Programación de estructuras de datos y algoritmos fundamentales TC1031

REFLEXION FINAL

Nicolas Aguirre v. A00832772

Profesor

Dr. Eduardo Arturo Rodríguez Tello

20 de Junio de 2022

Reflexión Final TC1031

Durante el curso de Programacion de estructura de datos y algoritmos fundamentales TC1031, pude aprender muchas estructuras de datos y algoritmos nuevos los cuales son de gran utilidad y durante mi carrera profesional los usaré mucho para cada proyecto que tenga que realizar. Al entender estos algoritmos, pude aprender lo importante que es tener un programa con una buena eficiencia, esto se logra al usar, de la mejor forma, estructuras de datos y algoritmos. Entre más eficiente sea un programa, mucho mejor para el proyecto ya que esto hace que el programa sea más portable y corra más rápido. Cada evidencia me dejó nuevos conocimientos y al integrar todo lo aprendido en el proyecto final, pude notar cada detalle aprendido al aplicarlo al programa para poder detectar ataques en redes ciberneticas.

Reflexiones:

1.3) Es importante saber el correcto uso e implementación de los diferentes algoritmos de ordenamiento y búsqueda ya que para cada problema o situación se debe usar el mas correcto. Algunas veces es mas eficiente usar uno que el otro ya que los tiempos y uso de recursos pueden variar. Los algoritmos de ordenamiento son de los mas usados ya que estos hacen la manera mas sencilla que es quitando el elemento más pequeño y poniéndolo de primero y luego el segundo mas pequeño y se pone después del primero y asísucesivamente. De igualmanera los algoritmos de búsqueda son muy utilizados y útiles ya que casi siempre se necesita buscar elementos o datos en un programa y pues esta es la mejor manera de hacerlo. Para nuestro desarrollo del proyecto, estos algoritmos nos van a ayudar mucho para ordenar cierta información durante la realización de este mismo. Gracias al conocer estos algoritmos podremos determinar el más acertado y eficiente para cada situación.

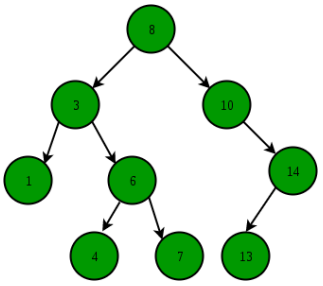
Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

2.3) Una estructura de datos es una forma particular de organizar los datos en una computadora para que pueda usarse de manera efectiva. La idea es reducir las complejidades de espacio y tiempo de las diferentes tareas. Entre las estructuras de datos lineales se pueden encontrar varias como arrays, linked lists, stacks y queue.
Cada una de estas estructuras de datos nos ayudan a resolver diferentes tipos de problemas y dependiendo de el caso es mejor usar una que otra debido a los tiempos de eficiencia dependiendo de su complejidad. En este avance del proyecto hicimos uso de listas doblemente ligadas. Gracias a ellas pudimos manejar la información más fácil ya que en esta estructura de datos se puede acceder a cada dato de una manera mas sencilla y eficiente. Para recorrer esta estructura de datos solo se hace uso de nodos los cuales apuntan ya sea al siguiente dato o al anterior. Gracias a esto, se puede repasar una lista mas facil y se puede buscar informacion de una manera mas sencilla y eficiente.
La siguiente tabla muestra los tiempos de complejidad de cada operacion que se puede hacer en una doubly linked list:

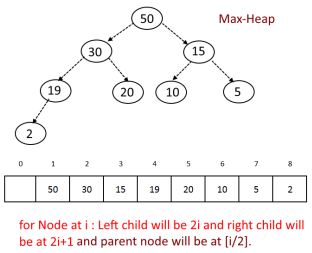
Operación	**Complejidad: Peor caso**	**Complejidad: caso promedio**
insertar al inicio o final	O(1)	O(1)
eliminar al inicio o final	O(1)	O(1)
buscar	O(n)	O(n)
acceder	O(n)	O(n)

3.4) En la ciencia de datos, tener una buena organización y distribución de los datos, es muy importante y de alta relevancia ya que de esto depende la eficiencia del programa y qué tan bueno puede ser. Aquí es donde entran los BST; estas estructuras de datos son unas de las mas importantes y simples para estas estructuras con miles de datos.



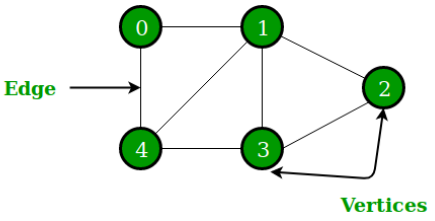
En un árbol de búsqueda binaria, el subárbol izquierdo tiene los valores mas bajos que el nodo y el subárbol derecho tiene valores mayores que el nodo. En este se pueden hacer 3 tipos de recorridos: post-order, pre-order y inorder. En esta búsqueda su eficiencia esta entre [log2(N+1)] y N.
Gracias a su estructura ordenada, los bst se pueden usar para muchas aplicaciones como indexing y multi-level indexing, aplicar algoritmos de búsqueda, mantener ordenada la data, etc.

Igualmente, el MaxHeap es un bst muy importante y fácil de usar y gracias a su fácil acceso se puede tener una alta eficiencia. Este nos ayuda a tener un fácil acceso al elemento más grande de una lista. Posteriormente, se puede eliminar este item y el maxheap se reorganiza automáticamente y se vuelve a poner el elemento mas grande siguiente. Un heap es básicamente un binary tree pero ordenado linealmente en un array. Gracias a su fácil acceso al elemento mas grande, su eficiencia es de O(1).

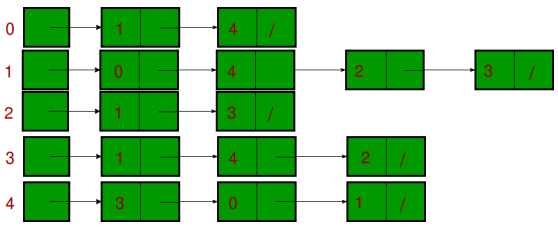


Gracias a estos conocimientos de estructuras de datos, se puede hacer un análisis o un algoritmo para determinar si una red esta infectada al implementar un conteo de los registros. Ya que cada que alguien entra a una red, se crea un registro, se puede crear un contador de los registro, y si hay muchos registros con errores al entrar en un corto periodo de tiempo o que se vean sospechosos, se puede determinar si se está siendo atacado o no.

4.3) Los grafos son una estructura de datos la cual consiste de 2 elementos principales: los vértices y aristas. Estos grafos se pueden representar de dos maneras: lista de adyacencia y matriz de adyacencia. Con los grafos se puede organizar información y datos de una manera conectada como una red. Esto nos ayuda a crear redes y organizar datos conectados. La forma en la que se ordena y se guardan los grafos es de gran ayuda y esto lo hace eficiente.

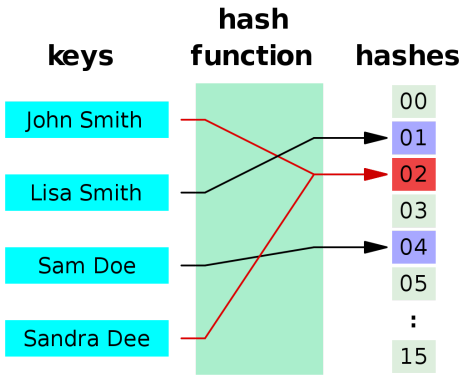


En esta actividad se organizaron las ips por una lista de adyacencia, lo cual guarda espacio ya que su eficiencia es de O(|V|+|E|) (v: vértices, e: aristas/edges). De igual manera para encontar las ips con mayor adyacencias o más vecinos, usamos un maxheap el cual tiene una eficiencia de O(1), lo cual es un tiempo muy eficiente para nuestro trabajo.



El uso de grafos nos facilitó organizar las ips de una manera más fácil y cómoda ya que estas quedan direccionadas a las ips que intentan acceder o atacar y quedan con su valor en la arista. Gracias a esto podemos ver cuáles son los posibles bots que están atacando nuestra red ya que estos serían los que tiene más incidencias en la red. Teniendo un grafo con informacion, pudimos determinar cual era el boot master y asi conocer el comportamiento de las ips. Tambien se usó el algoritmo shortestPath para encontrar los recorridos mas cortos entre una ip inicio y destino y etso se realizo con una Complejidad de O(ElogV).

5.2) Las tablas hash son una estructura de datos que usa para almacenar pares de llaves y valores. Estas tablas hash usan una funcion *hash* la cual se usa para calcular un indice en la tabla en donde el elemento va a ser insertado o buscado. El tiempo promedio de complejidad al manipular la tabla hash (como insertar datos, buscarlos o eliminarlos) es de O(1) ya que esta crea la llave en O(1) y accede a esa celda o bucket en O(1) para insertar el dato, de igual manera para borrarlo. En el peor de los casos puede ser O(n). (Garg, n.d.)



Al tener muchos valores, puede suceder que el bucket que la función *hash* calcula para el elemento, ya este ocupada. Cuando esto sucede, se le llama colision y la funcion debe encontrar otro bucket que no este ocupado. Por ende se debe tener un manejo de colisiones. El manejo de colisiones puede verse con hashing abierto o cerrado. Para el hashing abierto se usa encadenamiento separado en el cual se almacenan los datos con ayuda de una lista la cual se enlaza al indice del bucket calculado, esto se implementa al hacer una funcion llamada *chaining* complejidad O(n) (geeksforgeeks, 2022). En el hashing cerrado, se almacenan los elementos en la misma tabla al buscar el siguiente bucket libre, esto se implementa con la una funcion llamada *quadratic*. La funcion *quadratic* tiene un tiempo de complejidad O(N * L), donde N es la longitud del vector de datos que le demos y L es el tamaño de la tabla hash. (GeeksForGeeks, 2022).

Gracias al hashing, podemos almacenar los datos de las ips de una forma muy eficiente ya que podemos acceder a los datos de las ips de una manera muy rapida ya que a cada ip se le asigna un bucket en especifico. Al tener todas las ips con su información en cada bucket individualmente, podemos acceder a los datos más importantes de las ips como: las aristas saliendo, aristas entrando, y las ips a donde se dirige cierta ip. El uso del hashing es muy bueno para poder almacenar mucha información y poder accesar muy rapido y eficiente a ella. En este caso, se almacenaron todas las ips y su informacion al transformarlas a un valor entero y luego hacer uso de la función hashing con manejo de colisiones quadratic para así poder acceder en O(1) a cualquier información de cualquier ip, lo cual es una gran ayuda para tener un programa muy eficiente y rapido.

Conclusión:

Para finalizar, puedo decir que este curso fue de gran desarrollo en mi carrera profesional ya que pude aprender muchas cosas nuevas como las anteriormente mencionadas. Todas estas estructuras de datos y algoritmos serán sin duda alguna la base para todos mis siguientes proyectos ya que la aplicacion de estos es indispensable para un buen trabajo. Aparte de todos estos algoritmos y estructuras de datos, me llevo lo importante que es hacer un programa eficiente y lo mucho que afecta a la eficiencia cada linea de codigo que uno escribe.

En cada una de las actividades integradoras fuimos haciendo mas eficiente el programa y cada vez nos percatamos de nuevos posibles cambios para hacer que el programa sea lo más eficiente posible. El resultado final del proyecto; la evidencia 5.2, creo que es la mas eficiente en mi opinion ya que usamos estructuras de datos muy eficientes como listas ligadas, grafos con listas de adyacencia, hashing, etc. Cabe decir que cada programa siento que se puede mejorar un poco ya que siempre existen mejores maneras de manejar datos. Creo que una de las mejoras sería usar hashing abierto con encadenamiento externo ya que este es una manera mas eficiente de manejar las colisiones al implementar la tabla hash en el programa.

Bibliografía

GeeksForGeeks. (n.d.). Sorting Algorithms. GeeksforGeeks. Retrieved June 20, 2022, from https://www.geeksforgeeks.org/sorting-algorithms/

Programiz. (n.d.). Sorting Algorithm. Programiz. Retrieved June 20, 2022, from https://www.programiz.com/dsa/sorting-algorithm

Complexity for doubly linked lists - PHP 7 Data Structures and Algorithms [Book]. (n.d.). O'Reilly Media. Retrieved June 20, 2022, from https://www.oreilly.com/library/view/php-7-data/9781786463890/c5319c42-c462-43a1-b33d-d683f3ef7e35.xhtml

Doubly Linked List | Set 1 (Introduction and Insertion). (2022, June 14). GeeksforGeeks. Retrieved June 20, 2022, from https://www.geeksforgeeks.org/doubly-linked-list/

Guide, S. (2022, May 31). Overview of Data Structures | Set 1 (Linear Data Structures). GeeksforGeeks. Retrieved June 20, 2022, from https://www.geeksforgeeks.org/overview-of-data-structures-set-1-linear-data-structures/

What are linear data structures? (n.d.). Educative.io. Retrieved June 20, 2022, from https://www.educative.io/answers/what-are-linear-data-structures

Árbol binario de búsqueda. (n.d.). Wikipedia. Retrieved May 23, 2022, from https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%

BAsqueda AVL Tree And Heap Data Structure In C++. (2022, May 4). Software Testing Help. Retrieved May 23, 2022, from https://www.softwaretestinghelp.com/avl-trees-andheap-data-structure-in-cpp/

Importance of Binary Search Trees | by Michael Skotar. Retrieved May 23, 2022, from https://medium.com/@michaelskotar/importance-of-binary-search-treesd354afc6e347

When would I want to use a heap? (n.d.). Stack Overflow. Retrieved May 23, 2022, from https://stackoverflow.com/questions/749199/when-would-i-want-to-use-a-heap

Educative. (n.d.). Min Heap vs. Max Heap. Educative.io Educative.io. Retrieved June 14, 2022, from https://www.educative.io/answers/min-heap-vs-max-heap

GeeksForGeeks. (2022, May 7). Graph and its representations. GeeksforGeeks. Retrieved June 14, 2022, from https://www.geeksforgeeks.org/graph-and-its-representations/

Educative. (n.d.). What is chaining in hash tables? Educative.io. Retrieved June 19, 2022, from https://www.educative.io/answers/what-is-chaining-in-hash-tables

Garg, P. (n.d.). Basics of Hash Tables Tutorials & Notes | Data Structures. HackerEarth. Retrieved June 19, 2022, from https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/

GeeksForGeeks. (2022, February 8). Quadratic Probing in Hashing. GeeksforGeeks. Retrieved June 19, 2022, from https://www.geeksforgeeks.org/quadratic-probing-in-hashing/

geeksforgeeks. (2022, May 26). Hashing | Set 2 (Separate Chaining). GeeksforGeeks. Retrieved June 19, 2022, from https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/