Especificación de algoritmos

Facultad de Informática - UCM

21 de octubre de 2020

Indice

1. Bibliografia..3
2. Objetivos...4
3. Especificación. Propiedades. Tipos.............................5
4. Predicados..12
5. Ejemplos de especificaciones...................................23

Bibliografía Recomendada

- Algoritmos correctos y eficientes: Diseño razonado ilustrado con ejercicios. *Matí-Oliet, N.; Segura Diaz, C. M., Verdejo Lopez, A.*. Ibergarceta Publicaciones, 2012.
- Especificación, Derivación y Análisis de Algoritmos: ejercicios resueltos. *Narciso Martí Oliet, Clara María Segura Díaz y Jose Alberto Verdejo López*. Colección Prentice Práctica, Pearson Prentice-Hall, 2006.
  Edición anterior del mismo libro.

Capítulo 1 de ambos libros.

- Ejercicios resueltos: todos menos el 1.5, 1.6, 1.7
- Ejercicios propuestos: todos menos el 1.2.

Objetivos

- Diferencia entre especificar e implementar
- Diferencia entre una especificación en lenguaje natural y una especificación formal
- Partes de una especificación formal
- Aprender predicados sencillos
- Usar los predicados para construir especificaciones

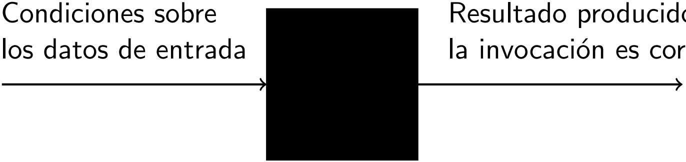
Especificación. Propiedades. Tipos.

- Especificar un algoritmo.

qué hace el algoritmo.

Precondición
Condiciones sobre
los datos de entrada

Postcondición
Resultado producido si
la invocación es correcta.



- Implementar,

cómo *se consigue la funcionalidad pretendida*.

Especificación. Propiedades. Tipos.

Una especificación para indicar el problema que queremos resolver con varias implementaciones que resuelven el problema.

Especificación 1: Dado un vector de números enteros A ordenado de menor a mayor y un número entero x, devolver un valor booleano b que indique si alguno de los elementos del vector $A[0], \ldots, A[n-1]$ es igual al valor dado x.

Implementación: Algoritmo de búsqueda secuencial o algoritmo de búsqueda binaria.

Especificación 2: Ordenar de menor a mayor un vector de números enteros A.

Implementación: Ordenación rápida (quicksort), ordenación por mezclas (mergesort), ordenación por inserción (insertion sort)...

Usos de una especificación:

- Los **usuarios** del algoritmo:
  la especificación debe contener la información necesaria para utilizar el algoritmo.
- El **implementador** del algoritmo:
  - la especificación debe definir los requisitos que cualquier implementación válida debe satisfacer.
  - Ha de dejar suficiente **libertad** para que se elija la implementación que se estime más adecuada con los recursos disponibles.

Especificación. Propiedades. Tipos.

Propiedades de una buena especificación:

Precisión Ha de responder a cualquier pregunta sobre el uso del algoritmo sin tener que acudir a la implementación del mismo. El lenguaje natural no permite la precisión requerida si no es sacrificando la brevedad.

Brevedad Ha de ser significativamente más breve que el código que especifica.

Claridad Ha de transmitir la intuición de lo que se pretende. A veces el lenguaje formal ha de ser complementado con explicaciones informales.

Los lenguajes formales ofrecen a la vez precisión y brevedad. A veces deben ser complementado con explicaciones informales para obtener claridad.

Ventajas de una **especificación formal** frente a una informal:

- Eliminan ambigüedades que el usuario y el implementador podrían resolver de formas distintas, dando lugar a errores de uso o de implementación que aparecerían en ejecución.
- Permiten realizar una verificación formal del algoritmo. Este tema se tratará en el capítulo **3** y consiste en razonar sobre la corrección del algoritmo mediante el uso de la **lógica**.
- Permite generar automáticamente un conjunto de casos de prueba que permitirán probar la corrección de la implementación. La especificación formal se usa para **predecir** y **comprobar** el resultado esperado.

Especificación. Propiedades. Tipos.

Especificación informal: dado un vector a de componentes enteras, devolver un valor booleano b que indique si existe algún elemento cuyo valor es igual a la suma de todos los elementos que le preceden en el vector.

Ambigüedades:

- No quedan claras todas las obligaciones del usuario: (1) ¿serían admisibles llamadas con vectores vacíos?; (2) ¿y con un elemento?
- En caso afirmativo, ¿cuál será el valor devuelto por la función?
- Tampoco están claras las obligaciones del implementador. Por ejemplo, si el vector tiene mas de un elemento y $a[0] = 0$ la función, ¿ha de devolver **true** o **false**?

Tratar de explicar en lenguaje natural todas estas situaciones llevaría a una especificación más extensa que el propio código de la función.

- Utilizamos una técnica llamada *especificación pre/post* basada en lógica de predicados.
- La precondición y la postcondición se expresan mediante predicados lógicos.

$Q \equiv \{0 \leq \mathbf{a.size()}\}$
$essuma\ (\mathbf{vector{<}int{>}\ a})\ dev\ \mathbf{bool\ sol}$
$R \equiv \{\mathbf{sol}\ \equiv\ \exists \mathbf{w} : \mathbf{0} \leq \mathbf{w} < \mathbf{n} : \mathbf{a}[\mathbf{w}] = (\sum \mathbf{u} : \mathbf{0} \leq \mathbf{u} < \mathbf{w} : \mathbf{a}[\mathbf{u}])\}$

No es ambigua:

1. sí son correctas las llamadas con vectores vacíos y en ese caso ha de devolver **false**;
2. las llamadas con vectores con más de un elemento y $a[0] = 0$ han de devolver $b = \mathbf{true}$.

Predicados

Sintaxis de la lógica de primer orden que utilizamos:

Un predicado es una expresión e de tipo **bool**.

Si P y Q son predicados, también lo son:

1. $\neg P$, (no P).
2. $P \wedge Q$, (P y Q).
3. $P \vee Q$, (P o Q).
4. $P \rightarrow Q$, (P entonces Q).
5. Cuantificación universal: $(\forall w : Q(w) : P(w))$, (para todo valor perteneciente al rango Q se cumple P).
6. Cuantificación existencial: $(\exists w : Q(w) : P(w))$, (existe un valor perteneciente al rango Q para el que se cumple P) .

Las variables que se utilizan en los cuantificadores deben ser diferentes de todas las variables del programa.

Predicados. Cuestiones

Dado un vector a de números enteros, y variables de tipo entero: n, i, y x. Escribe predicados para expresar:

- El valor de la variable n es mayor que cero.
  positivo(n) $\equiv$
- El valor de la variable i es mayor o igual que cero y menor que el valor de la variable n.
  rango(i,n) $\equiv$
- Todas las componentes de un vector a son positivas.
  vectorPositivo(a) $\equiv$
- Existe una componente en el vector a cuyo valor es igual que el de la variable x.
  buscar(a,x) $\equiv$
- El valor de la variable x es impar.
  impar(x) $\equiv$

Variables libres y ligadas.

- Es **muy importante** distinguir los dos tipos de variables que pueden aparecer en un predicado:
  - Variables ligadas. Son las que están afectadas por un cuantificador, es decir se encuentran dentro de su ámbito.
  - Variables libres. Son las variables que no se encuentran afectadas por ningún cuantificador.
- Por ejemplo en el predicado $p(n, a) \equiv$

$$\forall w : 0 \leq w < n : \mathit{impar}(w) \rightarrow (\exists u : u \geq 0 : a[w] = 2^u),$$

  - las variables **libres** son n y a,
  - las variables **ligadas** son w y u.
- Cuando se anidan cuantificadores las variables ligadas deben tomar nombres diferentes.
- Las variables libres de los predicados que se utilicen en las especificaciones serán variables del programa.

Dado un vector v de números enteros y variables de tipo entero n, k_1 y k_2, indica que expresan los siguientes predicados y las variables libres y ligadas que utilizan.

- $p_1(v) \equiv \exists n \in \mathbb{N} : (\forall k : 0 \leq k < v.size() : v[k] < n)$
- $p_2(v, n) \equiv \forall k : 0 \leq k < v.size() : v[k] < n$
- $p_3(v) \equiv \exists n : 0 \leq n < v.size() : (\forall k : 0 \leq k < v.size() : v[k] < v[n])$
- $p_4(v) \equiv \forall k : 0 \leq k < v.size() - 1 \land v[k] \,\%2 == 0 : v[k+1] \,\%2 == 1$
- $p_5(v) \equiv \forall k : 0 \leq k < v.size() \land v[k] \,\%2 == 0 : (\exists n : k < n < v.size() : v[n] \,\%2 == 1)$
- $p_6(v, k_1, k_2) \equiv \forall k : k_1 \leq k < k_2 : v[k] > 0$

- Otras expresiones cuantificadas que utilizaremos.
- Son expresiones **de tipo entero** en lugar de booleano.

$\sum w : Q(w) : e(w)$	suma de las $e(w)$ tales que $Q(w)$
$\prod w : Q(w) : e(w)$	producto de las $e(w)$ tales que $Q(w)$
$\text{máx}\, w : Q(w) : e(w)$	máximo de las $e(w)$ tales que $Q(w)$
$\text{mín}\, w : Q(w) : e(w)$	mínimo de las $e(w)$ tales que $Q(w)$
$\#\ w : Q(w) : P(w)$	De los valores que cumplen $Q(w)$ cuantos verifican $P(w)$

donde Q y P son predicados y e es una expresión entera:

Predicados. Cuestiones

Dado un vector v de números enteros y una variable de tipo entero n, indica que expresan las siguientes funciones y predicados.

- $suma(v) = \sum k : 0 \leq k < v.size() : v[k]$
- $spar(v) = \sum k : 0 \leq k < v.size() \land k\,\%2 == 0 : v[k]$
- $p_1(v) \equiv \forall k : 0 \leq k < v.size() : (\sum i : 0 \leq i < k : v[i]) < (\sum j : k+1 \leq j < v.size() : v[j])$
- $cuenta(v) = \# k : 0 \leq k < v.size() : (\sum i : 0 \leq i < k : v[i]) < (\sum j : k+1 \leq j < v.size() : v[j])$
- $maximo(v) = \max k : 0 \leq k < v.size() : v[k]$
- $p_2(v, n) \equiv (\max h : 0 \leq h < v.size() : v[h]) < n$
- $p_3(v, n) \equiv (\max h : 0 \leq h < v.size() : v[h]) == v[n]$
- $contmax(v) = \# k : 0 \leq k < v.size() : v[k] == maximo(v)$

Predicados.

- Por definición, el significado de los cuantificadores cuando el rango $Q(w)$ al que se extiende la variable cuantificada es **vacío**, es el siguiente:

$$
\begin{array}{ll}
(\forall w : Q(w) : P(w) \equiv \mathbf{true}) & \\
(\exists w : Q(w) : P(w) \equiv \mathbf{false}) & \\
(\sum w : \mathbf{false} : e(w)) = 0 & \\
(\prod w : \mathbf{false} : e(w)) = 1 & \\
(\max w : \mathbf{false} : e(w)) & \text{indefinido} \\
(\min w : \mathbf{false} : e(w)) & \text{indefinido} \\
(\# \ w : \mathbf{false} : P(w)) = 0 &
\end{array}
$$

Predicados. Mas cuestiones

Escribe un predicado que exprese:

- Los valores de un vector v de números enteros positivos en índices par son pares y los valores en índices impar son impares.
  parImpar(v) $\equiv$
- El valor i-ésimo de un vector w es igual a la suma de las componentes de un vector v desde i hasta el final del vector.
  sumaC(v,w,i) $\equiv$
- El valor de todas las componentes de un vector w es la suma de las componentes de un vector v desde dicha componente hasta el final del vector (w es el vector acumulado de v).
  vectorAcumulados(v,w) $\equiv$
- Todos los valores a la izquierda de una posición p (incluida) en un vector v son menores que todos los valores a la derecha de la posición p.
  particion1(v,p) $\equiv$
- x es el valor mínimo de un vector v.

Predicados. Mas cuestiones

Escribe un predicado que exprese:

- x es la suma todos los valores de un vector v menos el valor mínimo.
  sumaTotal(v,x) $\equiv$
- Todas las componentes de un vector s están en el vector v o en el vector w.
  pertenece(s,v,w) $\equiv$
- La componente i-esima de un vector v aparece una única vez en el vector.
  unico(v,i) $\equiv$
- Un vector v no tiene elementos repetidos.
  repetidos(v) $\equiv$
- Un vector v está ordenado.
  ordenado(v) $\equiv$

Significado

- Los predicados nos sirven para expresar relaciones entre los valores que toman los parámetros y las variables del programa.
- El estado de ejecución de un algoritmo es una asociación de las variables del algoritmo con valores compatibles con su tipo.

Ejemplo Search

```
1 bool Search (std::vector<int> const& a, int x)
2 {
3     int i = 0;
4     while (i < a.size() && a[i] != x)
5          i++;
6     return i < a.size();
7 }
```

LLamada al algoritmo: `Search(a,3)` Siendo $a = \{2, 5, 6, 1, 3\}$
Estado después de ejecutar la linea 3:
$a = \{2, 5, 6, 1, 3\}$; $x = 3$; $i = 0$

Significado

- Los estados cumplen propiedades definidas por predicados
- El estado $a = \{2, 5, 6, 1, 3\}$; $x = 3$; $i = 0$
  cumple la propiedad $a.size() \geq 0 \wedge x \geq 0 \wedge i == 0$
- El estado después de ejecutar la línea 5 tres veces será:
  $a = \{2, 5, 6, 1, 3\}$; $x = 3$; $i = 3$
  cumple la propiedad $a.size() \geq 0 \wedge x \geq 0 \wedge i == 3$
- Es más interesante definir propiedades genéricas. Por ejemplo, Una propiedad que se cumple en todas las vueltas del bucle de la línea 4 (antes de empezar el bucle, mientras se ejecuta el bucle y después de terminar el bucle) es:
  $\forall k : 0 \leq k < i : a[k] \neq x$

Ejemplos de especificaciones

La especificación debe definir la precondición y la postcondición del algoritmo.

- La precondición indica las propiedades que cumplen los parámetros de entrada.
- La postcondición expresa las propiedades del valor de retorno de la función y de los parámetros de salida.
- Se expresa por medio de predicados.

- Especificar un algoritmo que calcule el máximo de un vector:

$$\{a.size() > 0\}$$
$$maximo\ (\mathbf{vector{<}int{>}}\ a) : dev\ \textbf{int}\ m$$
$$\{m == \max\ w : 0 \le w < a.size() : a[w]\}$$

  - La precondición $a.size() > 0$ es necesaria para asegurar que el rango del máximo no es vacío.

- Especificar un algoritmo que calcule un número primo mayor o igual que un cierto valor:

$$\{n > 1\}$$
$$unPrimo\ (\textbf{int}\ n)\ dev\ \textbf{int}\ p$$
$$\{p \geq n \wedge (\forall w : 1 < w < p : p \mod w \neq 0)\}$$

  - ¿Sería correcto devolver $p = 2$?
  - Nótese que la postcondición no determina un único p. El implementador tiene la libertad de devolver cualquier primo mayor o igual que n.

- Devolver el **menor** número primo mayor o igual que un cierto valor:

$$\{n > 1\}$$
$$menorPrimo\ (\textbf{int}\ n)\ dev\ \textbf{int}\ p$$
$$\{p \geq n \wedge primo(p) \wedge (\forall w : w \geq n \wedge primo(w) : p \leq w)\}$$

donde $primo(x) \equiv (\forall w : 1 < w < x : x \mod w \neq 0)$

- Obsérvese que el uso del predicado auxiliar $primo(x)$ hace la especificación a la vez más modular y legible.
- Nótese que un predicado **no** es una implementación. Por tanto no le son aplicables criterios de eficiencia: $primo(x)$ es una propiedad y **no** sugiere que la comprobación haya de hacerse dividiendo por todos los números menores que x.
- La postcondición podría expresarse de un modo más conciso:

$$p = (\min w : w \geq n \wedge primo(w) : w)$$

sin que de nuevo esta especificación sugiera una forma de implementación.

- Especificar una función que calcule la suma de los valores de un vector exceptuando el valor mínimo y devuelva la suma y el número de sumandos.

$$
\begin{array}{l}
\{a.size() > 0\} \\
minimo\ (\textbf{vector<int>}\ a)\ dev\ \{\textbf{int}\ s, \textbf{int}\ n\} \\
\{sumaTotal(a, s) \wedge \\
(n == \#k : 0 \leq k < a.size() \wedge \neg minimo(a, a[k]) : 1)\}
\end{array}
$$

- Especificar un procedimiento que *positiviza* un vector. Ello consiste en reemplazar los valores negativos por ceros.

$$\{a.size() \geq 0 \wedge a = A\}$$
$$\textbf{void } \textit{positivizar } (\textbf{vector<int>} \ \& \ \textbf{a})$$
$$\{\forall w : 0 \leq w < n : (A[w] < 0 \rightarrow a[w] = 0) \wedge$$
$$(A[w] \geq 0 \rightarrow a[w] = A[w])\}$$

- La condición $a = A$ nos sirve para poder dar un nombre al valor del vector a **antes** de ejecutar el procedimiento. a en la postcondición se refiere a su valor **después** de ejecutarlo.
- En la postcondición debe indicarse que los valores positivos del vector no cambian.