

Problema B

Nombre: Juan Sebastian Vargas
Codigo: 201215310
Fecha: 20/05/16

1. Algoritmo Solución

La primera parte es la lectura de datos con el formato descrito en el enunciado, un numero, y en una nueva línea,(System.in) se encuentran las pesas.

Para este algoritmo se investigó sobre soluciones posibles y se encontró que tiene cierta similitud con el problema de partición y se utilizo para solucionar este, consta de un método que me dice si un conjunto dado se puede dividir en dos y que la suma de ellos sea igual, notese que si se puede dividir en partes iguales, y de esta manera X seria descubrible. Primero se calcula si la suma del conjunto es par, de caso contrario no se podría dividir, y luego recursivamente se busca un subconjunto que su cumpla esto.

La idea central es añadir los elementos e ir comprobando si es probable medir x. Con alguna fracción del conjunto inicial. Se encontró otra implementación y era usando programación dinámica, la cual puede llegar a ser mucho mas rápida que la que se escogio, pero lo negativo de esta es que implementando programación dinámica la complejidad temporal dependería de la suma de todos los elementos, la cual puede llegar a ser muy grande. Y si el numero de pesas crece, también lo haría la suma

2. Complejidades

Para este paso tenemos una Complejidad Temporal de $O(2^2)$, pero logramos una Espacial de $O(0)$

```

36         posible=findPartition(temp,temp.size());
37     }
38     //se genera respuesta
39     String rta=null;
40     if(posible)
41     {rta="S";}
42     else
43     {rta="N";}
44     System.out.println(rta); // Se imprime la salidaDisplay the string.
45 }
46 public static int length(int n)
47 {
48     return (int)(Math.log10(n)+1);
49 }
50
51 // A utility function that returns true if there is a
52 // subset of arr[] with sun equal to given sum
53 static boolean isSubsetSum (ArrayList<Integer> arr, int n, int sum)
54 {
55     // Base Cases
56     if (sum == 0)
57         return true;
58     if (n == 0 && sum != 0)
59         return false;
60
61     // If last element is greater than sum, then ignore it
62     if (arr.get(n-1) > sum)
63         return isSubsetSum (arr, n-1, sum);
64
65     /* else, check if sum can be obtained by any of
66     the following
67     (a) including the last element
68     (b) excluding the last element
69     */
70     return isSubsetSum (arr, n-1, sum) ||
71         isSubsetSum (arr, n-1, sum-arr.get(n-1));
72 }
73
74 // Returns true if arr[] can be partitioned in two
75 // subsets of equal sum, otherwise false
76 static boolean findPartition (ArrayList<Integer> arr, int n)
77 {
78     // Calculate sum of the elements in array
79     int sum = 0;
80     for (int i = 0; i < n; i++)
81         sum += arr.get(i);
82
83     // If sum is odd, there cannot be two subsets
84     // with equal sum
85     if (sum%2 != 0)
86         return false;
87
88     // Find if there is subset with sum equal to half
89     // of total sum
90     return isSubsetSum (arr, n, sum/2);
91 }

```

Figura 1: Algoritmo