Proceso Selectivo IOI 2023 - Contest 1 - Solucionario

Federación Olímpica Peruana de Informática

18 de junio de 2023

Tutorial del problema: "Excursiones"

Autor(es): Racsó Galván

Grupo 1 (Complejidad Esperada: $O((n+q)\log n)$)

Ya que no hay modificaciones en la cantidad de flamencos en las islas, podríamos usar DSU para saber cuándo un puente debe ser construido o no y luego ver una manera eficiente de combinar los dos árboles que se unen.

Para combinar los árboles nos bastará con agregar el de menor tamaño al de mayor tamaño, de esta manera solo tendremos que modificar los valores del primero, logrando una complejidad de $O(n \log n)$ (Para que un elemento sea procesado, si está en un árbol de tamaño S, la siguiente vez deberá estar en un árbol de tamaño al menos $2S$, así que es procesado a lo mucho $O(\log n)$). Si calculamos en un Sparse Table los ancestros que están a una distancia igual a 2^k para algún $k \geq 0$ podremos consultar el LCA de dos islas en $O(\log n)$ y con esto cubrimos los dos tipos de operación disponibles.

Grupo 2 (Complejidad Esperada: $O(q \log n + n\alpha(n))$)

En primera instancia debemos notar que si conocemos todas las consultas podremos saber qué puentes se van a construir y cuáles no, así que podríamos construir el grafo correspondiente y ejecutar las consultas en orden pero con un grafo de referencia fijo.

Ese preprocesamiento se puede realizar con DSU en $O(q + n \log n)$ u $O(q + n\alpha(n))$, donde α es la función inversa de Ackermann.

Entonces, sobre el grafo construido podemos calcular los DFS Timestamps (ya que si no es un árbol, es un bosque), profundidad y la suma de valores desde la raíz del componente hasta cada nodo. También construiremos un Sparse Table para poder calcular el LCA entre un par de nodos en $O(\log n)$.

Ahora podemos usar un DSU nuevo para ejecutar las consultas en orden:

- `1 A B`: Verificamos en el DSU si A y B están conectados o no, en caso contrario imprimimos 1 y los unimos, si ya están conectados solo imprimimos 0.
- `2 A X`: A todos los nodos en el subárbol del nodo A se les debe agregar $X - a_A$.
- `3 A B`: Si ambos nodos están unidos, podemos calcular el LCA entre ambos nodos y la respuesta sería $p_A + p_B - 2p_L + a_L$, donde L es el LCA y p_X es la suma de valores desde la raíz de la componente hasta X. En caso contraro, la respuesta es `-1`.

Podemos usar un Fenwick tree o Segment tree para sumar x a un subárbol, ya que si usamos los DFS timestamps esto será equivalente a sumarle un valor a un rango y para obtener p_X es consultar el valor de sola posición. Esto se puede lograr con cualquiera de las dos estructuras en $O(\log n)$.

Tutorial del problema: "Zapatos"

Autor(es): Racsó Galván

Asumiremos para cada subtarea que ambos arreglos L y R están ordenados y que $n \leq m$ (pues de lo contrario nos basta con intercambiar los valores).

Grupo 1 (Complejidad Esperada: $O(n \log n + m \log m)$ por caso)

Ya que $n = m$, la forma óptima de asignar los pares es el i-ésimo de cada arreglo y así consecutivamente.

Grupo 1 (Complejidad Esperada: $O(n \log n + m \log m + nm)$ por caso)

Podemos plantear la siguiente función:

$$DP(i,j) = \begin{matrix}\text{Mínimo desbalance de asignar los primeros } i \\ \text{elementos de } L \text{ con un subconjunto de los primeros } j \text{ elementos de } R\end{matrix}$$

Es natural pensar que $DP(i,j) = \infty$ para todo $i > j$ ya que es imposible realizar la asignación. Además se puede deducir que $DP(0,0) = 0$.

Ahora debemos plantear el paso recursivo, que consiste en evaluar las dos posibles situaciones:

- Ignoramos el elemento j de R: Esto es equivalente a tomar el resultado $DP(i, j-1)$.
- Asignamos el elemento i de L al elemento j de R: Esto es igual a tomar el valor $\max\{|L_i - R_j|, DP(i-1, j-1)\}$.

Obviamente consideraremos el menor de ambos valores, así que podremos resolver el problema en $O(nm)$.

Grupo 3 (Complejidad Esperada: $O(n \log n + m \log m + (n+m) \log MAX)$ por caso)

Podemos notar que si un desbalance D permite hacer una asignación correcta, entonces $D+1$ también lo permite, así que usaremos búsqueda binaria sobre el desbalance óptimo D.

Para verificar si se puede realizar una asignación válida, nos basta con analizar cada elemento L_i en orden y asignarlo al mínimo elemento de R que todavía no esté asignado y pertenezca al rango $[L_i - D, L_i + D]$. Se puede demostrar que esta validación siempre es correcta.

Tutorial del problema: "Consultas de ARN"

Autor(es): Racsó Galván

Grupo 1 (Complejidad Esperada: $O\left(n \cdot \left(\sum_{i=1}^{m} |P_i| + |Q_i|\right) + \sum_{i=1}^{n} |s_i| \sum\right)$)

Para resolver este grupo nos bastaba con interar sobre cada una de las cadenas de ARN disponibles y verificar que su prefijo coincida con P_i y su sufijo coincida con Q_i usando la función `substr` o un `for`.

Grupo 2 (Complejidad Esperada: $O\left(n \cdot m + \sum_{i=1}^{m} |P_i| + |Q_i| + \sum_{i=1}^{n} |s_i|\right)$)

Para mejorar la complejidad respecto al primer grupo, deberemos usar hashing y así poder comparar en $O(1)$ el prefijo y el sufijo de cada cadena de ARN disponible.

Grupo 3 (Complejidad Esperada: $O\left(\sqrt{\sum_{i=1}^{m} |P_i|} \cdot \sum_{i=1}^{n} |s_i| + \sum_{i=1}^{m} |Q_i|\right)$)

Para resolver este grupo debemos aprovechar que hay a lo mucho $O\left(\sqrt{\sum_{i=1}^{m} |P_i|}\right)$ longitudes diferentes para los P_i, lo que quiere decir que cada cadena s_i deberá estar asociada con esa cantidad de consultas diferentes.

La observación anterior nos da la idea de que si insertamos la cadena s_i una vez por cada patron P_i diferente con el que coinc, la complejidad total podrá pasar en tiempo. Ahora la pregunta es cómo lograrlo, la respuesta es sencilla: Podemos aprovechar los DFS timestamps del trie formado por las cadenas s_i (el timestamp solo aumentará cada vez que encontremos una posición i, no necesariamente por cada nodo del trie), de esta forma un prefijo P_i tendrá asociado un rango continuo de cadenas s_i y así será más sencillo iterar sobre ellas.

Podemos guardar en cada nodo (si existe) asociado al prefijo P_i el número de consulta i, de esta forma podremos visitar solo los nodos que tienen consultas guardadas, insertar todas las cadenas revertidas de s_j dentro de su rango efectivo a un trie y responder a las consultas usando Q_i.

Esta idea puede parecer imprudente, pero ya vimos que la suma de las longitudes de las cadenas de los rangos efectivos no puede exceder a $O\left(\sqrt{\sum_{i=1}^{m}|P_i|}\cdot\sum_{i=1}^{n}|s_i|\right)$, así que es seguro usarla.

Grupo 4 (Complejidad Esperada: $O\left(\sum_{i=1}^{m}(|P_i|+|Q_i|)+\sum_{i=1}^{n}|s_i|\right)$)

Para resolver este grupo nos basta con realizar una sola optimización a la solución al grupo 3, ya que por los límites esa idea debería darnos TLE en este grupo. La optimización consiste en notar que si una consulta (P_j, Q_j) tiene un rango efectivo de $[l, r]$ sobre las cadenas s_i, podemos aprovechar que todo conteo sobre un rango es equivalente a la resta del conteo hasta la posición r con el conteo hasta la posición $(l-1)$, así que podríamos mantener una cola por cada posición del DFS Timestamp del Trie y agregaríamos la consulta j con la orden de restar a la posición $(l-1)$ y con la orden de sumar a la posición r.

Luego de haber asignado las consultas y órdenes al DFS Timestamp, podemos iterar sobre cada posición usando un Trie en el cual insertaremos las cadenas s_i revertidas asociadas a las posiciones de los timestamps y realizaremos las órdenes. Ya que todo lo manejaremos con tries y DFS, la complejidad se vuelve lineal respecto a la suma de todas las cadenas s_i, P_j y Q_j.

Tutorial del problema: "Zona de Internet"

Autor(es): Racsó Galván

Grupo 1 (Complejidad Esperada: $O(n^4)$)

Para resolver este grupo nos basta notar que la solución óptima siempre cumple alguna de las siguientes dos condiciones:

- Es la circunferencia circunscrita al triángulo formado por 3 puntos diferentes.
- Es una circunferencia tal que hay 2 puntos diferentes que forman una diametro en la misma.

Entonces podemos probar todas las ternas en $O(n^3)$ y todos los pares en $O(n^2)$, luego de algo de análisis de geometría analítica se puede obtener el centro y radio de la circunferencia asociada y verificar si la cantidad de puntos dentro de la misma es mayor o igual a k en $O(n)$, dándonos una complejidad de $O(n^4)$.

Grupo 2 (Complejidad Esperada: $O(n^2 \log n \log(R \cdot \varepsilon^{-1})), \varepsilon = 10^{-4}$)

Para resolver este grupo nos basta notar que una característica común de las dos condiciones mencionadas en el grupo 1 es que hay un punto que está en el borde de la circunferencia; sin embargo, esto no es suficiente para poder identificar una solución candidata, así que deberemos fijar el tamaño del radio también. Podremos usar búsqueda binaria sobre el radio óptimo y luego obtener el centro óptimo a partir del mismo.

Suponiendo que hemos fijado el punto (x_i, y_i) y el radio R, deberemos diseñar un algoritmo para saber si existe alguna circunferencia con el punto (x_i, y_i) en su borde y con radio R en un tiempo prudente.

Observación: Las circunferencias posibles solo pueden rotar respecto a (x_i, y_i), así que sus centros forman una circunferencia de radio R con centro en (x_i, y_i), además esta naturaleza hace que los demás puntos (si es posible) se encuentren dentro de la circunferencia solo en un rango de ángulos $[l, r]$.

Si tomamos como referencia los ejes X y Y para obtener los ángulos efectivos para cada punto (x_j, y_j), debemos considerar dos ángulos clave:

- El ángulo formado por el eje X y el vector que va del punto (x_i, y_i) al punto (x_j, y_j). Notemos que este es el ángulo en el que estaría la cuerda formada por estos puntos. Llamaremos a este ángulo α_1.
- El ángulo formado por la cuerda y el centro de la circunferencia; sin embargo, no tenemos ni idea de donde pueda estar el centro. A pesar de lo anterior, dicho ángulo en realidad es $\arccos\left(\frac{\frac{D}{2}}{R}\right)$, donde $D = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Llamaremos a este ángulo α_2.

Si visualizamos gráficamente las circunferencias podremos notar que el rango efectivo para el punto (x_j, y_j) es $[\alpha_1 - \alpha_2, \alpha_1 + \alpha_2]$, los cuales corresponden al hecho de que el centro de la circunferencia esté del lado horario y antihorario de la cuerda, respectivamente.

Nota: Solo los puntos tales que $D \leq 2R$ tienen rango efectivo, el resto no tendrá.

Luego de ordenar los ángulos, podemos considerar que los rangos pueden tener límite inferior negativo, así que eso lo podemos solucionar iterando por 2 vueltas, es decir, agregando el rango $[\alpha_1 - \alpha_2 + 2\pi, \alpha_1 + \alpha_2 + 2\pi]$. Con esto podremos saber si existe alguna circunferencia válida y además calcular su centro mediante la expresión:

$$(x_O, y_O) = (x_i + R\cos(\alpha), y_i + R\sin(\alpha))$$

Donde α es el ángulo en el que se llegó a un acumulado de al menos k puntos.

La complejidad final será de $O(n^2 \log n \log(R \cdot \varepsilon^{-1}))$.