Boletín 4:
Range Minimum Query

Nombre del estudiante: Dazhi Enrique Feng Zong
Número de matrícula: 2019435710

Introducción

Este cuarto boletín se enfoca en la consulta de Rango Mínimo (RMQ). RMQ es una operación utilizada para encontrar el elemento mínimo en un rango específico de un conjunto de datos.

Para abordar este problema, se explorarán dos enfoques: Segment Tree y Sparse Table.

Segment Tree es una estructura de datos basada en árboles binarios que permite realizar consultas de rango eficientemente. Cada nodo del árbol representa un segmento del conjunto de datos y almacena información relevante, como el mínimo valor en ese segmento.

La Sparse Table es otra técnica para abordar RMQ. Consiste en precalcular y almacenar respuestas para subrangos específicos del conjunto de datos en una tabla, lo que permite realizar consultas de rango en tiempo constante.

Ambas implementaciones serán analizadas experimentalmente para comparar su desempeño. Se variará el tamaño de entrada y se compararán los tiempos de construcción y consulta para la operación RMQ.

Las implementaciones utilizadas se basan en el código proporcionado en las Secciones 2.4.4 y 9.3 del libro Competitive Programming 4.

`https://github.com/stevenhalim/cpbook-code/blob/master/ch2/ourown/segmenttree_ds.cpp`
`https://github.com/stevenhalim/cpbook-code/blob/master/ch9/SparseTable.cpp`

Finalmente, se llevará a cabo una discusión detallada de los resultados obtenidos, destacando las ventajas y desventajas de cada implementación.

Desarrollo

Análisis Teórico

Segment Tree

- **Construcción:** $O(n)$, donde n es el tamaño del conjunto de datos. Este proceso implica dividir recursivamente el conjunto de datos en segmentos hasta alcanzar hojas individuales, asignando información relevante a cada nodo del árbol.
- **Consulta:** $O(\log n)$, donde n es el tamaño del conjunto de datos. Esto se debe a la búsqueda binaria a lo largo del árbol para encontrar la información del rango deseado.
- **Espacio:** $O(n)$, ya que se almacena información en cada nodo para cada elemento en el conjunto de datos.

Sparse Table

- **Construcción:** $O(n \log n)$, donde n es el tamaño del conjunto de datos. Este proceso implica precalcular y almacenar respuestas para subrangos específicos del conjunto de datos en la tabla, permitiendo consultas de rango en tiempo constante.
- **Consulta:** $O(1)$, ya que las respuestas para los subrangos están precalculadas y almacenadas en la tabla.
- **Espacio:** $O(n \log n)$, ya que se almacenan las respuestas para subrangos específicos del conjunto de datos.

Análisis Experimental

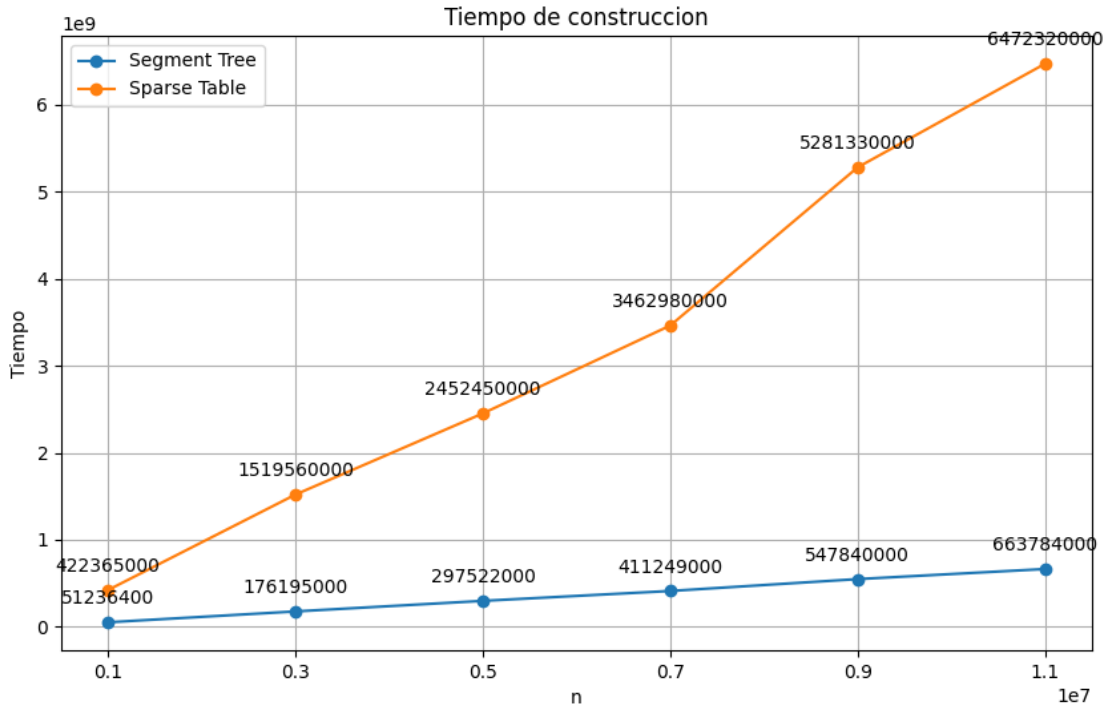
Se realizó un experimento para evaluar el rendimiento de cada implementación. Se generó un arreglo con números aleatorios y, en cada iteración, el tamaño del arreglo aumentó en incrementos de 2,000,000. De esta manera, se crearon arreglos de tamaños sucesivos: 1,000,000, 3,000,000, hasta 11,000,000.

En cada iteración, se registraron dos mediciones. En primer lugar, se midió el tiempo necesario para construir la estructura correspondiente. Posteriormente, se evaluó el tiempo requerido para realizar consultas de Rango Mínimo (RMQ) con intervalos de manera aleatoria.

Cada experimento se repitió 40 veces. Las primeras 10 repeticiones se consideraron "cold runs" y no se incluyeron en el análisis final. Las 30 repeticiones restantes fueron consideradas en el análisis.

Las pruebas se llevaron a cabo utilizando el servidor *chome* de la Universidad de Concepción.

Tiempo Segment Tree vs Sparse Table



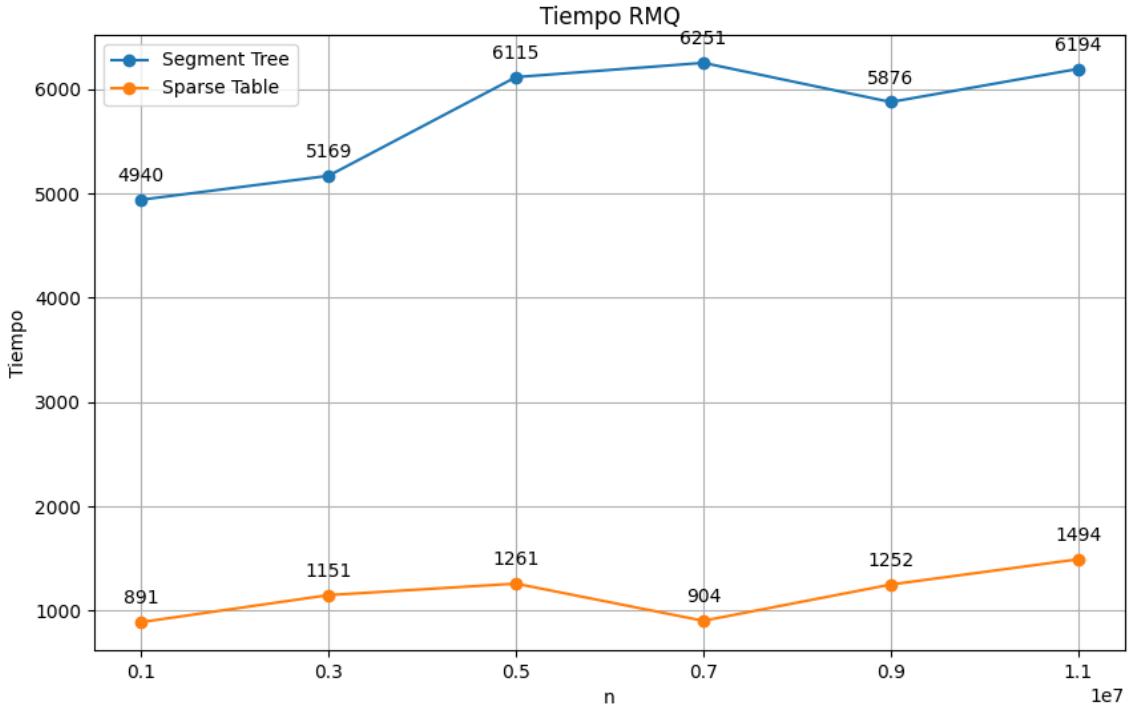


Tabla de los resultados

Donde n representa el número de elementos, $t1$ representa el tiempo de construcción, $t2$ representa el tiempo de consulta (ambos tiempos en nanosegundos) y v representa la varianza.

n	t1	v1	t2	v2
1000000	5.12364e+07	1.04832e+13	4939.67	787945
3000000	1.76195e+08	1.40742e+11	5168.9	616362
5000000	2.97522e+08	8.52997e+12	6114.7	1.54442e+06
7000000	4.11249e+08	1.10733e+13	6251.43	3.77966e+06
9000000	5.4784e+08	2.50392e+13	5875.87	926295
11000000	6.63784e+08	4.29991e+13	6193.63	1.23997e+06

Cuadro 1: Segment Tree

n	t1	v1	t2	v2
1000000	4.22365e+08	2.60446e+12	891.233	776775
3000000	1.51956e+09	1.40482e+16	1150.57	601921
5000000	2.45245e+09	1.28663e+14	1260.57	1.18463e+06
7000000	3.46298e+09	1.33835e+16	904.267	169355
9000000	5.28133e+09	6.23845e+14	1251.67	2.30177e+06
11000000	6.47232e+09	8.89882e+15	1494	2.9574e+06

Cuadro 2: Sparse Table

Discusión y Conclusión

Los resultados obtenidos respaldan de manera concluyente el análisis teórico realizado. Se observó que el tiempo de construcción del Segment Tree sigue una complejidad lineal, mientras que la construcción de la Sparse Table exhibe una complejidad logarítmica en función de n, siendo considerablemente superior al tiempo del Segment Tree.

En relación al tiempo de consulta para la operación de RMQ, se evidenció un comportamiento más o menos constante para la Sparse Table, aunque la diferencia con el tiempo logarítmico del Segment Tree no resultó muy significativa en el conjunto de datos actual. Es posible que con tamaños de entrada más grandes, se haga mayor la diferencia

Esta diferencia en los tiempos se explica intuitivamente: la Sparse Table precalcula todas las respuestas para permitir un acceso constante, mientras que el Segment Tree opera como un árbol, implicando un tiempo logarítmico para las consultas RMQ.

La elección entre Segment Tree y Sparse Table para RMQ dependerá de las necesidades específicas y los recursos disponibles. Por ejemplo, si se quiere realizar múltiples consultas RMQ y el tiempo de construcción no es un factor crítico, la elección podría inclinarse hacia la Sparse Table. Si se busca un equilibrio, con un tiempo de construcción menor pero con un tiempo de RMQ mayor, el Segment Tree podría ser preferible.

Asimismo, es importante considerar el espacio utilizado por ambas estructuras. Dado que la Sparse Table almacena las respuestas para permitir RMQ en tiempo constante, su requerimiento de espacio es mayor. En consecuencia, si se busca una implementación más liviana, podría ser preferible optar por el Segment Tree.

Un aspecto distintivo del Segment Tree es su capacidad para realizar la operación de actualización, permitiendo modificar algún nodo o valor del árbol en tiempo logarítmico, una funcionalidad que la Sparse Table no posee.

En resumen, el Segment Tree es la opción superior para RMQ en casos donde se hagan pocas consultas. La mayor ventaja de la Sparse Table es su tiempo de consulta constante, pero, si esta característica no se aprovecha, el Segment Tree ofrece un mejor desempeño en términos de tiempo de construcción, espacio y, además, proporciona la operación de actualización. La elección entre ambas estructuras es contextual y dependerá de las necesidades y recursos específicos de cada situación.