Nombre: Salet Yasmin Gutierrez Nava

Practica 1

Parte 1:

Piensa en una situación donde hayas utilizado una estructura de datos convencional como un array, una lista enlazada o una pila. Detalla sobre los beneficios y las limitaciones de esa estructura en ese contexto específico. ¿Por que eligiste esa estructura de datos?

R.- En donde yo utilice fue en la materia de Programacion II donde se realizo un banco y se hizo entre grupos de 3 personas, lo cual primero nos dieron las ordenes que es lo que se tiene que ver dentro de la aplicacion, lo cual de acuerdo a eso teniamos que ver que es lo mejo para implementar.

Al principio no teniamos en claro que estructura de datos se utilizaria entonces cambiamos a cada rato, pero despues vimos que la cola con prioridades es la mas conveniente por el mismo motivo de las prioridades que nos dieron como restricciones.

Nos trajeron vario beneficios ya que al querer hacer los depositos y retiros de acuerdo a que edad y si tenian alguna discapacidad u otros detalles mas entonces se tendria en espera o se atenderia de manera mas rapida.

Compara y contrasta las características, ventajas y desventajas en términos de tiempo de ejecución, capacidad de almacenamiento y facilidad de implementación de un arraylist y un linkedlist

R.- A continuacion se mencionara las características, ventajas y desventajas en el ArrayList y LinkedList.

Característica	ArrayList	LinkedList
Implementación	Se basa en arreglos dinámicos.	Se basa en nodos enlazados.
Ejemplo de Declaración	Su estructura es: ArrayList<Integer> arrayList = new ArrayList<>();	Su estructura es: LinkedList<Integer> linkedList = new LinkedList<>();
Ejecución	Su acceso aleatório es más rápido lo cual es por índice O(1).	Su acceso aleatorio es lento (O(n)), y el acceso secuencial es rápido (O(1)).
Capacidad de Almacenamiento	Requiere un tamaño inicial, se expande automáticamente al sobrepasar la capacidad.	No requiere un tamaño inicial, crece automáticamente según sea necesario.
Inserción/eliminación	No es muy eficiente en inserciones y eliminaciones frecuentes, ya que se deben desplazar elementos.	Es eficiente en inserciones y eliminaciones en el medio, ya que esto solo se actualizan punteros.

Facilidad de Implementación	Es fácil de implementar, simplemente se crea y se agrega elementos.	Un poco más complejo de implementar debido a la gestión de nodos y punteros.

Tablas de Ventajas y Desventajas

LinkedLists:

Ventajas	**Desventajas**
Inserción y eliminación eficientes en el medio de la lista.	Acceso aleatorio lento, O(n) para acceder por índice.
Tamaño dinámico, crece automáticamente según se agregan elementos.	Uso de memoria adicional debido a los punteros.
Buena gestión de memoria fragmentada.	Mayor complejidad de implementación en comparación con ArrayList.
Eficiente en la manipulación de elementos.	x
Mejor rendimiento en inserciones/eliminaciones frecuentes.	x
Ideal para aplicaciones con cambios frecuentes en el tamaño.	x

ArrayList:

Ventajas	**Desventajas**
Acceso aleatorio rápido, O(1) para acceder por índice.	Inserción y eliminación ineficientes en el medio (O(n)).
Uso de memoria eficiente en colecciones grandes y bien dimensionadas.	Requiere un tamaño inicial, puede haber desperdicio de memoria.
Fácil de implementar y usar, similar a un arreglo.	
Mejor rendimiento en recorridos y acceso por índice.	
Adecuado para aplicaciones con acceso aleatorio frecuente.	

Elige dos estructuras de datos convencionales que hayas implementado en el pasado. Compara y contrasta sus características, ventajas y desventajas en términos de tiempo de ejecución, capacidad de almacenamiento y facilidad de implementación. Reflexiona sobre en qué situaciones específicas preferirías utilizar una sobre la otra y por qué.
R.-

Característica	Array	Lista Enlazada
Estructura de datos	Los elementos se almacenan en un arreglo contiguo en memoria.	Los elementos se almacenan en nodos y cada nodo contiene un puntero al siguiente nodo.
Acceso aleatorio	El acceso a los elementos en un arreglo es rápido y en tiempo constante.	El acceso a los elementos en una lista enlazada es lento y requiere recorrer todos los nodos previos.
Inserción y eliminación	La inserción y eliminación de elementos pueden ser costosas, ya que se deben mover los elementos restantes en la memoria para mantener la contigüidad del arreglo.	La inserción y eliminación de elementos son rápidas y solo se requiere actualizar los punteros de los nodos.
Capacidad de almacenamiento	La capacidad del arreglo es fija y determinada de antemano.	La capacidad de la lista enlazada es dinámica y se puede aumentar o disminuir según sea necesario.
Espacio de almacenamiento	Si la capacidad del arreglo es mayor que el número de elementos que se están almacenando, se desperdicia espacio de memoria.	Los nodos se pueden almacenar en cualquier parte de la memoria, lo que significa que se utiliza solo el espacio necesario para almacenar los elementos.
Implementación	Las operaciones de lectura y escritura en un arreglo son muy simples de implementar y no requieren mucho código adicional.	Las operaciones de inserción y eliminación pueden requerir un código adicional para actualizar los punteros de los nodos correctamente.
Uso	Los arreglos son buenos para el acceso aleatorio y la lectura secuencial.	Las listas enlazadas son buenas para la inserción y eliminación de elementos.

Ventajas del array:
- Acceso aleatorio rápido.
- Fácil de implementar.

- Eficiente en términos de tiempo de ejecución para el acceso aleatorio y la lectura secuencial.

Desventajas del array:

- Costoso en términos de tiempo de ejecución para la inserción y eliminación de elementos.
- Tamaño fijo y determinado de antemano.
- Espacio desperdiciado si la capacidad del arreglo es mayor que el número de elementos que se están almacenando.

Ventajas de la lista enlazada:

- Inserción y eliminación rápidas.
- Espacio eficiente en términos de almacenamiento.
- Capacidad de almacenamiento dinámico.

Desventajas de la lista enlazada:

- Acceso secuencial lento.
- Difícil de implementar.
- No es eficiente en términos de tiempo de ejecución para el acceso aleatorio y la lectura secuencial.

Prefiero el array ya que es mas facil de comprender y además es lo primero que nos enseñan a todos, tambien porque tiene muchas menos desventajas y sobre todo ayuda en el tiempo que se tiene que ir almacenando la memoria.

Parte 2:

¿Qué implicaciones tiene la elección de una estructura de datos en el diseño y la eficiencia de un programa?

R.- Las estructuras de datos son formas de organizar y almacenar datos en la memoria de una computadora, y cada una tiene características específicas que pueden afectar el rendimiento del programa en términos de velocidad, uso de memoria y facilidad de implementación, por lo que se mencionará a continuacion su eficiencia:

Eficiencia en tiempo de ejecución: En tener una estructura de datos puede afectar directamente el tiempo que tarda un programa en realizar ciertas operaciones. Algunas estructuras de datos son más eficientes que otras para operaciones específicas. Por ejemplo, las listas vinculadas son excelentes para inserciones y eliminaciones rápidas, mientras que los arrays son más rápidos para acceder a elementos directamente por índice. Lo cual, las elecciones adecuadas de la estructura de datos puede mejorar el rendimiento del programa y reducir el tiempo de ejecución.

Uso de memoria: Cada estructura de datos tiene un espacio diferente de memoria. Algunas pueden requerir más espacio en memoria que otras para almacenar la misma cantidad de datos.

Complejidad del código: Diferentes estructuras de datos implican diferentes complejidades en la implementación del código. Algunas estructuras pueden requerir más líneas de código o algoritmos más complejos para realizar operaciones específicas. Esto puede aumentar la probabilidad de errores y hacer que el mantenimiento y la depuración sean más difíciles.

Facilidad de lectura y mantenimiento: Al usar la estructura de datos adecuada para representar los datos en un programa, el código puede ser más claro y más fácil de entender, lo que facilita el mantenimiento y la colaboración entre desarrolladores.

Escalabilidad: Algunas estructuras de datos pueden manejar grandes cantidades de datos sin bajar la calidad del rendimiento. Además esto ayuda cuando se tiene programas que deben manejar un conjunto de datos en constante crecimiento, por lo que, es crucial seleccionar estructuras de datos que puedan escalar de manera eficiente.

Flexibilidad: Algunas estructura de datos son más flexibles y pueden adaptarse a diferentes escenarios y requisitos, mientras que otras son más especializadas y pueden ser más eficientes para casos específicos. La elección adecuada dependerá de las necesidades particulares del programa.

¿Cuál es la diferencia entre una estructura de datos convencional y una estructura de datos avanzada?
R.- La diferencia entre una estructura de datos convencional y una estructura de datos avanzada se puede mencionar las siguientes:

- *Estructuras de datos convencionales:*

Las estructuras de datos convencionales son conocidas y utilizadas en la programación asi tambien son estructuras de datos básicas además que es fundamental para cualquier lenguaje de programación y estos se enseñan en cursos introductorios de programación. Algunas de las estructuras de datos convencionales más comunes son:

1. Arrays o arreglos.
2. Listas.
3. Colas.
4. Pilas.

- *Estructuras de datos avanzadas:*

Las estructuras de datos avanzadas son aquellas que se centran en solucionar problemas y optimizar operaciones específicas más complejas. Estas estructuras suelen estar diseñadas para abordar desafíos de rendimiento, escalabilidad y complejidad de datos en aplicaciones y sistemas más exigentes. Algunas de ellas son:

1. Árboles.
2. Grafos.
3. Tablas hash.
4. Heaps (montículos).

¿Cuáles son algunas ventajas de utilizar estructuras de datos avanzadas en comparación con las convencionales?

R.- Algunas de las ventajas vendrian ser las siguientes:

Ventaja	Estructuras de Datos Convencionales	Estructuras de Datos Avanzadas
Eficiencia en operaciones específicas	Son adecuadas para tareas básicas como inserción, eliminación y búsqueda en pequeñas cantidades de datos.	Están optimizadas para operaciones más complejas y específicas, lo que permite un mejor rendimiento en escenarios más exigentes.
Escalabilidad	Pueden tener problemas de rendimiento y escalabilidad cuando se utilizan para grandes volúmenes de datos o situaciones complejas.	Se diseñan para manejar grandes cantidades de datos y escalar de manera eficiente.
Uso eficiente de memoria	Pueden requerir más espacio de memoria, especialmente cuando se trabaja con datos dispersos o con una cantidad significativa de elementos vacíos.	Están diseñadas para optimizar el uso de memoria, lo que puede ser crucial en sistemas con recursos limitados o en entornos móviles.
Flexibilidad y adaptabilidad	Pueden tener limitaciones en cuanto a sus operaciones y tipos de datos que pueden manejar.	Ofrecen mayor flexibilidad y adaptabilidad, lo que las hace más adecuadas para resolver problemas complejos y diversas situaciones.
Complejidad de operación	Tienden a tener operaciones más sencillas y fáciles de entender debido a su naturaleza básica.	Pueden tener operaciones más complejas, pero también proporcionan algoritmos y mecanismos optimizados para resolver problemas específicos de manera eficiente.
Facilidad de implementación en situaciones comunes	Son fáciles de implementar y suelen estar disponibles directamente en los lenguajes de programación.	Pueden requerir una implementación más compleja y, en algunos casos, es posible que no estén disponibles directamente en el lenguaje, lo que puede implicar implementaciones personalizadas o el uso de bibliotecas externas.

¿Cuáles son las aplicaciones comunes de los árboles B en problemas reales?

R.- En general, los árboles B son útiles en aplicaciones que requieren acceso y búsqueda eficiente de datos, especialmente cuando se trabaja con grandes conjuntos de datos o en escenarios donde el rendimiento y la escalabilidad son críticos. Su estructura equilibrada y propiedades específicas hacen que sean una elección popular en diversas áreas de la informática, Algunas de las aplicaciones comunes de los árboles B en problemas reales son las siguientes:

1. Bases de datos.
2. Sistemas de archivos.
3. Sistemas de indexación
4. Sistemas de almacenamiento en caché
5. Sistemas de redes
6. Bases de datos distribuidas
7. Sistemas de información geográfica (SIG).
8. Bases de datos de tiempo real.
9. Sistemas de archivos de registro.
10. Sistemas de bases de datos de claves-valor.

¿Cuáles son las aplicaciones comunes de los heaps en problemas reales?
R.- Algunas de las aplicaciones comunes de los heaps en problemas reales son las siguientes:

1. Colas de prioridad.
2. Programación dinámica.
3. Algoritmos de ordenación.
4. Gestión de memoria.
5. Algoritmos de búsqueda.
6. Algoritmos de compresión.
7. Cronogramas y planificación.
8. Algoritmos de búsqueda de k-ésimo elemento más grande o más pequeño.

¿Cómo se comparan las estructuras de datos convencionales (como arrays y listas enlazadas) con los árboles B en términos de tiempo de ejecución, capacidad de almacenamiento y facilidad de implementación?
R.- A continuacion se mencionara las comparaciones que se toman en cuenta:

	Arrays	**Listas enlazadas**	**Árboles B**
Tiempo de ejecución	Brindan un tiempo de acceso O(1). Pero, la inserción y eliminación de elementos son costosas, ya que requieren mover los elementos para hacer espacio. La inserción y eliminación en cualquier posición diferente del final del array tienen un tiempo promedio de O(n).	Las listas enlazadas proporcionan un tiempo de acceso O(n) en promedio, ya que es necesario recorrer los enlaces para llegar al elemento deseado. Sin embargo, la inserción y eliminación de elementos pueden ser más eficientes (O(1)) si se tiene acceso directo a los	Los árboles B garantiza que el tiempo de acceso, inserción y eliminación tenga una complejidad logarítmica O(log n) en promedio. Esta característica hace que los árboles B sean especialmente útiles en aplicaciones donde se necesitan operaciones eficientes sobre grandes conjuntos de datos.

		nodos que se desean manipular.	
Capacidad de almacenamiento	Tienen una capacidad fija y estática, lo que significa que se debe especificar su tamaño al crearlos. Si se requiere más espacio del asignado inicialmente, es necesario crear un nuevo array más grande y copiar los elementos.	Utilizan memoria dinámica, lo que permite crecer y reducir según sea necesario, adaptándose a la cantidad exacta de elementos que se están utilizando. Sin embargo, esto introduce una sobrecarga debido a los punteros que conectan los nodos de la lista.	Utilizan memoria dinámica y pueden crecer y reducir su capacidad según sea necesario. Sin embargo, comparados con las listas enlazadas, pueden requerir una mayor sobrecarga debido a la estructura jerárquica de los nodos.
Facilidad de implementación	Son estructuras de datos relativamente sencillas de implementar y entender, ya que están presentes en la mayoría de los lenguajes de programación y su acceso es directo. Sin embargo, la gestión de su tamaño puede resultar complicada en ciertos escenarios.	Implementar listas enlazadas puede ser más complejo que los arrays debido a la necesidad de gestionar los enlaces entre los nodos. Aunque las listas enlazadas son una buena opción para ciertas situaciones, su mantenimiento y manipulación pueden requerir un mayor esfuerzo.	La implementación de árboles B puede ser más compleja que las estructuras anteriores debido a las reglas de equilibrio que deben mantenerse para asegurar que se mantenga el orden y rendimiento deseado. Sin embargo, hay bibliotecas y herramientas que facilitan su uso.

¿Cómo afecta la elección de una estructura de datos al rendimiento y la eficiencia de un programa?

R.- La elección de la estructura de datos puede tener un impacto significativo en el rendimiento y la eficiencia de un programa. Es esencial para lograr un programa que funcione de manera eficiente y pueda manejar grandes volúmenes de datos. Lo cual a continuacion se mostrara el como afecta:

- Tiempo de ejecución: Algunas estructuras permiten operaciones más rápidas que otras (búsqueda, inserción, etc.).
- Uso de memoria: Cada estructura tiene su sobrecarga de memoria, lo que afecta la cantidad de memoria que consume el programa.
- Capacidad de almacenamiento: Algunas estructuras pueden crecer o reducirse dinámicamente, mientras que otras tienen una capacidad fija.

- Facilidad de implementación y mantenimiento: Algunas estructuras son más fáciles de implementar y mantener que otras.
- Escalabilidad: La elección adecuada garantiza que el programa funcione bien con grandes volúmenes de datos o crecimiento futuro.

¿Cuáles son algunas consideraciones importantes al seleccionar una estructura de datos para manejar grandes volúmenes de datos?
R.- Al seleccionar una estructura de datos para manejar grandes volúmenes de datos, es importante tener en cuenta ciertas consideraciones para garantizar un rendimiento óptimo.A continuacion se mencionaran las consideraciones importantes a tomar en cuenta:

1. Eficiencia en el tiempo de ejecución.
2. Uso eficiente de memoria.
3. Capacidad de escalabilidad.
4. Optimización de consultas y operaciones.
5. Facilidad de implementación y mantenimiento.
6. Equilibrio entre lectura y escritura.
7. Persistencia y durabilidad.

¿Qué implicaciones tiene la elección de una estructura de datos en la flexibilidad y escalabilidad de un programa?
R.- La elección de una estructura de datos tiene implicaciones importantes en la flexibilidad y escalabilidad de un programa. Aquí están algunas de las principales implicaciones:

1. Flexibilidad:
   - Algunas estructuras de datos tienen una capacidad fija, lo que puede limitar la flexibilidad si el tamaño de los datos cambia con el tiempo. Estructuras de datos dinámicas como listas enlazadas o hash tables permiten adaptarse a diferentes tamaños de datos.

2. Escalabilidad:
   - Las estructuras de datos que permiten una distribución paralela de tareas pueden mejorar la escalabilidad en sistemas que necesiten procesar grandes cantidades de datos en paralelo.

¿Cuál es la relación entre la complejidad del código y la elección de una estructura de datos adecuada?
R.- La relación entre la complejidad del código y la elección de una estructura de datos adecuada se puede resumir en los siguientes puntos:

1. Eficiencia de las operaciones: Una estructura de datos bien adaptada a las operaciones requeridas por el programa puede mejorar significativamente la eficiencia del código.

2. Facilidad de implementación: La elección de una estructura de datos que se ajuste bien al problema que se está resolviendo puede simplificar la implementación del código.

3. Adaptabilidad a los requisitos del problema: La elección de una estructura de datos adecuada puede reflejar mejor los requisitos específicos del problema, lo que puede hacer que el código sea más claro y legible.

4. Reducción de errores: Una estructura de datos bien elegida puede disminuir la probabilidad de errores en el código. Al utilizar una estructura que se ajuste naturalmente a las operaciones necesarias, es menos probable que se cometan errores en la manipulación de datos.

5. Mantenibilidad: Una elección adecuada de estructura de datos puede facilitar el mantenimiento del código a lo largo del tiempo. Un código más claro y bien organizado, gracias a una elección de estructura apropiada, será más fácil de mantener y modificar en el futuro.