



CADP - resumen de final

Conceptos de Algoritmos, Datos y Programas (Universidad Nacional de La Plata)



Escanea para abrir en Studocu

TEORIA/PRACTICA CADP

DATO: Es una clase de objetos ligados a un conjunto de operaciones para crearlos y manipularlos.

TIPOS DE DATOS

- **SIMPLE:** Aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

* DEFINIDO POR EL LENGUAJE: Son provistos por el lenguaje y tanto la representación como sus operaciones y valores son reservadas al mismo.

* DEFINIDO POR EL PROGRAMADOR: Permiten definir nuevos tipos de datos a partir de los tipos simples.

- **COMPUESTO:** Pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

DATO NUMERICO:

Representa el conjunto de números que se pueden necesitar. Estos números pueden ser enteros o reales (SIMPLES/ORDINALES).

DATO LÓGICO:

Permite representar datos que pueden tomar dos valores verdadero o falso. (SIMPLES/ORDINALES).

DATO CARÁCTER:

Representa un conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato de tipo carácter contiene solo un carácter. (ORDINAL/SIMPLE).

DATO STRING:

Representa un conjunto finito de caracteres. Como máximo representa 256 caracteres. En general se utilizan para representar nombres. (COMPUESTO).

VARIABLES / CONSTANTE

VARIABLES: Es una zona de memoria cuyo contenido va a ser alguno de los tipos mencionados anteriormente. La dirección inicial de esta zona se asocia con el nombre de la variable. Puede cambiar su valor durante el programa.

CONSTANTES: Es una zona de memoria cuyo contenido va a ser alguno de los tipos mencionados anteriormente. La dirección inicial de esta zona se asocia con el nombre de la variable. NO puede cambiar su valor durante el programa.

PRE – POST CONDICIONES / READ – WRITE

PRE – CONDICION: Es la información que se conoce como verdadera antes de iniciar el programa (o módulo).

POST – CONDICION: Es la información que debería ser verdadera al concluir el programa (o módulo), si se cumplen adecuadamente los pasos especificados.

READ: Es una operación que contienen la mayoría de los lenguajes de programación. Se usa para tomar datos desde un dispositivo de entrada (por defecto desde teclado) y asignarlos a las variables correspondientes.

WRITE: Es una operación que contienen la mayoría de los lenguajes de programación. Se usa para mostrar el contenido de una variable, por defecto en pantalla.

ESTRUCTURA DE CONTROL / SECUENCIA

IF – CASE

ESTRUCTURA DE CONTROL: Todos los lenguajes de programación tienen un conjunto mínimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Como mínimo deben contener: secuencia, decisión e iteración.

SECUENCIA: La estructura de control más simple, está representada por una sucesión de operaciones (por ej. asignaciones), en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.

DECISION: En un algoritmo representativo de un problema real es necesario tomar decisiones en función de los datos del problema.

SELECCIÓN: Permite realizar distintas acciones dependiendo del valor de una variable de tipo ordinal.

ESTRUCTURA DE ITERACIÓN – CONTROL

WHILE – REPEAT UNTIL

ITERACIÓN: Puede ocurrir que se desee ejecutar un bloque de instrucciones desconociendo el número exacto de veces que se ejecutan.

Para estos casos existen en la mayoría de los lenguajes de programación estructurada las estructuras de control iterativas condicionales.

Como su nombre lo indica las acciones se ejecutan dependiendo de la evaluación de la condición.

Estas estructuras se clasifican en pre-condicionales y post-condicionales.

PRECONDICIONAL: Evalúa la condición y en caso de ser verdadera, ejecuta las acciones. Se repite mientras la condición es verdadera.

Puede ejecutarse 0, 1 o más veces.

POSTCONDICIONAL: Ejecuta las acciones y luego evalúa la condición. Se repite mientras la condición es falsa. Puede ejecutarse 1 o más veces.

ESTRUCTURA DE CONTROL REPETITIVA

FOR

REPETICION: Es una extensión natural de la secuencia. Consiste en repetir N veces un bloque de acciones. Este número de veces que se deben ejecutar las acciones es fijo y conocido de antemano.

TIPOS DE DATO DEFINIDOS POR EL USUARIO

TDU:

Un tipo de dato definido por el usuario es aquel que no existe en la definición del lenguaje, y el programador es el encargado de su especificación.

VENTAJAS:

- **FLEXIBILIDAD:** En el caso de ser necesario modificar la forma en que se representa el dato, sólo se debe modificar una declaración en lugar de un conjunto de declaraciones de variables.
- **DOCUMENTACIÓN:** Se pueden usar como identificador de los tipos, nombres auto explicativos, facilitando de esta manera el entendimiento y lectura del programa.
- **SEGURIDAD:** Se reducen los errores por uso de operaciones inadecuadas del dato a manejar, y se pueden obtener programas más confiables.

SUBRANGO: Es un tipo ordinal que consiste de una sucesión de valores de un tipo ordinal (predefinido o definido por el usuario) tomado como base.

OPERACIONES PERMITIDAS: Asignación; Comparación; Todas las operaciones permitidas para el tipo base.

OPERACIONES NO PERMITIDAS: Depende del tipo base.

MODULARIZACIÓN

MODULARIZAR: Significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos.

MODULO: Tarea específica bien definida. Se comunican entre sí adecuadamente y cooperan para conseguir un objetivo en común. Encapsula acciones tareas o funciones. En ellos se pueden representar los objetivos relevantes del problema a resolver.

VENTAJAS

- **MAYOR PRODUCTIVIDAD:** Al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, incrementando la productividad (es decir reduciendo el tiempo de desarrollo global del sistema).
- **REUSABILIDAD:** Un objetivo fundamental de la Ingeniería de Software es la reusabilidad, es decir la posibilidad de utilizar repetidamente el producto de software desarrollado. Naturalmente la descomposición funcional que ofrece la modularización favorece el reúso.
- **FACILIDAD DE CRECIMIENTO:** Los sistemas de software reales crecen (es decir aparecen con el tiempo nuevos requerimientos del usuario). La modularización permite disminuir los riesgos y costos de incorporar nuevas prestaciones a un sistema en funcionamiento.
- **LEGIBILIDAD:** Un efecto de la modularización es una mayor claridad para leer y comprender el código fuente. El ser humano maneja y comprende con mayor facilidad un número limitado de instrucciones directamente relacionadas.

PROCEDIMIENTOS:

Conjunto de instrucciones que realizan una tarea específica y retorna 0, 1 o más valores.

FUNCIÓN:

Conjunto de instrucciones que realizan una tarea específica y retorna 1 valor de tipo simple.

ALCANCE DE LAS VARIABLES

VARIABLES GLOBALES: Pueden ser usadas en todo el programa (incluyendo módulos).

VARIABLES LOCALES: Pueden ser usadas sólo en el cuerpo del programa.

VARIABLES LOCALES DEL PROCESO: Pueden ser usadas sólo en el proceso que están declaradas.

COMUNICACIÓN ENTRE MÓDULOS

PARAMETROS

- VALOR: El módulo recibe un valor, puede realizar operaciones y/o cálculos, pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo.
- REFERENCIA: El módulo recibe una dirección, puede realizar operaciones y/o cálculos, que producirán cambios y tendrán incidencia fuera del módulo.

ESTRUCTURA DE DATOS

Permite al programador definir un tipo al que se asocian diferentes datos que tienen valores lógicamente relacionados y asociados bajo un nombre único.

ELEMENTOS: Depende si los elementos son del mismo tipo o no.

- HOMOGÉNEA: Los elementos que la componen son del mismo tipo.
- HETEROGÉNEA: Los elementos que la componen pueden ser de distinto tipo.

TAMAÑO: Hace referencia a si la estructura puede variar su tamaño durante la ejecución del programa.

- ESTÁTICA: El tamaño de la estructura no varía durante la ejecución del programa.
- DINÁMICA: El tamaño de la estructura puede variar durante la ejecución del programa.

ACCESO: Hace referencia a como se pueden acceder a los elementos que la componen.

- SECUENCIAL: Para acceder a un elemento particular se debe respetar un orden predeterminado, por ejemplo, pasando por todos los elementos que le preceden, por ese orden.
- DIRECTO: Se puede acceder a un elemento particular, directamente, sin necesidad de pasar por los anteriores a él, por ejemplo, referenciando una posición.

LINEALIDAD: Hace referencia a como se encuentran almacenados los elementos que la componen.

- LINEAL: Está formada por ninguno, uno o varios elementos que guardan una relación de adyacencia ordenada donde a cada elemento le sigue uno y le precede uno, solamente.
- NO LINEAL: Para un elemento dado pueden existir 0, 1 o más elementos que le suceden y 0, 1 o más elementos que le preceden.

REGISTRO: Es un tipo de datos estructurado, que permite agrupar diferentes clases de datos en una estructura única bajo un sólo nombre.

La característica principal es que un registro permite representar la información en una única estructura.

ARRAY

Un arreglo (ARRAY) es una estructura de datos compuesta que permite acceder a cada componente por una variable índice, que da la posición de la componente dentro de la estructura de datos.

VECTOR: Es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de un índice.

HOMOGENEA: Los elementos pueden ser del mismo tipo.

ESTATICA: El tamaño no cambia durante la ejecución (se calcula en el momento de compilación).

INDEXADA: Para acceder a cada elemento de la estructura se debe utilizar una variable 'índice' que es de tipo ordinal.

El **RANGO** DEBE SER DE TIPO ORDINAL. (char, entero, booleano, subrango)

El **TIPO** DEBE SER UN TIPO ESTÁTICO. (char, entero, booleano, subrango, real, registro, vector).

RECORRIDOS

Consiste en recorrer el vector de manera total o parcial, para realizar algún proceso sobre sus elementos.

RECORRIDO – TOTAL: Implica analizar todos los elementos del vector, lo que lleva a recorrer completamente la estructura.

RECORRIDO – PARCIAL: Implica analizar los elementos del vector, hasta encontrar aquel que cumple con lo pedido. Puede ocurrir que se recorra todo el vector.

DIMENSIONES F / L

FISICA: Es la cantidad máxima de elementos que se pueden guardar en el arreglo. No puede modificarse durante la ejecución del programa.

LÓGICA: Es la cantidad máxima de elementos que se pueden guardar en el arreglo. No puede modificarse durante la ejecución del programa.

BUSQUEDAS

Significa recorrer el vector buscando un valor que puede o no estar en el vector. Se debe tener en cuenta que no es lo mismo buscar en un vector ordenado que en uno que no lo esté.

VECTOR DESORDENADO: Se debe recorrer todo el vector (en el peor de los casos), y detener la búsqueda en el momento que se encuentra el dato buscado o en el que se terminó el vector.

VECTOR ORDENADO: Se debe recorrer el vector teniendo en cuenta el orden:

BUSQUEDA MEJORADA / BUSQUEDA BINARIA

ALOCACION DE MEMORIA

Hasta ahora, cualquier variable que se declare en un programa es alojada en la memoria estática de la CPU.

Hasta ahora, cualquier variable que se declare en un programa es alojada en la memoria estática de la CPU.

Obviamente al permanecer en la memoria siguen ocupando memoria.

MEMORIA ESTATICA: char, boolean, integer, real, string, subrango, registro, vector.

Tipo de variable	Bytes que ocupa
Char	1 byte
Boolean	1 byte
Integer	6 bytes
Real	8 bytes
String	Tamaño + 1
Subrango	Depende el tipo
Registro	La suma de sus campos
Vector	Dimensión física * tipo elemento

Para solucionar los problemas mencionados anteriormente los lenguajes permiten la utilización de tipos de datos que permiten reservar y liberar memoria dinámica durante la ejecución del programa a medida que el programador lo requiera.

PUNTERO

Es un tipo de variable usada para almacenar una dirección en memoria dinámica. En esa dirección de memoria se encuentra el valor real que almacena. El valor puede ser de cualquiera de los tipos vistos (char, boolean, integer, real, string, registro, arreglo u otro puntero).

Un puntero es un tipo de datos simple.

Siempre ocupa 4 bytes de memoria estática.

Cuando quiere cargar contenido reserva memoria dinámica y cuando no necesita más el contenido la libera.

Cuando la variable puntero reserve memoria ahí se ocupará la memoria dinámica (la cantidad de bytes de memoria dinámica dependerá del tipo de elementos que maneje el puntero).

CREACION: Implica reservar una dirección memoria dinámica libre para poder asignarle contenidos a la dirección que contiene la variable de tipo puntero.
new(variable tipo puntero).

ELIMINACIÓN: Implica liberar la memoria dinámica que contenía la variable de tipo puntero. dispose(variable tipo puntero).

LIBERACIÓN: Implica cortar el enlace que existe con la memoria dinámica. La misma queda ocupada pero ya no se puede acceder. Nil

DISPOSE (p)

Libera la conexión que existe entre la variable y la posición de memoria.

Libera la posición de memoria.

La memoria liberada puede utilizarse en otro momento del programa.



p:=nil

Libera la conexión que existe entre la variable y la posición de memoria.

La memoria sigue ocupada.

La memoria no se puede referenciar ni utilizar.

ASIGNACIÓN entre punteros: Implica asignar la dirección de un puntero a otra variable puntero del mismo tipo. :=

CONTENIDO de un puntero: Implica poder acceder al contenido que contiene la dirección de memoria que tiene una variable de tipo puntero. ^

- if (p = nil) then, compara si el puntero p no tiene dirección asignada.
- if (p = q) then, compara si los punteros p y q apuntan a la misma dirección de memoria.
- if (p^ = q^) then, compara si los punteros p y q tienen el mismo contenido.
- no se puede hacer read (p), ni write (p), siendo p una variable puntero.
- no se puede asignar una dirección de manera directa a un puntero, p:= ABCD
- no se pueden comparar las direcciones de dos punteros (p<q).

MEMORIA DE UN PROGRAMA

MEMORIA ESTÁTICA: A modo de simplicidad consideraremos sólo las variables locales, variables globales de programa y constantes.

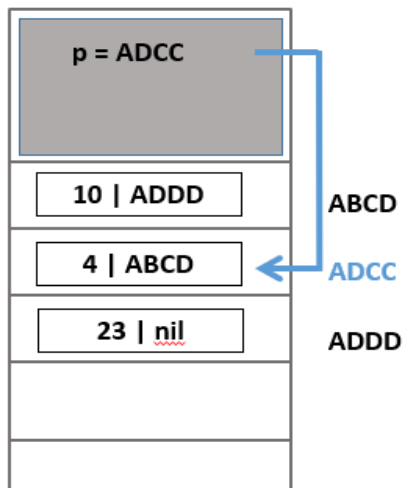
MEMORIA DINÁMICA: A modo de simplicidad consideraremos sólo cuando en la ejecución de un programa se reserva o libera memoria.

LISTA

Es una colección de nodos. Cada nodo contiene un elemento (valor que se quiere almacenar en la lista) y una dirección de memoria dinámica que indica donde se encuentra el siguiente nodo de la lista.

Toda lista tiene un nodo inicial.

Los nodos que la componen pueden no ocupar posiciones contiguas de memoria. Es decir, pueden aparecer dispersos en la memoria, pero mantienen un orden lógico interno.



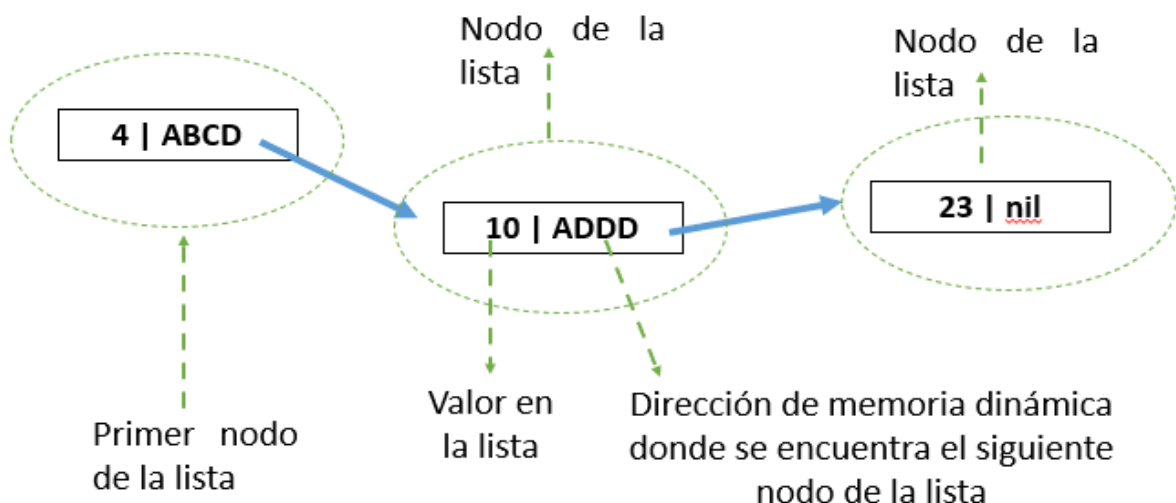
fase 9-1

En **memoria estática** se declara una variable tipo PUNTERO (ya que son las únicas que pueden almacenar direcciones). La dirección almacenada en esa variable representa la dirección donde comienza la lista. Inicialmente ese puntero no contiene ninguna dirección.



Luego a medida que se quiere agregar elementos a la lista (nodo), se reserva una dirección de **memoria dinámica** y se carga el valor que se quiere guardar.

El último nodo de la lista indica que la dirección que le sigue es nil.



CARACTERISTICAS:

HOMOGENEA: Los elementos pueden ser del mismo tipo.

DINAMICA: El tamaño puede cambiar durante la ejecución del programa.

LINEAL: Cada nodo de la lista tiene un nodo que lo sigue (salvo el último) y uno que lo antecede (salvo el primero).

SECUENCIAL: El acceso a cada elemento es de manera secuencial, es decir, para acceder al elemento 5 (por ejemplo) debo pasar por los 4 anteriores.

CREACIÓN: Implica marcar que la lista no tiene una dirección inicial de comienzo.

RECORRIDO: Implica posicionarse al comienzo de la lista y a partir de allí ir “pasando” por cada elemento de la misma hasta llegar al final.

AGREGAR ADELANTE: Implica generar un nuevo nodo y agregarlo como primer elemento de la lista.

AGREGAR AL FINAL: Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

***RECORDAR:** NIL ES DONDE ESTÁ UBICADO EL FINAL DE LA LISTA*

BUSCAR UN ELEMENTO: Significa recorrer la lista desde el primer nodo buscando un valor que puede o no estar. Se debe tener en cuenta si la lista está o no ordenada.

LISTA DESORDENADA: Se debe recorrer toda la lista (en el peor de los casos), y detener la búsqueda en el momento que se encuentra el dato buscado o en el que la lista se terminó.

LISTA ORDENADA: Se debe recorrer la lista teniendo en cuenta el orden. La búsqueda se detiene cuando se termina la lista o el elemento buscado es mayor al elemento actual.

ELIMINAR ELEMENTO: Implica recorrer la lista desde el comienzo pasando nodo a nodo hasta encontrar el elemento y en ese momento eliminarlo (dispose). El elemento puede no estar en la lista.

- Si la lista está desordenada seguramente la búsqueda se realizará hasta encontrar el elemento o hasta que se termina la lista.

- Si la lista está ordenada seguramente la búsqueda se realizará hasta que se termina la lista o no se encuentre un elemento mayor al buscado.

INSERTAR ELEMENTO: Implica agregar un nuevo nodo a una lista ordenada por algún criterio de manera que la lista siga quedando ordenada.

Existen 4 casos:

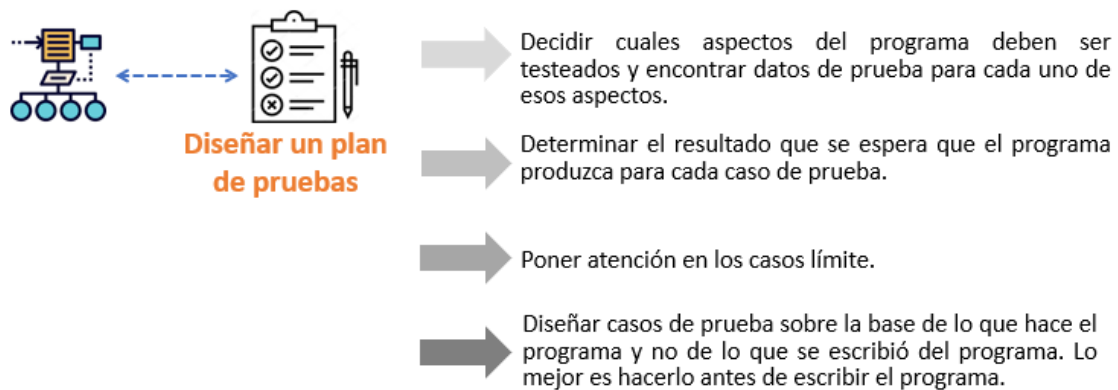
- que la lista esté vacía.
- que elemento vaya al comienzo de la lista (es menor al 1er nodo de la lista)
- que elemento vaya al “medio” de la lista (es menor al último nodo de la lista)
- que elemento vaya al final de la lista (es mayor al último nodo de la lista)

CORRECCIÓN DE PROGRAMAS

Cuando se desarrollan los algoritmos hay dos conceptos importantes que se deben tener en cuenta: CORRECCIÓN y EFICIENCIA del programa.

Un programa es correcto si se realiza de acuerdo a sus especificaciones.

TESTING: El propósito del Testing es proveer evidencias convincentes que el programa hace el trabajo esperado.



Isaa 11.1

Una vez que el programa ha sido implementado y se tiene el plan de pruebas:

- Se analiza el programa con los casos de prueba.
- Si hay errores se corrigen.

Estos dos pasos se repiten hasta que no haya errores.

DEBUGGING: Es el proceso de descubrir y reparar la causa del error.

Para esto pueden agregarse sentencias adicionales en el programa que permiten monitorear el comportamiento más cercanamente.

Los **errores** encontrados pueden ser de **tres tipos:**

SINTACTICOS: Se detectan en la compilación.

LOGICOS: Generalmente se detectan en la ejecución.

DE SISTEMA: Son muy raros los casos en los que ocurren.

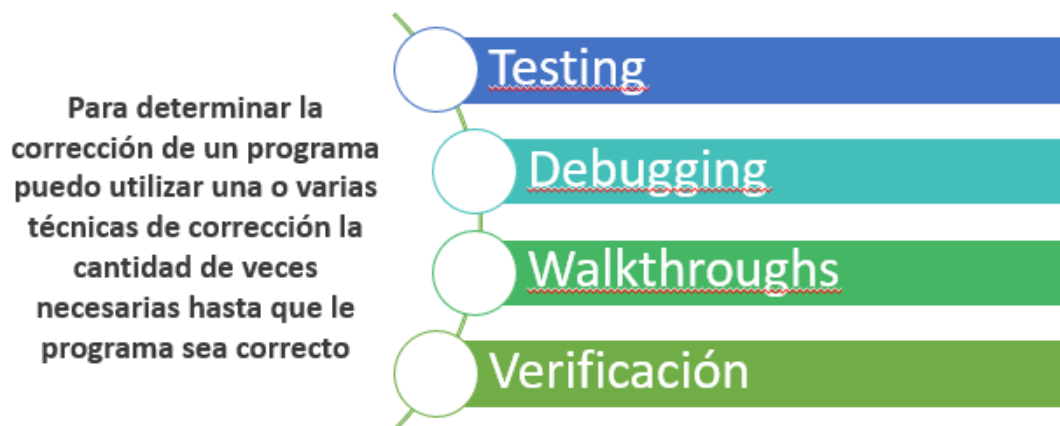
WALKTHROUGHS: Es el proceso de recorrer un programa frente a una audiencia.

La lectura de un programa a alguna otra persona provee un buen medio para detectar errores.

Esta persona no comparte preconceptos y está predispuesta a descubrir errores u omisiones.

A menudo, cuando no se puede detectar un error, el programador trata de probar que no existe, pero mientras lo hace, puede detectar el error, o bien puede que el otro lo encuentre.

VERIFICACIÓN: Es el proceso de controlar que se cumplan las pre y post condiciones del mismo.



EFICIENCIA DE PROGRAMAS

Una vez que se obtiene un algoritmo y se verifica que es correcto, es importante determinar la eficiencia del mismo.

El análisis de la eficiencia de un algoritmo estudia el tiempo de ejecución de un algoritmo y la memoria que requiere para su ejecución.



lase 11-2

El análisis de la eficiencia de un algoritmo estudia el tiempo de ejecución de un algoritmo y la memoria que requiere para su ejecución.

MEMORIA: Se calcula (como hemos visto previamente) teniendo en cuenta la cantidad de bytes que ocupa la declaración en el programa de:

constante/s ||||| variable/s global/es ||||| variable/s local al programa/es

TIEMPO DE EJECUCION: puede calcularse haciendo un análisis empírico o un análisis teórico del programa.


El tiempo de un algoritmo puede definirse como una función de entrada:

Existen algoritmos que el tiempo de ejecución no depende de las características de los datos de entrada sino de la cantidad de datos de entrada o su tamaño.

Existen otros algoritmos donde el tiempo de ejecución es una función de la entrada “específica”, en estos casos se habla del tiempo de ejecución del “peor” caso. En estos casos, se obtiene una cota superior del tiempo de ejecución para cualquier entrada.

ANALISIS EMPIRICO

Requiere la implementación del programa, luego ejecutar el programa en la máquina y medir el tiempo consumido para su ejecución.



Fácil de realizar.

Obtiene valores exactos para una máquina determinada y unos datos determinados.

Completamente dependiente de la máquina donde se ejecuta

Requiere implementar el algoritmo y ejecutarlo repetidas veces (para luego calcular un promedio).

se 11-2

ANALISIS TEÓRICO

Implica encontrar una cota máxima (“peor caso”) para expresar el tiempo de nuestro algoritmo, sin necesidad de ejecutarlo.

A partir de un programa correcto, se obtiene el tiempo teórico del algoritmo y luego el orden de ejecución del mismo. Lo que se compara entre algoritmos es el orden de ejecución.

Dado un algoritmo que es correcto se calcula el tiempo de ejecución de cada una de sus instrucciones. Para eso se va a considerar:

Una instrucción elemental utiliza un tiempo constante para su ejecución, independientemente del tipo de dato con el que trabaje. 1UT.