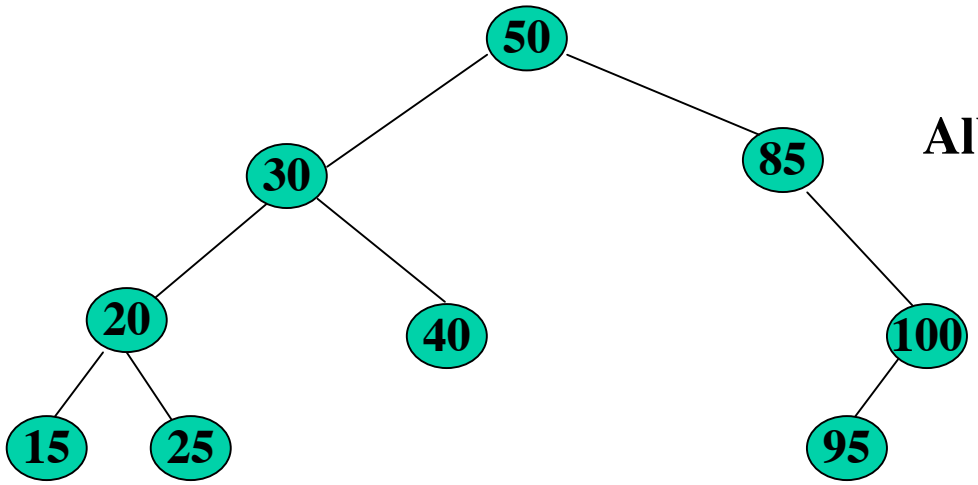
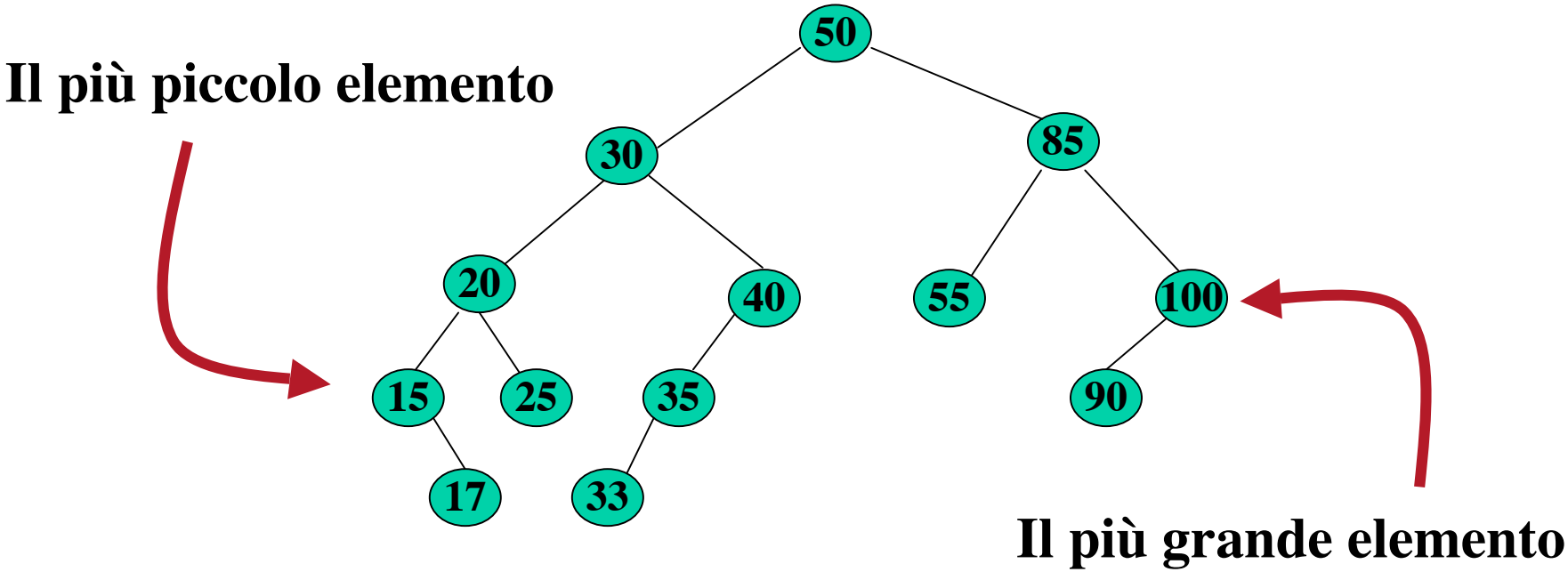


Albero binario di ricerca

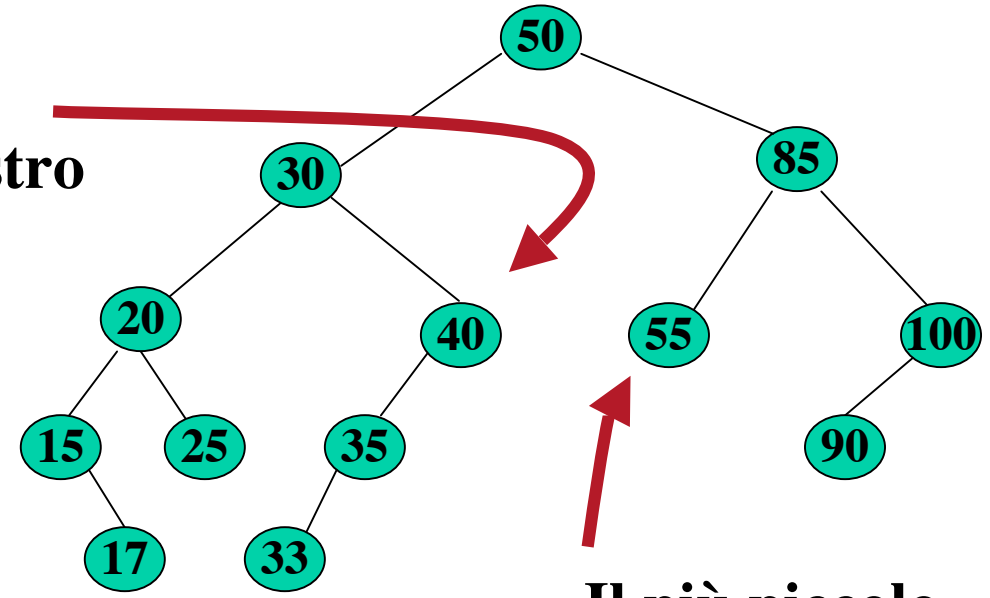
Un albero binario di ricerca é un albero binario in cui ogni nodo ha un'etichetta minore o uguale a quelle dei nodi nel sottoalbero radicato nel figlio destro e maggiore o uguale a quella dei nodi nel sottoalbero radicato nel figlio sinistro

Nell'esempio abbiamo usato gli interi, ma si può utilizzare per le etichette un qualsiasi insieme totalmente ordinato (per esempio caratteri o stringhe).

Per semplicità eliminiamo le ripetizioni nell'albero.



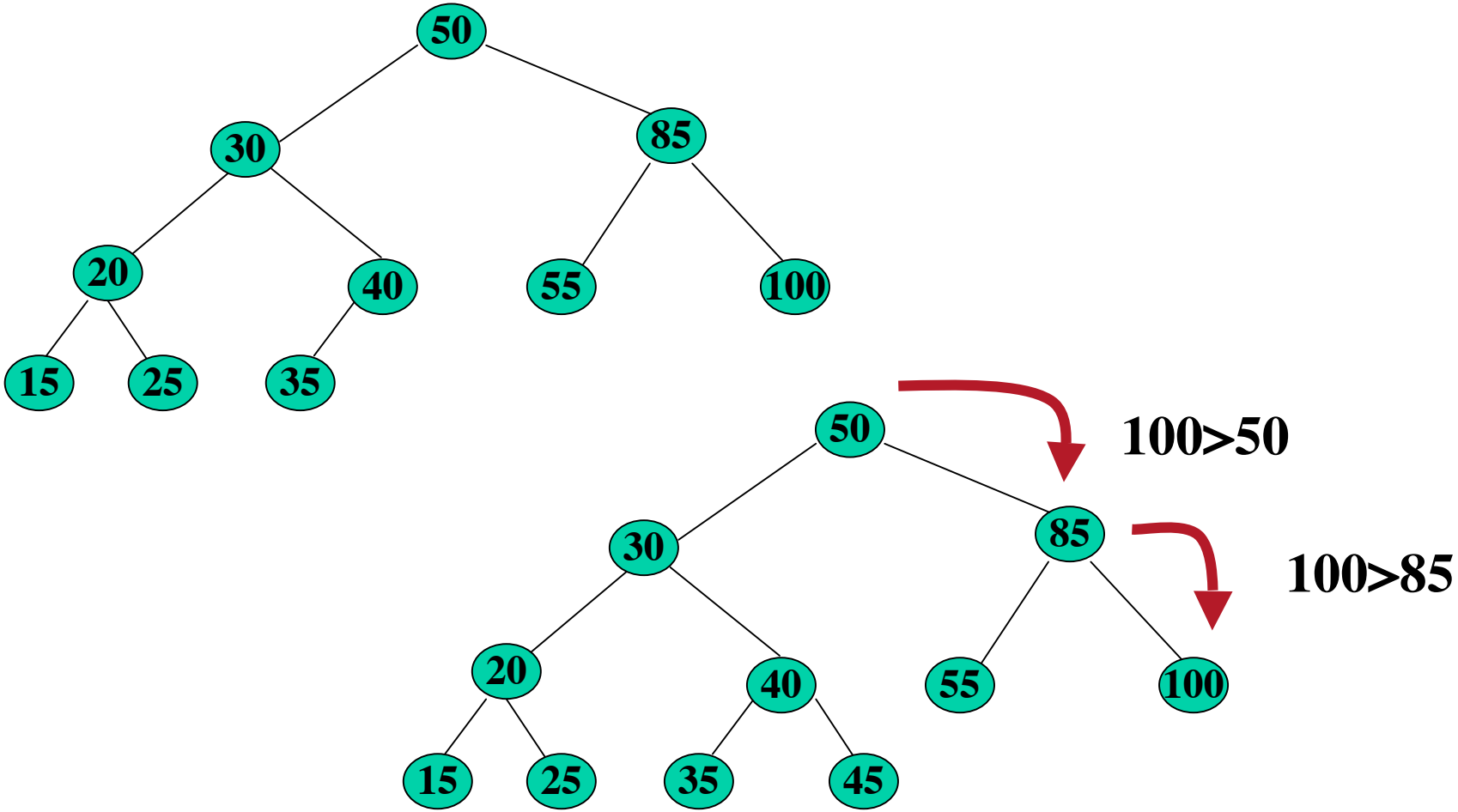
Il più grande elemento nel sottoalbero sinistro



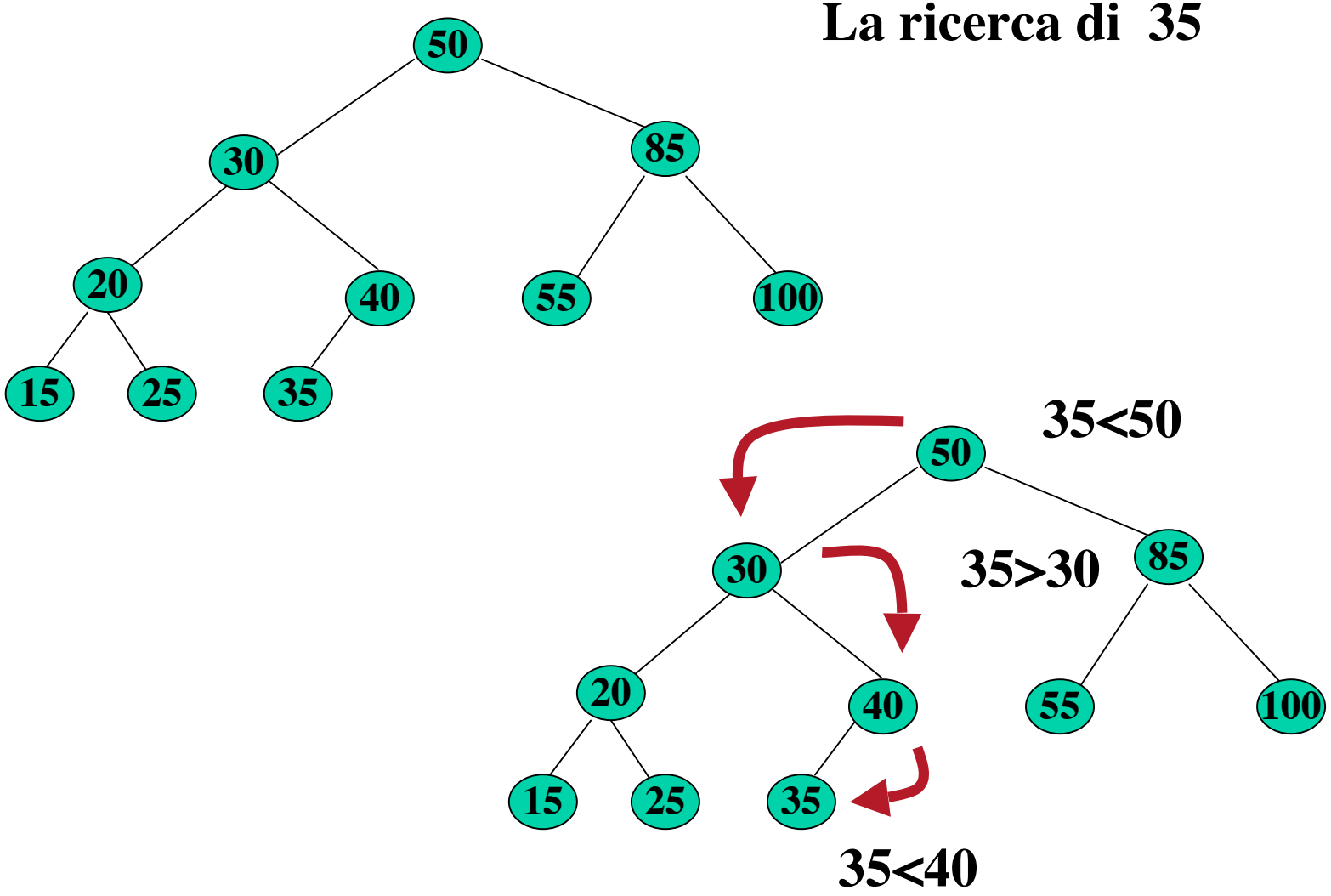
Il più piccolo elemento nel sottoalbero destro

La ricerca di un elemento in una albero binario di ricerca

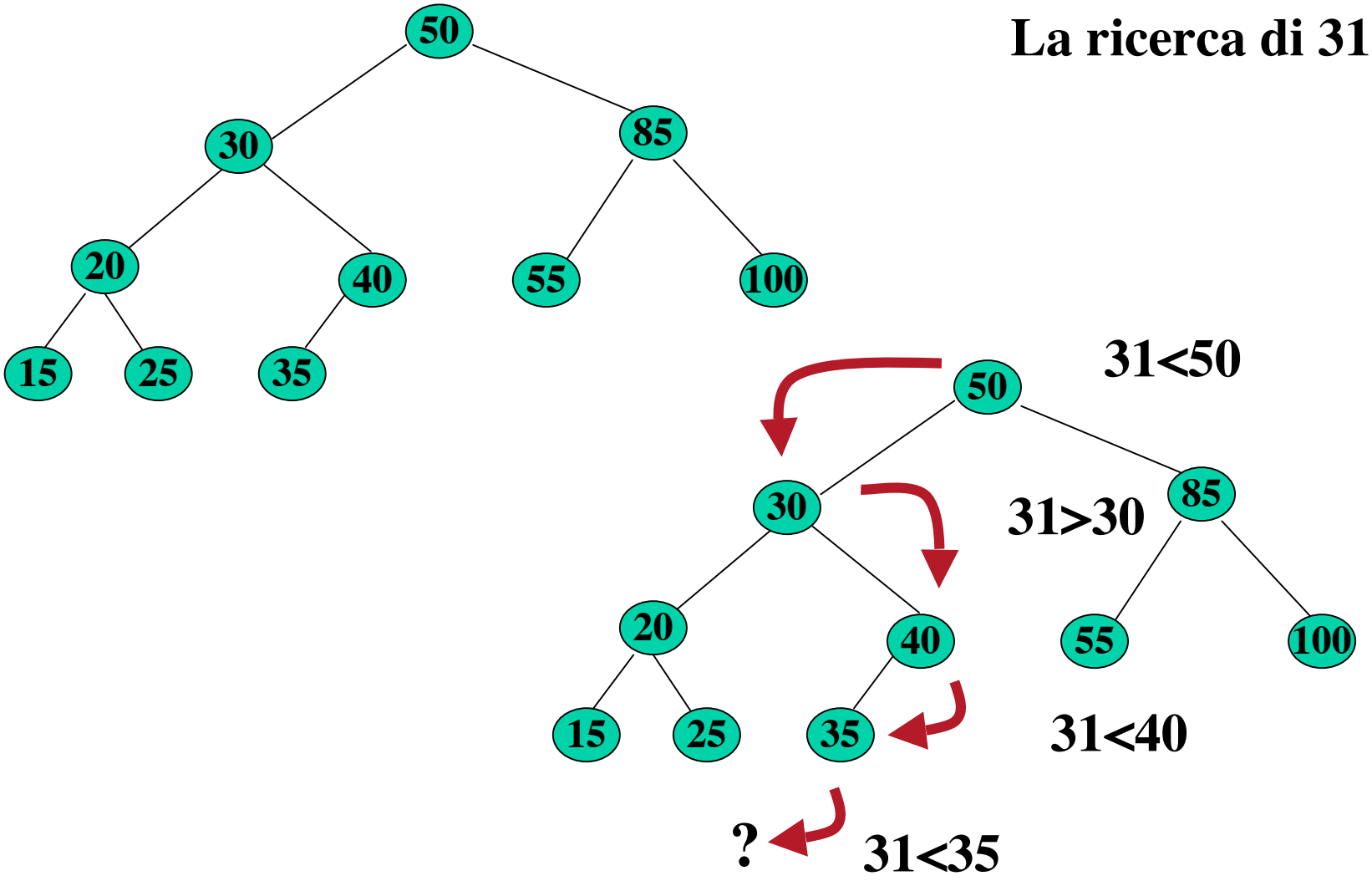
La ricerca di 100



La ricerca di un elemento in una albero binario di ricerca

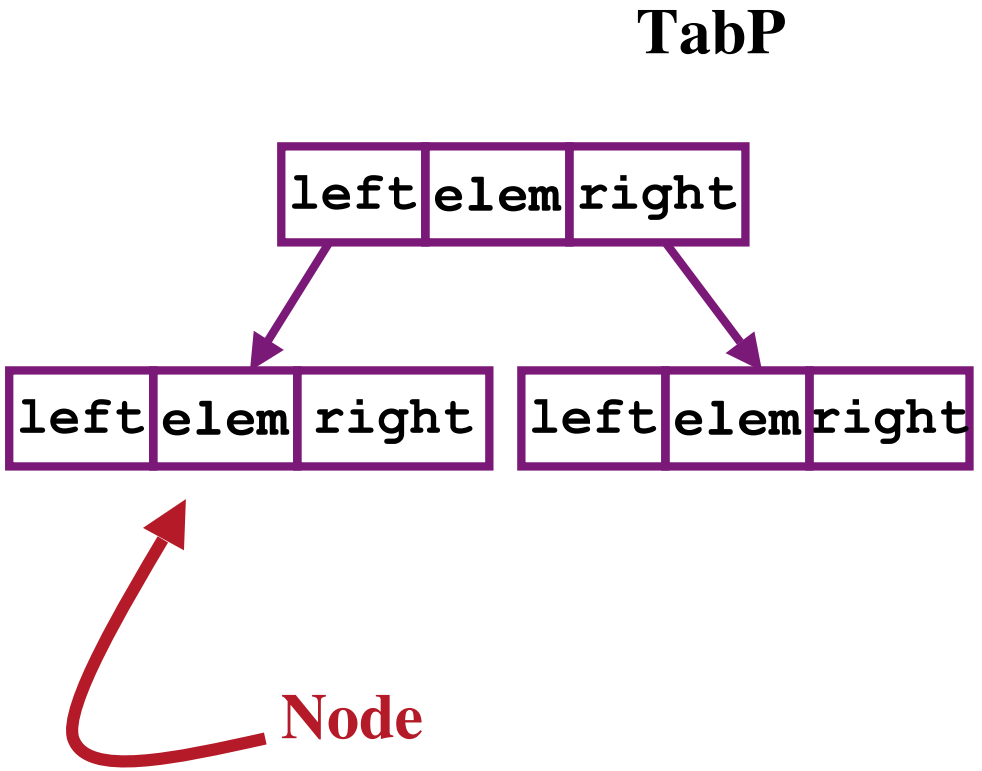


La ricerca di un elemento in una albero binario di ricerca



Implementazione ricerca in un albero binario di ricerca

- **Struttura dati**



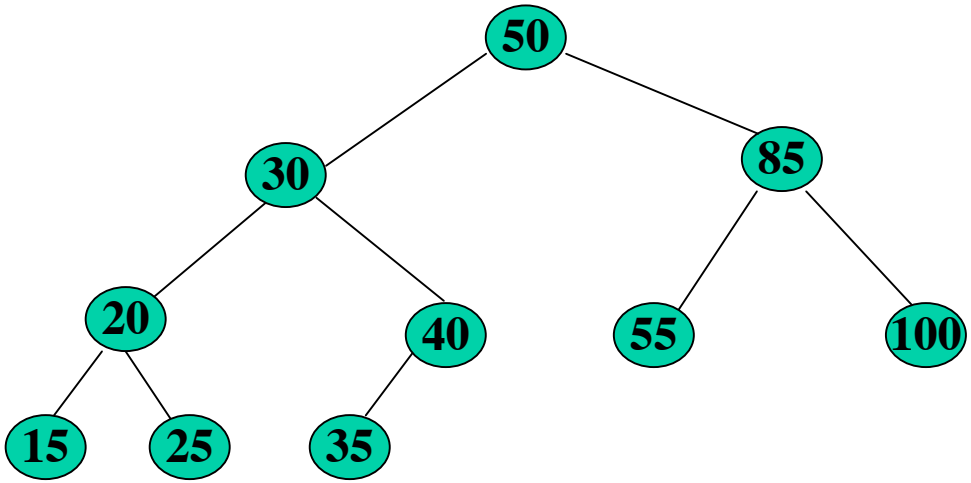

```
struct node {
  int  elem;
  struct node *left;
  struct node *right;
    };

typedef struct node Node;
typedef Node *TabP;
```

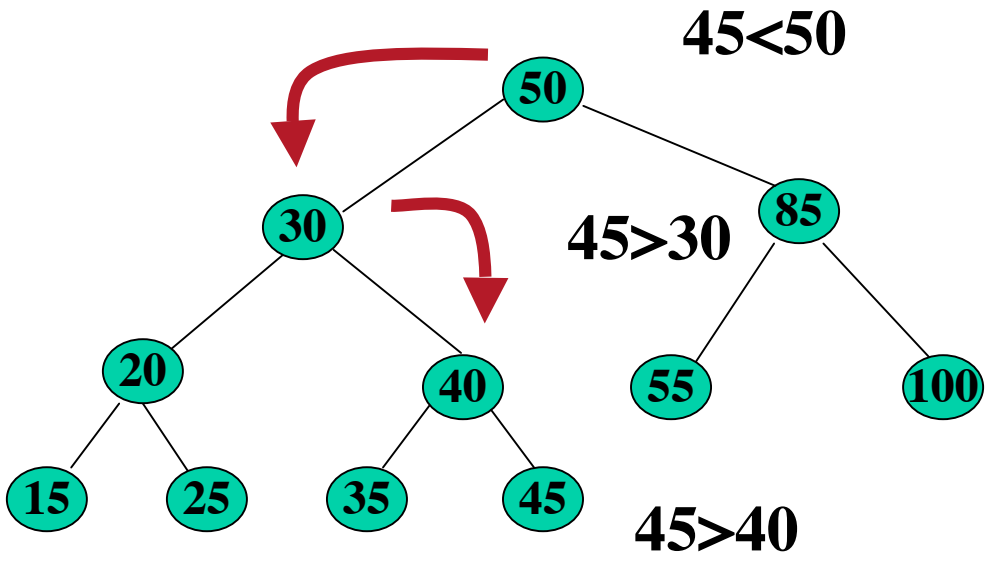
Implementazione ricerca in un albero binario di ricerca

```
TabP cerca(TabP t, int key)
/*postc: restituisce un puntatore a key  in t
*se presente, NULL altrimenti (anche quando la *collezione è
vuota) */
{if (t)
  {if (key == t->elem )  return t;
     else
       if (key < t->elem)
/* Più piccolo, cerca a sinistra */
     return cerca(t->left, key);
          else   /* Più grande, cerca a destra */
     return cerca(t->right, key);
  }
else
return NULL;
}
```

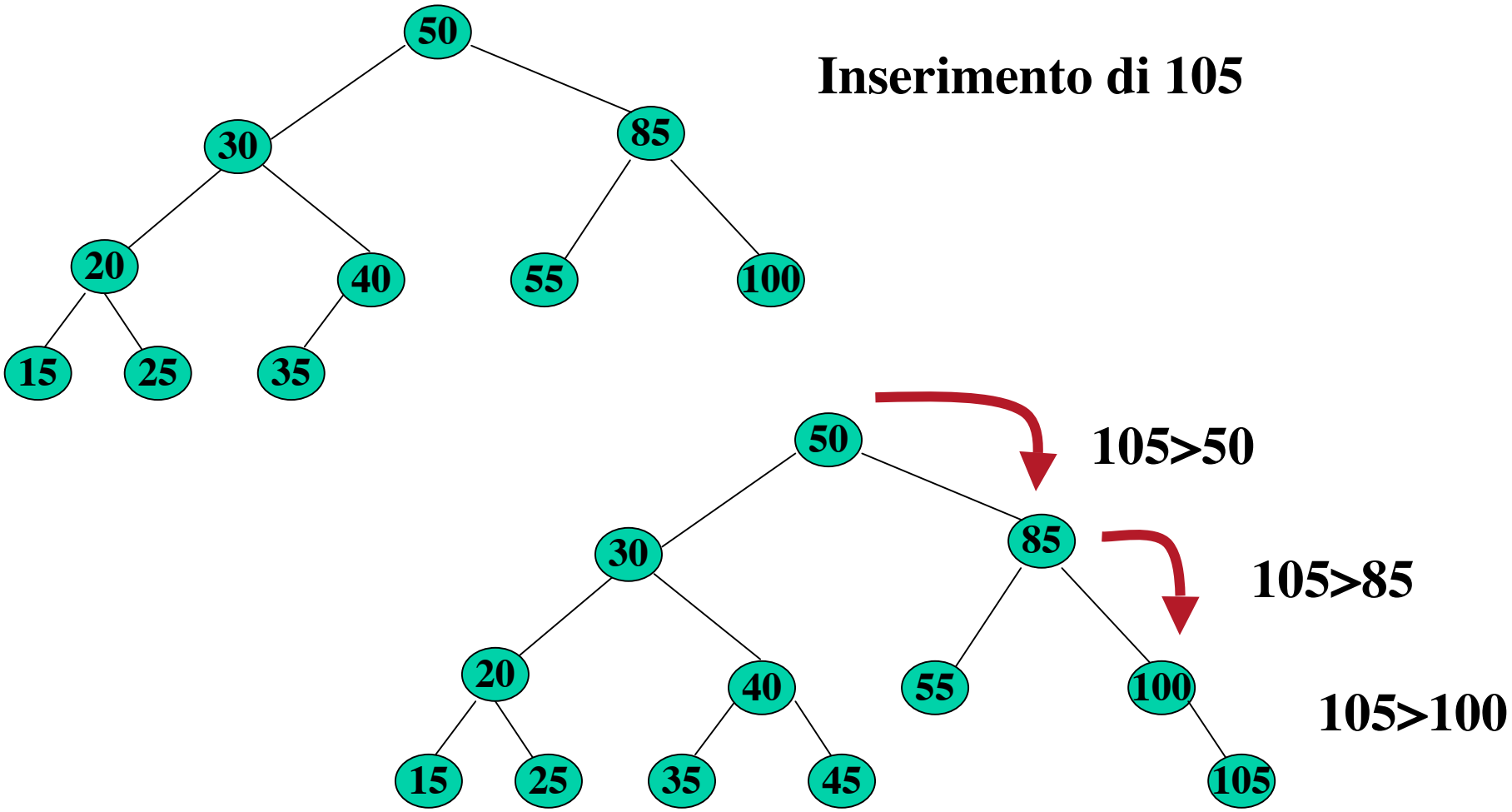
L’inserimento di un elemento in una albero binario di ricerca



Inserimento di 45



L'inserimento di un elemento in una albero binario di ricerca

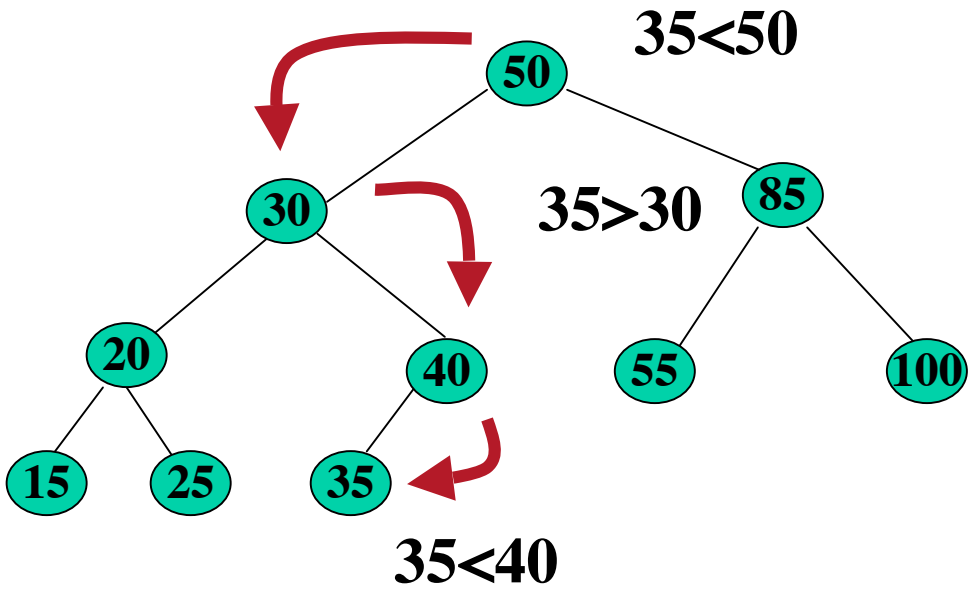


Implementazione inserimento in un albero binario di ricerca

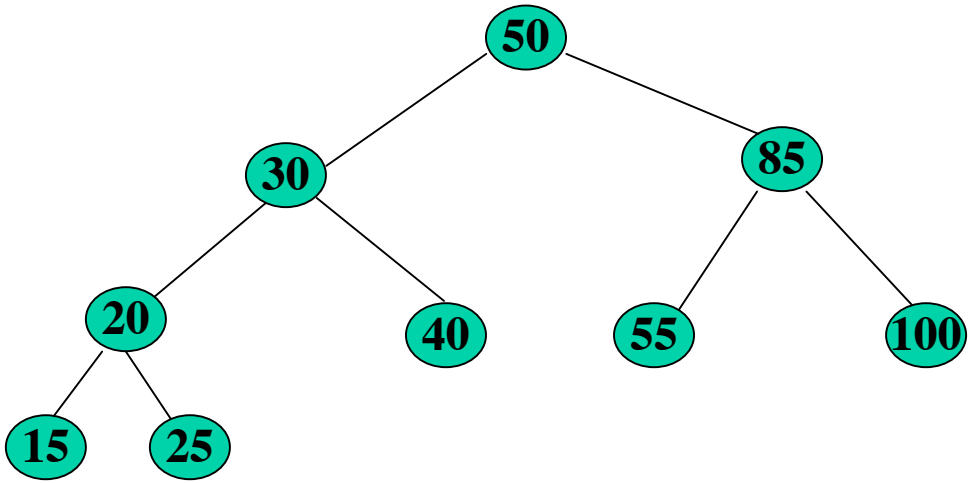
```
TabP addElem( TabP t, TabP new )
/* "versione funzionale". Aggiunge new alla collezione t, se non gia' presente
 *postc: restituisce t con l'aggiunta di el, se non gia' presente, la collezione immutata altrimenti */

{ if (!t) return new;
/* Se l'albero è vuoto il nuovo nodo e' la radice */
if(new->elem <t->elem)
/* Più piccolo, inserimento a sinistra */
        t->left = addElem(t->left,new);
else
        if(new->elem > t->elem )
/* Più grande, inserimento a destra */
                t->right = addElem( t->right, new);
        else return t;
return t;
 }
```

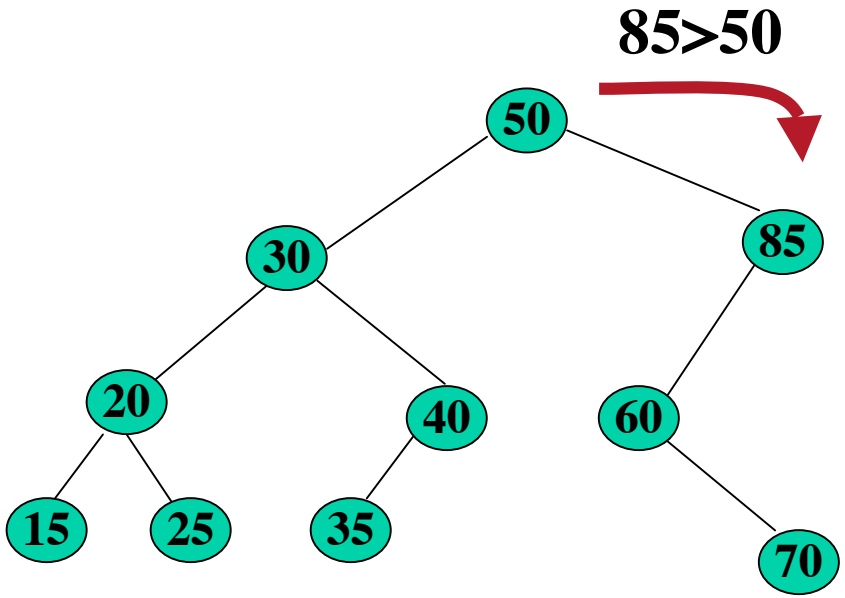
Implementazione cancellazione in un albero binario di ricerca



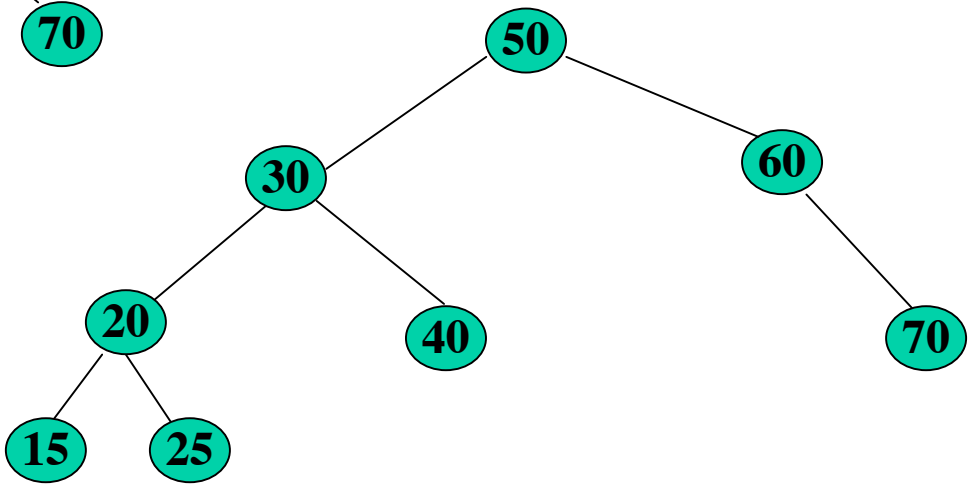
Cancellazione di 35



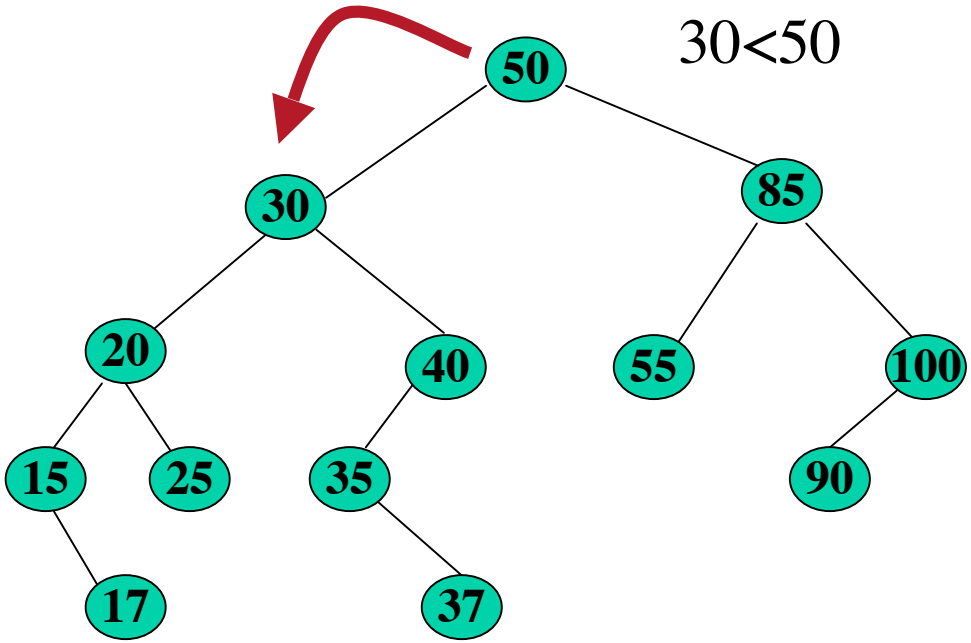
Implementazione cancellazione in un albero binario di ricerca



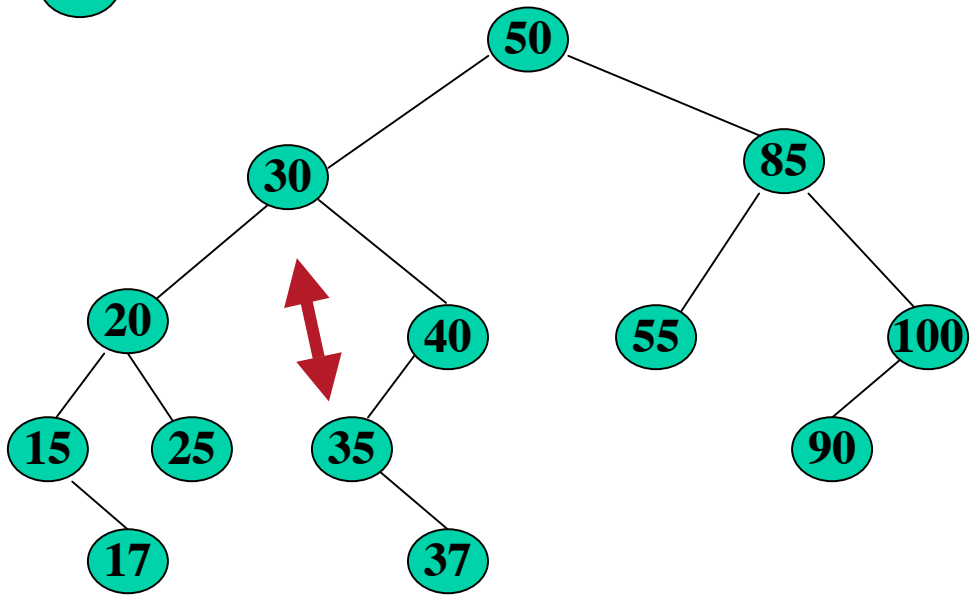
Cancellazione di 85



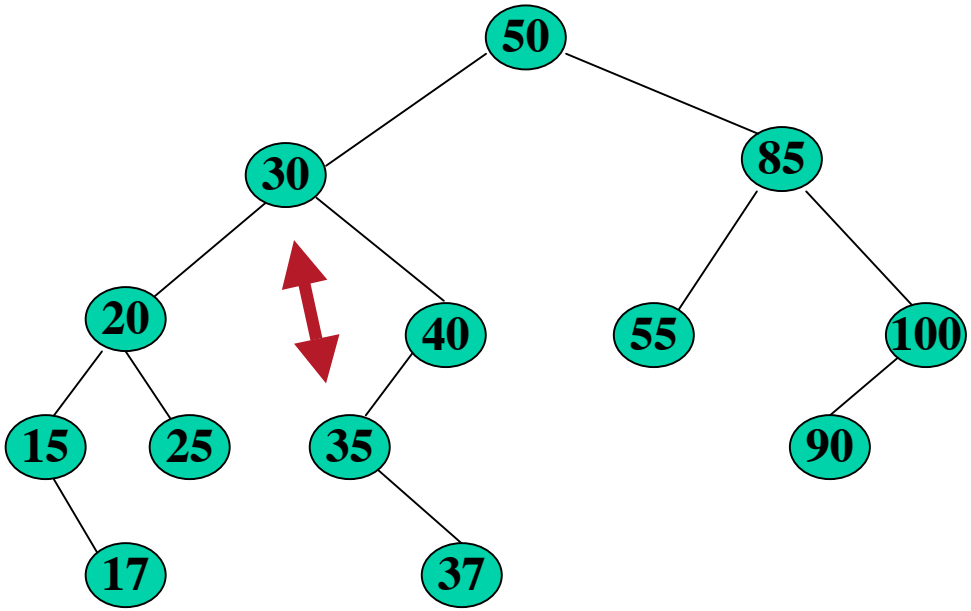
Implementazione cancellazione in un albero binario di ricerca



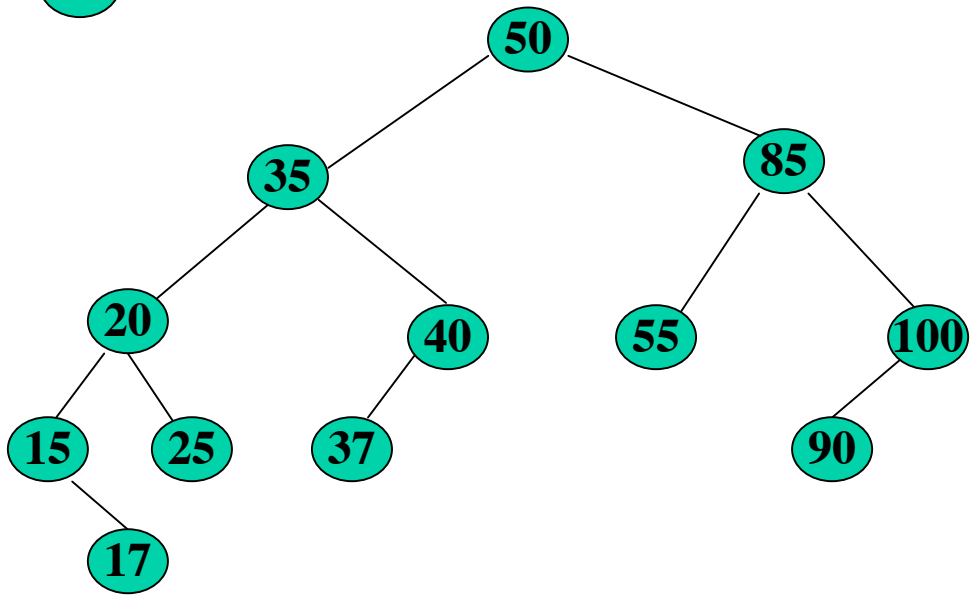
Cancellazione di 30



Implementazione cancellazione in un albero binario di ricerca



Cancellazione di 30



Implementazione cancellazione in un albero binario di ricerca

```
 int delMin( TabP *t )
/* cancella il nodo minimo in t e ne restituisce il valore
*prec (t != NULL && *t != NULL )
postc: restituisce il minimo valore  e  cancella quel nodo da *t */
{int app;
 TabP temp;
assert(t);
assert(*t);
if ((*t) -> left == NULL) /* ho trovato il  minimo */
   {app = (*t) ->elem;
    temp = *t;
    *t = (*t) -> right;
    free(temp);
    return app;}
else
return delMin( &((*t) ->left) );
}
```

```
int remEl( TabP *t,int val)
/* cancella la prima occorrenza di val in t
*prec: (t != NULL) && (*t != NULL)
postc: restituisce 1 se ha trovato e  cancellato l'elemento, 0 altrimenti */
{TabP  temp;
assert(t!=NULL);
 assert(*t!=NULL);
if (val==(*t) ->elem)
  {if ((*t)-> left == NULL) /* non ha il figlio sin*/
   {temp = *t; (*t) = (*t) ->right; free(temp);}
  else if ((*t) -> right == NULL) /* non ha il figlio des*/
           {temp = *t;(*t)= (*t) ->left;free(temp);}
         else /* ha due figli */
            (*t)-> elem = delMin(&((*t) ->right));
            return 1;}
if (val<(*t) ->elem)              /* cerco a sinistra */
    if ((*t) -> left != NULL)
             return remEl(&(*t) -> left,val);
    else return 0;                              /* (*t) -> left == NULL*/
else if((*t) -> right!= NULL)
       return remEl(&(*t) -> right,val);
       else return 0;             /* (*t) -> left == NULL*/
}
```