EJERCICIO NPC- PARTITION

Por: Ana Martín Conejo

DEFINIMOS EL PROBLEMA

El problema de Partition coge de input un conjunto S de números.
La pregunta está en si esos números pueden ser divididos en dos conjuntos A y A'=S-A tal que:

$$\sum_{x \in A} x = \sum_{x \in A'} x$$

Es decir, El problema de la partición de conjuntos es un desafío en el que tenemos un conjunto de números y necesitamos dividirlo en dos grupos de tal manera que la suma de los números en cada grupo sea la misma.
Por ejemplo,
digamos que tenemos un conjunto de números {3, 1, 4, 2, 1, 1}.
El objetivo sería dividir estos números en dos grupos donde la suma de los números en cada grupo sea la misma.
En este caso, una posible partición sería {3, 2, 1} y {4, 1, 1}.
Ambos grupos suman 6.

La idea sería entonces, si tenemos un subconjunto de números cuya suma es S1, entonces debe existir otro subconjunto que contenga los números restantes del conjunto original y cuya suma sea S2,
y ambos S1 y S2 deben ser iguales.
Es decir, si sumamos todos los números en un grupo y todos los números en el otro grupo, obtendríamos el mismo resultado.

DEMOSTRAMOS QUE PARTITION ES NP-COMPLETO

Queremos demostrar que este problema es NP-Completo, para ello vamos a ver los siguientes puntos:

1. Definimos Algoritmo no determinista que resuelve A,i.e., $A \in NP$.
2. Vemos que cualquier problema NPC, B, puede ser reducido a A.
3. La reducción de B a A trabaja en un tiempo polinomial.
4. El problema original A tiene solución si y solo si B tiene.

1.-DEFINIMOS EL ALGORITMO QUE RESUELVE A , $\boldsymbol{IE}, \boldsymbol{A \in NP}$

a) Verificar Cobertura de Elementos:
Para cada elemento x en una partición A y cada elemento x' en otra A', verificamos que todos los elementos pertenecientes al conjunto S estén cubiertos por alguna de las particiones. Es decir, aseguramos que ningún elemento de S quede sin asignar a ninguna de las particiones.

Este paso implica iterar sobre todos los elementos en las particiones A y A', lo cual toma tiempo lineal con respecto al tamaño de las particiones. Dado que el número total de elementos en las particiones es proporcional al tamaño del conjunto S, esta verificación puede hacerse en tiempo polinomial.

b) Inicializar Sumas: Inicializamos dos variables, S1 y S2, ambas a 0. Estas variables representarán las sumas de los elementos en las particiones A y A' respectivamente.

Este paso inicializa simplemente dos variables a cero, lo cual es una operación de tiempo constante.

c) Calcular Sumas Parciales: Recorremos cada elemento x en A y sumamos su valor a la variable S1. Lo mismo hacemos para cada elemento x' en A', sumando su valor a la variable S2.

Este paso implica recorrer todos los elementos en las particiones A y A', y sumar sus valores. Dado que el número total de elementos en las particiones es proporcional al tamaño del conjunto S, la suma de los elementos en cada partición se puede calcular en tiempo polinomial.

d) Verificar Igualdad de Sumas: Finalmente, verificamos si la suma de los elementos en A (S1) es igual a la suma de los elementos en A' (S2). Si ambas sumas son iguales, entonces la partición es válida.

Comprobar si dos números son iguales también es una operación de tiempo constante.

Luego podemos verificar que $\mathrm{A} \in \mathrm{NP}$.

2.-CUALQUIER PROBLEMA NPC, B, PUEDE SER REDUCIDO a A.

Para ello vamos a ver la reducción de SUBSET-SUM a PARTITION.

En el algoritmo SUBSET-SUM, dado un conjunto de números enteros no negativos y un valor objetivo (suma), se verifica si existe un subconjunto del conjunto dado cuya suma sea igual al valor objetivo especificado.
Por ejemplo, considerando el conjunto {3, 1, 4, 2, 8} y un valor objetivo de 9. En este caso, el problema de SUBSET-SUM busca determinar si hay un subconjunto de los números dados cuya suma sea igual a 9.
El subconjunto {3, 1, 4} suma 8, pero no es igual a 9.
Sin embargo, el subconjunto {3, 2, 4} suma 9, lo que satisface el objetivo.

La reducción de SUBSET-SUM a PARTITION propuesta se basa en transformar una instancia del problema SUBSET-SUM en una instancia equivalente del problema PARTITION, de modo que si podemos resolver eficientemente el problema PARTITION, también podemos resolver eficientemente el problema SUBSET-SUM.

Los pasos que se realizarán para la reducción serán:

a) Dada una instancia de SUBSET-SUM con un conjunto de enteros X y un número objetivo t:

b) Calculamos la suma S de todos los elementos en X.

c) Calculamos el valor: S-2t

d) Creamos un nuevo conjunto X_0 añadiendo S-2t al conjunto original X :
X ∪ {s − 2t}
e) Alimentamos el conjunto X_0 en el algoritmo de SET-PARTITION.

f) Si el algoritmo de SET-PARTITION devuelve que es posible dividir el conjunto en dos partes con sumas iguales, entonces devuelve true, lo que indica que también existe un subconjunto en X cuyos elementos suman exactamente t

g) Si el algoritmo devuelve que nos es posible dividir el conjunto en dos partes con sumas iguales, entonces devuelve false, lo que indica que no existe tal subconjunto en X.

Esto nos demuestra que nuestro problema B: SUBSET-SUM puede ser reducido a A: PARTITION.

3.- LA REDUCCIÓN DE B a A TRABAJA EN UN TIEMPO POLINOMIAL.

Para ver que trabaja en un tiempo polinomial vamos a ver la complejidad de las operaciones realizadas en el apartado anterior.

Cálculo de la suma S: Calcularla suma de los elementos en X implica sumar n elementos donde n es el tamaño del conjunto X, luego tendrá una complejidad O(n)

Cálculo de S-2t: Restar dos números tiene una complejidad de tiempo constante.

Crear el nuevo conjunto X_0: Agregar un elemento a un conjunto existente sería constante, que en este caso sería O(n^k)

Alimentar X_0 al algorimo PARTITION: Tendrá complejidad polinomial ya que nuestro algoritmo PARTITION tiene tiempo polinomial.

Concluimos entonces que la reducción en su conjunto funciona en tiempo polinomial. Esto significa que la transformación de una instancia de SUBSET-SUM a una instancia de PARTITION se realiza en tiempo polinomial, lo que demuestra que PARTITION es al menos tan difícil como SUBSET-SUM en términos de complejidad computacional.

4.- El problema original A tiene solución si y solo si B tiene.

Vamos a demostrar que ⟨X, t⟩ ∈ SUBSET-SUM iff ⟨X'⟩ i ∈ SET-PARTITION .

⇒: Supongamos que existe un conjunto de números en X que suman t. Entonces, los números restantes en X suman s-t. Por lo tanto, existe una partición de X_0 en dos conjuntos tales que cada partición suma s-t. Esto se debe a que al agregar s-2t a X para formar X_0, la suma total de X_0 $es\ 2s - 2t$, por lo que cada partición debe sumar s-t. Por lo tanto X_0 pertenece a PARTITION.

⇐: Supongamos ahora que existe una partición de X_0 en dos subconjuntos tales que la suma sobre cada conjunto es s-t. Uno de estos conjuntos contiene el número s-2t. Si eliminamos este número, obtenemos un conjunto de números cuya suma es t, y todos estos números están en X. Por lo que⟨X, t⟩ pertenece a SUBSET-SUM.

En conclusión, hemos demostrado que ⟨X, t⟩ ∈ SUBSET-SUM iff ⟨X'⟩ i ∈ SET-PARTITION.

CONCLUSIÓN

Como hemos demostrado que los anteriores puntos se cumplían, podemos decir que **PARTITION es NP-Completo.**

PREGUNTAS:

1) ¿Qué función cumple el gadget en la reducción de SUBSET-SUM a PARTITION?
2) ¿Cuál es la estrategia principal para demostrar que PARTITION es NP-Completo?
3) ¿Qué propiedad clave de los problemas NP-Completos se puede utilizar para poder demostrar que PARTITION pertenece a NPC?

REFERENCIAS:

https://www.andrew.cmu.edu/user/ko/pdfs/lecture-21.pdf

https://cs.stackexchange.com/questions/79767/prove-partition-is-np-complete-using-that-subsetsum-so-is-it

https://ai.dmi.unibas.ch/_files/teaching/fs16/theo/slides/theory-e05-handout4.pdf

https://www.geeksforgeeks.org/subset-sum-problem-dp-25/

https://web.njit.edu/~marvin/cs341/hw/hwsoln13.pdf

https://www.geeksforgeeks.org/set-partition-is-np-complete/?ref=lbp