



Gran Premio de México 2020 - Tercera fecha

05 de Diciembre de 2020

Libro de soluciones

Este documento contiene las soluciones esperadas para los 14 problemas usados durante la competencia.

Acing the contest

Como los concursantes deben escoger el orden en el que trabajan, se puede usar una máscara de bits para representar a los concursantes que ya han resuelto problemas.

De la descripción del problema cada concursante tiene tres opciones

- Hacer el problema j
- No hacer el problema j
- Dejar de resolver problemas

Sea $f(i, j, e)$ el máximo score que se puede obtener con el concursante actual que está resolviendo problemas, en el problema j , con energía e i en este caso es la máscara de bits.

Si el concursante hace la tarea j entonces el score será $v_j + f(i, j + 1, e - a_j)$ Si no la hace el score será $f(i, j + 1, e)$

Si deja de trabajar para calcular el score habrá que buscar con cada competidor k que no ha trabajado y el score será el máximo que se pueda obtener de $f(i|(1 \ll k), j, e_k)$ donde k es el índice de cada concursante que no ha trabajado aún.

La complejidad entonces será $O(N * (2^N) * P * E)$, donde E es el máximo valor de energía.

Beautiful necklaces

La idea está basada en el problema de encontrar el subarreglo más largo que contiene el mismo elemento, en un arreglo de enteros usando un Segment Tree. Ese problema se puede resolver guardando los siguientes valores en los nodos del Segment Tree:

- Máximo subarreglo con el mismo elemento que hemos encontrado en el rango del nodo hasta el momento
- El prefijo más largo con el mismo elemento en el rango del nodo
- El sufijo más largo con el mismo elemento en el rango del nodo

Cuando combinamos dos nodos, el hijo izquierdo y el hijo derecho, hay que quedarnos con el máximo subarreglo entre ambos nodos, pero también hay que considerar que el sufijo del nodo izquierdo se puede combinar con el prefijo del derecho si tienen el mismo elemento. El mayor prefijo del padre será el del hijo izquierdo y el mayor sufijo será el del hijo derecho, pero hay que considerar que, por ejemplo, si el hijo izquierdo tiene los mismos elementos en todo su rango y ese elemento es igual al prefijo del hijo derecho, entonces todo el nodo izquierdo se puede combinar con el prefijo del nodo derecho para convertirse en el mayor prefijo del padre; algo parecido sucede con el sufijo.

Para resolver el problema propuesto, podemos considerar los collares como arreglos de colores, sin embargo, hay que considerar dos cosas:

1. Como los collares son circulares, entonces los arreglos también deben serlo. Esto se puede manejar usando el truco de duplicar el arreglo con una copia de sí mismo. Sólo hay que tener cuidado cuando el collar tiene el mismo color en todas sus cuentas.
2. Las operaciones que puede hacer Alice se pueden ver como insertar o remover un elemento de un arreglo, por lo que en vez de un Segment Tree, necesitamos usar un Treap, ya que nos permite soportar estas operaciones. La mezcla de los nodos en el Treap será similar a como sería en el Segment Tree.

Una última consideración para la implementación es que los nodos del Treap deben guardar quién es su padre, ya que la entrada del problema no indica en qué collar Alice está haciendo la operación, sólo nos da los IDs de las cuentas, por lo que es necesario encontrar la raíz del Treap donde está el nodo que representa esos ID con tal de aplicar las operaciones.

Para mantener la belleza máxima de entre todos los collares que tiene Alice, se puede usar un map o un multiset.

Counting triangles

Para formar un triángulo se deben elegir dos líneas de las $N + 2$ que van de arriba hacia abajo en la imagen y una de las $K + 1$ líneas que son paralelas a la base. Entonces la solución es el producto de cuántas formas se pueden tomar dos de un conjunto de $N + 2$ por $K + 1$. Esto se calcula con la siguiente fórmula. $\frac{(N+2)*(N+1)*(K+1)}{2}$

Hay que tener cuidado para calcular el residuo. Una manera de calcularlo a pesar de tener la división es considerar que siempre alguno de $N + 2$ y de $N + 1$ será par, entonces primero se divide y se realizan todos los productos calculando el módulo entre cada uno de ellos.

Debugging the network

La solución es directa a la descripción del problema. Hay que dividir la cadena en números y letras, e interpretar los números de manera cuidadosa. Una vez hecho eso hay que repetir cada letra la cantidad de veces que especifica el número previo a el.

End of the year bonus

El bono $f(i)$ de la i -ésima persona se obtiene de la siguiente manera.

Si $p_i > p_{i-1}$ (tuvo mejor desempeño que la persona a su izquierda), entonces: $f(i) = B + f(i-1)$

Si $p_i > p_{i+1}$ (tuvo mejor desempeño que la persona a su derecha), entonces: $f(i) = B + f(i+1)$

Si $p_i > p_{i-1}$ y $p_i > p_{i+1}$ (tuvo mejor desempeño que las dos personas), entonces $f(i) = \max(B + f(i-1), B + f(i+1))$

Solo hay que tener cuidado con la implementación para considerar que la primer persona está sentada a la derecha de la última.

Fit them all

Una vista rápida a los límites establecidos en el problema muestra que la cantidad máxima de posible de cajas a posicionar no será muy grande. Esto permite idear una estrategia de solución en la que se pruebe colocar primero la caja más grande posible dentro de la plataforma, digamos de dimensiones $K \times K \times K$ y en cada posible posición que se pueda colocar esta caja colocar la siguiente caja menor en tamaño, y así sucesivamente hasta poner la caja más pequeña de dimension $1 \times 1 \times 1$. Si no hay una manera de posicionar las K cajas, entonces se intenta lo mismo comenzando con la caja que mide $(K-1) \times (K-1) \times (K-1)$, y así hasta encontrar una dimension de caja inicial en la que se puedan colocar todas las cajas, la respuesta sería esa dimensión. Sin embargo probar todas las posibles posiciones en cada tamaño posible de caja lleva la solución a Tiempo Límite.

Se puede observar que siempre que exista una solución donde se posicionen K cajas, existe tambien una solución en la que estas K cajas comparten al menos un lado horizontal y un lado vertical, aplicando esta consideración se puede seguir con la misma estrategia descrita previamente, pero reduciendo considerablemente la cantidad de posiciones que se prueban al intentar colocar una nueva caja posible. Entonces en la estragia se colocará siempre la caja más grande que queremos probar en la solución en una posición tal que al menos un lado horizontal y un lado vertical de esta caja compartan un lado horizontal y un lado vertical de la plataforma, y despues se intenta colocar de una por una en orden de descendente de tamaño las demás cajas, colocándolas siempre en posiciones donde al menos uno de sus lados horizontales y uno de sus lados verticales se comparta con alguna caja ya posicionada o con la plataforma.

Gold Fever

Sea $f(i)$, la máxima cantidad de dinero que se puede obtener si estamos en la casilla i $f(i) = g_i + \max_{j \in [a_i, b_i]} (f(i+j) - g_j)$.

Esta solución es de orden de complejidad $O(N^2)$, y daría tiempo límite, sin embargo se puede optimizar usando un segment tree, la DP es la misma, pero lo haremos de la siguiente manera.

Iniciando en la última posición, y recorriendo hacia atrás suponiendo que tenemos la solución de $f(j)$ para $j > i$ y que hemos guardado previamente en el segment tree las soluciones para los $f(j)$. Queremos calcular $f(i)$ con una consulta al segment tree obtenemos el valor de $\max_{j \in [a_i, b_i]} (f(i+j) - g_j)$, con j en el rango $[a_i, b_i]$. Después actualizamos el valor del segment tree en i , porque ya obtuvimos $f(i)$. La complejidad de esta implementación es $O(N \log N)$, y correrá en el tiempo límite dado.

How to Work Less to Pass a Programming Course in Planet E-13

Dado que el problema pide encontrar todas las posibles combinaciones posible, es necesario hacer una búsqueda exhaustiva, las combinaciones pueden probarse con una mascara de bits que varie entre 1 y $2^n - 1$ donde n es el número de calificaciones, para cada combinación obtener la calificación que se obtendría y en caso de cumplir (ser a menos la calificación deseada) se agregan a un vector llevando el rastro de la cantidad mínima de problemas que permite obtener al menos la calificación deseada.

El vector se ordena de acuerdo a la mascara de bits encontrada para que quede en el orden deseado y se imprime el número mínimo de problemas a resolver, la cantidad de combinaciones que permiten obtener al menos la calificación deseada con el número mínimo de problemas y se recorre el vector imprimiendo las combinaciones.

Is this the best deal?

Dado que siempre habrá 3 cantidades, llamémoslas t_1 , t_2 , t_3 , y suponiendo que tenemos una función ‘descuento(valor)’ que regresa el valor que se habría que pagar por hacer una compra con un total de *valor*, entonces solo hay 5 posibles maneras de hacer las compras:

$$s_1 = \text{discount}(t_1) + \text{discount}(t_2) + \text{discount}(t_3)$$

$$s_2 = t_1 + \text{discount}(t_2 + t_3)$$

$$s_3 = \text{discount}(t_1 + t_2) + t_3$$

$$s_4 = \text{discount}(t_1 + t_3) + t_2$$

$$s_5 = \text{discount}(t_1 + t_2 + t_3)$$

La respuesta es el valor más pequeño entre s_1 , s_2 , s_3 , s_4 , y s_5 .

Jaime's greedy delivery

A simple vista Se visualiza un floyd warshal para obtener el tiempo minimo entre todos los pares de ciudadss puntos, seria razonable, pero dado que las distancias/tiempos entre cualesquiera par de puntos entre los que hay alguna conexion es siempre la misma, y que por la cantidad de aristas el grafo puede ser disperso, basta con hacer N busquedas en amplitud para sacar las distancias.

Ya que se tienen las distancias minimas como preproceso se puede calcular el tiempo entre puntos que Jaime debe seguir en la ruta, esto es una dp:

Sea $f(i, t)$ la maxima cantidad de dinero que Jaime puede hacer si se encuentra en el punto i , y le queda t tiempo los puntos de entrega se van a expandir en dos, i , e $i + 1$.

Si i es par, entonces está en un lugar de entrega normal, si i es impar, está en una entrega especial (estas le dan dinero).

Cuando i es par, puede rechazar el dinero y pasarse a la siguiente entrega es decir: $f(i + 2, t - tiempo[punto_i][punto_i + 2])$ o puede hacer el pedido especial $v_i + f(i + 1, t - tiempo[punto_i][punto_i + 1])$

Cuando i es impar debe pasarse al siguiente punto, sin mas (ya ha recibido el dinero) $f(i + 1, t - tiempo[punto_i][punto_i + 1])$

Resolviendo esta función en $f(0, T)$ se tendrá la máxima cantidad de dinero que Jaime puede obtener.

K contestants

Para formar el equipo se deben tomar i alumnos del grupo A, y $k-c-i$ Alumnos del grupo B Entonces se pueden tomar $\binom{A}{K-c}$ del grupo A y $\binom{B}{K-c-i}$ del grupo B, para así tomar exactamente c alumnos del grupo C como solicitan los profesores.

Supongamos que se puede formar el equipo con 0 alumnos de cualquiera de los grupos A y B, entonces se podrían tomar los $K-c$ alumnos de la siguientes formas: $\sum i = 0^{K-c} \binom{B}{K-c-i} = \binom{A+B}{K-c}$

Para eliminar los casos donde $i = 0$ o $i = K-c$ tenemos $\binom{A+B}{K-c} - \binom{A}{K-c} - \binom{B}{K-c}$ Para obtener la cantidad total de formas habrá que multiplicar esto por la cantidad de maneras de tomar los c alumnos del grupo C y entonces la solución sería: $(\binom{A+B}{K-c} - \binom{A}{K-c} - \binom{B}{K-c}) * \binom{C}{c}$

Hay que tener cuidado con el manejo de los residuos considerando los valores que se están restando, y contemplar los casos en donde no hay manera de formar algún equipo.

Let's count words

Los límites de este problema son suficientes para encontrar la cantidad de palabras usadas para generar las demás probando si cada par de palabras son iguales después de rotarlas una cantidad de veces. Digamos sea W_1 y W_2 dos palabras del mismo tamaño L , si después en alguna de las posibles L rotaciones de W_2 en alguna $W_1 = W_2$ entonces las dos palabras son generadas por una misma palabra. Entonces para resolver podemos inicializar un contador en 0, y para cada una de las N palabras probar, si ninguna de las palabras anteriores a ella son generadas por una misma palabra aumentar el contador en 1. Al finalizar con las N palabras, la respuesta está en el contador.

Mathematics society problem

Por la descripción que se da podemos observar que la cantidad total de dígitos a eliminar de N , digamos D es igual a la suma de la cantidad de veces que se debe eliminar cada uno de los 9 dígitos. El número resultante independientemente del orden en el que se eliminan los dígitos tendrá siempre un tamaño de $M = |N| - D$. Entonces, buscamos el número de mayor valor de entre los números que se pueden generar con exactamente M dígitos después de eliminar los dígitos con las restricciones dadas. Podemos notar entonces que como todos los posibles números a probar tienen la misma cantidad de dígitos, entonces, el número más grande comenzará siempre con el dígito más grande que se pueda quedar después de eliminar los D dígitos a N . De este modo podemos considerar una solución en la que en lugar de decidir qué dígitos de N se deben eliminar, determinar qué dígitos de N se deben quedar. Dado el dígito d en la posición i , este se debe quedar para generar el número más grande siempre y cuando se cumplan dos condiciones:

- Hay al menos una manera de eliminar todos los dígitos antes de la posición i que no han sido seleccionados.
- El dígito debe aparecer al menos D_d veces después de la posición i , de otro modo, no se podrían eliminar todas las ocurrencias del dígito d en N .

Network connection

Supongamos que conocemos un valor de frecuencia f que se quiere utilizar y queremos determinar si con ese valor de frecuencia se pueden posicionar las antenas de algún modo tal que no se exceda el presupuesto B . Una de las maneras de determinar esto es intentar colocar las antenas en posiciones de modo que se minimice el presupuesto utilizado con usando la frecuencia f , así si el presupuesto mínimo necesario usando la frecuencia f es mayor a B entonces no hay manera de acomodar la red de modo que se conecte toda la carretera con la frecuencia f y el presupuesto dado. Este subproblema se puede resolver con programación dinámica. Sea $DP[i][j]$ el menor costo de acomodar las primeras i antenas de modo que haya una conexión entre Nlogonia y la i -ésima antena estando esta en el metro j de la carretera. Podemos observar que el valor de $DP[i][j]$ está siempre determinado por las soluciones dadas para haber conectado Nlogonia hasta la antena $i - 1$ en las diferentes posibles posiciones que esta pueda estar, y que este valor será $DP[i][j] = \min(DP[i - 1][j - k] + \text{abs}(j - p_i))$ para k que está en el rango $[0, f]$. Para finalizar, hay que considerar que $DP[N][0...D]$ contiene los valores de conectar Nlogonia con las N antenas, entonces podemos observar que el presupuesto mínimo necesario para conectar las dos ciudades será $\min(DP[N][D - k])$ con k en el rango $[0, f]$.

Supongamos que hemos probado una frecuencia f y esta frecuencia puede ser usada para conectar toda la carretera, entonces se puede observar que la misma configuración de antenas puede conectar la carretera con cualquier frecuencia $f_1 \geq f$. Podemos entonces comprobar que la función de poder o no utilizar una frecuencia f es una función monótona creciente y por lo tanto podemos usar búsqueda binaria para encontrar el valor f más pequeño que tiene una solución usando la programación dinámica descrita para cada valor posible utilizado de f . El orden de complejidad de la solución es entonces $O(N * D * \log D)$.