República Bolivariana de Venezuela
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Técnicas Avanzadas de Programación
Semestre 1-2025

Implementaciones de *Heaps:* Binario, Binomial y Fibonacci

Estudiante:
César Carios
C.I 30.136.117

28 de mayo del 2025

Heap Binario

Los *heaps* son estructuras de datos en forma de árbol que soportan de una forma muy eficiente las operaciones de una cola de prioridad. En el caso del *heap* binario, se caracteriza por ser un árbol binario completo o semi completo. Esto significa que todos los niveles del árbol están llenos, excepto posiblemente el último, y en este último nivel, los nodos se llenan de izquierda a derecha. Esta característica permite representar eficientemente el *heap* utilizando un arreglo, donde las relaciones padre-hijo se calculan mediante índices: si un nodo está en el índice i, sus hijos izquierdos y derechos estarán en los índices $2i + 1$ y $2i + 2$ respectivamente, y su padre en el índice $\lfloor (i - 1)/2 \rfloor$.

Existen dos tipos principales de heaps binarios, definidos por la propiedad de orden que mantienen:

- **Min-heap**: En un min-heap, el valor de cada nodo padre es menor o igual que el valor de sus hijos. Esto implica que el elemento más pequeño del heap siempre se encuentra en la raíz. Para los experimentos realizados en esta investigación, se implementaron cada uno de los *heaps* de este tipo.
- **Max-heap**: En un max-heap, el valor de cada nodo padre es mayor o igual que el valor de sus hijos. Consecuentemente, el elemento más grande del heap se encuentra en la raíz.

En este tipo de *heap*, las operaciones de insertar y extraer el mínimo son $O(\log n)$, mientras que la operación de consultar el mínimo es $O(1)$[1]. Veamos el tiempo que se toma un *heap* binario en realizar N cantidad de las operaciones antes mencionadas:

Cuadro 1: Tiempos de ejecución (en microsegundos) del *heap* binario

Operaciones (N)	Inserción (µs)	Consulta (µs)	Extracción (µs)
10000	1000	0	4000
100000	7000	0	35 511
1000000	66 516	4999	499 376
10000000	660 698	41 680	6 662 128

Podemos observar que, a medida que la cantidad de operaciones N aumenta, hay un incremento general en los tiempos de ejecución para todas las operaciones, lo cual es esperable en cualquier algoritmo. Pero podemos notar que los tiempos de inserción muestran un crecimiento logarítmico, coherente con la complejidad teórica $O(\log n)$ de esta operación. Por ejemplo, al pasar de $N = 10^4$ a $N = 10^5$, el tiempo de inserción solo aumenta de 1000 μm a 7000 μm (un incremento de 7x). Similarmente, de $N = 10^5$ a $N = 10^6$, el tiempo crece de 7000 μm a 66516 μm (aproximadamente 9.5x). Este comportamiento sublineal confirma que, a pesar del aumento en la cantidad de elementos, el tiempo para insertar un nuevo elemento crece de manera controlada y eficiente, dado que la altura del árbol (y, por lo tanto, el número de intercambios) aumenta logarítmicamente.

Un aspecto notable en los resultados es el tiempo de consulta. Para $N = 10^4$ y $N = 10^5$, el tiempo reportado es de 0 μm. Esto sugiere que la operación de consulta del mínimo elemento en el *heap*, se realiza en un tiempo tan insignificante que es indetectable con la precisión de la medición para esas

[1] `https://www.geeksforgeeks.org/difference-between-binary-heap-binomial-heap-and-fibonacci-heap/`

escalas (se usó la librería *chronos* para esto). Para $N = 10^6$ y $N = 10^7$, los tiempos de consulta se vuelven mensurables (4999 μm y 41680 μm, respectivamente). Esto es consistente con la complejidad teórica de $O(1)$ para acceder al elemento de menor prioridad en la raíz.

Heap Binomial

Los *heaps* binomiales son una estructura de datos de tipo cola de prioridad más avanzada que los *heaps* binarios, diseñada para ofrecer una eficiencia superior en la operación de unión de dos *heaps*. A diferencia de un *heap* binario que es un único árbol semi completo, un *heap* binomial es una colección de árboles binomiales que cumplen con la propiedad de min-heap o max-heap. Esta estructura más compleja permite una mayor flexibilidad y un mejor rendimiento para un conjunto extendido de operaciones.

Una propiedad interesante de este tipo de *heap* es que para cualquier entero no negativo k, existe a lo sumo un árbol binomial de orden k en el *heap*. Esta propiedad es fundamental, ya que implica que un *heap* binomial con n nodos puede ser representado de forma única por los árboles binomiales correspondientes a los bits '1' en la representación binaria de n. Por ejemplo, un *heap* con 13 nodos ($13 = 1101_2 = 2^3+2^2+2^0$) contendrá un árbol B_3, un árbol B_2 y un árbol B_0. Además, los árboles raíz de un *heap* binomial se mantienen en una lista enlazada, ordenada por el grado de los árboles.

La mayoría de las operaciones en este tipo de *heap* es $O(\log n)$, y la de inserción es $O^*(1)$. Veamos los resultados del experimento hecho:

Cuadro 2: Tiempos de ejecución (en microsegundos) del *heap* binomial

Operaciones (N)	Inserción (µs)	Consulta (µs)	Extracción (µs)	Unión (µs)
10000	1001	0	3997	137 296
100000	12 004	996	52 810	2 174 965
1000000	131 678	9177	1 101 954	23 036 761
10000000	1 348 770	124 778	19 626 352	252 481 529

Similar al heap binario, los tiempos de ejecución para todas las operaciones en el heap binomial aumentan con el incremento de N. Sin embargo, la operación de unión es una adición clave que distingue al *heap* binomial, y sus tiempos de ejecución revelan una diferencia significativa en la eficiencia relativa de las operaciones. Los tiempos de inserción en el heap binomial muestran un crecimiento que se aproxima a $O(\log n)$. Por ejemplo, al pasar de $N = 10^4$ a $N = 10^5$, el tiempo aumenta de 1001 μs a 12004 μs (aproximadamente 12x). De $N = 10^5$ a $N = 10^6$, el incremento es de 12004 μs a 131678 μs (aproximadamente 11x). Este crecimiento logarítmico es consistente con la complejidad teórica de inserción amortizada de $O(1)$ y de peor caso $O(\log n)$, ya que la inserción se realiza creando un B_0 y uniéndolo al heap existente, lo que puede implicar una cascada de uniones. En el caso de las consultas, Para $N = 10^4$, la consulta reporta 0 μs, lo que sugiere una operación instantánea, consistente con la complejidad $O(1)$ para acceder al elemento mínimo en la raíz de uno de los árboles binomiales. A medida que N aumenta a 10^5, 10^6 y 10^7, los tiempos de consulta se vuelven mensurables (996 μs, 9177 μs y 124778 μs respectivamente). Aunque teóricamente es $O(1)$, la implementación real puede implicar buscar a través de la lista de raíces para encontrar el mínimo global, añadiendo una pequeña sobrecarga. La extracción del mínimo en el heap binomial muestra un comportamiento de crecimiento logarítmico, como se predice por su complejidad $O(\log n)$. Los tiempos aumentan de 3997 μs para $N = 10^4$ a 52810 μs para $N = 10^5$ (aproximadamente

13x), y de 1101954 μs a 19626352 μs (aproximadamente 17.8x) de $N = 10^6$ a $N = 10^7$. Esta operación es más costosa que la inserción, ya que implica localizar el árbol con el mínimo, eliminar su raíz, y luego unir los subárboles resultantes con el resto del heap. Por último, La operación de unión es una característica distintiva de los heaps binomiales y sus tiempos de ejecución son significativamente más altos que las operaciones de inserción, consulta y extracción. Los tiempos de unión crecen de 137296 μs para $N = 10^4$ a 2174965 μs para $N = 10^5$ (aproximadamente 15.8x), y de 23036761 μs a 252481529 μs (aproximadamente 10.9x) de $N = 10^6$ a $N = 10^7$. A pesar de estos valores absolutos elevados, la complejidad teórica de la unión es $O(\log n)$, lo cual es su principal ventaja comparativa. El mayor factor constante en la práctica se debe a la necesidad de fusionar las listas de raíces y combinar árboles binomiales del mismo orden.

Heap de Fibonacci

Los *heaps* de Fibonacci representan la estructura de datos de cola de prioridad más sofisticada, optimizada para ofrecer las mejores complejidades asintóticas amortizadas para una amplia gama de operaciones, superando en eficiencia a los heaps binarios y binomiales en escenarios específicos. Un *heap* de Fibonacci es una colección de árboles con raíces, no necesariamente árboles binomiales, que satisfacen la propiedad de min-heap en nuestro caso. A diferencia de los *heaps* binomiales, no hay una restricción estricta sobre el orden o la forma de estos árboles, lo que les permite ser más flexibles y realizar operaciones en un "modo perezoso". Los nodos de un*heap* de Fibonacci tienen punteros a su padre, a su primer hijo y a sus hermanos (implementados como una lista doblemente enlazada circular de hermanos). Cada nodo también almacena su grado (número de hijos) y un booleano marcado que se usa para gestionar las operaciones de disminución de clave.

La gran mayoría de las operaciones en este tipo de *heap* son $O^*(1)$, veamos los resultados obtenidos en nuestras pruebas:

Cuadro 3: Tiempos de ejecución (en microsegundos) del *heap* de Fibonacci

Operaciones (N)	Inserción (µs)	Consulta (µs)	Extracción (µs)	Unión (µs)
10000	1007	0	0	85 158
100000	10 523	0	0	844 624

Los tiempos de inserción para el heap de Fibonacci son de 1007 μs para $N = 10^4$ y 10523 μs para $N = 10^5$. El crecimiento es aproximadamente un factor de 10x cuando N aumenta 10x, lo cual es consistente con la complejidad teórica amortizada de $O(1)$. La inserción en un heap de Fibonacci implica simplemente añadir el nuevo nodo como un árbol de un solo nodo a la lista de raíces, una operación muy eficiente. Por otro lado, los tiempos de consulta son 0 μs para ambos $N = 10^4$ y $N = 10^5$. Esto se alinea perfectamente con la complejidad teórica de $O(1)$ amortizado para la operación de encontrar el mínimo (Find-Min). El heap de Fibonacci mantiene un puntero directo al nodo con el valor mínimo global, permitiendo un acceso instantáneo a este elemento. Los tiempos de extracción también son 0 μs para ambos $N = 10^4$ y $N = 10^5$. Esto es una observación inusual, ya que la extracción del mínimo es una de las operaciones más complejas en un heap de Fibonacci, con una complejidad amortizada de $O(\log n)$. Este resultado podría indicar que el tamaño de N no es lo suficientemente grande como para capturar el costo real de lo que implica hacer esta operación. Normalmente, la operación de extracción implica la remoción del mínimo y una fase de consolidación que reestructura los árboles, lo cual es significativamente más costoso que $O(1)$. La operación de unión en el heap de Fibonacci es de 85158 μs para $N = 10^4$ y 844624 μs para $N = 10^5$.

El crecimiento es aproximadamente 9.9x cuando N aumenta 10x. Esta operación tiene una complejidad teórica amortizada de $O(1)$, que se logra simplemente concatenando las listas de raíces de los dos heaps. Los tiempos absolutos, aunque más altos que la inserción, consulta y extracción (para los N probados), son relativamente eficientes si se considera que se está uniendo la totalidad de dos estructuras de datos complejas. El factor constante asociado puede ser mayor debido a la manipulación de las listas doblemente enlazadas y la actualización del puntero mínimo.

Como se puede ver en el cuadro de la medición de tiempos, solamente se logró experimentar con $N = 10000$ y $N = 100000$ porque cuando se intentaba probar con un N mayor, el compilador arrojaba un error de reserva de memoria. Esto tiene la siguiente explicación: A diferencia de los heaps binarios que se implementan eficientemente en un arreglo contiguo, y los heaps binomiales que, si bien usan punteros, organizan los nodos de manera más estructurada, los heaps de Fibonacci tienen una sobrecarga de memoria significativamente mayor por nodo. Cada nodo en un heap de Fibonacci típicamente necesita almacenar:

- El valor del elemento.
- Punteros a su padre, a su primer hijo, y a sus hermanos (lista doblemente enlazada circular: `next`, `prev`). Esto solo ya son varios punteros.
- Su grado (número de hijos).
- Un flag booleano `marcado`.

Cada uno de estos punteros y campos consume una cantidad considerable de memoria (típicamente 8 bytes por puntero en sistemas de 64 bits, y 4 bytes para el grado, 1 byte para el flag). Para N millones de nodos, la cantidad total de memoria requerida se dispara rápidamente. Por ejemplo, si cada nodo ocupa 40-50 bytes (valor + 4 punteros + grado + flag), entonces un millón de nodos requeriría 40-50 MB de memoria solo para los nodos, sin contar la sobrecarga del sistema operativo y del gestor de memoria. Para 10 millones de nodos, esto escalaría a 400-500 MB, que es una cantidad significativa, pero no inmanejable.

Consideraciones Finales

La elección de la estructura de heap más adecuada depende críticamente de las operaciones predominantes en la aplicación. Los tres tipos de heaps examinados ofrecen distintas ventajas y desventajas en términos de complejidad de implementación, consumo de memoria y rendimiento para operaciones clave. Podemos concluir que para aplicaciones generales con operaciones de inserción y extracción frecuentes y sin necesidad de uniones de heaps, el heap binario es la opción más sencilla y a menudo la más rápida. Si las uniones de heaps son un requisito importante, el heap binomial ofrece un buen equilibrio. Cuando las operaciones de inserción y, crucialmente, la disminución de clave son extremadamente frecuentes y la eficiencia asintótica amortizada es prioritaria, el heap de Fibonacci se posiciona como la estructura superior, asumiendo que se pueden manejar su complejidad de implementación y su mayor consumo de memoria.