Programación de estructuras de datos y

algoritmos fundamentales (Gpo 14)

Act 6.2 - Reflexión Final de Actividades Integradoras de la Unidad de Formación TC1031 (Evidencia Competencia)

Equipo:

Alejandro Daniel González Carrillo A01570396

Profesores:
Luis Humberto González Guerra

Monterrey, Nuevo León. 1 de diciembre de 2020.v

1. Cuales son las mas eficientes?

Tomando en cuenta todo lo aprendido en clase, se puede decir que no existe una mas eficiente sino que la mejor se acompla a la situacion a solucionar. Ya que si nos fueramos por la complejidad no se pudiera decir cual es la mejor debido a que las estructuras de datos no tienen complejidades sino sus funciones, es por esto que dependemos del problema para elegir la mejor opcion.

De acuerdo con los problemas resueltos se podria decir que la mejor y mas eficiente fue la de hashmaps. Debido a su gran eficiencia y su baja complejidad en sus funciones para ordenar sin necesidad de utilizar sort.

2. Cuales podrias mejorar y argumenta como haria esta mejora?

En mi caso mejoraria la actividad 1 debido a que nuestra solucion utilizo algoritmos muy ineficientes por lo cual utilizaria merge sort y binary search para esa actividad. Considero que lo mejor que puedo hacer para mejorar ahorita es buscar como aplicar diferentes estructuras de datos juntas y seguir investigando para continuar fundamentando mis bases. Y la manera seria a traves de pequeños proyectos con los cuales puedan aplicar.

Tambien algo que considero muy importante que pudiera ayudarme a mejorar en estructura de dtaos y algoritmos es aprender de programacion competitiva y practicar los nuevos conceptos, ya que mientras mas cosas dificiles aprenda lo dificil se vuelve facil.

Actividad 1

Los algoritmos son un conjunto sistematico de operaciones que nos permiten hacer calculos y hallar soluciones mas eficientes a diferentes tipos de problemas. Esta actividad no tenía mucha dificultad, sin embargo nos fuimos con la finta y elegimos las 2 opciones más sencillas de aplicar sin tomar en cuenta la complejidad y eficiencia. Por lo tanto, decidimos que si se mejorara el proyecto utilizaríamos binary search e merge sort. Debido a que el worst case de merge sort es O(n log (n)) mientras que el de bubble sort es O(n^2). En vez de sequential search (O(n)) sería binary search (O(log n)), debido a que puede ser casi igual de fácil implementar y su complejidad es muy similar con algoritmos más complicados de implementar.

Es importante que utilizemos algoritmos eficientes ya que en este caso no se noto mucho pero cuando son miles de datos el codigo puede llegar a tardarse minutos en correr y cuando mejoramos la eficiencia podriamos reducirlo a segundos.

Actividad 2

Durante el desarrollo de este proyecto, el cual era similar al pasado sin embargo, en este teníamos que utilizar Doubly Linked Lists para el almacenamiento y manejo de datos. Una de las cosas o desventajas que vi al momento de que utilizáramos Double linked list es que es mas lenta al momento de correr. Sin embargo, al utilizar un DLL a comparación de una SLL es que ahora tenemos un pointer prev el cual nos permite regresar a valores anteriores. Esto nos brinda muchas ventajas con la eficiencia del código. Primero que nada podemos atravesar en ambas direcciones la DLL permitiéndonos movernos libremente dentro de la linked list. Segundo nos podemos evitar muchos problemas al momento de querer ubicar un valor dado por el usuario. Y finalmente tercero se pueden agregar mas nodos antes de la linked list. Durante el desarrollo nomas utilizamos addFirst, addLast, sort, search, deconstructor y constructor. El addFirst nos permitía agregar un nodo siempre al inicio

de la linkedList. El addLast nos permitía agregar al final de la linkedlist un nodo. Para el sort utilizamos bubble sort ya que fue recomendado por el profesor y es una buena opción para lo que tratamos de hacer que es comparar valores. Y para el método search utilizamos una búsqueda secuencial la cual iba en cada nodo y comparaba el valor de la key con los dos parámetros que eran los IPs que el usuario introducía. Para concluir puedo decir que fue difícil ya que nunca habíamos realizado una doubly linked list sin embargo pude desarrollar mas mis conocimientos respecto a los pointers. Pudimos lograr que funcionara para un txt de 30 líneas, la idea era probarlo con el de 16800 sin embargo, el programa tarda mucho.

Actividad 3

Para este proyecto se nos requirio utilizar busqueda de arboles binarios para lograr encontrar los 5 ips mas accesadas de una bitacora donde las ips ya estaban ordenas. Durante el desarrollo, lo que hicimos fue leer los datos de la bitacora ya ordenada, y mandar los datos al arbol donde serian ordenados de manera inorden converso de acuerdo a su cantidad de acceso que tuvieran. La importancia de utilizar BST en estos casos es que nos permite organizar las informacion de manera jerarquica, de la misma manera nos permite tener un manejo de memoria eficiente, ya que no reserva más memoria de la que necesita. Lo eficiente de usar BST es que podemos lograr una complejidad de O(log(n)). Como podriamos determinar si una red esta infectada o no? Considerando la cantidad de veces que se intento acceder podemos saber que esa ip esta siendo mas atacada. Para lograr nuestra entrega en el main realizamos 3 funciones el leer datos que guarda toda la informacion de la bitacora en el programa. El ipToLong transforma el formato ip a un ip sin puerto y puntos de division. El ipToString transforma el formato ip y le remueve el puerto al ip. El BST.h tiene 5 metodos el constructor, destructor, add, inOrden y printInorden. El add añade un nodo al arbol con la ip (sin puerto), ip (valor long) y cantidad de accesos. El inOrden organiza de manera inorden converso los datos y imprime los 5 ips mas accesadas.

Actividad 4

Para el desarrollo de esta actividad integral se requirieron conocer y emplear terminos como grafos, listas adjacentes, leer archivos, imprimir listas adjacentes y mapas desordenados. Para la funcionalidad total de programa se realizaron 4 funciones loadGraph, ipToString, printNodos_ips, printListAdj. El loadGraph lo que hace es extraer la informacion del archivo txt y lo guarda tanto en un mapa desordenado y en una lista de adjacencia. Mediante referencia utilizamos los datos obtenidos y utilizamos ambas funciones printNodos_ips y printListAdj para imprimir.

Porque utilizar grafos para este estilo de situaciones? Debido a que nos permiten estudiar la interrelacion entre los datos que se estan estudiando. Por el lado de la eficiencia el uso de grafos con mapas desordenados nos permitio realizar lo que nos pedian en complejidad lineal O(n). Finalmente nos pudimos dar cuenta que el bootmaster lo podiamos relacionar con la cantidad de outdegrees que tuviera un nodo y asi hayamos la solución.

Actividad 5

Durante el desarrollo de este ultimo ejericicio integrador realizamos una busqueda que de acuerdo a un id que en este caso era el ip desplegara la informacion de todos los logs que tuvieran esa misma ip. El hashmap permite relacionar datos con valores, la idea del hashmap es no tener valores

con el mismo id. Entonces lo mas eficiente es usar hashamp o mapas cuando puede haber una relacion entre id y valor. El hashmap puede funcionar como una estructura de busqueda como se utilizo en la actividad. De la misma manera el hashmap inserta los valores de manera aleatoria. La complejidad de un mapa es O(log n) ya que esta basado en un arbol balanceado, mientras de que la de un mapa desordenado es constante ya que se insertan de manera aleatoria.

Para el codigo se leyeron datos de un archivo y se fueron asignando los ips como id y los valores de cada id como la fecha y problema en manera de estructura. Luego se imprimo la informacion de manera que la id considiera al momento de iterar sobre el hashmap. Tambien era necesario imprimir el size del hashmap para conocer la cantidad de accesos o la cantidad de problemas que habia en un ip.

Cuenta de Github: https://github.com/AlejandroGC