equil.cc

```
/* Este algoritmo es ineficiente */


int altura(const BinTree<int>& a) {
/* Pre: cierto */
/* Post: El resultado es la longitud del camino mas
largo desde la raiz a una de las hojas de a. */
  int h;
  if (a.empty()) h = 0;
  else {
     int h1 = altura(a.left());
     /* HI: El resultado es la longitud del camino mas
     largo desde la raiz a una de las hojas de a.left(). */
     int h2 = altura(a.right());
     /* HI: El resultado es la longitud del camino mas
     largo desde la raiz a una de las hojas de a.right(). */
     h = max(h1,h2) + 1;
  }
  return h;
}


bool equil(const BinTree<int>& a) {
/* Pre: cierto */
/* Post: El resultado indica si a es equilibrado. */
  bool r;
  if (a.empty()) r = true;
  else {
     int h1 = altura(a.left());
     int h2 = altura(a.right());
     if (abs(h1 - h2) > 1) r = false;
     else {
        r = equil(a.left());
        /* HI: r indica si a.left() es equilibrado. */
        if (r) r = equil(a.right());
        /* HI: r indica si a.right() es equilibrado. */
     }
  }
  return r;
}



/* El siguiente algoritmo es mas eficiente. Es lineal en el tamano de a. */


pair<bool,int> equil_res(const BinTree<int>& a) {
/* Pre: cierto */
/* Post: La primera componente del resultado indica si a es equilibrado.
Si la primera componente del resultado es cierta, la segunda componente
es la altura de a. */
  pair<bool,int> r;
  if (a.empty()) r = make_pair(true,0);
  else {
     r = equil_res(a.left());
     /* HI: r.first indica si a.left() es equilibrado. Si r.first es true,
     r.second es la altura de a.left(). */
     if (r.first) {
        pair<bool,int> r2 = equil_res(a.right());
     /* HI: r2.first indica si a.right() es equilibrado. Si r2.first es true,
     r.second es la altura de a.right(). */
        if (not r2.first) r.first = false;
        else if (abs(r.second - r2.second) > 1) r.first = false;
        else r = make_pair(true, max(r.second,r2.second) + 1);
     }
  }
  return r;
}
```