Programació recursiva (2a sessió)

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2010-2011)
Versió 0.1

Avís: aquesta presentació no pretén ser un substitut dels apunts oficials de l'assignatura.

On som?

- Tema 5: Programació recursiva
- (inici) Tema 6: Millores d'eficiència en programes recursius i iteratius
- 8a sessió

Avui

- Més tècniques d'immersió: reforçament de la precondició.
- Transformació automàtica d'algorismes recursius en iteratius.
- Millores d'eficiència usant immersió en algorismes recursius.

Immersió per reforçament de la precondició

- *Definició*: afegir a la precondició una versió afeblida de la postcondició
- *Objectiu*: escurçar el camí cap a la consecució de l'objectiu que marca la postcondició.
- *Clau*: l'afegitó ha de poder complir-se fàcilment (1a crida recursiva està obligada a complir-la).

Especificació + enfortiment

```
int suma_vect_int(const vector<int> &v)
/* Pre: cert */
/* Post: el valor retornat es la suma de tots els elements
   del vector */
```

Immersió per enfortiment de la pre

Immersió: afegir paràmetres

- paràmetre que contingui ja una suma parcial dels elements del vector
  - una execució d'aquesta consisteix en afegir un altre element a la suma parcial.
  - Satisfem *objectiu* i *clau*.
- paràmetre extra que indiqui fins a quina posició del vector conté la suma parcial

Espeficicació de la funció d'immersió

```
int i_suma_vect_int(const vector<int> &v, int i, int suma_parcial)
/* Pre: 0 <= i <= v.size(); suma_parcial=suma dels primers
   i elements de v */
{
...
}
/* Post: el valor retornat es la suma de tots els elements
  del vector v */
```

Exercici:

- Implementació de la funció d'immersió.
- Crida a la funció d'immersió des de `suma_vect_int`.

Implementació de la funció d'immersió

```
int i_suma_vect_int(const vector<int> &v, int i, int suma_parcial)
/* Pre: 0 <= i <= v.size(); suma_parcial = suma dels primers
   i elements de v */
{
  int suma;
  if (i == v.size()) suma = suma_parcial;
  else suma = i_suma_vect_int(v, i + 1, suma_parcial + v[i]);

  return suma;
}
/* Post: el valor retornat es la suma de tots els elements
   del vector v */
```

Crida a la funció d'immersió

```
int suma_vect_int(const vector<int> &v)
/* Pre: cert */
{
  return i_suma_vect_int(v, 0, 0);
}
/* Post: el valor retornat es la suma de tots els elements
   del vector */
```

La crida a `i_suma_vect_int` satisfà la seva prec?

Enfortiment de la pre en la cerca d'un elements en un vector ordenat: una pinzellada

```
int cerca_vect_int(const vector<int> &v, int n)
/* Pre: v.size() > 0; v esta ordenat de creixentment */

/* Post: El valor retornat es la posicio on es troba l'element n dins
el vector v. Si n no es troba a v, llavors el valor retornat es un
nombre negatiu. */
```

dóna

```
int i_cerca_vect_int(const vector<int> &v, int n, int esq, int dre)
/* Pre: v.size() > 0; v esta ordenat creixentment;
   0 <= esq <= v.size(); -1 <= dre < v.size(); esq <= dre + 1;
   n no es troba ni a v[0..esq-1] ni a v[dre+1..v.size()-1]
   ... */

/* Post: El valor retornat es la posicio on es troba l'element n dins
el vector v. Si n no es troba a v, llavors el valor retornat es un
nombre negatiu. */
```

Recursivitat lineal final

- Algorisme recursiu lineal: algorisme recursiu que a cada crida recursiva genera solament una crida recursiva.
- Funció recursiva lineal és final:
  - Si la darrera instrucció que s'executa és la crida recursiva.
  - El resultat de la funció és el resultat que s'ha obtingut de la crida recursiva, sense cap modificació.
- Motivació: mètode simple de transformar un algorisme recursiu lineal final en un algorisme iteratiu.

Recursivitat lineal no final enfront final I

Recursivitat lineal no final:

```
int i_suma_vect_int(const vector<int> &v, int i)
/* Pre: v.size() > 0; 0 <= i < v.size() */
{
  int suma;
  if (i == 0) suma = v[0];
  else suma = i_suma_vect_int(v, i - 1) + v[i];
  return suma;
}
/* Post: el valor retornat es la suma de tots els elements
del vector v fins a la posicio i */
```

Funció amb postcondició constant: la postcondició de la crida recursiva és la mateixa que la postcondició de la funció.

Recursivitat lineal no final enfront final I

Recursivitat lineal final:

```
int i_suma_vect_int(const vector<int> &v, int i, int suma_parcial)
/* Pre: 0 <= i <= v.size(); suma_parcial = suma dels primers
   i elements de v */
{
  int suma;
  if (i == v.size()) suma = suma_parcial;
  else suma = i_suma_vect_int(v, i + 1, suma_parcial + v[i]);
  return suma;
}
/* Post: el valor retornat es la suma de tots els elements
   del vector v */
```

Transformació d'un algorisme recursiu lineal final en un algorisme iteratiu I

Esquema de funció recursiva lineal final:

```
Tipus2 f(Tipus1 x)
/* Pre: Q(x) */
{
  Tipus2 s;

  if (c(x)) s = d(x);
  else {
    s = f(g(x));
  }
  return s;
}
/* Post: R(x,s) */
```

Esquema original:

```
Tipus2 f(Tipus1 x)
/* Pre: Q(x) */
{
  Tipus2 r,s;

  if (c(x)) s = d(x);
  else {
    r= f(g(x));
    s= h(x,r);
  }
  return s;
}
/* Post: R(x,s) */
```

Transformació d'un algorisme recursiu lineal final en un algorisme iteratiu II

Esquema de la funció iterativa equivalent

```
Tipus2 f(Tipus1 x)
/* Pre: Q(x) */
{
  Tipus2 s;

  while (not c(x)) {
     x = g(x);
  }
  s = d(x);
  return s;
}
/* Post: R(x,s) */
```

- Condició del `while`: condició del cas recursiu ($\neg c(x)$).
- Instruccions del `while`: instruccions de reducció abans de la crida recursiva (`g(x)`).
- Instruccions de postprocessament a la sortida del `while`: instruccions del cas directe (`s = d(x)`).

Implementació recursiva lineal

```
int i_suma_vect_int(const vector<int> &v, int i, int suma_parcial)
/* Pre: 0 <= i <= v.size(); suma_parcial = suma dels primers
   i elements de v */
{
  int suma;

  if (i == v.size()) suma = suma_parcial;
  else suma = i_suma_vect_int(v, i + 1, suma_parcial + v[i]);

  return suma;
}
/* Post: el valor retornat es la suma de tots els elements
   del vector v */
```

Transformació a iteratiu

Clau: adaptar l'esquema general d'una funció recursiva amb un sol paràmetre a tres paràmetres.

```
int i_suma_vect_int(const vector<int> &v, int i, int suma_parcial)
/* Pre: 0 <= i <= v.size(); suma_parcial=suma dels primers
   i elements de v */
{
  int suma;

  while (i != v.size()) {
    suma_parcial = suma_parcial + v[i];
    ++i;
  }
  suma = suma_parcial;

  return suma;
}
/* Post: el valor retornat es la suma de tots els elements
   del vector v */
```

Concepte d'eficiència

Cost d'un algorisme:

- Temporal: "nombre d'operacions de l'ordre de"
- Espacial: "nombre de bytes de l'ordre de"

Costos temporals sobre un vector de mida n:

- Cerca seqüencial: ?
- Cerca dicotòmica: ?
- Ordenació per selecció o inserció: ?
- Ordenació per barreja ("mergesort"): ?
- Ordenació ràpida ("quicksort"): ?

La successió de Fibonacci

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, \ldots$$

$$f(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f(n-1) + f(n-2) & \text{si } n \geq 2 \end{cases} \quad (1)$$

Implementació

```
int fibonacci(int n)
{
// Pre: n >= 0
int f;
if (n < 2) f = n;
else {
   f = fibonacci(n-1);
   int f1 = fibonacci(n-2);
   f = f + f1;
}
return f;
// Post: retorna el terme n-èssim de la successió de Fibonacci
}
```

Prova

- Execució de `fibonacci.cpp`
- Quin és el problema?
- Cost temporal?

Detecció de la repetició de càlculs en programes recursius

- Càlcul per duplicat de $f(n-2)$. Evitar-lo.
- Solució: immersió.
- Interès: cas d'immmersió augmentant la informació retornada.

```
#include <utility>
...
pair<int,int> i_fibonacci(int n) {
// Pre: n > 0
...
// Post: retorna a "first" el terme n-èssim de la successió
// de Fibonacci i a "second" el terme (n-1)-èssim.
}
```

Enfortiment de la prec i de la post.

Implementació funció d'immersió

```
pair<int,int> i_fibonacci(int n) {
// Pre: n > 0
pair<int,int> f2;
if (n == 1) {
    f2.first = 1;
    f2.second = 0;
}
else {
    f2 = i_fibonacci(n - 1);
    // HI: f2.first conté el terme (n-1)-èssim de la successió de Fibonacci;
    //     f2.second en conté el terme (n-2)-èssim
    f2 = pair<int,int> (f2.first + f2.second, f2.first);
}
return f2;
// Post: retorna a "first" el terme n-èssim de la successió de Fibonacci i
//       a "second" el terme (n-1)-èssim.
}
```

Recursivitat lineal? Final?

Exercici: crida a la funció d'immersió

Exercici

```
int fibonacci(int n) {
// Pre: n >= 0
...
// Post: retorna el terme n-èssim de la successió de Fibonacci
```

Solució

```
int fibonacci(int n) {
// Pre: n >= 0
  int f;

  if (n == 0) f = 0;
  else {
     pair<int,int> f2 = i_fibonacci(n);
     f = f2.first;
  }
  return f;
// Post: retorna el terme n-èssim de la successió de Fibonacci
```

Extra

- Justificacions d'algorismes de la sessió anterior. Per exemple: cerca dicotòmica.
- Practicar immersions. Per exemple: suma dels elements d'una matriu.