ÁREA 1. ALGORITMIA

SUBÁREA 1.1 ANÁLISIS Y DISEÑO DE ALGORITMOS

TEMAS IMPORTANTES:

**Conceptos de Algoritmos:**

Es un conjunto de pasos lógicos y estructurados que nos permiten dar solución aún problema.

La importancia de un algoritmo radica en desarrollar un razonamiento lógico matemático a través de la comprensión y aplicación de metodologías para la resolución de problemáticas, estas problemáticas bien pueden ser de la propia asignatura o de otras disciplinas como matemáticas, química y física que implican el seguimiento de algoritmos, apoyando así al razonamiento critico deductivo e inductivo.

La estructura de un Algoritmo se divide en 3 secciones principales:



Entrada: Es la introducción de datos para ser transformados.

Proceso: Es el conjunto de operaciones a realizar para dar solución al problema.

Salida: Son los resultados obtenidos a través del proceso.

**Pseudocódigo y Diagramas de Flujo:**

Pseudocódigo

Es empleado para representar la solución de un algoritmo empleando lenguaje natural escrito estableciendo la secuencia de pasos sin imprecisiones y de manera clara.

Ejemplo:

Proceso

Leer lista\_de\_variables;

variable<-expresion;

Escribir lista\_de\_expresiones;

FinProceso

Diagrama de flujo:

|  |  |
| --- | --- |
| Simbolo | Descripcion |
|  | El rectángulo se utiliza para identificar las acciones a realizar, es decir, este símbolo indica el proceso a realizar |
|  | El paralelogramo, indica la entrada o lectura de los datos |
|  | El rombo, es la caja de decisiones, representa las alternativas con solo dos posibles opciones SI y NO |
|  | Rectángulo con esquinas redondeadas o semicírculo, son utilizados para indicar el inicio y el final del algoritmo |
|  | El cono se utiliza para indicar una  salida en pantalla. |
|  | La flecha, indica la secuencia de  acciones por realizar, es decir, es  quien marca la continuidad y orden de ejecución de las acciones propias del problema por resolver |
|  | Representa la repetición de pasos a  a través de los ciclos |

Ejemplo:

Pseudocodigo:

Inicio

Leer n1, n2

suma=n1+n2

imprimir suma

Fin

Diagrama de Flujo

Diagrama

Descripción generada automáticamente

**Programas Simples:**

Debido a que el examen es de opción múltiple, no se les puede pedir que analicen o desarrollen grandes programas o incluso funciones largas. Por lo que lo más seguro es que vengan programas simples que incluyan los elementos básicos: ciclos, condicionales, arreglos (simples, paralelos, multidimensionales), funciones, parámetros, etc. Es probable que se les pida encontrar el error en un programa o función, indicar qué hace cierto código, indicar el resultado.

códigos de programas simples en la sección de códigos.

**Recursividad y Modularidad:**

Recursividad:

La recursividad es un concepto que se indica cuando un método se llama a si mismo. Cuando creamos un método recursivo debemos tener en cuenta que este tiene que terminar por lo que dentro del método debemos asegurarnos de que no se está llamando a si mismo todo el rato, Lo que quiere decir que el ciclo es finito.

Modularidad:

El modularidad consiste en dividir un programa en módulos que puedan compilarse por separado, sin embargo tendrá conexiones con otros módulos.

El modularidad también tiene principios y son los siguientes:

1. Capacidad de descomponer un sistema complejo.
2. Capacidad de componer a través de sus módulos.
3. Comprensión de sistema en partes.

Capacidad de descomponer un sistema complejo

Recuerdas el principio de «Divide y Vencerás», en este procedimiento se realiza algo similar, ya qué descompones un sistema en subprogramas (recuerda llamarlos módulos), el problema en general lo divides en problemas más pequeños.

Capacidad de componer a través de sus módulos

Indica la posibilidad de componer el programa desde los problemas más pequeños completando y resolviendo el problema en general, particularmente cuando se crea software se utilizan algunos módulos existentes para poder formar lo que nos solicitan, estos módulos que se integran a la aplicación deben de ser diseñados para ser reusables.

Comprensión de sistema en partes

El poder tener cada parte separada nos ayuda a la comprensión del código y del sistema, también a la modificación del mismo, recordemos que si el sistema necesita modificaciones y no hemos trabajado con módulos definitivamente eso será un caos.

Utiliza:

Funciones: Las funciones son bloques de código que realizan una tarea específica. Se pueden definir una vez y reutilizar en diferentes partes del programa. Por ejemplo, si un programa necesita realizar una operación matemática repetidamente, se puede crear una función para realizar esa operación y llamarla cada vez que sea necesario.

Clases: Las clases son bloques de código que encapsulan datos y funciones relacionados en un solo objeto. Permiten la creación de objetos que representan algo del mundo real y tienen propiedades y comportamientos. Por ejemplo, una clase Persona podría tener propiedades como nombre y edad, así como métodos para cambiar o mostrar esa información.

Módulos de importación: Los módulos de importación permiten la reutilización de código de otros programas. Por ejemplo, si un programa necesita realizar operaciones matemáticas avanzadas, se puede importar una biblioteca de matemáticas para tener acceso a funciones y métodos avanzados.

**Análisis de Algoritmos:**

<https://www.youtube.com/watch?v=BAr-EzHYM4I> - (Importancia de algoritmos eficientes 1/2)

<https://www.youtube.com/watch?v=MCxDF8-tp8U> - (Importancia de algoritmos eficientes 2/2)

<https://www.youtube.com/watch?v=4yv38XSOxS0> - (Análisis de algoritmos)

<https://www.youtube.com/watch?v=49qGo1zaXS4> - (Orden 1/2)

<https://www.youtube.com/watch?v=Iaf4BQ_Zbjw> - (Orden 2/2)

<https://www.youtube.com/watch?v=9D1ereWiEPY> - (Análisis de complejidad)

**Estrategias Algorítmicas:**

"Divide y vencerás" es un algoritmo de diseño que se utiliza para resolver problemas complejos, dividiéndolos en subproblemas más pequeños y manejables. El enfoque básico consiste en dividir el problema en subproblemas más pequeños y resolviéndolos por separado. Luego, se combinan las soluciones para obtener la solución al problema original. Este enfoque se basa en la idea de que es más fácil resolver problemas más pequeños y simples que abordar el problema en su totalidad.

El algoritmo se puede aplicar en muchos problemas diferentes, como ordenamiento de listas, búsqueda binaria, árboles binarios de búsqueda, entre otros.

En general, el enfoque de "divide y vencerás" es útil cuando se trata de problemas que pueden descomponerse en subproblemas más pequeños y manejables. Permite una solución más eficiente y escalable en comparación con otro enfoque

La programación dinámica es una técnica de diseño de algoritmos que se utiliza para resolver problemas computacionales complejos y optimización de problemas. La idea básica de la programación dinámica es dividir el problema en subproblemas más pequeños y solucionar cada subproblema solo una vez, almacenando su solución para evitar tener que resolverlo nuevamente en el futuro.

La programación dinámica es útil cuando el problema se puede dividir en subproblemas más pequeños, y cuando la solución óptima del problema se puede construir a partir de soluciones óptimas a subproblemas más pequeños. El enfoque de programación dinámica se utiliza para evitar la repetición de cálculos innecesarios que pueden ocurrir cuando se utiliza la fuerza bruta o enfoques recursivos.

La programación dinámica se puede aplicar a muchos problemas diferentes, como problemas de optimización, problemas de búsqueda y problemas de decisión. Ejemplos de problemas que se pueden resolver mediante programación dinámica incluyen:

* El cálculo de la distancia de edición entre dos cadenas de caracteres
* La búsqueda de la ruta más corta en un grafo
* La determinación de la mejor forma de cortar una barra de metal en piezas de diferentes longitudes
* La optimización de carteras de inversión.

Greedy (codicioso en español) es una técnica de diseño de algoritmos que se utiliza para resolver problemas de optimización. La idea básica detrás de un algoritmo greedy es tomar la decisión óptima localmente en cada paso, sin considerar las consecuencias a largo plazo. En otras palabras, en cada paso, el algoritmo elige la opción que parece ser la mejor en ese momento, sin preocuparse por cómo esta elección afectará las decisiones futuras.

La estrategia greedy puede ser efectiva para resolver ciertos tipos de problemas de optimización, especialmente cuando el problema tiene una estructura simple y la solución óptima se puede construir paso a paso. Algunos ejemplos de problemas que se pueden resolver mediante un algoritmo greedy incluyen encontrar el camino más corto en un grafo, la selección de un conjunto de elementos con un valor máximo y un costo mínimo, y la asignación de tareas a máquinas para minimizar el tiempo total de procesamiento.

Sin embargo, un algoritmo greedy no siempre produce la solución óptima global, y en algunos casos, puede llevar a soluciones subóptimas. Por lo tanto, es importante evaluar cuidadosamente si la estrategia greedy es apropiada para el problema en cuestión y considerar otras técnicas de diseño de algoritmos si es necesario.