

# Verifica bilanciamento nel numero dei nodi: definizioni.

**Definizione:** Un albero è bilanciato nel Numero dei Nodi, brevemente **n-bilanciato**, quando, per ogni sottoalbero t radicato in un suo nodo, il numero dei nodi del sottoalbero sinistro di t, meno il numero dei nodi del sottoalbero destro di t è in valore assoluto al più 1.

E' evidente che l'albero **vuoto** è n-bilanciato e che un albero t, con sottoalberi t1 e t2, rispettivamente con n1 e n2 nodi, è **n-bilanciato** sse

t1 e t2 sono **n-bilanciati**

e

$$|n1-n2| \leq 1.$$

# Verifica bilanciamento nel numero dei nodi.

```
int nBilIneff(TreePtr tPtr)
/* post: restituisce 1 se l'albero t e' bilanciato nel
numero di nodi, ossia se la differenza tra il numero dei
nodi nel sottoalbero sinistro e quello nel sottoalbero
destro di ogni nodo è minore di 1 in valore assoluto,
altrimenti restituisce 0. VERSIONE INEFFICIENTE */
{ int nnr, nnl;
  if (!tPtr) return 1;
  if (nBilIneff(tPtr->lPtr))
    if (nBilIneff(tPtr->rPtr))
      { nnl = numNodi(tPtr->lPtr);
        nnr = numNodi(tPtr->rPtr);
        if (abs(nnr - nnl) <= 1) return 1;
      }
  return 0;
}
```

# Verifica bilanciamento nel numero dei nodi.

```
int nBilEffAus(TreePtr tPtr)
/* Verifica il bilanciamento di tPtr con un unica
   scansione dell'albero. Ogni chiamata restituisce il
   numero dei nodi del sottoalbero radicato nel nodo
   della chiamata, se questo è bilanciato, mentre
   restituisce -1 altrimenti.
postc: restituisce -1 se l'albero non e' bilanciato nel
   numero dei nodi, un valore positivo altrimenti.*/
{ int nnl, nnr;
  if (!tPtr) return 0;
    else
      { nnl = nBilEffAus(tPtr->lPtr);
        if (nnl >= 0)
          { nnr = nBilEffAus(tPtr->rPtr);
            if (nnr >= 0 && abs(nnl-nnr) <= 1)
              return (nnl+nnr+1);
          }
        }
    }
  return -1;
}
```

# Verifica bilanciamento fine

```
int nBilEff(TreePtr tPtr)
/* verifica il bilanciamento nel numero di nodi
  di tPtr,
  *postc: restituisce 1 se l'albero è bilanciato, 0
  altrimenti
  */
{ if (nBilEffAus(tPtr) == -1) return 0;
  else return 1;
}
```

**Esercizio 2:** Si definisca la funzione C di prototipo

```
int hLSbil(TreePtr t);
```

che restituisce il numero dei nodi sbilanciati a sinistra in altezza, cioè dei nodi  $X$  tali che l'altezza del sottoalbero sinistro di  $X$  è maggiore di quella di quello destro.

```
int hSbil(TreePtr tPtr)
/*postc: restituisce il numero dei nodi sbilanciati a
  sinistra in altezza */
{int hsbil=0;
hSbilAus(tPtr,&hsbil);return hsbil;}

int hSbilAus(TreePtr tPtr,int *ris)
/*postc:restituisce altezza di t e calcola in ris il
  numero dei nodi sbilanciati a sinistra in altezza*/
{int lh,rh;
if (!tPtr) return -1; /* l'albero vuoto ha altezza -1*/
lh = hSbilAus(tPtr -> lPtr, ris);
rh = hSbilAus(tPtr -> rPtr,ris ) ;
if (lh > rh) {(*ris)++;return lh + 1;} else return
  rh+1;
}
```