Tecnológico de Monterrey

Materia

Programación de estructuras de datos y algoritmos fundamentales (Gpo 4)

Act 6.2

Profesor:

Dr. Jesús Guillermo Falcón Cardona

Franco Sotomayor Casale -A00831450

Fecha de entrega
29 Noviembre del 2021

Indice

Act 1.3

Importancia y eficiencia de diferentes algoritmos de ordenamiento y búsqueda.

Al comenzar la actividad 1.3 era difícil poder encontrar una idea o manera de poder almacenar los tipos de datos y poder ordenarlo todo al mismo tiempo. Al completar la actividad es mucho más claro lo importante y eficiente que son estos algoritmos de ordenamiento y búsqueda. Reconocer esto ya que al iniciar la actividad originalmente había usado el algoritmo de inserción. Al correr el programa se tardaba medio minuto para poder completar el ordenamiento completo de la bitácora.txt. Al cambiarlo para el algoritmo de Merge la diferencia era muy aparente. La diferencia entre ambos algoritmos era muy evidente ya que este medio minuto cambió hacia pocos segundos. Esto me ayudó a pensar y entender lo importante que es que los algoritmos sean eficientes, ya que probablemente existen bases de datos mucho más grandes de los que tenemos en este archivo .txt el usar un algoritmo inapropiado podría llevar a que tomen segundos o incluso minutos en poder acabar de hacer su propósito. Es importante reconocer que en el mundo real, es necesario escoger el algoritmo correcto ya que esos pocos minutos o segundos de diferencia podría crear problemas grandes.

Explicación de diferentes algoritmos de ordenamiento y búsqueda para esta situación problema, así como la complejidad computacional de cada uno de ellos.

Los algoritmos de ordenamiento y búsqueda que se usaron fueron Merge y Binary Recursive Search. El mergeSort como a su nombre divide es un algoritmo de división y conquista, la manera que sirve es que rompe el algoritmo entre dos algoritmos más pequeños hasta el punto en el cual se solucionen más fáciles. Al hacer la comparación de cada subarreglo se ordenan y se juntan después para hacer un arreglo ordenado. La complejidad computacional termina siendo O(nlog(n))

La manera en que sirve la búsqueda binaria recursiva es al igual un tipo algoritmo de división, lo que hace es que divide el algoritmo entre dos subarreglos y luego se forma recursiva. De esta manera puede reducir el espacio

cada tiempo, luego se comparan los valores de en medio para buscar un tipo de dato deseado. Si no se encuentra el dato de un lado del subarreglo se busca en el otro subarreglo, así se puede bajar el espacio para buscar. La complejidad computacional de este algoritmo termina siendo un O(log(n))

Act 2.3

El documento de reflexión deberá incluir de manera específica si el uso de un lista doblemente ligada para esta situación problema es la más adecuada o no así como sus ventajas y desventajas, así como la complejidad computacional para las operaciones solicitadas.

Al realizar esta actividad uno puede realizar la eficiencia y uso de tener una lista doble ligada. Puede simplificar el proceso de ordenamiento ya que se puede hacer un chequeo de ambos lados, en frente, y atrás. Las ventajas que podría identificar es como especifiqué anteriormente, tiene una flexibilidad extra que ayuda en la manera que sirve ya que en solamente tener un dato que identifique el siguiente, en este tenemos dos datos que identifican el siguiente y el pasado. De las pocas desventajas que uno podría identificar, podría ser que son más datos de información y al hacer una lista doble ligada la manera que funciona y el código en sí se vuelve más complejo en lógica y en práctica. Aun así, yo creo que esta diferencia se puede considerar muy pequeña haciendo que la lista doble ligada pueda ser la mejor opción.

Act 3.4

importancia y eficiencia del uso de BST en una situación problema de esta naturaleza

Como hemos podido observar y realizar a través de esta actividad es la importancia y eficiencia del uso de un BS en situaciones de la naturaleza en la que realizamos la actividad. Las propiedades que se encuentran dentro de los SST, los hacen eficientes y útiles a comparación de otros árboles binarios. Las ventajas que vienen con estos tipos de árboles incluyen: maneras fáciles de

almacenar información, hacer la insert y delete más rápidos que los Linked Lists o arrays, maneras flexibles de mover información, y son más rápidas también para acceder a información a comparación de las linked lists y arrays. En nuestra situación podemos observar que el árbol BST nos ayudó a almacenar la cantidad de veces que se accedió a un ip y conseguir ese ip en especifico. Las búsquedas se simplifican para poder conseguir los valores más altos de número de veces accedidas a un ip ya que sabemos que estas están a la derecha ya que terminan siendo los números más altos. Si se fuera a aplicar a otra situación problema de esta naturaleza podríamos identificar si es que una red está siendo accedida por un usuario inusual o desconocido y se podría actuar rápidamente en la situación ya que la manera que trabajan los BST permite acceder a esta información rápidamente.

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

El 31.63% de las computadoras en el mundo se encuentran infectadas por algún tipo de malware. Claramente podemos considerar esto como un gran problema ya que es posible que estas personas que cuentan con una computadora infectada puede no tener idea que esté infectada. A su misma vez hay muchos tipos de ataques que se pueden hacer hacia la red, uno de estos tipos de ataques se pueden detectar por la inyección de anuncios dentro de las páginas que se visitan, redireccionamientos que llevan a una instalación o descarga que nunca fue pedida. Por esto es que generalmente advierten que no entren a sitios no confiables. De las mejores o maneras más simples por las cuales se puede detectar si hay un virus dentro, los signos más comunes dentro de una red que está infectada es como el internet empieza a correr más lento de lo normal, a causa de la conexión de red se puede ingresar a la computadora haciendo que programas se caigan al alazar, búsquedas de internet se mueven por ninguna razón, sitios conocidos se ven y actúan diferente, y programas no reconocidos pueden comenzarse a ser instalados.

Act 4.3

Importancia y eficiencia del uso grafos en una situación problema de esta naturaleza

Al realizar esta actividad podemos observar las ventajas y desventajas de los grafos. Esta actividad en particular contenía una cantidad grande de información, en particular porque tiene muchas conexiones entre los varios

vértices que se pueden encontrar. Al inicio, para poder probar que funcionara el programa se podía tardar minutos para terminar de recibir los datos. Cuando pudimos hacer que las conexiones se hagan basadas en los vértices, llevamos a cabo el programa mucho más rápido que antes. Podemos ver como los grafos son una manera eficiente de almacenar datos que muestran una conexión con otra información. Un ejemplo de las maneras que se usan las bases de datos orientadas a grafos son las redes sociales y/o tiendas en línea. Basado en los datos que se consiguen y las relaciones que contienen, se pueden hacer cosas como recomendaciones personalizadas, finalmente creando una red de productos basados en personas. Se podría decir que de esta manera en la actividad pudimos encontrar aquellos que ip 's que son los boot master y conseguir todavía más información sobre cada vértice que se encuentra dentro del grafo.

Act 5.2

Importancia y eficiencia del uso de tablas hash

Una de las razones por las que las tablas hash son importantes, eficientes y recomendadas para su uso es porque su complejidad de tiempo total se encuentra en una constante O (1). Gracias a esto, trabajar con tablas hash es fácil de usar mientras se implementan en un algoritmo. Podemos compararlo con una matriz, para buscar a través de una matriz se generará una complejidad de tiempo de O (n) que muestra cómo si la estructura de datos aumenta de tamaño, también lo hace el tiempo de búsqueda. Esto muestra lo eficiente que pueden llegar a ser las tablas Hash gracias a su propiedad de incluir una llave única. También son muy útiles para almacenar datos emparejados que se utilizan mucho en la notación de objetos (JSON). Las hash tables son más rápidas comparadas a otras estructuras de datos cuando se trata de almacenar y acceder muchos valores, al hacer la situación problema ingresaba todos los valores increíblemente rápido si lo fuéramos a comparar a otras estructuras de datos.

Adecuado y suficiente para los propósitos propuestos

Al trabajar en la actividad considero que las Hash tables son más que adecuadas y suficientes para los propósitos propuestos. Si lo fuéramos a comparar con una lista ligada, el tiempo en que se tarda en encontrar los ips fue significativamente más rápido que con las Hash tables. Incluso el código para poder una Hash table comparada a una Lista ligada es mucho. Al aplicar la tabla

hacia la situación problema es fácil ver cómo podría ser usada en la vida real, ya que gracias a la propiedad de una llave única para cada ip, podíamos conseguir todas las razones por las cuales hubo una falla con esa ip.

ventajas, desventajas, y otros tipos de uso.

Algunas de las principales ventajas de las tablas hash: las tablas hash son más eficientes que los árboles de búsqueda u otra estructura de búsqueda de tablas, tienen elementos únicos. Algunas de las desventajas: las colisiones hash, se vuelven ineficaces cuando ocurren demasiadas colisiones, las tablas hash no permiten valores nulos.

Uno de los usos de las tablas hash es el sistema de archivos, cuando trabajamos en los archivos de nuestras computadoras, el hash se usa para vincular el nombre del archivo a la ruta del archivo. De esta manera, cuando uno interactúa con el sistema de archivos, puede ver el nombre del archivo y la ruta al archivo. Pero para almacenar realmente el enlace del nombre del archivo y la ruta y la ubicación física del archivo en el disco, el sistema usa un mapa que se implementa como una tabla hash. Otras aplicaciones pueden ser como verificación de contraseña, y compiladores.

Act 6.2

1. **¿Cuáles son las más eficientes?**

Al trabajar sobre la situación problema, podría decir que dependiendo de lo que se necesita hacer y cómo se incorpora la información, una estructura de datos será más eficiente que la otra. Esto lo podemos ver en la manera que la información para los definía todas las ips anteriormente, esto causaba que se tardara un poco en este proceso, pero finalmente todo tipo de estructura de dato es diferente. Para esta situación problema en específico, diría que la hashtable fue la más eficiente, la razón por la cual podemos decir que esta fue la más eficiente fue por la cantidad de líneas de códigos que se tardó en realizarse, y cuanto se tardo en realizar una búsqueda y muestra de una ip. Si lo comparamos hacia los grafos y BST podemos identificar que las hash tables requieren una cantidad mínima de código y es mejor en almacenar la gran cantidad de ips. También vemos como los grafos son mejores para situaciones donde un ip de origen y un ip de destino son definidos, gracias a su definición de aristas.

2. **¿Cuáles podrías mejorar y argumenta cómo harías esta mejora?**

Creo que se podría mejorar las listas ligadas, una de las maneras que sería útil hacer una mejora es posiblemente tratarla como una hash table. Al incorporar cada nodo con un id en específico podría ser más eficiente la búsqueda de ciertos datos y de ciertos algoritmos dentro de la lista ligada. Las funciones de esta llave única serían exclusivamente para funciones que involucren conseguir los datos a partir de un nodo, ya que si se requiere cambiar todos los nodos un método de búsqueda de lista ligada es suficiente. En lo que puede ayudar esta llave es en minimizar el tiempo de búsqueda para solamente un nodo y modificar la información solamente para este. Al trabajar con las listas ligadas me encontraba haciendo muchos While loops definiendo que corra por toda la lista ligada hasta encontrar lo buscado, por eso siento que al definir una llave única para la lista ligada podría minimizar este tiempo y minimizar el espacio que toman.