Estructuras de Datos Avanzadas

Adriana Ramirez Vigueras
Fhernanda Montserrat Romo Olea
Marco Antonio Velasco Flores

Tarea 1

Natalia Abigail Pérez Romero

Facultad de Ciencias

Semestre 2024-1

Entrega: 16 Octubre 2023 - 11:59 PM

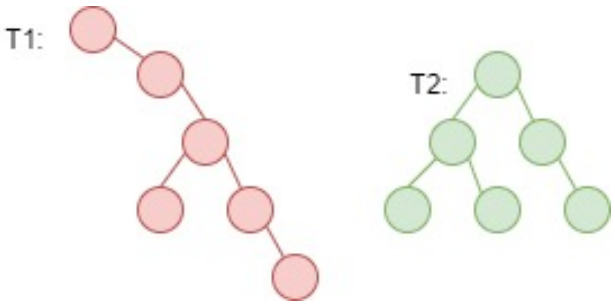
Pregunta 1

Decimos que un árbol binario de búsqueda T_1 puede ser RIGHT-CONVERTED a un árbol binario de búsqueda T_2 si es posible obtener T_2 de T_1 por a tráves de una serie de llamadas a la operación RIGHT-ROTATE.

- Da un ejemplo de dos árboles T_1 y T_2 tal que T_1 no pueda ser RIGHT-CONVERTED en T_2.
- Demuestra que si un árbol T_1 puede ser RIGHT-CONVERTED a T_2, entonces T_1 puede ser RIGHT-CONVERTED usando $O(n^2)$ operaciones RIGHT-ROTATE.

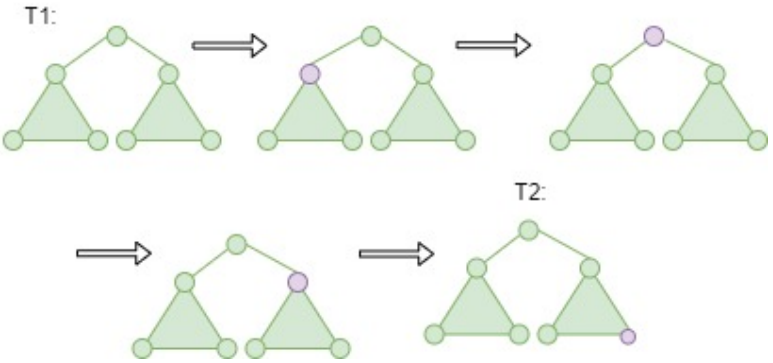
Respuesta

- Da un ejemplo de dos árboles T_1 y T_2 tal que T_1 no pueda ser RIGHT-CONVERTED en T_2. T_1 es un árbol inclinado hacia la derecha y T_2 es un árbol balanceado de forma que la única forma de transformar



- Demuestra que si un árbol T_1 puede ser RIGHT-CONVERTED a T_2, entonces T_1 puede ser RIGHT-CONVERTED usando $O(n^2)$ operaciones RIGHT-ROTATE.

  Sea u un vértice en T_1 el cual tiene un vértice u' el cual tiene un vértice u' en T_2 de forma que despues de un número de operaciones RIGHT-ROTATE u se encuentra en la posición de u', esto le tomara en el peor de los casos n (el número de vértices) por que ocupará el lugar de todos los vértices en T_1 antes de llegar a la posición de u'. En la figura siguiente podemos observar un ejemplo del movimiento.



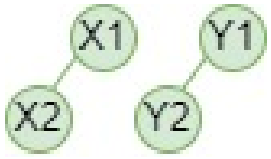
Lo anterior es verdad para todos los n vértices por ende si un árbol T_1 puede ser RIGHT-CONVERTED a T_2, entonces T_1 puede ser RIGHT-CONVERTED usando $O(n^2)$ operaciones RIGHT-ROTATE.

Pregunta 2

Muestra que dado un conjunto T de n nodos $x_1, x_2, \ldots, x_n$ con valores y prioridades distintas, el árbol treap asociado a T es único. Hint: utiliza inducción sobre n.

Respuesta

Demostración por contradicción: Suponer que para T tenemos más de un árbol treap asociado. Entonces G y G' son árboles asociados a T, entonces $V(G) = V(G') = T = \{x_1, x_2, \ldots, x_n\}$, luego tenemos que $E(G) = E(G')$ dado que si $x_2 \prec x_1$,es decir que x_2 sea hijo de x_1, entonces $y_2 \prec y_1$, es decir y_2 es hijo de y_1. En el siguiente árbol podemos observar lo anterior ademas de que $x_2 < x_1$ y por tanto $y_2 < y_1$:



Por lo que $e(x_1, x_2) = e(y_1, y_2)$ para todos los vértices en $G = G'$. Entonces $G = G'$ lo cual es una contradicción al suponer que el árbol treap asociado a T no es único, por tanto si es único.

Pregunta 3

Se pueden utilizar las estructuras de búsqueda de rangos ortogonales para determinar si un punto particular (a, b) está en un conjunto dado, haciendo una consulta al rango $[a : a] \times [b : b]$.

1. Prueba que hacer una consulta así en un árbol KD toma tiempo $O(\log n)$.
2. ¿Cuál es la complejidad para una consulta así en un árbol de rangos?

Respuesta

1. Prueba que hacer una consulta así en un árbol KD toma tiempo $O(\log n)$.

   El algoritmo de busqueda para el kd-tree es

   1: Si v es una hoja
   2: Regresa si v esta en R
   3: De otra manera:
   4: Si la región $lc(v)$ (del hijo izquierdo de v) está contenida en R entonces: recorrer el subárbol izquierdo de V y regresa los puntos almacenados.
   5: De otra manera:

6: Si la región $lc(v)$ intersecta R entonces: busca en el subarbol $lc(v)$ en R
7: Si la región $rc(v)$ esta contenida en R entonces: recorrer el subarbol derecho de v y regresa los puntos almacenados
8: De otra manera:
9: Si la región $rv(v)$ intersecta R entonces: busca en el subarbol rc en R

De forma que para la consulta del punto (a, b) el algoritmo verifica en que subarbol esta por 4 y 7 y desciende por 5 u 8 sobre el subarbol izquierdo o derecho respectivamente, hasta llegar a una hoja. De manera que recorre la altura del árbol para hallar el punto una vez y como la altura del árbol es $\log n$. Toma $O(\log n)$ hacer una consulta de un punto particular (a, b) al rango $[a : a] \times [b : b]$ en un árbol KD

2. ¿Cuál es la complejidad para una consulta así en un árbol de rangos?

El algoritmo de busqueda para el árboles de rango requiere de dos rutinas:

EncuentraNodoDivisor(τ, x, x'):

1: $v \leftarrow raiz(T)$
2: **while** v no es una hoja y $x' \leq x_v$ o $x > x_v$ **do**
3: **if** $x' \leq x_v$ **then**
4: $v \leftarrow lc(v)$
5: **else**
6: $v \leftarrow rc(v)$
7: **end if**
8: **end while**

Algoritmo2DRangeQuery($\tau, [x : x'] \times [y : y']$)

1: $v_x \leftarrow$ EcuentraNodoDivisor(τ, x, x')
2: **if** v_s es una hoja **then**
3: Verifica si el punto en v_s debe ser reportado.
4: **else**
5: Sigue el camino a x y realiza la busqueda en una 1 dimension en los subarboles derechos del camino.
6: $v \leftarrow lc(v_s)$
7: **while** v no es una hoja **do**
8: **if** $x \leq x_v$ **then**
9: Realiza una busqueda de una dimension sobre $\tau_{assoc}(rc(v)), [y : y']$
10: $v \leftarrow lr(v)$
11: **else**
12: $v \leftarrow rc(v)$
13: **end if**
14: **end while**
15: Verifica si el punto debe ser reportado
16: De forma similar sigue el camino de $rc(v_s)$ a x', haz la busqueda en una dimension sobre $[y : y']$ en las estructuras asociadas de los subarboles deñ camino y verifica si el punto almacenado en la hoja donde el camino termina debe ser reportado.
17: **end if**

Entonces en la consulta de un punto (a, b) en $[a : a] \times [b : b]$ por 1 busca el nodo divisor que

contiene a a lo que toma $O(\log n)$ (la altura del árbol principal), si el nodo que contiene a es una hoja entonces lo regresa. Si no es una hoja entonces realiza una busqueda sobre τ_{assoc} (estructura asociada) en el rango $[y : y']$ lo que toma $O(k)$ donde $k = 1$ porque en el rango solo hay un punto.

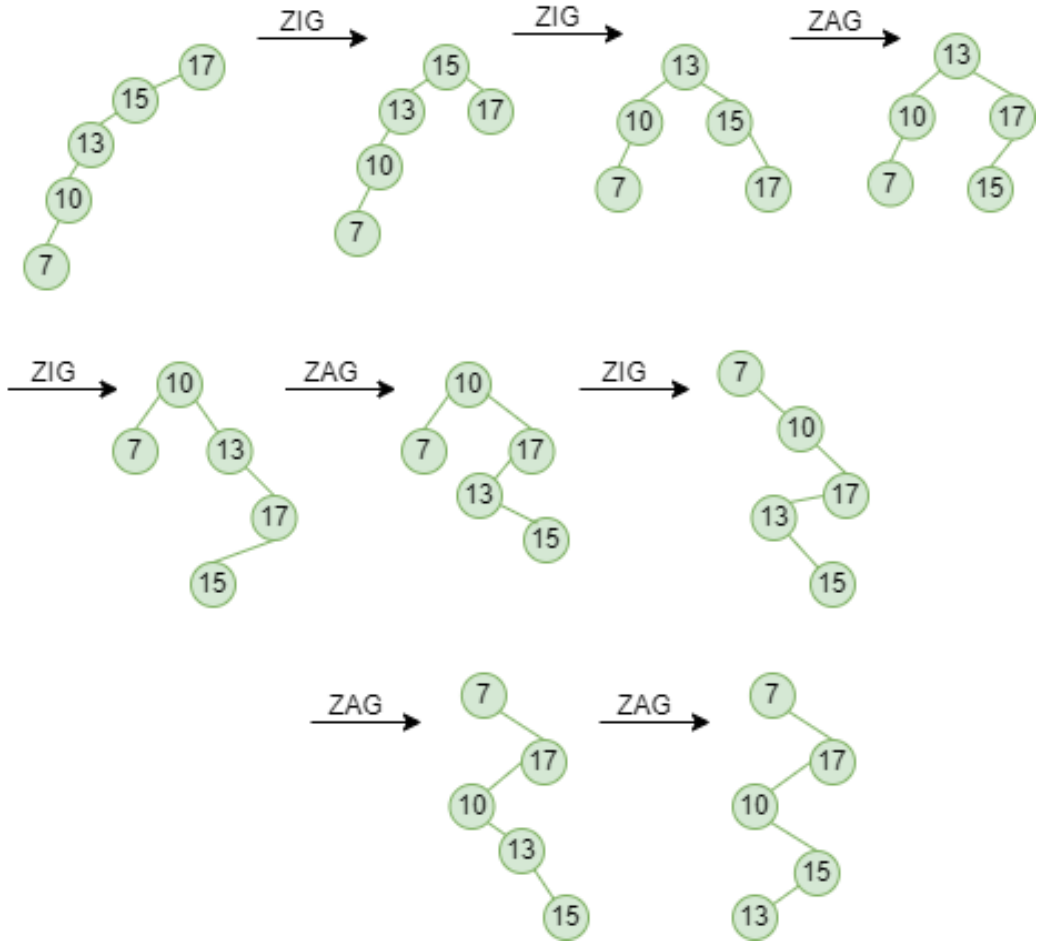
Por tanto la consulta de un punto (a, b) en $[a : a] \times [b : b]$ es $O(\log n + 1) \to O(\log n)$

Pregunta 4

Describe una secuencia de accesos a un árbol splay T de n nodos, con $n \geq 5$ impar, que resulte en T siendo una sola cadena de nodos en la que el camino para bajar en el árbol alterne entre hijo izquierdo e hijo derecho.

Respuesta

La secuencia es 17,15,13,10,7 y podemos ver los accesos en la siguiente figura



Pregunta 5

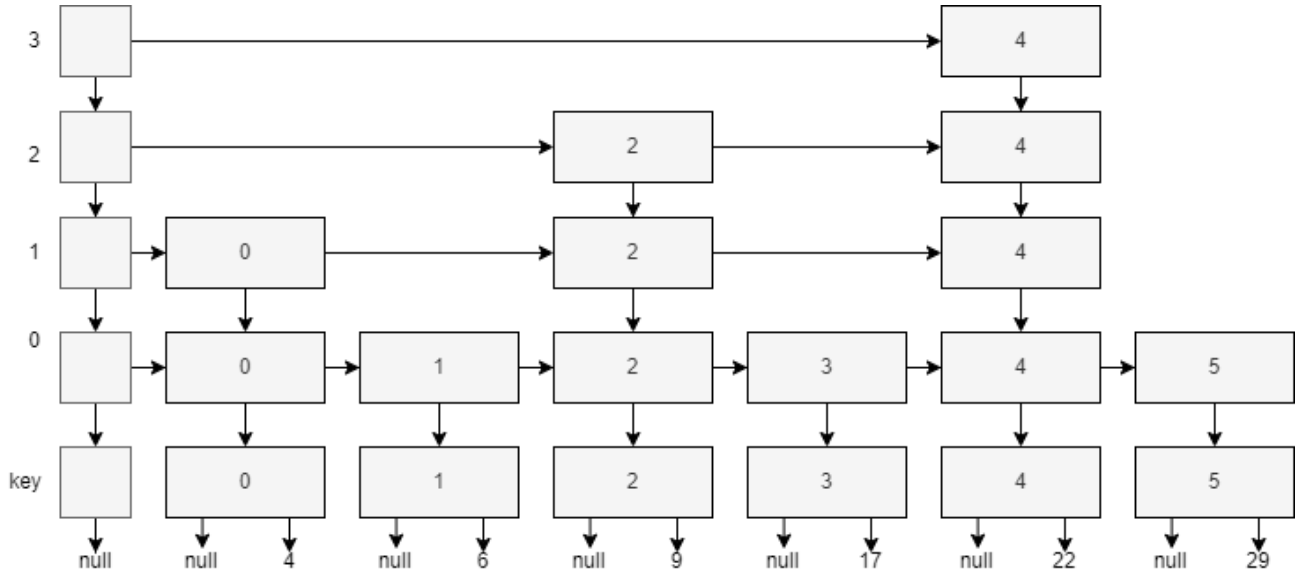
Describe cómo modificar una *skip-list* L para poder realizar las siguientes dos operaciones en tiempo esperado $O(\log n)$:

- Dado un índice i, obtener el elemento de L en la posición i.
- Dado un valor x, obtener la cantidad de elementos en L menores a x.

Respuesta

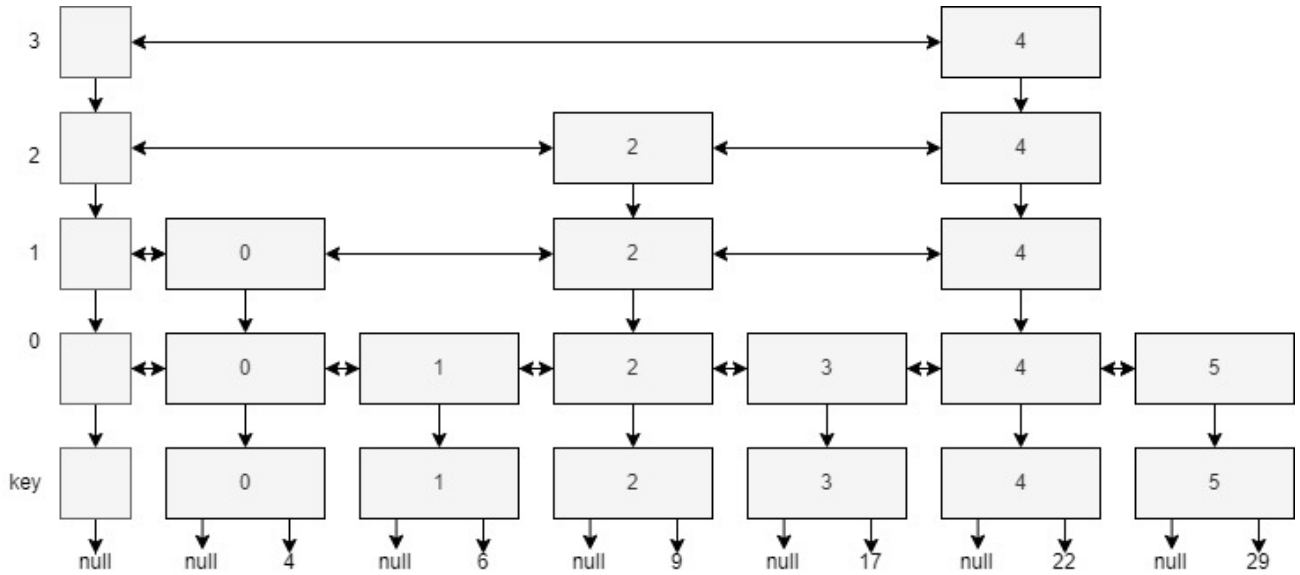
- Dado un índice i, obtener el elemento de L en la posición i. Modificar los nodos de los nodos de la skip-list L al añadir un campo que almacene su índice en la lista original de forma que podremos realizar la busqueda del elemento usando el indice i en lugar de la llave. Para esto será necesario modificar el algoritmo de insercion de forma que tomará en cuenta el indice del nodo anterior y del nodo posterior para reconstruir la skip-list a partir del indice del elemento insertado, en el peor de los casos será necesario modificar el índice de todos los elementos de la lista.

  En la siguiente imagen podemos observar la modificación:



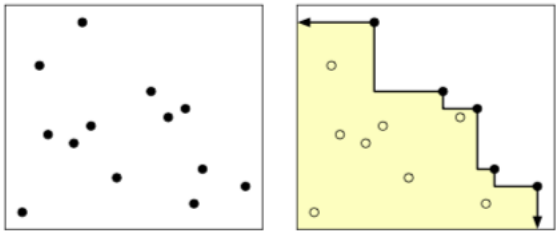
- Dado un valor x, obtener la cantidad de elementos en L menores a x. Modificar los nodos para almancenar el nodo anterior (atributo `prev`), de forma que sabremos que el nodo actual es el último por que el apuntador de siguiente apunta a null, y podemos identificar al primero porque su apuntador de `prev` apunta al placeholder (la columa de nodos que nos indica el nivel) si el `key,next,down,prev=null`

  De manera que podemos realizar la búsqueda del valor x, luego desplazarse al anterior por el atributo `prev` aumentando un contador hasta encontrar el primer nodo y regresar el contador.

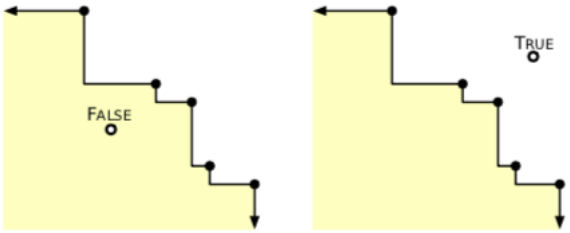


Pregunta 6

Let P be a set of n points in the plane. The staircase of P is the set of all points in the plane that have at least one point in P both above and to the right.



1. Describe an algorithm to compute the staircase of a set of n points in $O(n \log n)$ time.
2. Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE? (x, y) that returns TRUE if the point (x, y) is above the staircase, or FALS otherwise. Your data structure should use $O(n)$ space, and your ABOVE? algorithm should run in $O(logn)$ time.



Respuesta

1. Describe an algorithm to compute the staircase of a set of n points in $O(n \log n)$ time.

escalera={}
Ordenar los puntos con respecto a y
Sacar el primer punto $MAX_y = y(p_1)$
Comparar los puntos p_j tal que y de p_j sea igual a y de MAX_y y tomar de los puntos el que tenga la x mayor (este es el punto más a la derecha) asignarlo como MAX_x
for p_i en P **do**
    if $x(p_i) < MAX_x$ **then** //¿Es punto máximo con respecto a x?
        //no, entonces
        Sacar el máximo punto actual $MAX_y = p_i$
        Comparar los puntos p_j tal que y de p_j sea igual a y de MAX_y y tomar de los puntos el que tenga la x mayor (este es el punto más a la derecha) asignarlo como MAX_x
    else $x(p_i) \geq MAX_x$ // sí, es el maximo
        escalera.add($p_i | x = MAX_x \&\& MAX_y$)
    end if
end for

En este algoritmo nos interesa ordenar los elementos con respecto de y (lo que toma $O(n \log n)$) y luego escoger el punto dentro de las y más grandes donde la x sea la mayor (lo que toma $O(n)$). De esta forma escoger "la altura"del escalon usando y y luego "la anchura"del escalon usado x para que todos los puntos del plano tengan un punto arriba (tomamos el punto más arriba es decir el mayor con respecto de y) y a la derecha (tomamos el punto más a la derecha es decir el mayor con respecto de x). Y la complejidad del algoritmo queda $O(n + n \log n) \to O(n \log n)$

2. Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE? (x, y) that returns TRUE if the point (x, y) is above the staircase, or FALS otherwise. Your data structure should use $O(n)$ space, and your ABOVE? algorithm should run in $O(\log n)$ time.

   Podemos utilizar un árbol binario de busqueda y mantenerlo ordenado con respecto a y de forma que si tenemos que desplazarnos en la busqueda hacia arriba o abajo el último nodo accedido se convierte en la raíz lo que acelerara nuestra busqueda. En clase vimos que estos árboles utilizan $O(n)$ en espacio dado que solo es necesario almancenar n nodos.

   Propongo el siguiente algoritmo ABOVE($tree, punto(x, y)$): Notar que si p esta en la escalera entonces $\forall z$ punto en la escalera, se cumple que $x < x(z)$ porque si esto fuera falso implicaría que existe un punto en P que no tiene un punto a la derecha.

```
1: m ← raíz
2: if y == y(m) then //¿El punto esta a la altura de m?
3:     if x ≥ x(m) then //¿El punto esta más a la derecha que m?
4:         return True
5:         //Sí, entonces esta fuera de la escalera
6:     else
7:         return False
8:         //No, entonces esta en la escalera
9:     end if
10: end if
11: if y > y(m) then //¿El punto esta arriba de m?
12:     if ¿m es el mayor elemento (la raiz) en el árbol original? then
13: return True//Esta arriba de la escalera
14:     else
15:         if x ≤ m(x) then //¿El punto esta más a la izquierda que m?
16: return False // Sí,entonces no esta arriba de la escalera
17:         else x > m(x)
18:             // No, entonces busca en el subárbol derecho
19:             ABOVE(rightTree, p(x, y))
20:         end if
21:     end if
22: end if
23: if y < y(m) then //¿El punto esta abajo de m?
24:     if ¿m es una hoja? then
25:
26:         if x ≤ m(x) then //¿El punto esta más a la izquierda que m?
27: return False // Sí,entonces no esta arriba de la escalera
```

```
28:         else x > m(x)//¿El punto esta más a la derecha que m?
29: return True
30:         end if
31:     else
32:         //no, entonces busca en el subárbol izquierdo
33:         ABOVE(leftTree, p(x, y))
34:     end if
35: end if
```

La idea es observar por cada escalon (marcado por una y y x que nos indica la altura y la anchura) y observar si el punto esta fuera de todos los escalones: la y nos indica si esta abajo o arriba del escalon y la x si esta dentro del escalon (a la izquierda) o no (esta a la derecha). Como utiliza una modificación de la búsqueda binaria (buscando en cada subárbol) la complejidad del algoritmo es $O(\log n)$

Pregunta 7

Sea S un conjunto de n segmentos de línea sin cruces entre ellos. Queremos responder rápidamente a consultas del tipo: dado un punto p encontrar al primer segmento en S por el que pasa el rayo vertical con origen en p y dirección hacia arriba. Da una estructura de datos para resolver este problema. Acota el tiempo de consulta y el espacio requerido por tu estructura. ¿Cuál es el tiempo de pre-procesamiento?

Respuesta

<Tu respuesta aquí>

Pregunta 8

En algunas aplicaciones solo nos interesa el número de puntos que caen dentro de un rango y no reportar cada uno de ellos. En este caso nos gustaría evitar el término $O(k)$ en el tiempo de consulta.

1. Describe cómo un árbol de rangos de una dimensión puede adaptarse para que una consulta así se pueda realizar en tiempo $O(\log n)$.

2. Usando la solución al problema para una dimensión, describe cómo se pueden responder consultas de conteo en rangos de d dimensiones en tiempo $O(\log^d n)$.

3. Describe cómo se puede usar la técnica de cascada para mejorar el tiempo de consulta en un factor $O(\log n)$ para dos y más dimensiones.

Respuesta

<Tu respuesta aquí>

Pregunta 9

Diseña e implementa una versión de un Treap que incluya la operación $get(i)$, que regrese la llave con rank i en el Treap. (Hint: Haz que cada nodo, u, mantenga un registro del tamaño del subárbol enraizado en u.

Respuesta

<Tu respuesta aquí>

Pregunta 10

Implementa un TreapList, una implementación de la interfaz lista como un Treap. Cada nodo en el Treap debería almacenar un elemento de la lista. Todas las operaciones de la Lista como $get(i)$, $set(i, x)$, $add(i, x)$ y $remove(i)$ deben tener una complejidad de $O(\log n)$ esperado.

Respuesta

<Tu respuesta aquí>