i-ésima iteração, $dp[j] = C(i, j)$.

Prova. Primeiro, provamos a seguinte sub-invariante de loop:

Lema 3.3. ***Assuma*** *a hipótese de indução do lema principal, isto é, que temos inicialmente $dp[j] = C(i, j)$ para todo $0 \leq j \leq x$. Então, durante a $(i+1)$-ésima iteração do loop externo, o seguinte é válido: para todo $0 \leq w \leq x$ temos que antes da iteração $w = w_0$ do loop interno, $dp[j]$ representa quantas combinações de soma j há usando apenas as moedas $c[0], \ldots, c[i+1]$ para todo $0 \leq j < w_0$.*

Prova. Antes da primeira iteração $w = c[i+1]$ do loop interno, temos que $j < w \implies j < c[i+1]$, o que implica evidentemente que não há combinação com essa soma que use $c[i+1]$. Dessa forma, temos que $C(i+1, j) = C(i, j) = dp[j]$, pela hipótse indutiva.

Por fim, suponha válido para $w = w_0$. Toda combinação que usa as moedas $c[0], \ldots, c[i+1]$ é de duas formas:

- $a_{i+1} = 0$ ($C(i, w_0 + 1)$ possibilidades)
- $a_{i+1} \geq 1$, e portanto a combinação obtida retirando uma repetição de $c[i+1]$ é uma combinação válida de soma $w_0 + 1 - c[i+1]$ usando as moedas $c[0], \ldots, c[i+1]$. Isso totaliza $C(i+1, w_0 + 1 - c[i+1]$.

Assim,

$$C(i+1, w_0+1) = C(i, w_0+1) + C(i+1, w_0+1-c[i+1])$$

Uma vez que $w_0 + 1 - c[i+1] \leq w_0$, pela hipótese indutiva desse sub-lema temos que nesse instante $C(i+1, w_0+1-c[i+1] = dp[w_0+1-c[i+1]$. Por outro lado, temos que $dp[w_0+1]$ não foi alterada até agora nesse loop interno, e portanto pela hipótese indutiva do lema principal $dp[w_0+1] = C(i, w_0+1)$. Portanto, a ação do algoritmo é

$$dp[w_0+1] \leftarrow (dp[w_0+1] + dp[w_0+1-c[i+1] = C(i, w_0+1) + C(i+1, w_0+1-c[i+1])$$

garantindo a manutenção da invariante descrita nesse sub-lema. □

Dessa forma, a invariante principal (i.e, do loop externo) segue com um simples argumento indutivo. A inicialização de $dp[0] \leftarrow 1$ e $dp[j] \leftarrow 0$ para todo $j > 0$ garante que $dp[j] = C(-1, j)$ para todo $0 \leq j \leq x$.

Assim, aplicando o lema anterior para $i = -1$, demonstramos o caso base: a validade ao final da iteração $i = 0$. Analogamente, o passo indutivo também segue da aplicação do lema anterior.

□