Examen de Algorítmica y Complejidad (Plan 2014)

25 de enero de 2023

Nº matrícula:_____ **Nombre:** _____

Apellidos: _____

1) **Problema** (**2.5 puntos**). Dado un conjunto de números enteros positivos se busca encontrar el subconjunto de mayor tamaño cuya suma sea N, otro número entero positivo objetivo. Ejemplo: dado un N=7 y el siguiente array:

0	1	2	3	4	5	6
5	**10**	**3**	**2**	**1**	**2**	**7**

los elementos {3,2,2} suman 7 y el subconjunto tiene tamaño 3.

SE PIDE: Implementar un algoritmo en Java, basado en el ***esquema Voraz***[1], que encuentre una solución a este problema con ***complejidad***[2] ***O(n²)***. El algoritmo deberá tener la siguiente cabecera:

```
ArrayList<Integer> subconjuntoSuma(int[] numeros, int nObjetivo)
```

donde *numeros* es un array que contiene los elementos y *nObjetivo* es el valor suma objetivo (N). El método deberá devolver el subconjunto de elementos seleccionados o *null* si no se encuentra solución. Se podrán implementar todos los métodos y clases adicionales que se consideren necesarios.

```
// Es un problema muy parecido a encontrar el número objetivo, pero además se pide que tenga el mayor tamaño posible. Elegir el máximo valor y añadirlo a la solución es lo que hacemos normalmente pero resulta en soluciones de tamaño pequeño porque se intenta cerrar la diferencia entre el objetivo y la suma. Cuando los valores grandes se han elegido, solo quedan los valores pequeños para llenar la diferencia pequeña.
// Se podría adoptar el criterio voraz de elegir añadir elementos a la solución tomando valores mínimos pero esto contradice el principio anterior. Si usamos los valores pequeños al principio, no los tendremos para acercarnos al valor exacto necesario. Pocas veces se encontrará una solución.
// Para resolver el problema hay que encontrar una manera de intentar aplicar el principio del máximo valor para crear subconjuntos grandes. Podemos recontextualizar el problema, haciendo que la solución consista en los valores retiraremos del vector. De esta manera podremos elegir el elemento máximo que nos acerque a la solución.

    public static int seleccion(ArrayList<Integer> candidatos, int dist){
        int diff = Integer.MAX_VALUE, aux;
        int mejor = candidatos.get(0);
        for(int i:candidatos){
            aux = dist-i;

            if(aux<diff){
                diff = aux;
                mejor = i;
            }
        }
        return mejor;
    }
```

1 Desarrollar un algoritmo que no esté basado en la estrategia voraz conllevará una puntuación de 0 en el ejercicio.
2 Desarrollar un algoritmo que encuentre una solución en complejidad superior a O(n²) conllevará un 0 en el ejercicio.

```java
public static ArrayList<Integer> subconjuntoSuma(int[] numeros, int nObjetivo){
    ArrayList<Integer> candidatos = new ArrayList<Integer>();
    ArrayList<Integer> solucion = new ArrayList<Integer>();
    int dist = -nObjetivo;
    for(int i: numeros){
        dist += i;
        candidatos.add(i);
    }

    while(dist>0 && !candidatos.isEmpty()){
        // Elegimos lo que mas nos acerque a dist = 0, que nos lleve a la solucion
        int sel = seleccion(candidatos, dist);
        candidatos.remove((Integer) sel);
        // Solo candidatos que se acercan a la solucion sin pasarse son factibles
        if(dist >= sel){
            dist -= sel;
            // Recordamos los elementos que tenemos que eliminar
            solucion.add((Integer) sel);
        }
    }

    if(dist==0){
        // Reconstruimos la suma
        ArrayList<Integer> devolver = new ArrayList<Integer>();
        for(int i: numeros) devolver.add(i);
        for(int i: solucion) devolver.remove((Integer) i);
        return devolver;
    } else return null;
}
```