ÁREA 1. ALGORITMIA

SUBÁREA 1.2 ESTRUCTURAS DE DATOS

TEMAS IMPORTANTES:

**Estructuras de Datos Simples:**

Existen varias estructuras de datos simples que se utilizan en programación. Aquí hay algunos ejemplos:

* Arrays: los arrays son estructuras de datos que se utilizan para almacenar una colección de elementos del mismo tipo en una única variable. Los elementos en un array están indexados y se pueden acceder individualmente.
* Listas enlazadas: una lista enlazada es una estructura de datos lineal en la que cada elemento se almacena en un nodo separado, y cada nodo contiene una referencia al siguiente nodo en la lista. Las listas enlazadas se utilizan comúnmente para implementar otras estructuras de datos, como pilas y colas.
* Pilas: las pilas son estructuras de datos lineales en las que los elementos se insertan y eliminan por un extremo, conocido como la parte superior de la pila. La lógica es de tipo LIFO (Last-In-First-Out).
* Colas: las colas son estructuras de datos lineales en las que los elementos se insertan por un extremo y se eliminan por el otro, conocido como la parte trasera de la cola. La lógica es de tipo FIFO (First-In-First-Out).
* Árboles: un árbol es una estructura de datos no lineal en la que cada elemento se almacena en un nodo, y cada nodo tiene cero o más nodos hijos. Los árboles se utilizan comúnmente para representar jerarquías y relaciones de orden entre elementos.
* Gráficos: un gráfico es una estructura de datos no lineal que consta de un conjunto de nodos (vértices) y un conjunto de arcos (conexiones) entre ellos. Los gráficos se utilizan para representar relaciones complejas entre elementos.

Estas son solo algunas de las estructuras de datos más comunes y simples utilizadas en programación. Hay muchas otras estructuras de datos más avanzadas y complejas utilizadas en la práctica, como los árboles de búsqueda binaria, los hash tables y los heaps, por nombrar algunas.

**Pilas y Colas:**

Las pilas y las colas son estructuras de datos fundamentales en ciencias de la computación y programación. Ambas se utilizan para almacenar y acceder a elementos de datos de una manera específica y tienen su propia lógica y operaciones.

Una pila es una estructura de datos en la que los elementos se insertan y eliminan solo por un extremo, conocido como la parte superior de la pila. La lógica es de tipo LIFO (Last-In-First-Out), lo que significa que el último elemento insertado en la pila es el primero en ser eliminado. Las operaciones básicas que se pueden realizar en una pila son push (insertar un elemento en la parte superior de la pila) y pop (eliminar el elemento superior de la pila).

Por otro lado, una cola es una estructura de datos en la que los elementos se insertan por un extremo y se eliminan por el otro, conocido como la parte trasera de la cola. La lógica es de tipo FIFO (First-In-First-Out), lo que significa que el primer elemento insertado en la cola es el primero en ser eliminado. Las operaciones básicas que se pueden realizar en una cola son enqueue (insertar un elemento en la parte trasera de la cola) y dequeue (eliminar el elemento frontal de la cola).

Ambas estructuras de datos se utilizan comúnmente en la programación para una variedad de propósitos, como la implementación de algoritmos de búsqueda, recorrido y ordenamiento de árboles y grafos, y en la gestión de tareas en sistemas operativos y programación concurrente.

Ejemplos de pilas:

* Evaluación de expresiones aritméticas: en este caso, los operandos y operadores se almacenan en una pila y se van retirando según la prioridad de operaciones.
* Revertir una cadena de caracteres: en este caso, los caracteres de la cadena se insertan en una pila y luego se van retirando en orden inverso.
* Verificación de equilibrio de paréntesis: en este caso, se utilizan pilas para verificar si los paréntesis en una expresión están balanceados y cerrados correctamente.

Ejemplos de colas:

* Procesamiento de solicitudes en un sistema: en este caso, se utilizan colas para manejar solicitudes de procesamiento en un orden específico, en función del tiempo en que se recibieron.
* Impresión de documentos: en este caso, se pueden utilizar colas para gestionar la impresión de varios documentos, asegurándose de que se impriman en el orden correcto.
* Navegación en un árbol: en este caso, se puede utilizar una cola para recorrer un árbol en un orden específico, utilizando una estructura de árbol enlazado y agregando los nodos hijos de cada nodo en una cola para ser procesados en orden.

**Listas Enlazadas y sus Tipos:**

Las listas enlazadas son una estructura de datos en la que los elementos están vinculados mediante punteros. Cada elemento, o nodo, contiene un valor y un puntero que apunta al siguiente elemento en la lista. Los tipos de listas enlazadas más comunes son:

1. Lista enlazada simple: En esta lista cada nodo tiene un puntero que apunta al siguiente nodo. El último nodo de la lista tiene un puntero que apunta a NULL.
2. Lista enlazada doble: En esta lista cada nodo tiene dos punteros, uno que apunta al siguiente nodo y otro que apunta al nodo anterior. El primer nodo tiene un puntero anterior que apunta a NULL y el último nodo tiene un puntero siguiente que apunta a NULL.
3. Lista enlazada circular: En esta lista el último nodo de la lista tiene un puntero que apunta al primer nodo, creando así un ciclo. Esta lista puede ser simple o doble.
4. Lista enlazada circular doble: En esta lista cada nodo tiene dos punteros, uno que apunta al siguiente nodo y otro que apunta al nodo anterior. El primer nodo tiene un puntero anterior que apunta al último nodo y el último nodo tiene un puntero siguiente que apunta al primer nodo.

Las situaciones en las que se debe usar cada tipo de lista enlazada dependen de las necesidades específicas del problema que se esté resolviendo. Sin embargo, a continuación, se presentan algunas consideraciones generales para cada tipo de lista:

* Lista enlazada simple: Es útil cuando se necesita una estructura de datos para almacenar y recorrer una colección de elementos en una sola dirección. Es eficiente para agregar y eliminar elementos al principio de la lista, pero no es tan eficiente para hacerlo al final. Además, no es adecuada para búsquedas o acceso aleatorio a los elementos de la lista.
* Lista enlazada doble: Es útil cuando se necesita recorrer una colección de elementos en ambas direcciones y cuando se necesitan operaciones eficientes de agregar y eliminar elementos al principio y al final de la lista. También es útil cuando se necesita realizar búsquedas o acceso aleatorio a los elementos de la lista.
* Lista enlazada circular: Es útil cuando se necesita una estructura de datos circular en la que el último elemento se conecta con el primer elemento, como en un reloj. También es útil cuando se necesitan operaciones de agregar y eliminar elementos al principio y al final de la lista de manera eficiente.
* Lista enlazada circular doble: Es útil cuando se necesitan operaciones de agregar y eliminar elementos al principio y al final de la lista de manera eficiente, y cuando se necesita recorrer la lista en ambas direcciones. También es útil cuando se necesita una estructura de datos circular en la que cada elemento tiene un puntero tanto al siguiente como al elemento anterior.

En resumen, la elección del tipo de lista enlazada dependerá de las necesidades específicas del problema que se esté resolviendo. Cada tipo de lista tiene ventajas y desventajas que deben tenerse en cuenta al decidir qué tipo usar.

**Árboles y sus Tipos:**

Los árboles son una estructura de datos jerárquica en la que cada nodo tiene un padre y cero o más hijos. El nodo superior, sin padre, se llama la raíz del árbol. Los tipos de árboles más comunes son:

* Árbol binario: Un árbol binario es un árbol en el que cada nodo tiene como máximo dos hijos. Los hijos se denominan hijo izquierdo y hijo derecho. Los árboles binarios pueden ser de búsqueda o no.
* Árbol de búsqueda binario: Un árbol de búsqueda binario es un árbol binario en el que los valores almacenados en los nodos están ordenados, de tal manera que para cada nodo, los valores de todos los nodos del subárbol izquierdo son menores que el valor del nodo y los valores de todos los nodos del subárbol derecho son mayores.
* Árbol AVL: Un árbol AVL es un árbol de búsqueda binario en el que para cada nodo, la altura de los subárboles izquierdo y derecho difiere en un máximo de 1. Si se produce una violación de esta propiedad, se realiza una o más rotaciones para restaurarla.
* Árbol B: Un árbol B es un árbol en el que cada nodo puede tener más de dos hijos. Se utiliza para almacenar grandes cantidades de datos en disco, ya que minimiza el número de accesos a disco.
* Árbol rojo-negro: Un árbol rojo-negro es un árbol de búsqueda binario en el que cada nodo tiene un color, rojo o negro, y se mantiene balanceado mediante una serie de reglas de color y rotación.

Los árboles tienen varios algoritmos importantes asociados a ellos, entre los que se incluyen:

1. Búsqueda: Buscar un elemento en un árbol se realiza comparando el elemento buscado con el elemento en el nodo actual y, si es necesario, avanzando por el árbol hacia la rama correspondiente (izquierda o derecha) según el valor del elemento. En un árbol de búsqueda binario, este algoritmo tiene una complejidad de O(log n) en el peor caso.

Texto

Descripción generada automáticamente

1. Inserción: Insertar un nuevo elemento en un árbol se realiza encontrando la ubicación correcta para el elemento según su valor y creando un nuevo nodo en esa ubicación. En un árbol de búsqueda binario, este algoritmo también tiene una complejidad de O(log n) en el peor caso.

Texto

Descripción generada automáticamente

1. Eliminación: Eliminar un elemento de un árbol implica encontrar el nodo que contiene el elemento y eliminarlo, manteniendo las propiedades del árbol. En un árbol de búsqueda binario, este algoritmo también tiene una complejidad de O(log n) en el peor caso.

Texto

Descripción generada automáticamente

1. Recorrido en orden: Recorrer un árbol en orden significa visitar todos los nodos del árbol en orden ascendente, según los valores de los elementos almacenados en los nodos. Este algoritmo se puede implementar mediante una recursión.

Texto

Descripción generada automáticamente

1. Recorrido en preorden: Recorrer un árbol en preorden significa visitar el nodo raíz antes de visitar los nodos hijos. Este algoritmo también se puede implementar mediante una recursión.

Texto

Descripción generada automáticamente

1. Recorrido en postorden: Recorrer un árbol en postorden significa visitar los nodos hijos antes de visitar el nodo raíz. Este algoritmo también se puede implementar mediante una recursión.

Texto

Descripción generada automáticamente

1. Balanceo: Mantener un árbol balanceado es importante para asegurar una buena eficiencia en las operaciones de búsqueda, inserción y eliminación. Los algoritmos de balanceo se utilizan para garantizar que la altura del árbol sea óptima y, en consecuencia, que las operaciones sean eficientes.

Un heap, también conocido como montículo, es una estructura de datos basada en árboles que permite acceder rápidamente al elemento de mayor (o menor) valor en un conjunto de elementos.

Hay dos tipos principales de heaps: max heap y min heap. En un max heap, el valor de cada nodo es mayor o igual que los valores de sus hijos, mientras que en un min heap, el valor de cada nodo es menor o igual que los valores de sus hijos.

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Los heaps se pueden implementar mediante un árbol binario completo, lo que significa que cada nivel del árbol está completamente lleno, excepto tal vez el último nivel, que se llena de izquierda a derecha. Esto permite que la estructura de datos sea muy eficiente en términos de memoria y tiempo de acceso.

La inserción en un heap se realiza agregando un nuevo elemento al final del árbol y luego "subiendo" el elemento hasta que se coloca en su posición correcta. La eliminación del elemento de mayor (o menor) valor se realiza extrayendo el primer elemento del heap, reemplazándolo con el último elemento y luego "bajando" el elemento hacia su posición correcta.

Algoritmo de construcción de max-heap:

Texto, Carta

Descripción generada automáticamente

Algoritmo de eliminación de max-heap:

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

**Grafos**:

Características de los Grafos:

* Están compuestos por nodos y aristas.
* Pueden ser dirigidos (con flechas) o no dirigidos.
* Las aristas pueden tener peso o no.
* Se pueden representar mediante matrices o listas.
* Se utilizan para representar relaciones entre objetos o entidades.
* Componentes de los Grafos:

Nodos o vértices: son los puntos en el grafo que se conectan mediante aristas. Cada nodo puede tener un identificador único.

Aristas: son los enlaces entre los nodos. Cada arista puede tener una dirección (en grafos dirigidos) o no tenerla (en grafos no dirigidos). También pueden tener peso o etiquetas para representar una propiedad o valor asociado.

Funcionamiento de los Grafos:

Los grafos se utilizan para modelar relaciones entre objetos o entidades. En un grafo, los nodos se representan como puntos y las aristas se representan como líneas que conectan estos puntos. Los grafos se pueden utilizar para resolver una variedad de problemas en informática, como algoritmos de búsqueda, redes de computadoras, análisis de datos y optimización de rutas.

Implementación de los Grafos:

Los grafos se pueden implementar en la programación utilizando estructuras de datos como matrices o listas. Las matrices de adyacencia se utilizan para representar grafos ponderados o no dirigidos, mientras que las listas de adyacencia se utilizan para representar grafos dirigidos o no ponderados.

Tipos de Grafos:

* Grafos no dirigidos: en los que las aristas no tienen una dirección.
* Grafos dirigidos: en los que las aristas tienen una dirección.
* Grafos ponderados: en los que las aristas tienen un peso o valor asociado.
* Grafos cíclicos: en los que hay un camino que comienza y termina en el mismo nodo.
* Grafos acíclicos: en los que no hay un camino que comience y termine en el mismo nodo.

En computación, los grafos se utilizan para resolver una variedad de problemas. Algunos ejemplos de aplicaciones de los grafos en la informática incluyen:

* Algoritmos de búsqueda: Los grafos se utilizan en algoritmos de búsqueda como el algoritmo de búsqueda en anchura (BFS) y el algoritmo de búsqueda en profundidad (DFS) para encontrar caminos o soluciones a problemas.
* Redes de computadoras: Los grafos se utilizan en la topología de redes de computadoras, donde los nodos representan dispositivos de red y las aristas representan conexiones de red.
* Análisis de datos: Los grafos se utilizan en análisis de datos para modelar relaciones entre elementos de datos. Por ejemplo, en la minería de datos, los grafos se utilizan para identificar patrones y relaciones entre elementos de datos.
* Optimización de rutas: Los grafos se utilizan en la optimización de rutas para encontrar la ruta más corta o más eficiente entre dos nodos en una red.
* Representación de estructuras de datos: Los grafos se utilizan como estructuras de datos para representar y almacenar información. Por ejemplo, en la representación de un mapa de carreteras, cada intersección se puede representar como un nodo y cada carretera se puede representar como una arista que conecta dos intersecciones.

Algoritmos específicos de grafos:

Algoritmo de búsqueda en anchura (BFS): se utiliza para recorrer un grafo de manera sistemática y encontrar todos los nodos que están a una distancia determinada de un nodo inicial.

Texto

Descripción generada automáticamente

Algoritmo de búsqueda en profundidad (DFS): se utiliza para explorar un grafo en profundidad, es decir, visitando primero todos los nodos que están a una distancia mínima del nodo inicial.

Algoritmo de Dijkstra: se utiliza para encontrar el camino más corto entre dos nodos en un grafo ponderado.

Texto

Descripción generada automáticamente

Algoritmo de Kruskal: se utiliza para encontrar el árbol de expansión mínima de un grafo no dirigido y ponderado.

Texto

Descripción generada automáticamente

Algoritmo de Bellman-Ford: se utiliza para encontrar el camino más corto entre dos nodos en un grafo ponderado, incluso si hay aristas con pesos negativos.

Algoritmo de Prim: se utiliza para encontrar el árbol de expansión mínima de un grafo no dirigido y ponderado.

Algoritmo de Floyd-Warshall: se utiliza para encontrar todos los caminos más cortos entre todos los pares de nodos en un grafo ponderado.

Algoritmo de detección de ciclos: se utiliza para encontrar ciclos en un grafo.

**Archivos**:

<https://www.geeksforgeeks.org/file-organization-in-dbms-set-1/>

<https://www.geeksforgeeks.org/file-organization-in-dbms-set-4/?ref=lbp>

<https://www.geeksforgeeks.org/file-organization-in-dbms-set-3/?ref=lbp>

**Ordenamiento y Búsqueda:**

Algoritmos de ordenamiento:

Ordenamiento de burbuja (Bubble sort): compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Repite este proceso hasta que no se realice ningún intercambio en una iteración.

Texto

Descripción generada automáticamente

Ordenamiento por inserción (Insertion sort): inserta cada elemento de la lista en su posición correcta en una lista ordenada, construyendo la lista ordenada elemento por elemento.

Texto

Descripción generada automáticamente

Ordenamiento rápido (Quick sort): selecciona un pivote, divide la lista en dos sublistas: una con elementos menores que el pivote y otra con elementos mayores que el pivote. Luego aplica recursivamente el mismo proceso a cada sublista.

Texto

Descripción generada automáticamente

Ordenamiento por selección (Selection sort): busca el elemento más pequeño en la lista y lo intercambia con el primer elemento. Luego busca el siguiente elemento más pequeño y lo intercambia con el segundo elemento, y así sucesivamente.

Ordenamiento por mezcla (Merge sort): divide la lista en dos sublistas, ordena cada sublista de manera recursiva y luego fusiona las dos sublistas ordenadas en una lista ordenada.

Algoritmos de búsqueda:

Búsqueda secuencial (Sequential search): busca el elemento deseado en una lista de manera secuencial, comparando cada elemento con el elemento deseado hasta que se encuentra una coincidencia.

Texto

Descripción generada automáticamente

Búsqueda binaria (Binary search): busca el elemento deseado en una lista ordenada dividiendo repetidamente la lista por la mitad y comparando el elemento deseado con el elemento en el medio de la lista.

Texto

Descripción generada automáticamente

Búsqueda por interpolación (Interpolation search): es una variación de la búsqueda binaria que utiliza una fórmula para estimar la posición del elemento deseado en la lista, en lugar de simplemente dividir la lista por la mitad.

interpolation\_search(A, valor):

n = longitud(A)

izquierda = 0

derecha = n-1

mientras izquierda <= derecha y valor >= A[izquierda] y valor <= A[derecha]:

posicion = izquierda + int((float(derecha - izquierda) / (A[derecha] - A[izquierda])) \* (valor - A[izquierda]))

si A[posicion] == valor:

devolver posicion

sino si A[posicion] < valor:

izquierda = posicion + 1

sino:

derecha = posicion - 1

devolver -1