Resolución de Problemas

Algoritmos

Grupo ISCyP[1]

Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

24 de septiembre de 2018

[1] Erratas y sugerencias a ldaniel@um.es. Partes de la imagen está descargada de internet y si tuviera derechos de autor/distribución conocidos, por favor, avisen.

Índice de Contenidos

Desarrollo

Lógica de los ordenadores

- La lógica tradicional ha considerado que la humanidad del razonador estaba implícita.[1]
- Los ordenadores han liberado de una restricción técnica en la complejidad de las reglas de un sistema lógico.
- Nuestra necesidad de unas reglas orientadas al modelo humano de razonamiento han sido oscurecidas por los ordenadores para los que tener reglas es mucho más sencillo. Un humano es incapaz de razonar con estas reglas debido a los errores "humanos".

[1] Theoretical Introduction to Programming. Bruce Mills.

Abstracción

- **Abstracción** es modularidad y re-usabilidad en un mismo paquete.
- Cuando la misma abstracción es empleada para distintos casos, podemos ahorrar tiempo y esfuerzo aplicando el mismo razonamiento en ambos casos.
- **PERO** una abstracción debería ser **limpia, clara y práctica**. Una buena teoría es una teoría que ayuda a clarificar el código, no a oscurecerlo.
- Una abstracción incorrecta, demasiado técnica o demasiado formal convertirá nuestro código en símbolos sin sentido. *"El código escrito por un humano no es nunca escrito verdaderamente para una computadora."* Eso es una mala excusa de un mal programador.

Abstracción

- Una abstracción debe ser aprendida con un claro entendimiento del entorno/contexto necesario para su aplicación.

> Euclides parte de que podemos dibujar una línea o un círculo. Mientras esto se mantenga, la geometría Euclídea aplica. Una vez que esto no se cumpla, su uso no está justificado. Pero eso no hace que no sea una teoría robusta cuando se cumplen sus precondiciones.

- Decir que nos encontramos en un espacio Euclídeo tiene ciertas implicaciones y ciertas operaciones que podemos **re-utilizar**.

Modelos

> - La ciencia de la computación es sobre computadoras,
> tanto como la Ciencia Cosmológica es sobre telescopios.
> *Edsger Dijkstra*

- Los modelos reflejan la naturaleza de la mente humana, no la naturaleza del cosmos, como mucho, reflejan cómo la mente humana puede interactuar con ese cosmos. La mayor restricción en estos modelos es la mente humana.
- Con la ciencia de la computación ocurre exactamente lo mismo. La teoría de la computación y los lenguajes de programación han sido diseñados para los humanos. (Aunque en muchas ocasiones reflejan el sistema Von Neumann más de lo que nos gustaría aceptar.)
- Una criatura que fuese capaz de entender completamente nuestras computadoras, no necesitaría ningún lenguaje de programación de alto nivel para operarlas.

Programación y literatura

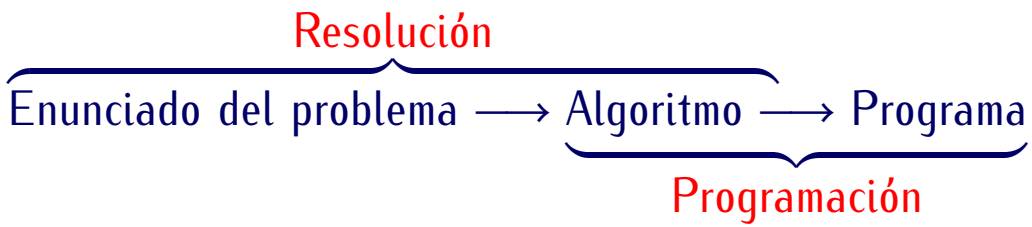
> ... Cuando escribas un programa, piensa en el principalmente como un trabajo literario.
> *Donald Knuth*

- El código debe ser escrito para que sea claro por si mismo, pero también con buenos comentarios.
- Otra persona puede tener que entenderlo y en muchas ocasiones esa persona **puedes ser tú meses más tarde** cuando ya no te acuerdes de los detalles.
- Describe lo que es obvio.
- La parte más extensa de la vida de un programa es el **mantenimiento**. Corrección de errores, actualizaciones,...
- Aunque existen errores en los trabajos matemáticos, la densidad es mucho menor que en sus programas contemporáneos. La urgencia del software es un hecho pero la realidad es que *los trabajos matemáticos están hechos para ser comprendidos* al contrario que el software.

Solución de problemas con programas

Las fases del proceso de resolución de problemas con ordenadores son:

- **Fase de resolución del problema**
  Fase cuyo objetivo es encontrar un **algoritmo**.
- **Fase de implementación/programación**
  Fase en la que dado un **algoritmo** se programa éste en el ordenador.



- **Algoritmo**. Solución constructiva de un problema.

En estas transparencias nos centramos en la primera fase.

Fase de resolución del problema

1. ***Enunciado.*** Definición clara del problema.
2. ***Análisis.*** **¿qué hay que hacer? ¿qué datos se necesitan para resolver el problema? ¿qué información debe proporcionar su resolución?**
3. ***Diseño.*** **¿cómo lo hacemos?** Construir un **algoritmo**.

   En ocasiones la distinción entre análisis y diseño no está clara.

   - *Es posible que en el análisis los datos de entrada y/o salida te vengan dados por el proceso de resolución del problema y tengas poco que diseñar (el proceso viene dado).*
   - *En otras ocasiones el enunciado es tan ambiguo que es necesario empezar por la fase de diseño y buscar qué procesos dan una solución. Entonces, en función del proceso elegido se determinan las entradas y salidas necesarias, que incluso especifican al propio enunciado.*

4. ***Verificación.*** Comprobar que realiza las tareas para las que ha sido diseñado. Produce el resultado correcto y esperado.

Ejercicio 1

Determinar el área de un triángulo. (wikipedia)

Desarrollo

Algoritmos

En general, no existe ningún consenso definitivo en cuanto a la definición formal de algoritmo. Según la wikipedia.

Definición (Algoritmo)

Conjunto de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad.

Definición (Algoritmia)

Los algoritmos son el objeto de estudio de la algoritmia.

Matemáticas y algoritmos

Muchas demostraciones matemáticas son constructivas siguiendo una serie de pasos hasta llegar a la solución. Un algoritmo matemático es una demostración formal.
P.e. Algoritmo de la división Euclidea. Dados dos números naturales D y d, existen dos números únicos c y r tales que $D = d \times c + r$ y $r < c$.

Desarrollo

Las 4 reglas básicas para expresar un algoritmo

1. Usar el **infinitivo** para las acciones: calcular, dividir, andar, mostrar, ...
   Ej: **Sumar** dos números. **Mostrar** un mensaje.

2. Usar los condicionales **si-entonces**.
   Ej: **Si** el número no es nulo, **entonces** calcular el inverso.

3. Usar la expresión **mientras-hacer** o **hacer *n*-veces** para expresar tareas repetitivas.
   Ej: **Mientras** que el número no sea nulo, **hacer** decrementar en una unidad dicho número. **Hacer 3-veces** avanzar un paso.

4. Usar **variables** para guardar/recuperar valores.
   Ej: **Guardar** en x el valor 10 (**hacer** $x = 10$). **Calcular** $x + 20$ (recuperar el valor de x para sumarle 20 unidades).

También existen representaciones gráficas: **diagramas de flujo**.

Ejemplos

Ejemplo. Algoritmo para sumar dos números

1. **Colocar** los números: el primero encima del segundo, de tal manera que las unidades, decenas, centenas, etc., de los números queden alineadas. **Trazar** una línea debajo del segundo número.
2. **Empezar** por la columna más a la derecha.
3. **Sumar** los dígitos de dicha columna.
4. **Si** la suma es mayor a 9, **entonces anotar** un 1 encima de la siguiente columna a la izquierda y anotar debajo de la línea las unidades de la suma. **Si** no es mayor **anotar** la suma debajo de la línea. En ambos casos se escribe debajo de la columna correspondiente.
5. **Si** hay más columnas a la izquierda, **pasar** a la siguiente columna a la izquierda y volver a 3.
6. **Retornar** el número de debajo de la línea como la solución.

Ejemplo. Algoritmo para hacer amigos

The Big Bang Theory, Sheldon y su algoritmo para hacer amigos

Desarrollo

Algunas Técnicas de Diseño - I

- Top-down (Descendente)
  - Basado en "Divide y vencerás".
  - Descomposición del problema en partes más simples hasta llegar a subproblemas de solución inmediata (o conocida).
  - La solución es una composición de las soluciones de los subproblemas.
  - Se puede representar mediante un grafo Y/O.
  - Los subproblemas se pueden programar de forma independiente.
  - Las modificaciones en los módulos son más fáciles.
  - Se puede verificar la solución más fácilmente.

Torres de Hanoi.	Integral de una suma
	$\int_a^b f(x) + g(x)\, dx = \int_a^b f(x)\, dx + \int_a^b g(x)\, dx$

Ejercicio 2

Calcula el área de todas las "caras" de un cilindro.

Algunas Técnicas de Diseño - II

- Stepwise Refinement
  - Primero se describen pocos pasos y de manera incompleta.
  - Se amplían las descripciones de manera más detallada con pasos más específicos.
  - Se tienen diferentes niveles de refinamiento antes de obtener un algoritmo claro, preciso y completo.
  - **No confundir con diseño descendente.**

Vamos de compras

Algunas Técnicas de Diseño - III

- Backtracking
  - Donde se requiere una búsqueda exhaustiva de todas las posibilidades.
  - Puede visualizarse como un estructura de árbol.

Sal del laberinto

  - Ramificación y poda: cuando se quiere optimización.

Algunas Técnicas de Diseño - IV

- Algoritmo Voraz
  - Para búsqueda de soluciones óptimas en problemas combinatorios.
  - En cada paso se busca la mejor solución local.
  - Presenta el problema de posible aparición de mínimos locales.

Rellena la mochila