Millores d'eficència en recursió
Programació 2
Facultat d'Informàtica d'Informàtica, UPC

Professorat de PRO2

Tardor 2022

- Col·laboracions (en ordre alfabètic): Juan Luis Esteban, Ricard Gavaldà, Conrado Martínez, Fernando Orejas
- Aquestes transparències **no** substitueixen els apunts de l'assignatura, els complementen

Part I

Eficiència per eliminació de càlculs repetits

Iteració:

- Afegim variables locals que recorden càlculs ja efectuats per a la propera iteració
- No apareixen en la Pre ni la Post. L'especificació no canvia
- Però apareixen a l'invariant. Cal dir què valen a cada iteració

Eficiència per eliminació de càlculs repetits

Recursió:

- Les variables locals no serveixen. Es creen noves a cada crida

Eficiència per eliminació de càlculs repetits

Recursió:

- Les variables locals no serveixen. Es creen noves a cada crida
- Funció d'immersió d'eficiència, recursiva: Nous paràmetres d'entrada o de sortida

Eficiència per eliminació de càlculs repetits

Recursió:

- Les variables locals no serveixen. Es creen noves a cada crida
- Funció d'immersió d'eficiència, recursiva: Nous paràmetres d'entrada o de sortida
- S'han d'afegir a la Pre/Post!

Eficiència per eliminació de càlculs repetits

Recursió:

- Les variables locals no serveixen. Es creen noves a cada crida
- Funció d'immersió d'eficiència, recursiva: Nous paràmetres d'entrada o de sortida
- S'han d'afegir a la Pre/Post!
- La funció desitjada no és recursiva, crida a la d'immersió

Part I

Concepte d'immersió d'eficiència

- Font freqüent d'ineficiència: Repetir càlculs ja fets
- En programes iteratius: Guardar variables temporals que guarden resultats d'una iteració a la següent
- En programes recursius, una variable temporal és local *a cada crida recursiva*. No guarda resultats d'una crida a l'altra
- Immersió d'eficiència: Introducció de paràmetres o resultats addicionals per transmetre valors ja calculats en/a altres crides
- Pot haver de fer-se *a més* una immersió/generalització per tal de possibilitar la solució recursiva.

Determinar si un arbre és equilibrat

Concepte important en estructures de dades avançades:
Un arbre és equilibrat si i només si

- els seus dos fills són equilibrats, i a més
- la diferència d'alçades dels subarbres fills no supera la unitat.

Determinar si un arbre és equilibrat

```
// Pre: cert
// Post: retorna cert si i només si a és un arbre equilibrat
bool equilibrat(const BinTree<int> &a);
```

Determinar si un arbre és equilibrat

```
// Pre: cert
// Post: retorna cert si i només si a és un arbre equilibrat
bool equilibrat(const BinTree<int> &a);
```

Suposem que ja tenim implementada la funció

```
  // Pre: cert
  // Post: retorna la longitud del camí més llarg de l'arrel
  //       a una fulla de l'arbre a
int alcaria(const BinTree<int> &a);
```

i que tarda temps proporcional a la mida de l'arbre

Implementació, II

```
bool equilibrat(BinTree<int> &a) {
  if (a.empty()) return true;
  else return abs(alcaria(a.left())-alcaria(a.right())) <= 1 and
              equilibrat(a.left()) and equilibrat(a.right());
}
```

Anàlisi de l'eficiència

- Quin és el cost de l'algorisme?
  - Analitzem l'arbre de crides...

Anàlisi de l'eficiència

- Quin és el cost de l'algorisme?
  - Analitzem l'arbre de crides...
  - Pensem en un arbre lineal (cap fill dret, fill només esquerre) ...

Anàlisi de l'eficiència

- Quin és el cost de l'algorisme?
  - Analitzem l'arbre de crides...
  - Pensem en un arbre lineal (cap fill dret, fill només esquerre) ...
  - $|a|^2/2$

Anàlisi de l'eficiència

- Quin és el cost de l'algorisme?
  - Analitzem l'arbre de crides...
  - Pensem en un arbre lineal (cap fill dret, fill només esquerre) ...
  - $|a|^2/2$
  - (don't panic: A EDA practicarem això)

Anàlisi de l'eficiència

- Quin és el cost de l'algorisme?
  - Analitzem l'arbre de crides...
  - Pensem en un arbre lineal (cap fill dret, fill només esquerre) ...
  - $|a|^2/2$
  - (don't panic: A EDA practicarem això)
- Com evitem repetir càlculs, recorrer cada arbre molts cops?

Solució: immersió d'eficiència

Retornar més informació per evitar repetir càlculs:

```
   // Pre: cert
   // Post: en el parell retornat
   //       - "first" indica si a es un arbre equilibrat
   //       - "second" conté l'alçaria de l'arbre si a és equilibrat
pair<bool, int> i_equilibrat(const BinTree<int>& a);
```

Solució: immersió d'eficiència

Retornar més informació per evitar repetir càlculs:

```
   // Pre: cert
   // Post: en el parell retornat
   //       - "first" indica si a es un arbre equilibrat
   //       - "second" conté l'alçaria de l'arbre si a és equilibrat
pair<bool, int> i_equilibrat(const BinTree<int>& a);
```

Crida inicial:

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

Implementació de la funció d'immersió

```
   // Pre: cert
   // Post: en el parell retornat
   //   - "first" indica si a es un arbre equilibrat
   //   - "second" és l'alçària de l'arbre, si a és equilibrat
pair<bool, int> i_equilibrat(const BinTree<int>& a) {
   if (a.empty()) return make_pair<true, 0>;
   else {
     pair<bool, int> e1 = i_equilibrat(a.left());
     //  HI1: e1.first indica si a.left() és un arbre equilibrat
     //       e1.second és l'alçaria d'a.left(), si és equilibrat
     if (e1.first) {
         ...
     } else { // e1.first == false
         return make_pair(false, -1);
     }
}
```

Implementació de la funció d'immersió

```
if (e1.first) {
  pair<bool, int> e2 = i_equilibrat(a.right());
  //  HI2: e2.first indica si a.right() és un arbre equilibrat
  //       e2.second és l'alçària d'a.right(), si és equilibrat
  bool eq = e2.first and (abs(e1.second - e2.second) <= 1);
            // ja sabem que e1.first == true
  if (eq)
    return make_pair(true,1 + max(e1.second, e2.second));
  else
    return make_pair(false, -1); //l'alçaria és irrellevant

}
```

Implementació de la funció d'immersió

```
   // Pre: cert
   // Post: en el parell retornat
   //   - "first" indica si a es un arbre equilibrat
   //   - "second" és l'alçaria de l'arbre, si a és equilibrat
pair<bool, int> i_equilibrat(const BinTree<int>& a) {
   if (a.empty()) return make_pair<true, 0>;
   else {
     pair<bool, int> e1 = i_equilibrat(a.left());
     if (e1.first) {
       pair<bool, int> e2 = i_equilibrat(a.right());
       bool eq = e2.first and (abs(e1.second - e2.second) <= 1);
                 // ja sabem que e1.first == true
       if (eq)
         return make_pair(true,1 + max(e1.second, e2.second));
       else
         return make_pair(false, -1); //l'alçaria és irrellevant

     } else { // e1.first == false
         return make_pair(false, -1);
     }
}
```

Mitjana de valors

```
// Pre: a no està buit
// Post: el resultat és la mitjana dels valors de l'arbre a
double mitjana(const BinTree<double> & a)
```

Mitjana de valors

```
// Pre: a no està buit
// Post: el resultat és la mitjana dels valors de l'arbre a
double mitjana(const BinTree<double> & a)
```

No sembla fàcil calcular la mitjana d'un arbre a partir de les mitjanes dels seus subarbres i la informació de l'arrel.

Mitjana de valors

```
// Pre: a no està buit
// Post: el resultat és la mitjana dels valors de l'arbre a
double mitjana(const BinTree<double> & a)
```

No sembla fàcil calcular la mitjana d'un arbre a partir de les mitjanes dels seus subarbres i la informació de l'arrel.
Hem de pensar una altra estratègia...

Mitjana de valors

Per exemple aprofitar funcions ja fetes.

```
// Pre: a no està buit
// Post: el resultat és la mitjana dels valors de l'arbre a
double mitjana(const BinTree<double> & a)
{
  return sum(a)/size(a);
}
```

Mitjana de valors

Per exemple aprofitar funcions ja fetes.

```
// Pre: a no està buit
// Post: el resultat és la mitjana dels valors de l'arbre a
double mitjana(const BinTree<double> & a)
{
  return sum(a)/size(a);
}
```

Bé, `sum` no la tenim feta, però seria fàcil de fer.

Mitjana de valors

Per exemple aprofitar funcions ja fetes.

```
// Pre: a no està buit
// Post: el resultat és la mitjana dels valors de l'arbre a
double mitjana(const BinTree<double> & a)
{
  return sum(a)/size(a);
}
```

Bé, `sum` no la tenim feta, però seria fàcil de fer.
Intentem fer una operació que ens torni la mida d'un arbre i la suma dels seus elements.

Mitjana de valors

```
// Pre: cert
// Post: s és la suma dels elements d'a i m és la mida d'a
double mitjana_aux(const BinTree<double> & a, double & s, int & m)
```

Mitjana de valors

```
// Pre: cert
// Post: s és la suma dels elements d'a i m és la mida d'a
double mitjana_aux(const BinTree<double> & a, double & s, int & m){
  if (a.empty()) {s = 0.0; m = 0;}
  else{
        double si, sd;
        int mi, md;
        mitjana_aux(a.left(), si, mi);
        /* HI1: si és la suma dels elements d'a.left() i
                mi és la mida d'a.left() */
        mitjana_aux(a.right(), sd, md);
        /* HI2: sd és la suma dels elements d'a.right() i
                md és la mida d'a.right() */
        s = a.value() + si + sd;
        m = 1 + mi + md;
  }
```

La justificació d'aquest codi és estàndard i es deixa com a exercici.

Mitjana de valors

Per últim cal donar el codi de l'operació `mitjana`

```
// Pre: a no està buit
// Post: el resultat és la mitjana dels valors de l'arbre a
double mitjana(const BinTree<int> & a)
{
        double suma;
        int mida;
        mitjana_aux(a, suma, mida);
        return suma/mida;
}
```

Arbre de mitjanes

Donat un arbre de doubles, construir-ne un altre de la mateixa forma que a cada node conté la mitjana dels valors del subarbre arrelat al node corresponent de l'original

```
// Pre: cert
// Post: retorna l'arbre de mitjanes d'a
BinTree<double> arbre_mitjanes(const BinTree<double>& a);
```

Arbre de mitjanes

Donat un arbre de doubles, construir-ne un altre de la mateixa forma que a cada node conté la mitjana dels valors del subarbre arrelat al node corresponent de l'original

```
// Pre: cert
// Post: retorna l'arbre de mitjanes d'a
BinTree<double> arbre_mitjanes(const BinTree<double>& a);
```

Dificultat: la mitjana d'un arbre NO es pot calcular a partir de l'arrel i les mitjanes dels dos subarbres

Arbre de mitjanes: Solució ineficient

```
BinTree<double> arbre_mitjanes(const BinTree<double>& a) {
    if (not a.empty()) {
        double x = a.value();
        BinTree<double> b1 = arbre_mitjanes(a.left());
        BinTree<double> b2 = arbre_mitjanes(a.right());
        double s1 = suma(a.left()); double s2 = suma(a.right());
        int n1 = talla(a.left()); int n2 = talla(a.right());
        return BinTree<double>((x+s1+s2)/(1+n1+n2),b1,b2);
    }
}
```

Ineficient: Tres recorreguts d'a (recursió, suma, mida)

Arbre de mitjanes

Immersió d'eficiència: un sol recorregut que retorni a més suma i mida

```
// Pre: b és buit
// Post: b és l'arbre de mitjanes d'A, s conté
// la suma dels nodes d'A, i n conté el nombre de nodes d'A
void ie_arbre_mitjanes(const BinTree<double>& a, BinTree<double>& b,
                       double& s, int& n);
```

Arbre de mitjanes

Immersió d'eficiència: un sol recorregut que retorni a més suma i mida

```
// Pre: b és buit
// Post: b és l'arbre de mitjanes d'A, s conté
// la suma dels nodes d'A, i n conté el nombre de nodes d'A
void ie_arbre_mitjanes(const BinTree<double>& a, BinTree<double>& b,
                       double& s, int& n);
```

Crida inicial:

```
BinTree<double> arbre_mitjanes(const BinTree<double>& a) {
  double s; int n;
  BinTree<double> b;
  ie_arbre_mitjanes(a,b,s,n);
  return b;
}
```

Arbre de mitjanes

```
void ie_arbre_mitjanes(const BinTree<double>& a, BinTree<double>& b,
                       double& s, int& n) {
   if (a.empty()) { s = 0; n = 0; }
   else {
       double s1, s2; int n1, n2;
       double x = a.value();
       BinTree<double> b1, b2;
       ie_arbre_mitjanes(a.left(),b1,s1,n1);
       // HI1: b1 és l'arbre de mitjanes d'a.left(), s1 és la suma
       // dels nodes d'a.left(), i n1 és el nombre de nodes d'a.left()
       ie_arbre_mitjanes(a.right(),b2,s2,n2);
       // HI2: b2 és l'arbre de mitjanes d'a.right(), s2 és la suma
       // dels nodes d'a.right(), i n2 és el nombre de nodes d'a.right()
       s = x + s1 + s2;
       n = 1 + n1 + n2;
       b = BinTree<double>(s/n,b1,b2);
   }
}
```

Cost lineal, un sol recorregut

Part I

Implementació recursiva

```
// Pre: n ≥ 0
// Post: retorna F_n
int fibonacci(int n) {
    if (n <= 1) return n;
    else return fibonacci(n-1) + fibonacci(n-2);
}
```

Implementació recursiva

```
// Pre: n ≥ 0
// Post: retorna F_n
int fibonacci(int n) {
    if (n <= 1) return n;
    else return fibonacci(n-1) + fibonacci(n-2);
}
```

Quants cops cridem fibonacci(n-i) en executar fibonacci(n)?

Implementació recursiva

```
// Pre: n ≥ 0
// Post: retorna F_n
int fibonacci(int n) {
    if (n <= 1) return n;
    else return fibonacci(n-1) + fibonacci(n-2);
}
```

Quants cops cridem fibonacci(n-i) en executar fibonacci(n)?
Resposta: F_i cops (intenteu demostrar-ho per inducció)

Cost temporal?

F_n creix com ϕ^n

($\phi = (1+\sqrt{5})/2$ = raó àuria = solució de $(\phi = 1 + 1/\phi)$ = $\simeq 1.618033\ldots$)

Càlcul molt lent

Truc

Suposem que tenim el parell $\langle F_{n-1}, F_{n-2} \rangle$. Llavors, el parell $\langle F_n, F_{n-1} \rangle$ és

$$\langle F_n, F_{n-1} \rangle = \langle F_{n-1} + F_{n-2}, F_{n-1} \rangle$$

Detecció de la repetició de càlculs en programes recursius

```
#include <utility>

// Pre: n > 0
// Post: retorna ⟨F_n, F_{n-1}⟩
pair<int,int> i_fibonacci(int n);
```

Implementació funció d'immersió

```
// Pre: n > 0
// Post: retorna ⟨F_n, F_{n-1}⟩
pair<int,int> i_fibonacci(int n) {
    if (n == 1) return make_pair(1,0);
    } else {
        pair<int,int> p = i_fibonacci(n - 1);
        // HI: p.first i p.second contenen F_{n-1} i F_{n-2} resp.
        return make_pair(p.first + p.second, p.first);
    }
}
```

Crida a la funció d'immersió

```
// Pre: n ≥ 0
// Post: retorna F_n
int fibonacci(int n) {
    if (n == 0) return 0;
    else return i_fibonacci(int n).first;
}
```

Alternativa

Funcions que retornen més d'un valor

$\rightarrow$ paràmetres per referència

```
// Pre: n > 0
// Post: retorna f1 = F_n i f2 = F_{n-1}
void i_fibonacci(int n, int& f1, int& f2);
```