

Programación I

● 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
Solución

Problema del
Mundo Real

Análisis

Modelo

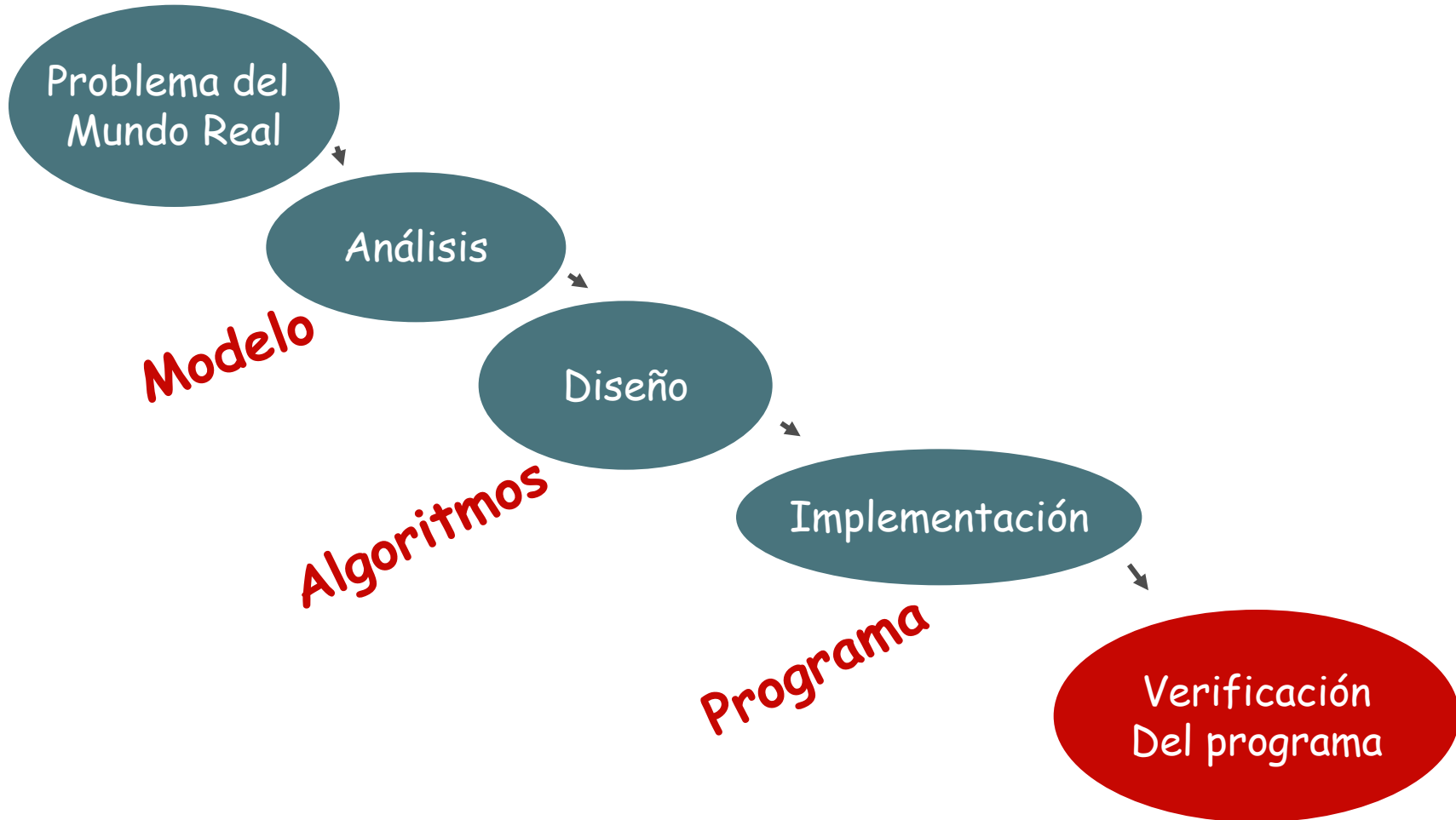
Diseño

Algoritmos

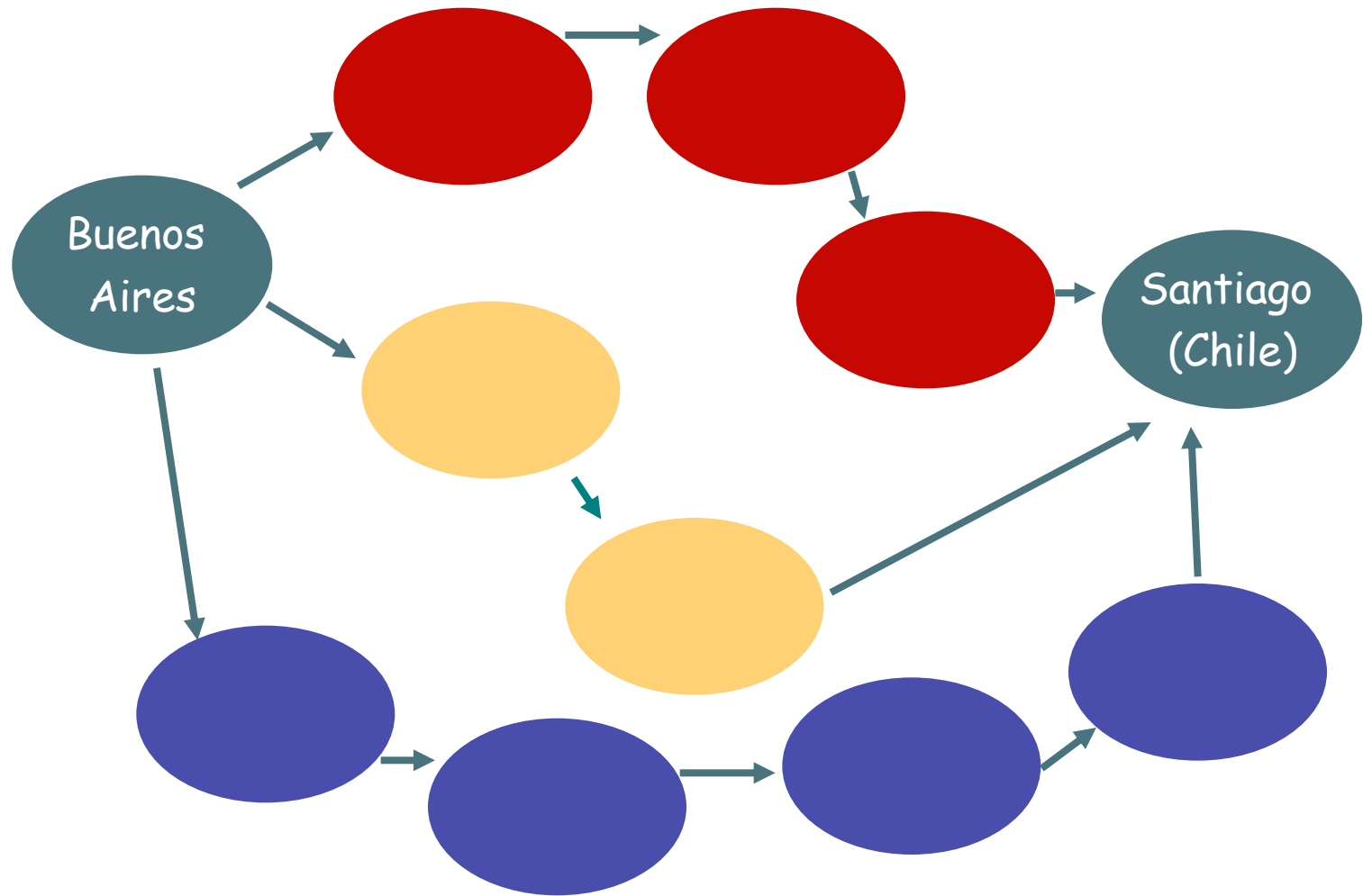
Implementación

Programa

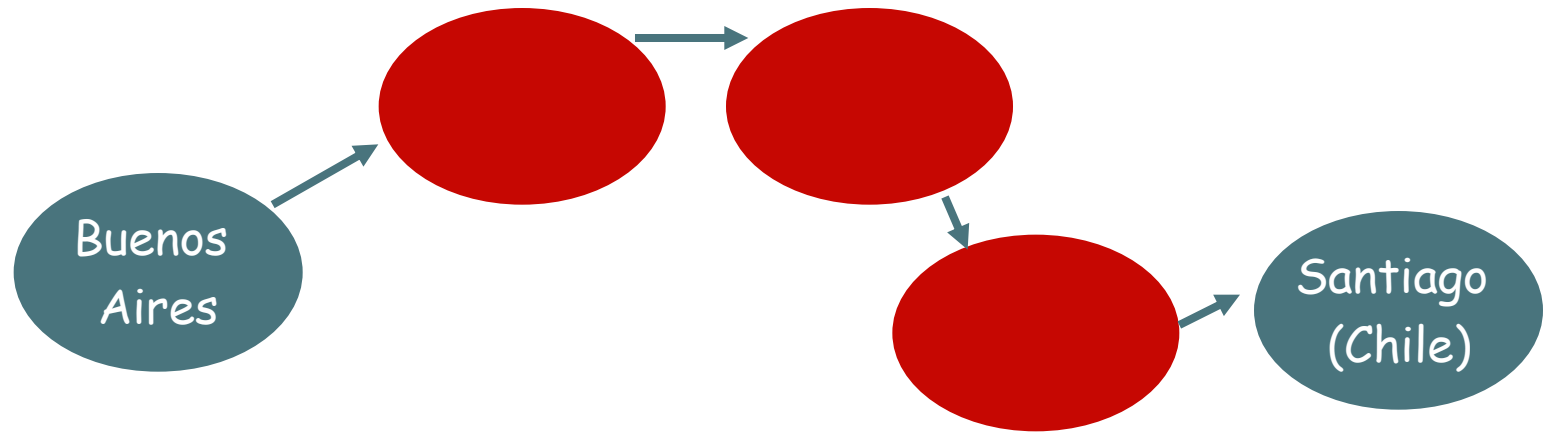
Verificación
Del programa



Motivación de la Clase de Hoy – Nos vamos de viaje

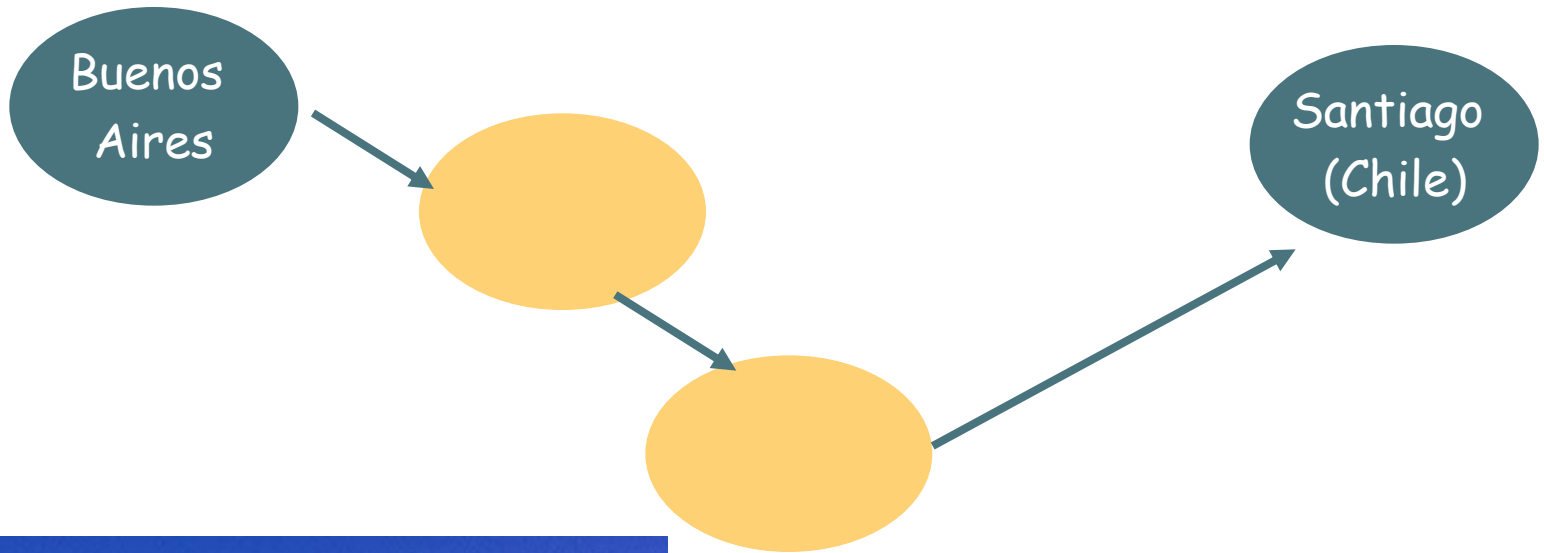


Viajamos de Bs.As a Santiago de Chile



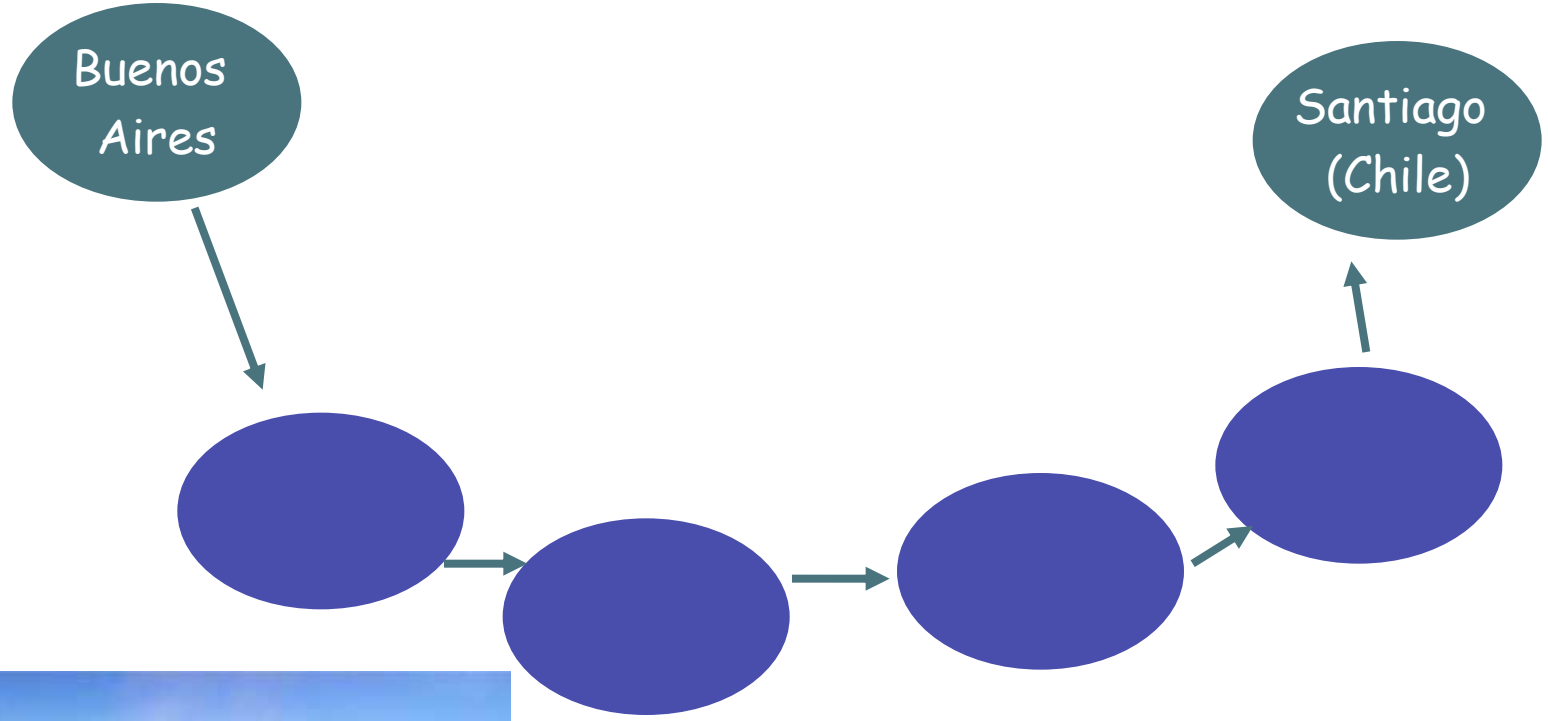
El camino rojo (Cruce por Mendoza) es mas largo pero está en muy buen estado y puedo visitar lugares turísticos (Aconcagua, museos, etc).

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.

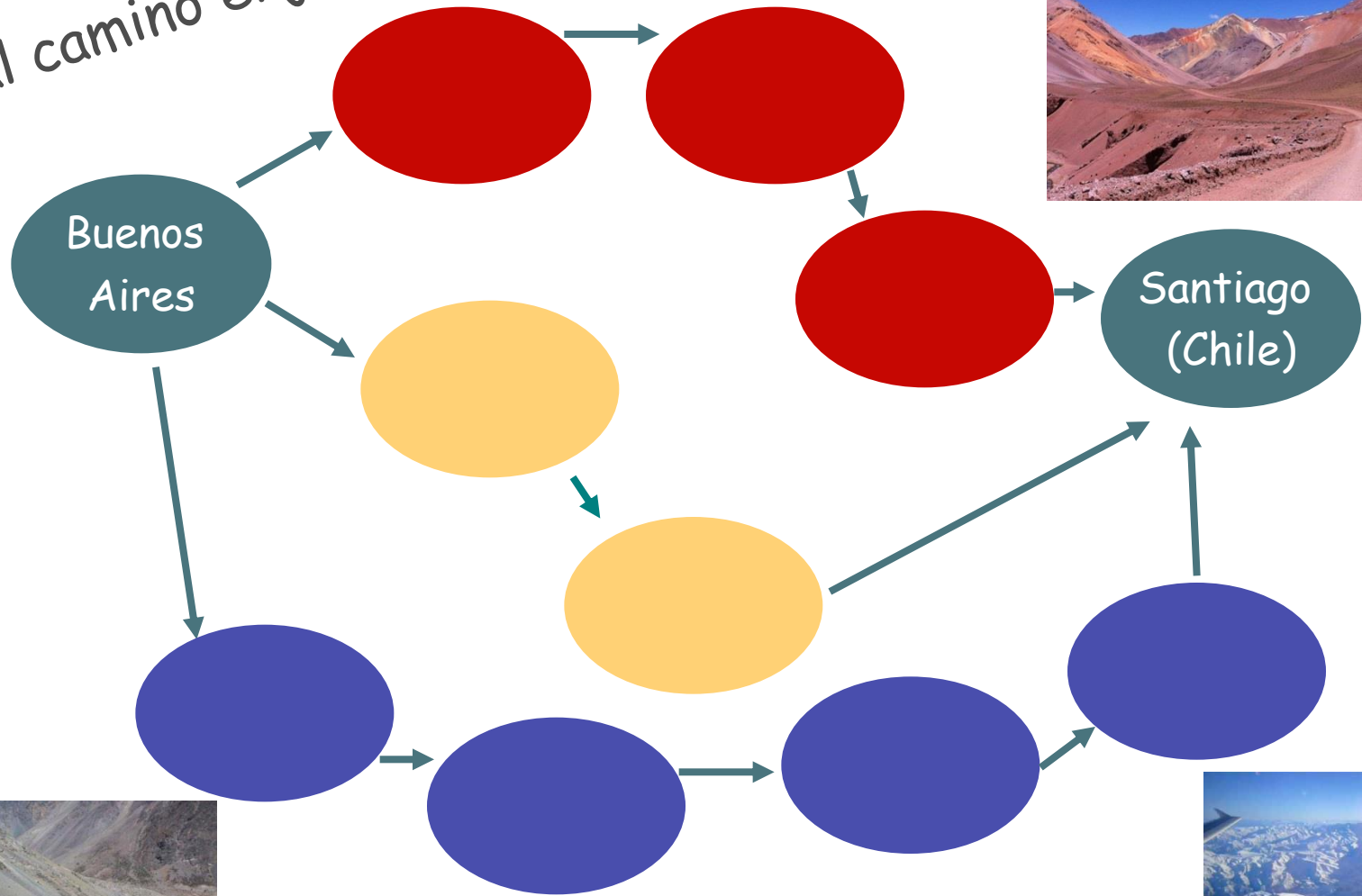
Viajamos de Bs.As a Santiago de Chile



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

Viajamos de Bs.As a Santiago de Chile

¿Cuál camino elijo?



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 (Documentación): 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



Legibilidad (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

Un programa es correcto si cumple con las especificaciones del problema a resolver. Por esta razón, es que la especificación debe ser completa, precisa y no ambigua.

Corrección de Programas

Un programa es **correcto** si cumple con las especificaciones del problema a resolver. Por esta razón, es que la especificación debe ser completa, precisa y no ambigua.

Para medir la corrección de un programa, el programador cuenta con diferentes técnicas:

- ✓ **TESTING**
- ✓ **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 en el problema.

- 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 mediante el cual se pueden identificar y corregir errores

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.

¿Cuál es la mejor solución?

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 aritmético/lógica simple**.

Eficiencia – Análisis Teórico

Reglas Generales para el cálculo del tiempo de ejecución

- Regla 1: Sentencias consecutivas
- Regla 2: For / For anidados
- Regla 3: While / Repeat...Until
- Regla 4: If / else

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

Eficiencia – Análisis Teórico – Regla 1



Supongamos que se desea conocer el valor en grados centígrados de una temperatura medida en grados Fahrenheit. Se tiene la siguiente función que recibe un valor expresado en F y lo devuelve en C

```
Function convertirFaC (tem:real): real;  
begin  
    convertirFaC := (tem-32) * 5 /9;  
end.
```

¿Cuántas operaciones elementales realiza la función?

Recordar que una operación elemental \rightarrow una unidad de tiempo

Eficiencia – Análisis Teórico – Regla 2 (FOR)



Se tiene el siguiente programa que lee las temperaturas registradas durante el mes de mayo de 2019 en una ciudad de la Pcia de Bs. As y calcula la temperatura promedio. ¿Cuántas operaciones elementales realiza el programa?

```
Program temperaturas;
```

```
Var valor, total: real;  
    n, i: integer;
```

```
Begin
```

```
    total:= 0; read (n);
```

```
    for i:= 1 to n do begin
```

```
        read (valor);
```

```
        total:= total + valor;
```

```
    end;
```

```
    prom:= total div n;
```

```
    write('Temperatura Promedio:', prom);
```

```
end;
```

Se debe calcular la cantidad de operaciones elementales que se ejecutan dentro del FOR y multiplicarla por la cantidad de veces que se ejecuta la instrucción FOR.

Además, la instrucción FOR realiza:

- asignación inicial $i:=1$ (1)

- testeo de $i \leq 30$ (31)

- incrementos de $i:= i+1$ ($30*2$) entonces $1+ 31 + 60 = 92$ op

En general : $3*n+2$, siendo n la cantidad de repeticiones

Total- $\rightarrow (2*n) + (3*n+2) + 3 = 5n + 5$ op. elem.

Eficiencia – Análisis Teórico – Regla 2 (FOR)



Se tiene el siguiente programa que lee las temperaturas registradas durante el mes de mayo de 2019 en una ciudad de la Pcia de Bs. As y calcula la temperatura promedio. ¿Cuántas operaciones elementales realiza el programa?

```
Program temperaturas;
```

```
Var valor, total: real;
```

```
Begin
```

```
  total:= 0;
```

```
  for i:= 1 to 30 do begin
```

```
    read (valor);
```

```
    total:= total + valor;
```

```
  end;
```

```
  prom:= total div 30;
```

```
  write('Temperatura Promedio:', prom);
```

```
end;
```

Se debe calcular la cantidad de operaciones elementales que se ejecutan dentro del FOR y multiplicarla por la cantidad de veces que se ejecuta la instrucción FOR.

Además, la instrucción FOR realiza:

- asignación inicial $i:=1$ (1)

- testeo de $i \leq 30$ (31)

- incrementos de $i:= i+1$ (30×2) entonces $1 + 31 + 60 = 92$ op

En general : $3 \times n + 2$, siendo n la cantidad de repeticiones

Total -> $(2 \times 30) + (3 \times 30 + 2) + 3 = 155$ op. elem.

Cálculo de eficiencia (Memoria y Tiempo)



Supongamos ahora que se cuenta con la función Contar que recibe un vector de temperaturas y devuelve la cantidad de veces que aparece la temperatura con valor 10 en el vector. ¿Cuánta memoria y cuántas unidades de tiempo emplea el módulo?

```
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) and (i mod 2 = 0)  
        then can10 := can10 + 1;  
    contar := can10;  
end.
```

Memoria???

Analicemos a continuación
el tiempo de ejecución...

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 and I < 5) then {3}
```

```
            can10 := can10 + 1 {4}
```

```
        else
```

```
            can10:= i * I + 4; {5}
```

```
    contar := can10; {6}
```

```
end.
```

- La línea {1} -> 1 unidad de tiempo
- La línea {2} -> $3n + 2 = 92$ unidades de tiempo
- La línea {3} evalúa dos condiciones -> 3 unidad de tiempo
- La línea {4} -> 2 unidades de tiempo y la línea {5} consume 3 \rightarrow 3 unidades.
- Por la tanto, dentro del FOR se cuentan 3 unidades $\rightarrow 6 * 30$
- La línea {5} -> 1 unidad

Total de operaciones = $2 + 180 + 92$ (como máximo!!!) ¿Por qué?

Cantidad de unidades de tiempo = 274 (como máximo!!!)

Eficiencia – Regla 2 (FOR ANIDADOS)



Aplicando la Regla del FOR, analicemos ahora el tiempo de ejecución del siguiente programa:

```
Program FA;  
Var  
  valor,i,j,suma :integer;  
Begin  
  suma:=0; {1}  
  for j:= 1 to 300 do {2}  
    for i:= 1 to 200 do {3}  
      suma:= suma + I; {4}  
    end  
  end  
End.
```

Cantidad de operaciones (unidades de tiempo)

- La línea {1} -> 1
- La línea {2} -> $3n + 2 = 3 \cdot 300 + 2 = 902$
- La línea {3} -> $3n + 2 = 3 \cdot 200 + 2 = 602$
- La línea {4} -> 2

$$(((2 * 200 + 602) * 300) + 902) + 1 = 301.503 \text{ ut}$$

$$\{4\} * 200 + \{3\}$$

$$\{4\} * 200 + \{3\} * 300$$

$$\{4\} * 200 + \{3\} * 300 + \{2\}$$

$$\{4\} * 200 + \{3\} * 300 + \{2\} + \{1\}$$

Eficiencia – Regla 3 (WHILE/REPEAT...UNTIL)



Supongamos que el siguiente programa calcula la nota promedio de un alumno de Informática a partir de las notas obtenidas en sus exámenes finales. ¿Cuál es el tiempo de ejecución de la solución propuesta?

Program W;

Var

nota,i,suma, prom :integer;

Begin

suma:=0; {1}

total:= 0; {2}

read (nota);

while (nota<>-1) **do begin** {3}

total:= total+1; {4}

suma_N:= suma_N + nota; {5}

read (nota);

end;

prom:= suma_N / total; {6}

writeln (`Promedio: ', prom)

End.

Se debe calcular la cantidad de operaciones elementales que se ejecutan dentro del WHILE y multiplicarla por la cantidad de veces que se ejecuta el WHILE. Como no se conoce esa cantidad se considera el PEOR CASO. Por ejemplo, se supone una cantidad de notas n...

Cantidad de operaciones (unidades de tiempo)

- La línea {1} -> 1
- La línea {2} -> 1
- La línea {3} -> n + 1
- La línea {4} -> 2
- La línea {5} -> 2
- La línea {6} -> 2

$$((4 * n) + n+1) + 4 = (5n + 5) \text{ ut}$$

¿Qué significa que las unidades de tiempo queden expresadas en términos de n?

Eficiencia – Regla 4 (IF THEN/ELSE)



Calcular la cantidad de operaciones elementales 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;  
end;  
End.
```

En el caso de una sentencia IF en su forma completa (then/else), debe calcularse la cantidad de operaciones que se realizan en cada parte y se debe elegir aquella que consuma mas tiempo → mayor cantidad de operaciones (el PEOR CASO).

$$((2 * 3000) + 9002) + 1 = 15.003 \text{ ut}$$

$$(((2 * 200 + 602) * 300) + 902) + 1 = 301.503 \text{ ut}$$

Total de operaciones = 1 + 301.503 = 301.504 (como máximo!!!) ¿Por qué?