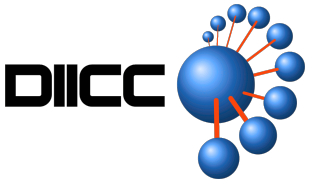


ESTRUCTURAS DE DATOS Y ALGORITMOS AVANZADOS

BOLETÍN N°2

SEPTIEMBRE 2023

ERICH GERMÁN GRÜTTNER DÍAZ

Matrícula: 2023300942

1. Introducción

El presente boletín tiene como objetivo comparar el rendimiento de operaciones para estructuras Binomial Heap y Binary Heap, en particular la inserción, unión y eliminación de datos.

Se mostrará los resultados de diferentes experimentos, con el objetivo de determinar el desempeño de las operaciones variando el tamaño del Heap, desde 10 hasta 100.000 datos.

Estos experimentos se ejecutan mediante una aplicación construida en lenguaje C++, que contiene los algoritmos a probar y diversas utilidades para la medición de rendimiento, como, por ejemplo, la librería Chrono.

Posteriormente se muestran los gráficos asociados a los resultados, generados a través de la librería Matplotlib, usando Phyton.

Dentro de las hipótesis a demostrar están:
- Similar rendimiento en operaciones de inserción y borrado O(log N)
- En operación de unión, rendimiento de O(N) para Binary Heap y O(log N) para Binomial Heap

Finalmente, en base a los resultados obtenidos, se presentan las conclusiones. Son comentados los rendimientos, las diferencias y los posibles escenarios óptimos para su uso.

2. Descripción estructuras a ser comparadas

2.1 Binomial Heap

Es una estructura de datos utilizada para implementar una cola de prioridad eficiente. Los binomial heap son una variante de los árboles binarios y se utilizan para realizar operaciones de inserción, extracción y unión en tiempo casi constante. [1]

Fueron inventados por Jean Vuillemin en 1978. Están compuestos por una colección de árboles binarios llamados “binomial trees”, que siguen ciertas propiedades específicas. Cada árbol binomial en un binomial heap que debe cumplir con las siguientes propiedades:

1. Cada árbol binomial es un árbol binario en el que cada nodo tiene un padre y puede tener hijos.

2. Cada árbol binomial cumple con la propiedad del **binary heap**, lo que significa que el valor de cada nodo es menor o igual que el valor de sus hijos (en el caso de un **min-heap**) o mayor o igual (en el caso de un **max-heap**).

3. No hay dos árboles binomiales con el mismo grado en el heap. El grado de un árbol binomial se refiere al número de hijos que tiene su nodo raíz.

4. Los árboles binomiales están organizados en una estructura jerárquica de manera que los árboles con grados crecientes se almacenan en una lista o conjunto, de modo que el heap puede tener árboles binomiales de diferentes grados.

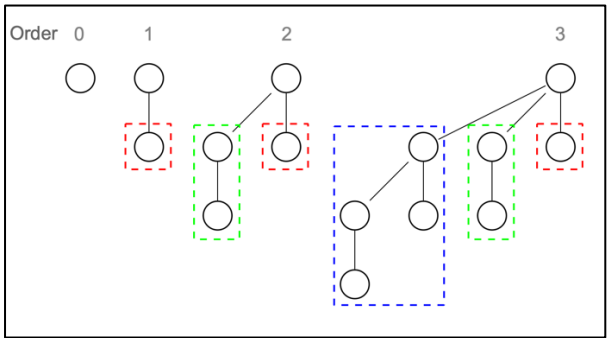


Figura 1 - Ejemplo Binomial Heap

2.2 Binary Heap

Es una estructura de datos basada en un árbol binario especializado que se utiliza comúnmente para implementar colas de prioridad. Fue desarrollado por J.W.J. Williams en 1964, como una estructura de datos para el HeapSort. [2] Principales propiedades:

1. En un montículo binario mínimo, para cualquier nodo dado, el valor del nodo es menor o igual que los valores de sus nodos hijos. En un montículo binario máximo, el valor del nodo es mayor o igual que los valores de sus nodos hijos. Esto garantiza que el nodo superior del montículo (la raíz) tenga el valor más bajo (en un montículo mínimo) o el valor más alto (en un montículo máximo).

2. Un montículo binario es un árbol binario completo, lo que significa que todos los niveles del árbol están completamente llenos, excepto posiblemente el último nivel, que se llena de izquierda a derecha.

3. Para facilitar la implementación y el acceso eficiente, un montículo binario se suele representar utilizando un arreglo (array). La posición de un nodo en el arreglo se calcula de manera que se mantenga la estructura de árbol completo.

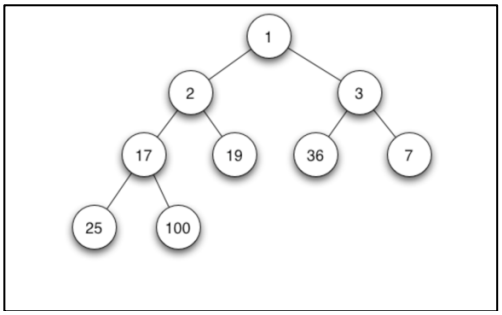


Figura 2 - Ejemplo Binary Heap

3. Descripción de los experimentos

3.1 Creación/inserción

Consiste, en base a carga de datos, de la creación de las dos estructuras Heap, insertando información y reorganizando la estructura.

Se consideraron 5 archivos de entrada de input, de diferentes tamaños: 10, 100, 1000, 10000 y 100000 registros. Cada uno de ellos contiene como primer elemento su tamaño y luego una lista desordenada de números enteros.

3.2 Unión

Se carga, en base a un archivo, dos Heaps y luego se opera la unión, implementada en forma diferente para cada estructura.

Se consideraron 5 archivos de entrada de input, de diferentes tamaños: 10, 100, 1000, 10000 y 100000 registros. Cada uno de ellos contiene como primer elemento su tamaño y luego una lista desordenada de números enteros.

3.3 Eliminación

Para este caso se busca el elemento mínimo, se elimina, y se mide el tiempo en que el Heap se "reconstruye".

Se consideraron 5 archivos de entrada de input, de diferentes tamaños: 10, 100, 1000, 10000 y 100000 registros. Cada uno de ellos contiene como primer elemento su tamaño y luego una lista desordenada de números enteros.

3.4 Procesamiento y gráficos

Cada prueba se realiza 30 veces y se obtiene el promedio del tiempo de ejecución, a través de la librería Chrono de C++.

Los tiempos quedan almacenados en archivos CSV, que luego permiten realizar gráficos de desempeño mediante Python y la librería Matplotlib.

3.5 Equipo de pruebas

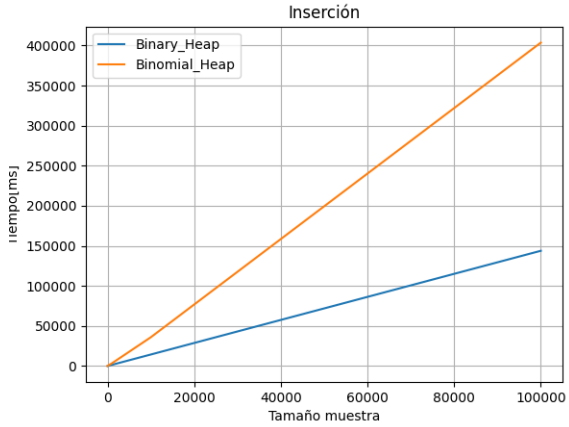
Para la realización de las pruebas se utilizó un equipo MacbookPro con procesador M1 y 8Gb de memoria. El chip M1 tiene 8 núcleos (4 de alta eficiencia a 3.2 GHz + 4 de alto rendimiento a 2.0 GHz) y una velocidad de transferencia de 50Gb por segundo.

3.6 Código fuente

El código fuente de la aplicación se encuentra disponible en el repositorio de Github [4], donde además se pueden encontrar los archivos gráficos, los archivos CSV de resultados, y los datasets de input.

4. Resultado de los experimentos

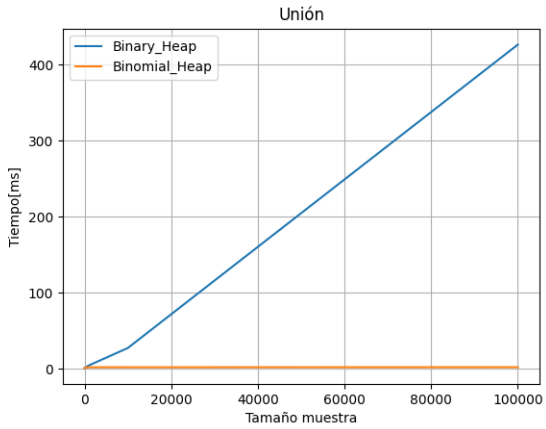
4.1 Creación/Inserción – Binomial Heap vs Binary Heap



Tamaño muestra	Binomial Heap [ms]	Binary Heap [ms]
10	37.7	15.5333
100	434.1	146.067
1000	3331.2	1464.3
10000	35909.7	14377.7
100000	403616	143818

- Se esperaría un rendimiento similar O(log N) de ambas estructuras para esta operación, sin embargo, Binary Heap presenta un mejor rendimiento. Se podría atribuir a la implementación ya que, a diferencia del Binomial Heap, el código interno utiliza la librería “algorithm” que está optimizada para el rendimiento.

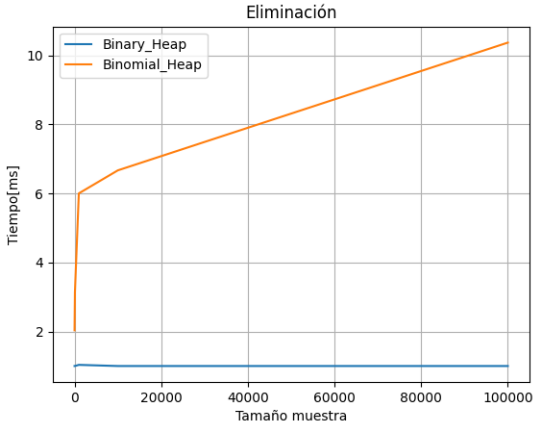
4.2 Unión – Binomial Heap vs Binary Heap



Tamaño muestra	Binomial Heap [ms]	Binary Heap [ms]
10	1	1.03333
100	1.3	1.06667
1000	2	4.4
10000	2	27.3333
100000	2.06667	426.633

- De acuerdo a [3] se espera un rendimiento, para el Binomial Heap de O(N) y para Binary Heap de O(log N), lo que se puede observar en el gráfico, en particular con tamaños de muestra superior a 10000.

4.3 Eliminación – Binomial Heap vs Binary Heap



Tamaño muestra	Binomial Heap [ms]	Binary Heap [ms]
10	2.03333	1
100	3.13333	1
1000	6	1.03333
10000	6.66667	1
100000	10.3667	1

- De igual forma, se esperaba un rendimiento similar entre ambas estructuras O(log N), lo que ocurre solamente para tamaños pequeños de muestra. A partir de los 10.000 datos la diferencia se hace notoria y deja como ganador al Binary Heap.

5. Conclusiones

- Como hipótesis, de acuerdo a lo indicado en [3], se podrían esperar los siguientes rendimientos de operaciones y estructuras:

Operación	Binomial Heap	Binary Heap
Creación/Inserción	O(log N)	O(log N)
Eliminación	O(log N)	O(log N)
Unión	O(N)	O(log N)

  Pero, de acuerdo a los resultados obtenidos, solamente se aprecia una semejanza parcial. Se atribuye, principalmente, a la implementación, ya que para el caso del Binary Heap, se utilizó la librería “algorithm” que está más optimizada que el código realizado para Binomial Heap.
- Se confirma el mejor rendimiento del Binary Heap para realizar la unión, dada la naturaleza de su estructura.
- Para el caso de Binary Heap fue utilizado como Min-Heap para poder ser comparado con el Binomial Heap, lo que habla de su versatilidad en ambos escenarios.

6. Referencias

[1] Wikipedia, Binomial Heap. [En línea]. Disponible: https://en.wikipedia.org/wiki/Binomial_heap

[2] Wikipedia, Binary Heap. [En línea]. Disponible: https://en.wikipedia.org/wiki/Binary_heap

[3] GeeksforGeeks, Difference Betweeen Binary Heap…. [En línea]. Disponible: https://www.geeksforgeeks.org/difference-between-binary-heap-binomial-heap-and-fibonacci-heap/

[4] Github, Repositorio de código fuente. [En línea]. Disponible: https://github.com/egruttner/FEDA2-Boletin_2/tree/main/code