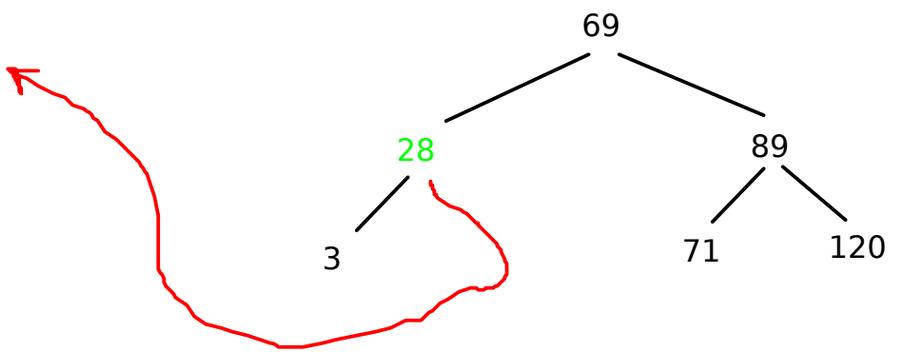
```
/****************************************
*  Inserimento
****************************************/

/* bist_insert: crea una foglia contenente item e la inserisce in ordine
nell'albero di radice *r. Se esiste un nodo con la stessa chiave, non inserisce
niente.
NB: la radice *r puo' venire modificata, quindi ne passo l'indirizzo (in
alternativa si potrebbe restituire la nuova radice, modificando il prototipo
della funzione) */

void bist_insert( Bit_node *r, Item item ) {
  Bit_node qf, q = *r, new = bit_new( item );
  Key k = key(item);

  if ( q == NULL ) {
    /* inserisco nell'albero vuoto */
    *r = new;
    return;
  }



  if ( bist_searchfather ( *r, k, &qf, &q ) == 0 ) {
    /* la chiave c'e' gia' , non inserisco niente */
    printf( "%d c'e' gia' \n" );
    return;
  }



  /* qf e' il padre del nuovo nodo */
  if ( cmp (k, key( qf -> item ) ) < 0 )
    qf -> l = new;
  else
    qf -> r = new;
}
```

Esempio: inserisco un Item con chiave 40



al termine di bist_searchfather: qf ---> 28

40 > 28

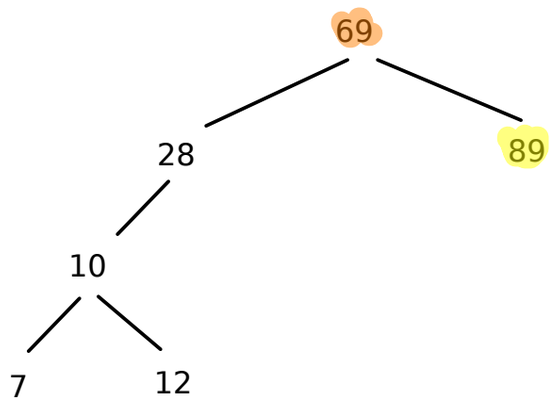
=> attacco figlio destro al nodo qf

```
/****************************************
*  Cancellazione
****************************************/
/* bist_delete: cancella il nodo con chiave k (chiamiamo x tale nodo) dell'albero di radice *r.
Restituisce -1 se non esiste nodo con chiave k.
Prima di cancellare x, viene scelto un nodo s che andra'  a sostituire x:
  CASO 1 - se x non ha figli, x e' sostituito dall'albero vuoto, quindi s vale NULL.
  CASO 2 - se x ha un unico figlio, s e' il figlio di x.
  CASO 3 - se x ha due figli, allora s e` il maggiore fra i nodi minori di x; questo
    significa che s e' il nodo piu' a destra nel sottoalbero sinistro di x.
NB: la radice *r puo' venire modificata, quindi ne passo l'indirizzo (in
alternativa si potrebbe restituire la nuova radice, modificando il prototipo
della funzione) */

int bist_delete( Bit_node *r, Key k ) {
  Bit_node x, xf, s = NULL;
```

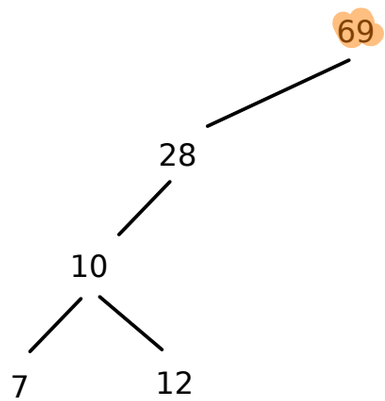
Cancellazione dell'item con chiave 89

CASO 1:

prima



dopo

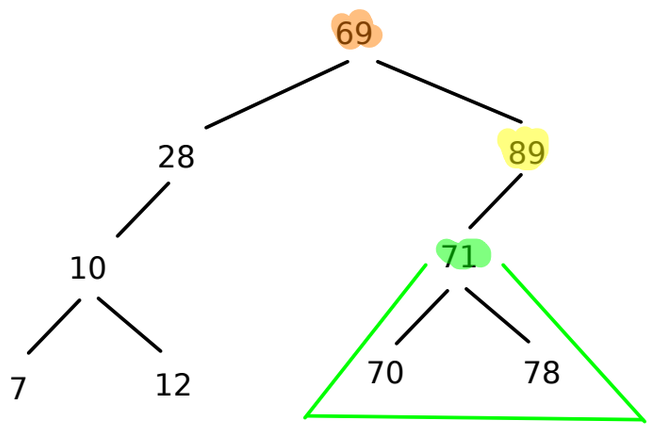


```
/****************************************
*  Cancellazione
****************************************/
/* bist_delete: cancella il nodo con chiave k (chiamiamo x tale nodo) dell'albero di radice *r.
Restituisce -1 se non esiste nodo con chiave k.
Prima di cancellare x, viene scelto un nodo s che andra'  a sostituire x:
  CASO 1 - se x non ha figli, x e' sostituito dall'albero vuoto, quindi s vale NULL.
  CASO 2 - se x ha un unico figlio, s e' il figlio di x.
  CASO 3 - se x ha due figli, allora s e` il maggiore fra i nodi minori di x; questo
    significa che s e' il nodo piu' a destra nel sottoalbero sinistro di x.
NB: la radice *r puo' venire modificata, quindi ne passo l'indirizzo (in
alternativa si potrebbe restituire la nuova radice, modificando il prototipo
della funzione) */

int bist_delete( Bit_node *r, Key k ) {
  Bit_node x, xf, s = NULL;
```

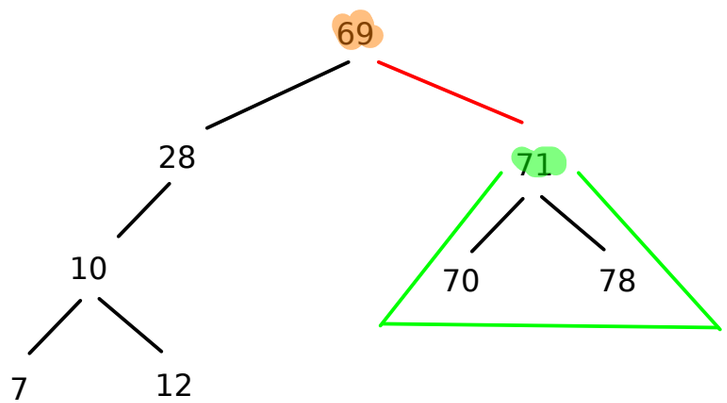
Cancellazione dell'item con chiave 89

CASO 2:

prima



dopo



```
/****************************************
*  Cancellazione
****************************************/
/* bist_delete: cancella il nodo con chiave k (chiamiamo x tale nodo) dell'albero di radice *r.
Restituisce -1 se non esiste nodo con chiave k.
Prima di cancellare x, viene scelto un nodo s che andra'  a sostituire x:
  CASO 1 - se x non ha figli, x e' sostituito dall'albero vuoto, quindi s vale NULL.
  CASO 2 - se x ha un unico figlio, s e' il figlio di x.
  CASO 3 - se x ha due figli, allora s e' il maggiore fra i nodi minori di x; questo
    significa che s e' il nodo piu' a destra nel sottoalbero sinistro di x.
NB: la radice *r puo' venire modificata, quindi ne passo l'indirizzo (in
alternativa si potrebbe restituire la nuova radice, modificando il prototipo
della funzione) */

int bist_delete( Bit_node *r, Key k ) {
  Bit_node x, xf, s = NULL;
```

Cancellazione dell'item con chiave 89

CASO 3: