Instituto Tecnológico y de Estudios Superiores de Monterrey

Análisis de algoritmos y estructuras de datos fundamentales

Profesor de la materia:
Guillermo Falcón

Actividad 3.3 – Árbol desplegado: Implementando un Splay Tree

Jorge Del Barco Garza A01284234

Octubre 13 del 2021

Av. Eugenio Garza Sada 2501 Sur, Tecnológico, 64849 Monterrey, N.L.

¿Qué es un Splay Tree?

Una estructura de datos creada por Robert Tarjan y Daniel Sleator en 1985. De acuerdo con información recabada del departamento de informática de la universidad de Cornell, un splay tree, o árbol biselado en español, es simplemente un árbol de búsqueda binario. No obstante, cuenta con una característica especial que recae en que cuando un elemento es buscado y encontrado dentro del árbol, este se vuelve a organizar para que, sin romper las propiedades de un BST, el elemento encontrado quede en la raíz de la estructura de datos. Esto tiene como propósito hacer que el acceso al elemento sea más rápido que la búsqueda anterior, si es que se vuelve a buscar ese dato nuevamente, con una complejidad computacional de O(1). (Cornell University, s.f) Como cualquier otro árbol binario, estas estructuras de datos cuentan con las operaciones básicas tales como: inserción, búsqueda y eliminación.

Para el análisis de complejidad de un árbol biselado se considera la complejidad amortizada. La complejidad amortizada es utilizada para describir la complejidad de un algoritmo cuyo peor caso es alcanzado de vez en cuando, tomando así el caso promedio que ocurre como la complejidad del algoritmo.

Los árboles desplegados (AD) garantizan "que para cualquier secuencia de M operaciones en un árbol, empezando desde un árbol vacío, toma a lo más un tiempo de $O(M \log n$. A pesar de que esto no garantiza que alguna operación en particular tome un tiempo de $O(n)$, si asegura que no existe ninguna secuencia de operaciones que sea mala. En general, cuando una secuencia de M operaciones toma tiempo $O(M\ f(n))$, se dice que el costo amortizado en tiempo de cada operación es $O(f(n))$. Por lo tanto, en un AD los costos amortizados por operación son de $O(\log n)$." (Vásquez & Vega, 2009)

¿Para qué se ocupa un Splay Tree (aplicaciones)?

Un árbol biselado o desplegado tiene varios usos, se utilizan principalmente en sistemas donde acceder a un mismo elemento de manera repetitiva es común. De acuerdo con información de la universidad de Cornell, los routers de red son buen ejemplo, pues deben apuntar los paquetes de información a gran velocidad a distintos cables basados en una IP. Muchas veces esos "pings" ocurren de manera repetitiva y seguida, por lo que incurrir en un costo mayor la primera vez que se debe de enviar un paquete de datos a cierta dirección es eficiente. Todo con porque en los siguientes "pings" seguidos del primero se enviarán de manera instantánea junto con sus datos.

Otro uso importante donde los árboles desplegados predominan es para las memorias cache. Es decir, elementos en memoria que recientemente ha utilizado el procesador de una computadora y posiblemente los vuelva a utilizar. Los árboles son utilizados para crear algoritmos de memoria cache. (Teja, 2011)

Un tercer uso para los árboles biselados de acuerdo con Douglas W. Jones sería que, a través del algoritmo de "splaying", se pueden comprimir datos de manera eficiente bajo

ciertas condiciones. El encriptado de datos y procesamiento de imágenes áreas donde puede resultar útil este algoritmo.

¿Cómo se implementa un Splay Tree?

Es importante recordar que un árbol desplegado es un BST pero con una funcionalidad adicional que permite que cuando un dato en el árbol es leído, este pasa a ser la raíz del la estructura para requerir menor esfuerzo computacional si es que se busca ese dato nuevamente. Lo que esto quiere decir, es que el árbol cuenta con operaciones básicas como inserción, búsqueda, y remoción de elementos en la estructura de datos previamente estudiadas. A continuación, se describe un poco lo que es la implementación de esta estructura y sus métodos, así como su algoritmo de despliegue o de “splaying”.

De acuerdo con la investigación de Tarjan y Sleator, y con lo mencionado anteriormente, el splaying es una reetructuración del árbol sin perder las propiedades de un BST en lo que un elemento se abre paso hasta la raíz. Existen varios algoritmos que llevan a cabo esta misma acción, pero varían dependiendo de ciertas condiciones. A continuación, en cada árbol el nodo leído es x.

El primer caso llamado “zig” dice: “Si $p(x)$, el padre de x, es la raíz del árbol, rota la arista juntando x con $p(x)$.” (Sleator et al, 1985)

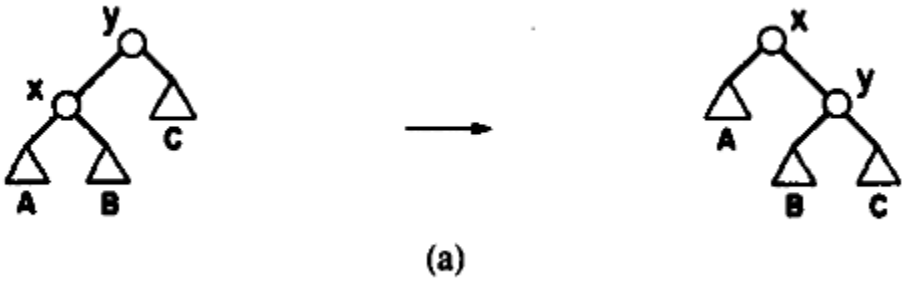

Figura 1.1 – Rotación de nodos x y y movimiento “zig”

Para el segundo caso, el movimiento llamado “zig-zig” es el siguiente: “Si $p(x)$ no es la raíz del árbol y tanto x como $p(x)$ son ambos nodos hijo izquierdos o derechos, rotar la arista juntando $p(x)$ con su abuelo $g(x)$ y luego rota la arista juntando x con $p(x)$.” (Sleator, 1985)

Figura 1.2 – Caso 2 “zig-zig”.

Para el 3er caso, denominado “zig-zag”, se tienen las siguientes condiciones: “si $p(x)$ no es la raíz y x es un hijo izquierdo y $p(x)$ es un hijo derecho, o viceversa, rota la arista juntando x con $p(x)$ y luego rotar la arista juntando x con la nueva $p(x)$.” (Sleator et al, 1985)

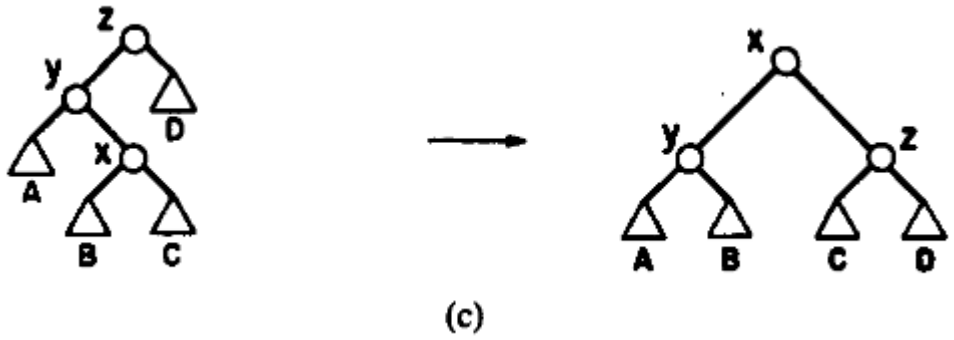

Figura 1.3 – Caso 2 “zig-zag”.

Como se puede observar gracias a las imágenes, los nombres de los movimientos como “zig”, “zig-zig” y “zig-zag” vienen cómo va cambiando la posición del nodo que será la nueva raíz. Después de haber visto los tres casos, y sus precondiciones para realizar el “splaying”, procederemos a ver como sería la implementación de estos algoritmos en pseudocódigo. El siguiente pseudocódigo fue obtenido de la página web del departamento de ciencias de la computación Loroya Marymont University en Los Ángles California [https://cs.lmu.edu/~ray/notes/splaytrees/].

```
while N is not the root:
    if N is a child of the root:
        // ZIG:
        Rotate about the root to bring N to the root
    else:
        P := N.parent
        G := P.parent  // the grandparent of N
        if N and P are both left or both right children:
            // ZIG-ZIG:
            Rotate about G then about P to bring N up two levels
        else:
            // ZIG-ZAG:
            Rotate about P then about G to bring N up two levels
```

Figura 1.4 – Pseudocódigo para algoritmo de “splaying”.

Referencias

Cornell University. (n.d.). Splay trees. Department of Computer Science. https://www.cs.cornell.edu/courses/cs3110/2013sp/recitations/rec08-splay/rec08.html

Jones, D. W. (1988). Application of splay trees to data compression. Commun. ACM, 31(8), 9966-1007. https://doi.org/10.1145/63030.63036

Loyola Marymont University. (n.d.). Splaytrees. https://cs.lmu.edu/~ray/notes/splaytrees/

Sleator, D. D., & Tarjan, R. E. (1985). Self-adjusting binary trees. Association for Computing Machinery, 32(3), 653-657.

Teja, R. (Diciembre 13, 2011). Splay Tree. Recuperado de: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwj734W3wcjzAhV1lWoFHXDTCU0QFnoECCEQAQ&url=http%3A%2F%2Fcs.indstate.edu%2F~rcheruku%2Fsplaytree.pdf&usg=AOvVaw170uI2ZkaGTfN8mFll6vJK

Vásquez, A., & Vega, Mg. (2009). Árboles Biselados. Revista de Ingeniería de Sistemas e Informática, 6(1). Recuperado de: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwid0suCusjzAhXtlmoFHd2-Ae0QFnoECA8QAQ&url=https%3A%2F%2Fsisbib.unmsm.edu.pe%2FBibVirtual%2Fpublicaciones%2Frisi%2F2009_n1%2Fv6n1%2Fa06v6n1.pdf&usg=AOvVaw0UqK6JHXAEk2H7a_6S7dlw