SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE MINATITLÁN

INGENIERÍA EN SISTEMAS COMPUTACIONALES

"MANUAL DE PRÁCTICAS "

MATERIA

ESTRUCTURA DE DATOS

MINATITLÁN, VER. AGOSTO DEL 2023

3.2 ÍNDICE DEL MANUAL DE PRÁCTICAS

ÍNDICE

3.1 INTRODUCCIÓN

El presente manual dará a conocer las prácticas relacionadas con los temas de la materia Estructura de Datos, los cuales están divididos en 11 prácticas con respecto al temario de la materia:

1) 1. Creación de TDA's.
2) Uso de memoria estática y dinámica.
3) Análisis de complejidad algorítmica: en tiempo y espacio. (Mejor, promedio y peor caso.
4) Conversión de algoritmos iterativos a recursivos.
5) Serie de ejercicios con Recursividad indirecta o cruzada, múltiple.
6) Operaciones básicas con pilas y colas de manera estática, usando arreglos.
7) Reutilización de clases definidas para pilas, colas y listas enlazadas dinámicas.
8) Operaciones con árboles: Creación, inserción, eliminación y búsqueda en a.b.b, conversión de árboles generales a binarios. Y sus Recorridos sistemáticos: Preorden, inorden, postorden.
9) Algoritmos de Ordenamiento por burbuja, Quicksort, ShellSort y Radix.
10) Algoritmos de ordenamiento de mezcla directa, mezcla natural e intercalación.
11) Método de búsqueda secuencial, binaria y HASH.

3.2 JUSTIFICACIÓN

Un Manual de prácticas puede definirse como un compendio de documentos que contemplan una serie de aportes a la práctica científica y social de los alumnos que se encuentren realizando dicha práctica, las cuales también incluyen las normas y procedimientos que orientarán el desempeño del alumno y facilitarán la integración de la teoría con la práctica, en un contexto real de aprendizaje.

Este manual de prácticas está basado según el contenido de “el libro Guía para la elaboración y registro de textos o trabajos académicos”, con el que cuenta el Tecnológico Nacional de México.

El manual de prácticas servirá como apoyo de aprendizaje para los alumnos de la materia de Estructuras de Datos, así como apoyo didáctico para los maestros de dicha materia, ya que se presentarán consejos y sugerencias para dicha realización de las prácticas, también se dará materia de apoyo para estas mismas.

3.3 OBJETIVO GENERAL DEL MANUAL DE PRÁCTICAS

El objetivo que se pretende lograr con este manual de prácticas es estudiar las estructuras de datos más importantes utilizadas en la representación en un ordenador de los datos necesarios para resolver un problema y las estrategias algorítmicos principales.

3.4 DESARROLLO

3.4.1 Práctica 1 Creación de TDA´s.

3.4.1.1 Objetivo

El alumno aprenderá el concepto e implementación de Tipos Abstractos de Datos.

3.4.1.2 Introducción

Un *Tipo de dato abstracto* (en adelante ***TDA***) es un conjunto de datos u objetos al cual se le asocian *operaciones*.

El TDA provee de una interfaz con la cual es posible realizar las operaciones permitidas, abstrayéndose de la manera en cómo estén implementadas dichas operaciones.

Esto quiere decir que un mismo TDA puede ser implementado utilizando distintas estructuras de datos y proveer la misma funcionalidad

3.4.1.3 Correlación Los Temas Y Subtemas Del Programa De Estudio Vigente.

Esta actividad corresponde al subtema 1.2. Tipos de Datos abstractos y 1.3. Ejemplos de TDA's

3.4.1.4 Material Y Equipo Necesario

Una computadora, con los requerimientos necesarios para instalar Java y el IDE como Netbeans u otro. El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans

3.4.1.5 Metodología

Durante la clase teórica, se explicará primeramente la definición de estructura de datos, y su clasificación, posteriormente, se define lo que es un Tipo de Dato Abstracto, partiendo primero de la definición de abstracción con algunos ejemplos en la vida cotidiana.

Una vez que quede claro el concepto, se muestra a nivel de programación y de forma práctica un TDA de una figura geométrica como es un círculo, y se analiza la abstracción de sus datos y las operaciones con un círculo.

En el desarrollo de la práctica los alumnos realizarán un ejemplo guiado por el profesor sobre la abstracción de datos.

Pasos:

1) Analizar el concepto de Estructura de Datos, abstracción y Tipo de Datos Abstractos
2) Realizar un Tipo de dato Abstracto mediante código de java, con una clase llamada TDACirculo y su implementación
3) Investigar y analizar otro ejemplo para TDA por equipo e implementarlo en código de Java.
4) Realizar de manera individual la abstracción de un TDA, llamado TDAEsfera y su implementación de los métodos: área, volumen, circunferencia, y diámetro.
5) Realizar el reporte de la práctica por equipo anotando las conclusiones.

3.4.1.6 Sugerencias Didácticas

- Uso de video-proyecciones en tiempo real para demostrar a los alumnos la creación de TDA's
- Uso de Moodle como plataforma para el envío y recepción de los informes de práctica.

3.4.1.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.1.8 Bibliografías

- Apoyarse en páginas de videotutoriales: https://www.youtube.com/watch?v=6aOODlzpMOc
- https://www.youtube.com/watch?v=ToG-ussuFME
- (1) 05 - Construyendo el TDA Esfera (EDDJava) - YouTube

3.4.2 Práctica 2 Uso de memoria estática y dinámica.

3.4.2.1 Objetivo

El alumno conocerá el concepto de memoria estática y dinámica y su debido manejo en la implementación de programas.

3.4.2.2 Introducción

MEMORIA ESTÁTICA:

- Siempre asigna espacio de memoria antes de utilizar los datos concretos.
- Existirá memoria en el tiempo de ejecución del programa que nunca será utilizada.
- Es la memoria que se reserva en el momento de la compilación antes de comenzar a ejecutar el programa.

¿Qué podemos encontrar relacionado con el manejo de memoria estática?

- Variables estáticas

- Variables globales
- Miembros estáticos de clases
- Literales de cualquier tipo

MEMORIA DINÁMICA

Se realiza en tiempo de ejecución del programa después de haber leído los datos y conocer del tamaño exacto de memoria a solicitar. Tiene mejor adaptación el proceso de solución de necesidades sin desperdiciar.

Es aquella que se reserva en tiempo de ejecución después de leer los datos y de conocer el tamaño exacto del problema a resolver. El sitio donde se almacenan los objetos se le denomina HEAP = MONTÍCULO, pero el sitio preciso donde se encuentra tal montículo depende del compilador y el tipo de puntero utilizado en reserva de memoria dinámica.

Su tamaño y forma es variable (o puede serlo) a lo largo de un programa, por lo que se crean y destruyen en tiempo de ejecución. Esto permite dimensionar la estructura de datos de una forma precisa: se va asignando memoria en tiempo de ejecución según se va necesitando.

3.4.2.3 Correlación Con Los Temas Y Subtemas Del Programa De Estudio Vigente.

Esta actividad corresponde al subtema 1.4. Manejo de memoria, 1.4.1. Memoria estática y 1.4.2. Memoria dinámica.

3.4.2.4 Material Y Equipo Necesario

Una computadora, con los requerimientos necesarios para instalar Java y el IDE como Netbeans u otro. El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans

3.4.2.5 Metodología

Durante esta práctica, el alumno, analizará el concepto de memoria, memoria estática y memoria dinámica y por equipo realizarán un cuadro comparativo donde concluyan sus ventajas, desventajas y características de cada una de estas memorias, una vez que hayan implementado al menos un código para cada caso.

Pasos iniciales:

1) Analizar durante la clase le concepto de memoria, memoria estática, memoria dinámica
2) Realizar un ejercicio donde se aplique la memoria estática con un arreglo de tamaño fijo, donde el usuario escribirá sus valores y otro arreglo de tamaño fijo, donde el programador defina sus valores de inicio.
3) Realizar un ejercicio donde se aplique la memoria dinámica mediante un arreglo, donde el programador no defina el tamaño de este, sino el usuario en tiempo de ejecución indicará el tamaño necesario.
4) Realizar un cuadro comparativo de las características, ventajas y desventajas de cada una de estas memorias en su implementación.
5) Realizar el reporte de la práctica correspondiente.

3.4.2.6 Sugerencias Didácticas

- Se sugiere que el alumno cumpla con todos los requisitos solicitados con anterioridad.

3.4.2.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.2.8 Bibliografías

- Visitar la siguiente liga: https://www.youtube.com/watch?v=o9C-mvYP7Yc https://youtu.be/79LOsYYZm_8

3.4.3 práctica 3 Análisis de complejidad algorítmica: en tiempo y espacio. (Mejor, promedio y peor caso).

3.4.3.1 Objetivo

El alumno analizará algoritmos, determinando su complejidad en tiempo y espacio, y concluyendo cual es el mejor, promedio y peor caso.

3.4.3.2 Introducción

Algoritmo: El conjunto de pasos para resolver un problema

Complejidad: Significa la cantidad de recursos que se utilizan para llegar a una meta

En ciencias de la computación se estudian las propiedades de los algoritmos. La complejidad temporal es una de estas propiedades y se utiliza mucho en la práctica porque la eficiencia de los algoritmos que creamos es un factor muy importante. Ya que se traduce en un costo directo en la operación de un sistema o aplicación. Si un algoritmo puede realizar el mismo trabajo que otro, pero utilizando menos recursos vamos a usarlo para reemplazar al menos eficiente.

La complejidad temporal no es una medida de cuánto tarda en ejecutarse un algoritmo sino de cómo varía el tiempo de ejecución cuando existe una variación en la cantidad de datos de entrada. Es decir, no nos importa si el algoritmo realiza su trabajo en un minuto o en dos horas, sino cuánto más tarda en correr cuando hay diez datos en la entrada comparado con cuánto tarda con dos datos en la entrada. Desde luego que los números que acabamos de utilizar son arbitrarios e irrelevantes. Lo importante es el concepto. La complejidad temporal no tiene unidad, es una medida relativa.

Además de la complejidad temporal existe también la complejidad espacial que representa la variación del consumo de memoria del algoritmo según la variación en la cantidad de datos de entrada. Si comprendemos correctamente la primera vamos a haber comprendido la segunda en forma indirecta porque la idea detrás de ellas es idéntica. Por supuesto que cuando estudiamos el algoritmo vamos a hacer observaciones diferentes para calcular cada una. Pero al igual que la complejidad temporal, la complejidad espacial no tendrá una unidad de medida (como por ejemplo bytes) porque es una medida relativa.

A complejidad espacial y la complejidad temporal son algunas de estas propiedades y se utilizan mucho en la práctica porque la eficiencia de los algoritmos que creamos es un factor muy importante. El tiempo de CPU y la cantidad de memoria que utiliza un programa se traducen en uno de los principales costos de operación. Poder elegir algoritmos basados en estas propiedades es crucial.

La complejidad espacial no es una medida de cuánta memoria utiliza un algoritmo al ejecutarse sino de cómo varía dicho consumo cuando existe una variación en la cantidad de datos de entrada. Es decir, no nos importa si el algoritmo realiza su trabajo utilizando 10KB o 3GB, sino cuánto más consume cuando hay diez datos en la entrada comparado con cuánto consume con dos datos en la entrada. Desde luego que los números que acabamos de utilizar son arbitrarios e irrelevantes. Lo importante es el concepto. La complejidad espacial no tiene unidad, es una medida relativa.

3.4.3.3 Correlación Con Los Temas Y Subtemas Del Programa De Estudio Vigente.

Esta actividad corresponde al subtema 3.4 1.5. Análisis de algoritmos, 1.5.1. Complejidad en el tiempo, 1.5.2 Complejidad en el espacio y 1.5.3. Eficiencia de los algoritmos.

3.4.3.4 Material Y Equipo Necesario

Una computadora, con los requerimientos necesarios para instalar Java y el IDE como Netbeans u otro. El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans.

3.4.3.5 Metodología

En el desarrollo de la práctica los alumnos analizarán la información proporcionada por el docente de manera expositiva y por medio de la actividad de lectura de comprensión, conocerán más a fondo los conceptos de complejidad temporal y espacial.

1) Realizará lectura de comprensión, para la asimilación de conceptos a profundad.
2) Se analizará durante la clase, la explicación de la Notación 0
3) Mediante el recurso de cuestionario, se analizan preguntas de análisis algorítmico:
    1. ¿Qué es un algoritmo?
    2. ¿Qué es el tiempo de ejecución de un algoritmo?
    3. ¿Qué es el análisis de un algoritmo?
    4. Menciona 4 funciones típicas en el análisis de algoritmos
    5. ¿Cuál es el escenario más favorable en los algoritmos?
    6. ¿Qué es un logaritmo?
    7. ¿Para qué se usa la notación O?
    8. Anota la tabla de funciones en orden ascendente de índice de crecimiento
4) Realizará 3 algoritmos diferentes para el problema de la subsecuencia de la **suma Máxima.**

Dada la secuencia de enteros (posiblemente negativos) A1, A2,…An, encontrar e identificar la subsecuencia correspondiente) el valor máximo de $\sum_{k=i}^{j} A_{K,}$ cuando todos los enteros son negativos entendemos que la subsecuencia de suma máxima es la vacía, siendo su suma 0.

Realizar el reporte de la práctica indicando el análisis algorítmico y su conclusión, determinando cual es el mejor, peor y caso promedio, y explicando el porqué.

3.4.3.6 Sugerencias Didácticas

- Permitir que los alumnos utilicen diversas maneras de pensamiento para expresar una solución a cada problemática.

- Incentivar el trabajo en equipo.

3.4.3.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.3.8 Bibliografías

- **Análisis y complejidad de algoritmos | Undefined World**

3.4.4 Práctica 4 Conversión de algoritmos iterativos a recursivos

3.4.4.1 Objetivo

El alumno codificará los algoritmos iterativos usando la técnica de la recursividad.

3.4.4.2 Introducción

Diseño de Algoritmos Recursivos

Para que una función o procedimiento recursivo funcione se debe cumplir que:

- ✓ Existe una salida no recursiva del procedimiento o función y funciona correctamente en ese caso.
- ✓ Cada llamada al procedimiento o función se refiere a un caso más pequeño del mismo.
- ✓ Funciona correctamente todo el procedimiento o función.

Para poder construir cualquier rutina recursiva teniendo en cuenta lo anterior, podemos usar el siguiente método:

- Primero, obtener una definición exacta del problema.
- A continuación, determinar el tamaño del problema completo a resolver. Así se determinarán los valores de los parámetros en la llamada inicial al procedimiento o función.
- Tercero, resolver el caso base en el que problema puede expresarse no recursivamente. Esto asegurará que se cumple el punto 1 del test anterior.

- Por último, resolver correctamente el caso general, en términos de un caso más pequeño del mismo problema (una llamada recursiva). Esto asegurará cumplir con los puntos 2 y 3 de la prueba.

Cuando usar recursividad y cuándo iteración

La potencia de la recursión reside en la posibilidad de definir un número infinito de objetos mediante un enunciado finito. De igual forma, un número infinito de operaciones de cálculo puede describirse mediante un programa recursivo finito, incluso si este programa no contiene repeticiones explícitas.

Los algoritmos recursivos son apropiados principalmente **cuando el problema a resolver**, o la función a calcular, o la estructura de datos a procesar, **están ya definidos recursivamente.**

Hay varios factores a considerar en la decisión sobre si usar o no una solución recursiva en un problema. La solución recursiva puede necesitar un considerable gasto de memoria para las múltiples llamadas al procedimiento o función, puesto que deben almacenarse las direcciones de vueltas y copias de las variables locales y temporales. Si un programa debe ejecutarse frecuentemente (como un compilador) y/o debe ser muy eficiente, el uso de programación recursiva puede no ser una buena elección.

Sin embargo, para algunos problemas una solución recursiva es más natural y sencilla de escribir para el programador. Además, la recursividad es una herramienta que puede ayudar a reducir la complejidad de un programa ocultando algunos detalles de la implementación. Conforme el costo del tiempo y el espacio de memoria de las computadoras disminuye y aumenta el costo del tiempo del programador, puede ser útil usar soluciones recursivas en estos casos.

En general, si la solución no recursiva no es mucho más larga que la versión recursiva, usar la no recursiva. De acuerdo con esta regla, los ejemplos que hemos visto no son buenos ejemplos de programación recursiva, aunque sirven para ilustrar cómo comprender y escribir procedimientos y funciones recursivas, sería más eficiente y sencillo escribirlas iterativamente.

La iteración y la recursividad cumplen con el mismo objetivo: ejecutar un bloque de sentencias ***n*** veces o que con una condición de fin adecuada lo termine. Es importante distinguir las diferentes instancias de la ejecución, los valores de las variables determinan la instancia particular que está siendo resuelta en ese momento. Es necesario que la ejecución del bloque termine, es decir, el mismo bloque debe contener la condición de corte que permita que cualquier instancia menor del problema pueda ser resuelta directamente. Entonces, ¿cómo decir cuál alternativa es más adecuada? A veces la definición del problema no induce ninguna estructura de control en particular, recién al plantear un método de resolución es posible elegir entre iteración y recursividad. Esta decisión tiene que ver con la costumbre que tenga el programador al plantear sus algoritmos y, una vez planteado un método de resolución, se busca cuál herramienta conviene más.

Hay problemas que aparecen más naturales para la iteración y para otros para la recursividad, por ejemplo, en problemas con contadores, sumatorias y productoras lo natural es la iteración, pero en problemas en los que distinguimos un caso base y uno general, como factorial, Fibonacci, MCD lo natural es la recursión. Pero existe otro factor muy importante a tener en cuenta: la eficiencia.

Un conjunto de objetos está definido recursivamente siempre que:

- ✓ (B) algunos elementos del conjunto se especifican explícitamente.
- ✓ (R) el resto de los elementos del conjunto se definen en términos de los elementos ya definidos donde (B) se llama base (R) se llama cláusula recursiva.

3.4.4.3 Especificar La Correlación Con El O Los Temas Y Subtemas Del Programa De Estudio Vigente.

Esta actividad corresponde al subtema 2.2 Procedimientos recursivos y 2.3. Casos recursivos.

3.4.4.4 Material Y Equipo Necesario

El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans.

3.4.4.5 Metodología

El alumno deberá analizar primero cada uno de los siguientes problemas:

- ✓ Programa para calcular el factorial de un número.
- ✓ Programa para calcular la potencia de un número.
- ✓ Programa para calcular la suma de N primeros enteros.
- ✓ Programa que lea un número entero mayor o igual que cero en base decimal y muestre su equivalente en binario.
- ✓ Imprimir la tabla de multiplicar de N.
- ✓ Calcular el máximo común divisor.

3.4.4.6 Sugerencias Didácticas

- Para que una persona pueda resolver un problema, siempre es indispensable la concentración, una buena alimentación y un buen descanso. Se ha comprobado que cuando la mente tiene sueño o hambre, es difícil llegar a concentrarse y mucho más aun a resolver un problema. Si el alumno está preocupado o aturdido, es mejor que haga una pausa, y que relaje su mente en algo más, minutos después puede retomar el ejercicio, listo para concluirlo con éxito.

3.4.4.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.4.8 Bibliografías

- **¿Qué es la recursividad en la programación con Java | Tokio (tokioschool.com)**
- **5.2. Ciclo interactivo (Algoritmos de Programación con Python) (uniwebsidad.com)**

3.4.5 Práctica 5 Serie de ejercicios con Recursividad indirecta o cruzada, múltiple

3.4.5.1 Objetivo

El alumno aprenderá a resolver problemas mediante recursividad.

3.4.5.2 Introducción

Tipos de recursión:

- **Recursividad simple**: Aquella en cuya definición sólo aparece una llamada recursiva. Se puede transformar con facilidad en algoritmos iterativos.
- **Recursividad múltiple**: Se da cuando hay más de una llamada a sí misma dentro del cuerpo de la función, resultando más difícil de hacer de forma iterativa.
- **Recursividad cruzada o indirecta**: Son algoritmos donde una función provoca una llamada a sí misma de forma indirecta, a través de otras funciones.

3.4.5.3 Especificar La Correlación Con El O Los Temas Y Subtemas Del Programa De Estudio Vigente.

Esta actividad corresponde al subtema 2.2 Procedimientos recursivos y 2.3. Casos recursivos.

3.4.5.4 Material Y Equipo Necesario

1. El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans.

3.4.5.5 Metodología

Ejercicios por realizar durante las horas de práctica en el laboratorio de cómputo:

1. Por medio de la recursividad indirecta, determinar si un número es par o impar.
2. Por medio de la recursividad múltiple realizar la serie Fibonacci.
3. Usando recursividad directa imprimir un arreglo de cadena de caracteres.
4. Resolver la torre de Hannoi de manera recursiva.
5. Implementar el algoritmo de Ackerman de manera recursiva.

3.4.5.6 Sugerencias Didácticas

Investigar ¿qué es la serie Fibonacci y en qué consiste?, documentarse sobre la torre de Hannoi y el Algoritmo de Ackerman, antes de la clase práctica.

3.4.5.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando

detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.5.8 Bibliografías

- [¿Qué es la recursividad en la programación con Java | Tokio (tokioschool.com)]

3.4.6 Práctica 6 Operaciones básicas con pilas y colas de manera estática, usando arreglos.

3.4.6.1 Objetivo

El alumno aprenderá a crear la estructura pila y cola mediante arreglos.

3.4.6.2 Introducción

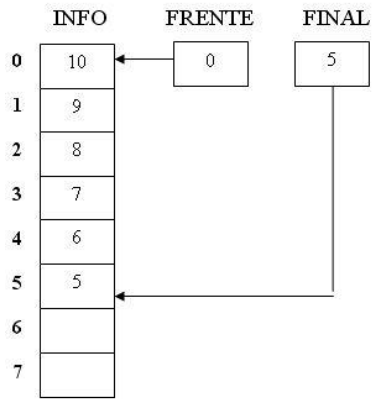
- ✓ Uno de los conceptos más útiles en computación es la pila o stack
- ✓ Es un conjunto de elementos, en la que:
  - Los elementos se añaden y se remueven por un solo extremo
  - Este extremo es llamado “tope” de la pila.
  - Ejemplo:
  - Cuando un empleado se va de vacaciones, le llega correo a su escritorio.
  - Las cartas se van “apilando”.
  - Al regresar de vacaciones, la última carga en llegar será la primera que revisará
  - Al terminar de revisarla, la nueva carta del tope de la pila habrá cambiado
  - Del “pilo” de cartas, la más nueva que queda, será la siguiente en ser revisada.

Estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción PUSH se realiza por un extremo denominado final y la operación de extracción POP por el otro extremo llamado frente; también llamado FIFO (first in, first out).

Es una estructura de acceso restrictiva de sus elementos.

Características

- Capacidad especifica.
- Por muchos elementos que tenga siempre se le puede añadir un más.
- Si está vacía no se puede eliminar ningún elemento.



3.4.6.3 Especificar La Correlación Con El O Los Temas Y Subtemas Del Programa De Estudio Vigente.

Esta actividad corresponde al subtema 3.1. Pilas, 3.1.2. Operaciones básicas y 3.1.3. Aplicaciones.

3.4.6.4 Material Y Equipo Necesario

El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans.

3.4.6.5 Metodología

El alumno define e identifica la estructura lineal Pilas y sus operaciones. Aprenderá a crear una pila estática, una pila dinámica y a diseñar e implementar las distintas operaciones de una pila.

En el desarrollo de esta práctica, el alumno realizará en el laboratorio de cómputo, los siguientes ejercicios:

1. Crear una pila estática con un arreglo de 10 elementos

2. Crear una estructura cola mediante un arreglo de 5 elementos, implementando los punteros correspondientes.

3. Implementación de pilas y colas manejando memoria dinámica.

3.4.6.6 Sugerencias Didácticas

- Se sugiere que previo a la realización de esta práctica, el alumno cuenta ya con conocimientos elementales sobre programación orientada a objetos.

3.4.6.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.6.8 Bibliografías

- **Stacks de desarrollo web y ejemplos. - Syntonize**

3.4.7 Práctica 7 Reutilización de clases definidas para pilas, colas y listas enlazadas dinámicas

3.4.7.1 Objetivo

Implementar mediante clases definidas, pilas, colas y listas

3.4.7.2 Introducción

Una pila (stack en inglés) es una estructura de datos lineal que solo tienen un único punto de acceso fijo por el cual se añaden, eliminan o se consultan elementos. El modo de acceso a los elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir).
Cuando se crea un programa en el que es necesario manejar memoria dinámica el sistema operativo divide el programa en cuatro partes que son: texto, datos (estáticos), pila y una zona libre. En la última parte es donde queda la memoria libre para poder utilizarla de forma dinámica. En el momento de la ejecución habrá tanto partes libres como partes asignadas al proceso por lo cual si no se liberan las partes utilizadas de la memoria y que han quedado inservibles es posible que se “agote” esta parte y por lo tanto la fuente de la memoria dinámica. También la pila cambia su tamaño dinámicamente, pero esto no depende del programador sino del sistema operativo.

3.4.7.3 Especificar La Correlación Con El O Los Temas Y Subtemas Del Programa De Estudio Vigente.

Esta actividad corresponde al subtema 3.2. Colas, 3.2.2. Operaciones básicas

Y 3.2.3. Aplicaciones**.**

3.4.7.4 Material Y Equipo Necesario

El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es:

Netbeans.

3.4.7.5 Metodología

El alumno define e identifica la estructura lineal Pilas y sus operaciones, mediante implementación de clases definidas en la API De java.

En el desarrollo de esta práctica, el alumno realizará en el laboratorio de cómputo, los siguientes ejercicios:

1. Crear una pila usando la clase Stack y sus métodos.
2. Crear una estructura cola usando la clase Queue y sus métodos
3. Implementación de listas mediante la clase Array y sus métodos

3.4.7.6 Sugerencias Didácticas

- Se sugiere que previo a la realización de esta práctica, el alumno cuenta ya con conocimientos elementales sobre programación orientada a objetos.

3.4.7.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.7.8 Bibliografías

- Stacks de desarrollo web y ejemplos. - Syntonize

3.4.8 Práctica 8. Operaciones con árboles: Creación, inserción, eliminación y búsqueda en a.b.b, conversión de árboles generales a binarios. Y sus Recorridos sistemáticos: Preorden, inorden, postorden.

3.4.8.1 Objetivo

El alumno aprenderá la implementación de operaciones básicas con un árbol.

3.4.8.2 Introducción

TERMINOLOGÍA DE ARBOLES

- NODO: Es la unidad sobre la que se construye el árbol y puede tener 0 o más nodos hijos.
- RAÍZ DEL ÁRBOL: todos los árboles tienen un único nodo raíz, todos los demás elementos o nodos se derivan o descienden de él. El nodo raíz no tiene padre, es decir, no es hijo de nadie, es decir, es el primero.
- NODO HOJA O TERMINAL: Si in nodo no tiene ningún descendiente se le

conoce como nodo hoja.

- HIJO: Cada uno de los apuntadores en el nodo raíz, se refiere a un hijo.
- HIJO IZQUIERDO: Es el primer nodo en el subárbol izquierdo.
- HIJO DERECHO: Es el primer nodo en el subárbol derecho.
- HERMANO: Dos nodos son hermanos si son hijo izquierdo y derecho del mismo padre.
- DESCENDIENTES: Los hijos de un nodo se conocen como descendientes.
- SUBÁRBOL: Son los descendientes o hijos que tienen un ascendente llamado padre.
- ÁRBOL BINARIO: Es un conjunto finito de elementos, que está dividido en tres
- subconjuntos separados. El primero es el nodo raíz, el segundo es el subárbol izquierdo y el tercero es el subárbol derecho.
- NODOS INTERIORES: No son padres ni hijos, los nodos con uno o dos subárboles que no son hojas ni raíz.

3.4.8.3 Especificar La Correlación Con El O Los Temas Y Subtemas Del Programa De Estudio Vigente.

4.1. Arboles

4.1.2. Operaciones básicas con árboles binarios

4.1.3. Aplicaciones

3.4.8.4 Material Y Equipo Necesario

El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans.

3.4.8.5 Metodología

El alumno define e identifica la estructura no lineal árbol y grafo y sus operaciones. En el desarrollo de esta práctica, el alumno realizará en el laboratorio de cómputo, los siguientes ejercicios:

1. Crear un árbol binario de búsqueda

2. Realizar las operaciones básicas con él: creación, inserción, eliminación y búsqueda, así como sus recorridos sistemáticos: preorden, postorden, inorden.

3.4.8.6 Sugerencias Didácticas

- Se sugiere que previo a la realización de esta práctica, el alumno cuenta ya con conocimientos elementales sobre programación orientada a objetos.

3.4.8.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.8.8 Bibliografías

- Árbol binario - Qué es, definición y concepto | 2023 | Economipedia

3.4.9 Práctica 9 Algoritmos de Ordenamiento por burbuja, Quicksort, ShellSort y Radix.

3.4.9.1 Objetivo

El alumno conocerá y analizará los algoritmos de ordenamiento interno.

3.4.9.2 Introducción

En esta unidad se tratarán los temas de ordenación, los métodos existentes para llevar a cabo el ordenamiento. Existen varios métodos para ordenar la información, los cuales tienen como finalidad organizar los datos en un orden creciente o decreciente mediante una regla prefijada (numérica, alfabética, etc.). Atendiendo al tipo de elemento que se quiera ordenar la ordenación puede ser:

- Interna
- Externa

En este caso analizaremos la ordenación interna la cual se puede clasificar en dos tipos:

- Métodos directos.
- Métodos logarítmicos.

Y estudiaremos los métodos directos, los más conocidos son:

- Ordenación por intercambio. (Burbuja).
- Ordenación por inserción. (Inserción).
- Ordenación por selección. (Shell).

Los cuales veremos a continuación.

ORDENAMIENTO:

Es la operación de arreglar los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento. El ordenamiento se efectúa con base en el valor de algún campo en un registro. El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado.

Ej. de ordenamientos:

Dir. telefónico, tablas de contenido, bibliotecas y diccionarios, etc.

El ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia tal que represente un orden, el cual puede ser numérico, alfabético o incluso alfanumérico, ascendente o descendente.

¿Cuándo conviene usar un método de ordenamiento?

Cuando se requiere hacer una cantidad considerable de búsquedas y es importante el factor tiempo.

TIPOS DE ORDENAMIENTOS:

Los 2 tipos de ordenamientos que se pueden realizar son: los internos y los externos.

Los internos:

Son aquellos en los que los valores a ordenar están en memoria principal, por lo que se asume que el tiempo que se requiere para acceder cualquier elemento sea el mismo (a[1], a[500], etc).

Los externos:

Son aquellos en los que los valores a ordenar están en memoria secundaria (disco, cinta, cilindro magnético, etc), por lo que se asume que el tiempo que se requiere para acceder a cualquier elemento depende de la última posición accesada (posición 1, posición 500, etc).

Eficiencia en tiempo de ejecución:

Una medida de eficiencia es:

Contar el # de comparaciones (C)

Contar el # de movimientos de items (M)

Estos están en función del #(n) de items a ser ordenados.

Un "buen algoritmo" de ordenamiento requiere de un orden nlogn comparaciones.

La eficiencia de los algoritmos se mide por el número de comparaciones e intercambios que tienen que hacer, es decir, se toma n como el número de elementos que tiene el arreglo o vector a ordenar y se dice que un algoritmo realiza O(n2) comparaciones cuando compara n veces los n elementos, n x n = n2.

ALGORITMOS ORDENAMIENTO POR INTERCAMBIO

Algoritmo iterativo que realiza el ordenamiento utilizando intercambio de variables utilizando una variable auxiliar. En cada iteración se analiza cual elemento es mayor y utilizando la variable auxiliar se realiza el cambio de orden si es necesario.

En este tipo de algoritmos se toman los elementos de dos en dos, se comparan y se INTERCAMBIAN si no están en el orden adecuado. Este proceso se repite hasta que se ha analizado todo el conjunto de elementos y ya no hay intercambios.

Entre estos algoritmos se encuentran el BURBUJA y QUICK SORT.

ORDENACION BURBUJA

El bubble sort, también conocido como ordenamiento burbuja, funciona de la siguiente manera: Se recorre el arreglo intercambiando los elementos adyacentes

que estén desordenados. Se recorre el arreglo tantas veces hasta que ya no haya cambios. Prácticamente lo que hace es tomar el elemento mayor y lo va recorriendo de posición en posición hasta ponerlo en su lugar. A pesar de que el ordenamiento de burbuja es uno de los algoritmos más sencillos de implementar, su orden O(n2) lo hace muy ineficiente para usar en listas que tengan más que un número reducido de elementos. Incluso entre los algoritmos de ordenamiento de orden O(n2), otros procedimientos como el Ordenamiento por inserción son considerados más eficientes. Dada su simplicidad, el ordenamiento de burbuja es utilizado para introducir el concepto de algoritmo, o de algoritmo de ordenamiento para estudiantes de ciencias de la computación. Aunque algunos investigadores han criticado al ordenamiento de burbuja y su popularidad en la educación de la computación, recomendando que ya no debe ser enseñado.

El ordenamiento de burbuja es asintóticamente equivalente, en tiempos de ejecución con el Ordenamiento por inserción en el peor de los casos, pero ambos algoritmos difieren principalmente en la cantidad de intercambios que son necesarios. Por esta razón, muchos libros de algoritmos modernos evitan usar el ordenamiento de burbuja, utilizando en cambio el ordenamiento por inserción. El ordenamiento de burbuja interactúa vagamente con el hardware de las CPU modernas. Requiere al menos el doble de escrituras que el ordenamiento por inserción, el doble de pérdidas de cache, y asintóticamente más predicción de saltos.

QUICK SORT ORDENACION

El ordenamiento rápido (quicksort en inglés) es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo

proporcional a n log n. Esta es la técnica de ordenamiento más rápida conocida. Fue desarrollada por C. Antony R. Hoare en 1960. El algoritmo original es recursivo, pero se utilizan versiones iterativas para mejorar su rendimiento (los algoritmos recursivos son en general más lentos que los iterativos, y consumen más recursos).

3.4.9.3 Especificar La Correlación Con El O Los Temas Y Subtemas Del Programa De Estudio Vigente.

5.1. Algoritmos de Ordenamiento interno

3.4.9.4 Material Y Equipo Necesario

El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans.

3.4.9.5 Metodología

Desarrollar durante las clases los algoritmos de ordenamiento interno: burbuja, shellsort, radix y Quicksort.

Se comprobará cada uno de ellos, mediante ejecución y prueba de escritorio usando los datos de los ejercicios al pizarrón para comprobar que estén correctos.

Se analizarán cada uno de los métodos de ordenamiento, a nivel de análisis algorítmico, para determinar el mejor caso.

3.4.9.6 Sugerencias Didácticas

Como alternativa, ver videos previamente seleccionados para el uso y funcionamiento de cada uno de los métodos de ordenamiento interno.

3.4.9.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.9.8 Bibliografías

- Algoritmo de Ordenamiento de Burbuja (Bubble Sort) (pleets.org)

3.4.10 Práctica 10 Algoritmos de ordenamiento de mezcla directa, mezcla natural e intercalación

3.4.10.1 Objetivo

El alumno comprende y aplica los algoritmos de ordenamientos externo para el uso adecuado en el desarrollo de aplicaciones que permita solucionar problemas del entorno.

3.4.10.2 Introducción

Un método de ordenación que trabaja sobre un conjunto de datos que se encuentra en memoria (ejemplo, un arreglo, una lista) se dice que es un método de ordenación interna. Por

el contrario, si el conjunto de datos almacenados en archivos no puede ser cargado en memoria (por ejemplo, por razones de tamaño) y el método de ordenación opera sobre los archivos, se dice que es de ordenación externa. Evidentemente, en la ordenación interna se accede a los elementos de dato más fácilmente, mientras que en la ordenación externa se accede a los elementos de dato de forma secuencial o al menos en grades bloques.

INTERCALACION DIRECTA

Intercalación (MERGE) es el proceso de combinar dos o más archivos (arreglos) ordenados en un tercer archivo ordenado. Este algoritmo de comparación es estable ya que se mantiene el orden relativo de registros con claves iguales. Es un tipo de algoritmo “DIVIDE Y VENCERAS” Fue inventado por John Von Newmann en 1945. Esta técnica funciona de la siguiente manera:

- Dividir el archivo en n sub-archivos de tamaño 1 e intercalar pares adyacentes (inconexos) de archivos.

Entonces tenemos más o menos n/2 archivos de tamaño 2.

- Repetir el proceso hasta que solo reste un archivo de tamaño n.

Cuando los elementos iguales son indistinguibles, como con los enteros, o más generalmente, cualquier dato en el que el elemento es la llave, la estabilidad no es un problema. Sin embargo, asumamos que los siguientes pares de números se van a ordenar por su primera componente: (4, 1) (3, 7) (3, 1) (5, 6)

En este caso, el resultado puede ser dos ordenaciones posibles, uno que mantiene el orden relativo de los registros con llaves iguales y otro que no: (3, 7) (3, 1) (4, 1) (5, 6) (Se mantiene) (3, 1) (3, 7) (4, 1) (5, 6) (Se cambia)

Los ordenamientos inestables pueden cambiar el orden relativo, pero en el ordenamiento estable nunca sucede esto.

MEZCLA NATURAL

La idea básica es realizar particiones tomando secuencias ordenadas de máxima longitud en lugar de secuencias ordenadas de tamaño fijo previamente determinadas. Luego se realiza la fusión de esas secuencias ordenadas, alternativamente sobre dos archivos. Repitiendo este proceso secuencialmente, se logra que el archivo quede completamente ordenado. Para aplicar este algoritmo, se necesitarán cuatro archivos. El archivo original y tres archivos auxiliares. De estos cuatro archivos, dos serán considerados de entrada y dos de salida, alternativamente en cada paso del algoritmo. El proceso termina cuando en la finalización de un paso, el segundo archivo de salida quede vacío.

3.4.10.3 Especificar La Correlación Con El O Los Temas Y Subtemas Del Programa De Estudio Vigente.

5.2. Algoritmos de ordenamiento externos

3.4.10.4 Material Y Equipo Necesario

El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans.

3.4.10.5 Metodología

En el desarrollo de la práctica los alumnos realizarán un ejemplo guiado por el profesor.

Ejemplo:

a) Método de ordenamiento natural
b) Método de ordenamiento directo
c) Método de ordenamiento por mezcla

3.4.10.6 Sugerencias Didácticas

Se sugiere que el alumno cuente con los conocimientos básicos para realizar los ejercicios propuestos.

3.4.10.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando

detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.10.8 Bibliografías

- gbaudino/MetodosDeOrdenamiento: Métodos de ordenamiento con sus especificaciones, características y su código en Python 3. Comparativa final entre los algoritmos de ordenamiento. (github.com)

3.4.11 Práctica 11 Método de búsqueda secuencial, binaria y HASH

3.4.11.1 Objetivo

El alumno conoce, comprende y aplica los algoritmos de búsqueda para el uso adecuado en el desarrollo de aplicaciones que permita solucionar problemas del Entorno.

3.4.11.2 Introducción

BUSQUEDA HASH

En este método se requiere que los elementos estén ordenados.

El método consiste en asignar el índice a cada elemento mediante una transformación del elemento, esto se hace mediante una función de conversión llamada función hash. Hay diferentes funciones para transformar el elemento y el número obtenido es el índice del elemento. La principal forma de transformar el elemento es asignarlo directamente, es decir al 0 le corresponde el índice 0, al 1 el 1, y así sucesivamente pero cuando los elementos son muy grandes se desperdicia mucho espacio ya que necesitamos arreglos grandes para almacenarlos y estos quedan con muchos espacios libres, para utilizar mejor el espacio se utilizan funciones más complejas. La función de hash ideal debería ser biyectiva, esto es, que a cada elemento le corresponda un índice, y que a cada índice le corresponda un elemento, pero no siempre es fácil encontrar esa función, e incluso a veces es inútil, ya que puedes no saber el número de elementos a almacenar. La función de hash depende de cada problema y de cada finalidad, y se pueden utilizar con números o cadenas, pero las más utilizadas son:

1.- Restas sucesivas: Esta función se emplea con claves numéricas entre las que existen huecos de tamaño conocido, obteniéndose direcciones consecutivas.

BUSQUEDA EXTERNA

Para realizarla, es necesario contar con un array o vector ordenado. Luego tomamos un elemento central, normalmente el elemento que se encuentra a la mitad del arreglo, y lo comparamos con el elemento buscado. Si el elemento buscado es menor, tomamos el intervalo que va desde el elemento central al principio, en caso contrario, tomamos el intervalo que va desde el elemento central hasta el final del intervalo. Procedemos de esta manera con intervalos cada vez menores hasta que lleguemos a un intervalo indivisible, en cuyo caso el elemento no está en el vector, o el elemento central sea nuestro elemento. De esta forma la complejidad computacional se reduce a O (ln N).

SECUENCIAL

Si buscamos el número 3 que se encuentra en la posición 16 de la tabla, el programa realizará 16 comparaciones antes de llegar hasta el número. Si buscamos el 78 le bastará con una comparación, pero si buscamos el último de la tabla, el 0, tendremos que hacer tantas comparaciones como elementos tenga la tabla. Sobre una tabla de N elementos.

hay que hacer N comparaciones para localizar el número en el peor de los casos, y en promedio N/2. Entendemos por este promedio el de la estadística de las búsquedas sobre un número grande de tablas cuyos números están colocados al azar y en los que también se elige uno al azar para buscar. El caso de buena suerte, buscar el primero, es irrelevante.

3.4.11.3 Especificar La Correlación Con El O Los Temas Y Subtemas Del Programa De Estudio Vigente.

6.1. Búsqueda Secuencial, 6.2 Búsqueda Binaria y 6.3 Búsqueda HASH

3.4.11.4 Material Y Equipo Necesario

El software por emplear es Java SE Development KIT (JDK) y el IDE sugerido es: Netbeans.

3.4.11.5 Metodología

Ejercicios sencillos por resolver

1) Método de búsqueda secuencial
2) Método de búsqueda binaria
3) Método de búsqueda HASH

3.4.11.6 Sugerencias Didácticas

El alumno debe de contar con los conocimientos necesarios para poder cumplir con los ejercicios propuestos.

3.4.11.7 Reporte Del Alumno

El alumno debe de realizar la actividad detallando paso a paso la elaboración de esta, incluyendo capturas, mediante el formato de un reporte de prácticas dando detalle de los resultados obtenidos, así como su conclusión y aprendizajes obtenidos.

3.4.11.8 Bibliografías

- ¿Qué Es Un Hash Y Cómo Funciona? | Blog oficial de Kaspersky