María Guzmán Valdezate

Guillermo López de Arechavaleta Zapatero

Práctica 4

Estructuras de datos

Grupo 201 GII

Contenido

[Descripción y Análisis de Métodos 3](#_Toc192701428)

[Pruebas y gráficas 4](#_Toc192701429)

[Conclusiones 5](#_Toc192701430)

## Descripción y Análisis de Métodos

**seleccionMultiple(List<E> lista, int[] seleccionados)**

Descripción: Devuelve una nueva lista con los elementos seleccionados según los índices proporcionados.

**seleccionInversaMultiple(List<E> lista, int[] eliminados)**

Descripción: Devuelve una nueva lista excluyendo los elementos cuyos índices están en el array de eliminados.

**particion(List<E> lista, int[] destino)**

Descripción: Divide una lista en sublistas según el array de destino. Cada elemento de la lista original se coloca en una sublista determinada por el índice en el array de destino.

## Pruebas y gráficas

Para validar las complejidades teóricas, se ha diseñado una clase de prueba que mide los tiempos de ejecución de los tres métodos utilizados en el programa utilizando el test facilitado y debidamente implementado TestRendimientoSeleccionListas.

Se realizan múltiples pruebas con tamaños de lista crecientes y se registran los tiempos de ejecución de los tres métodos. A continuación, se presentan los resultados en una tabla y sus respectivas representaciones gráficas.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tamaño de la lista** | **SeleccionMultiple (ms)** | **SeleccionInversaMultiple (ms)** | **Particion (ms)** |
| 0 | 0 | 0 | 0 |
| 10000 | 27 | 35 | 46 |
| 20000 | 86 | 113 | 175 |
| 30000 | 212 | 346 | 413 |
| 40000 | 354 | 695 | 706 |
| 50000 | 496 | 742 | 1018 |
| 60000 | 734 | 1005 | 1486 |
| 70000 | 970 | 1376 | 2716 |
| 80000 | 1293 | 1854 | 2639 |
| 90000 | 1639 | 2281 | 3530 |
| 100000 | 2053 | 2944 | 4140 |

Las diferencias que encontramos en los tiempos tienen que deberse a la implementación, ya que la implementación del test es igual en los tres métodos.

Con respecto a la implementación, si comparamos las implementaciones nos damos cuenta de que, el método seleccionMultiple simplemente recorre todos los elementos de un array de enteros y añade a una nueva lista el elemento en la posición correspondiente, lo que tiene una O(n) donde n es la longitud del array seleccionados.

Comparándolo con el segundo método seleccionInversaMultiple vemos que este segundo método se dedica a crear un array de longitud igual a la lista, con otro bucle recorre este array y cambia los valores correspondientes al array eliminados con un -1 para que se les pueda identificar después, gracias a un último bucle donde se guardaran solo aquellos números donde el numero sea mayor o igual que 0, lo que tiene una O(2n+m) donde n es la longitud de la lista y m es la longitud del array eliminados.

Por último, el último método partición donde vemos un bucle que se dedica a recorrer el array destino para justo después usar otro bucle para añadir cada elemento a la sublista, excepto en los casos en los que el indiceSublista = -1, lo que causa que su O () en el peor de los casos posibles.

Como hemos visto los valores de O () cuadran con los obtenidos en el experimento y representados en la gráfica.

## Conclusiones

seleccionMultiple es el más eficiente, con una complejidad de O(n), lo que se refleja en tiempos de ejecución menores en comparación con los otros métodos.

seleccionInversaMultiple es un poco menos eficiente debido a la necesidad de realizar múltiples recorridos sobre la lista y marcar elementos, con una complejidad de O (2n + m).

particion es el método menos eficiente en términos de tiempo de ejecución, con una complejidad de O(n²) en el peor de los casos, lo que hace que su rendimiento se degrade significativamente a medida que el tamaño de la lista aumenta.

Los tiempos de ejecución medidos en las pruebas siguen las tendencias esperadas según las complejidades teóricas calculadas, confirmando que el análisis matemático previo es una herramienta válida para predecir el rendimiento de los algoritmos.

Además, con este trabajo hemos observado la importancia de la optimización y el rendimiento del código que desarrollamos, ya que, al trabajar con grandes volúmenes de datos, las diferencias en el tiempo de ejecución entre distintas implementaciones pueden ser significativas.