Millores d'eficiència en programes recursius i iteratius

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2010-2011)
Versió 0.4

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

On som?

- Tema 5: Programació recursiva
- 9a sessió

Avui

- Més exemples d'immersions d'eficiència en algorismes recursius (sobre tipus abstractes de dades)
- Millores d'eficiència en algorismes iteratius.

Determinar si dos arbres son equilibrats

Un arbre és equilibrat si i només si:

- Els arbres fills són equilibrats.
- La diferència d'alçades dels subarbres fills no supera la unitat.

```
bool equilibrat(const BinTree<int> &a) {
   // Pre: cert

   // Post: el valor retornat indica si a es un arbre equilibrat
}
```

Concepte d'arbre equilibrat: clau en estructures de dades avançades.

Implementació I

```
bool equilibrat(const BinTree<int> &a) {
   // Pre: cert
   bool b1;

   if (a.empty()) b1 = true;
   else {
      b1 = equilibrat(a.left());
      bool b2 = equilibrat(a.right());
      int h1 = alcada(a.left());
      // HI1 : h1 es l'alçada del fill esquerre d'a
      int h2 = alcada(a.right());
      // HI2 : h2 es l'alçada de fill dret d'a
      b1 = (abs(h1 - h2) <= 1) and b1 and b2;
   }
   return b1;
   // Post: el valor retornat indica si a es un arbre equilibrat
}
```

Implementació II

Suposem que ja tenim implementada la funció

```
int alcada(const BinTree<int> &a) {
  // Pre: cert

  // Post: el valor retornat es la longitud del camí més llarg de l'arrel
  //       a una fulla de l'arbre a
}
```

Anàlisi de l'eficiència

- Problemes d'eficiència? (anàlisi de l'arbre de crides)
- Quin és el cost de l'algorisme?
- Millores d'eficiència:
  - Millorant el codi seguint l'esquema de la implementació proposada.
  - Reenginyeria (immersió).

Solució: immersió d'eficiència

Retornar + informació per evitar càlculs redundants

```
pair<bool, int> i_equilibrat(const BinTree<int> &a) {
   // Pre: cert

   // Post: en el valor retornat
   //        - "first" indica si a es un arbre equilibrat
   //        - "second" conté l'alçada de l'arbre
}
```

Implementació de la funció d'immersió

```
pair<bool, int> i_equilibrat(const BinTree<int> &a) {
   // Pre: cert
   pair<bool, int> e;
   if (a.empty()) {
      e.first = true;
      e.second = 0;
   }
   else {
      pair<bool, int> e1 = i_equilibrat(a.left());
             // HI1: e1.first indica si el fill esquerre d'a es un arbre equilibrat
             //         e1.second en conté l'alçada
      pair<bool, int> e2 = i_equilibrat(a.right());
             // HI2: e2.first indica si el fill dret d'a es un arbre equilibrat
             //         e2.second en conté l'alçada
      e.first = (abs(e1.second - e2.second) <= 1) and
                e1.first and e2.first;
      e.second = 1 + max(e1.second, e2.second);
   }
   return e;
   // Post: en el valor retornat
   //       - "first" indica si A es un arbre equilibrat
   //       - "second" conté l'alçada de l'arbre
}
```

Millora d'eficiència addicional?

Crida a la funció d'immersió

```
bool equilibrat2(const BinTree<int> &a) {
   // Pre: cert
   pair<bool, int> e = i_equilibrat(a);
   return e.first;
   // Post: indica si a es un arbre equilibrat
}
```

Funció exponencial

Sèrie de Taylor de l'exponencial

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} + ... \tag{1}$$

```
double exponencial(int x, int n) {
   // Pre: x > 0; n >= 0

   // Post: el valor retornat es la suma dels n primers termes de l'expansio
   //       en sèrie de Taylor de e^x
}
```

Implementació I

```
double exponencial(int x, int n) {
   // Pre: x > 0; n >= 0
   double e = 0;
   int i = 0;
   // Inv: 0 <= i <= n;
   //       e conté la suma dels i primers termes de l'expansio en sèrie
   //       de Taylor de la funció exponencial
   while (i < n) {
       int p = potencia(x, i);
       int f = factorial(i);
       e += double(p)/f;
       ++i;
   }
   return e;
   // Post: el valor retornat es la suma dels n primers termes de l'expansio
   //        en sèrie de Taylor de e^x
}
```

Implementació II

Especificació de les dues funcions auxiliars:

```
int potencia(int x, int n) {
   // Pre: n >= 0; x != 0 si n = 0

   // Post: retorna x^n
}
```

Definició de la Pre: evitar indeterminació 0^0 quan $x = n = 0$.

```
int factorial(int n) {
   // Pre: n >= 0

   // Post: retorna n!
}
```

Millora d'eficiència

Càlculs repetits ($i > 0$):

- A cada iteració càlcul de potencies des de zero però `potencia(x, i) = x*potencia(x, i - 1)`
- A cada iteració càlcul del factorial de zero però `factorial(i) = i*factorial(i - 1)`

Solució: reciclar càlculs (introduir noves variables locals)

Implementació (2a versió)

```
double exponencial(int x, int n) {
   // Pre: x > 0; n >= 0
   double e = 0;
   int p = 1; // p conté x^0
   int f = 1; // f conté 0!
   int i = 0;
   // Inv: 0 <= i <= n;
   //       p conté x^i; f conté i!;
   //       e conté la suma dels i primers termes de l'expansio en sèrie
   //       de Taylor de la funció exponencial
   while (i < n) {
       e += double(p)/f;
       p *= x;
       ++i;
       f *= i;
   }
   return e;
   // Post: e conté la suma dels n primers termes de l'expansio en sèrie
   //        de Taylor de e^x
}
```

Dubte

No hauria de ser?

```
...
while (i < n) {
    e += double(p)/f;
    p *= x;
    f *= i;
    ++i;
}
...
```

Pista: justificar a partir de l'invariant.

Millora d'eficiència

Problema `exponencial2`

- Vessaments evitables (`p` i `f` es fan massa grans)
- Variables temporals innecessaries

Solució:

- Sèrie de Taylor de l'exponencial

$$e^x = t_0 + t_1 + t_2 + t_3 + \ldots + t_n + \ldots \tag{2}$$

- Relació de recurrència:
  - $t_n = 1$ si $n = 0$
  - $t_n = \frac{x}{n} t_{n-1}$ si $n > 0$

Implementació (3a versió)

```
double exponencial(int x, int n) {
   // Pre: x > 0; n >= 0
   double e = 0;
   double t = 1; // t conté x^0/(0!)
   int i = 0;
   // Inv: 0 <= i <= n
   //       t conté x^i/(i!);
   //       e conté la suma dels i primers termes de l'expansio en sèrie
   //       de Taylor de la funció exponencial
   while (i < n) {
       e += t;
       ++i;
       t = t*x/i;
   }
   return e;
   // Post: el valor retornat es la suma dels n primers termes de l'expansió
   //        en sèrie de Taylor de e^x
}
```

Funció cosinus

Sèrie de Taylor del cosinus

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots \tag{3}$$

```
double cosinus(double x, int n) {
   // Pre: n >= 0

   // Post: e conté la suma dels n primers termes de l'expansio en sèrie
   //       de Taylor del cosinus de x
}
```

Pista

- $t_n = 1$ si $n = 0$
- $t_n = ...t_{n-1}$ si $n > 0$

Exercicis

Discutir estratègies eficients per als problemes següents:

- Element dominador més alt d'una pila de naturals (recursiu).
- Element dominador més avançat d'una llista de naturals (iteratiu).
- Vector mitjanament ordenat (iteratiu).