Estructuras de datos básicas

Víctor Racsó Galván Oyola

17 de mayo de 2021

Índice

Fenwick Tree

Concepto

Idea

Todo entero n tiene una cantidad de bits $O(\log_2 n)$, así que a cada posición hay que darle responsabilidad sobre una cantidad de elementos que sea una potencia de 2. La posición $i \geqslant 1$ tendrá la respuesta parcial de los elementos en el rango $[i - LSO(i) + 1, i]$, donde $LSO(i)$ es la máxima potencia de 2 que divide a i. **La estructura original solo maneja posiciones indexadas en 1**, aunque hay versiones que permiten indexación en 0.

Concepto

Idea

Todo entero n tiene una cantidad de bits $O(\log_2 n)$, así que a cada posición hay que darle responsabilidad sobre una cantidad de elementos que sea una potencia de 2. La posición $i \geqslant 1$ tendrá la respuesta parcial de los elementos en el rango $[i - LSO(i) + 1, i]$, donde $LSO(i)$ es la máxima potencia de 2 que divide a i. **La estructura original solo maneja posiciones indexadas en 1**, aunque hay versiones que permiten indexación en 0.

Ventajas

Podemos responder el resultado de cualquier prefijo de elementos ($[1, x]$ para cualquier x) en $O(\log n)$. Necesita solo $n + 1$ de memoria y su constante de complejidad suele ser ligera.

Concepto

Idea

Todo entero n tiene una cantidad de bits $O(\log_2 n)$, así que a cada posición hay que darle responsabilidad sobre una cantidad de elementos que sea una potencia de 2. La posición $i \geqslant 1$ tendrá la respuesta parcial de los elementos en el rango $[i - LSO(i) + 1, i]$, donde $LSO(i)$ es la máxima potencia de 2 que divide a i. **La estructura original solo maneja posiciones indexadas en 1**, aunque hay versiones que permiten indexación en 0.

Ventajas

Podemos responder el resultado de cualquier prefijo de elementos ($[1, x]$ para cualquier x) en $O(\log n)$. Necesita solo $n + 1$ de memoria y su constante de complejidad suele ser ligera.

Desventajas

Si la función con la que unimos los resultados parciales no tiene inversa, no podremos consultar cualquier rango $[l, r]$ con $l > 1$.

Consulta y actualización

Bajo la condición de que la posición i del Fenwick Tree debe cubrir las posiciones $[i - LSO(i) + 1, i]$, tenemos que analizar las dos operaciones básicas:

Consulta y actualización

Bajo la condición de que la posición i del Fenwick Tree debe cubrir las posiciones $[i - LSO(i) + 1, i]$, tenemos que analizar las dos operaciones básicas:

Consulta

¿Cómo consultar el prefijo $[1, pos]$ bajo esta estructura?

Consulta y actualización

Bajo la condición de que la posición i del Fenwick Tree debe cubrir las posiciones $[i - LSO(i) + 1, i]$, tenemos que analizar las dos operaciones básicas:

Consulta

¿Cómo consultar el prefijo $[1, pos]$ bajo esta estructura?

Respuesta

Ya que la posición pos cubre $[pos - LSO(pos) + 1, pos]$, podemos quitarle su LSO y nos iremos a la siguiente posición que falta cubrir: $pos - LSO(pos)$, si repetimos esto hasta que pos se vuelva 0 entonces obtendremos todo el rango $[1, pos]$

Consulta y actualización

Bajo la condición de que la posición i del Fenwick Tree debe cubrir las posiciones $[i - LSO(i) + 1, i]$, tenemos que analizar las dos operaciones básicas:

Consulta

¿Cómo consultar el prefijo $[1, pos]$ bajo esta estructura?

Respuesta

Ya que la posición pos cubre $[pos - LSO(pos) + 1, pos]$, podemos quitarle su LSO y nos iremos a la siguiente posición que falta cubrir: $pos - LSO(pos)$, si repetimos esto hasta que pos se vuelva 0 entonces obtendremos todo el rango $[1, pos]$

Actualización

¿Cómo modificamos la posición pos con el valor x bajo esta estructura?

Consulta y actualización

Bajo la condición de que la posición i del Fenwick Tree debe cubrir las posiciones $[i - LSO(i) + 1, i]$, tenemos que analizar las dos operaciones básicas:

Consulta

¿Cómo consultar el prefijo $[1, pos]$ bajo esta estructura?

Respuesta

Ya que la posición pos cubre $[pos - LSO(pos) + 1, pos]$, podemos quitarle su LSO y nos iremos a la siguiente posición que falta cubrir: $pos - LSO(pos)$, si repetimos esto hasta que pos se vuelva 0 entonces obtendremos todo el rango $[1, pos]$

Actualización

¿Cómo modificamos la posición pos con el valor x bajo esta estructura?

Respuesta

Ya que necesitamos la siguiente posición x tal que $pos \in [x - LSO(x) + 1, x]$, podemos irnos a $pos + LSO(pos)$ y se puede demostrar que esta es la menor posición mayor que pos tal que cumpla con la condición.

Problema de muestra

Automatic Banking

Dadas las dimensiones (a_i, b_i) de s ranuras, calcular en qué ranura caerán c monedas, de las cuales sabemos sus dimensiones (u_i, v_i). Una moneda i cae en la ranura j de menor índice tal que $a_j \geqslant u_i$ y $b_j \leqslant v_i$. Hallar la suma de los índices por los que caen las monedas.

Problema de muestra

Automatic Banking

Dadas las dimensiones (a_i, b_i) de s ranuras, calcular en qué ranura caerán c monedas, de las cuales sabemos sus dimensiones (u_i, v_i). Una moneda i cae en la ranura j de menor índice tal que $a_j \geqslant u_i$ y $b_j \leqslant v_i$. Hallar la suma de los índices por los que caen las monedas.

Solución I

Para cada moneda, hacemos una búsqueda lineal y cortamos apenas encontremos el índice que deseamos.
Complejidad: $O(sc)$. Veredicto: TLE.

Problema de muestra

Automatic Banking

Dadas las dimensiones (a_i, b_i) de s ranuras, calcular en qué ranura caerán c monedas, de las cuales sabemos sus dimensiones (u_i, v_i). Una moneda i cae en la ranura j de menor índice tal que $a_j \geqslant u_i$ y $b_j \leqslant v_i$. Hallar la suma de los índices por los que caen las monedas.

Solución I

Para cada moneda, hacemos una búsqueda lineal y cortamos apenas encontremos el índice que deseamos.
Complejidad: $O(sc)$. Veredicto: TLE.

Pista I

La solución no tiene porqué ser *online*, es decir, no tenemos por qué responder a cada moneda cuando se nos da.

Problema de muestra

Automatic Banking

Dadas las dimensiones (a_i, b_i) de s ranuras, calcular en qué ranura caerán c monedas, de las cuales sabemos sus dimensiones (u_i, v_i). Una moneda i cae en la ranura j de menor índice tal que $a_j \geqslant u_i$ y $b_j \leqslant v_i$. Hallar la suma de los índices por los que caen las monedas.

Solución I

Para cada moneda, hacemos una búsqueda lineal y cortamos apenas encontremos el índice que deseamos.
Complejidad: $O(sc)$. Veredicto: TLE.

Pista I

La solución no tiene porqué ser *online*, es decir, no tenemos por qué responder a cada moneda cuando se nos da.

Pista II

¿Qué tal si nos aseguramos de que, para cada moneda j, hemos procesado todas las ranuras con $a_i \geqslant u_j$, cómo aprovecharíamos eso?

Problema muestra - II

Solución

- Ordenaremos las ranuras y monedas por su primera compontente de manera no ascendente.
- Luego, procesaremos con *two pointers* todas las ranuras aún no procesadas con primera compontente mayor o igual a la de nuestra moneda.
- Al procesar una ranura i, actualizaremos un Fenwick Tree en la posición b_i con el valor i. Nuestro Fenwick Tree mantendrá el mínimo en un rango.
- Al haber procesado todas las ranuras con $a_i \geqslant u_j$ para nuestra moneda j, debemos tomar la de menor índice con $b_i \leqslant v_j$, así que consultaremos el rango $[1, v_j]$ de la estructura y ese será el índice de la moneda.

Complejidad: $O(s \log s + c \log c + (s + c) \log MAX)$. Veredicto: AC

Segment Tree

Concepto

Idea

¡Volvamos el algoritmo *Merge Sort* una estructura de datos!

Concepto

Idea

¡Volvamos el algoritmo *Merge Sort* una estructura de datos!

Ventajas

Es una estructura muy flexible, permite obtener respuestas parciales de cualquier rango $[l, r]$ en tiempo logarítmico, así como modificación de una posición o un rango en la misma complejidad.

Concepto

Idea

¡Volvamos el algoritmo *Merge Sort* una estructura de datos!

Ventajas

Es una estructura muy flexible, permite obtener respuestas parciales de cualquier rango $[l, r]$ en tiempo logarítmico, así como modificación de una posición o un rango en la misma complejidad.

Desventajas

La constante de la complejidad puede ser alta, además necesita $4n$ de memoria o $2n$ dependiendo de la implementación.

Problema de muestra

Sereja and Brackets

Dada una cadena de paréntesis s, se nos darán m consultas de la forma $[l, r]$. Se nos pide responder a cada consulta con la subsecuencia balanceada de máxima longitud que se puede obtener de $s[l, r]$.

Problema de muestra

Sereja and Brackets

Dada una cadena de paréntesis s, se nos darán m consultas de la forma $[l, r]$. Se nos pide responder a cada consulta con la subsecuencia balanceada de máxima longitud que se puede obtener de $s[l, r]$.

Solución I

Verificar en tiempo lineal todo el rango, agregando todos los paréntesis abiertos que se puedan y cerrando cuando se pueda.
Complejidad: $O(nq)$.

Problema de muestra

Sereja and Brackets

Dada una cadena de paréntesis s, se nos darán m consultas de la forma $[l, r]$. Se nos pide responder a cada consulta con la subsecuencia balanceada de máxima longitud que se puede obtener de $s[l, r]$.

Solución I

Verificar en tiempo lineal todo el rango, agregando todos los paréntesis abiertos que se puedan y cerrando cuando se pueda.
Complejidad: $O(nq)$.

Pista I

¿Nos importa algo además de la cantidad de paréntesis abiertos que aún no se cierran y la cantidad de paréntesis cerrados que tenemos pendientes de abrir por la izquierda?

Problema de muestra - II

Solución

- Consideraremos cada nodo de nuestro segment tree como una tupla (a, c, r) que nos representa a una cadena en la cual ya tenemos una subsecuencia de longitud r que está balanceada y que los caracteres que no usamos de dicha cadena tienen a paréntesis abiertos sin cerrar y c paréntesis cerrados sin abrir.
- Al unir dos nodos i y j (las posiciones del nodo i están a la izquierda de las de j), debemos hacer las siguientes asignaciones para el nuevo nodo:

$$r' = r_i + r_j + 2 \cdot \text{mín}\{a_i, c_j\}$$
$$a' = a_i + a_j - \text{mín}\{a_i, c_j\}$$
$$c' = c_i + c_j - \text{mín}\{a_i, c_j\}$$

  Esto porque vamos a emparejar los paréntesis abiertos de i con los cerrados de j para aumentar la respuesta, y obviamente estos dejarán de aportar al a y c del nuevo nodo.

Complejidad: $O(n + m \log n)$.

Disjoint Set Union

Concepto

Idea

Podemos unir conjuntos usando “Representantes” de cada conjunto para mejorar la eficiencia.

Concepto

Idea

Podemos unir conjuntos usando “Representantes” de cada conjunto para mejorar la eficiencia.

Ventajas

La complejidad es casi lineal en el mejor de los casos. Es muy compatible con relaciones de equivalencia y representantes de clase.

Concepto

Idea

Podemos unir conjuntos usando “Representantes” de cada conjunto para mejorar la eficiencia.

Ventajas

La complejidad es casi lineal en el mejor de los casos. Es muy compatible con relaciones de equivalencia y representantes de clase.

Desventajas

Solo se puede revertir las uniones una a la vez desde la más reciente (no permite revertir antes sin transformarse a persistente).

Intentemos un problema en Codechef - I

Fill The Matrix

Se tienen q valores a_{ij} de una matriz de $n \times n$, se dice que una matriz es *buena* si existe un arreglo A tal que $B_{ij} = |A_i - A_j|$. Determinar si existe alguna forma de que la matriz dada parcialmente sea *buena*.

Intentemos un problema en Codechef - I

Fill The Matrix

Se tienen q valores a_{ij} de una matriz de $n \times n$, se dice que una matriz es *buena* si existe un arreglo A tal que $B_{ij} = |A_i - A_j|$. Determinar si existe alguna forma de que la matriz dada parcialmente sea *buena*.

Solución I

Podemos probar todas las posibles matrices que correspondan con la información dada si esta es consistente.
Complejidad $O(n^2 2^{n^2-q})$.

Intentemos un problema en Codechef - I

Fill The Matrix

Se tienen q valores a_{ij} de una matriz de $n \times n$, se dice que una matriz es *buena* si existe un arreglo A tal que $B_{ij} = |A_i - A_j|$. Determinar si existe alguna forma de que la matriz dada parcialmente sea *buena*.

Solución I

Podemos probar todas las posibles matrices que correspondan con la información dada si esta es consistente.
Complejidad $O(n^2 2^{n^2-q})$.

Observación I

Siempre se puede tomar un arreglo A tal que $A_i \in \{0, 1\}$ para todo $i = 1, \ldots, n$.

Intentemos un problema en Codechef - I

Fill The Matrix

Se tienen q valores a_{ij} de una matriz de $n \times n$, se dice que una matriz es *buena* si existe un arreglo A tal que $B_{ij} = |A_i - A_j|$. Determinar si existe alguna forma de que la matriz dada parcialmente sea *buena*.

Solución I

Podemos probar todas las posibles matrices que correspondan con la información dada si esta es consistente.
Complejidad $O(n^2 2^{n^2-q})$.

Observación I

Siempre se puede tomar un arreglo A tal que $A_i \in \{0, 1\}$ para todo $i = 1, \ldots, n$.

Observación II

Nos basta con que los datos actuales coincidan con una matriz buena para tener la posibilidad de rellenar el resto de posiciones para armar una matriz buena.

Intentemos un problema en Codechef - II

Conclusión

Nos basta que $A_i \neq 1 - A_i$ para todo A_i, lo cual es completamente lógico, pero según las relaciones propuestas por B_{ij}, podríamos llegar a esa igualdad siguiendo alguna secuencia de relaciones, así que debemos verificar correctamente si se da o no dicha situación.

Intentemos un problema en Codechef - II

Conclusión

Nos basta que $A_i \neq 1 - A_i$ para todo A_i, lo cual es completamente lógico, pero según las relaciones propuestas por B_{ij}, podríamos llegar a esa igualdad siguiendo alguna secuencia de relaciones, así que debemos verificar correctamente si se da o no dicha situación.

Solución II

Podemos mantener, por cada nodo i, dos copias de este, una que sea el valor A_i y otra que sea el valor $1 - A_i$, de manera que cuando tengamos los valores de B_{ij} procederemos como corresponda:

- $B_{ij} = 0$, entonces $A_i = A_j$ y además $1 - A_i = 1 - A_j$.
- $B_{ij} = 0$, entonces $A_i = 1 - A_j$ y además $1 - A_i = A_j$.

Ahora, los conjuntos que mantendremos en el DSU serán los valores que deben ser iguales dados los B_{ij}, así que:

- $B_{ij} = 0$, entonces unimos (A_i, A_j) y además $(1 - A_i, 1 - A_j)$.
- $B_{ij} = 0$, entonces unimos $(A_i, 1 - A_j)$ y además $(1 - A_i, A_j)$.