TEORIA 14

TEMAS de la CLASE

1 Corrección y Eficiencia

- Concepto Corrección
  - Técnicas para medir la corrección de un programa
- Concepto de Eficiencia
  - Métodos para medir la eficiencia de un programa

Recordemos las etapas cumplidas hasta llegar a la escritura del programa...

Problema del Mundo Real

Análisis

Modelo

Diseño

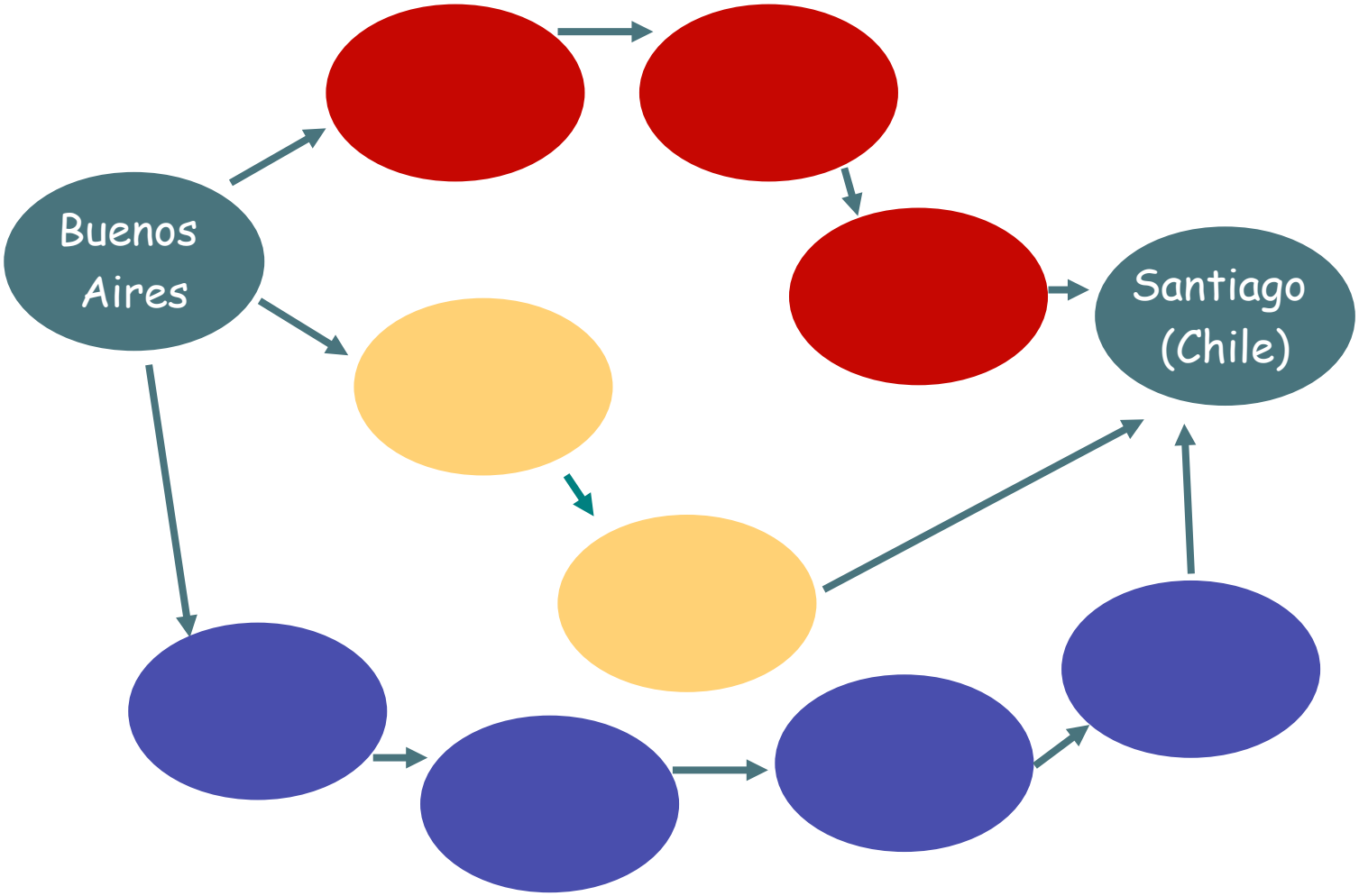
Algoritmos

Implementación

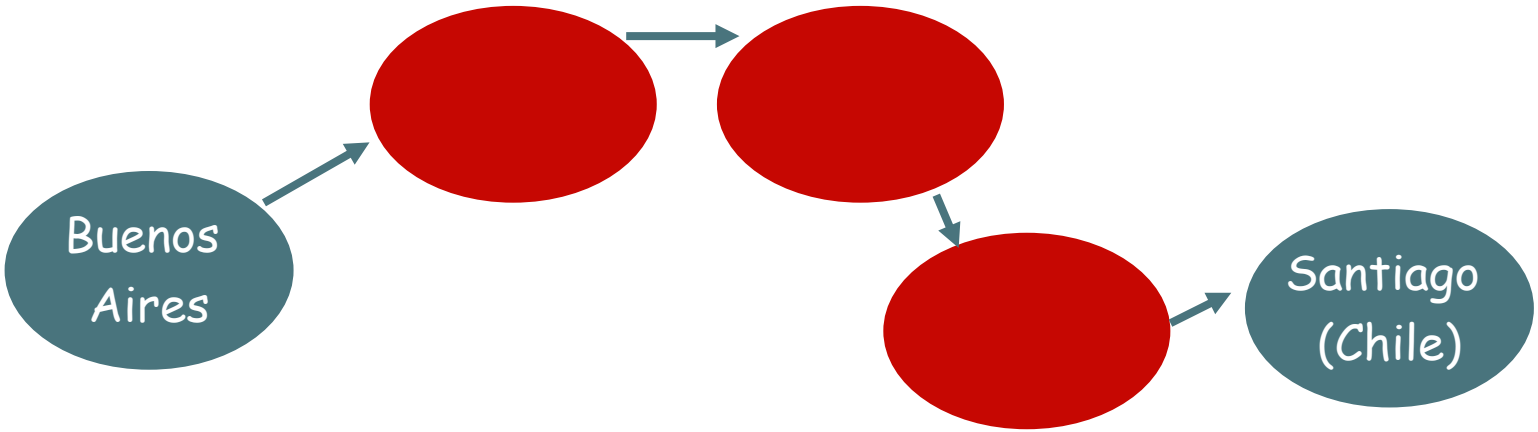
Programa

Verificación Del programa

Motivación de la Clase de Hoy – Nos vamos de viaje

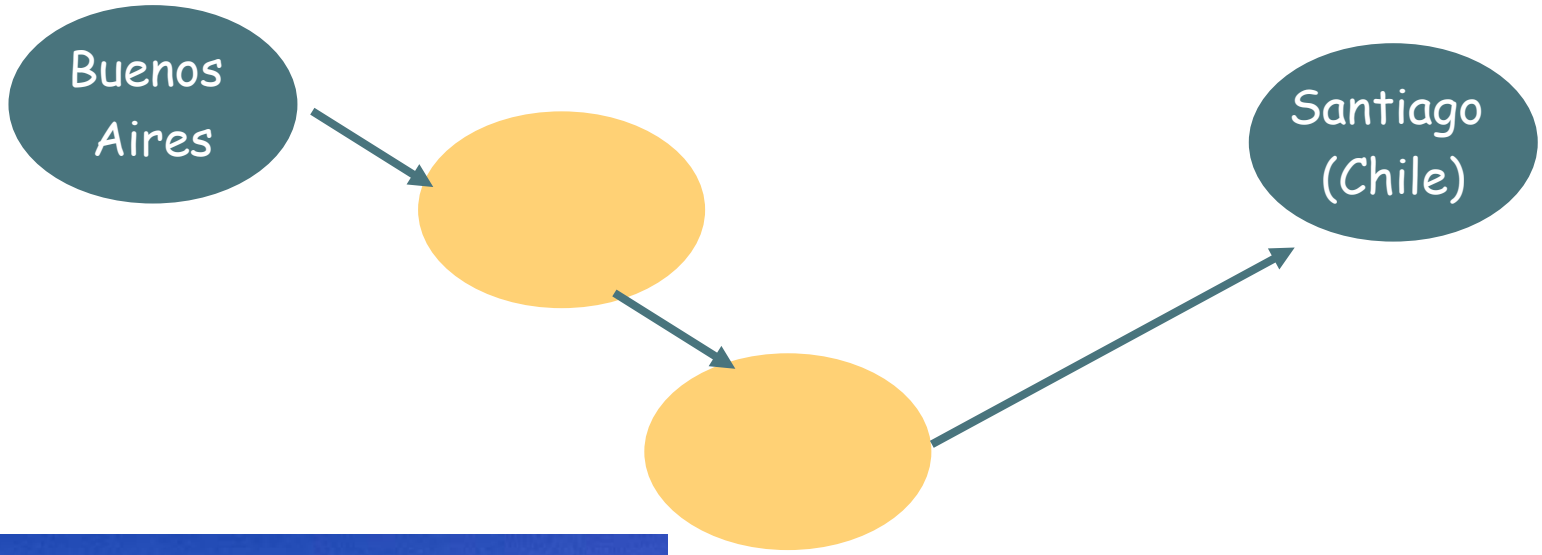


Viajamos de Bs.As a Santiago de Chile



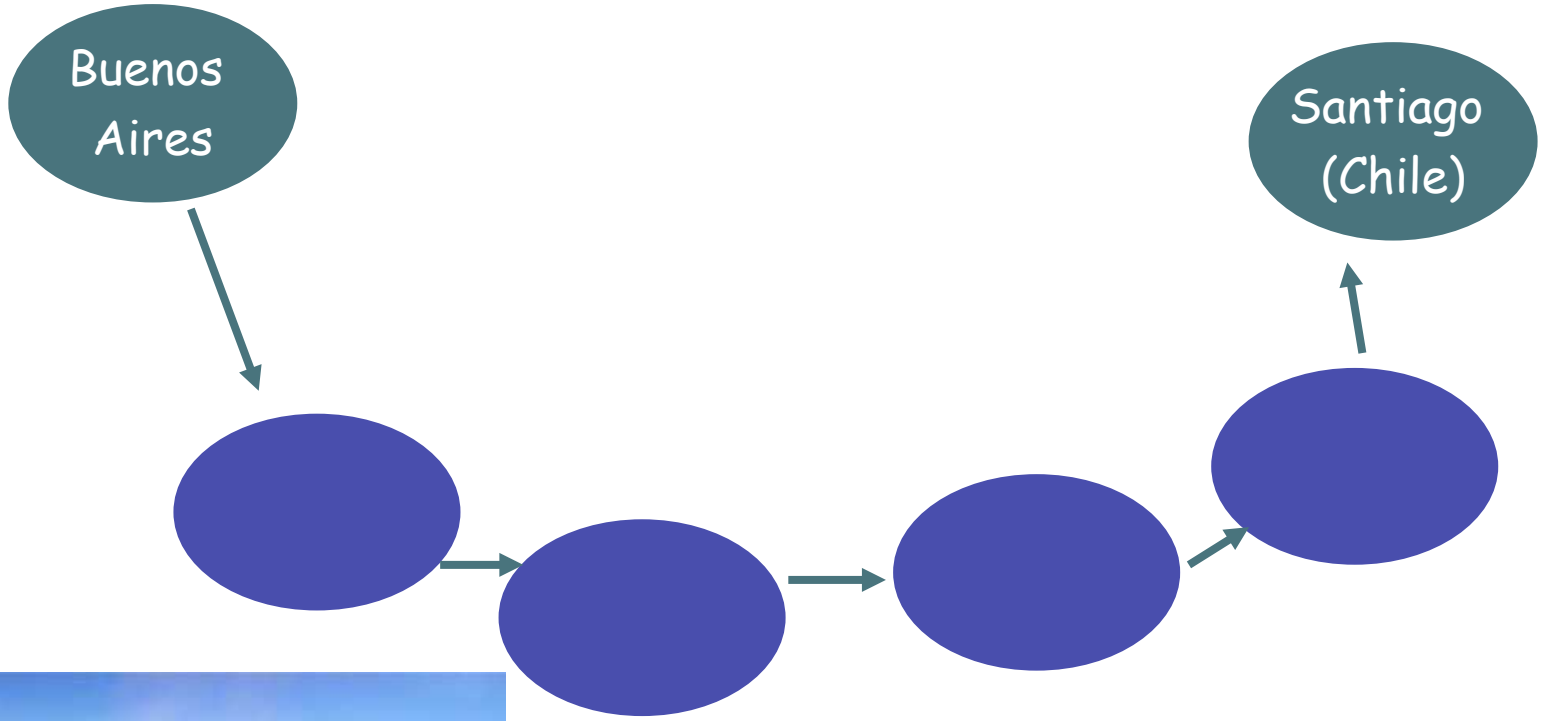
El camino rojo (Cruce por Mendoza) es mas largo pero está en muy buen estado y puedo visitar lugares turísticos (Aconcagua, museos, etc).

Viajamos de Bs.As a Santiago de Chile



El camino amarillo (Cruce por San Juan) es mas corto, menos transitado y tiene muy lindas vistas pero tiene tramos sin asfaltar.

Viajamos de Bs.As a Santiago de Chile



El camino azul (por aire) es el mas corto en tiempo pero hace escala en varias ciudades.

CADP

Bs.As - Santiago de Chile ¿Cuál camino elijo?

CADP

Bs.As - Santiago de Chile ¿Cuál camino elijo?

- Cuando debemos tomar decisiones, existe siempre una serie de factores que se deben analizar y buscar optimizar.

- En algunos casos, debemos buscar optimizar algunos en pos de sacrificar otros, dependiendo del contexto y las necesidades específicas de la situación.

Calidad de un Programa

Calidad de un Programa

Corrección (¿Hace lo que se le pide?)

El grado en que una aplicación satisface sus especificaciones y consigue los objetivos encomendados por el cliente

Fiabilidad (¿Lo hace de forma fiable todo el tiempo?)

El grado que se puede esperar que una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida

Eficiencia (¿Qué recursos hardware y software necesito?)

La cantidad de recursos hardware y software que necesita una aplicación para realizar las operaciones con los tiempos de respuesta adecuados

Calidad de un Programa

Integridad (¿Puedo controlar su uso?)

El grado con que puede controlarse el acceso al software o a los datos a personal no autorizado

Facilidad de uso (¿Es fácil y cómodo de manejar?)

El esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados

Facilidad de mantenimiento (¿Puedo localizar los fallos?)

El esfuerzo requerido para localizar y reparar errores → Se va a vincular con la modularización y con cuestiones de legibilidad y documentación.

Calidad de un Programa

- **Flexibilidad (¿Puedo añadir nuevas opciones?)**

  El esfuerzo requerido para modificar una aplicación en funcionamiento

- **Facilidad de prueba (¿Puedo probar todas las opciones?)**

  El esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos

- **Portabilidad (¿Podré usarlo en otra máquina?)**

  El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo

Calidad de un Programa

- **Reusabilidad (¿Podré utilizar alguna parte del software en otra aplicación?)**

  Grado en que las partes de una aplicación pueden utilizarse en otras aplicaciones

- **Interoperabilidad (¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?**

  El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos

- **Legibilidad :** El código fuente de un programa debe ser fácil de leer y entender. Esto obliga a acompañar a las instrucciones con comentarios adecuados. Relacionado con la presentación de documentación.

Calidad de un Programa

Documentación: todo el proceso de análisis y diseño del problema y su solución debe estar documentado mediante texto y/o gráficos para favorecer la comprensión, la modificación y la adaptación a nuevas funciones.

Un programa bien documentado será fácil de leer y mantener.

Se aconseja la inserción de comentarios en el programa.

Los identificadores se deberán elegir de manera tal que sean autoexplicativos

Es recomendable realizar un comentario general del objetivo del programa

Cuando se realiza el mantenimiento de un programa no sólo se actualiza el código, sino también los comentarios del programa.

Los comentarios intercalados en el programa (documentación on line), deben realizarse con criterio para contribuir a la claridad del programa.

Vamos a trabajar con dos conceptos fundamentales a la hora de programar:

- **Corrección**
- **Eficiencia**

Corrección de Programas

Una vez escrita una posible solución es necesario verificar que cumple con el objetivo propuesto.

Esta tarea se conoce como:

Corrección de programa

Corrección de Programas

- **Una vez que se ha escrito un programa se debe probar que el mismo es correcto.**
- **Un programa es correcto si se realiza de acuerdo a sus especificaciones.**
- **Por esta razón es muy importante una completa especificación.**

Corrección de Programas

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

Técnicas que asisten al programador:

- ✓ **TEST**
- ✓ **WALKTHROUGH**
- ✓ **DEBUGGING**

Estas técnicas usadas complementariamente proveen evidencias para la corrección.

Corrección de Programas – Técnica de Testing

La técnica de Testing es el proceso mediante el cual se proveen evidencias convincentes respecto a que el programa hace el trabajo esperado.

¿Como se proveen evidencias?

✓Diseñar un plan de pruebas.	✓Poner atención en los casos límite
✓Decidir cuales aspectos del programa deben ser testeados y encontrar datos de prueba para cada uno de esos aspectos.	✓Diseñar casos de prueba sobre la base de lo que hace el programa y no de lo que se escribió del programa.
✓Determinar el resultado que se espera que el programa produzca para cada caso de prueba	✓Mejor aún, diseñar casos de prueba antes de que comience la escritura del programa. (Esto asegura que las pruebas no están pensadas a favor del que escribió el programa)

Cuando se tiene el plan de pruebas y el programa, el plan debe aplicarse sistemáticamente.

Corrección de Programas – Técnica de Testing

Durante este proceso es importante analizar las postcondiciones en función de las precondiciones establecidas.

- **Las precondiciones, junto con las postcondiciones, permiten describir la función que realiza un programa, sin especificar un algoritmo determinado.**

- **Las precondiciones describen los aspectos que se consideran verdaderos antes que el programa comience a ejecutarse, por ejemplo: entradas de datos disponibles.**

- **Las postcondiciones describen los aspectos que deben cumplirse cuando el programa terminó.**

Corrección de Programas – Técnica de Debugging

La técnica de Debugging es el proceso de localización del error

Puede involucrar:

- el diseño y aplicación de pruebas adicionales para ubicar y conocer la naturaleza del error.
- el agregado de sentencias adicionales en el programa para poder monitorear su comportamiento más cercana

Los errores pueden provenir de varios caminos, por ejemplo:

- El diseño del programa puede ser defectuoso.
- El programa puede usar un algoritmo defectuoso.

A veces el error es tan evidente que se reconoce rápidamente en qué lugar está la falla.

Otras veces se puede necesitar agregar sentencias de salida adicionales que sirven como punto de control o para señalar cambios en ciertas variables claves.

Corrección de Programas – Técnica de Walkthroughs

La técnica de Walkthroughs consiste en recorrer el programa ante 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.

Eficiencia de Programas

Una vez que se cuenta con una solución correcta es necesario medir cuántos recursos se utilizan. En particular aquí se analizan: Tiempo de Ejecución y Memoria utilizada.

- Para cada problema se pueden tener varias soluciones algorítmicas correctas,
- Sin embargo el uso de recursos (tiempo, memoria) de cada una de esas soluciones puede ser muy diferente.

Eficiencia de un Algoritmo

Se define la eficiencia como una métrica de calidad de los algoritmos, asociada con una utilización óptima de los recursos del sistema de cómputo donde se ejecutará el programa, principalmente la memoria utilizada y el tiempo de ejecución empleado.

Eficiencia de un Algoritmo

Una vez que se obtiene un algoritmo y se comprueba que es correcto, es importante determinar cuántos recursos, como tiempo de ejecución o espacio en memoria, se requiere para la solución del problema.

Por lo tanto, si se quieren optimizar los recursos de memoria y tiempo, cabe preguntarse:

- **¿Cómo calcular el espacio ocupado por un programa?**
- **¿Cómo calcular el tiempo requerido por un programa ?**

En caso de ser necesario se podrá analizar:

- **¿Cómo reducir el espacio ocupado por un programa?**
- **¿Cómo reducir el tiempo de ejecución de un programa?**

Eficiencia de un Algoritmo

Medición de la Memoria utilizada en un programa

- Se puede calcular únicamente la cantidad de memoria estática que utiliza el programa.

- Se analizan las variables declaradas y el tipo correspondiente.

¿Cuánta memoria se utiliza?

```
{declaración de tipos}
Type
cadena10 = string [10];
PtrString = ^cadena10;
Datos = record
            Nombre: cadena10;
            Apellido: cadena10;
            Edad: integer;
            Altura: real
        End;
Personas = array [1..100] of datos;
PtrDatos = ^datos;

var
  frase : PtrString;
  s : cadena10;
  puntero : PtrDatos;
  p : personas;
```

Eficiencia de un Algoritmo

Medición del Tiempo de ejecución de un programa

Depende de distintos factores:

- Los datos de entrada al programa
  - Tamaño
  - Contenido
- La calidad del código generado por el compilador utilizado
- La naturaleza y rapidez de las instrucciones de máquina empleadas en la ejecución del programa
- El tiempo del algoritmo base.

Eficiencia

El tiempo de ejecución de un programa debe definirse como una función de la cantidad de datos de entrada.

- Para algunos programas, el tiempo de ejecución se refiere al tiempo de ejecución del "peor" caso. En estos casos, se obtiene una cota superior del tiempo de ejecución para cualquier entrada.

- Ejemplos: Problema de búsqueda secuencial en vectores y listas.

Eficiencia

El tiempo de ejecución de un programa puede calcularse de dos maneras:

- Análisis Empírico
- Análisis Teórico

Eficiencia – Análisis empírico

Para realizar un análisis empírico, es necesario ejecutar el programa y medir el tiempo empleado en la ejecución

Inconveniente: este análisis tiene varias limitaciones porque puede dar una información pobre de los recursos consumidos:

- Obtiene valores exactos para una máquina y unos datos determinados
- Es completamente dependiente de la máquina donde se ejecuta
- Requiere implementar el algoritmo y ejecutarlo repetidas veces.

Eficiencia – Análisis Teórico

Para realizar un análisis teórico, es necesario establecer una medida intrínseca de la cantidad de trabajo realizado por el algoritmo. Esto nos permite comparar algoritmos y seleccionar la mejor implementación.

Ventajas

- Obtiene valores aproximados
- Es aplicable en la etapa de diseño de los algoritmos, uno de los aspectos fundamentales a tener en cuenta. Se puede aplicar sin necesidad de implementar el algoritmo.
- Independiente de la máquina donde se ejecute
- Permite analizar el comportamiento

Eficiencia – Análisis Teórico

Consideraciones generales para tener en cuenta al hacer el cálculo teórico:

- Considerar el número de operaciones elementales que emplea el algoritmo.
- Considerar que una operación elemental utiliza un tiempo constante para su ejecución, independientemente del tipo de dato con el que trabaje.
- Suponer que cada operación elemental se ejecutará en una unidad de tiempo (dejando de lado la magnitud).
- Suponer que una operación elemental es una asignación, una comparación o una operación aritmética simple.

Eficiencia – Análisis Teórico

Reglas Generales para el cálculo del tiempo de ejecución

- **Regla 1: For**
- **Regla 2: For anidados**
- **Regla 3: sentencias consecutivas**
- **Regla 4: If / else**

Los comentarios, declaraciones y operaciones de entrada/salida (Read / Write), no se consideran al realizar el cálculo

Eficiencia – Análisis Teórico

Recordemos un ejemplo ya visto: Se necesita conocer la cantidad de veces que aparece la temperatura con valor 10 en un vector de temperaturas.

```
Type temperaturas = array [1..30] of real;

Function contar ( tem:temperaturas): integer;
Var i: 1..30; can10 : integer;
begin
  can10 := 0;
  {recorrido total del vector}
  For  i  := 1  to  30 do
    If  (tem [i] = 10 ) then  can10 := can10 + 1;
  contar := can10;
end.
```

¿Memoria utilizada por el módulo?

¿Tiempo empleado por el módulo?

Eficiencia – Análisis Teórico

- Cálculo Teórico del tiempo de ejecución:

```
Type temperaturas = array [1..30] of real;

Function contar ( tem:temperaturas): integer;
 Var i: 1..30; can10 : integer;
 begin
   can10 := 0;                          {1}
   {recorrido total del vector}
   For  i  := 1  to  30 do              {2}
     If (tem[i] = 10) then              {3}
                          can10 := can10 + 1; {4}
   contar := can10;                     {5}
 end.
```

- Las líneas {1} y {5} cuentan una unidad cada una, entonces tenemos **2**
- La línea {3} evalúa una condición, cuenta **1** unidad y la línea {4} cuenta **2** unidades. Por la tanto, cada vez que se ejecutan utilizan 3 unidades -> **3 * 30**
- La línea {2} tiene una inicialización, testeo de i <=30 e incremento de i, entonces **1** de la asignación, más **31** para todos los test y **30*2** para el incremento, entonces 1+ 31 + 60 = 92
- Total = 2 + 90 + 92 (como máximo!!!) ¿Por qué?

Eficiencia – Análisis Teórico

Supongamos que disponemos de un módulo que calcula la suma de los n primeros números naturales.

¿Memoria?

¿Cantidad de operaciones?

```
Function sumar ( n : integer): integer;
  Var total: integer; i: integer;
  begin
    total := 0;                  {1}
   For  i  := 1  to  n do        {2}
        total := total + i;      {3}
   sumar := total;               {4}
  end;
```

- {1} -> 1 asignación = 1 unidad de tiempo
- {4} → 1 asignación = 1 unidad de tiempo
- {3} → (1 asignación + 1 suma) * n = 2 * n
- {2} → una asignación (i:=1) + testeos de i<=n + incremento de i (i:= i+1)
  → 1 + (n + 1) + 2*n = 3*n + 2
- **T(N) ==> total = 1 + 1 + 2*n + 3*n + 2 = 5*n + 4 ➔ O(n)**

Eficiencia – Ejemplos

Consideremos los siguientes bloques de instrucciones:

```
For i:= 'A' to 'Z' do
  cont [i]:= 0;
```

```
contA:= 0;
contE := 0;
contI  := 0;
contO := 0;
contU := 0;
```

- ¿Cuántas unidades de tiempo se consumen en cada bloque?
- ¿Se puede establecer una relación directa entre la cantidad de líneas de código y el tiempo de ejecución?

NO!!!

Eficiencia – Ejemplos

Analice el tiempo de ejecución del siguiente programa:

```
Program uno;
Var
  valor, i, j, suma :integer;
 Begin
  read (valor);
  if (valor >8) then begin
                       suma:=0;
                       for i:= 1 to 3000 do
                         suma:= suma + I;
                    end
                    else begin
                        suma:=0;
                        for j:= 1 to 300 do
                           for i:= 1 to 200 do
                              suma:= suma + I;
                    end;
   end;
End.
```

Recordar

¿Cantidad de operaciones?

Regla del FOR

Regla del IF

Eficiencia en soluciones modularizadas

Cálculo del tiempo de ejecución en una solución modularizada

- Si se tiene un programa con módulos, es posible calcular el tiempo de ejecución de los distintos procesos, uno a la vez, partiendo de aquellos que no llaman a otros. Debe haber al menos un módulo con esa característica.

- Después puede evaluarse el tiempo de ejecución de los procesos que llamaron a los módulos anteriores y así sucesivamente.

Eficiencia – Tiempo de ejecución- Conclusiones

Algunas condiciones en las cuales el tiempo de ejecución de un programa se puede ignorar en favor de otros factores:

- Si un programa se va a utilizar algunas veces, el costo de su escritura y depuración es el dominante.
- Si un programa se va a ejecutar sólo con entradas "pequeñas", la velocidad de crecimiento puede ser menos importante.
- Un algoritmo eficiente pero complicado puede no ser apropiado para el mantenimiento por parte de un tercero.

Pensemos…

¿Los siguientes conceptos influyen o no sobre el tiempo de ejecución de un programa?

- Cantidad de datos de entrada
- Cantidad de líneas de código
- Orden en que se ingresan o aparecen los datos
- Cantidad de iteraciones presentes en el programa
- Uso de variables globales
- Uso de variables locales
- Uso de parámetros