

Laboratorio 4: Range Minimum Query

10 de Noviembre, 2025

Alumno	Mirko Peñailillo Vasquez
Profesor	José Fuentes Sepúlveda
Ayudante	Luciano Argomedo
N ^{ro} Matricula	2021900322

1 Resumen

El objetivo de este laboratorio es implementar, comprender y analizar el código de estructuras de datos de complejidad media para el problema de RMQ.

Se estudiaron las siguientes estructuras: Sparse Table y Segment Tree.

2 Análisis Teórico

2.1 Sparse Table

Es una estructura estática que preprocesa todas las combinaciones de rangos de tamaño potencia de dos. Su construcción tiene un costo de tiempo $O(n \log n)$ y un uso de memoria del mismo orden. Una vez construida, permite responder consultas de rango en tiempo $O(1)$. No permite actualizaciones, ya que cualquier cambio en el arreglo requeriría reconstruir toda la tabla. Su ventaja principal es la rapidez de las consultas, siendo ideal para datos que no cambian.

2.2 Segment Tree

Es una estructura de tipo árbol binario balanceado que almacena información agregada sobre subintervalos del arreglo. Su construcción requiere un tiempo y espacio de $O(n)$, donde n es el tamaño del arreglo. Permite realizar consultas de rango en tiempo $O(\log n)$ y además soporta actualizaciones de elementos individuales en el mismo orden. Por esto, es una estructura dinámica, adecuada para escenarios donde el arreglo puede modificarse después de su construcción. Sin embargo, su tiempo de construcción y sus consultas son más costosas en comparación con estructuras estáticas.

3 Implementación

Las implementaciones de ambas estructuras fueron obtenidas de el repositorio de github del curso [1].

4 Resultados Experimentales

4.1 Decisiones de inicialización

- Para la experimentación realizada sobre el efecto del tamaño de la secuencia en los tiempos de ejecución, se consideraron valores n en el rango de 1000 a 100000, con incrementos de 1000. Para cada tamaño n , se generó un vector A con valores aleatorios uniformemente distribuidos en el intervalo $[-10^6, 10^6]$. Este vector fue utilizado como entrada tanto para la construcción del Segment Tree como de la Sparse Table.
- En el caso de las mediciones de tiempo de construcción, para cada n se inicializó el vector A , y se midió el tiempo total que toma construir la estructura completa. En el caso de la sparse table, se empleó una función de combinación por mínimo (`fmin`), mientras que en el segment tree se usó su constructor estándar con la misma operación de mínimo.

- En los experimentos de consultas (queries), la estructura de datos fue construida una única vez por cada tamaño n antes de comenzar la medición. Posteriormente, se generaron $Q = 4096$ consultas aleatorias. Cada consulta corresponde a un rango $[i, j]$, donde los índices i y j fueron generados de manera uniforme tal que $0 \leq i < j \leq n$, asegurando que j sea exclusivo. Estos rangos se almacenaron en un vector de pares queries.
- Durante la medición del tiempo de consultas, se recorrió el conjunto de las Q consultas ejecutando la operación de rango mínima (query para el segment tree y query_idempotent para la sparse table). De este modo, el tiempo registrado corresponde exclusivamente al costo de las consultas, excluyendo el tiempo de construcción.
- Para cada experimento realizado se utilizó $RUNS = 64$ en el programa "uhr.cpp" disponible en el repositorio del curso [1].

4.2 Uso de datasets externos

No se utilizaron datasets externos para la experimentación.

4.3 Características de la maquina donde se realizaron los experimentos

- Procesador: AMD Ryzen 5 3600 (3.6 GHz)
- Memoria RAM: 16GB DDR4 (3600 MHz)

4.4 Descripción de los resultados mediante tablas o graficos

- En cada figura podemos observar los graficos correspondientes a los resultados obtenidos durante la experimentación con cada estructura. Los graficos correspondientes a construcción corresponden a las figuras 1 y 2, mientras que los graficos correspondientes a queries corresponden a las figuras 3 y 4. Finalmente los graficos 5 y 6 corresponden a graficos comparativos entre ambas estructuras.
- Notar que en cada figura se puede observar un ligero ruido por efecto de cache/OS, pero se pueden observar claramente las tendencias de las curvas.
- Graficos
En la figura 1, correspondiente al grafico del tiempo promedio de construcción del segment tree con tamaño de secuencia variable, se observa una curva con crecimiento lineal n , lo que concuerda con la complejidad temporal teórica de $O(n)$ para su construcción.

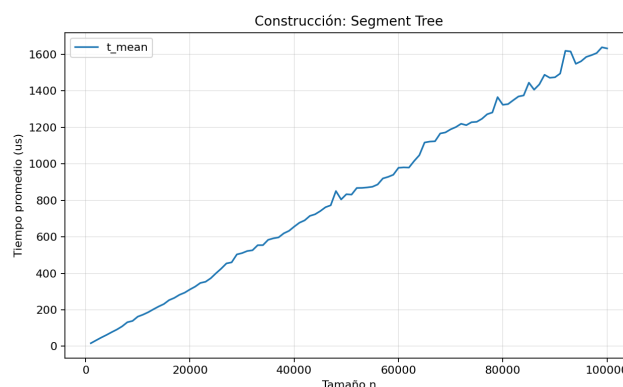


Figure 1: Experimentación construcción segment tree

En la figura 2, correspondiente al grafico del tiempo promedio de construcción de la sparse table, se observa una tendencia aproximadamente lineal con respecto al tamaño n , aunque con un tiempo de construcción considerablemente mayor que el del segment tree. Si bien la complejidad teórica de esta estructura es $O(n \log n)$, en el rango experimental evaluado el factor $\log n$ es relativamente pequeño, por lo que el crecimiento se aproxima a una recta. Además, la menor localidad de memoria y el uso de una matriz bidimensional durante el preprocesamiento incrementan el tiempo absoluto de construcción.

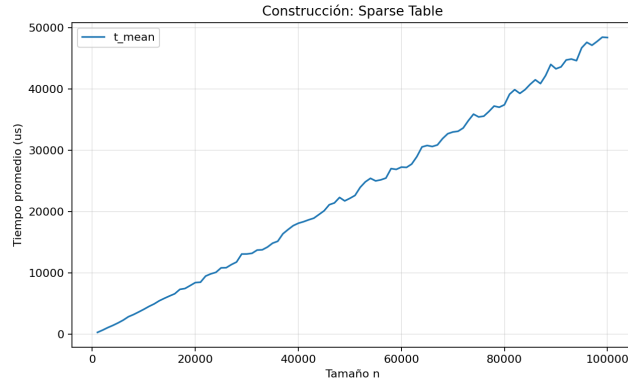


Figure 2: Experimentacion construccion sparse table

En la figura 3, correspondiente al gráfico del tiempo promedio de consultas del segment tree (4096 consultas por tamaño n), se observa un aumento gradual del tiempo total conforme crece n , lo que es coherente con la complejidad $O(\log n)$ de las operaciones de consulta.

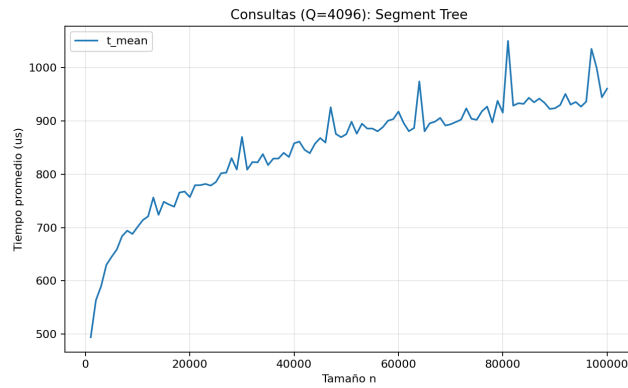


Figure 3: Experimentacion consultas segment tree

En la figura 4, correspondiente al grafico del tiempo promedio de consultas de la sparse table, se aprecia que el tiempo se mantiene practicamente constante para todos los tamaños de secuencia, lo que valida la complejidad $O(1)$ por consulta teorica de esta estructura.

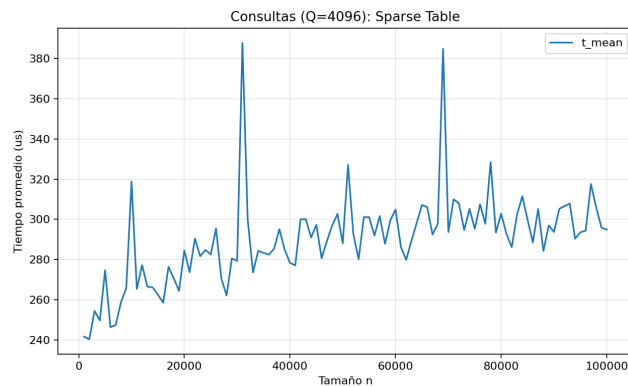


Figure 4: Experimentacion consultas sparse table

En la figura 5, donde se comparan los tiempos de construcción de ambas estructuras, se aprecia que la curva correspondiente al segment tree se mantiene practicamente constante en todo el rango de tamaños, mientras que la sparse table presenta un crecimiento lineal pronunciado. Esto indica que, aunque ambas estructuras tienen complejidades de construccion teóricas $O(n)$ y $O(n \log n)$ respectivamente, en la practica

el segment Tree logra una construcción mucho más eficiente debido a su acceso secuencial a memoria y menor sobrecarga computacional.

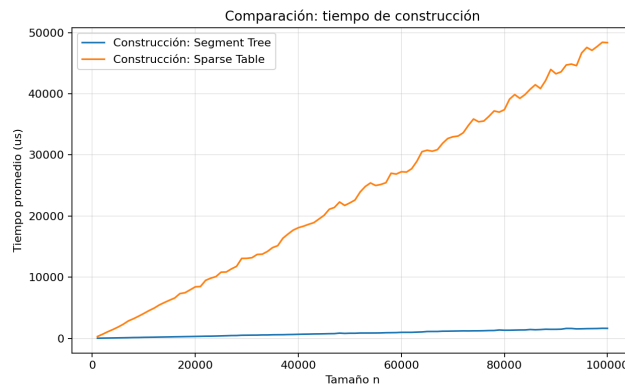


Figure 5: Comparacion resultados construccion

En la figura 6, correspondiente a la comparación de los tiempos de consulta de ambas estructuras, se evidencia que la sparse table responde las consultas en tiempos notablemente menores y prácticamente constantes, mientras que el segment Tree muestra un incremento progresivo con n , confirmando así la diferencia entre las complejidades $O(1)$ y $O(\log n)$.

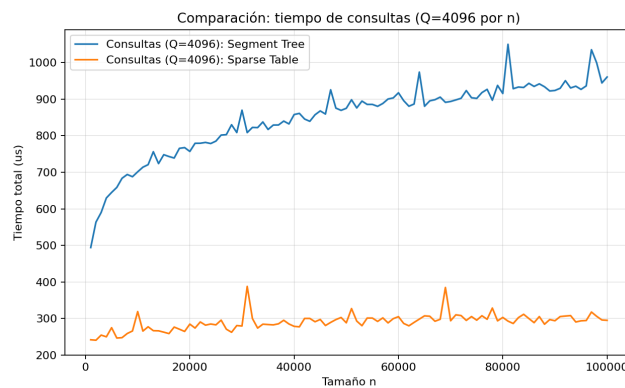


Figure 6: Comparacion resultados consultas

5 Conclusiones

5.1 Validacion de complejidades temporales teoricas

En los resultados presentados se puede observar que, si bien existe ruido, los resultados experimentales, en general, presentan una tendencia a las complejidades temporales que fueron discutidas en clases y presentadas en la sección Análisis Teórico de este informe.

- En los experimentos de construcción, el segment tree muestra un crecimiento lineal con el tamaño de la secuencia, consistente con una complejidad temporal de $O(n)$. Por otro lado, la sparse table presenta un crecimiento claramente más pronunciado. La diferencia entre ambas estructuras se acentúa a medida que aumenta n , donde la construcción de la sparse table resulta aproximadamente una orden de magnitud más costosa.
- En los experimentos de consultas, se observa que los tiempos de la sparse Tables se mantienen prácticamente constantes independientemente del tamaño del arreglo, confirmando su complejidad de $O(1)$ por consulta. En contraste, los tiempos de consulta del segment tree aumentan lentamente con n , lo cual concuerda con una complejidad de $O(\log n)$. Aun considerando el ruido experimental, la diferencia de pendiente entre ambas curvas valida las complejidades teóricas analizadas.

5.2 Que algoritmo usar y cuando

Ambas estructuras cumplen el mismo proposito, responder consultas de rango, pero difieren en su enfoque.

- El segment tree resulta mas adecuado cuando el arreglo de entrada puede cambiar entre consultas, ya que permite actualizaciones en $O(\log n)$ sin necesidad de reconstruir la estructura completa. Su costo de construccion es bajo y su uso de memoria moderado.
- La Sparse Table, en cambio, está diseñada para escenarios estaticos. Si el conjunto de datos no cambia, ofrece una ventaja significativa al responder consultas en tiempo constante $O(1)$, a costa de un mayor tiempo y espacio de construccion.
- Por tanto, en aplicaciones donde se realizan muchas consultas sobre un arreglo fijo (por ejemplo, analisis de rangos en datos preprocesados o consultas repetitivas), la sparse table es la eleccion mas eficiente. En cambio, en contextos donde el contenido del arreglo varia o se requiere actualizar frecuentemente los valores, el Segment Tree ofrece un mejor equilibrio entre tiempo de construccion, consulta y flexibilidad.

6 Referencias

References

- [1] jfuentess. "Edaa." Repositorio en GitHub, Accessed: Oct. 7, 2025. [Online]. Available: <https://github.com/jfuentess/edaa>.