Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2013

Departamento de Computación - FCEyN - UBA

Especificación - clase 1

Introducción a la especificación de problemas

Cursada

- clases teóricas
  - Diego Garbervetsky y Javier Marenco
- clases prácticas
  - Pablo Turjanski, Martín Urtasun, Gabriela Di Piazza y Silvina Dengra
- sitio web de la materia: www.dc.uba.ar/people/materias/algo1
- régimen de aprobación
  - parciales
    - 3 parciales
    - 3 recuperatorios (al final de la cursada)
  - trabajos prácticos
    - 3 entregas
    - 3 recuperatorios (cada uno a continuación)
    - grupos de 4 alumnos
  - examen final
    (si lo dan en Julio cuenta 70 % la nota de la cursada)

algo1-alu@dc.uba.ar

algo1-doc@dc.uba.ar

¿Qué es una computadora?

¿Qué es una computadora?

Dispositivo para manipulación simbólica que ejecuta cualquier algoritmo

Matemáticamente, es una función recursiva parcial universal.

¿Qué es un algoritmo?

Definición de Algoritmo

Un algoritmo es una sucesión de pasos primitivos que permiten resolver un problema.

¿Con qué pasos primitivos contamos?

Ejemplo: Problema sumar dos números naturales.

Algoritmo de suma escolar, por columnas
Algoritmo de sucesor n veces.

Pasos primitivos

Problema: sumar dos números naturales

- Algoritmo suma escolar: sumo las unidades del primero a las del segundo, después las decenas y así ("llevándome uno de acarreao" cuando hace falta)
- Algoritmo sucesor: voy sumando uno al primero y restando uno al segundo, hasta que llegue a cero
- Algoritmo moderno: escribo el primero en una calculadora, aprieto $+$, escribo el segundo, aprieto $=$

Los tres algoritmos para distintas operaciones primitivas:

- sumar dos dígitos y acarreo, y concatenar resultados
- sumar uno, restar uno y comparar por 0
- apretar una tecla de una calculadora

Definición de programa. Propiedades

Definición: Un programa es la descripción de un algoritmo en un lenguaje de programación.

¿El programa siempre termina?

¿Es correcto? Es decir, ¿resuelve el problema?

modular, goloso, deterministico, paralelo, probabilistico, polinomial, cálculo científico

1936

Alonzo Church, define el cáculo lamdba

- "An Unsolvable Problem of Elementary Number Theory", American Journal of Mathematics, 1936
- "A Note on the Entscheidungsproblem", Journal of Symbolic Logic, 1936

Alan Turing, define máquina universal, sus limitaciones.

- "On computable numbers with an application to the Entscheidungsproblem", Journal of The Londom Mathematical Society, 1936.

Algoritmos y Estructuras de Datos I

Objetivos

El principal: **aprender a programar**

- Especificar problemas
  - describirlos en lenguaje no ambiguo, formal
- Escribir programas sencillos
  - tratamiento de secuencias
- Razonar acerca de estos programas
  - visión abstracta del computo
  - manejo simbólico y herramientas para demostrar
  - Correctitud de un programa respecto de su especificación

Algoritmos y Estructuras de Datos I

Contenidos

- Especificación
  - describir problemas en un lenguaje formal (preciso, claro, abstracto)
- Programación funcional (Haskell)
  - parecido al lenguaje matemático
  - escribir de manera simple algoritmos y estructuras de datos
  - correctitud por inducción
- Programación imperativa (C++)
  - paradigma más difundido
  - más eficiente
  - se necesita para seguir la carrera
  - demostraciones de correctitud

Algoritmos sobre secuencias

Búsqueda lineal y búsqueda binaria.

Un algoritmo de ordenamiento (upsort).

Intercalación ordenada (merge).

Noción de complejidad tiempo.

Problemas y soluciones

Si vamos a escribir un programa, es porque hay un problema a resolver.

- necesitamos describirlo de manera precisa
- no siempre es claro que haya solución

Ejemplos de problemas:

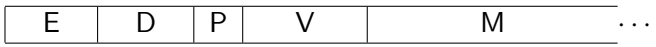
- Calcular la edad de una persona
  Tal vez podamos crear un programa, pero necesitamos decir qué datos de entrada vamos a tener.
- Calcular el n-ésimo número primo
  Se pueden usar programas
- Calcular rápido el n-esimo número primo
  Se pueden usar programas, pero...¿rápido ? . . .
- Aprobar Algoritmos I
  Seguro que requiere programas

Especificación, algoritmo, programa

1. especificación = descripción del problema
   - ¿qué problema tenemos?
   - en lenguaje formal
   - es un contrato que da las propiedades de los datos de entrada y las propiedades de la solución
   - Edsger Dijkstra, Richard Hoare (años 70)
2. algoritmo = descripción de la solución escrita para humanos
   - ¿cómo resolvemos el problema?
3. programa = descripción de la solución para la computadora
   - también, ¿cómo resolvemos el problema?
   - usando un lenguaje de programación

Etapas en el desarrollo de programas

1. especificación: definición precisa del problema
2. diseño: elegir una solución y dividir el problema en partes
3. programación: escribir algoritmos e implementarlos en algún lenguaje
4. validación: ver si el programa cumple con lo especificado
5. mantenimiento: corregir errores y adaptarlo a nuevos requerimientos



1. Especificación

El planteo inicial del problema puede ser vago y ambiguo.
Al especificar damos una descripción clara y precisa
en lenguaje formal, por ejemplo, el lenguaje de la lógica matemática.
Ejemplo, el problema calcular de edad de una persona, es vago porque:

- ¿Cuáles son los datos? fecha de nacimiento, ADN
- ¿Qué forma van a tener? días, años enteros, fracción de años
- ¿Cómo recibo los datos? manualmente, archivo en disco, sensor
- ¿Cómo devuelvo el resultado? pantalla, papel, voz alta, email

Una especificación formaliza algunas, como:

- *Necesito una función que, dadas dos fechas en formato dd/mm/aaaa, de la cantidad de días que hay entre ellas.*

Todavía faltan los requerimientos no funcionales

- Forma de entrada de parámetros y de salida de resultados, tipo de computadora, tiempo con el que se cuenta para programarlo, etc. No son parte de la especificación funcional y quedan para otras materias

2. Diseño

Etapa en la que se responde

- ¿Varios programas o uno muy complejo?
  - ¿cómo dividirlo en partes, qué porción del problema resuelve cada una?
  - ¿distintas partes en distintas máquinas?
  - estudio de las partes que lo componen
    - (mini) especificación de cada parte
    - un programador recibe una sola (o una por vez)
- ¿Programas ya hechos con los que interactuar?
- lo van a ver en Algoritmos y Estructuras de Datos II y en Ingeniería del Software I.

3. Programación - Algoritmos

Escribir un algoritmo para dar solución al problema:

- Asumimos que están definidos los pasos primitivos
- Dar la sucesión de pasos para llegar al resultado buscado de manera efectiva

Ejemplo. Dada la especificación:

> *Necesito una función que, dadas dos fechas a y b en formato dd/mm/aaaa, me devuelva la cantidad de días que hay entre ellas.*

Un algoritmo es:

1. restar el año de *b* al año de *a*
2. multiplicar el resultado por 365
3. sumarle la cantidad de días desde el 1° de enero del año de *b* hasta el día *b*
4. restarle la cantidad de días desde el 1° de enero del año de *a* hasta el día *a*
5. sumarle la cantidad de 29 de febrero que hubo en el período
6. devolver ese resultado, acompañado de la palabra "días"

¿Son todos primitivos?

- si no, hay que dar algoritmos para realizarlos.
  Por ejemplo para 3, 4 y 5.

3. Programación - Programas

- traducir el algoritmo (escrito o idea) para que una computadora lo entienda
- lenguaje de programación
  - vamos a empezar usando uno: Haskell
  - después, C++
- hay muchos otros
  - pueden ser más adecuados para ciertas tareas
  - depende del algoritmo, de la interfaz, tiempo de ejecución, tipo de máquina, interoperabilidad, entrenamiento de los programadores, licencias, etc.

Representación de los datos

- ya vimos que era importante en el ejemplo de calcular la edad de una persona (días, meses, años, fracciones de año)
- otro ejemplo:
  - contar la cantidad de alumnos de cada sexo
  - ¿qué dato conviene tener?
    - foto
    - foto del documento
    - ADN
    - valor de la propiedad *sexo* para cada alumno
- estructuras de datos
  - necesarias para especificar
  - puede ser que haya que cambiarlas al programar
  - las van a estudiar a fondo en *Algoritmos y Estructuras de Datos II*

4. Validación

Asegurarse de que un programa cumple con la especificación

- testing
  - probar el programa con muchos datos y ver si hace lo que tiene que hacer
  - en general, para estar seguro de que anda bien, uno tendría que probarlo para infinitos datos de entrada
  - si hay un error, con algo de suerte uno puede encontrarlo
  - no es infalible (puede pasar el testing pero haber errores)
  - en *Ingeniería del Software I* van a ver técnicas de testing
- verificación formal
  - demostrar matemáticamente que un programa cumple con una especificación. Se llama çorrectutud de un programa respecto de una especificación". Requiere que el lenguaje de especificación sea formal y admita una teoría de prueba, y una semántica.
  - una demostración cubre infinitos valores de entrada a la vez (abstracción)
  - es infalible (si está demostrado, el programa no tiene errores)
  - en esta materia vamos a estudiar cómo demostrar la correctitud de programas sencillos para problemas sencillos.

5. Mantenimiento

- tiempo después, encontramos errores
  - el programa no cumplía la especificación
  - la especificación no describía correctamente el problema
- o cambian los requerimientos
- puede hacerlo el mismo equipo u otro
- justifica las etapas anteriores
  - si se hicieron bien la especificación, diseño, programación y validación, las modificaciones van a ser más sencillas y menos frecuentes

Especificación de problemas

Es un contrato que define qué se debe a resolver y qué propiedades debe tener una solución:

- define el qué y no el cómo

No usamos problemas para especificar problemas.

Después de programar la especificación de problemas sirve para:

- testing
- verificación formal de correctitud
- derivación formal
  (construir un programa a partir de la especificación)

Parámetros de un problema

- ejemplo de problema: arreglar una cafetera
- para solucionarlo, necesitamos más datos
  - ¿qué clase de cafetera es?
  - ¿qué defecto tiene?
  - ¿cuánto presupuesto tenemos?
- se llaman parámetros del problema
- los valores de los parámetros se llaman argumentos
- cada combinación de valores de los parámetros es una instancia del problema
  - una instancia: "arreglar una cafetera de filtro cuya jarra pierde agua, gastando a lo sumo $30"

Problemas funcionales

- no podemos especificar formalmente cualquier problema
- simplificación (para esta materia)
  - problemas que puedan solucionarse con una función
    - parámetros de entrada
    - un resultado para cada combinación de valores de entrada

Tipos de datos

Cada parámetro tiene un tipo de datos

- conjunto de valores para los que hay ciertas operaciones definidas. Cada tipo de datos lleva un nombre.

Por ejemplo:

- parámetros de tipo *fecha*
  - valores: ternas de números enteros
  - operaciones: comparación, obtener el año,...
- parámetros de tipo *dinero*
  - valores: números reales con dos decimales
  - operaciones: suma, resta,...

Encabezado de un problema

problema $nombre(parámetros) = nombreRes : tipoRes$

- *nombre*: nombre que le damos al problema
  - será resuelto por una función con ese mismo nombre
- *nombreRes*: nombre que le damos al resultado
- *tipoRes*: tipo de datos del resultado
- *parámetros*: lista que da el tipo y el nombre de cada uno

Ejemplo

- problema $resta(minuendo, sustraendo : \text{Int}) = res : \text{Int}$
- la función se llama resta
- da un resultado que vamos a llamar res
  - y es de tipo Int (los enteros)
- depende de dos parámetros: minuendo y sustraendo
  - también son de tipo Int

Contratos

- la función que solucione el problema va a ser llamada o invocada por un usuario
  - puede ser el programador de otra función
- especificación = contrato entre el programador de una función que resuelva el problema y el usuario de esa función

Por ejemplo:

- problema: calcular la raíz cuadrada de un real
- solución (función)
  - va a tener un parámetro real
  - va a calcular un resultado real
- para hacer el cálculo, debe recibir un número no negativo
  - compromiso para el usuario: no puede proveer números negativos
  - derecho para el programador de la función: puede suponer que el argumento recibido no es negativo
- el resultado va a ser la raíz del número
  - compromiso del programador: debe calcular la raíz, siempre y cuando haya recibido un número no negativo
  - derecho del usuario: puede suponer que el resultado va a ser correcto

Partes de una especificación

Tiene 3 partes

1. encabezado (ya lo vimos)
2. precondición
   - condición sobre los argumentos
   - el programador da por cierta
   - lo que requiere la función para hacer su tarea
   - por ejemplo: "el valor de entrada es un real no negativo"
3. poscondición
   - condición sobre el resultado
   - debe ser cumplida por el programador, siempre y cuando el usuario haya cumplido la precondición
   - lo que la función asegura que se va a cumplir después de llamarla (si se cumplía la precondición)
   - por ejemplo: "la salida es la raíz del valor de entrada"

El contrato dice:

El programador va a hacer un programa P tal que si el usuario suministra datos que hacen verdadera la precondición, entonces P va a terminar en una cantidad finita de pasos y va a devolver un valor que hace verdadera la poscondición.

- si el usuario no cumple la precondición y P se cuelga o no cumple la poscondición...
  - ¿el usuario tiene derecho a quejarse? No.
  - ¿se viola el contrato? No. El contrato prevé este caso y dice que P solo debe funcionar en caso de que el usuario cumpla con la precondición
- si el usuario cumple la precondición y P se cuelga o no cumple la poscondición...
  - ¿el usuario tiene derecho a quejarse? Sí.
  - ¿se viola el contrato? Sí. Es el único caso en que se viola el contrato. El programador no hizo lo que se había comprometido a hacer.

Lenguaje naturales y lenguajes formales

- lenguajes naturales
  - idiomas (castellano)
  - mucho poder expresivo (modos verbales –potencial, imperativo–, tiempos verbales –pasado, presente, futuro—, metáforas, etc.)
  - con un plus (conocimiento del contexto, experiencias compartidas ambigüedad e imprecisión)
  - debemos evitarlos al especificar
- lenguajes formales
  - limitan lo que se puede expresar
  - todas las suposiciones quedan explícitas
  - relación directa entre lo escrito (sintaxis) y su significado (semántica)
  - pueden tratarse formalmente
    - símbolos manipulables directamente
    - seguridad de que las manipulaciones son válidas también para el significado
  - ejemplo: aritmética
    - lenguaje formal para los números y sus operaciones
    - resolvemos problemas simbólicamente sin pensar en los significados numéricos y llegamos a resultados correctos
  - en *Teoría de Lenguajes* y *Lógica y Computabilidad* van a estudiarlos en profundidad

Especificación formal

- encabezado
  (nombre del problema, argumentos de entrada, salida)
  - la precondición (lo que la función requiere)
  - la poscondición (lo que asegura)
- usamos un lenguaje formal para describir la precondición y la postcondición
- ejemplo de problema: calcular la raíz cuadrada de un número
  - lo llamamos *rcuad*
  - Float representa el conjunto $\mathbb{R}$ que la computadora puede manipular.

  problema $rcuad(x : \text{Float}) = result : \text{Float}$ {
  requiere $x \geq 0$;
  asegura $result * result == x$;
  }

Ejemplos

problema $suma(x : \text{Int}, y : \text{Int}) = result : \text{Int}$ {
asegura $result == x + y$;
}

problema $f^{-1}(y : \text{Int}) = result : \text{Int}$ {
asegura $f(res) == y$;
aux $f(z) = ...$
}

Sobrespecificación

Es cuando la postcondición impone más restricciones de las necesarias. Limita excesivamente los posibles algoritmos que resuelven el problema.
Ejemplo:
problema $distinto(x : \text{Int}) = res : \text{Int}$ {
asegura $res > x$;
asegura $primo(res)$;
}
en vez de
asegura $not(res == x)$;