Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Puebla

Programación de estructuras de datos y algoritmos fundamentales (Gpo 1)

Fecha: 03/11/2021

Profesor Daniel Pérez Rojas

Act 3.3 - Árbol desplegado - Implementando un Splay Tree

Alumnos:

Iker Guerrero González - A00830026

Aldo Mauricio Cruz Lozada - A01732372

Francisco Rocha Juárez - A01730560

Árbol desplegado. Implementación de un Splay Tree. Actividad 3.3

Un árbol binario de búsqueda es una estructura de datos para representar tablas y listas de manera que su acceso, inserción y eliminación sea fácil. En un árbol de distribución de n nodos, todas las operaciones del árbol de búsqueda estándar tienen un límite de tiempo amortizado de O(log n) por operación.

La eficiencia de los árboles esparcidos no proviene de una restricción estructural explícita, como ocurre con los árboles balanceados, sino de la aplicación de una heurística de reestructuración simple, llamada esparcimiento, cada vez que se accede al árbol.

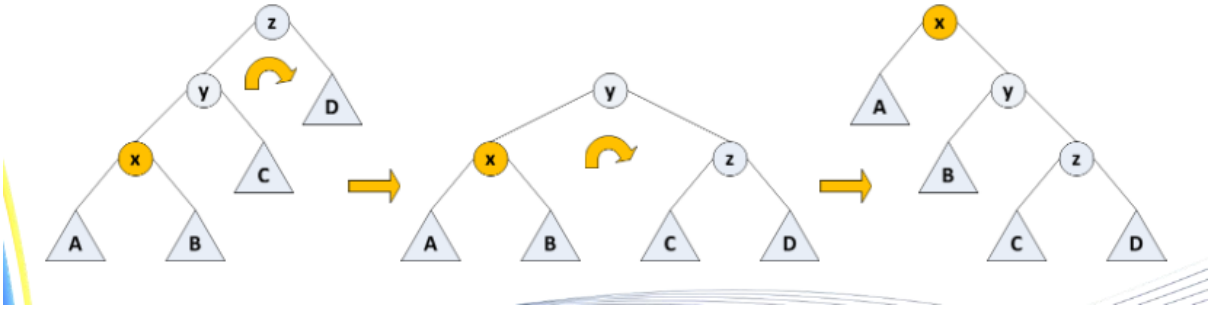
La fuerza impulsora detrás de los “splay trees” es el concepto de que "para que el tiempo de acceso total sea pequeño, los elementos a los que se accede con frecuencia deben estar cerca de la raíz del árbol".

“Splaying” es la operación clave de los árboles de distribución: después de acceder a un nodo, la distribución mueve el nodo a la raíz del árbol. Hay dos métodos principales de distribución: • De abajo hacia arriba • De arriba hacia abajo.

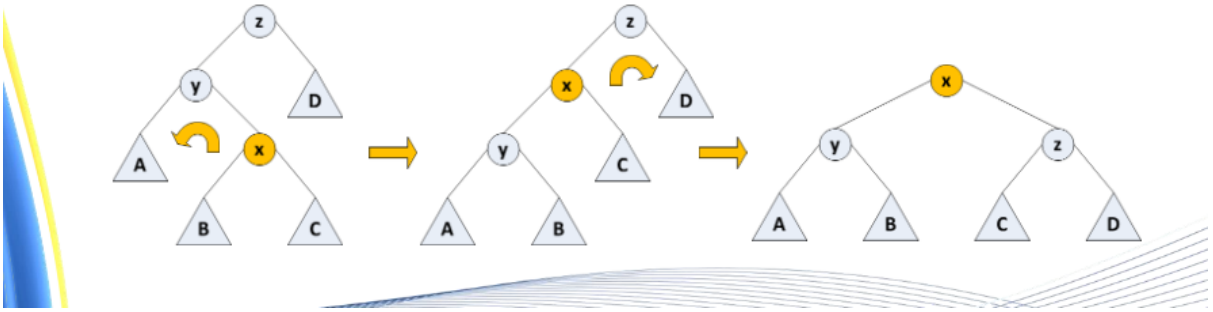
En la distribución de abajo hacia arriba, se accede a un elemento de árbol de distribución al estilo BST: si el elemento existe en el árbol, el árbol se distribuye en ese elemento antes de que se devuelva. Si el elemento no existe, el árbol se distribuye en el último elemento no nulo atravesado.

En el despliegue de abajo hacia arriba, se repiten tres operaciones básicas de despliegue desde el elemento desplegado hasta la raíz: una de las tres operaciones, zig, zig-zig o zig-zag, se elige en función de la configuración del elemento, su padre y abuelo: para describir estas operaciones, sea x el elemento que se muestra, p (x) sea el padre de x y g (x) sea el abuelo de x.

Operación de distribución básica: Zig-Zig: si x y p (x) son ambos hijos izquierdos o derechos y p (x) ≠ raíz 1) Gire el borde que une p (x) yg (x) 2) Gire el borde que une x y p (x).



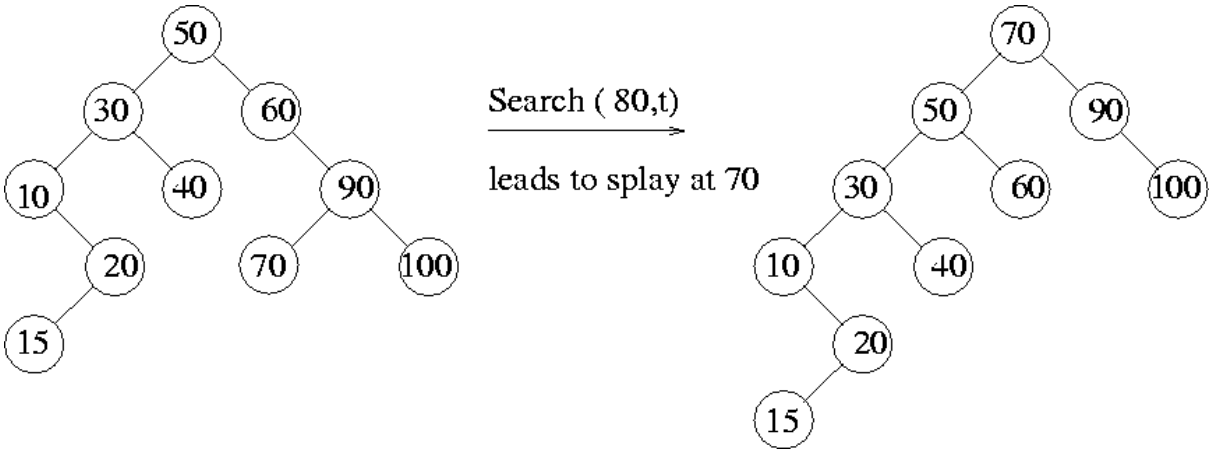
Operación de distribución básica: Zig-Zag: si x es un hijo de la derecha y p (x) es un hijo de la izquierda (o viceversa) yp (x) ≠ root 1) Gire el borde que une x y p (x) 2) Gire el borde que une x (en su nueva posición) y el original g (x).



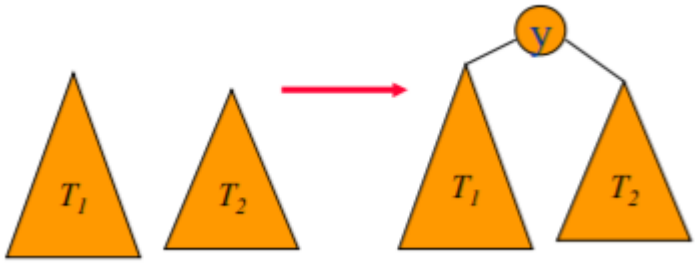
Operación de expansión básica: Zig-Zig-Zig y Zig-Zag mueve x hacia arriba dos bordes a la vez, ¿qué pasa con las profundidades impares? y p (x).

Operación: acceso (i, t) - Entradas: Árbol t y un elemento i para acceder - Salida: Un puntero al nodo que contiene ior nulo si no se encuentra - Algoritmo: 1) Recorrido desde la

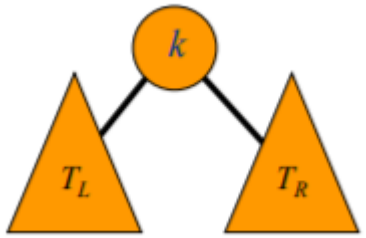
raíz de t yendo a la izquierda si el nodo actual es <i, right if> i, deteniendo if = i, o deteniendo y almacenando el nodo anterior if = null 2) Si el recorrido se detiene en i, se abre en iy devuelve un puntero a la nueva raíz de t 3) De lo contrario, se abre en el nodo anterior no nulo y devuelve nulo.



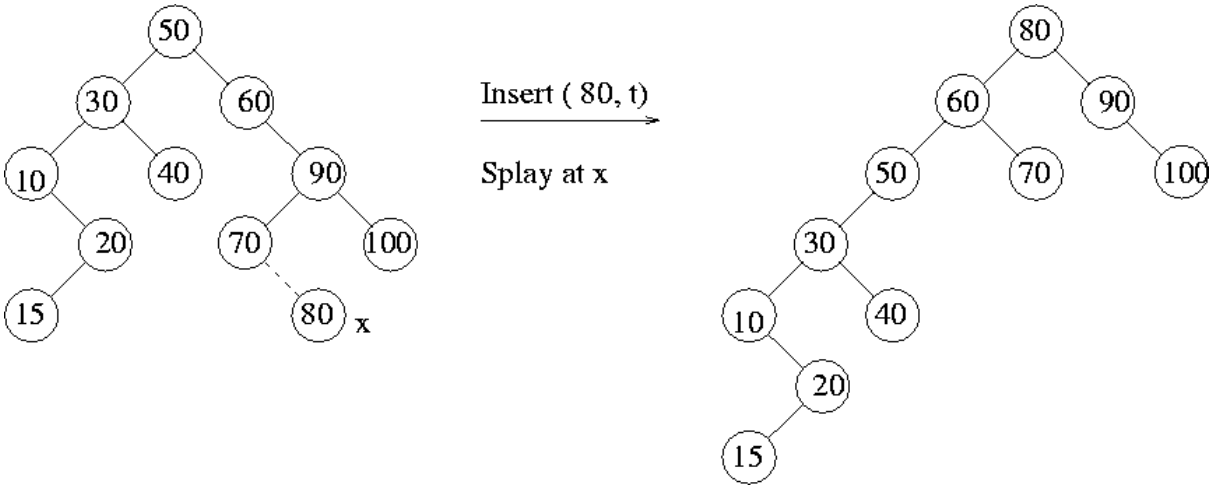
Operación: join (t1, t2) –Entradas: Árboles t1 y t2 donde todos los elementos en t1 son menores que todos los elementos en t2 – Salida: Un solo árbol que combina t1 y t2 – Algoritmo: 1) Acceda al elemento más grande en t1 2) La raíz de t1 es ahora el elemento más grande 3) Apunta el hijo derecho nulo de la raíz de t1 a t2 4) Devuelve t1.



Operación: dividir (i, t) –Entradas: árbol t y un valor i que dividir – Salida: árboles t1 y t2 que contienen los elementos de t <i and> respectivamente – Algoritmo: 1) Realizar acceso (i, t) 2) Si t 's root <i: t2 = right (t), right (t) = null, t1 = t3) Si t's root> i: t1 = left (t), left (t) = null, t2 = t4) Devuelve t1 y t2.

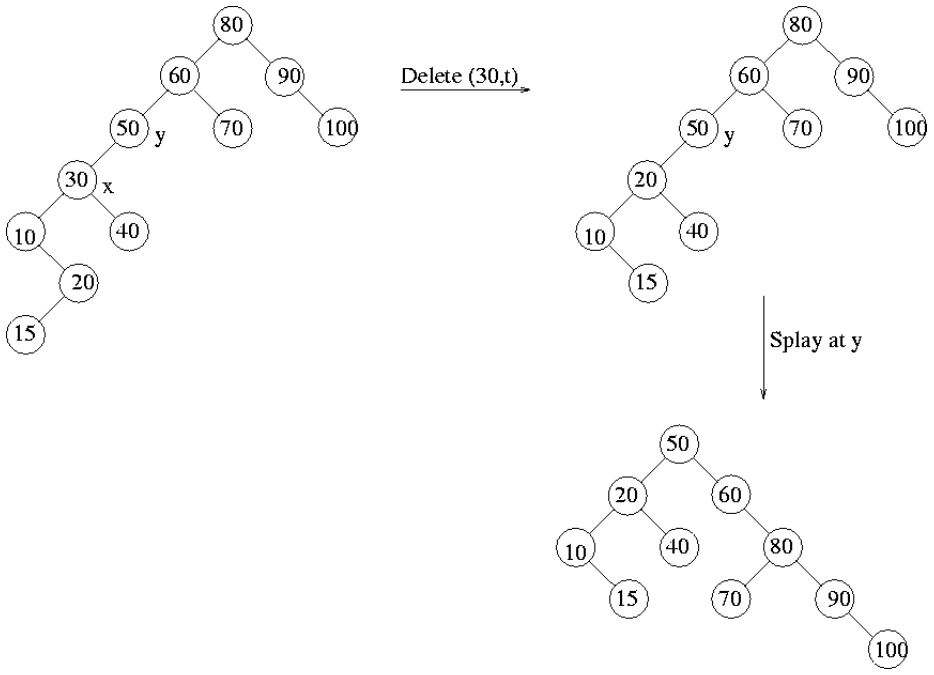


Operación: insertar (i, t) –Entradas: Árbol ty un valor i para insertar – Salida: Árbol t con el elemento i insertado – Algoritmo: 1) Realizar división (i, t) para obtener t1 y t2 2) Crear un nuevo árbol t con i en el elemento raíz 3) Establecer a la izquierda (t) = t1 4) Establecer a la derecha (t) = t2 5) Retornar t.



Operación: eliminar (i, t) –Entradas: Árbol ty un valor i para eliminar – Salida: Árbol t con el elemento i eliminado – Algoritmo: 1) Realizar acceso (i, t) 2) Guardar un puntero al nodo raíz de t 3) Realiza la combinación (izquierda (t), derecha (t)) para obtener la nueva t 4) Libera el antiguo nodo raíz de t 5) Devuelve la nueva t.

Árbol desplegado. Implementación de un Splay Tree. Actividad 3.3



Ventajas:

- Nada mucho peor que las estructuras de datos no ajustables
- Necesita menos espacio ya que no se necesita información de saldo.
- El ajuste a los patrones de uso significa que pueden ser más eficientes para ciertas secuencias.
- La reproducción reduce la profundidad de los nodos en la ruta de acceso aproximadamente a la mitad.

Desventajas:

- Más ajustes.
- Ajustes en los accesos, no solo actualizaciones.
- Las operaciones individuales pueden ser costosas.

Comparación con un BST balanceado:

- El teorema de equilibrio [1] demuestra que el tiempo total de acceso es O [(m + n) log n + m] para m accesos de un árbol de distribución de n elementos.
- El tiempo de acceso total promedio de casos para una BST balanceada es m log n.

- Para secuencias de acceso muy grandes, el tiempo de acceso total del árbol de distribución es del mismo orden de una BST balanceada.

Referencias APA:

Sleator, D. D., & Tarjan, R. E. (1983). Self-adjusting binary trees. Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, 235–245. https://0-doi-org.biblioteca-ils.tec.mx/10.1145/800061.808752**.**