Ejercicio 3: Regalos por la fama.

Descripción:

Mª Gabriela y Eduardo Fernando han recibido n regalos por su estupendo trabajo en una serie de televisión de reconocida fama. Cada regalo viene en una caja destinada a ambos. Como no tienen suficiente tiempo para desempaquetar y mirar que es cada cosa, han decidido utilizar el siguiente criterio: Cada uno debe quedarse con la misma cantidad de peso, para ello cuentan los pesos de cada una de las cajas $P_1, \dots, P_n$(números enteros positivos). Al cabo de un rato aun no han conseguido hacer el reparto que desean.

Nuestro objetivo en este problema será diseñar un algoritmo que nos diga si dado el **número de regalos que tenemos (n)** y el **peso de cada uno de ellos($\mathbf{P_1}, \dots, \mathbf{P_n}$)**, nos diga si es posible dividir este conjunto en dos subconjuntos de igual peso, sin dejar ningún regalo fuera.

Naturaleza n-etápica:

Que un conjunto de regalos sea subdivisible en dos subconjuntos de igual peso dependerá del número de regalos que tengamos y del peso de estos. Estas características se verán reflejadas en una función booleana S(m,k), la cual nos dice si esta subdivisión es posible o no. En esta función la m representa el peso del conjunto de regalos que consideramos, y la k el número de regalos que consideramos. Concretamente, esta función nos dice si podemos hacer un subconjunto de peso m con los k regalos que tenemos. Por lo tanto, siendo **D** la suma del peso de los n regalos que tenemos, la solución consistirá en ver si se puede hacer un subconjunto de peso **D/2** teniendo en cuenta todos los regalos que tenemos, lo cual sería equivalente a averiguar el valor de **S(D/2 , n)**.

Dicho esto, podemos ver la solución al problema como una sucesión de decisiones, en la que en cada etapa se verá la factibilidad de crear un subconjunto con el peso y los regalos actuales, hasta llegar al objetivo.

Primero tomaríamos una decisión para el subconjunto de regalos de k = 1, con peso m = 1 (etapa 1), después tomaríamos otra decisión sobre el subconjunto de regalos de K = 1, con peso m = 2 (etapa 2). Seguiremos tomando decisiones incrementando en una unidad el peso del subconjunto que queremos realizar (m), hasta que lleguemos al peso de nuestro objetivo: m = D/2. Una vez tomada la decisión para este peso, incrementaríamos la k y volveríamos a tomar las decisiones para todo el abanico de pesos hasta que volvamos a llegar a nuestro peso límite. Este proceso seguiría hasta llegar a la toma de decisión del conjunto de n regalos para un peso de D/2, que es el objetivo al que intentamos llegar.

Por lo tanto, este problema tiene una clara naturaleza n-etápica.

Verificación del POB:

Partiremos del enunciado del principio de optimalidad de Bellman, el cual nos dice que para que un problema satisfaga dicho principio, si en una sucesión óptima de decisiones, cada subsucesión es a su vez óptima. Es decir, si miramos una solución parcial de la solución general, esta debe ser una solución óptima al subproblema asociado.

Entonces, si tenemos una solución óptima para nuestro problema (S(m,k) = true), las soluciones parciales a los subproblemas también tienen que ser óptimas para que se cumpla el principio de optimalidad. Concretamente en programación dinámica, para que una solución sea óptima la solución parcial al subproblema que le precede (subproblema de tamaño -1 unidad) también tiene que serlo.

Por lo tanto, para demostrar que se verifica el P.O.B., partiremos de una solución óptima al problema, cuya solución parcial al subproblema precedente no es óptima, lo cual no es admisible para el cumplimiento del principio, y demostraremos que dicha solución parcial nos conduce a una solución general **no óptima**, lo cual sería contradictorio y verificaría el P.O.B. para nuestro problema.

- ➔ Solución general óptima: $\boldsymbol{S(m,k) = true}$
- ➔ Solución parcial no óptima para el subproblema precedente: $\boldsymbol{S(m,k-1)\ V\ S(m-P_k,k-1) = false}$

A partir de esta solución parcial calculamos la solución para la siguiente etapa:

$$\boldsymbol{S(m,k) = S(m,k-1)\ V\ S(m-P_k,k-1) = false}$$

Llegamos así a la conclusión de que $\boldsymbol{S(m,k) = false}$, habiendo partido de que $\boldsymbol{S(m,k) = true}$, por lo tanto es contradictorio, lo cual demuestra que toda solución parcial de una solución óptima es óptima en nuestro problema, o lo que es lo mismo, **se cumple el principio de optimalidad de Bellman**.

Planteamiento de una recurrencia:

Como se dijo anteriormente, llamamos S(m,k) a la función que nos dice si es factible obtener un subconjunto de peso m eligiendo determinados regalos de los k que disponemos.

Ahora pasaremos a definir dicha función para posteriormente poder implementar un algoritmo basado Programación Dinámica que resuelva nuestro problema:

$$S(m,k) = \begin{cases} true & si\ m = 0 \\ false & si\ k = 0\ y\ m > 0 \\ S(m,k-1)\ V\ S(m-P_k,k-1) & si\ m \geq P_k\ y\ k > 0 \\ S(m,k-1) & si\ m < P_k\ y\ k > 0 \end{cases}$$

Una vez definida nuestra ecuación recurrente, podemos definir un algoritmo basándonos en ella.

El siguiente algoritmo encuentra el valor que buscamos. Irá guardando todos los valores obtenidos en una tabla con el mismo nombre que la función para cuando se precise recurrir a la recurrencia a la hora de calcular un nuevo valor, sacar este de la tabla en vez de volver a realizar la ejecución recurrente de la función, ya que este valor ya habrá sido previamente calculado. Además el algoritmo incluirá sentencias que dejen marcas que posteriormente permitan construir el conjunto de regalos de uno de ellos. Los regalos restantes formarán el otro subconjunto:

```
func HAY_REPARTO(P(1..n), S(0..D/2, 0..n)) return (Boolean × Conjunto)
  for k in 0..n loop S(0,k) ← True
  for m in 1..D/2 loop S(m,0) ← False
  for k in 1..n loop
    for m in 1..D/2 loop
      if m < P(k) then
        S(m,k) ← S(m,k-1)
        marca(m,k) ← False
      elsif S(m-P(k), k-1) then
        S(m,k) ← S(m-P(k), k-1)
        marca(m,k) ← True
      else
        S(m,k) ← S(m, k-1)
        marca(m,k) ← False
  end loop
  {Ahora calculamos la colección de regalos}
  if S(D/2, n) then
    i ← D/2
    j ← n
    C ← ∅
    while j > 0 loop
      if marca(i,j) then
        C ← C ∪ {j}
        i ← i - P(j)
        j ← j - 1
      else
        j ← j - 1
    else
      return (False, ∅)
```

El algoritmo es O(nD)

Implementación en c++:

```
P.push_back(2);
P.push_back(2);
P.push_back(3);
P.push_back(3);
//P.push_back(8);//FALSE
P.push_back(4);//TRUE

for(int k = 0; k <= n; k++){
  S[0][k] = true;
}
for(int m = 1; m < (D/2); m++){
  S[m][0] = false;
}
for(int k = 1; k <= n; k++){
  for(int m = 1; m <= (D/2); m++){
    if(m < P[k]){
      S[m][k] = S[m][k-1];
      marca[m][k] = false;
    }
    else if( ( S[m - P[k]][k-1] == true ) && ( ( (m - P[k]) >= 0) && ( (k-1) >= 0 ) ) ){
      S[m][k] = true;
      marca[m][k] = true;
    }
    else{
      S[m][k] = S[m][k-1];
    }
  }
}

if(S[D/2][n] == true) cout << "TRUE" << endl;
else cout << "FALSE" << endl;
```

Caso de ejemplo:

➔ Caso en el que tenemos un conjunto del cual no podemos hacer un reparto paritario:

Pesos de los regalos (P_i) :

2	2	3	3	8

$D = P_1 + P_2 + P_3 + P_4 + P_5 = 2 + 2 + 3 + 3 + 8 = 18$

Por lo tanto la solución a nuestro problema nos la dará S(D/2, n) -> S(9,5) .

La tabla resultante sería:

m \ K	0	1	2	3	4	5
0	1	1	1	1	1	1
1	0	0	0	0	0	0
2	0	1	1	1	1	1
3	0	0	0	1	1	1
4	0	0	1	1	1	1
5	0	0	0	1	1	1
6	0	0	0	0	1	1
7	0	0	0	1	1	1
8	0	0	0	0	1	1
9	0	0	0	0	0	0

S(9,5) = 0
El conjunto de regalos inicial no puede ser subdivido en dos subconjuntos paritarios.

Las celdas sombreadas son las celdas que el algoritma ha marcado para posteriormente definir los subconjuntos en el caso de que S(D/2, n) = 1.

➔ Caso en el que tenemos un conjunto del cual podemos hacer un reparto paritario:

Pesos de los regalos (P_i) :

2	2	3	3	4

$D = P_1 + P_2 + P_3 + P_4 + P_5 = 2 + 2 + 3 + 3 + 4 = 14$

Por lo tanto la solución a nuestro problema nos la dará S(D/2, n) -> S(7,5) .

La tabla resultante sería:

m \ K	0	1	2	3	4	5
0	1	1	1	1	1	1
1	0	0	0	0	0	0
2	0	1	1	1	1	1
3	0	0	0	1	1	1
4	0	0	1	1	1	1
5	0	0	0	1	1	1
6	0	0	0	0	1	1
7	0	0	0	1	1	1

S(7,5) = 1
El conjunto de regalos inicial puede ser subdivido en dos subconjuntos paritarios.

Una vez que sabemos que S(D/2, n) = 1, podemos afirmar que el conjunto inicial de regalos se puede dividir en dos del mismo peso. Ahora bastaría con seguir las marcas como el algoritmo indica y tendríamos también los regalos que componen dichos subconjuntos.