Gran Premio de México 2021 - Primera Fecha

August 28th, 2021

Libro de soluciones

Este documento contiene las soluciones esperadas para los 13 problemas usados durante la competencia.

Las siguientes personas apoyaron desarrollando el set de problemas ya sea creando o mejorando enunciados, soluciones, casos de prueba, verificadores de entradas y salidas:

Lina Rosales

Saraí Ramírez

Eddy Ramírez

Leonel Juarez

Alan Enrique Ontiveros

Abraham Macias

Emilio López

Fernando Fiori

Moroni Silverio

Juan Pablo Marín

Alien Crop Triangles

Por: Emilio López

Para solucionar el problema, primero es necesario calcular el área total a cubrir, esto se logra sumando las áreas de todos los triángulos de la entrada. Puede hacerse de forma sencilla utilizando la fórmula de Herón:

$$\text{area} = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$

donde s es el semiperímetro:

$$s = \frac{a+b+c}{2}$$

Ya conociendo el área en m^2 a cubrir, pasamos a calcular los kg de semillas necesarios, redondeando para arriba. El enunciado indica que se necesita 1 kg cada 30 m^2, por lo que

$$\text{kg necesarios} = \text{ceil}\left(\frac{\text{area}}{30}\right)$$

Luego usamos una mochila (*knapsack*) para minimizar el costo de comprar los kg necesarios. Debemos elegir la opción más barata que tiene al menos los kg necesarios – puede ser que resulte más barato comprar kilos de más; no hace falta que sea exactamente la misma cantidad de kg calculada. Este costo será la solución que imprimiremos.

Para finalizar, debemos tener especial cuidado de tratar los casos bordes de la entrada correctamente, entre los que se destacan los siguientes dos:

- Si no hay ningún triángulo en la entrada, no es necesario comprar nada (salida 0).
- Si tenemos que comprar algo y las únicas bolsas disponibles están vacías (peso 0), no se podrá resolver el problema (salida -1).

Basel Problem

Por: Alan Enrique Ontiveros

Sea $f(z) = \dfrac{\pi \cot(\pi z)}{z^n}$ una función, donde $n \geq 2$. Esta función será útil para calcular $\zeta(n)$ usando el teorema del residuo, porque $f(z)$ tiene polos en cada entero.

Integrando $f(z)$ sobre un cuadrado muy grande centrado en el origen del plano complejo, con vertices en los puntos: $\{(N+1/2)(1+i), (N+1/2)(-1+i), (N+1/2)(-1-i), (N+1/2)(1-i)\}$, donde $N \to \infty$.

Se puede mostrar que: $\oint_\Gamma f(z)dz = 0$. Por el teorema del residuo esa integral es:

$$\oint_\Gamma f(z)dz = 2\pi i \sum_{k=-\infty}^{\infty} Res_{z=k} f(z)$$

Comparando ambas expresiones, tenemos que:

$$\sum_{k=-\infty}^{\infty} Res_{z=k} f(z) = 0$$

Vemos que $f(z)$ tiene polos de orden 1 en cada entero diferente de 0, esto es, si $k \neq 0$ es un entero, entonces:

$$\begin{aligned}
Res_{z=k} f(z) &= \lim_{z\to k} f(z)(z-k) \\
&= \lim_{z\to k} \frac{\pi \cot(\pi z)(z-k)}{z^n} \\
&= \lim_{z\to k} \frac{\pi \cos(\pi z)(z-k)}{\sin(\pi z) z^n} \\
&= \lim_{z\to k} \frac{\pi \cos(\pi z)}{z^n} \cdot \lim_{z\to k} \frac{z-k}{\sin(\pi z)} \\
&= \frac{\pi \cos(\pi k)}{k^n} \cdot \lim_{z\to k} \frac{1}{\pi \cos(\pi z)} \\
&= \frac{\pi \cos(\pi k)}{k^n} \cdot \frac{1}{\pi \cos(\pi k)} \\
&= \frac{1}{k^n}
\end{aligned}$$

Como n es un número par entero positivo, tenemos que:

$$2\sum_{k=1}^{\infty} \frac{1}{k^n} + Res_{z=0} f(z) = 0$$

$$\zeta(n) = -\frac{1}{2} Res_{z=0} f(z)$$

Entonces, encontrar $Res_{z=0} f(z)$ es lo mismo que encontrar el coeficiente de z^{-1} en la serie de Laurent para $f(z)$, o de manera equivalente, encontrar el término constante en $\dfrac{\pi z \cot(\pi z)}{z^n}$.

Finalmente para encontrar todos los valores de $\zeta(n)$ al mismo tiempo, podemos observar que $\zeta(n)$ es el coeficiente de z^n en la serie de Taylor: $-\dfrac{1}{2}\pi z \cot(\pi z)$.

Sabemos que $\zeta(n) = \dfrac{p_n}{q_n}\pi^n$, por lo que:

$$\sum_{\substack{n=0 \\ n \text{ even}}}^{\infty} \zeta(n) z^n = \sum_{\substack{n=0 \\ n \text{ even}}}^{\infty} \frac{p_n}{q_n} (\pi z)^n = -\frac{1}{2}\pi z \cot(\pi z)$$

Sustituyendo $x = \pi z$, tenemos que:

$$g(x) = \sum_{\substack{n=0 \\ n \text{ even}}}^{\infty} \frac{p_n}{q_n} x^n = -\frac{1}{2} x \cot(x)$$

Observemos que $g(x) = -\frac{1}{2}x\cot(x) = -\frac{1}{2}\cdot\frac{\cos(x)}{\sin(x)/x}$ es el cociente de dividir dos series de Taylor, y es la función que genera todas las respuestas que buscamos.

Podemos extraer los primeros n coeficientes de manera eficiente usando FFT (NTT con el módulo $119\times 2^{23}+1$) tomando la convolución de $\cos(x) = \frac{1}{0!} - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \cdots$ con el inverso multiplicativo de $\sin(x)/x = \frac{1}{1!} - \frac{1}{3!}x^2 + \frac{1}{5!}x^4 - \frac{1}{7!}x^6 + \cdots$ (el inverso se puede calcular usando el método de Newton-Raphson) con una complejidad de $O(n\log n)$ si ignoramos los coeficientes después de x^n de ambas series.

Cypher Decypher

Por: Eddy Ramírez

Este es un problema clásico. Dado que el valor de los números nunca será mayor a 10^6, se puede usar una criba para calcular los números primos. Una vez que se tenga la criba C podemos saber si un número es primo si $C[x] = 1$ y no es primo si $C[x] = 0$. Ahora, si para cada valor i, j dado en la entrada, contamos cuantos valores $C[x] = 1$ tales que $i \leq x \leq j$ estaremos encontrando la respuesta correcta, pero, la iteración agregada hará que el envío sea rechazado por exceder el tiempo límite de ejecución.

Podemos crear un nuevo arreglo S tal que $S[x] = S[x-1] + C[x]$, de esta manera podemos observar que $S[x]$ tiene el valor acumulado de cuántos números primos hay entre 1 e i (la suma de cuántos $C[i] = 1$ hasta i). Usando este arreglo, podemos encontrar la cantidad de primos que hay entre i y j simplementa calculando $S[j] - S[i-1]$.

Delivering Pizza

Por: Moroni Silverio

La mejor manera de solucionarlo es un ingrediente a la vez ignorando los demás empezando por el primer ingrediente, al leer la entrada sabemos cuáles serán los únicos que pasarán por ahí y deben atenderse según el orden de la entrada.

Para los demás ingredientes, de igual manera enfocarse en el j-ésimo ingrediente si los ingredientes anteriores han sido resueltos, ya que no se agregarán más pedidos a ese ingrediente después. Lo que se debe tener presente es el momento en que la persona atendiendo el ingrediente J se desocupa, una vez que lo hace tiene a elegir a todos los pedidos que llegaron antes de que se desocupara. Si no llegaron pedidos antes de eso, debe escoger el primer pedido que llegará después de que se desocupe. Esto se hace con dos colas de prioridad, una para llevar el control de los pedidos que llegaron antes de que se desocupara y otra para llevar el control de los pedidos que llegan después de que se desocupe.

Escape Room

Por: Moroni Silverio

Podemos observar que se puede dar la respuesta a cualquiera de las preguntas tales que la coordenada no sea un '.' en el mapa con solo observar el mapa. Entonces, nos interesa encontrar cómo responder las preguntas que se hagan a estas coordenadas: El problema de encontrar el camino más corto entre dos puntos en un mapa como el entregado en la entrada es bien conocido que puede ser resuelto aplicando una búsqueda en amplitud, entonces, una posible solución al problema sería para cada consulta realizar la búsqueda en amplitur partiendo de esa coordenada para encontrar el camino más corto hasta la salida, en caso de que exista algún camino. Esta solución si bien es correcta, es muy lenta con la cantidad de consultas que se podrían dar como entrada, y sería rechazada por exceder el tiempo límite de ejecución.

Una característica de la búsqueda en amplitud es que no solo encuentra el camino más corto partiendo de un nodo origen a un destino, sino que encuentra el camino más corto del nodo origen a cualquier otro nodo que es alcanzable en el mapa partiendo del origen, podemos observar que el camino más corto de una posición del mapa a la salida, es el mismo que el de la salida hacia esa posición, entonces, podemos calcular el camino más corto de cualquier posición del mapa a la salida realizando una búsqueda en amplitud que tenga como origen la salida del mapa. Es importante tener en cuenta las prioridades con las que se debe responder en caso de que un punto tenga más de un camino con la misma distancia a la salida. Una vez realizado este precómputo, se puede calcular cada respuesta en $O(1)$.

Fixing Subtitles

Por: Moroni Silverio

Para solucionar el problema se tiene que "parsear" la entrada para separar los subtítulos del tiempo en el que comienzan. Teniendo esa información y el tiempo correctamente "parseado", solo hay que sumar para obtener la nueva hora. Todas las líneas que no tienen una hora deben ser impresas como se reciben.

Se deben considerar casos donde el delay sea 0.

Game of Baker

Por: Lina Rosales

HeatWave

Por: Fernando Fiori

Primero describo una solución para un tamaño de w fijo y luego se puede extender a cualquier tamaño < 200.

Observación 1) quizás no se pueden reemplazar todas las ocurrencias de una palabra porque luego puede no ser reversible la operación, se pueden reemplazar ocurrencias que no se superpongan. Ejemplo: T=000, w=00, la operación correcta de reemplazos devuelve T'=20 o T'=02, pero nunca T'=22.

Observación 2) Si conozco todas las ocurrencias de una palabra w en T, una forma óptima de elegir los reemplazos es elegir la primera ocurrencia de w (recorriendo T de izquierda a derecha), saltear $|w|$ caracteres, y repetir el proceso. No hay una manera que comprima más el texto que ésta. (Demostración, sea i el índice de la primera ocurrencia de w en T, y j el índice de la siguiente ocurrencia, $i < j$. Supongamos que no elijo i como reemplazo, pero sí j. Luego puedo seguir eligiendo ocurrencias de w en T que empiecen en $j + |w|$, o sea en $S = T_{j+|w|}...T_{|T|-1}$. Pero si hubiésemos elegido la ocurrencia de i, entonces podríamos seguir eligiendo ocurrencias en $R = T_{i+|w|}...T_{|T|-1}$. Pero S está contenido en R, y nunca vamos a encontrar más ocurrencias en un pedazo de texto que en el texto completo, ya que puedo omitir esos primeros j-i caracteres. Con lo que elegir j en vez de i no nos da una mejor solución.)

Ahora bien, si buscamos una $|w|$ de un tamaño dado que nos maximice la compresión de T, podemos ir recorriendo T hasheando la ventana de tamaño w. Podemos usar hash polinomial ya que el tamaño máximo de la cadena es mayor que 64 bits. Vamos manteniendo un hashmap con hash de cadena como key, y como value un par cantidad de ocurrencias occ, último índice donde se la vio $last$. De esta manera si $last < indice\_actual - |w|$, entonces hago $occ++$ y $last = indice\_actual$.

Una vez que termino de recorrer T reviso cada entrada del map y me quedo con la palabra de mayor cantidad de ocurrencias.

Realizo este proceso para cada tamaño posible de w (de 2 a b) y me quedo con la mejor.

Luego es solamente hacer una cuenta y mostrar el resultado.

La complejidad final es $O(|T| * b)$, sin contar el costo de acceder a valores del hashmap.

Introducing Teleporting Machine

Por: Moroni Silverio

Para obtener el timepo sin la maquina teletransportadora basta con sumar $ciudad[n] - ciudad[i]$ para toda i.

Ahora, para calcular el tiempo total con la maquina conectando las ciudades i y j necesitamos:

- El tiempo total sin maquina T.
- Lo que ahorramos en tiempo D: el numero de ciudades que usaran la maquina por la distancia que se ahorrara que es la distancia entre i y j
- El costo de usar la máquina C: El numero ciudades que hay hasta i por el costo de usar la maquina

Entonces, cuando la máquina se pone entre las ciudades i y j, el costo será $T_{i,j} = T - D + C$, se debe encontrar el i, j que minimiza el valor de $T_{i,j}$. Esto se puede lograr iterando sobre cada ciudad i donde se puede poner la máquina y establecer j como la ciudad más lejana que está a una distancia menor o igual a los K kilómetros máximos que puede conectar la máquina de transportación, encontrar la ciudad j para cada valor posible de i se puede lograr con una busqueda binaria, y entonces el algoritmo para encontrar los valores i, j que minimizan $T_{i,j}$ es $O(NlogN)$.

Just Send the Email

Por: Abraham Macías

Sean los empleados nodos y las relaciones manager-subordinado aristas, la estructura de la organización de la compañía forma un árbol enraizado en 1. En este problema se pide hallar el valor esperado de la distancia entre un nodo aleatorio u del árbol a la hoja más cercana. Notar que de un nodo u se puede mover a cualquiera de sus hijos o al padre, así que se podría considerar un árbol no dirigido.

Para encontrar la respuesta, se puede hallar la suma de las distancias más cortas a una hoja desde cada nodo. Para hallar la distancia más corta de un nodo a la hoja más cercana, se puede hacer una BFS (búsqueda en amplitud) multi-origen desde todas las hojas hacia los demás nodos (hay que meter al inicio todas las hojas a la cola y después ejecutar una BFS). La distancia inicial de cada hoja es 1.

Después de hallar la suma de las distancias, hay que multiplicarlo por el inverso multiplicativo modular de n. Para hallar el inverso modular se puede usar el pequeño teorema de Fermat o el algoritmo extendido de Euclides.

Kids at the Party

Por: Saraí Ramírez

Se puede observar que para que una cantidad M de niños asistan a la fiesta, es necesrio que $N \ mod \ M = 0$. Es decir que M sea un divisor de N. Dado que Jaime quiere que su amigo Churro siempre asista y que no puede haber más de 6 niños en la fiesta, entonces el valor de M cumple que $2 \leq M \leq 6$. Entonces, para encontrar las diferentes cantidades de niños que pueden asistir a la fiesta será necesario encontrar todos los números M con las restricciones dadas que sean divisores de N. Un inconveniente es que el valor N puede ser tan grande que no se puede almacenar en un entero, entonces, se debe encontrar una estrategia que nos permita identificar los posibles valores de M a pesar del tamaño del número. Una manera de realizar esto es aplicar criterios de divisibilidad:

Se pueden utilizar criterios de divisibilidad de los números 2,3,4,5,6 para identificar los posibles valores de M dado N:

- 2 es un divisor de N si el último dígito de N es par.
- 3 es un divisor de N si la suma de los dígitos de N es un múltiplo de 3
- 4 es un divisor de N si los últimos dos dígitos de N son un múltiplo de 4
- 5 es un divisor de N si el último dígito de N es 5 o 0.
- 6 es un divisor de N si 2 y 3 son divisores de N.

Leonel and the powers of two

Por: Leonel Juárez

La notación que se le pide a Leonel, no es en realidad inventada por su profesor. Es como tal la función recursiva que se utiliza para hacer realizar el cálculo de un exponente con el algoritmo conocido como exponenciación binaria: `https://es.wikipedia.org/wiki/Exponenciaci%C3%B3n_binaria`.

El problema entonces se reduce a implementar la función recursiva que define a la notación que se le pidió a Leonel, teniendo cuidado de no dejar espacios en blanco y de que se sigue la impresión según las 3 reglas definidas.

Moon Dancers

Por: Juan Pablo Marín

Una característica importante a notar en el problema, es que, debido a que todos los danzantes se sentarán siempre en una posición tal que el ángulo es entero, entonces hay exactamente 359 posibles posiciones para un danzante que se levanta para rotar hacia otro danzante, además como todos los danzantes que se levantan rotarán el mismo ángulo, podemos pensar en una estrategia para resolver el problema en el que encontremos el máximo número de parejas que se pueden hacer si los danzantes rotaran una cantidad fija R, y encontrar el máximo entre los 359 valores que puede tomar R.

Dado R, podemos observar que el danzante i podrá ser pareja con el danzante j, si $(i + R)\ mod\ 360 = j$. Definamos un grafo dirigido G, donde los vertices son cada una de las posiciones donde hay un danzante, y existe una arista que sale de la posición i inicial de un danzante a la posición j si y solo si hay un danzante en la posición j y el danzante de la posición i puede llegar al danzante j al hacer la rotación R. Dado G, podemos visualizar el problema como colorear la máxima cantidad de nodos que nos sea posible usando dos colores A y B , de modo que si dos nodos u, y v están coloreados y hay una arista que los une, entonces u está coloreado del color A y v es de color B. Para realizar esta tarea observemos que por las características que tienen las rotaciones, cada vertice tendrá a lo más una arista de salida y una arista de entrada para cada una de las rotaciones posibles, entonces, podremos verificar que no habrá cadenas ni ciclos que se intersecten en G, así, si tomamos una componente conexa de G podríamos "estirarla"de modo que se forme una línea, debido a esta característica, podemos ver que si la componente conexa tiene C_v nodos de G, entonces, se pueden hacer $\lfloor \frac{C_v}{2} \rfloor$ parejas de esa componente. Entonces, para contar cuántas parejas se pueden hacer en una rotación, basta con sumar la cantidad de parejas que se pueden realizar por cada componente del grafo. La respuesta al problema será el valor más grande que se obtenga después de obtener la respuesta para cada una de las 359 rotaciónes posibles.

Gran Premio de México 2021 - Segunda Fecha

September 25th, 2021

Libro de soluciones

Este documento contiene las soluciones esperadas para los 11 problemas usados durante la competencia.

Las siguientes personas apoyaron desarrollando el set de problemas ya sea creando o mejorando enunciados, soluciones, casos de prueba, verificadores de entradas y salidas:

Eddy Ramírez

Abraham Macias

Alan Enrique Ontiveros

Roberto Solís

Moroni Silverio

Juan Pablo Marín

Alice Birthday

Por: Abraham Macías

Este problema se puede resolver con DP con máscaras de bits. Sea $dp(mask,\ i)$ el número de formas de eliminar aristas en el subgrafo inducido por los nodos en $mask$, de tal forma que hay i componentes conexas en dicho subgrafo inducido después de eliminar dichas aristas. Un nodo u está en $mask$ si el u-ésimo bit en $mask$ está prendido.

Calculemos la DP en orden creciente de $mask$. El valor de $dp(mask,\ i)$ para $i \geq 2$ se puede hallar de la siguiente forma:

$$dp(mask,\ i) = \sum_{sub} dp(sub,\ 1) * dp(mask\ XOR\ sub,\ i-1)$$

tal que sub es un subconjunto de mask y sub tiene al nodo $lsb(mask)$, donde $lsb(mask)$ es el menor nodo en $mask$ (el bit menos significativo prendido en $mask$).

La razón de porque esto funciona es que, cuando sumamos $dp(sub,\ 1) * dp(mask\ XOR\ sub,\ i-1)$, estamos sumando la cantidad de formas que hay de eliminar aristas tal que la componente conexa donde esta $lsb(mask)$ sea sub, y el resto de nodos en $mask$ ($mask\ XOR\ sub$) estén repartidas en $(i-1)$ componentes conexas. Así que para hallar $dp(mask,\ i)$, consideramos todas las posibles compenentes conexas donde puede estar $lsb(mask)$.

El valor de $dp(mask,\ 1)$ se puede hallar de la siguiente forma:

$$dp(mask,\ 1) = 2^{E(mask)} - \sum_{i=2}^{n} dp(mask,\ i)$$

donde $E(mask)$ es la cantidad de aristas en el subgrafo inducido por $mask$. La razón de esto es que hay $2^{E(mask)}$ formas de eliminar las aristas en este subgrafo, así que la suma de todas las $dp(mask, i)$ para $i \geq 1$ debe ser igual a $2^{E(mask)}$. Simplemente despejamos $dp(mask,\ 1)$ para hallarlo.

La respuesta para una determinada K está en $dp(2^n - 1,\ k)$. Las submáscaras de cada máscara se pueden hallar de la siguiente forma: https://cp-algorithms.com/algebra/all-submasks.html.

La complejidad es $O(n * (3^n))$;

Benford's Law

Por: Abraham Macías

Primero notemos que la respuesta se puede hallar utilizando la propiedad de linealidad sobre el valor esperado. Para cada cubeta i y cada dígito d, podemos calcular de forma separada cuánto aporta la cubeta i al valor esperado de c_d. Se puede ver que el valor esperado de c_d se puede representar de la siguiente forma:

$$E(c_d) = \sum_{i=1}^{n} P_{i,d}$$

Donde $P_{i,d}$ es la probabilidad de que la cantidad de pelotas en la cubeta i tenga al dígito d como primer dígito. Veamos una forma de calcular los valores de $E(c_d)$ en $O(NM)$:

Para cada cubeta i de 1 a N, iteremos para cada j de 0 a M, y veamos cuál es la probabilidad de que durante los M pasos restantes, Alan inserte j pelotas en la cubeta i. Hay $\binom{M}{j}$ formas de elegir los j pasos en los que Alan inserte las pelotas en la cubeta i, y para los $M - j$ pasos restantes hay $(N-1)^{M-j}$ formas de elegir dónde insertar las pelotas restantes. En total existen N^M formas de insertar las pelotas en los M pasos, así que con todo lo anterior, si definimos a $H(j)$ como la probabilidad de insertar j pelotas en la cubeta i, tenemos que:

$$H(j) = \frac{\binom{M}{j} * (N-1)^{M-j}}{N^M}$$

Sólo basta obtener cuál es el primer dígito de $a_i + j$ para saber cuál es el $E(c_d)$ al que aporta $H(j)$.

Para mejorar la complejidad, hay que notar que en realidad no hace falta iterar por todas las j, ya que, por ejemplo, cuando llegamos a $a_i + j = 100$, sabemos que todos los números entre 100 y 199 tienen al 1 como primer dígito, así que podemos sumarle a $E(c_1)$ todos los $H(j)$ tal que $100 \leq a_i + j \leq 199$ en tiempo constante si precaulculamos las sumas de prefijos de $H(j)$, y así con todos los rangos de números que tienen al mismo primer dígito.

Así que, si precalculamos todas las sumas de prefijos de $H(j)$ y hacemos dichos saltos para los números que tienen al mismo primer dígito, podemos obtener una complejidad de $O(max(a_i) + M)$ para hacer el precalculo y $O(9 * log_{10}(K) * N)$ para obtener los valores esperados, ya que existen $O(9 * log_{10}(K))$ rangos de números que tinen al mismo primer dígito.

Cut the Deck!

Por: Juan Pablo Marín

Si bien se puede probar el juego después de hacer un corte para cada carta i, esto sería $O(N^2)$ y con valores grandes de N se excedería el tiempo de ejecución.

Observemos que, si iniciamos un contador en 0, y sumamos 1 cada que se tiene una carta azul, y le restamos 1 cada que se obtiene una carta roja, Bob perderá el juego cuando este contador se vuelva negativo (se han visto más cartas rojas que azules). Podemos ver que, como existen la misma cantidad de cartas rojas y azules, el contador siempre iniciará y terminará en 0 después de voltear todas las cartas. Entonces, si Bob hace el corte en el punto donde este contador tiene el menor valor posible (convirtiendo ese en 0), podremos ver que en ese juego el contador nunca será menor de 0 y por lo tanto Bob gana.

Entonces solo hay que encontrar el indice i más pequeño tal que el contador en ese punto sea el igual al valor mínimo que obtiene el contador en la configuración inicial.

Dislike the Raisins

Por: Moroni Silverio

Llamemos T a la cantidad de unidades que tiene la caja de cereal, entonces $T = C + R$. Sea CS el total de cucharadas que Jaime puede obtener de la caja, esto es $CS = \lceil T/S \rceil$

Para que Jaime obtenga la máxima cantidad de cucharadas sobresalientes, el necesitará que las cucharadas que no son sobresalientes sea mínima, eso es cuando estas cucharadas tengan la máxima cantidad de pasas, sea CR la cantidad de cucharadas que se pueden hacer llenando la cucharada con la máxima cantidad de pasas que sea posible en cada una de ellas, esto es: $CR = \lceil R/S \rceil$ Y entonces la máxima cantidad de cucharadas sobresalientes que Jaime podrá obtener son $CS - CR$.

Para obtener la menor cantidad de cucharadas sobresalientes, podemos distribuir las pasas al menos una por cucharada, entonces, la mínima cantidad de cucharadas sobresalientes que Jaime puede obtener será $max(CS - R, 0)$

E-13 Storage Unit

Por: Roberto Solís

Dados los tamaños de cada video, precalcular las sumas de prefijos de estos tamaños para poder eficientemente calcular cual es la suma de videos de una posición i a una posición $i + R - 1$. Teniendo las sumas de prefijos se podrían probar diferentes valores de R hasta encontrar uno que haga que la suma exceda de la capacidad, una manera mas eficiente es hacer una búsqueda binaria sobre el valor de R. La solución sería por tanto de O(N log N).

Flipped Factorization

Por: Alan Enrique Ontiveros

0.1 Conceptos previos

- Sea $f : \mathbb{N} \to \mathbb{R}$ una función que recibe un entero positivo y devuelve cualquier cosa. A $f(n)$ la llamamos *función aritmética*.
- Si $f(n)$ es artimética y además cumple que $f(mn) = f(m)f(n)$ para enteros positivos coprimos m y n, es decir, con $\gcd(m, n) = 1$, decimos además que $f(n)$ es multiplicativa. Se ve que solo hay dos posibles casos:
  - $f(n) = 0$ para toda $n \in \mathbb{N}$. Este caso no es interesante.
  - $f(1) = 1$, asumiremos que toda función multiplicativa $f(n)$ cumple con esto.
- Una función multiplicativa $f(n)$ queda totalmente definida si sabemos a qué es igual en potencias de primos, es decir, $f(p^a)$ para p primo y $a \in \mathbb{N}$. Esto es cierto, pues podemos factorizar n en primos de la forma $n = {p_1}^{a_1}{p_2}^{a_2}\cdots$, y como dos primos distintos siempre son coprimos, entonces por definición $f(n) = f({p_1}^{a_1}{p_2}^{a_2}\cdots) = f({p_1}^{a_1})f({p_2}^{a_2})\cdots$.
- Sean $f(n)$ y $g(n)$ dos funciones aritméticas. Definimos a la función $h(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right)$ como la *convolución de Dirichlet* de $f(n)$ y $g(n)$ (la suma itera por todos los divisores positivos d de n) y escribimos $h(n) = (f * g)(n)$.
- Teorema: si $g(n)$ y $h(n)$ son funciones multiplicativas, entonces su convolución de Dirichlet también es multiplicativa.
- Sea $f(n)$ una función aritmética, definimos a $F(n) = \sum_{i=1}^{n} f(i)$ como la *función de sumas parciales* de $f(n)$. Usualmente, siempre usaremos minúsculas para la función aritmética y mayúsculas para su función de sumas parciales.

0.2 Subproblema 1

Primero resolvamos este subproblema: sean $h(n)$ y $u(n)$ dos funciones aritméticas, y sea $f(n) = (h * u)(n)$ su convolución de Dirichlet. ¿Cómo hallamos $F(n)$ de forma eficiente? Por definición tenemos que:

$$F(n) = \sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} \sum_{d|i} h(d)u\left(\frac{i}{d}\right)$$

Reacomodemos la suma de la siguiente manera: vemos que cada entero i aporta todos sus divisores d a la suma con el término $h(d)u\left(\frac{i}{d}\right)$. Ahora iteremos primero por los valores de d, entonces i es múltiplo de d y podemos reindexar $i = dk$ para $k \in \mathbb{N}$. Vemos que d no puede exceder n y es al menos 1, entonces:

$$\begin{aligned} F(n) &= \sum_{d=1}^{n} \sum_{k=1}^{dk \leq n} h(d)u\left(\frac{dk}{d}\right) \\ &= \sum_{d=1}^{n} h(d) \sum_{k=1}^{\lfloor n/d \rfloor} u(k) \end{aligned}$$

La suma interior no es más que $U(\lfloor n/d \rfloor)$, donde $U(n)$ es la función de sumas parciales de $u(n)$. Por lo tanto:

$$F(n) = \sum_{d=1}^{n} h(d)U(\lfloor n/d \rfloor)$$

La ventaja de reacomodar la suma así es que muchas veces es fácil calcular $U(n)$, incluso con complejidad constante.

0.3 Subproblema 2

Del subproblema anterior supongamos que $h(n)$ también es multiplicativa, con la restricción adicional de que $h(p) = 0$ para cualquier primo p. ¿Esa restricción cómo ayuda a reducir la complejidad para hallar $F(n)$? Sea $d = {p_1}^{a_1}{p_2}^{a_2}\cdots$ dado en su factorización en primos, entonces $h(d) = h({p_1}^{a_1})h({p_2}^{a_2})\cdots$, y si alguna a_i es igual a 1, tendremos que $h(d) = 0$. Eso quiere decir que para que $h(d)$ sea distinto de cero, todos los exponentes de cada primo de d deben ser al menos 2, a estos números d los llamaremos *powerful*.

¿Cuántos enteros positivos powerful hay menores o iguales a n? Hay $O(\sqrt{n})$, y la demostración se queda como ejercicio. Por lo tanto, solo hay $O(\sqrt{n})$ términos en la expansión de $F(n)$, y si tanto $h(n)$ como $U(n)$ pueden hallarse con complejidad constante, tenemos que $F(n)$ puede hallarse con una complejidad de $O(\sqrt{n})$. Para generar todos los números powerful menores o iguales a n, podemos usar una DFS o BFS precalculando todos los primos hasta $\lfloor\sqrt{n}\rfloor$.

0.4 Problema principal

Notemos que la transformación que le hacemos a un entero positivo m descrita en el problema puede describirse con una función multiplicativa que cumple $f(p^a) = a^p$, y lo que queremos hallar es $F(n)$. Si pudiéramos hallar de forma adecuada $h(n)$ y $u(n)$ tales que:

- $f(n) = (h * u)(n)$
- $h(n)$ y $u(n)$ son multiplicativas
- $h(p) = 0$
- $U(n)$ se puede hallar fácilmente

entonces habremos resuelto el problema con una complejidad de $O(\sqrt{n})$.

De forma arbitraria (más bien al tanteo) podemos escoger $u(n) = 1$ e intentar "despejar" la función $h(n)$:

$$\begin{aligned}
f(p) &= u(p) + h(p)\\
f(p^2) &= u(p^2) + u(p)h(p) + h(p^2)\\
f(p^3) &= u(p^3) + u(p^2)h(p) + u(p)h(p^2) + h(p^3)\\
f(p^4) &= u(p^4) + u(p^3)h(p) + u(p^2)h(p^2) + u(p)h(p^3) + h(p^4)\\
&\vdots
\end{aligned}$$

$$\begin{aligned}
&\implies h(p) = f(p) - 1 = 1 - 1 = 0\\
&\implies h(p^2) = f(p^2) - 1 - h(p) = 2^p - 1\\
&\implies h(p^3) = f(p^3) - 1 - h(p) - h(p^2) = 3^p - 1 - (2^p - 1) = 3^p - 2^p\\
&\implies h(p^4) = f(p^4) - 1 - h(p) - h(p^2) - h(p^3) = 4^p - 1 - (2^p - 1) - (3^p - 2^p) = 4^p - 3^p\\
&\vdots
\end{aligned}$$

Y por inducción, podemos demostrar que $h(p^a) = a^p - (a-1)^p$ para $a \geq 2$.

Finalmente, tenemos que $U(n) = n$ y que $h(p^a)$ prácticamente tiene complejidad constante, entonces mientras calculamos los números powerful con la DFS/BFS, también hay que calcular simultáneamente el valor de $h(n)$.

Grid of Letters

Por: Juan Pablo Marín

Podría ser tentador intentar simular todos los caminos (como se hace en un word search tradicional), pero con las características del problema hay muchísimos caminos lo que haría esta solución exceder el tiempo límite de ejecución.

Supongamos que la posición (i, j) tiene la letra c, el camino más grande que termina en esta posición será 1 + el máximo de los vecinos que tenga una letra $c - 1$. Entonces, podemos encontrar primero el camino más grande que se puede hacer en todas las posiciones que tienen la letra 'A', después con la 'B', etc... y guardar el valor más grande que se encuentre, si se almacenan previamente las posiciones en las que aparece cada una de las letras en la matriz, la complejidad sería O(N*M).

Haunted House

Por: Abraham Macías

El problema se puede resolver simulando los periodos activos de cada fantasma. Primero notemos que sólo nos interesan los primeros $MX = 10^5 + N$ segundos, ya que en el peor caso una persona que entra en el segundo 10^5 a la casa va a llegar a la últma habitación en el segundo $10^5 + N - 1$. Simulemos los primeros MX segundos.

Si suponemos que los periodos de actividad y descanso de un fantasma no son alterados por espantar a una persona, entonces un fantasma con energía x va a tener $O(\frac{MX}{2x})$ periodos activos. Como las energías de los fantasmas forman una permutación, la cantidad total de rangos activos será:

$$\frac{MX}{2*1} + \frac{MX}{2*2} + \frac{MX}{2*3} + \cdots + \frac{MX}{2N} = (\frac{MX}{2})(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{N})$$

La suma $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{N}$ es conocida como la suma harmónica, y tiende a $O(log(N))$, por lo que tendremos $O(MX * log(N))$ rangos activos en total.

¿Qué pasa con el hecho de que el fantasma tiene que descansar justo después de espantar a alguien? Notemos que esto añadirá un rango adicional de descanso y después uno de actividad. Así que la cantidad total de rangos activos será $O(MX * log(N) + M)$ en el peor caso. Si por cada rango activo podemos encontrar de forma óptima si se espanta a alguna persona, entonces se podrá resolver el problema en tiempo.

Procesemos los fantasmas desde $i = 0$ hasta $i = N - 1$ y llevemos un BST (puede ser std::set en C++) con el tiempo de llegada de todas las personas activas (que no han sido espantadas hasta el momento).

Para el fantasma i, simulemos todos los rangos activos de este fantasma empezando desde el segundo 0. Para saber si este fantasma i espanta a alguien en el rango activo que empieza en el segundo x, búsquemos en el BST si hay alguna persona activa que llegó en algún segundo del rango $[x - i, x - i + p_i)$, ya que una persona que visite la habitación i en el segundo T, tendría que haber entrado a la casa en el segundo $T - i$, y si hay múltiples personas que llegaron en ese rango, tomamos la que llegó primero. Esto se puede hacer en $O(log(M))$ con $lower\_bound(x - i)$ de un std::set en C++. Si espanta a alguien, anotemos su respuesta, eliminemos a la persona del BST (ya que abandona la casa) y simulemos el siguiente rango activo.

Complejidad: $O((MX * log(N) + M) * log(M))$.

Integer Multiplicative Persistence

Por: Juan Pablo Marín

Para este problema habría que realizar la operacion tal como se indica en la descripción del problema. Para hacer la multiplicación de cada dígito se puede realizar obteniendo cada dígito como el residuo de dividir el número entre 10 y después quitarlo dividiendo el número entre 10, hasta que ya no haya más dígitos que quitar. Se repite este proceso hasta que la multiplicación resultanto tenga solo 1 dígito y se regresa la cantidad de veces que se tuvo que realizar la operación.

John in the Amusement Park

Por: Eddy Ramírez

Este problema se puede resolver utilizando DP. Al momento que abre el parque $tiempo = 0$ John debe tomar la desición de participar o no participar en alguna de las atracciones. Suponiendo que John decide participar en la atracción i iniciando en el tiempo $t_{i,j}$, entonces John abrá recibido un total h_i de felicidad en el tiempo $t_{i,j} + d_i$, y se encontrará con tener que tomar la misma desición que tenía al principio, ¿Qué atracción de las que puede participar que inician en $t_{i,j} + d_i$ le ayudaría a obtener la máxima felicidad?. Podemos observar entonces que hay una relación de recurrencia para la máxima felicidad $H(tiempo)$ que John puede obtener que depende de las atracciones en las que puede participar que inician en o después de $tiempo$, esto es, en las atracciones que tienen algún $t_{i,j} \geq tiempo$, esta relación está dada entonces por:

$H(tiempo) = max(H(t_{i,j} + d_i) + h_i)$

Entonces si se implementa esta relación de recurrencia, buscamos el valor de $H(0)$, el máximo que John puede obtener desde que abre el parque. Manteniendo memorización en cada uno de los tiempos posibles, se necesitará $O(T)$ de memoria, y $O(S^2)$ en cómputo, donde $S = \sum_{i=1}^{N} t_i$

K-Binary Repetitive Numbers

Por: Juan Pablo Marín, Abraham Macías, Alan Enrique Ontiveros

La idea de solución se basa en el principio de inclusión y exclusión. La observación importante es que una cadena de tamaño K solo puede ser generada por cadenas más pequeñas con un tamaño d que divida a K. Entonces, si tenemos que d divide a K las 2^d cadenas de tamaño d se pueden concatenar K/d veces para tener una cadena de tamaño K que es K-binary repetitive. Sin embargo, si contamos todas esas cadenas podríamos estar contando cadenas de más. Si una cadena S es K-binary repetitive y es formada por una cadena de tamaño d que a su vez es d-binary repetitive, entonces estaremos contando S tantas veces como divisores tenga d. Consideremos entonces solo cadenas de longitud $d = K/p$ tal que p es un número primo que divide a K, entonces todas las 2^d cadenas NO son K-binary repetitive. Entonces podemos contar todas las cadenas que no son K-binary repetitive usando el teorema de inclusión exclusión con todos los subconjuntos que se pueden generar de los factores primos p que dividen a K.

La cantidad de cadenas K-binary repetitive que hay será restar las cadenas no K-binary repetitive de las posibles 2^K cadenas, y esto es igual a:

$$2^K - \sum_{k=0}^{w(K)} (-1)^k \sum_{1 \le i_1 < \cdots < i_k \le w(K)} 2^{K/\prod_{j=1}^{k} p_{i_k}}$$

Que tiene complejidad $O(2^w(K))$ donde $w(K)$ es la cantidad de factores primos de K.

Para poder responder de manera eficiente las preguntas hay que precomputar los factores primos de cada posible N, esto se puede hacer con una criba en $O(N)$.