|  |  |  |
| --- | --- | --- |
|  |  |  |

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: uso de librería “algorithm” que debiera operar más rápidamente que un desarrollo custom, tabals

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 de tipo cola de prioridad que se basa en una colección de árboles binomiales. Se desarrolló para la manipulación eficiente de colas de prioridad, que a menudo se utilizan en algoritmos como el algoritmo de Dijkstra para encontrar el camino más corto y el algoritmo de Prim para encontrar el árbol de expansión mínimo.

1. \*\*Árboles Binomiales\*\*: Un montículo binomial está compuesto por una colección de árboles binomiales. Un árbol binomial es un tipo específico de árbol que sigue algunas reglas:

- Un árbol binomial de orden k tiene exactamente 2^k nodos.

- Cada nodo en el árbol tiene una relación padre-hijo con otros nodos, y todos los hijos de un nodo se organizan en una lista enlazada con el mismo orden.

2. \*\*Orden del Montículo\*\*: En un montículo binomial, cada árbol binomial mantiene la propiedad de orden del montículo, lo que significa que la clave de un nodo es mayor o igual a las claves de sus hijos (en un montículo mínimo) o es menor o igual a las claves de sus hijos (en un montículo máximo).

3. \*\*Combinación de Árboles Binomiales\*\*: Los árboles binomiales en un montículo binomial se organizan de manera que cada orden sea único. Cuando se fusionan dos árboles binomiales del mismo orden, crean un nuevo árbol binomial de orden superior. Esta operación de fusión se realiza durante varias operaciones en el montículo para mantener su estructura.

4. \*\*Operaciones\*\*:

- \*\*Inserción\*\*: Insertar un elemento en un montículo binomial implica crear un nuevo árbol binomial de orden 0 (un solo nodo) y luego fusionarlo con el montículo binomial existente.

- \*\*Unión\*\*: La fusión de dos montículos binomiales es una operación fundamental. Implica fusionar los árboles binomiales de ambos montículos para crear un nuevo montículo binomial que conserva la propiedad del orden del montículo.

Los montículos binomiales son una estructura de datos eficiente para muchas operaciones relacionadas con colas de prioridad y se utilizan en algoritmos que requieren acceso rápido a elementos con prioridades específicas.

## 2.2 Binary Heap

Es una estructura de datos fundamental en ciencias de la computación que se utiliza comúnmente para implementar colas de prioridad y realizar operaciones de ordenamiento eficientes. Aquí tienes una descripción de un montículo binario:

1. \*\*Montículo Binario\*\*: Un montículo binario es un árbol binario completo que satisface una propiedad especial llamada "propiedad del montículo" o "heap property". Hay dos tipos de montículos binarios: el montículo mínimo y el montículo máximo.

2. \*\*Propiedad del Montículo Mínimo\*\*: En un montículo mínimo, para cada nodo, el valor del nodo es menor o igual que el valor de sus hijos. Esto significa que el nodo superior (raíz) del montículo mínimo siempre contiene el elemento con la menor clave en la cola de prioridad.

3. \*\*Propiedad del Montículo Máximo\*\*: En un montículo máximo, para cada nodo, el valor del nodo es mayor o igual que el valor de sus hijos. Esto significa que el nodo superior (raíz) del montículo máximo siempre contiene el elemento con la mayor clave en la cola de prioridad.

4. \*\*Operaciones Comunes\*\*: Los montículos binarios admiten varias operaciones eficientes, como la inserción de elementos, la eliminación del elemento principal (ya sea el mínimo o el máximo, según el tipo de montículo), y la construcción rápida de un montículo a partir de una lista de elementos.

En resumen, un montículo binario es una estructura de datos eficiente que se utiliza para mantener elementos en orden y permite un acceso rápido al elemento de mayor o menor prioridad, según sea necesario. Los montículos binarios son fundamentales en algoritmos como el algoritmo de Dijkstra, el algoritmo de Prim y el heapsort.

# 3. Descripción de los experimentos

## 3.1 Creación/inserción

Consiste, en base a carga de datos, la creación de las dos estructuras Heap.

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: 100, 1000, 10000, 100000 y 1000000 registros. Cada uno de ellos contiene como primer elemento su tamaño y luego una lista ordenada de números enteros ascendentes, comenzando por el número 1 y finalizando con el número de su tamaño prestablecido.

## 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]](#cuatro), 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

|  |  |
| --- | --- |
|  |  |

## 4.2 Unión – Binomial Heap vs Binary Heap

|  |  |
| --- | --- |
|  |  |

## 4.3 Eliminación – Binomial Heap vs Binary Heap

|  |  |
| --- | --- |
|  |  |

## 4.4 Experimentos adicionales

|  |  |
| --- | --- |
|  |  |

1.- Se observa que la búsqueda binaria tiene un desempeño mejor cuando los valores a buscar se encuentran en la zona central de arreglo, y la búsqueda galopante tiene un desempeño cercano a la búsqueda binaria, solamente cuando la ubicación del elemento está cerca del principio del arreglo.

2.- En búsquedas con elementos cercanos al principio del arreglo (10%) y de tamaño menor (Ej.: 100 registros), los 3 algoritmos resultan competitivos, incluso la búsqueda secuencial.

# 5. Conclusiones

Hipótesis

Operation Binary Heap Binomial Heap

insert O(log N) O(log N)

delete O(log N) O(log N)

union O(N) O(log N)

* El uso de librería algorith debería ser más rápido

# 

# 6. Referencias

[1] Wikipedia, Linear Search. [En línea]. Disponible: https://en.wikipedia.org/wiki/Linear\_search

[2] Wikipedia, Binary Search. [En línea]. Disponible: https://en.wikipedia.org/wiki/Binary\_search\_algorithm

[3] Wikipedia, Exponential Search. [En línea]. Disponible: https://en.wikipedia.org/wiki/Exponential\_search

[4] Github, Repositorio de código fuente. [En línea]. Disponible: https://github.com/egruttner/FEDA2-Boletin\_1/tree/main/code