Universidad de La Habana

Facultad de Matemática y Computación

Proyecto de Diseño de Análisis y Algoritmos

Análisis y resolución de ejercicios de la plataforma Codeforces

Raudel Alejandro Gómez Molina C-411

Proyecto en github

Índice

1. Reducción de Emparejamiento

1.1. Ejercicio

Link del problema en la plataforma **Codeforces**

- límite de tiempo por test: 8 segundos
- límite de memoria por test: 512 megabytes

Te dan un grafo bipartito con n_1 vértices en la primera parte, n_2 vértices en la segunda parte, y m aristas. El emparejamiento máximo en este grafo es el subconjunto máximo posible (en tamaño) de aristas de este grafo tal que ningún vértice está incidente a más de una arista elegida.

Debes procesar dos tipos de consultas en este grafo:

- 1 — elimina el número mínimo posible de vértices de este grafo para que el tamaño del emparejamiento máximo se reduzca exactamente en 1, e imprime los vértices que has eliminado. Luego, encuentra un emparejamiento máximo en este grafo e imprime la suma de los índices de las aristas que pertenecen a este emparejamiento;
- 2 — este tipo de consulta se realizará solo después de una consulta de tipo 1. Como respuesta a esta consulta, debes imprimir las aristas que forman el emparejamiento máximo que has elegido en la consulta anterior. Ten en cuenta que debes resolver el problema en modo en línea. Esto significa que no puedes leer toda la entrada de una vez. Solo puedes leer cada consulta después de escribir la respuesta a la consulta anterior. Usa las funciones `fflush` en C++ y `BufferedWriter.flush` en Java después de cada escritura en tu programa.

Entrada

La primera línea contiene cuatro enteros n_1, n_2, m y q ($1 \le n_1, n_2 \le 2 \cdot 10^5; 1 \le m \le \min(n_1 \cdot n_2, 2 \cdot 10^5); 1 \le q \le 2 \cdot 10^5$).

Luego siguen m líneas. La i-ésima contiene dos enteros x_i y y_i ($1 \le x_i \le n_1; 1 \le y_i \le n_2$), lo que significa que la i-ésima arista conecta el vértice x_i en la primera parte y el vértice y_i en la segunda parte. No hay pares de vértices que estén conectados por más de una arista.

Luego siguen q líneas. La i-ésima contiene un entero, 1 o 2, indicando la consulta i-ésima. Restricciones adicionales sobre las consultas:

- el número de consultas de tipo 1 no superará el tamaño del emparejamiento máximo en el grafo inicial;
- el número de consultas de tipo 2 no superará 3;
- cada consulta de tipo 2 será precedida por una consulta de tipo 1;
- tu solución solo puede leer la i-ésima consulta después de imprimir la respuesta a la consulta $(i-1)$-ésima y vaciar el búfer de salida.

Salida

Para una consulta de tipo 1, imprime la respuesta en tres líneas de la siguiente manera:

- la primera línea debe contener el número de vértices que eliminas;
- la segunda línea debe contener los índices de los vértices que eliminas, de la siguiente manera: si eliminas el vértice x de la parte izquierda, imprime x; si eliminas el vértice y de la parte derecha imprime $-y$ (índice negativo);
- la tercera línea debe contener la suma de los índices de las aristas en algún emparejamiento máximo en el grafo resultante. Las aristas están numeradas del 1 al m.

Para una consulta de tipo 2, imprime la respuesta en dos líneas de la siguiente manera:

- la primera línea debe contener el tamaño del emparejamiento máximo;
- la segunda línea debe contener los índices de las aristas que pertenecen al emparejamiento máximo. Ten en cuenta que la suma de estos índices debe ser igual al número que imprimiste al final de la consulta de tipo 1 anterior.

Después de imprimir la respuesta a una consulta, no olvides vaciar el búfer de salida.

1.2. Etiquetas

- Flujo
- Emparejamiento
- Cubrimiento Mínimo

1.3. Solución

En el ejercicio se tiene un grafo **bipartito** en el que se debe responder a dos tipos de consultas, en la primera se debe eliminar la mínima cantidad de vértices para que el emparejamiento máximo del grafo se reduzca exactamente en 1 y en la segunda se debe informar las aristas que pertenecen al emparejamiento máximo en el grafo que se tiene actualmente. Una observación importante es que la interactividad del ejercicio no permite responder las consultas de forma **offline** (recibir todas las consultas hacer algún tipo de preprocesamiento conociendo todas las posibles consultas y luego responder cada una de forma individual), por lo que es necesario responder las consultas de forma individual sin conocer las demás e ir manteniendo el estado del grafo.

Definiciones:

- Se define un emparejamiento máximo de un grafo G como $M(G)$.
- Se define un cubrimiento mínimo de un grafo G como $C(G)$.
- Se define el grafo inducido de $G = V, E$, por V' tal que $V' \subset V$ como $I(G, V')$.

1.3.1. Fuerza bruta

El primer acercamiento para resolver este problema es primeramente encontrar un emparejamiento máximo en el grafo de entrada, para esto ya que dicho grafo es **bipartito** se puede usar el algoritmo de **Khun** (basado en el teorema de **Khun**) o **Flujo Máximo**, estos dos algoritmos funcionan en una complejidad de $O(VE)$.

Luego cuando se necesite analizar una consulta del tipo 1, se puede iterar por todos los posibles conjuntos de vértices V' tal que $V' \subset V$ y $|M(I(G, V'))| = |M(G)| - 1$ y elegir el conjunto de mayor cardinalidad, este algoritmo es algo lento ya que su complejidad es de $O(2^{|V|}VE)$.

Para las consultas de tipo 2 solo se debe guardar el emparejamiento encontrado en la consulta de tipo 1 anterior e informar las aristas que pertenecen a dicho emparejamiento, en una complejidad de $O(q)$ donde q es la cardinalidad del emparejamiento máximo del grafo actual.

1.3.2. Teorema Kőnig

El teorema de **Kőnig** plantea que dado un grafo **bipartito** G se cumple que $|C(G)| = |M(G)|$, además plantea una equivalencia entre estos 2 conjuntos es decir como dado un emparejamiento máximo encontrar un cubrimiento mínimo y viceversa.

Demostración:
Sea un grafo **bipartito** $G = V, E$ con A y B los conjuntos de vértices que representan la bipartición, definamos $Q = M(G)$ y $A_q = \{x \in A \mid \exists y \text{ tal que } \langle x, y\rangle \in Q\}$ y $B_q = \{y \in B \mid \exists x \text{ tal que } \langle x, y\rangle \in Q\}$.

En primer lugar si se tiene un cubrimiento de vértices W se cumple que 2 aristas de Q no inciden sobre un mismo vértice de W porque de lo contrario Q no sería un emparejamiento, por otro lado cada arista en Q debe ser cubierta por un vértice de W ya que de lo contrario W no sería un cubrimiento de vértices. Por tanto si $|Q| = |W|$ entonces se puede decir que W es un cubrimiento mínimo.

Sea Z el conjunto de formado por la unión de $A \setminus A_q$ y el conjunto de vértices que a los que se puede llegar usando caminos Q alternantes partiendo de los vértices de $A \setminus A_q$.

Ahora si se define $K = (A_q \setminus Z) \cup (B_q \cap Z)$ se puede demostrar que K es un cubrimiento de vértices. Observe que para cada arista $\langle x, y\rangle \in E$ con $x \in A$ y $y \in B$ se cumple que $x \in A_q \vee y \in B_q$ porque si no Q no sería máximo: si $\langle x, y\rangle$ pertenece a uno de los caminos Q alternantes entonces $y \in B_q \cap Z$ porque todas las aristas que pertenecen a un camino Q alternante tienen un vértice en B_q, de lo contrario se tendría un camino Q aumentativo; ahora si $\langle x, y\rangle$ no pertenece a ninguno de los caminos Q alternantes entonces $x \in A_q \setminus Z$ porque si $x \notin A_q$ se cumple que $\langle x, y\rangle$ pertenece a un camino Q alternante.

Solo resta demostrar que $|K| = |Q|$. Observe que para toda arista $\langle x, y\rangle \in Q$ con $x \in A$ y $y \in B$ se cumple que: si $\langle x, y\rangle$ pertenece a uno de los caminos Q alternantes entonces $y \in B_q \cap Z \wedge x \notin A_q \setminus Z$ porque $x \in Z$; ahora si $\langle x, y\rangle$ no pertenece a ninguno de los caminos Q alternantes entonces $x \in A_q \setminus Z \wedge y \notin B_q \cap Z$ porque $x \notin Z$. Entonces como $(A_q \setminus Z) \cap (B_q \cap Z) = \emptyset$ por lo enunciado anteriormente se cumple que $|K| = |Q|$.

1.3.3. Obtener el cubrimiento mínimo

Como se demostró anteriormente en el teorema de **Kőnig** existe una equivalencia entre el emparejamiento máximo y el cubrimiento mínimo en un grafo **bipartito**.

Si se toma la demostración anterior para encontrar un cubrimiento mínimo es necesario primeramente encontrar un emparejamiento máximo, para esto se puede usar el algoritmo de **Khun** (basado en el teorema de **Khun**) o **Flujo Máximo**, ambos en una complejidad de $O(VE)$.

Luego se puede encontrar el conjunto de vértices Z mediante un **DFS** restringiendo el uso de caminos que no sean Q alternantes, en una complejidad de $O(V + E)$.

Solo resta formar el conjunto K, lo cual se puede lograr mediante los conjuntos A, B, A_q, B_q y Z, en una complejidad de $O(V)$.

1.3.4. Utilizando Kőnig y el cubrimiento mínimo

Si se emplea el teorema de **Kőnig** demostrado anteriormente se puede pasar a analizar el problema de forma diferente, es decir existe una equivalencia entre $M(G)$ y $C(G)$ donde G es un grafo **bipartito** ahora se puede buscar la menor cantidad de vértices tal que al eliminarlos la cardinalidad del cubrimiento mínimo disminuya exactamente en 1 y dicho conjunto cumplirá con las restricciones del problema original.

Se puede realizar la siguiente observación si V_c es un cubrimiento de vértices de $G = V, E$ entonces para cualquier vértice $v \in V_c$, sea $V_c' = V_c - v$ y $V' = V - v$, se cumple que V_c' es un cubrimiento de vértices del grafo $I(G, V')$. Para demostrar el anterior enunciado observe que para toda arista $e \in E$ se cumple que e incide sobre alguno de los vértices de V_c, por tanto las únicas aristas de E que no inciden sobre V_c' son las que únicamente incidían sobre v y dichas aristas no pertenecen a $I(G, V')$, por lo que como las aristas de $I(G, V')$ son subconjunto de E entonces se cumple que V_c' cubre todas las aristas de $I(G, V')$.

Ahora dado el razonamiento anterior si V_c es un cubrimiento mínimo de vértices de $G = V, E$ entonces para cualquier vértice $v \in V_c$, sea $V_c' = V_c - v$ y $V' = V - v$, se cumple que V_c' es un cubrimiento mínimo de vértices del grafo $I(G, V')$. Como ya se demostró que V_c' es un cubrimiento de vértices solo falta demostrar que V_c es mínimo. Para ello suponga que existe un cubrimiento menor V_c'' del grafo $I(G, V')$, entonces si se añade v a dicho cubrimiento se obtiene un cubrimiento de G, ya que todas las aristas que estaban en G y no en $I(G, V')$ incidían sobre v lo cual contradice el hecho de que V_c sea mínimo.

Luego si se elimina cualquier vértice v de G que participe en un cubrimiento mínimo se obtiene un grafo que cuya cardinalidad del cubrimiento mínimo disminuye en 1 con respecto a la cardinalidad del cubrimiento mínimo en el grafo original.

Ahora se puede utilizar el resultado demostrado anteriormente para responder eficientemente las consultas de tipo 1. Para ello es necesario primeramente obtener un cubrimiento mínimo en el grafo de entrada (esto se puede lograr mediante el algoritmo descrito anteriormente) y cada vez que se proporcione una consulta de este tipo seleccionar un vértice de dicho conjunto.

Para las consultas de tipo 2, es necesario mantener un emparejamiento que se corresponda con el cubrimiento mínimo de vértices del grafo actual. Para ello se puede emplear el algoritmo descrito anteriormente para encontrar el cubrimiento mínimo en el grafo inicial y cuando se procese una consulta del tipo 1 eliminar la arista del emparejamiento que incide sobre el vértice que se va a eliminar (esto siempre es posible porque para todo vértice del cubrimiento mínimo inicial existe una arista del emparejamiento que incide sobre dicho vértice y para cada arista del emparejamiento inicial exactamente uno de sus 2 vértices pertenece al cubrimiento mínimo), por tanto al eliminar dicha arista se obtiene un emparejamiento con cardinalidad igual al cubrimiento mínimo de vértices y por el teorema de **Kőnig** se puede asegurar que este emparejamiento es máximo.

Entonces para resolver el ejercicio primero es necesario encontrar un cubrimiento mínimo y el emparejamiento máximo asociado a dicho cubrimiento esto tiene una complejidad de $O(VE)$ empleando el algoritmo descrito anteriormente, las consultas de tipo 1 se pueden responder en una complejidad de $O(1)$ y las de tipo 2 en una complejidad de $O(q)$ donde q es la cardinalidad del emparejamiento máximo del grafo actual.

1.4. Implementación

Puede encontrar la implementación del ejercicio en el siguiente enlace enlace.

2. Número de subarreglos k-buenos

2.1. Ejercicio

Link del problema en la plataforma **Codeforces**

- límite de tiempo por prueba: 2 segundos
- límite de memoria por prueba: 256 megabytes

Sea bit(x) el número de unos en la representación binaria de un número entero no negativo x.

Un subarreglo de un arreglo se llama k-bueno si consiste solo en números que no tienen más de k unos en su representación binaria. Es decir, un subarreglo (l, r) del arreglo a es bueno si para cualquier i tal que $l \leq i \leq r$ se cumple la condición bit$(a_i) \leq k$.

Se te da un arreglo a de longitud n, que consiste en enteros no negativos consecutivos que comienzan desde 0, es decir, $a_i = i$ para $0 \leq i \leq n - 1$ (en indexación basada en 0). Necesitas contar el número de subarreglos k-buenos en este arreglo.

Dado que la respuesta puede ser muy grande, imprime el resultado módulo $10^9 + 7$.

Entrada

Cada prueba consiste en múltiples casos de prueba. La primera línea contiene un número entero t ($1 \leq t \leq 10^4$) — el número de casos de prueba. Las siguientes líneas describen los casos de prueba.
La única línea de cada caso de prueba contiene dos enteros n, k ($1 \leq n \leq 10^{18}, 1 \leq k \leq 60$).

Salida

Para cada caso de prueba, imprime un solo número entero: el número de subarreglos k-buenos módulo $10^9 + 7$.

2.2. Etiquetas

- Máscara de bits
- Divide y Vencerás
- Programación Dinámica

2.3. Solución

En el ejercicio se tiene un arreglo de los números de 0 a $n-1$ en orden creciente y se debe responder el número de subarreglos k-buenos. La principal dificultad para el ejercicio es contar eficientemente el número de estos subarreglos así que primeramente se puede abordar un algoritmo de fuerza bruta para resolver el ejercicio.

Nota: En la solución del ejercicio se usarán números soportados por la aritmética de la computadora (hasta 64 bits), por lo que las operaciones con los bits de los números se consideran constantes. Para llevar esta solución a otro modelo de cómputo puede considerar las operaciones de bits sobre n en una complejidad de $O(\log(n))$.

2.3.1. Fuerza bruta

Para resolver este ejercicio observe que se pueden probar con todos los posibles subarreglos y determinar por cada subarreglo si este es k-bueno o no. Para ello se puede encontrar todos los subarreglos en una complejidad de $O(n^2)$ y comprobar si un subarreglo es k-bueno tiene una complejidad de $O(n)$, para una complejidad total de $O(n^3)$.

2.3.2. Mejorando la fuerza bruta

Observe que se puede mejorar el algoritmo anterior haciendo la siguiente observación: si se tiene que un subarreglo (l, r) es k-bueno entonces se puede asegurar que para todo x tal que $l < x \leq r$ se cumple que el subarreglo (l, x) es k-bueno, por otro lado si se tiene que un subarreglo (l, r) no es k-bueno entonces se puede asegurar que para todo x tal que $r \leq x \leq n$ se cumple que el subarreglo (l, x) no es k-bueno.
Ahora si para cada índice l se calcula el mayor r tal que (l, r) es k-bueno se puede obtener el número de subarreglos k-buenos que comienzan en l. Para esto se puede iterar por todos los índices x de 1 hasta n en una complejidad de $O(n)$ y para cada índice x, iterar y desde x hasta que se cumpla que $y > n$ o $\text{bit}(a_y) > k$ con una complejidad de $O(n)$, para una complejidad total de $O(n^2)$.

2.3.3. Divide y Vencerás

Ahora para continuar mejorando la solución del ejercicio se puede utilizar un enfoque se divide y vencerás. Para ello se puede buscar la mayor potencia de 2 que es menor que n o simplemente el último bit activo de n y elegir esta potencia como pivote, observe que en la parte izquierda del pivote solo se deben buscar subarreglos k-buenos de 0 a $2^q - 1$ (q es el último bit activo de n) mientras en la parte derecha del pivote todos los números tienen activado el bit q (q es el último bit activo para cada uno de estos números) por lo que se puede buscar subarreglos $(k-1)$-buenos de 0 a $n - 2^q - 1$ (aquí el truco radica en restar 2^q a todos los números en la parte izquierda del pivote y trabajar desde 0 a $n - 2^q - 1$).

Entonces resta resolver la parte de la conquista eficientemente, es decir buscar los arreglos k-buenos que tengan el extremo izquierdo de a la izquierda del pivote y el extremo derecho a la derecha del del pivote. Para ello se define la función $f(n,k) = (c,l,r)$, la función $f(n,k)$ retorna el número de subarreglos k-buenos de 0 a $n-1$, la longitud del subarreglo k-bueno más grande que comienza en 0 y la longitud de del subarreglo k-bueno más grande que termina en $n-1$, respectivamente, observe que l y r pueden ser igual a 0. Ahora la cantidad de subarreglos pertenecientes a la conquista simplemente es la multiplicación del tamaño del subarreglo más grande que termina en $2^q - 1$ ($l1$ almacena este valor) y el tamaño del subarreglo más grande que comienza en 2^q ($r1$ almacena este valor), ya que existen todos los posibles subarreglo (x,y) k-buenos que pertenecen a la conquista cumplen que $2^q - l1 \leq x \leq 2^q - 1 \wedge 2^q \leq y \leq 2^q + r1 - 1$ por que cualquier otro subarreglo que pertenezca a la conquista y comience antes de $2^q - l1$ o termine después de $2^q + r1 - 1$ no es k-bueno porque si no los valores de $l1$ y $r1$ no estarían correctamente calculados. El resto de la solución usando este método consiste en el mantenimiento de las invariantes descritas, a continuación se presenta un pseudocódigo:

```
f(n, k) {
    q <-- último bit activo de n

    if(k < 0)
        return 0, 0, 0

    if(n == 0)
        return 1, 1, 1

    if(2**q == n)
        q--

    c1, l1, r1 = f(2**q, k)
    c2, l2, r2 = f(n - 2**q, k - 1)

    l = l1 == 2**q ? l1 + l2 : l1
    r = r2 == n - 2**q ? r2 + r1 : r2

    return c1 + c2 + r1 * l2, l, r
}
```

Solo falta analizar la complejidad temporal, en cada llamado recursivo dividimos la entrada en 2 y hacemos 2 llamadas recursivas, la conquista tiene una complejidad de $O(1)$ por lo que el $T(n)$ recursivo quedaría de la siguiente manera: $T(n) = 2T(n/2) + c$, por lo que la complejidad final es $O(n)$.

2.3.4. Programación Dinámica

Si se observa el algoritmo descrito en la sección anterior, cuando se divide el tamaño de la entrada en la parte izquierda del pivote siempre se hace buscan los subarreglos de 0 a $n-1$, donde n siempre es una potencia de 2, por lo que si se memoriza los valores de $f(n,k)$ cuando n es una potencia de 2, la suma de todos los llamados que se realizan en la parte izquierda tiene una complejidad total de $O(\log(n)k)$.

```
f(n, k) {
    q <-- último bit activo de n

    if(k < 0)
        return 0, 0, 0
```

```
    if(n == 0)
        return 1, 1, 1

    if(dp[q][k] and n == 2**q)
        return dp[q][k]

    if(2**q == n)
        q--

    c1, l1, r1 = f(2**q, k)
    c2, l2, r2 = f(n - 2**q, k - 1)

    l = l1 == 2**q ? l1 + l2 : l1
    r = r2 == n - 2**q ? r2 + r1 : r2

    if(2**q == n)
        dp[q][k] = c1 + c2 + r1 * l2, l, r

    return c1 + c2 + r1 * l2, l, r
}
```

Entonces con esta modificación el $T(n)$ recursivo quedaría de la siguiente manera: $T(n) = T(n/2) + c$, por lo que la complejidad del $T(n)$ es $O(\log(n))$ y la complejidad final es $O(\log(n)k)$.

2.4. Implementación

Puede encontrar la implementación del ejercicio en el siguiente enlace enlace.

3. Distribución Justa

Link del problema en la plataforma **Codeforces**

3.1. Ejercicio

- límite de tiempo por prueba: 0.35 segundos
- límite de memoria por prueba: 1024 megabytes

Un empresario tiene N planos, cada uno describiendo un tipo de edificio. Cada plano especifica la altura del edificio mediante dos enteros G y R.

- G: Altura del piso de planta baja. Puede ser cero, lo que indica que el edificio no tiene planta baja.
- R: Altura de cada piso residencial. Cada edificio tiene al menos un piso residencial.

El empresario quiere distribuir todos estos planos entre sus dos hijos, Alicia y Bob. Cada hijo construirá exactamente un edificio de cada plano que se le asigne, eligiendo el número de pisos residenciales para cada edificio.

El empresario quiere evitar mostrar favoritismo hacia cualquiera de los hijos, por lo que busca una distribución justa de los planos. Decidió que una distribución justa es aquella en la que es posible construir los edificios de tal manera que la suma de las alturas de los edificios construidos por cada hijo sea la misma. ¿Puedes decir si existe una distribución justa?

Considera el siguiente ejemplo con $N = 3$ planos:

- $G = 1$ y $R = 1$ (las alturas posibles son 2, 3, 4,...);
- $G = 0$ y $R = 3$ (sin planta baja, las alturas posibles son 3, 6, 9,...);
- $G = 2$ y $R = 1$ (las alturas posibles son 3, 4, 5,...).

En este caso, una distribución justa posible es asignar el segundo plano a Alicia y el resto a Bob. Aunque Alicia recibe un solo plano mientras que Bob recibe dos, pueden construir dos pisos residenciales en el primer tipo de edificio (altura 3), dos pisos residenciales en el segundo (altura 6) y un piso residencial en el tercero (altura 3). De esta manera, la suma de las alturas de los edificios construidos por cada hijo sería 6.

Entrada

La primera línea contiene un número entero N ($1 \leq N \leq 2 \cdot 10^5$) que indica el número de planos.

Cada una de las siguientes N líneas contiene dos enteros G ($0 \leq G \leq 2 \cdot 10^5$) y R ($1 \leq R \leq 10^9$), que denotan respectivamente la altura de la planta baja y la altura de cada piso residencial especificada por el plano correspondiente. La suma de las alturas de las plantas bajas de todos los planos es como máximo $2 \cdot 10^5$.

Salida

Imprime una sola línea con la letra mayúscula **Y** si existe una distribución justa de los planos, y la letra mayúscula **N** en caso contrario.

3.2. Etiquetas

- Problema de Partición
- Suma de Subconjunto
- Teoría de Números
- Mochila DP

3.3. Solución

Para este problema se deben particionar el conjunto de los planos de los edificios en 2 subconjuntos, para asignar a Alice y a Bob respectivamente de tal manera que exista una configuración en la que la suma de los edificios construidos por cada uno sea igual.

Para la solución de este problema primero se eliminará la restricción de la suma de las alturas de las plantas bajas y luego se resolverá el ejercicio usando esta restricción.

Definiciones:

- Se define sum(A) como la suma de todos los elementos del conjunto A.

3.3.1. Teoría de Números

Se define los conjuntos A y B como el conjunto de índices de los edificios que construye Alice y Bob respectivamente. Entonces se cumple que $|A| = p$, $|B| = q$, $p + q = n$, $A = \{a_1, a_2, \ldots a_p\}$, $B = \{b_1, b_2, \ldots b_q\}$, $p > 0$ y $q > 0$.

Luego el ejercicio se reduce a encontrar si existe una solución de a la siguiente ecuación:

$$g_{a_1} + r_{a_1} x_{a_1} + g_{a_2} + r_{a_2} x_{a_2} + \ldots + g_{a_p} + r_{a_p} x_{a_p} = g_{b_1} + r_{b_1} x_{b_1} + g_{b_2} + r_{b_2} x_{b_2} + \ldots + g_{b_q} + r_{b_q} x_{b_q} \quad (1)$$

donde $x_i > 0$, $\forall i : 1 \leq i \leq n$. Transformando la ecuación anterior:

$$r_{a_1}x_{a_1} + r_{a_2}x_{a_2} + \ldots + r_{a_p}x_{a_p} - r_{b_1}x_{b_1} - r_{b_2}x_{b_2} - \ldots - r_{b_q}x_{b_q} = g_{b_1} + g_{b_2} + \ldots g_{b_q} - g_{a_1} - g_{a_2} - \ldots - g_{a_p} \tag{2}$$

$$r_{a_1}x_{a_1} + r_{a_2}x_{a_2} + \ldots + r_{a_p}x_{a_p} - r_{b_1}x_{b_1} - r_{b_2}x_{b_2} - \ldots - r_{b_q}x_{b_q} = k \tag{3}$$

entonces una condición necesaria para que la ecuación tenga solución es que:

$$\text{mcd}(r_1, r_2, \ldots, r_n) \mid k \tag{4}$$

Sea la ecuación de $ax + by = c$, con $a > 0 \wedge b > 0$ por el teorema de **Bezout** se tiene que:

$$x = x_0 - k\frac{b}{d} \wedge y = y_0 + k\frac{a}{d} \tag{5}$$

donde x_0 y y_0 son soluciones particulares de la ecuación y $d = \text{mcd}(a, b)$. Observe que siempre se puede garantizar que exista una solución $x = x'$ y $y = y'$, con $x' > 0 \wedge y' < 0$, ya que independientemente del signo de x_0 y y_0 se puede seleccionar un k negativo con módulo lo suficientemente grande para encontrar x' y y'.

Ahora volviendo al ejercicio, por la observación anterior se puede asegurar que existe una solución mayor que 0 para x_{a_1} la ecuación:

$$r_{a_1}x_{a_1} + w_1\text{mcd}(r_{a_2}, r_{a_3}, \ldots, r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) = k \tag{6}$$

y siguiendo misma idea se puede asegurar que existe una solución positiva para cada uno de los x_{a_i} $\forall i : 2 \leq i \leq p-1$ en las siguientes ecuaciones:

$$r_{a_2}x_{a_2} + w_2\text{mcd}(r_{a_3}, \ldots, r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) = w_1\text{mcd}(r_{a_2}, \ldots, r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) \tag{7}$$

$$r_{a_3}x_{a_3} + w_3\text{mcd}(r_{a_4}, \ldots, r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) = w_2\text{mcd}(r_{a_3}, \ldots, r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) \tag{8}$$

$$\vdots$$

$$r_{a_{p-1}}x_{a_{p-1}} + w_{p-1}\text{mcd}(r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) = w_{p-2}\text{mcd}(r_{a_{p-1}}, r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) \tag{9}$$

Ahora se debe probar que la siguiente ecuación tiene soluciones con las restricciones anteriores:

$$r_{a_p}x_{a_p} - r_{b_1}x_{b_1} - r_{b_2}x_{b_2} - \ldots - r_{b_q}x_{b_q} = w_{p-1}\text{mcd}(r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) \tag{10}$$

Usando el enfoque anterior se puede asegurar que existe una solución mayor que 0 para cada uno de los x_{b_i} $\forall i : 2 \leq i \leq q$ en las siguientes ecuaciones:

$$e_q\text{mcd}(a_p, r_{b_1}, \ldots, r_{b_q-1}) - r_{b_q}x_{b_q} = w_{p-1}\text{mcd}(r_{a_p}, r_{b_1}, r_{b_2}, \ldots, r_{b_q}) \tag{11}$$

$$e_{q-1}\text{mcd}(a_p, r_{b_1}, \ldots, r_{b_q-2}) - r_{b_{q-1}}x_{b_{q-1}} = e_q\text{mcd}(a_p, r_{b_1}, \ldots, r_{b_q-1}) \tag{12}$$

$$\vdots$$

$$e_2\text{mcd}(a_p, r_{b_1}) - r_{b_2}x_{b_2} = e_3\text{mcd}(a_p, r_{b_1}, r_{b_2}) \tag{13}$$

Finalmente se puede asegurar una solución mayor que 0 para x_n y b_1:

$$r_{a_p}x_{a_p} - r_{b_1}x_{b_1} = e_2\text{mcd}(a_p, r_{b_1}) \tag{14}$$

Por tanto se puede asegurar (4) es una condición necesaria y suficiente para que (3) tenga solución. Además de esto la demostración anterior proporciona un algoritmo para encontrar dichas soluciones (las soluciones particulares de la ecuación de **Bezout** se pueden encontrar usando el algoritmo **Extendido de Euclides**).

Entonces se ha demostrado que el ejercicio es equivalente a encontrar dos conjuntos P y Q tales que, $P \cup Q = \{g_1, g_2, \ldots, g_n\}$, $P \neq \emptyset$, $Q \neq \emptyset$, $P \cap Q = \emptyset$ y además debe cumplirse que:

$$\text{mcd}(r_1, r_2, \ldots, r_n) \mid \text{sum}(P) - \text{sum}(Q) \tag{15}$$

donde los p_i y q_i son los elementos que de los conjuntos P y Q respectivamente.

3.3.2. Reducción al Problema de Partición

El **Problema de Partición** es un problema **NP-completo** que plantea si dado un conjunto de números es posible particionar dicho conjunto en 2 subconjuntos tal que la suma sea la misma.

En esta sección se demostrará que el problema del ejercicio analizado es **NP-completo**, para ello primero es necesario demostrar que el problema es **NP** y luego basta con demostrar que cualquier instancia del **Problema de Partición** puede ser reducida una instancia del problema del ejercicio.

Primeramente observe que dado una partición de G en P y Q es posible comprobar en tiempo polinomial si $\text{mcd}(r_1, r_2, \ldots, r_n) \mid \text{sum}(P) - \text{sum}(Q)$ lo que demuestra que el problema del ejercicio está en **NP**.

Entonces se tiene un conjunto W, a partir de este conjunto se puede construir una instancia del ejercicio, para ello seleccione los r_i tal que $\text{mcd}(r_1, \ldots, r_n) = \text{sum}(W)$ y los para los g_i tome exactamente los elementos del conjunto W. Ahora observe que la única manera de particionar W en dos subconjunto P y Q tal que $\text{sum}(W) \mid \text{sum}(P) - \text{sum}(Q)$, es que $\text{sum}(P) - \text{sum}(Q) = 0$ porque $\text{sum}(P) + \text{sum}(Q) = \text{sum}(W)$. Con lo cual se ha reducido el **Problema de Partición** al problema del ejercicio analizado, lo cual demuestra que el problema del ejercicio es **NP-completo**.

En las próximas secciones se reducirá el problema de la partición a otros problemas **NP-completos**.

3.3.3. Reducción a la Suma de Subconjunto

La **Suma de Subconjunto** es un problema **NP-completo** que plantea si dado un conjunto de números es posible seleccionar un subconjunto tal que la suma de los elementos de dicho conjunto es igual a k.

En esta sección se reducirá el problema de la **Suma de Subconjunto** al **Problema de Partición**, para demostrar que el **Problema de Partición** es **NP-completo**.

Sea un conjunto W, observe que dado el conjunto P, tal que $P \subseteq W$, es posible comprobar en tiempo polinomial que $\text{sum}(P) = \text{sum}(W)/2$ lo cual demuestra que la **Problema de Partición** es un problema **NP**.

Sea un conjunto W y un entero k, se define un conjunto W' formado por todos los elementos de W más los elementos $w_1 = 2k$ y $w_2 = \text{sum}(W)$. Ahora si se trata de formar una partición de W' con la restricción del **Problema de Partición**, observe w_1 y w_2 no pueden estar en el mismo conjunto ya que su suma sería mayor que $\text{sum}(W) + k$, por tanto el conjunto que contenga a w_2 debe contener además un subconjunto de W cuya suma sea k, que esta precisamente es una instancia del problema de la **Suma de Subconjunto**. Por otro lado si existe W'' tal que $W'' \subseteq W \wedge \text{sum}(W'') = k$, entonces el conjunto formado por los elementos de W'' y w_2 sería una partición de W'' que cumple con las restricciones del **Problema de Partición**.

3.3.4. Reducción al Cubrimiento de Vértices

El **Cubrimiento de Vértices** es un problema **NP-completo** que plantea si dado un grafo $G = V, E$ y un entero k es posible seleccionar un subconjunto de vértices V' de tamaño k de tal manera que todas las arista de G incidan sobre al menos uno de los vértices de V'.

En esta sección se reducirá el problema de la **Suma de Subconjunto** al **Cubrimiento de Vértices**, para demostrar que la **Suma de Subconjunto** es **NP-completo**.

Sea un conjunto W y un entero k, observe que dado el conjunto P, tal que $P \subseteq W$, es posible comprobar en tiempo polinomial que $\text{sum}(P) = k$ lo cual demuestra que la **Suma de Subconjunto** es un problema **NP**.

Sea un grafo $G = V, E$ y un entero k, donde $V = \{1, 2, \ldots, n\}$, se definen los enteros en base 4 $a_i \ \forall i : 1 \leq i \leq n$ y $b_{i,j} \ \forall i, j :< i, j >\in E$. Ahora se define una matriz de $|V| + |E|$ filas y $|E| + 1$ columnas donde cada fila pertenece a un entero a_i o un entero $b_{i,j}$. En la primera columna coloque un 1 en cada fila que pertenece a los a_i y un 0 en cada fila correspondiente a los $b_{i,j}$. Cada una de las restantes columnas corresponde a una arista $\langle i, j\rangle \in E$, ahora para llenar la columna correspondiente a la arista $\langle i, j\rangle$ coloque un 1 en cada fila que pertenece a los a_x, donde $x = i \vee x = j$, un 1 en cada fila correspondiente a los $b_{x,y}$, donde $x = i \vee x = j \vee y = i \vee y = j$.

Se define k' de la siguiente manera:

$$k' = 4^{|E|}k + \sum_{i=0}^{|E|-1} 2 \cdot 4^i \tag{16}$$

Tome un subconjunto de vértices y aristas (V', E') tales que:

$$\sum_{i \in V'} a_i + \sum_{\langle i,j\rangle \in E'} b_{i,j} = k' \tag{17}$$

Primeramente observe que cada $b_{i,j}$ (con $\langle i, j\rangle \in E'$) puede tener como máximo un 1 en cada column, por tanto en cada una de las $|E|$ últimas columnas la suma máxima es 3, por lo que no hay acarreos. Entonces los únicos números que aportan $4^{|E|}$ son los a_i y como $k' > 4^{|E|}k$ se cumple que $|V'| = k$. Por otro lado como k' tiene un 2 en cada columna que pertenecen a una arista $\langle i, j\rangle \in E$, se cumple que para cada arista $\langle i, j\rangle \in E \ i \in V' \vee j \in V'$, lo cual implica que V' es un **Cubrimiento de Vértices** de tamaño k.

Ahora suponga que existe un subconjunto de vértices V', donde V' es un **Cubrimiento de Vértices** de tamaño k, luego seleccione los a_i tales que $i \in V'$ y los $b_{i,j}$ tales que $i \in V'$ o $j \in V'$ pero no ambos a la vez. Al sumar todos estos números observe que se obtiene un subconjunto de números cuya suma es k'.

Se presenta un ejemplo a continuación:

- Se define un grafo $G = V, E$ y un entero $k = 2$, donde $V = \{1, 2, 3, 4\}$ y $E = \{\langle 1, 2\rangle, \langle 2, 3\rangle, \langle 3, 4\rangle, \langle 4, 1\rangle\}$, el conjunto $V' = 1, 3$ es un **Cubrimiento de vértices** de tamaño k. Se define la matriz en la siguiente tabla:

		$\langle 1,2\rangle$	$\langle 2,3\rangle$	$\langle 3,4\rangle$	$\langle 4,1\rangle$
a_1	1	1	0	0	0
a_2	1	1	1	0	0
a_3	1	0	1	1	0
a_4	1	0	0	1	1
$b_{1,2}$	0	1	0	0	0
$b_{2,3}$	0	0	1	0	0
$b_{3,4}$	0	0	0	1	0
$b_{4,1}$	0	0	0	0	1

- $k' = 4^4 \cdot 2 + \sum_{i=0}^{3} 2 \cdot 4^i = 682$
- Si se selecciona los a_i y los $b_{i,j}$ a partir de V' se obtienen los números $a_1 = 321$, $a_3 = 276$, $b_{1,2} = 64$, $b_{2,3} = 16$, $b_{1,4} = 1$, $b_{3,4} = 4$ y se cumple que $a_1 + a_3 + b_{1,2} + b_{2,3} + b_{1,4} + b_{3,4} = 682 = k$

  Por tanto queda reducido el problema del **Cubrimiento de vértices** al problema de la **Suma de Subconjunto**.

3.3.5. Fuerza bruta

Como se demostró anteriormente existe una condición necesaria y suficiente para que el problema tenga solución. Entonces para obtener un algoritmo de fuerza bruta que resuelva el problema, basta con encontrar todos los subconjuntos propios no vacíos de G y comprobar que la partición que define cada subconjunto cumpla con la condición del

problema. Esto se puede lograr mediante un **Backtrack**, pasar por cada índice i llevando una suma global s y tomar 2 decisiones sumar g_i a s o no, y por último al llegar al final comprobar la condición del problema. Este algoritmo tiene una complejidad de $O(2^n)$, la cual como se puede observar es exponencial.

3.3.6. Restricción Polinomial

Ahora se considerará la restricción del problema original, es decir se cumple que $\text{sum}(G) \leq 2 \cdot 10^5$. Entonces es posible memorizar los estados del **Backtrack** enunciado en la sección anterior, es decir llevar además una arreglo de 2 dimensiones $dp[i][w]$ esto significa que se puede lograr un subconjunto con peso w usando solo índices menores que o iguales que i, observe que ahora el algoritmo ha reducido su complejidad ya que la nueva complejidad es igual a la cantidad de estados del **Backtrack** que en este caso sería $O(\text{sum}(G)n)$ y como $\text{sum}(G) \leq 2 \cdot 10^5$ sería $O(2 \cdot 10^5 n)$.

Para mostrar una implementación más eficientemente desde el punto de vista computacional se pueden establecer las transiciones de la **Dinámica** y eliminar la recursividad del **Backtrack**, a continuación se muestra un pseudocódigo:

```
for(i in n) {
    dp[i][0] = true
}

for(i in n) {
    dp[i][g[i]] = true

    for(w in sum(G)) {
        if(i != 0 and dp[i - 1][w]) {
            dp[i][w] = true
        }

        if(w >= g[i] and i != 0 and dp[i - 1][w - g[i]]) {
            dp[i][w] = true
        }
    }
}
```

las transiciones del algoritmo anterior son las siguientes:

- Siempre es posible obtener un conjunto que use índices menores iguales que i con suma $g[i]$ o suma 0.
- Si es posible formar un conjunto con indices menores o iguales a $i-1$ con suma w, también es posible obtener un conjunto cons sum w usando índices menores o iguales que i.
- Si es posible formar un conjunto con indices menores o iguales a $i-1$ con suma $w - g[i]$, también es posible obtener un conjunto cons sum w usando índices menores o iguales que i.

La cantidad total de estados de la **Dinámica** es $\text{sum}(G)n$ y resolver cada estado cuesta $O(1)$, para una complejidad total de $O(\text{sum}(G)n)$. La **Dinámica** empleada en este es exactamente igual a la **Dinámica de la mochila** pero en este caso solo interesa conocer si es posible o no lograr ese peso.

Ahora haga la siguiente observación si la cantidad de elementos $n \leq 2 \cdot 10^5$ y $\text{sum}(G) \leq 2 \cdot 10^5$, entonces en G hay a lo sumo $\sqrt{n}$ elementos distintos, por tanto se puede aplicar el algoritmo de la **Dinámica de la mochila comprimida** que funciona en $O(cw)$, donde w es la suma de todos los elementos del conjunto y c es la cantidad de elementos distintos, en el caso particular del ejercicio la complejidad sería $O(2 \cdot 10^5 \sqrt{n})$ lo cual es más eficiente que el algoritmo visto anteriormente.

Para la implementación de la **Dinámica de la mochila comprimida** primeramente se debe calcular las ocurrencias de cada valor y luego definir $dp[i][w]$, donde $dp[i][w]$ guarda la cantidad mínima de elementos con peso $g'[i]$ para

lograr un subconjunto con suma w que use índice menores o iguales que i (g' son los elementos del conjunto G luego de eliminar los pesos repetidos) o -1 en caso de que no sea posible. Las transiciones para el algoritmo quedarían de la siguiente manera:

- Siempre es posible obtener un conjunto que use índices menores iguales que i con suma 0, usando 0 elementos de peso $g'[i]$.
- Si $dp[i-1][w] \geq 0$ entonces $dp[i][w]$, es decir si se puede obtener un subconjunto de suma w con índices menores o iguales que $i-1$ entonces también se puede encontrar un subconjunto con suma w que use índices menores o iguales que i
- Si $dp[i][w-g'[i]] \geq 0 \wedge dp[i][w-g'[i]] < g'_c[i]$ entonces $dp[i][w] = dp[i][w-g'[i]]+1$, es decir si la mínima cantidad de elementos con peso $g'[i]$ para formar un conjunto de suma w es menor que la cantidad de elementos con peso $g'[i]$ disponibles entonces la mínima la mínima cantidad elementos con peso $g'[i]$ necesarios es exactamente $dp[i][w-g'[i]]+1$. Observe que la primera transición se debe comprobar antes de esta para garantizar mínimo y esto siempre garantiza que sea mínimo ya que se verifica si es posible garantizar un peso w sin usar $g'[i]$ y después usando $g'[i]$, pero apoyado en el valor de la subestructura óptima de la **Dinámica**.

a continuación se muestra un pseudocódigo:

```
c <-- cantidad de elementos distintos de G
g_count <-- array de tuplas con los pesos de los elementos de G y
            las ocurrencias de cada elemento

for(i in c) {
    dp[i][0] = 0
}

for(i in c) {
    for(w in sum(G)) {
        if (i != 0 and dp[i - 1][j] >= 0)
        {
            dp[i][j] = 0;
            continue;
        }

        if (j >= g_count[i].f and dp[i][j - g_count[i].f] >= 0
            and dp[i][j - g_count[i].f] < g_count[i].s)
        {
            dp[i][j] = dp[i][j - g_count[i].f] + 1;
            continue;
        }
    }
}
```

La cantidad total de estados de la **Dinámica** es $\text{sum}(G)\sqrt{n}$ y resolver cada estado cuesta $O(1)$, para una complejidad total de $O(\text{sum}(G)\sqrt{n})$.

3.4. Implementación

Puede encontrar la implementación del ejercicio en el siguiente enlace enlace.