Tecnológico de Monterrey

Campus Monterrey

Programación de estructuras de datos y algoritmos fundamentales

Act 6.2 - Reflexión Final de Actividades Integradoras de la Unidad de Formación TC 1031 (Evidencia Competencia)

Profesor: Dr. Eduardo Arturo Rodríguez Tello

A01571226 Mariano Barberi Garza

25 de Noviembre de 2022

https://github.com/MarianoBarberi/TC1031-Portafolio_Final-.git

Introducción

Los ciberataques cada vez son más frecuentes y más peligrosos para todo mundo ya que en estos últimos años toda nuestra documentación se ha ido transformando hacía un ente digital, por lo cual ya no son sólo cuentas de facebook, o email las que corren el riesgo de ser hackeadas, sino que cuentas bancarias y documentos oficiales también, es por esto que es de suma importancia saber cómo rastrear estos mismos ataques y cómo defenderte de esos mismos cuando estás controlando algún servidor o creando alguna aplicación, es por eso que debemos saber los diversos métodos para mejorar la eficiencia de nuestro código con ejemplos como lo son el método de hashing, los grafos y varios algoritmos de ordenamiento y búsqueda.

Reflexión 1.3

Importancia y Eficiencia del Uso de los Diferentes Algoritmos de Ordenamiento y Búsqueda

Algoritmos de Búsqueda

Búsqueda Secuencial:

La búsqueda secuencial consiste en recorrer de inicio a fin un array desde él y comprobar si alguno de los elementos es el vector buscado.

Esto lo hace de complejidad O(n), y en un archivo con muchos elementos a buscar haría el proceso muy lento y tedioso.

Búsqueda Binaria:

Consiste en analizar, en primer lugar el elemento central del vector, y elige el lado superior o inferior dependiendo de lo que se busque, después de cada comparación elimina la mitad de los elementos en el arreglo bajo búsqueda.

Esto lo hace que sea una complejidad de O(1) haciéndolo una forma de buscar muy eficaz y rápida, así agilizando cualquier proceso que quieras hacer dentro de tu programa que requiera buscar algún valor.

Algoritmos de Ordenamiento

Bubble Sort:

El bubble sort funciona de manera en la que empieza desde un lado y verifica si el valor siguiente es mayor o menor, si el programa quiere que lo cambie lo cambia, y eso hace con cada uno de los valores hasta verificar todo sin mover nada, significando que habrán muchos swaps en este algoritmo, pero aunque es fácil de implementar no es muy eficiente con su complejidad de O(n^2),

Selection Sort:

A diferencia del bubble sort, este algoritmo de ordenamiento no hace tantos swaps, la gran diferencia entre selection y bubble es solo eso, envés de cambiar siempre solo selecciona el dato hasta encontrar uno menor y luego lo cambia, aunque es una mejor del bubble sort sigue teniendo complejidad de O(n^2).

Insertion Sort:

Este algoritmo tiene un marcador que va moviendo mediante avanza así marcando lo que aún no se ordena, selecciona el valor más cercano a ese marcador y va comparando los valores que ya están ordenados para insertarlo en donde debería. O(n^2).

Merge Sort:

Este algoritmo siendo más difícil de implementar, por sus dos funciones necesarias vale mucho la pena ya que su complejidad de O(nlogn) crea un programa muy ágil, y como lo hace es que divide el array en dos y luego hace

lo mismo con las dos divisiones hasta que queden valores individuales y en ese momento empieza a hacer comparaciones para ordenar los valores.

Quick Sort:

Tenemos un pivote en este algoritmo lo cual hace posible que cualquier número que sea mayor al pivote se mueva a la derecha de él y cualquier valor que sea menor se queda ahí, esto se repite cambiando de pivote hasta que todo quede ordenado teniendo una complejidad de O(nlogn)

Reflexión 2.3

Importancia de uso de Doubly Linked List sobre Linked List

Primero debemos entender que es lo que puede hacer una Doubly Linked List que una Linked List no pueda, y esto es que la DLL puede viajar para adelante y para atrás, por lo cual hace que sea más eficaz, haciendo que en la LL hacer una inserción o borrado de cualquier nodo en una posición dada la complejidad es de O(n), en cambio en una DLL es de O(1), gracias a esto en bases de datos enormes es mucho más fácil y rápido utilizar una DLL aunque consuma más memoria.

Inserción y Borrado

Sabiendo que en la Doubly Linked List se puede viajar para adelante y atrás, al momento de querer hacer una inserción o un borrado en cierta posición es de gran ayuda, ya que no tiene que recorrer toda la base de datos en caso de que la posición

querida sea la última, este mejor desempeño al insertar y borrar datos no impacta mucho en esta solución, ya que la actividad no nos pide insertar ni borrar datos, pero emplear esta solución si nos ayuda en otras aplicaciones.

Búsqueda

La complejidad de la búsqueda es O(n) en todos los casos, lo cual podría ser mejor en una LL en algún caso, pero cuando queramos implementar ordenamientos la DLL será mucho mejor y es por eso que usamos la DLL.

Merge Sort

Este algoritmo siendo un poco complicado de implementar, por sus dos funciones (en este caso tres [split, merge, merge Sort]) necesarias vale mucho la pena ya que su complejidad de O(nlogn) crea un programa muy ágil, y como lo hace es que divide el array en dos y luego hace lo mismo con las dos divisiones hasta que queden valores individuales y en ese momento empieza a hacer comparaciones para ordenar los valores.

Esto es clave para la implementación de esta solución, ya que para dar un rango de fechas debemos primero tenerlas ordenadas para así poder verificar el momento del ataque más fácilmente, es de suma importancia que la complejidad de esto no sea enorme, ya que si tenemos una mala implementación de la ordenación nuestro programa al lidiar con tantos datos puede durar mucho tiempo, y si pensamos en incrementar la base de datos exponencialmente sería casi imposible con una computadora normal ordenar todos los datos si su complejidad es mayor.

Reflexión 3.4

Importancia y Eficiencia del uso de BST en Situación Problema

Binary Tree

Árbol ordenado, donde cada nodo tiene como máximo dos hijos, pueden tener cero hijos, un hijo o dos, pero no más. Un “rooted tree” naturalmente tiene niveles que se delimitan por la distancia desde la raíz, por lo que para cada nodo se puede definir una noción de hijos como los nodos conectados a él un nivel por debajo.

Binary Search Tree

Es un “binary tree” donde cada nodo en el árbol del lado izquierdo tiene un valor menor que el de la raíz, y cada nodo en el árbol derecho tiene un valor mayor que la raíz. Creando así una situación donde cada nodo puede ser comparado para saber si es mayor o menor, también se podría ordenar de manera alfabética, pero en este caso donde se nos pide ordenar IP’s primero debemos hacer que la ip tenga un valor numérico. O(n) donde n es la altura del árbol.

IP’s

En esta implementación fue importante darle un valor numérico a las IP’s para poder ordenarlo, la forma para hacer esto es multiplicar 256^[posición] * [valor del punto], por ejemplo si tenemos 190.10.20.40 se multiplicaría 256^0 * 40 + 256^1 * 20 + 256^2 * 10 + 256^3 * 190 = y te daría un valor numérico que se puede ordenar más fácilmente.

Heap Sort

Técnica para clasificar que se basa en comparaciones, basada en la estructura de datos Binary Heap. Se parece a la ordenación por selección en la que primero encontramos el elemento mínimo, se coloca el mismo al principio y repetimos este mismo proceso para los elementos restantes.

¿Cómo podrías determinar si una red está infectada o no?

Hablando sobre la situación problema y de su tema de infección, podemos determinar si un dispositivo está infectado o no a través de las conexiones entre nodos . Nos ayuda bastante utilizar esta estructura de datos (BST), para conocer de dónde viene la probable infección buscada. Ya que, es muy probable que no toda la red de datos esté comprometida por el ataque, sino que puede ser solo algún subárbol de la red y podemos conocer el origen de la infección.

Reflexión 4.3

Importancia y Eficiencia del Uso Grafos

Los ataques cibernéticos que estamos viendo cada día son más poderosos y cada vez es más fácil crear ataques y que estos dañen a los dispositivos de los usuarios, es por eso que es de suma importancia rastrear de dónde vienen estos ataques y saber la como dar seguimiento después de haber encontrado este mismo, pero para defendernos del ataque lo primordial es encontrarlo, ahí es donde entran los grafos.
Lo que hace tan importante el uso de grafos para la ciberseguridad es que una base de datos en grafos le da la posibilidad de crear entidades y relaciones de cualquier tipo,

esto termina creando muchas formas de las cuales se pueden hacer simulaciones para ver dónde en tu base de datos hay deficiencias.

Métodos implementados

Grafo:

Un grafo es una estructura matemática que permite modelar mediante una representación gráfica formada por nodos o vértices que muestra a los actores y las aristas que sirven para representar los lazos o relaciones entre los actores, las aristas pueden estar ponderadas, así haciendo que algunas conexiones tengan más importancia que otras.

Algoritmo de Dijkstra:

El algoritmo de Dijkstra es un algoritmo de complejidad O(n^2) (n es el número de vértices). Se utiliza para encontrar la ruta más corta o menos costosa desde cualquier nodo de origen a cualquier otro nodo en el gráfico. El algoritmo fue diseñado por Edsger Wybe Dijkstra en 1959.

Reflexión 5.2

Tablas de Hash

Estructura de datos que asocia llaves o claves con valores. Permite el acceso a elementos (como teléfono y dirección) almacenados por una clave generada usando ya sea el nombre, el número de cuenta o el ID.

Función de Hashing

Es la manera en la que se da una posición dentro de la tabla de hashing a un valor en concreto, para así transformar una entrada en un índice, este índice determina una posición aleatoria a lo largo de una longitud dada.

Para diseñar una buena función de hashing existen varios métodos, y los más populares son Selección de Dígitos, Residuales, Cuadrática, y Folding.

Colisiones

Aunque es muy probable con una buena implementación de las tablas de hash que no hayan colisiones, las implementaciones pueden no ser perfectas, por esto existe el manejo de colisiones, ya que no siempre se va a proporcionar la posición eficientemente el cien por ciento de las veces, así que la tabla de hashing debe considerar como manejar las colisiones, algunos métodos populares son el método lineal, el método cuadrática, la reasignación aleatoria y el doble hashing.

Eficiencia y Uso en Situación Problema

Gracias a la necesidad de relacionar cada ip independientemente con información, como por ejemplo donde se registran los ataques salientes y entrantes a la misma IP.

Tenemos en cuenta que a medida que hay más datos habrá más probabilidad de que el número de colisiones sea mayor, por ende si no está bien implementado el manejo de colisiones la complejidad del programa subiría drásticamente, así tomando más tiempo para llevarse a cabo.

Implementación

Para implementar la tabla de hash, el primer paso que hicimos fue convertir la dirección IP en un string simple en valores ASCII, después utilizamos el método de residuos ya que creímos que era la implementación más sencilla con mejor eficiencia, en cuanto al método de manejo de colisiones utilizamos el método cuadrático y finalmente para almacenar las IPs ordenadas usamos Priority Queue con Max Heap, así no tenemos duda que cuando ejecutemos el programa se tendrá una complejidad de O(nlogn).

Conclusión

Concluyendo con este proyecto y esta materia quiero decir que quedé muy satisfecho con el curso, aunque fue muy cargado de material me gustó mucho haber aprendido todos estos diferentes métodos y la implementación de los mismo, muchas veces he sentido en clases pasadas que nos centramos mucho en la teoría y no hacemos mucho código, pero esta clase fue un respiro de aire fresco en cuanto a eso ya que siento que mejoré mucho no solo en cuanto a mis habilidades de pensar como ingeniero, sino que también mis habilidades como programador se vieron beneficiadas gracias a este curso.

¿Cuáles son las más eficientes?

Dentro de los métodos que vimos en este curso hay unos que claramente son más eficientes que otros, pero es importante notar que no todos los programas necesariamente deben hacer uso de la implementación más eficiente el 100 por ciento de las veces, ya que para algunos programas va a ser casi lo mismo de tiempo en el que se tarde el programa más eficiente a un programa imperfecto, pero cuando se quiere escalar a que un programa maneje una base de datos como miles de valores es de suma importancia implementar el método de hashing con las tablas de hash, o implementar grafos con heap para que este programa no tarde incluso días en buscar y ordenar toda la información.

¿Cuáles podrías mejorar y argumenta cómo harías esta mejora?

En las actividades que entregamos tuvimos algunos problemas con la implementación del BST ya que pudimos haberlo implementado mejor si hubiéramos hecho uso de un maxheap en lugar de haber usado un BST normal, pero más que nada mejorar la implementación de cualquier método visto en el curso, porque aunque las computadoras sean perfectas, nosotros no.

Bibliografía

S, J. I. (n.d.). CAPIT1. Retrieved November 22, 2022, from https://ccia.ugr.es/~jfv/ed1/c/cdrom/cap5/cap56.htm#:~:text=La%20búsqueda%20secuencial%20consiste%20en,array%20con%20el%20valor%20buscado

Florez, S. A. (2018, November 27). Análisis Comparativo Algoritmos de Ordenamiento. Pereira Tech Talks. Retrieved November 22, 2022, from https://pereiratechtalks.com/analisis-de-algoritmos-de-ordenamiento/

Sharma, A. (2022, November 15). Difference between a singly linked list and a doubly linked list. PrepBytes Blog. Retrieved November 26, 2022, from https://www.prepbytes.com/blog/linked-list/difference-between-a-singly-linked-list-and-a-doubly-linked-list/#:~:text=Accessing%20elements%20in%20a%20Doubly,list%20is%20O(n).

Binary search trees. CS 225 | Binary Search Trees. (n.d.). Retrieved November 27, 2022, from https://courses.engr.illinois.edu/cs225/fa2022/resources/bst/

Most common binary tree interview questions & answers [for Freshers & Experienced]. upGrad blog. (2022, November 23). Retrieved November 27, 2022, from https://www.upgrad.com/blog/binary-tree-interview-questions-answers/

Heap sort. GeeksforGeeks. (2022, September 22). Retrieved November 27, 2022, from https://www.geeksforgeeks.org/heap-sort/

Orlando. (2021, February 11). El Rol de los Grafos en la Ciberseguridad: Ciberseguridad. GraphEverywhere. Retrieved November 22, 2022, from https://www.grapheverywhere.com/el-rol-de-los-grafos-en-la-ciberseguridad/#:~:text=La%20Ciberseguridad%20también%20es%20Big%20Data&text=Para%20cubrir%20de%20forma%20fiable,correcta%20con%20el%20contexto%20adecuado.

Tema: Tablas Hash. - UDB. (n.d.). Retrieved November 25, 2022, from https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-con-estructuras-de-datos/2020/i/guia-8.pdf