Programación 2
Mejoras en la eficiencia de algoritmos recursivos

Fernando Orejas

Eliminación de cálculos repetidos en algoritmos recursivos

Eliminación de cálculos repetidos

Algoritmos recursivos:

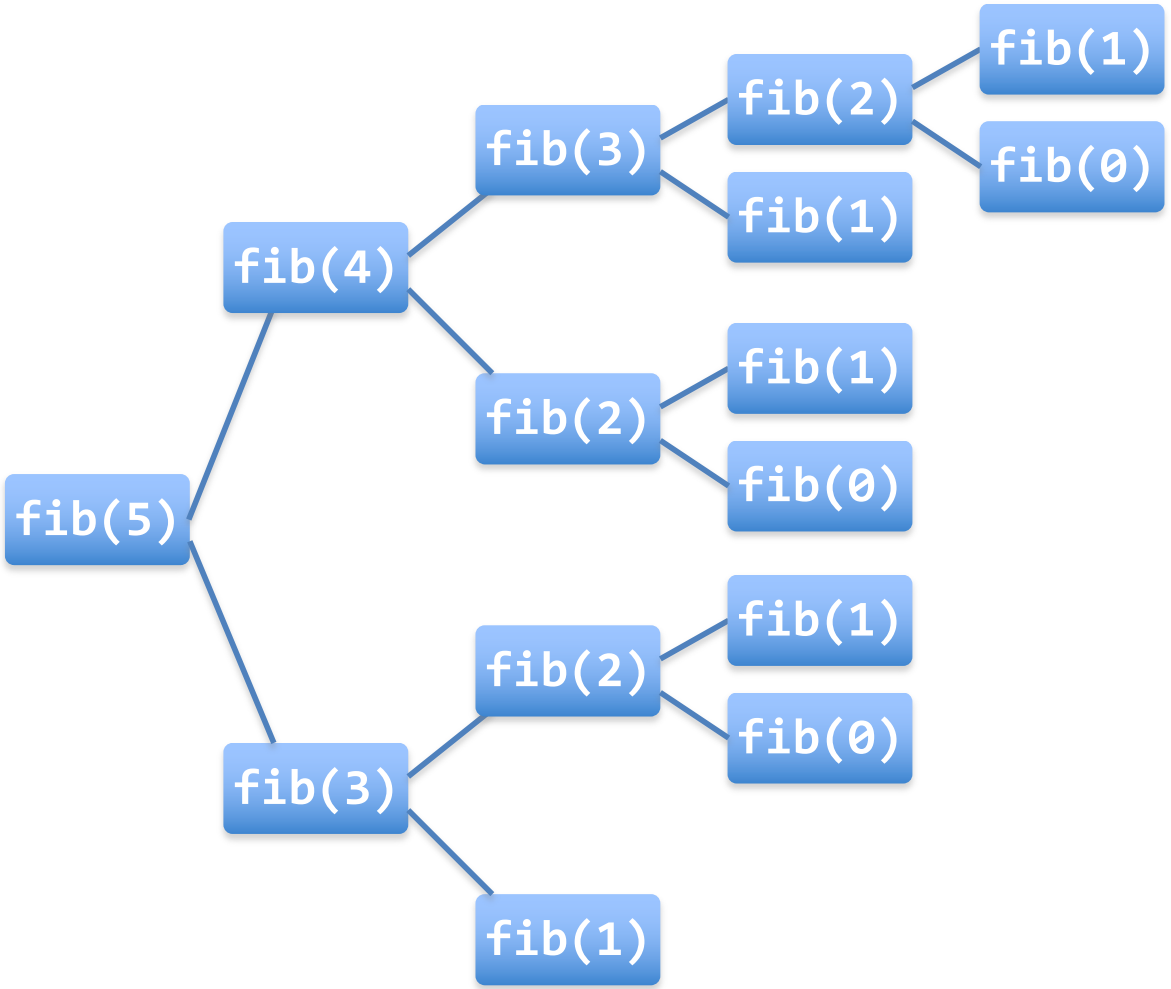
- Las variables locales son inútiles
- Hacemos una inmersión modificando la Pre/Post, donde los parámetros adicionales guardan resultados que permiten no repetir cálculos
- La función deseada hace una llamada a la inmersión

Números de Fibonacci

```
int fib (int n) {
// Pre:  n >= 0
// Post: Retorna el número de Fibonacci de orden n.
```

```
int fib (int n) {// versión recursiva
// Pre:  n >= 0
// Post: Devuelve el número de Fibonacci de orden n.

    if (n == 0  or n == 1) return 1;
    else return fib(n-2) + fib(n-1);
}
```



¡¡Muy ineficiente!!
¡¡Exponencial!!

```
/* Pre: 0 < i <= n, fact es el número de fibonacci de orden
i, fant es el número de fibonacci de orden i-1 */
/* Post: Retorna el número de Fibonacci de orden n */

int fib1 (int n; int i, int fact, int fant);
```

```
/* Pre: 0 < i <= n, fact es el número de fibonacci de orden
i, fant es el número de fibonacci de orden i-1 */
/* Post: Retorna el número de Fibonacci de orden n */

int fib1 (int n; int i, int fact, int fant);

int fib (int n) {
    if (n == 0) return 1;
     else return fib1(n,1,1,1);
}
```

```
/* Pre: 0 < i <= n, fact es el número de fibonacci de orden
i, fant es el número de fibonacci de orden i-1 */
/* Post: Retorna el número de Fibonacci de orden n */

int fib1 (int n; int i, int fact, int fant) {
    if (n == i) return fact;
    else return fib1(n, i+1, fact+fant, fact);
}

int fib (int n) {
    if (n == 0) return 1;
     else return fib1(n,1,1,1);
}
```

Árboles equilibrados

Un árbol es equilibrado si:

- Está vacío o
- Sus dos hijos son equilibrados y
- La diferencia de altura entre sus hijos es uno, como máximo

Árbol equilibrado

```
// Pre:  true
// Post: nos dice si el árbol a está equilibrado

bool equilibrado(const BinTree <int> &a);
```

Árbol equilibrado

```
// Pre:  true
// Post: nos dice si el árbol a está equilibrado

bool equilibrado(const BinTree <int> &a);
```

Supongamos que ya tenemos implementada la función altura:

```
// Pre:  true
// Post: nos retorna la altura del árbol a

int altura(const BinTree <int> &a);
```

Árbol equilibrado

```
// Pre:  true
// Post: nos dice si el árbol a está equilibrado

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

¡¡¡Muy Ineficiente!!!

Árbol equilibrado (inmersión de eficiencia)

Retornamos más información para evitar repetir los cálculos

```
// Pre:  true
/* Post: En el par retornado:
   first indica si a es un árbol equilibrado y
   second es la altura de a, si a es equilibrado */
pair<bool,int> i_equil(const BinTree <int> &a);

// Llamada inicial
bool equilibrado(const BinTree <int> &a){
   pair<bool,int> p = i_equil(a);
   return p.first();
}
```

Árbol equilibrado (inmersión de eficiencia)

Retornamos más información para evitar repetir los cálculos

```
pair<bool,int> i_equil(const BinTree <int> &a){
  if (a.empty()) return make_pair(true,0);
  else{
    pair<bool,int> p1 = i_equil(a.left());
    if(not p1.first()) return make_pair(false,-1);
    else{
      pair<bool,int> p2 = i_equil(a.right());
      bool eq = p2.first() and (abs(p1.second()-
                                p2.second()) <= 1);
      if (eq) return make_pair(true,
                      max(p1-second(),p2.second())+1);
      else{return make_pair(false,-1);
    }
  }
}
```

Media de los valores de un árbol

```
// Pre:  a no está vacío
// Post: Retorna la media de los valores de a


BinTree<double> media(BinTree<double> &a};
```

Media de los valores de un árbol

```
// Pre:  a no está vacío
// Post: Retorna la media de los valores de a


BinTree<double> media(BinTree<double> &a}{
    return suma(a)/size(a);
}
```

```
// Inmersión de eficiencia
// Pre:  true
// Post: s es la suma de los valores de a y n es el
//    número de nodos de a

void aux_medias(BinTree<double> &a, double &s, int &n){
   if (a.empty()) {
      s = 0.0; n = 0;
   else{
      double si, sd; int ni, nd;
      aux_medias(a.left(), si, ni);
      aux_medias(a.right(), sd, nd);
      s = si + sd + a.value();
      n = ni + nd + 1;
   }
}
```

Media de los valores de un árbol

```
// Llamada inicial
// Pre:  a no está vacío
// Post: Retorna la media de los valores de a


BinTree<double> media(BinTree<double> &a}{
    double s; int n;
    aux_media(a, s, n);
    return s/n;
}
```

Árbol de medias

Dado un árbol de doubles, se quiere retornar otro, con la misma forma, en que cada nodo contenga la media de los nodos del subárbol original enraizado en ese nodo

```
// Pre:  true
// Post: retorna el árbol de medias de a

BinTree<double> a_medias(BinTree<double> &a};
```

```
// Pre:  true
// Post: retorna el árbol de medias de A


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

```
// Inmersión de eficiencia
// Pre:  true
// Post: b es el árbol de medias de A, s contiene
//   la suma de los nodos de A y n contiene el número de
//   nodos de A

void i_medias(BinTree<double> &a, BinTree <double> &b,
double &s, int &n);
```

```
// Inmersión de eficiencia
// Pre:  true
// Post: b es el árbol de medias de A, s contiene
//   la suma de los nodos de A y n contiene el número de
//   nodos de A

void i_medias(BinTree<double> &a, BinTree <double> &b,
double &s, int &n);

// Llamada inicial


BinTree<double> a_medias(Arbol<double> &a){
   double s; int n;
   BinTree <double> &b;
   i_medias(a, b, s, n);
   return b;
}
```

```
// Pre:  a = A, b está vacío
// Post: b es el árbol de medias de A, s contiene
//   la suma de los nodos de A y n contiene el número de
//   nodos de A


void i_medias(BinTree <double> &a, BinTree <double> &b,
              double &s, int &n){
   if (a.empty()) {s = 0; n = 0;}
   else {
      double s1,s2; int n1, n2;
       BinTree <double> b1, b2;
      double x = a.value();
      i_medias(a.left(),b1,s1,n1);
      i_medias(a.right,b2,s2,n2);
      s = x+s1+s2;
      n = n1+n2+1;
      b = BinTree <double> (s/n,b1,b2);
   }
}
```