EGOI Qualifier 2022 Contest 1 - Solucionario

Federación Olímpica Peruana de Informática

10 de julio de 2022

Tutorial del problema: "El paseo turístico más largo"

Autor(es): Racsó Galván

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

Para resolver este grupo nos basta con aprovechar el hecho de que solo existe un único camino entre cada par de nodos; por lo tanto, nos basta con fijar cada nodo s y hallar la distancia desde s hacia los demás nodos usando DFS o BFS.

La complejidad será de $O(n^2)$, pues haremos n veces un DFS/BFS de complejidad $O(n)$.

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

Para resolver este grupo hay diversas formas:

- Solución teórica: Seleccionamos un nodo cualquiera u y hallamos el nodo que esté más alejado de u, sea v. Hallamos la distancia entre v y el nodo más alejado w y la respuesta será $dis(u, v) + 1$. Ya que el grafo es un árbol podemos hallar los nodos más alejados con BFS/DFS en $O(n)$.
- Solución con DFS: Podemos mantener una variable global ans, asumimos que 1 es la raíz del árbol y definimos $DFS(u)$ como la distancia más larga desde u hacia alguna de las hojas de su subárbol.

  Entonces podemos calcular:

$$DFS(u) = \begin{cases} 0 & \text{Si } u \text{ es una hoja (no tiene hijos)} \\ \max\limits_{v \text{ es hijo de } u} \{DFS(v)\} + 1 & \text{Si } u \text{ tiene hijos} \end{cases}$$

  Por otro lado, si tenemos un nodo u con hijos $v_1, v_2, \ldots, v_k$, el camino más largo que pasa por el nodo u como punto más alto es igual a la suma de los dos máximos de todos los $DFS(v_i) + 1$ (o solo el máximo si $k = 1$). Podemos obtener el máximo y el segundo máximo en tiempo $O(k) = O(|Adj[u]|)$ para actualizar la respuesta ans. Ya que el DFS visita cada nodo una sola vez, la complejidad será:

$$T(n) = \sum_{i=1}^{n} |Adj[u]| = O(n)$$

  Donde la segunda igualdad viene del hecho de que la suma de las listas de adyacencia es igual a $2|E|$, pero al ser un árbol se tiene que $E = O(n)$.

Tutorial del problema: "¿Cuántos números?"

Autor(es): Racsó Galván

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

Para resolver este grupo podemos usar Programación Dinámica y definir la función $DP(pos, rem)$ que nos dará la cantidad de números de n cifras que se pueden construir con los dígitos válidos y que sean múltiplos de b **dado que** ya hemos puesto sus primeros pos dígitos y nos han llevado a un residuo de rem módulo b.

Entonces, planteamos la siguiente observación:

Si tenemos un entero $\overline{a_1 a_2 ... a_k}$) y le queremos agregar un dígito x a la derecha, el residuo del resultado cambia a:

$$R(\overline{a_1 a_2 ... a_k x}) = 10 \cdot R(\overline{a_1 a_2 ... a_k}) + x \mod b$$

Así que podemos calcular el nuevo residuo en $O(1)$ luego de fijar el dígito a agregar a la derecha.

Tendremos entonces la siguiente recurrencia:

$$DP(n, 0) = 1, DP(n, x > 0) = 0$$

$$DP(pos, rem) = \sum_{i=1}^{k} DP(pos + 1, (10 \cdot rem + c_i) \mod b)$$

Lo cual nos da una complejidad de $O(nbk)$, pues hay $O(nb)$ estados y cada uno realiza un trabajo de $O(k)$.

Grupo 2 (Complejidad Esperada: $O\left(b(k + b^2 \log n)\right)$)

Para resolver este grupo podemos plantear un grafo en función al hecho de agregar un dígito a la derecha del número. Notemos que el número formado en sí no nos importa, solo su residuo módulo b, así que definiremos el siguiente grafo:

$$V = \{0, 1, \ldots, b - 1\}$$

$$E = \{(rem, (10 \cdot rem + c_i) \mod b) \forall i = 1, \ldots, k\}$$

Así que el problema se reduce a hallar la cantidad de caminos desde el nodo 0 (pues inicialmente tenemos residuo de 0 al ser 0 nuestro número) al nodo 0 (queremos los múltiplos de b).

La teoría de grafos nos señala que si construimos la matriz de adyacencia M del grafo anterior, almacenando en $M_{i,j}$ la cantidad de aristas $(i, j) \in E$ y la elevamos a la n, tendremos en $M^n_{i,j}$ la cantidad de caminos de longitud n desde el nodo i al nodo j.

Podemos usar exponenciación rápida para obtener una complejidad de $O(b^3 \log n)$ para hallar $M^n_{0,0}$. Ya que construir la matriz de adyacencia nos toma $O(bk)$, tendremos una complejidad de $O\left(b(k + b^2 \log n)\right)$.

Nota: Hay que tener cuidado con el orden en el que se va a multiplicar, notemos que la operación de agregar un dígito a algún extremo **no es conmutativa**. Por otro lado hay que considerar que si la cantidad de cifras que quiero agregar al final es k hay que multiplicar por 10^k, no solo por 10.

Grupo 3 (Complejidad Esperada: $O\left(k + b^2 \log n)\right)$)

Para resolver este grupo debemos notar que no nos importa mucho saber la cantidad de caminos entre los nodos u y v, sino la cantidad de caminos que terminan en el nodo v, pues esto representa el residuo final que tendrá el número construido.

Esto nos permite reducir la matriz M a solo un arreglo A (recordemos que A_i almacena la cantidad de números construidos con la misma longitud tales que su residuo es i), de manera que la actualización ahora se podrá hacer en $O(b^2)$ en una forma muy similar a la del grupo 2.

La complejidad se reduce a $O\left(k + b^2 \log n)\right)$, pues construir A toma solo $O(k)$.

Tutorial del problema: "Diferencias Minimas"

Autor(es): Racsó Galván

Grupo 1 (Complejidad Esperada: $O\left(n! \cdot n\right)$)

Para resolver este grupo nos basta usar la función `next_permutation` para probar todas las posibles permutaciones de A (y por ende, emparejamientos) y así obtener la mínima suma de diferencias.

La complejidad será de $O(n! \cdot n)$, pues hay $n!$ permutaciones a analizar y cada una nos toma $O(n)$.

Grupo 2 (Complejidad Esperada: $O(n \cdot 2^n)$)

Para resolver este grupo podemos notar que el problema, al ser un emparejamiento, se puede resolver usando Programación Dinámica con Bitmask. Definiremos $DP(mask)$ como la menor suma posible **dado que solo nos quedan los elementos de A cuyo bit en $mask$ esté prendido**. Por ejemplo, si tenemos

$$DP(5) = DP(\overline{101}_2) \rightarrow \text{Los elementos } a_0 \text{ y } a_2 \text{ siguen disponibles para asignar}$$

Obviamente nuestra respuesta la obtendremos al llamar a $DP(2^n-1)$; sin embargo, tenemos que considerar algunas cosas para poder resolver el problema con el único argumento que tiene nuestra función:

- La cantidad de emparejamientos ya realizados es $n - |mask|$, donde $|m|$ es la cantidad de bits prendidos de m. Esto quiere decir que el siguiente elemento de B a emparejar es $B_{n-|mask|}$.
- Podemos probar todos los bits prendidos de $mask$ como el elemento de A a emparejar con $B_{n-|mask|}$ y obtener el mínimo de todos.

Entonces, nuestra recursión se verá de la siguiente forma:

$$DP(mask) = \begin{cases} 0 & mask = 0 \\ \min\limits_{\substack{k=0 \\ \text{el bit } k \text{ esta prendido en } mask}}^{n-1} \{DP(mask \oplus 2^k) + |a_k - b_{n-|mask|}|\} & mask > 0 \end{cases}$$

Donde $\oplus$ es la función Bitwise XOR y nos permite apagar el bit k prendido en $mask$ con la expresión $mask \oplus 2^k$.

Con lo anterior podemos obtener una complejidad de $O(n2^n)$.

Grupo 3 (Complejidad Esperada: $O(n \log n)$)

Para resolver este grupo debemos notar que la mejor asignación es el k-ésimo menor de cada grupo, así que nos basta con ordenar A y B y finalmente asignar los elementos en las mismas posiciones como parejas.

Prueba

Supongamos que la observación anterior **no** es correcta, eso quiere decir que existe un emparejamiento (a'_i, b'_i) tal que $b'_i \leq b'_{i+1}$ para todo i válido pero existe alguna posición j tal que $a'_j > a'_{j+1}$ y cuyo valor es **estrictamente menor que** el de la solución propuesta en al menos un caso.

Entonces, tenemos 3 casos posibles para a'_j:

- $a'_j \leq b'_j$:

$$|a'_j - b'_j| = b'_j - a'_j$$
$$|a'_{j+1} - b'_{j+1}| = b'_{j+1} - a'_{j+1}$$
$$Costo_{optimo} = b'_j - a'_j + b'_{j+1} - a'_{j+1}$$

  Sin embargo, si emparejamos b'_j con a'_{j+1} y b'_{j+1} con a'_j obtendremos

$$|a'_j - b'_{j+1}| = b'_{j+1} - a'_j$$
$$|a'_{j+1} - b'_j| = b'_j - a'_{j+1}$$
$$Costo_{ordenados} = b'_{j+1} - a'_j + b'_j - a'_{j+1}$$

  Si restamos los costos:

$$Costo_{optimo} - Costo_{ordenados} = b'_j - a'_j + b'_{j+1} - a'_{j+1} - (b'_{j+1} - a'_j + b'_j - a'_{j+1}) = 0$$

Notamos que en este caso los dos costos son iguales, así que la solución óptima no es estrictamente menor que la propuesta.

- $b'_j < a'_j \leq b'_{j+1}$:
  En este caso, tenemos 2 casos posibles para a'_{j+1}:
  - $b'_j \leq a'_{j+1}$: En este caso tenemos que:

$$|a'_j - b'_j| = a'_j - b'_j$$
$$|a'_{j+1} - b'_{j+1}| = b'_{j+1} - a'_{j+1}$$
$$Costo_{optimo} = a'_j - b'_j + b'_{j+1} - a'_{j+1}$$

    Sin embargo, si emparejamos b'_j con a'_{j+1} y b'_{j+1} con a'_j obtendremos

$$|a'_j - b'_{j+1}| = b'_{j+1} - a'_j$$
$$|a'_{j+1} - b'_j| = a'_{j+1} - b'_j$$
$$Costo_{ordenados} = b'_{j+1} - a'_j + a'_{j+1} - b'_j$$

    Si restamos los costos:

$$Costo_{optimo} - Costo_{ordenados} = a'_j - b'_j + b'_{j+1} - a'_{j+1} - (b'_{j+1} - a'_j + a'_{j+1} - b'_j) = 2a'_j - 2a'_{j+1} = 2(a'_j - a'_{j+1})$$

    Sin embargo, el valor $a'_j - a'_{j+1}$ es positivo, lo cual implica que $Costo_{optimo}$ es mayor que $Costo_{ordenados}$, lo cual es una contradicción, pues el primero debería ser menor.
  - $a'_{j+1} < b'_j$:
    En este caso tenemos que:

$$|a'_j - b'_j| = a'_j - b'_j$$
$$|a'_{j+1} - b'_{j+1}| = b'_{j+1} - a'_{j+1}$$
$$Costo_{optimo} = a'_j - b'_j + b'_{j+1} - a'_{j+1}$$

    Sin embargo, si emparejamos b'_j con a'_{j+1} y b'_{j+1} con a'_j obtendremos

$$|a'_j - b'_{j+1}| = b'_{j+1} - a'_j$$
$$|a'_{j+1} - b'_j| = b'_j - a'_{j+1}$$
$$Costo_{ordenados} = b'_{j+1} - a'_j + b'_j - a'_{j+1}$$

    Si restamos los costos:

$$Costo_{optimo} - Costo_{ordenados} = a'_j - b'_j + b'_{j+1} - a'_{j+1} - (b'_{j+1} - a'_j + b'_j - a'_{j+1}) = 2a'_j - 2b'_j = 2(a'_j - b'_j)$$

    Sin embargo, el valor $a'_j - b'_j$ es positivo, lo cual implica que $Costo_{optimo}$ es mayor que $Costo_{ordenados}$, lo cual es una contradicción, pues el primero debería ser menor.

- $b'_{j+1} < a'_j$:
  En este caso, tenemos 3 casos posibles para a'_{j+1}:

- $b'_{j+1} \leq a'_{j+1}$: En este caso tenemos que:

$$|a'_j - b'_j| = a'_j - b'_j$$
$$|a'_{j+1} - b'_{j+1}| = a'_{j+1} - b'_{j+1}$$
$$Costo_{optimo} = a'_j - b'_j + a'_{j+1} - b'_{j+1}$$

Sin embargo, si emparejamos b'_j con a'_{j+1} y b'_{j+1} con a'_j obtendremos

$$|a'_j - b'_{j+1}| = a'_j - b'_{j+1}$$
$$|a'_{j+1} - b'_j| = a'_{j+1} - b'_j$$
$$Costo_{ordenados} = a'_j - b'_{j+1} + a'_{j+1} - b'_j$$

Si restamos los costos:

$$Costo_{optimo} - Costo_{ordenados} = a'_j - b'_j + a'_{j+1} - b'_{j+1} - (a'_j - b'_{j+1} + a'_{j+1} - b'_j) = 0$$

Notamos que en este caso los dos costos son iguales, así que la solución óptima no es estrictamente menor que la propuesta.

- $b'_j \leq a'_{j+1} < b'_{j+1}$:
En este caso tenemos que:

$$|a'_j - b'_j| = a'_j - b'_j$$
$$|a'_{j+1} - b'_{j+1}| = b'_{j+1} - a'_{j+1}$$
$$Costo_{optimo} = a'_j - b'_j + b'_{j+1} - a'_{j+1}$$

Sin embargo, si emparejamos b'_j con a'_{j+1} y b'_{j+1} con a'_j obtendremos

$$|a'_j - b'_{j+1}| = a'_j - b'_{j+1}$$
$$|a'_{j+1} - b'_j| = a'_{j+1} - b'_j$$
$$Costo_{ordenados} = a'_j - b'_{j+1} + a'_{j+1} - b'_j$$

Si restamos los costos:

$$Costo_{optimo} - Costo_{ordenados} = a'_j - b'_j + b'_{j+1} - a'_{j+1} - (a'_j - b'_{j+1} + a'_{j+1} - b'_j) = 2b'_{j+1} - 2a'_{j+1} = 2(b'_{j+1} - a'_{j+1})$$

Sin embargo, el valor $b'_{j+1} - a'_{j+1}$ es positivo, lo cual implica que $Costo_{optimo}$ es mayor que $Costo_{ordenados}$, lo cual es una contradicción, pues el primero debería ser menor.

- $a'_{j+1} < b'_j$:
En este caso tenemos que:

$$|a'_j - b'_j| = a'_j - b'_j$$
$$|a'_{j+1} - b'_{j+1}| = b'_{j+1} - a'_{j+1}$$
$$Costo_{optimo} = a'_j - b'_j + b'_{j+1} - a'_{j+1}$$

Sin embargo, si emparejamos b'_j con a'_{j+1} y b'_{j+1} con a'_j obtendremos

$$|a'_j - b'_{j+1}| = a'_j - b'_{j+1}$$
$$|a'_{j+1} - b'_j| = b'_j - a'_{j+1}$$

$$Costo_{ordenados} = a'_j - b'_{j+1} + b'_j - a'_{j+1}$$

Si restamos los costos:

$$Costo_{optimo} - Costo_{ordenados} = a'_j - b'_j + b'_{j+1} - a'_{j+1} - (a'_j - b'_{j+1} + b'_j - a'_{j+1}) = 2b'_{j+1} - 2b'_j = 2(b'_{j+1} - b'_j)$$

Sin embargo, el valor $b'_{j+1} - b'_j$ es no negativo, lo cual implica que $Costo_{optimo}$ es mayor que $Costo_{ordenados}$, lo cual es una contradicción, pues el primero debería ser menor.

Como conclusión, el tener A ordenado nos da la respuesta óptima.

Tutorial del problema: "Conteo de cifras"

Autor(es): Racsó Galván

Grupo 1 (Complejidad Esperada: $O((r-l+1)\log r)$)

Para resolver este grupo nos basta con iterar sobre cada valor $i = l, \ldots, r$, contar su cantidad de cifras c_i en $O(\log i)$ y agregar a la respuesta $i \cdot c_i$.

La complejidad será de $O((r-l+1)\log r)$.

Grupo 2 (Complejidad Esperada: $O(\log r)$)

Para resolver este grupo debemos notar que en el rango $[10^k, 10^{k+1}-1]$ todos los números tienen la misma cantida de cifras, así que podemos calcular el rango $[x, y]$ que sea la intersección de $[10^k, 10^{k+1}-1]$ y $[l, r]$ para agregar $k \cdot (x + (x+1) + \ldots + y) = k \cdot \left(\frac{x(x+1)}{2} - \frac{(y-1)y}{2}\right)$ a la respuesta.

Ya que $10^k \leq r \to k \leq \log r$ como máximo, tendremos una complejidad de $O(\log r)$.

Nota 1: Ya que 10^{19} no entra en `long long`, era necesario usar `unsigned long long` para dar una respuesta correcta.

Nota 2: Para aquellas que no sabían inverso modular, tenían que calcular $\frac{x(x+1)}{2}$ analizando el caso cuando x es par (divides x entre 2) o impar (divides el $x+1$ entre 2). La otra opción era multiplicar $x(x+1)$ con $\frac{MOD+1}{2}$ (note que este valor es un entero).

Nota 3: Para facilitar la implementación podemos definir la función $f(x)$ que calcula el conteo de cifras desde el 1 hasta el x, de forma que la respuesta será $f(r) - f(l-1)$.