Instituto Tecnológico y de Estudios Superiores de Monterrey

Programación de Estructuras de datos y Algoritmos Fundamentales.

Elmer Osiel Avila Vargas – A00826359

Luis Humberto Gonzáles Guerra

Algoritmos de Ordenamiento y de Búsqueda

El reto inicialmente pretendió realizar el ordenamiento de una cantidad considerablemente grande datos, así como la búsqueda de un cierto rango de los mismos, para realizar dicha tarea existen diversos métodos los cuales se investigaron, comprendieron y estudiaron a lo largo de este periodo; cada uno de estos métodos es preferentemente aplicable dependiendo de la situación que se presenta y tienen un grado de eficiencia diferente, es decir, algunos son más complejos que otros, por lo que se tardan en realizar su tarea en más tiempo, por ello el análisis de cada uno de estos es importante, pues con ello tendríamos la certeza de cual aplicar. Entre estos métodos se encuentran el de intercambio, el cual consiste en una lectura sucesiva de los datos y el intercambio según la comparación; el método burbuja el cual compara un elemento con el de la siguiente posición y los recorre en dependencia de ello; la selección directa, que busca el valor más grande (o más chico) de forma directa y lo intercambia a la posición final o inicial respectivamente; entre otros. De entre los métodos de ordenamiento que se mencionaron y los demás estudiados existe uno que destaca y que fue seleccionado para su implementación en la problemática, el merge sort, ya que este tiene una complejidad y una tendencia de tiempo de ejecución mucho más baja que los demás, éste presenta una complejidad O(n log n) en el peor de los casos, mientras que los demás una complejidad cuadrática O(n^2), que a grandes cantidades de datos representa una diferencia importante entre los tiempos que tardan para lograr su objetivo. Por otra parte se analizaron 2 algoritmos de búsqueda, el secuencial y el binario; el primero de ellos con una complejidad O(n) es una simple comparativa del elemento buscado dato por dato, lo cual representa una gran cantidad de comparaciones y mayor tiempo al utilizarlo con bases de datos muy grandes; el segundo de complejidad O(log n) tiene una precondición, que los datos deben de estar ordenados, se aprovecha de esto y comienza por la mitad, procede a comparar de un lado y de otro para decidir hacía que dirección continuar su búsqueda nuevamente entre la mitad de los datos no descartados, repite este proceso hasta que encuentre el dato, por lo que va descartando una gran cantidad de elementos rápidamente. Aprovechando la naturalidad del reto propuesto se lograron implementar

los algoritmos con la menor complejidad, generando así una muy buena eficiencia al resolver la problemática.

Es importante mencionar que existe muchas situaciones en la vida real de ésta índole, donde existe una cantidad gigantesca de datos y unos pocos a seleccionar, por lo que la implementación de algoritmos de este tipo para llegar a un resultado en el menor tiempo posible es muy importante, por mencionar algunos ejemplos: las redes sociales generalmente presentan una barra de búsqueda que pretende regresar al usuario ciertos datos en dependencia de lo que escriba la persona, por lo que se tiene que aplicar algún algoritmo como los estudiados durante el periodo, de forma que eficientice esta búsqueda entre las magnitudes gigantescas de datos que manejan, y así lograr la satisfacción del cliente; o un departamento dentro de una empresa que maneja grandes cantidades de archivos de los clientes en folders apilados, para recordar su posición se tienen que registrar en un ordenador y cuando se tenga que consultar uno de ellos se implementan estos algoritmos para reducir tiempos. Es aquí donde reside la importancia de todo este tipo de métodos, pues con la cantidad de datos que se generan y registran hoy en día (big data) es vital reducir los tiempos de búsqueda para todo tipo de entidades.

Listas Doblemente Encadenadas

La lista doblemente encadenada es una estructura lineal de nodos en la cual cada uno de estos se encuentra conectado tanto hacia su siguiente posición como a la anterior, esto se logra a través de los apuntadores, lo cuales permiten que el contenido de una variable sea la dirección de otra variable, es por ello que como su nombre lo indica “apuntan” hacia una variable, de este modo se facilita el acceso a los diferentes valores ya que podemos hacer que un apuntador se mueva por las direcciones de memoria, y al obtener una de estas podemos modificar su contenido e incluso cambiar las direcciones a las que apunta, de tal manera que se nos permite manejar la memoria de una forma dinámica.

A partir de la teoría anterior ya se había analizado otra estructura, la de enlace simple hacia la siguiente posición, la cual también permite movernos entre nodos, sin embargo, podemos decir que lo lograba de una forma más rustica pues para generar un simple nodo en la última posición tenemos que movernos desde el primer nodo que tiene un apuntador inicial “head”, el cual permite llegar a cualquier lugar sin excepciones, de tal forma que para tener acceso a la posición final se tiene que recorrer toda la estructura; además en el hipotético caso de que se tuviera que devolver

en 2 posiciones no se puede ya que no hay conexiones que logren esa forma de moverse, por lo que se tendría que iniciar nuevamente un apuntador temporal desde “head”. Es aquí donde entra la importancia y las ventajas de las listas doblemente encadenadas, pues al hacer que los nodos apunten tanto hacia adelante como atrás, podemos fijar apuntadores iniciales en cualquier lugar de la estructura sobre los que podemos movernos con libertad, de este modo creamos 2 apuntadores con dicha tarea, “head” (primer nodo de la estructura) y “tail” (último nodo), con los que podemos tener un acceso directo a los valores más alejados para crear un apuntador temporal y movernos según nos convenga. Regresando al ejemplo anteriormente mencionado, crear un nodo en la última posición, podemos ingresar directamente a “tail” y desde ahí generar otro nodo con las conexiones correctas y que además cuente ahora con el apuntador base “tail”; por otro lado, en caso de querer modificar la penúltima posición ya no sería necesario recorrer prácticamente toda la estructura de nuevo, sino que con el enlace de “tail” hacia posiciones anteriores se lograría tener acceso con el uso de un solo movimiento de un apuntador temporal iniciado en la última posición.

Este tipo de estructuras hace muy conveniente el acceso a cualquier nodo, sin embargo, es preciso mencionar que su complejidad al momento de generar un cambio de nodos en la estructura aumenta bastante, pues cambiar todas las conexiones y acomodarlas de la forma correcta se puede volver algo confuso. Es por ello que en la presente actividad se implementó el método de ordenamiento burbuja, el cual es bastante simple y permite un cambio de conexiones a través de la lógica, sin mucha dificultad y sin la necesidad de generar nuevos apuntadores temporales.

Dicho todo lo anterior, es más sencillo explicar el papel de las estructuras mencionadas en un problema de la naturaleza de la actividad. Si comparamos estrictamente las 2 estructuras, es indudable que las doblemente encadenadas tienen un mejor uso debido a la movilidad y acceso que existe entre nodos, si bien es más complejo el implementar los cambios de la estructura, no es imposible, y es tarea del programador buscar la solución lógica para resolver este tipo de situaciones.

Cabe mencionar que, si bien este tipo de estructuras no es muy eficiente y funciona prácticamente igual que librerías como la de vector, pero con menos libertad, su entendimiento es muy importante ya que sienta bases hacia temas más complejos y genera un mayor pensamiento lógico para futuras estructuras, además genera una buena práctica de un tema muy importante y a menudo complejo para algunos estudiantes, los apuntadores.

Uso de Arboles Binarios de Búsqueda

Los árboles binarios de búsqueda son estructuras de datos conformadas por nodos de manera jerárquica, cuyo contenido puede variar, es decir, consiste en información enlazada en base a un dato que podemos considerar como de mayor prioridad. Estas estructuras se basan en la posibilidad de cada uno de los nodos tenga una bifurcación a dos nodos hijos, los cuales se ordenan en consideración de algún factor, de tal forma que las cantidades más pequeñas se encuentren del lado izquierdo y las mayores del derecho; esto permite agilizar las búsquedas de forma abrumadora, pues resulta que al emplear recursividad en algún método de búsqueda no se tienen que recorrer todos los datos para encontrar uno de ellos, sino que se realizan unas pocas comparaciones, debido a que ya se tienen un conocimiento previo de donde podría estar alojado el dato que se quiere encontrar. Existen diferentes tipos de árboles binarios de búsqueda, entre ellos se encuentra el tipo AVL, el cual reduce en forma significativa la cantidad de comparaciones aún más, ya que hablamos de un árbol balanceado, por ejemplo, si se tuvieran almacenados 10,000 datos, tan solo con una comparación ya se estarían descartando 5000 de ellos, esto resultaría en una complejidad logarítmica y una gran eficiencia en el tiempo.

Mencionado lo anterior es sencillo observar que los árboles binarios tienen aplicaciones muy prácticas en un sinfín de problemas de diferentes índoles, y la actividad presente es prueba de ello, en esta se empleó una estructura de tipo árbol bastante básica, la cual simplemente se enfoca en ordenar por valores grandes y pequeños como se menciona en la teoría del párrafo anterior, sin embargo, se logró que de una bitácora con cerca de 16,800 registros se realizara una estructura cuyos nodos guardaran una parte del registro dado las veces que se repetía y luego desplegar los datos con las mayores cantidades de repeticiones, en un tiempo considerablemente corto, todo ello gracias a la forma en que están organizadas estas estructuras, pues fue sencillo determinar cómo llegar rápidamente al nodo que se estaba buscando. Para la resolución del problema se empleó un despliegue de información basados en teoría existente, ya que el conocimiento previo del acomodo permite mostrar toda la información al usuario de diferentes formas, ya sea en orden ascendente, descendente, nivel por nivel (de padres a hijos), entre otras, y debido a la naturaleza del problema se empleó la impresión de datos descendente, ya que la llave era el número de repeticiones y, por ende, se mostrarían los datos con mayor cantidad de repeticiones.

Este problema representa una de las muchas aplicaciones que tienen los árboles binarios; por ejemplificar otras, se puede hablar del rol actual de la información en todo tipo de empresas, donde su dependencia a ella es tan importante que un mal manejo de esta se traduce en el fracaso y desmantelamiento. Debido a las cantidades inmensas de información que las empresas almacenan es de vital importancia emplear este tipo de estructuras, de tal manera que se reduzcan al máximo la cantidad de tiempo al realizar la búsqueda de un dato en específico.

Finalmente, cabe mencionar que la eficiencia de los árboles binarios recae en el hecho de que cada elemento añadido sigue un cierto orden, por lo que si se emplean de forma inteligente se puede dar solución a muchos problemas sin la necesidad de emplear algún algoritmo de ordenamiento; es decir, la inserción de datos es estructurada mientras que en otros casos se añade simple información en diferentes bloques y luego se ordena, por lo que podemos decir que se realizan 2 tareas a la vez, sin embargo, esto no aplica para todo tipo de situaciones, y determinar la mejor solución a la problemática, es reto del programador, es aquí donde reside la importancia de aprender diversos algoritmos de estructuración y orden.

Importancia y Eficiencia del Uso Grafos

Los grafos son redes de información o conjuntos de nodos (grupos de información) que se relacionan entre sí a través de arcos (conexiones), en pocas palabras, son estructuras de datos en una forma muy general. Existen varios tipos de grafos: dirigidos, no dirigidos, ponderados, no ponderados, con o sin ciclos, conexo e inconexos, y en dependencia a ello caen dentro de una categoría, por ejemplo, un grafo conexo y sin ciclos puede ser catalogado como árbol, cuya estructura ya fue anteriormente analizada, por otro lado, un grafo sin ciclos e inconexo es conocido como un bosque. Como estos podemos describir varios ejemplos, y cada uno de ellos tiene ventajas y desventajas específicas, por lo que su uso genera un gran abanico de posibilidades para resolver distintos problemas en todo tipo de áreas de estudio.

Dicho esto, es preciso mencionar que para la presente actividad no representaban la mejor opción para darle solución. Si bien se ajustó de tal forma que se resolviera la problemática involucrando el uso de grafos, a través de una lista de adyacencia, realmente no fue parte fundamental de la resolución final. Debido a la naturaleza del problema existía una cuestión al momento de aplicar grafos que hacía dudable su uso, pues podían presentarse múltiples casos en que una ip atacara más de una vez a una misma ip, entonces, al implementar el grafo podían suceder 2 casos:

1.- Un nodo apuntaría múltiples veces hacia un mismo nodo, lo cual era innecesario, pues no genera algún tipo de utilidad importante y en cambio representaría un grafo que no atiende realmente a la teoría. El nodo es el que contiene la información, y no el arco, por lo que en este caso pudiese haberse implementado el que solo se lea una vez el arco, y en caso de repetición que el nodo guardara la cantidad de veces que un nodo apuntaba hacia este, sin embargo, esto no representaba una forma eficiente de resolver la situación ya que existían otras formas más viables y menos complejas.

2.- Si se quería evitar este caso de repeticiones innecesarias también existía la posibilidad de emplear una matriz de adyacencia, sin embargo, esto aumentaría en gran cantidad la cantidad de memoria utilizada y de la misma forma no resolvería la problemática de una manera eficiente. Al mismo tiempo esta opción no atendía a lo que pedía el enunciado, el uso de una lista de adyacencia.

Es por ello que finalmente para dar solución puntual a las preguntas enunciadas en el problema de una forma eficiente se empleó un “unorder_map”, el cual teóricamente es una estructura de información enlazada a través de un valor clave, el cual no se repite.

Es preciso mencionar que, si bien en este caso la aplicación de grafos no era lo más conveniente, existen muchos otros en los que sí los es. Es por ello que actualmente los grafos se involucran en una gran cantidad de situaciones, por ejemplo: En los mapas, los cuales pueden ser representados de una excelente forma como grafos siendo estos ponderaros (km) y dirigidos (dirección de las calles), tienen una buena aplicación en la búsqueda de rutas cortas para un usuario, lo cual es uno de sus usos más comunes; otro caso se encuentra en las redes sociales, debido a que estas pueden representar la relación de usuarios a través de grafos donde dicha relación sea un arco y las personas los nodos, se puede emplear un algoritmo de búsqueda para el apartado común de “recomendaciones” o “personas que quizás conozcas”, ya que a través de las conexiones se puede determinar la cantidad de nodos a las que se conecta otro que no está conectado contigo, de tal manera que entre más relaciones parecidas de este tipo existan, hay una mayor posibilidad de que conozcas a la persona de dicho nodo. Finalmente, otra de las aplicaciones más importantes de la teoría de grafos y que actualmente está revolucionado el mundo entero, se encuentra en las redes neuronales, las cuales pueden ser representadas a partir de grafos y se involucran en machine learning e inteligencia artificial.

Importancia y Eficiencia del Uso de las Tablas Hash

El uso de la técnica de hashing, es decir, la implementación de tablas hash para la estructuración de datos, permite mapear información en un contenedor asociativo de 2 valores, donde uno representa una llave que conduce al otro. De esta forma, podemos alojar strings como llaves que nos conduzcan hacia estructuras o vectores de datos. El uso de un hash table genera una estructuración de datos eficaz y permite el acceso a ellos de una forma altamente eficiente, pues resulta que para una gran parte de los casos la función de búsqueda tiene una complejidad de orden constante O(1), sin embargo, en el peor de los casos dicha complejidad es lineal O(n).

Dicho esto, es preciso mencionar que, para la presente actividad y el paso final de la resolución del reto se empleó el uso de tablas hash, de forma que la llave consistía en un string cuyo valor era una ip, mientras que el valor al que conducía era un vector de pares de información (la fecha en que la ip había intentado un acceso y la causa del error), esto permitió la lectura de la bitácora entera de manera que al encontrar una ip ya mapeada, a esta simplemente se le agregaran valores en el vector. Finalmente, cuando el usuario intentara consultar la información, se desplegaría un resumen de todos los datos mapeados respecto a la ip de entrada, mientras que la cantidad de accesos se obtendría con el uso de la función size() en el vector de valores. Para todo ello se empleó unordered_map, una librería ya existente en el lenguaje cuyas funciones resultan ser bastantes eficientes y la cual genera una tabla pertinente a toda la teoría ya redactada. Es así como se logró dar orden a más de noventa y un mil líneas de información en cuestión de segundos de una forma realmente sencilla, mientras que la realización de consultas es casi de manera instantánea.

Por otro lado, si pensamos en otra forma de dar solución a la problemática que se propone para esta actividad, a simple vista podemos darnos cuenta de que otras estructuras como las listas encadenadas o algunos tipos de grafos, que fueron empleados para las actividades anteriores, realmente no corresponden o tienen una buena funcionalidad en este reto. En el caso de los árboles binarios de búsqueda de tipo AVL o Splay Tree, pudiésemos idear alguna forma de implementarlos y que resultaría con una solución bastante eficiente, sin embargo, si lo representamos visualmente podemos observar que el uso de hash table se ajusta mucho mejor al tipo de solución y estructura que requiere la actividad, mientras que la eficiencia de búsqueda también resulta ser mejor por la complejidad que presenta.

Cabe destacar que la mayoría de los lenguajes de programación de alto nivel proveen de librerías para el uso de este tipo de tablas, lo cual simplifica en gran magnitud la resolución del problema, mientras que la implementación de otros tipos de estructuras resultaría ser más compleja, tediosa y no la más conveniente.

Conclusión Final

A través de la implementación de diferentes tipos de estructuras para darle resolución a las distintas partes del reto propuesto, considero he logrado detectar las múltiples aplicaciones y funcionalidades que estas tienen, tomando en cuenta no solo su posibilidad de aplicación, si no la más conveniente dependiendo de la problemática que se quiera resolver, pues aunque presenten funcionales similares, cada una de las estructuras tiene se propia identidad y la hace más conveniente para algunos tipos de casos. Al mismo tiempo, esta comprensión permite visualizar el uso y combinación de estructuras de datos en busca de la solución más óptima.

Hablando estrictamente de la resolución del reto propuesto y, dado que ya se tenía una bitácora de información a analizar, se determinó que para su ordenamiento y la búsqueda de una dirección determinada sería bueno emplear el algoritmo merge sort y la búsqueda binaria, debido a que las complejidades de estos son muy buenas, como ya se ha mencionado anteriormente. Por otro lado, para encontrar las direcciones con la mayor cantidad de accesos y errores, por ende con intensiones maliciosas, se podían implementar arboles binarios de búsqueda balanceados como lo son el AVL y el Splay Tree, los cuales igualmente presentan las mejores complejidades dando como resultado tiempos de búsqueda bajos. Por su parte, las tablas hash también representaron una muy buena opción para la resolución final, y posiblemente la mejor gracias a su complejidad constante en la mayoría de los casos de búsqueda de un elemento.

En cuanto a las estructuras de datos que se utilizaron y que se encuentran abiertas a un gran abanico de mejora debido al cómo se implementaron, se encuentran las listas doblemente encadenadas y los grafos, sin embargo, soy consciente de que aunque no son las mejores opciones para el caso que se solucionó, si lo son en otras situaciones; la primera de ellas por el hecho de que se vuelve compleja su reorganización y poco eficiente al momento de la búsqueda de un valor, si bien se pudiese mejorar trabajando en la reestructuración de enlaces al utilizar otros algoritmos más eficientes (ya que se usó bubble sort por la facilidad de implementación que provee), generaría un mayor trabajo e innecesario; la segunda estructura debido a que la matriz de adyacencia no

ayudaría de forma eficiente para contabilizar cuantas veces atacó una IP a otras misma, ya que solo te menciona si hay conexión o no entre nodos con un booleano, mientras que la lista de adyacencia aumentaría la complejidad de búsqueda en comparación con una tabla hash; la estructura de grafos se podría haber mejorado empleando una clase para el nodo, donde se guardara información de los ataques y errores, sin embargo, no sería más eficiente o fácil de implementar que una tabla de hash.

Es importante mencionar que otros de los temas vistos durante el curso que han sido de gran importancia, relacionados a la programación en general, fueron la detección de complejidad de un algoritmo y la recursividad, pues estos nos permitieron expandir nuestro pensamiento lógico además de definir y observar cuales eran las mejores maneras de atacar una problemática a través parámetros como el tiempo de ejecución, lo cual es una parte muy importante para el usuario. La relación de todo lo estudiado durante el curso nos permitió abrir horizontes hacia un nuevo estilo de programación de una forma eficaz y eficiente, tomando en cuenta siempre el aprovechamiento de los recursos, pues soy consciente de que anteriormente hubiese resuelto la problemática con el uso de vectores que, si bien son una estructura, no representan la mejor solución en este caso.