Competencia Peruana de Informática Online 2020 - Solucionario

Federación Olímpica Peruana de Informática

12 de julio de 2020

Tutorial del problema: "Leche y Café"

Autor: Renato Chavez

Grupo 1 (Complejidad Esperada: $O(n)$)

Para resolver los casos de prueba de este grupo podemos simular el proceso usando alguna estructura que represente a una fracción (podemos definir un `struct` o usar `pair<int, int>`) y tendremos una complejidad de $O(n)$.

Definimos las variables L_i y C_i para denotar a la fracción de la bebida que es leche y café, respectivamente, para definir la siguiente recursión. Notemos que $L_0 = 1$ y $C_0 = 0$:

- Si el i-ésimo caracter es `L`:

$$L_i = \frac{L_{i-1}}{2} + \frac{1}{2}$$
$$C_i = \frac{C_{i-1}}{2}$$

- Si el i-ésimo caracter es `C`:

$$L_i = \frac{L_{i-1}}{2}$$
$$C_i = \frac{C_{i-1}}{2} + \frac{1}{2}$$

El resto se reduce a operar.

Grupo 2 (Complejidad Esperada: $O(n)$)

Para resolver los casos de prueba de este grupo ya no es posible usar la idea anterior, pues los valores del denominador ascienden hasta 2^{100000}, por lo cual tendremos que analizar el proceso detenidamente:

Podemos notar que gran parte de la responsabilidad de definir quién tiene mayor concentración se la lleva el último caracter. Además, la única forma de que $L_n = C_n = \frac{1}{2}$ se da cuando:

- Si s_n es `L`,

$$L_n = \frac{L_{i-1}}{2} + \frac{1}{2} = \frac{1}{2} \longrightarrow L_{n-1} = 0$$
$$C_n = \frac{C_{i-1}}{2} = \frac{1}{2} \longrightarrow C_{n-1} = 1$$

  Lo cual es inviable, pues inicialmente tenemos leche, así que siempre habrá leche en la bebida.

- Si s_n es `C`,

$$L_n = \frac{L_{i-1}}{2} = \frac{1}{2} \longrightarrow L_{n-1} = 1$$
$$C_n = \frac{C_{i-1}}{2} + \frac{1}{2} = \frac{1}{2} \longrightarrow C_{n-1} = 0$$

  Lo cual nos da a entender que no debemos agregar café en las primeras $n-1$ acciones, así que la única forma en la que puede haber un empate es cuando todas excepto la última letra son `L`.

Por otro lado, cuando ni L_{n-1} ni C_{n-1} son 0, tendremos que:

- Si s_n es `L`,

$$L_n = \frac{L_{i-1}}{2} + \frac{1}{2} > \frac{1}{2}$$
$$C_n = \frac{C_{i-1}}{2} < \frac{1}{2}$$

  Así que habría más leche que café.

- Si s_n es `C`,

$$L_n = \frac{L_{i-1}}{2} < \frac{1}{2}$$
$$C_n = \frac{C_{i-1}}{2} + \frac{1}{2} > \frac{1}{2}$$

  Así que habría más café que leche.

Ahora el problema se reduce a verificar el caso de desempate en $O(n)$ y revisar el último caracter.

Tutorial del problema: "Anagramas Especiales"

Autor: Alfred Chavez

Consideremos que Σ es el alfabeto compuesto por letras mayúsculas, minúsculas y dígitos.

Grupo 1 (Complejidad Esperada: $O(|s_1| \cdot |s_2| \log |s_1|)$)

Para resolver los casos de prueba de este grupo podemos notar que solo una cadena de longitud $|s_1|$ puede ser anagrama de s_1, así que la cantidad de dicha subcadenas de s_2 es $\max\{0, |s_2| - |s_1| + 1\}$, o simplemente $O(|s_2|)$.

Además, podemos definir como "representante" de un conjunto de anagramas al mínimo lexicográfico de todos. Por ejemplo, para el conjunto de anagramas $\{ABC, ACB, BCA, BAC, CBA, CAB\}$, el representante sería `ABC`. Es sencillo notar que el representante es el resultado de ordenar los caracteres de cualquiera de los anagramas.

Las anteriores observaciones nos permiten fijar cada una de las cadenas a analizar, ordenarlas y compararlas con el resultado de ordenar s_1, lo cual tomará $O(|s_1| \log |s_1|)$, dándonos una complejidad final de $O(|s_1| \cdot |s_2| \log |s_1|)$.

Grupo 2 (Complejidad Esperada: $O(|s_1| \cdot |s_2| \cdot |\Sigma|)$)

Para resolver los casos de prueba de este grupo debemos notar que en realidad podemos ahorrarnos el ordenar los caracteres de las cadena y almacenar las frecuencias de cada caracter en un vector de tamaño 62. De esta manera podemos verificar que una cadena es anagrama de otra si sus vectores de frecuencias coinciden. Para evitar factores logarítmicos, podemos mapear las letras minúsculas al rango $[0, 25]$, las mayúsculas al rango $[26, 51]$ y los dígitos al rango $[52, 61]$ mediante alguna función.

Finalmente, podemos verificar en $O(\Sigma)$ la igualdad de las frecuencias para cada cadena posible, dándonos una complejidad de $O(|s_1| \cdot |s_2| \cdot |\Sigma|)$.

Grupo 3 (Complejidad Esperada: $O(|s_1| + |s_2| \cdot |\Sigma|)$)

Para resolver los casos de prueba de este grupo aplicaremos la idea anterior pero usando la técnica de *sliding window*, la cual nos permitirá mejorar la complejidad significativamente notando lo siguiente:

Sea F_i el vector de frecuencias de la subcadena $s_2[i, i + |s_1| - 1]$, entonces podemos obtener F_{i+1} mediante la siguiente igualdad:

$$F_{i+1} = F_i - s_2[i] + s_2[i + |s_1|]$$

Donde $+x$ significa sumar 1 al contador del mapeo del caracter x y $-x$ significa restar 1.

Esto nos permite pasar de una posición inicial a la siguiente realizando solo 2 operaciones constantes, así que obtendremos F_{i+1} a partir de F_i en $O(1)$.

Gracias a la técnica anterior, podemos obtener de manera directa F_1 en $O(|s_1|)$ y luego obtener el resto de F_i en $O(|s_2|)$ (pues a lo mucho hay $O(|s_2|)$ vectores F_i que calcular) y verificar si la cadena es un anagrama en $O(|\Sigma|)$.

La complejidad obtenida es de $O(|s_1| + |s_2| \cdot |\Sigma|)$.

Tutorial del problema: "Clústeres"

Autor: Aldo Culquicondor

Grupo 1 y 2 (Complejidad Esperada: $O(kn)$)

Para resolver los casos de prueba de este grupo nos basta realizar la simulación del proceso usando una búsqueda lineal para encontrar la máquina con mayor capacidad requerida y menor índice. Podemos evitar factores logarítmicos extra usando un arreglo para la capacidad restante de cada máquina (cap_i) y así obtener una complejidad de $O(kn)$. Al final, la memoria usada por la máquina i será de $M - cap_i$

Tutorial del problema: "4 de julio"

Autor: Walter Erquinigo

Grupo 1, 2 y 3 (Complejidad Esperada: $O(2^n)$)

Para resolver los casos de prueba de este grupo nos basta usar un backtracking similar al del problema de la mochila, pues solo necesitamos saber cuáles platos comerá Aldo para determinar su llenura y felicidad final. Planteamos la función recursiva $f(pos, llenura, felicidad, cnt)$, la cual probará todos los posibles casos de elección de los platos dado que ya se procesaron los primeros pos y se alcanzó una llenura $llenura$ y felicidad $felicidad$ al elegir cnt de ellos. Consideremos que las descripciones de los platos están almacenadas en los arreglos l y f como se explica en el enunciado:

- Si $llenura + l_i \leq 100$, podemos usar la transición $f(pos + 1, llenura + l_i, felicidad + f_i, cnt + 1)$, que significa que elegimos el plato pos.
- Podemos ignorar el plato y pasar a $f(pos + 1, llenura, felicidad, cnt)$.

Cuando lleguemos a $pos = n$, todos los platos habrán sido procesados, así que debemos maximizar nuestra respuesta actual (variable global) con cnt si es que $felicidad >= 100$.

Grupo 4 (Complejidad Esperada: $O(nlf)$)

Consideremos que l es la máxima llenura posible y f es 100 para este problema.

Para resolver los casos de prueba de este grupo podemos plantear una recursión similar a la anterior pero obviando la variable cnt, pues es la que deseamos maximizar. Nuestra nueva recursión $f(pos, llenura, felicidad)$ nos dara la máxima cantidad de platos que puede comer Aldo dado que ya se han procesado los primeros pos y se ha alcanzado una llenura $llenura$ y una felicidad $felicidad$. Ahora podemos analizar las transiciones:

- Si $llenura + l_i \leq 100$, un posible candidato a solución es $1 + f(pos + 1, llenura + l_i, felicidad + f_i)$.

- Podemos ignorar el plato y un posible candidato a solución es $f(pos + 1, llenura, felicidad)$.

Ahora, el valor máximo de felicidad es 5000, lo cual haría que nuestra complejidad sea de $O(n^2lf)$. Para reducirla, debemos notar que solo nos interesa que un conjunto de platos obtenga una felicidad mayor o igual a 100 para que sea tomado en cuenta, así que podemos asumir que todos los valores ≥ 100 son válidos y los que son < 100 no lo son. Finalmente, en el tercer argumento, consideraremos siempre el mínimo entre 100 y su valor real.

En el caso base, es decir, cuando $pos = n$, devolveremos 0 solo si la felicidad es 100 (para que el conjunto sea válido) y devolveremos $-\infty$ si no lo es. Ya que hay $O(nlf)$ estados y cada uno se procesa en $O(1)$, nuestra complejidad es de $O(nlf)$.

Bonus: Se puede resolver el problema en $O(2^{\frac{n}{2}} + l^2 f)$ usando Meet in the Middle.