Semana 11 - Ejercicios de grupo

Métodos algorítmicos en resolución de problemas II
Facultad de Informática - UCM

	Nombres y apellidos de los componentes del grupo que participan	**ID Juez**
1	Daniel Hernández Martínez	MAR246
2	Jorge Arévalo Echevarría	MAR205
3	Miguel Verdaguer Velázquez	MAR285
4		

Instrucciones:

1. Para editar este documento, es necesario hacer una copia de él. Para ello:
    - Alguien del grupo inicia sesión con la cuenta de correo de la UCM (si no la ha iniciado ya) y accede a este documento.
    - Mediante la opción *Archivo* → *Hacer una copia*, hace una copia del documento en su propia unidad de *Google Drive*.
    - Abre esta copia y, mediante el botón *Compartir* (esquina superior derecha), introduce los correos de los demás miembros del grupo para que puedan participar en la edición de la copia.

2. La entrega se realiza a través del Campus Virtual. Para ello:
    - Alguien del grupo convierte este documento a PDF (dándole como nombre el número del grupo, 1.pdf, 2.pdf, etc…). Desde *Google Docs*, puede hacerse mediante la opción *Archivo* → *Descargar* → *Documento PDF*.
    - Esta misma persona sube el fichero PDF a la tarea correspondiente del *Campus Virtual*. Solo es necesario que uno de los componentes del grupo entregue el PDF.

Complejidad de problemas

Considerad los siguientes problemas de decisión

- *SUMA_INT*: dado un vector de n enteros A[1..n] y un entero S determinar si es posible encontrar un subconjunto de índices del vector tal que la suma de los elementos situados en dichos índices sea S:

$$\exists\, I \subseteq \{1,..,n\} \text{tal que } \sum_{i\in I} A[i] = S$$

- *SUMA_NAT* : dado un vector de n naturales y un natural S determinar si es posible encontrar un subconjunto de índices del vector tal que la suma de los elementos situados en dichos índices sea S.
- *PARTICION_INT*: dado un vector de n enteros determinar si se puede separar los elementos en dos partes disjuntas que sumen lo mismo:

$$\exists\, I_1, I_2 \subseteq \{1,..,n\} \text{tal que } I_1 \cup I_2 = \{1,..,n\}, I_1 \cap I_2 = \{\} \text{ y } \sum_{i\in I_1} A[i] = \sum_{i\in I_2} A[i]$$

- *PARTICION_NAT*: dado un vector de n naturales determinar si se puede separar los elementos en dos partes disjuntas que sumen lo mismo.

El objetivo es demostrar que *SUMA_INT* $\equiv^{p}$ *SUMA_NAT,* es decir, que los dos problemas son igual de difíciles siguiendo los pasos:

1. Demostrad que *SUMA_NAT* $\leq^{p}$ *SUMA_INT*
2. Demostrad cada uno de los pasos de esta cadena:

$$SUMA\_INT \leq^{p} PARTICION\_INT \leq^{p} PARTICION\_NAT \leq^{p} SUMA\_NAT$$

1. Para demostrar que *SUMA_NAT* $\leq^{p}$ *SUMA_INT* la transformación sería trivial. Al ser todos los números naturales también números enteros, no haría falta cambiar nada, cualquier entrada para SUMA_NAT se puede pasar por el algoritmo de SUMA_INT y será como resolver alguna entrada de SUMA_INT en la que no haya negativos.
2. El algoritmo de transformación polinómico de SUMA_INT a PARTICION_INT consiste en añadir al vector de entrada un nuevo elemento de valor 2S - sum_total(A), de forma que el vector pase a tener longitud 2S, y que si existe una partición que saque dos disjuntos de igual suma, ambas sumas van a ser S y por lo tanto, en el disjunto que no está el nuevo elemento habría un subconjunto de elementos que suman S. Este algoritmo es polinómico, de orden n, ya que consiste en hacer un único recorrido para calcular la suma de todos los elementos y luego sumarle 2S y añadirlo al vector.

3. El algoritmo de transformación polinómico de PARTICION_INT a PARTICION_NAT consiste en pasar a positivo todos los elementos negativos que haya. De esta forma, si existiera la partición que buscamos, lo único que haríamos sería cambiarlos de subconjunto, dejando la ecuación $\sum_{i \in I_1} A[i] = \sum_{i \in I_2} A[i]$ igual. Este algoritmo es polinómico, de orden n, ya que consiste en hacer un recorrido por todos los elementos, cambiando el signo a los negativos.
4. El algoritmo de transformación polinómico de PARTICION_NAT a SUMA_NAT consiste en dejar el vector igual y pasar como S la mitad de la suma de todos los elementos del vector. Al no haber restas, sabemos que si existe un subconjunto del vector que sume la mitad del total, el resto de elementos tienen que sumar la otra mitad del total. El algoritmo es polinómico, de orden n, ya que sólo habría que hacer un recorrido por los elementos para sumarlos y una división. Si la suma de todos los elementos fuera impar, sabríamos que no hay una partición, se podría responder eso o pasarle a SUMA_NAT alguna entrada que sepamos que vaya a dar falso.

1,5

Solo ideas que veo aquí

✓ 1,5

Demostraciones detalladas