Tecnologico de Monterrey
Edgar Cruz Vázquez - A01730577
TC1031.11: Programación de estructura de datos y algoritmos fundamentales (GPO 11)

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

Repaso de los temas vistos en clase

Escribir de forma individual un documento donde expongas tu reflexión sobre el desarrollo de competencias sobre las soluciones que presentaste en las diferentes actividades integradoras:

Durante el transcurso de este curso he visto varios algoritmos para la estructuración de datos, ya sea de ordenamiento o de búsqueda, todos teniendo sus pros y contras, también aprendiendo en qué tipo de situaciones se pueden emplear y las diferencias de unas de otras, en esta parte hablaremos de los algoritmos más eficientes en cada categoría,

Ordenamiento:

En estas 5 semanas, los algoritmos para el ordenamiento de datos que logramos ver fueron:

- Quicksort
- Mergesort
- Bubble Sort
- Insertion Sort

Siendo el mergesort con la mejor complejidad de tiempo (con un promedio,mejor y peor de O(n log(n))), pero teniendo una complejidad de espacio en su peor caso de O(n), cosa que comparándola con los demás algoritmos se queda atrás ya que el quicksort en complejidad de espacio es O(log(n)) y tanto Bubble Sort como el Insertion Sort una complejidad de O(1), siendo estos últimos como los mejores ya que son constantes pero estos a diferencia del mergesort cuentan con una complejidad de tiempo promedio de O(n^2), el cual no es muy bueno, esto se pudo ver en las primeras entregas de actividades, el cual pedía imprimir todos los datos de un archivo de manera ordenada y en esa actividad se usó el algoritmo de Bubble Sort ya que fue de muy fácil implementación, solamente requiriendo unas cuantas líneas de código pero al momento de ejecutar el código está llegaba a tardar entre 6 y 8 min, el cual eso es demasiado tiempo.
En términos generales el Mergesort resulta ser la mejor opción en cuestión de eficiencia del tiempo y el Bubble Sort e Insertion Sort mejores en la complejidad de Espacio, su implementación dependerá del tamaño de datos que se esté manejando.

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)

Tecnologico de Monterrey
Edgar Cruz Vázquez - A01730577
TC1031.11: Programación de estructura de datos y algoritmos fundamentales (GPO 11)

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

Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)

Búsqueda:

En cuanto a los algoritmos especializados en búsqueda vimos:

- Búsqueda secuencial o lineal
- Búsqueda binaria

Primero iniciando por la búsqueda binaria, es el método más simple, esta metodología verifica cada elemento en una lista de forma secuencial hasta que encuentra un elemento específico, la entrada al método de búsqueda lineal es una secuencia (como una matriz, una colección o una cadena) y el elemento que se debe buscar, la salida es verdadera si el elemento especificado está dentro de la secuencia provista o falso si no está en la secuencia, como este método verifica todos los elementos de la lista hasta que se encuentra el elemento especificado, en el peor de los casos, pasará por todos los elementos de la lista antes de encontrar el elemento requerido siendo esta una complejidad de O(n), en el mejor de los casos el dato requerido se encuentra al inicio de la secuencia volviendo se O(1), esta metodología se considera que es demasiado lento para usarse cuando se buscan elementos en listas grandes, la ventaja que tiene es que es muy simple y fácil de implementar.

La metodología es utilizado más en específico en una lista ordenada, este método comienza comparando el elemento buscado con los elementos en medio de la lista, si la comparación determina que los dos elementos son iguales, el método se detiene y devuelve la posición del elemento, en caso que el elemento buscado es mayor que el elemento central, vuelve a iniciar el método utilizando la mitad inferior de la lista ordenada, en caso contrario en donde el elemento buscado sea menor que el elemento central, vuelve a iniciar el método utilizando solo la mitad superior de la lista y así sucesivamente, en caso que el elemento buscado no esté dentro de la lista, el método devolverá un valor único que lo indica, por lo tanto, este método de búsqueda binaria reduce a la mitad el número de elementos comparados, en consecuencia la búsqueda binaria se ejecuta en tiempo logarítmico, lo que resulta en un rendimiento promedio de caso O(log n) y en mejor de los casos es que el dato a buscar se encuentre exactamente en el centro de la lista volviéndose O(1), por lo tanto la búsqueda binaria resulta mejor.

Estructuras:

Durante la materia tabien vimos estrucutras de datos, los cuales fueron:

- Linked list
- Hash table
- Árboles binarios
- Grafos

Tecnologico de Monterrey
Edgar Cruz Vázquez - A01730577
TC1031.11: Programación de estructura de datos y algoritmos fundamentales (GPO 11)

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

En caso que un dato se desee buscar la estructura el cual es buena en eso serian las tablas hash, ya que esta se separa en 2 partes, una siendo la clave que de ahí se obtiene el indie, y la tabla en donde están los datos, para la obtención de cierto dato guardado dentro de la tabla, se pasa el dato el cual se desea buscar, dentro del dato se hace la operación para obtener el índice para usarlo en la tabla y encontrar el dato en especifico, entonces en cuanto a búsqueda la tabla hash tiene un promedio de O(1), haciéndola constante y de las mejroes, en caso que la tabla se ocupe la forma de ordenar los datos tipo chain, para la resolución de las colisiones, su búsqueda se puede convertir en O(n), lo cual lo puede tardar más, estas estructuras se usan para la aceleración de búsqueda, registro y eliminación de conjuntos debido a que las tres operaciones dichas todas cuentan con una complejidad de O(1).

Preguntas

- ¿Cuáles son las más eficientes?

De los trabajos entregados durante esta materia, la actividad en la cual es la más eficiente es en la 4 “4.3” grafos, en esta actividad se implementó mucho de los conceptos aprendidos en clase y también el ahorro de tiempo fue muy bueno, se implementó el quicksort, pasar parámetros por referencia, recursividad, linked list, struct, apuntadores, la forma en la cual al momento de correr era rápido, esto debido al uso adecuado de apuntadores, la facilidad en la cual se podía acceder a los datos en cada nodo, la actividad integradora 4 fue la más eficiente por sobre todas.

- ¿Cuáles podrías mejorar y argumenta cómo harías esta mejora?

De las actividades integradoras, el primero que se podría mejorar sería la primera actividad “1.3”, ya que para obtener las líneas del archivo y de ahí ordenarlas, se hicieron varias funciones en donde recibían un vector string o un vector de vectores string y devolvían vectores, el problema radica que en ninguna función sus parámetros se pasaron por referencia, haciendo que en cada función se vayan creando copias, ocupando más espacio ya que al momento de pasar por referencia la modificación que se le haga será al parámetro recibido y así evitamos copias extras y también se ahorra memoria, otro de los problemas a resaltar es el ordenamiento ya que en el primer proyecto para ordenar todas las líneas se usó el Bubble Sort y sabemos que si bien la complejidad de espacio es constante O(1) la complejidad de tiempo puede llegar a ser O(n^2) haciendo que tarde demasiado en ejecutarse, al momento de buscar los rangos a imprimir se utilizó la busqueda secuencial haciendo que esta pueda tardar al momento de iterar ya que esta puede llegar a ser O(n), entonces los cambios que se pueden hacer para este proyecto serían

- Cambiar los parámetros por referencias.
- Utilizar otro método de ordenamiento (como Quicksort o mergesort).

Tecnologico de Monterrey
Edgar Cruz Vázquez - A01730577
TC1031.11: Programación de estructura de datos y algoritmos fundamentales (GPO 11)

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

- Cambiar la búsqueda secuencial por la búsqueda binaria.

Para la actividad integradora 2 “2.3”, surge el mismo caso, usando el Bubble sort cómo algoritmo para ordenamiento, en cuestión de parámetros aquí si se implementó la referencia por lo que a diferencia del primero este ocupa menos espacio, se sigue teniendo el mismo problema de búsqueda por lo cual esta debería cambiarse por la búsqueda binaria.

- Cambiar el método de ordenamiento
- Cambiar la búsqueda secuencial por la búsqueda binaria.

En el caso de la tercera actividad integradora “3.4” en donde se toca el tema de árboles binarios, esta no está bien implementado ya que para este caso se usó un árbol con la estructura BST, el objetivo de este árbol era encontrar los 5 ips que más se repetían, sabiendo que el valor mayor siempre se encontraría en la última posición de la derecha entonces se usó esa función 5 veces para obtener las 5 ips, pero lo malo de la implementación es que el árbol no está balanceado, por lo cual al momento de buscar el dato, esta se vuelve O(n), haciendo que al momento de correr el programa, llegue a tardar, para solucionar esto y ahorrar tiempo se podría implementar un AVL Tree, haciendo que el árbol siempre se encuentre balanceado y que al momento de buscar las ips con mayor repeticiones ahora este se volvería O(log(n)), haciendo mejor el objetivo de la actividad.

- Cambiar la estructura del árbol BST a uno AVL.

Al final en la última actividad “5.2” se podría decir que está bien implementada pero el problema viene al momento de insertar los datos a la tabla y también al buscar un dato, para poder solucionar esto sería crear una tabla con más espacios o también al momento de crear una llave se podría aplicar algún tipo diferente de conversión o suma antes de obtener la llave por el método de módulo.

- Hacer más grande la tabla
- Cambiar la forma de obtener la llave para conseguir el índice en donde se insertaran los datos y así evitar colisiones

Con esto último tratando de mejorar la complejidad al momento de buscar tratando de volverla O(1) lo más posible.