Diferentes Árboles como estructuras de datos

Daniel E. Hernández - 20180077

2019–03–07

Resumen

El siguiente documento busca explicar diferentes tipos de árboles utilizados como estructuras de datos.

1. Huffman Tree

1.1. Definición:

En 1951, David A. Huffman y sus compañeros de clase de teoría de la información del MIT tuvieron la oportunidad de elegir entre un trabajo trimestral o un examen final. El profesor, Robert M. Fano, asignó un trabajo sobre el problema de encontrar el árbol binario más eficiente. Huffman, incapaz de probar que ninguno de los códigos era el más eficiente, estaba a punto de abandonar y comenzar a estudiar para el final cuando se le ocurrió la idea de usar un árbol binario clasificado por frecuencias y rápidamente demostró que este método era el más eficiente.

Al hacerlo, Huffman superó a Fano, quien había trabajado con el inventor de la teoría de la información Claude Shannon para desarrollar un código similar. Construir el árbol de abajo hacia arriba garantiza la optimización, a diferencia de la codificación Shannon-Fano de arriba hacia abajo.[1]

1.2. Casos de uso:

1. **FLAC**: Compresión *lossless* de audio.[2]
2. **GZIP**: Compresión *lossless* de archivos. [3]
3. **PNG**: Compresión *lossless* de imágenes. [4]
4. **MP3**: *Encoding* de archivos de audio con compresión *lossy* (no utiliza el algoritmo para comprimir, lo utiliza únicamente para *encoding*). [5]

1.3. Implementación:

1. Crea un nodo hoja para cada símbolo y añadirlo a la cola de prioridades.
2. Mientras haya más de un nodo en la cola:
3. Eliminar los dos nodos de mayor prioridad (menor frecuencia) de la cola
4. Crea un nuevo nodo interno con estos dos nodos hijos y con un valor frecuencia igual a la suma de las frecuencias de los dos nodos.
5. Añada el nuevo nodo a la cola.
6. El nodo restante es el nodo raíz y el árbol está completo.

1.4. Ejemplo en graphviz:

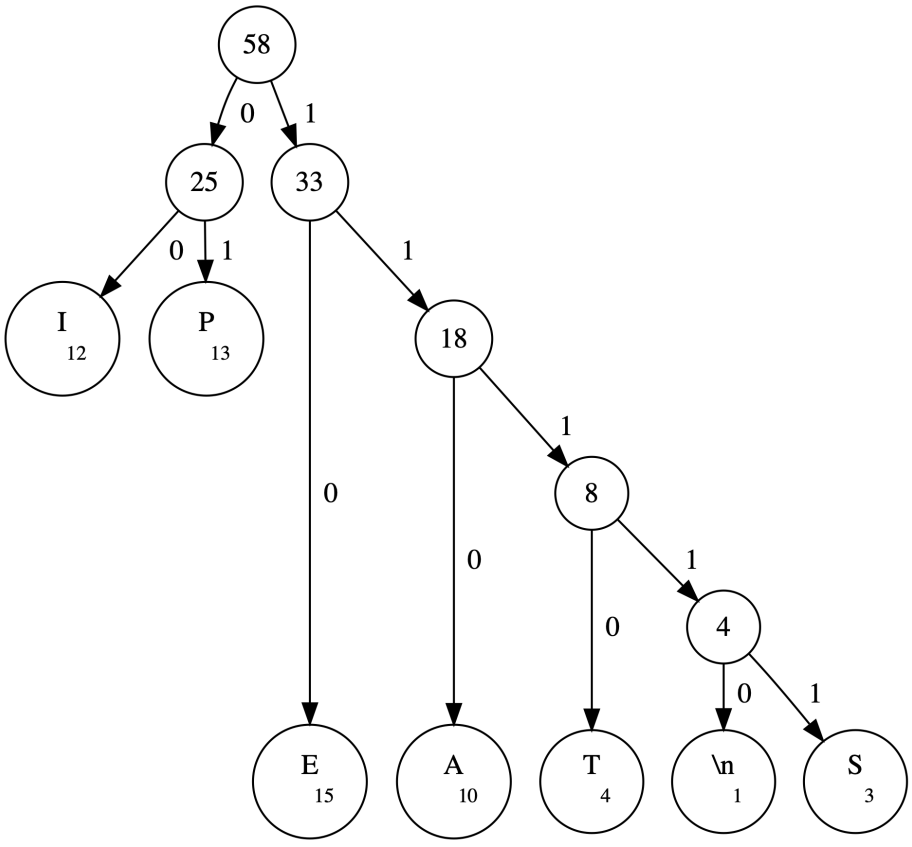

Figura 1: Huffman Aplicado

2. Min Binary Heap Tree

2.1. Definición:

El *Binary Heap Tree* fue introducido en 1964 por J. W. J. Williams como una estructura de datos para hacer *sorting*.[6] Este es un árbol de tipo completo y en el caso de ser un min-heap, el nodo padre es por definición siempre menor que su(s) respectivos nodos hijos.

2.2. Casos de uso:

1. **Ordenamiento estadístico**: Ordenar un arreglo de datos de menor a mayor en $O(\log n)$.
2. **Colas de prioridad**: Un ejemplo podría ser cambiarle de prioridad a un cliente dependiendo qué tanto se tarde el restaurante en servirle a él.
3. **Planificadores**: Un organizador de tareas en base a una prioridad dada por el usuario.

2.3. Implementación:

2.3.1. Insertar:

1. Se inserta un elemento en el próximo espacio disponible (abajo-derecha).
2. Se compara su valor con el nodo padre.
3. Mientras el valor del nodo insertado sea mayor que el del nodo padre, se intercambian de lugar ambos nodos.

2.3.2. Remover:

1. Se remueve el nodo raíz.
2. Se copia el último nodo insertado en la posición raíz.
3. Mientras el valor del nuevo nodo raíz sea mayor que el de alguno de sus hijos, se intercambia con el menor de ellos.

2.4. Ejemplo en graphviz:

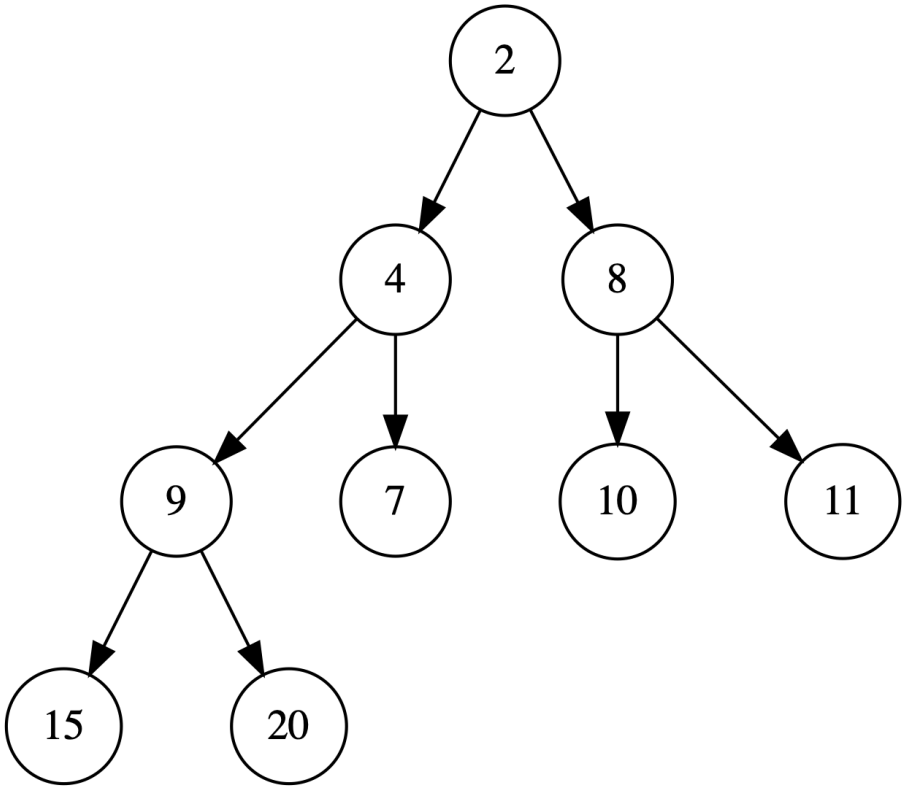

Figura 2: Min Binary Heap Tree

3. Treap

3.1. Definición:

El *Treap* fue descrito por primera vez por Raimund Seidel y Cecilia R. Aragón en 1989[7]; su nombre es un *portmanteau* de árbol *(tree)* y pila *(heap)*. Es un árbol cartesiano en el que cada nodo tiene una prioridad numérica (elegida al azar).

Al igual que con cualquier árbol de búsqueda binario, el orden de paso de los nodos es el mismo que el orden de las claves. La estructura del árbol viene determinada por el requisito de que esté ordenado en pilas: es decir, el número de prioridad para cualquier nodo que no sea de hoja debe ser mayor o igual a la prioridad de sus hijos. Así, al igual que con los árboles cartesianos en general, el nodo raíz es el nodo de máxima prioridad, y sus subárboles izquierdo y derecho se forman de la misma manera desde las subsecuencias del orden orden ordenado

a la izquierda y a la derecha de ese nodo. Siendo el lado izquierdo el nodo hijo con el valor más pequeño de ambos hermanos y el del lado derecho el nodo hermano con el valor más grande.

3.2. Casos de uso:

1. **Seguridad:** Gracias a que el *Treap* no contiene información o historia, resulta ser marginalmente más seguro.

2. **Eficiente:** Según algunos *benchmarks*[8] pueden ser más eficientes.

3.3. Implementación

1. Para buscar un dato determinado, aplique un algoritmo de búsqueda binario estándar en un árbol de búsqueda binario, ignorando las prioridades.

2. Para insertar un nuevo dato x en el árbol, genere una prioridad aleatoria y para x. La búsqueda binaria de x en el árbol y cree un nuevo nodo en la posición de la hoja donde la búsqueda binaria determina que debe existir un nodo para x. Luego, siempre y cuando x no sea la raíz del árbol y tenga un número de prioridad mayor que su padre z, realice una rotación de árbol que revierta la relación padre-hijo entre x y z.

3. Para eliminar un nodo x del árbol, si x es una hoja del árbol, simplemente retírelo. Si x tiene un solo hijo z, quite x del árbol y haga que z sea el hijo del padre de x (o haga z la raíz del árbol si x no tiene padre). Finalmente, si x tiene dos hijos, cambie su posición en el árbol por la posición de su sucesor inmediato z en el orden orden ordenado, dando como resultado uno de los casos anteriores. En este último caso, el swap puede violar la propiedad heap-ordering para z, por lo que puede ser necesario realizar rotaciones adicionales para restaurar esta propiedad.

3.4. Ejemplo en graphviz:

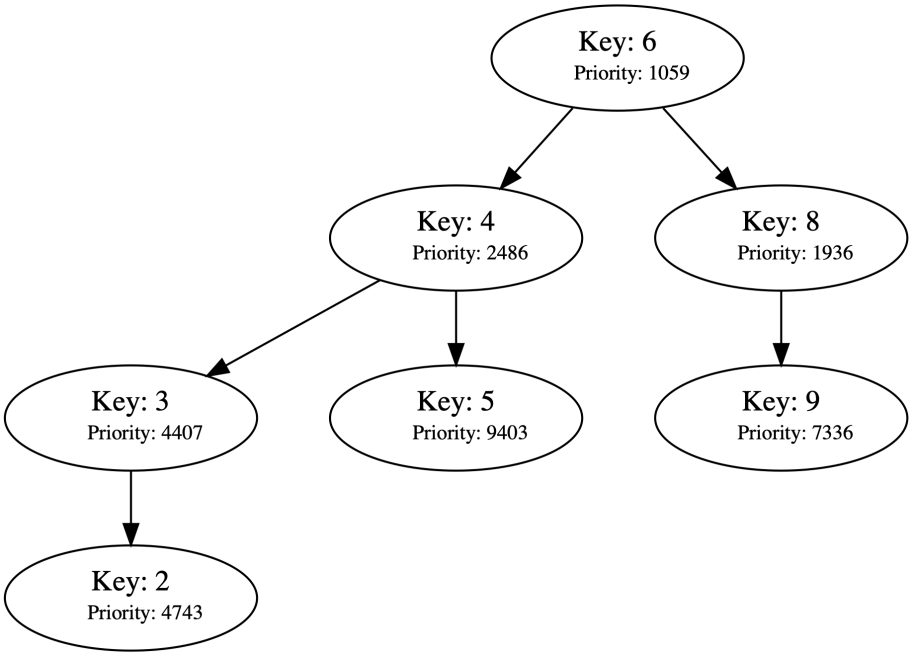

Figura 3: Treap

3.5. Red Black Tree

3.6. Definición:

En informática, un Red-Black Tree es una especie de árbol de búsqueda binaria autoequilibrado. Cada nodo del árbol binario tiene un bit extra, y ese bit se interpreta a menudo como el color (rojo o negro) del nodo. Estos bits de color se utilizan para asegurar que el árbol permanezca aproximadamente equilibrado durante las inserciones y eliminaciones.[9]

Tienen 5 Propiedades

1. Un nodo es o rojo o negro.
2. El nodo raíz es negro
3. Los nodos hijo de un nodo rojo son negros.
4. Si un nodo es rojo, entonces sus hijos son negros.
5. Todos los caminos de un nodo a sus descendientes `NIL` contienen la misma cantidad de nodos negros.

3.7. Casos de uso:

1. **Time-sensitive applications**
2. **Geometría computacional**
3. **Completely Fair Scheduler**: Maneja la asignación de recursos de la CPU para la ejecución de procesos, y tiene como objetivo maximizar la utilización general del CPU al mismo tiempo que maximiza el rendimiento interactivo.

3.8. Implementación

3.8.1. Insertar:

1. Insertar un nodo z rojo.
2. Se les da un nuevo color a los nodos y se rotan para arreglar las violaciones a las reglas del árbol.

Caso 0: Si z es raíz:

1. Se le cambia el color a z a negro.

Caso 1: Si el tío de z es rojo:

1. Se le cambia el color al abuelo, papá y tío.

Caso 2: Si el tío de z es negro en forma un triángulo:

1. Se rota z con su nodo padre.

Caso 3: Si el tío de z es negro en línea recta:

1. Se rota z con su nodo padre.
2. Se les cambia de color al abuelo y al padre de z.

3.9. Ejemplo en graphviz:

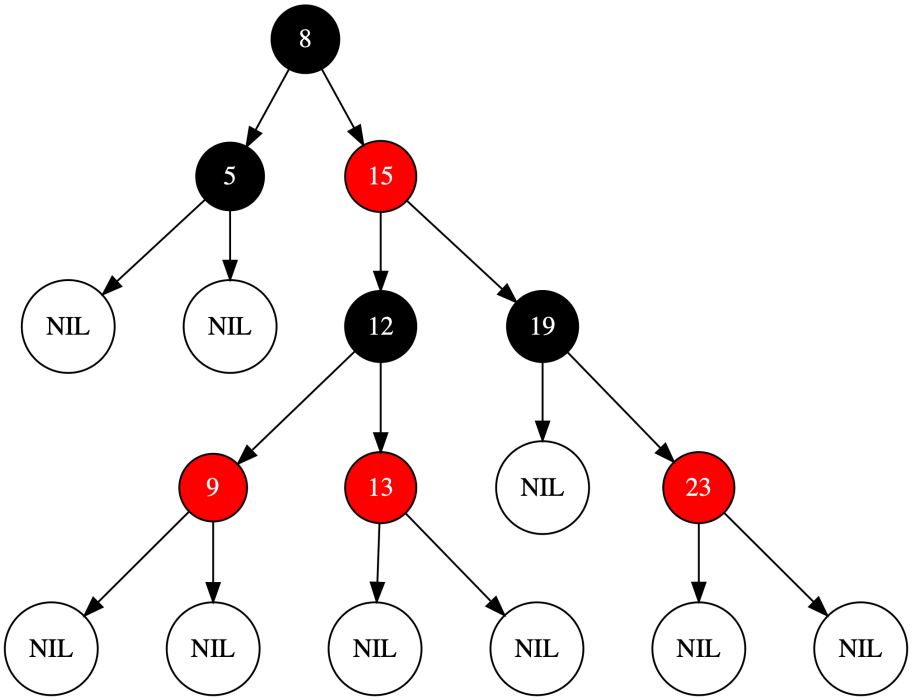

Figura 4: Red Black Tree

4. HTree *(hashed B-Tree)*

4.1. Definición:

Un HTree es una estructura de datos de árbol especializada para la indexación de directorios, similar a un árbol B. El algoritmo HTree se distingue de los métodos estándar del árbol B por su tratamiento de colisiones de hash, que pueden causar un *overflow* a través de múltiples hojas y bloques de índice. Los índices HT se utilizan en los sistemas de archivos ext3 y ext4 de Linux, y se incorporaron al núcleo de Linux alrededor de 2.5.40 [10].

Contiene dos tipos de bloques en un directorio indexeado.

1. **DX-block** *(directory indices block)*: Guarda el `hash-value`/`block-ID`.
   a) **Hash Value**: El valor en hash de un *entry name*.
   b) **Block-ID**: Ya sea el número de block lógico de el block hijo, o el siguiente nivel de índice.

2. **DE-block** *(directory entries block)*: Guarda los nombres de los directorios (nombres de archivos).

4.2. Casos de uso:

1. Directorios grandes en los *filesystems* `ext3` y `ext4`.

4.3. Implementación:

4.3.1. Operaciones:

1. **Búsqueda**: Busca una entrada de nombre en htree, o busca la entrada de nombre en la ruta htree.
2. **Insertar entrada de nombre**: Insertar una entrada de nombre en el DE-block en la ruta del árbol.
3. **Borrar entrada de nombre**: Eliminar una entrada de nombre del DE-block en la ruta htree.
4. **Dividir DE-block**: Si se inserta una entrada en un DE-block completo, se asigna un nuevo bloque y la mitad de las entradas de nombre del DE-block original se mueven al nuevo bloque.
5. **Dividir DX-block**: Cuando se asigna un nuevo DE-block, debe insertarse en el DX-block principal. Al insertar un DE-block en un DX-block completo, el DX-block se divide.
6. **Hacer crecer el árbol**: Un árbol crece si se produce una división de DE-block y todos los DX-block de la trayectoria del árbol están llenos. En este caso se añade otra capa al árbol.

4.3.2. Hash collision:

Las entradas en el directorio htree son indexadas por un valor hash del nombre. Existe una posibilidad improbable, pero finita, de que dos nombres cualesquiera no generen un valor de hash único. Esto se llama colisión de hash. Si estos nombres se extienden por más de un bloque, entonces la búsqueda de cualquiera de estos nombres necesita iterar en todos estos bloques, y esto debe ser manejado correctamente para PDO.

4.4. Ejemplo en graphviz:

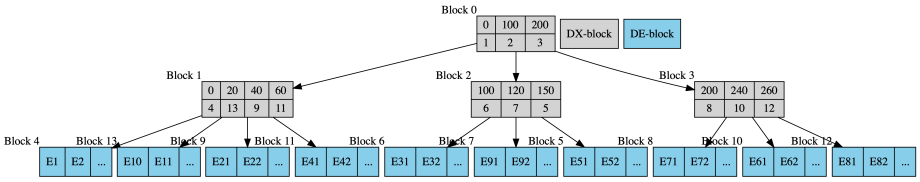

Figura 5: Htree

Referencias

[1] Gary Stix. *Scientific American Article | Huffman Coding*. en-US. https://www.huffmancoding.com/my-uncle/scientific-american. 1991.

[2] Institute of Electrical and Electronics Engineers. *2016 4th International Conference on Information and Communication Technology (ICoICT)*. English. http://ieeexplore.ieee.org/servlet/opac?punumber=7565234. 2016.

[3] Free Software Fundation. *GNU Gzip*. https://www.gnu.org/software/gzip/manual/gzip.html\#index-options-4. Nov. de 2008.

[4] libpng. *PNG Specification: Deflate/Inflate Compression*. http://www.libpng.org/pub/png/spec/1.2/PNG-Compression.html.

[5] *MP3' Tech - Overview of the MP3 Techniques*. http://www.mp3-tech.org/tech.html\#huffman.

[6] G. E. Forsythe. "Algorithms". En: *Communications of the ACM* 7.6 (jun. de 1964). http://portal.acm.org/citation.cfm?doid=512274.512284. ISSN: 00010782. DOI: 10.1145/512274.512284.

[7] Symposium on Foundations of Computer Science y IEEE Computer Society, eds. *30th Annual Symposium on Foundations of Computer Science , October 30 - November 1, 1989, Research Triangle Park, NC*. eng. OCLC: 20949758. Los Alamitos, Calif.: IEEE Computer Soc. Press, 1989. ISBN: 978-0-8186-1982-3.

[8] Pin-Sho Feng. *Pinporelmundo: Skip Lists Compared with Treaps and Red-Black Trees*. https://pinporelmundo.blogspot.com/2011/06/skip-lists-compared-with-treaps-and-red.html. Jun. de 2011.

[9] Thomas H. Cormen y Thomas H. Cormen, eds. *Introduction to Algorithms*. 2nd ed. Cambridge, Mass: MIT Press, 2001. ISBN: 978-0-262-03293-3.

[10] . *Add Ext3 Indexed Directory (Htree) Support [LWN.Net]*. https://lwn.net/Articles/11481/. Oct. de 2002.