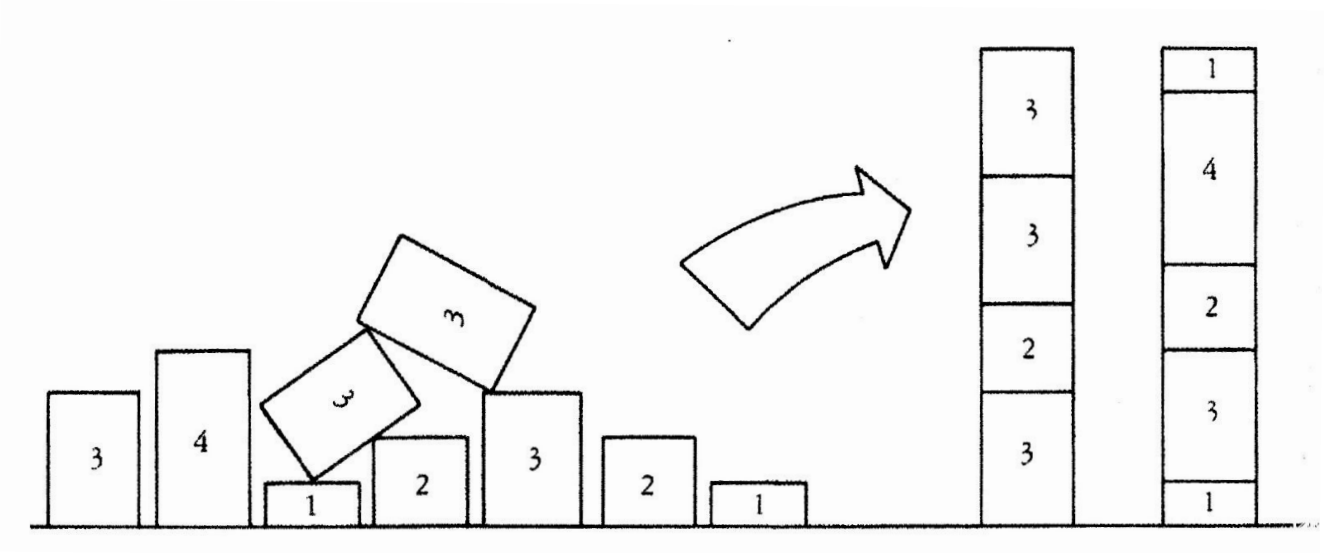
Partition Problem



Francisco Javier Arocas Herrera *alu0100906813@ull.edu.es*

Óscar Hernández Díaz *alu0101127163@ull.edu.es*

Gabriel Melián Hernández *alu0100819786@ull.edu.es*

Adrián Epifanio Rodríguez Hernández *alu0101158280@ull.edu.es*

Índice

1. Introducción

En ciencias de la computación, el Problema de la partición es un problema NP-completo, que visto como un problema de decisión, consiste en decidir si, dado un multiconjunto de números enteros, puede este ser particionado en dos "mitades" tal que sumando los elementos de cada una, ambas den como resultado la misma suma.

Más precisamente, dado un multiconjunto S de enteros: ¿existe alguna forma de dividir S en dos subconjuntos S1 y S2, tal que la suma de los elementos en S1 sea igual que la suma de los elementos en S2?

El problema de partición es equivalente a un caso particular del problema de la suma de subconjuntos, el cual dice: dado un conjunto S de enteros, ¿existe algún subconjunto S1 de S cuyos elementos suman exactamente t /2, donde t es la suma de todos los elementos de S? La equivalencia puede verse definiendo S2 como la diferencia S − S1. Por lo tanto, la solución con programación dinámica existente para resolver el problema de suma de subconjuntos, utilizando tiempo pseudo-polinómico, también es aplicable al problema de partición.

3.3DM a Partition

A continuación, definiremos 3DM, Partition, y veremos cómo podemos transformar desde 3DM a Partition:

a. Definición de 3DM

En matemáticas, el 3DM (3 Dimensional Matching) es una generalización del bipartite matching (O conocido como el 2-dimensional matching). Podemos definirlo de la siguiente forma:

Dado un conjunto X, Y, Z que es finito, y disjunto. Es decir, que no tienen ningún elemento en común. Y T, el cual es un subconjunto de $X \times Y \times Z$.

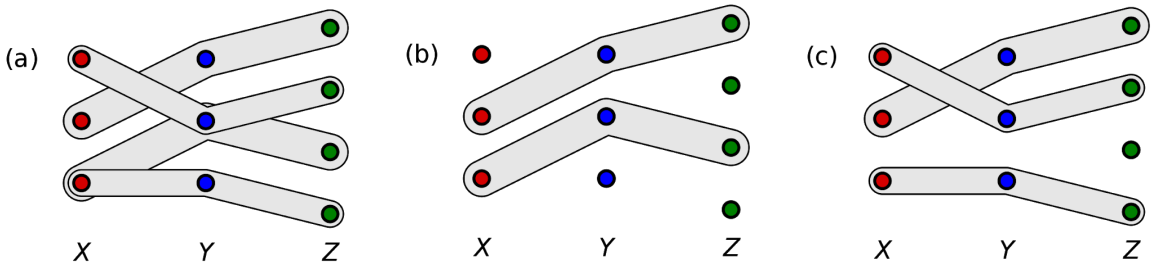
T se compone de tripletas (x, y, z), tal que $x \varepsilon X, y \varepsilon Y, z \varepsilon Z$.

Entonces, $M \subseteq T$, es un 3DM, en los siguiente caso: Para tres tripletas distintas $(x_1, y_1, z_1) \varepsilon M$ y $(x_2, y_2, z_2) \varepsilon M$. Tenemos que: $x_1 \neq x_2, y_1 \neq y_2, z_1 \neq z_2$.

Ejemplo

Podemos ver en la siguiente figura, combinaciones tridimensionales, donde cada conjunto está señalado por un color:

- ➔ **Conjunto X**: Rojo
- ➔ **Conjunto Y**: Azul
- ➔ **Conjunto Z**: Verde



Entrada inicial *Solución* *Otra posible solución*

En la primera imágen, vemos el conjunto T, formado por áreas grises. La segunda imagen, muestra un 3DM M con $|M| = 2$. La tercera imagen muestra un 3DM con $|M| = 3$. Este último, es un emparejamiento tridimensional máximo, es decir, maximiza $|M|$

b. Definición de Partition

Podemos definir como el **problema de la partición** (*Partition Problem),* como, dado un conjunto S de números enteros positivos, dividirlo en dos subconjunto $S_1 \, y \, S_2$. De forma que la suma de los números del primer conjunto es igual a la suma de los números del segundo conjunto. Por ejemplo:

Si tenemos un conjunto $S = \{3, 1, 1, 2, 2, 1\}$. Una solución del partition problem, podría ser, los dos siguientes subconjuntos:

- ➔ $S_1 = \{3, 1, 1\}$
- ➔ $S_2 = \{2, 2, 1\}$

Como podemos ver, ambos suman 5. (3 + 1 + 1 = 5) y (2 + 2 + 1) = 5. Este no es la única solución, ya que los siguientes subconjuntos, también son válidos:

- ➔ $S_1 = \{1, 1, 1, 2\}$
- ➔ $S_2 = \{2, 3\}$

Donde podemos ver, que la suma de los elementos de cada subconjunto, también suma 5.

c. Transformar 3DM a Partition

Usaremos de ejemplo la siguiente entrada en 3DM:

```
X = { a b c }
Y = { 1 2 3 }
Z = { x y z }
T = { a 2 y } , { b 1 z }, { c 3 x }, { a 1 x }
```

Al ser el conjunto T de tamaño 4, tendremos que hacer cuatro conjuntos binarios. Cada uno del conjunto de T. Lo que haremos será sustituir 001, en la posición que se encuentre la letra o número en su conjunto. Vamos a verlo mejor con un ejemplo:

$S(a1) = \{a \, 2 \, y\}$

Tenemos que a, se encuentra en la primera posición de X, por lo tanto, el primer conjunto de bits será: $001 \; 000 \; 000$. Sin embargo, el número dos, se encuentra en la segunda posición, por lo tanto: $000 \; 001 \; 000$. Lo mismo ocurre con la y, que se encuentra en la segunda posición. Por lo tanto el resultado de $s(a1)$ es:

$s(a1) = \{a, 2, y\} = [001 \; 000 \; 000 \mid 000 \; 001 \; 000 \mid 000 \; 001 \; 000]$

Si hacemos lo mismo con los otros tres conjuntos, pues nos queda de la siguiente forma:

$s(a1) = \{ a, 2, y \} = [001\ 000\ 000 \mid 000\ 001\ 000 \mid 000\ 001\ 000]$
$s(a2) = \{ b, 1, z \} = [000\ 001\ 000 \mid 001\ 000\ 000 \mid 000\ 000\ 001]$
$s(a3) = \{ c, 3, x \} = [000\ 000\ 001 \mid 000\ 000\ 001 \mid 001\ 000\ 000]$
$s(a4) = \{ a, 1, x \} = [001\ 000\ 000 \mid 001\ 000\ 000 \mid 001\ 000\ 000]$

El siguiente paso es calcular el sumatorio de los cuatro vectores de bits. Realizamos una suma binaria. El resultado de esta suma es:

$A = \Sigma(s(a)) = [010\ 001\ 001 \mid 010\ 001\ 001 \mid 010\ 001\ 001]$

El siguiente paso es calcular B, que es la suma binario de X, Y, Z de la siguiente forma:

$B = \{ \{a, b, c\}, \{1\ 2\ 3\}, \{x\ y\ z\ \} \}$
$B = [001\ 001\ 001 \mid 001\ 001\ 001 \mid 001\ 001\ 001]$

Tenemos que calcular $C\ y\ D$, cuya fórmula es la siguiente:

$C = 2(A) - B)$
$D = A + B$

Vamos a calcular C:
$2 * A = [100\ 010\ 010 \mid 100\ 010\ 010 \mid 100\ 010\ 010]$
$2 * A - B = [011\ 001\ 001 \mid 011\ 001\ 001 \mid 011\ 001\ 001]$

Por lo tanto es:
$C = [011\ 001\ 001 \mid 011\ 001\ 001 \mid 011\ 001\ 001]$

Calculamos D, que es:
$D = A + B = [011\ 010\ 010 \mid 011\ 010\ 010 \mid 011\ 010\ 010]$

Ahora pasamos los números a decimal, nos quedaría como resultado:

$s(a1) = [001\ 000\ 000 \mid 000\ 001\ 000 \mid 000\ 001\ 000] = 16781320$

$s(a2) = [000\ 001\ 000 \mid 001\ 000\ 000 \mid 000\ 000\ 001] = 2129921$

$s(a3) = [000\ 000\ 001 \mid 000\ 000\ 001 \mid 001\ 000\ 000] = 262720$

$s(a4) = [001\ 000\ 000 \mid 001\ 000\ 000 \mid 001\ 000\ 000] = 16810048$

$C = [011\ 001\ 001 \mid 011\ 001\ 001 \mid 011\ 001\ 001] = 52794057$

$D = [011\ 010\ 010 \mid 011\ 010\ 010 \mid 011\ 010\ 010] = 55157970$

Si ordenamos todos los resultados dentro de un vector, y además, lo ordenamos, nos quedaría de la siguiente forma:

$V = [262720, 2129921, 16781320, 16810048, 52794057, 55157070]$

Entonces, al aplicar el *partition*, en dos conjuntos. Nos da el siguiente resultado:

$V1 = [262720, 2129921, 16781320, 52794057]$

$V2 = [16810048, 55157970]$

El resultado, utilizando nuestro terminal:

```
Original Vector:
        [ 262720, 2129921, 16781320, 16810048, 52794057, 55157970 ]

V1:
        { 262720, 2129921, 16781320, 52794057 }

V2:
        { 16810048, 55157970 }


3DM Data:
         Vector X: { a, b, c }

         Vector Z: { x, y, z }
         Triplets: [ { a, 2, y } { b, 1, z } { c, 3, x } { a, 1, x }  ]
```

5. Definición del problema y variantes

Aunque es un problema NP-completo existe una solución pseudo polinomial y además existen heurísticas que resuelven el problema en muchos casos, ya sea de manera óptima o aproximada. Debido a esta condición, el problema Partition es conocido como “el problema difícil más fácil”.

Existe una versión de optimización del problema que consiste en dividir el conjunto S en dos subconjuntos S1 y S2 de forma que se minimice la diferencia entre la suma de elementos en S1 y la suma de elementos en S2. La versión de Optimización es NP-Hard, pero puede resolverse eficientemente en la práctica.

El problema Partition es un caso especial de dos problemas relacionados:

- En el problema de la suma de subconjuntos, el objetivo es encontrar un subconjunto de S cuya suma sea un cierto número objetivo T dado como entrada (El problema de Partition es el caso especial en el que T es la mitad de la suma de S).
- En el Partition de números de múltiples vías, hay un parámetro de número entero k, y el objetivo es decidir si S se puede dividir en k subconjuntos de igual suma, el problema Partition es el caso especial en el que k sea igual a 2.

Una variación de este problema es el problema de la 3-Partition, en donde el conjunto S debe particionarse en |S|/3 subconjuntos que sumen lo mismo.

Sin embargo, es bastante diferente al problema de 3-Partition, donde el número de subconjuntos no se fija de antemano; debería ser |S| / 3, donde cada subconjunto debe tener exactamente 3 elementos. el 3-Partition es mucho más difícil que el Partition y no tiene un algoritmo de tiempo pseudo polinomial a menos que P= NP, esto es porque el problema de 3-Partition permanece en la clase NP-completa incluso utilizando codificación unaria.

La restricción de requerir que el Partition tenga el mismo tamaño o que todos los enteros de entrada sean distintos también es NP-Hard.

El Partition product, es una variante del Partition donde el problema consiste en dividir un conjunto de números enteros en dos conjuntos con el mismo producto (en lugar de la misma suma), y es un problema NP-Hard complicado.

6. Dificultad computacional

El problema de Partition es NP-Hard. Esto puede demostrarse reduciendo el problema de la suma de subconjuntos. Una instancia de la suma de subconjuntos consta de un conjunto de S enteros positivos y una suma objetivo T < S, el objetivo es decidir si hay un subconjunto de S con suma exactamente T.

Dada dicha instancia, se construye una instancia de Partition en la que el conjunto de entrada contenga el conjunto original más dos elementos: z1 y z2, siendo z1 igual a la suma (S) y siendo z2 igual a 2T. La suma de este conjunto de entrada es suma(S) + z1 + z2 = 2 * suma(S) + 2T, por lo que la suma objetivo para el Partition es Suma(S) + T.

Suponemos que existe una solución S' para la instancia de la suma de subconjuntos. Entonces si suma(S') = T, entonces Suma(S' ∪ {z1}) = Suma(S) + T, por lo tanto S' ∪ {z1} es una solución a la instancia de Partition.

A la inversa, suponemos que existe una solución S'' para la instancia de Partition. Entonces, S'' debe contener z1 o z2, pero no ambos, ya que su suma es mayor que Suma(S) + T. Si S'' contiene z1, entonces debe contener elementos de S con una suma de exactamente T, entonces S'' menos z1 es una solución a la instancia de la suma de subconjuntos. Si S'' contiene z2, entonces debe contener elementos de S con una suma de exactamente Sum(S)-T, por lo que los otros objetos en S son una solución para la instancia de suma de subconjuntos.

7. Algoritmos aproximados

El problema de Partition es un caso especial de Partition de múltiples vías y de suma de subconjuntos. Por tanto, se puede resolver mediante algoritmos desarrollados para cada uno de estos problemas. Los algoritmos para el Partition de múltiples vías incluyen:

- **Greedy number Partitioning**: donde se recorren los números y se coloca cada número en el conjunto cuya suma actual es la más pequeña. Si los números no están ordenados, entonces el tiempo de ejecución es O(n) y la relación de aproximación es como máximo 3/2. La relación de aproximación significa la suma mayor en la salida del algoritmo, dividida por la suma mayor en una partición óptima.
- Ordenar los números aumenta el tiempo de ejecución a O(n log n) y mejora la relación de aproximación a 7/6. Si los números se distribuyen uniformemente en [0,1], entonces la proporción de aproximación es como máximo 1+O(log log n/n) casi seguro y 1+O(1/n) en expectativa.

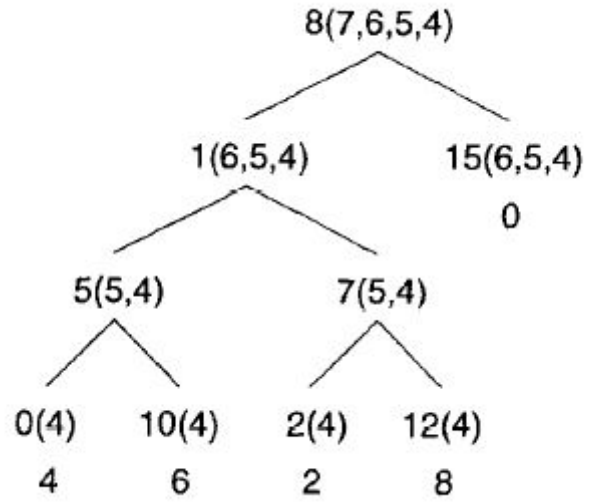
- **Largest Differencing Method** (también conocido como Karmarkar-Karp algorithm), ordena los números en orden descendente y reemplaza repetidamente los números por sus diferencias. La complejidad del tiempo de ejecución es O(n log n). En el peor de los casos, su relación de aproximación es similar, como máximo 7/6. Sin embargo, en el caso promedio, funciona mucho mejor que el algoritmo anterior, cuando los números se distribuyen uniformemente en [0,1], su relación de aproximación es como máximo 1+1/n^O(log n) en expectativa. También funciona mejor en experimentos de simulación.

- **El algoritmo Multifit**: utiliza una búsqueda binaria combinada con un algoritmo para empaquetar contenedores. En el peor de los casos, su relación de aproximación es 8/7.

8. Algoritmos exactos

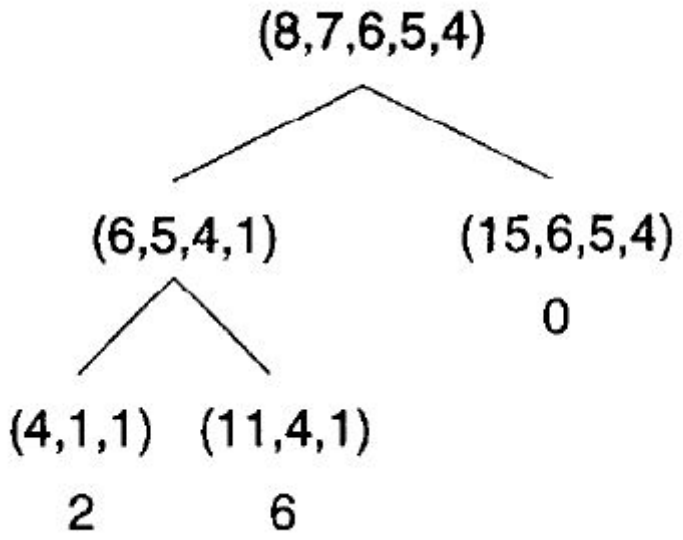
Hay algoritmos exactos, que siempre encuentran la partición óptima. Dado que el problema es NP-complejo, tales algoritmos pueden tomar un tiempo exponencial en general, pero pueden ser prácticamente utilizables en ciertos casos. Los algoritmos desarrollados para la partición de números de múltiples vías incluyen:

- El **Particionamiento de número de tiempo pseudo polinomial** (*Pseudo polynomial time number partitioning)*: Ocupa $O(nm)$ de memoria. Donde m es el número más grande en la entrada.
- El **Complete Greedy Algorithm (CGA)** (*O algoritmo Greedy Completo*): considera todas las particiones mediante la construcción de un árbol binario. Cada nivel del árbol corresponde a un número de entrada, donde la raíz corresponde al número más grande, el nivel inferior al siguiente número más grande, etc. Cada rama corresponde a un conjunto diferente en el que se puede colocar el número actual. Atravesar el árbol en orden de profundidad requiere solo $O(n)$ espacio, pero puede tomar $O(2^n)$ tiempo. El tiempo de ejecución se puede mejorar utilizando una heurística greedy: en cada nivel, se desarrolla primero la rama en la que se coloca el número actual en el conjunto con la suma más pequeña. Este algoritmo encuentra primero la solución encontrada por la partición numérica greedy, pero luego procede a buscar mejores soluciones. Algunas variaciones de esta idea son esquemas de aproximación de tiempo polinomial para el problema de suma de subconjuntos y, por lo tanto, también para el problema de partición.
  A continuación, un ejemplo del árbol binario resultante:



Este sería un ejemplo del conjunto: |8, 7, 6, 5, 4 |. Vemos como se ordena de mayor a menor, y se mantiene un elemento fuera del paréntesis (Que en la raíz del árbol es el de mayor peso). Posteriormente, en la rama derecha se suma el elemento que se encuentra fuera, con el de mayor tamaño del conjunto del paréntesis, y en la rama izquierda es el mismo proceso pero restando. El final de la rama es, cuando la suma de los elementos dentro del paréntesis es menor o igual al número que se encuentra fuera de este.

- El **Complete Karmarkar-Karp algorithm** (*CKK*) (O *algoritmo completo Karmarkar-Karp*) considera todas las particiones mediante la construcción de un árbol binario. Cada nivel corresponde a un par de números. La rama izquierda corresponde a ponerlos en diferentes subconjuntos (es decir, reemplazarlos por su diferencia), y la rama derecha corresponde a ponerlos en el mismo subconjunto (es decir, reemplazarlos por su suma). Este algoritmo encuentra primero la solución encontrada por el método de diferenciación más grande, pero luego procede a encontrar mejores soluciones. Funciona sustancialmente más rápido que CGA en instancias aleatorias. Su ventaja es mucho mayor cuando existe una partición igual y puede ser de varios órdenes de magnitud. En la práctica, los problemas de tamaño arbitrario pueden resolverse con *CKK* si los números tienen como máximo 12 dígitos significativos. *CKK* también puede ejecutarse como un algoritmo en cualquier momento: primero encuentra la solución KK y luego encuentra soluciones progresivamente mejores a medida que el tiempo lo permite (posiblemente requiriendo tiempo exponencial para alcanzar la optimización, en los peores casos). Requiere $O(n)$ espacio, pero en el peor de los casos podría tomar $O(2^n)$ tiempo.
  En la siguiente imagen un ejemplo:



  Vemos que su árbol binario es parecido al del algoritmo anterior. Pero en este caso, se mantiene el elemento resultante dentro del conjunto del paréntesis. Como vemos, esto hace que el árbol sea de mayor tamaño, además de que el algoritmo es más eficiente.

Los algoritmos desarrollados para la suma de subconjuntos incluyen:

- **Horowitz y Sahni:** Consume $O(2^{n/2} \cdot (n/2))$ de tiempo, pero requiere $O(2^{n/2})$ de espacio.
- **Schroeppel and Shamir:** Consume $O(2^{n/2} \cdot (n/4))$ de tiempo y requiere mucho menos espacio: $O(2^{n/4})$
- **Howgrave-Graham y Joux:** Consume $O(2^{n/3})$ pero es un algoritmo aleatorio que sólo resuelve el problema de decisión (no el problema de optimización) .

9. Instancias difíciles y transición de fase

Los conjuntos con solo una o ninguna partición tienden a ser más difíciles (o más costosos) de resolver en comparación con sus tamaños de entrada. Cuando los valores son pequeños en comparación con el tamaño del conjunto, es más probable que haya particiones perfectas. Se sabe que el problema sufre una "***transición de fase***"; siendo probable para algunos conjuntos y poco probable para otros. Si m es el número de bits necesarios para expresar cualquier número del conjunto y n es el tamaño del conjunto, entonces $m/n < 1$ tiende a tener muchas soluciones y $m/n > 1$ tiende a tener pocas o ninguna solución. A medida que nym se hacen más grandes, la probabilidad de una partición perfecta pasa a 1 o 0 respectivamente. Esto fue originalmente argumentado en base a evidencia empírica de Gent y Walsh, luego usando métodos de física estadística de Mertens, 130 y luego probado por Borgs, Chayes y Pittel.

10. Versión probabilística

Un problema relacionado, algo similar a la paradoja del cumpleaños (*Birthday problem*), es el de determinar el tamaño del conjunto de entrada de modo que tengamos una probabilidad de la mitad de que haya una solución, bajo el supuesto de que cada elemento del conjunto se selecciona aleatoriamente con distribución uniforme entre 1 y algún valor dado. La solución a este problema puede ser contraria a la intuición, como la paradoja del cumpleaños.

En la teoría de la probabilidad, el **problema del cumpleaños** (*Birthday problem*) o la paradoja del cumpleaños se refieren a la probabilidad de que, en un conjunto de n personas elegidas al azar, alguna pareja de ellos tenga el mismo cumpleaños. Según el principio, la probabilidad alcanza el 100% cuando el número de personas llega a 367 (ya que solo hay 366 cumpleaños posibles, incluido el 29 de febrero). Sin embargo, el 99,9% de probabilidad se alcanza con solo 70 personas y el 50% de probabilidad con 23 personas. Estas conclusiones se basan en la suposición de que cada día del año (excluyendo el 29 de febrero) es igualmente probable para un cumpleaños.

11. Ejemplos

Dado ***S = {3,1,1,2,2,1}***, una solución válida al problema de la partición son los dos conjuntos ***S1 = {1,1,1,2}*** y ***S2 = {2,3}***. Ambos conjuntos suman 5 y se dividen en S. Hay que tener en cuenta que esta solución no es única. ***S1 = {3,1,1}*** y ***S2 = {2,2,1}*** es otra solución.

No todos los conjuntos múltiples de enteros positivos tienen una partición en dos subconjuntos con igual suma. Un ejemplo de tal conjunto es ***S = {2,5}***.

12. Aplicaciones

Una aplicación del problema de la partición es la manipulación de las elecciones. Suponga que hay tres candidatos (A, B y C). Un solo candidato debe ser elegido utilizando una regla de votación basada en la puntuación. Ej. la regla del veto (cada votante vetas a un solo candidato y el candidato con menos vetos gana). Si una coalición quiere asegurarse de que C sea elegido, debe dividir sus votos entre A y B para maximizar el menor número de votos que cada uno de ellos obtiene. Si se ponderan los votos, entonces el problema se puede reducir al problema de la partición y, por lo tanto, se puede resolver de manera eficiente utilizando CKK. Lo mismo es cierto para cualquier otra regla de votación que se base en la puntuación.

13. Bibliografía

- Wikipedia: https://en.wikipedia.org/wiki/Partition_problem
- Definición: https://arxiv.org/ftp/cond-mat/papers/0310/0310317.pdf
- Github: https://github.com/AdrianEpi/CC-Partition