Tecnológico de Monterrey

Nombre: Rodrigo Castillo Francisco

Matricula: A01799191

Materia: Programación de estructuras de datos y algoritmos fundamentales

Profesor: Eduardo Arturo Rodríguez Tello

Investigación individual y reflexión

1. Importancia y eficiencia de los algoritmos de ordenamiento

En este trabajo se desarrolló un código para ordenar una bitácora de registros que representan un intento de ataque de red en el que se basa en la fecha (mes, día, hora, minuto, segundo) por lo tanto por ser muchos registros lo que se hizo fue utilizar varios algoritmos de ordenamiento para ordenar y hacer mayor eficiencia de la búsqueda de posibles ataques a la red. En este caso los algoritmos con mayor eficiencia fueron tanto QuickSort como MergeSort esto es debido a que tienen una complejidad de O (n log n) que los hace muy adecuados para grandes conjuntos de datos aunque pueden llegar a tener una probabilidad de tener un peor caso y ser $O(n^2)$.Por otra parte los datos obtenidos de parte de otros algoritmos de ordenamiento que se hicieron pruebas como BubbleSort, SelectionSort, SwapSort demostraron tener una complejidad siempre de $O(n^2)$ por lo cual no eran eficientes y tardaban muchos en encontrar lo que se les pedía. Por lo tanto, los datos obtenidos como permiten contrastar esos valores de la complejidad teórica, lo que es crucial a la hora de justificar un algoritmo y para entender el comportamiento ante diferentes distribuciones de datos.

2.Comparación de los algoritmos de ordenamiento y análisis de complejidad

Cada algoritmo de ordenamiento que se implemento (QuickSort, MergeSort, InsertionSort, BubbleSort, SelectionSort y SwapSort) tienen sus propias características y complejidades por ejemplo QuickSort y MergeSort tiene una complejidad de O (n log n) mientras que InsertionSort, BubbleSort, SelectionSort y SwapSort tienen una complejidad de $O(n^2)$ en el peor caso. Por lo tanto, en la al ordenar una bitácora de registros de este tamaño la elección de un algoritmo de O (n log n) es fundamental para obtener enseguida los resultados que se requieren. Las pruebas que se realizaron para cada algoritmo fundamental la importancia de usar O (n log n).

3. Elección del algoritmo de búsqueda

Para la elección del algoritmo de búsqueda una vez ordenada la bitácora, se implementa la búsqueda binaria para realizar B operaciones de consulta por fecha. La búsqueda binaria que se implementó en el código tiene una complejidad de O (n log n) por consulta, lo que para los datos resulta ser altamente eficiente cuando se trata de realizar varias búsquedas en un vector ordenado. Este algoritmo que se usó minimiza el tiempo de búsqueda, permitiendo que el administrador de la red identifique de manera rápida los registros correspondientes de una fecha. La elección de la búsqueda binaria se justifica por su alta eficiencia además de menor trabajo computacional.

4. Pseudocódigo y análisis del algoritmo para encontrar el primer par de registros con D días de diferencia.

Entrada:

V : vector de registros ordenados

D: diferencia en días

diffObjetivo = D * 24 * 3600 (Operación para convertir de días a segundos.)

i = 0

j = i

Mientras (i<tamaño(V) y j <tamaño(V)) hacer:

Si V[j].aSegundo() >=V[i].aSegundo() entonces:

diff ⟵ V.[j].aSegundos()- V[i].aSegundos()

Sino:

Diff ⟵ 0

Si diff == diffObjetivo entonces:

    Retornar (i, j)

  Sino si diff < diffObjetivo entonces:

    j ⟵ j + 1

  Sino:

    i ⟵ i + 1

  Si (i == j) entonces:

    j ⟵ j + 1

Retornar (-1, -1)

En este pseudocódigo que hice la estructura del algoritmo se basa en dos índices que son i y j que recorren el vector ordenado. En el peor caso cada uno de estos índices avanza de forma línea cuando pasan por el vector por lo que el algoritmo haría un recorrido de O(n) por lo tanto es muy eficiente que evaluar a todas las parejas posibles que eso vendría siendo un algoritmo con complejidad $O(n^2)$).

La elección de este algoritmo es fundamental poque aprovecha el hecho de que el vector ya se encuentra ordenado, lo que este permite usar podría decirse dos punteros que recorren de forma lineal el vector. Esto se aprovecha que, aunque tuviera un peor caso el tiempo de ejecución sea (O(n)). Al no recurrir a una búsqueda de varias parejas que en ese caso es ($O(n^2)$) el algoritmo es muy eficiente para detectar rápido el primer par de registros con la diferencia exacta en días esto es importante porque si la bitácora tiene muchos registros aumenta la eficiencia de encontrar un ataque de manera rápida que es fundamental para identificar y res9olver el problema de los ataques en la seguridad de una red.

En conclusión, a pesar de que el algoritmo de ordenamiento que fue en este caso MergeSort hiciera menos comparaciones, QuickSort demostró ser más rápido que el en un menor tiempo de ejecución gracias a su complejidad O (n log n). Los algoritmos de ordenamiento sea cual sea se pueden usar en donde sea siempre y cuando el trabajo necesite de sus requerimientos, pero en este trabajo se compararon y se analizaron muchos datos por lo tanto resultó ser mas eficiente un algoritmo de ordenamiento O (n log n) que en este caso fue QuickSort.