In questa lezione

Alberi binari di ricerca

[CLRS] cap.12

Dizionari

Un dizionario è una struttura dati costituita da un insieme con le operazioni di inserimento, cancellazione e verifica di appartenenza di un elemento.

I dati sono complessi, ma dotati di una chiave di identificazione. Assumendo che le chiavi siano diverse, otteniamo un insieme. Gli altri elementi del dato sono chiamati dati satellite. Spesso le chiavi sono prese in un insieme ordinato (per esempio chiavi intere)

Applicazioni Dizionari

applicazione	obiettivo	chiave	valore
elenco telefono	cercare un numero	nome	numero telefonico
bancaria	eseguire una transazione	numero di c.c.	dettagli transazione
condivisione file	trovare una canzone da scaricare	nome della canzone	l'ID di un calcolatore
file system	trovare un file sul disco	nome del file	la posizione del file
compilatore	trovare le proprietà di una varabile	nome della variabile	valore e tipo

Dizionari
Implementazioni elementari

Insieme di n elementi - caso peggiore

implementazione	insert	search
array	Θ(1)	Θ(n)
array ordinato	Θ(n)	Θ(lg n)
lista concatenata	Θ(1)	Θ(n)
lista concatenata ordinata	Θ(n)	Θ(n)

Si può fare meglio?

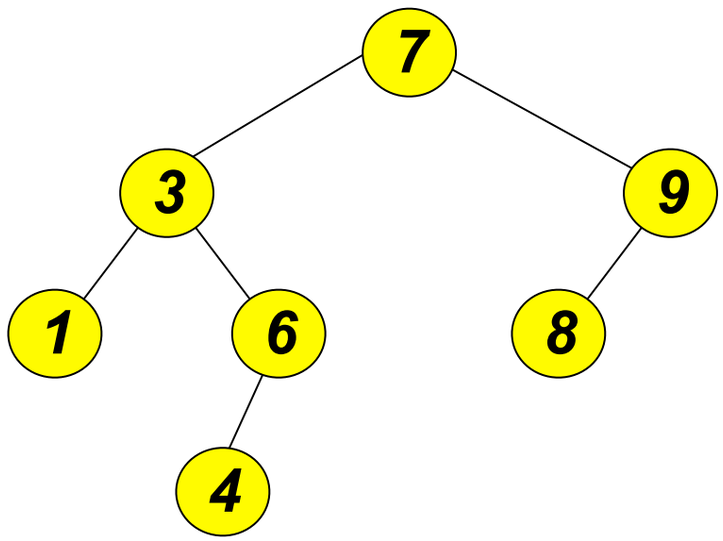
Alberi binari di ricerca

Un ABR è un albero binario in cui per ogni nodo v

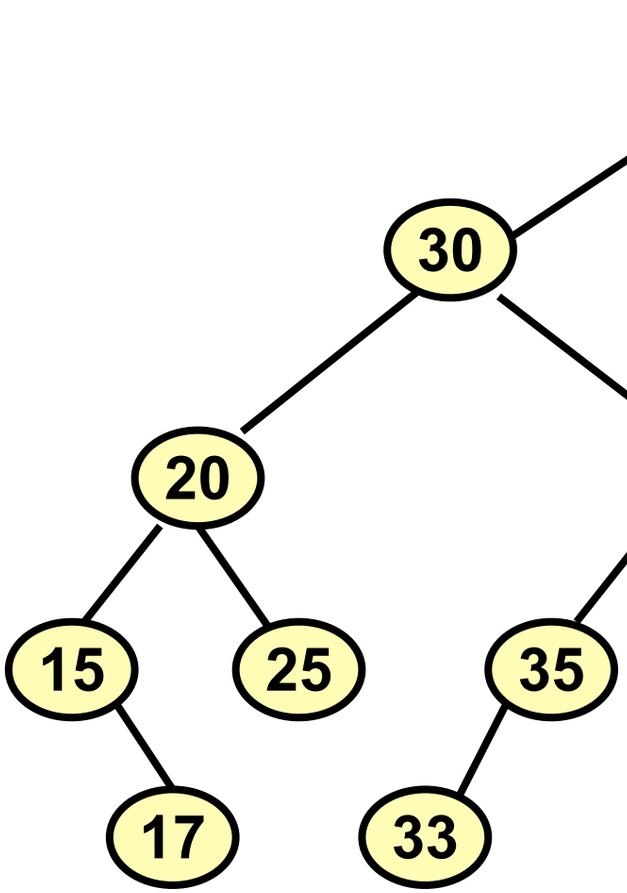
- **le chiavi dei nodi nel sottoalbero sinistro di v sono minori o uguali alla chiave di v e**
- **le chiavi dei nodi nel sottoalbero destro di v sono maggiori o uguali alla chiave di v.**

Esempio:

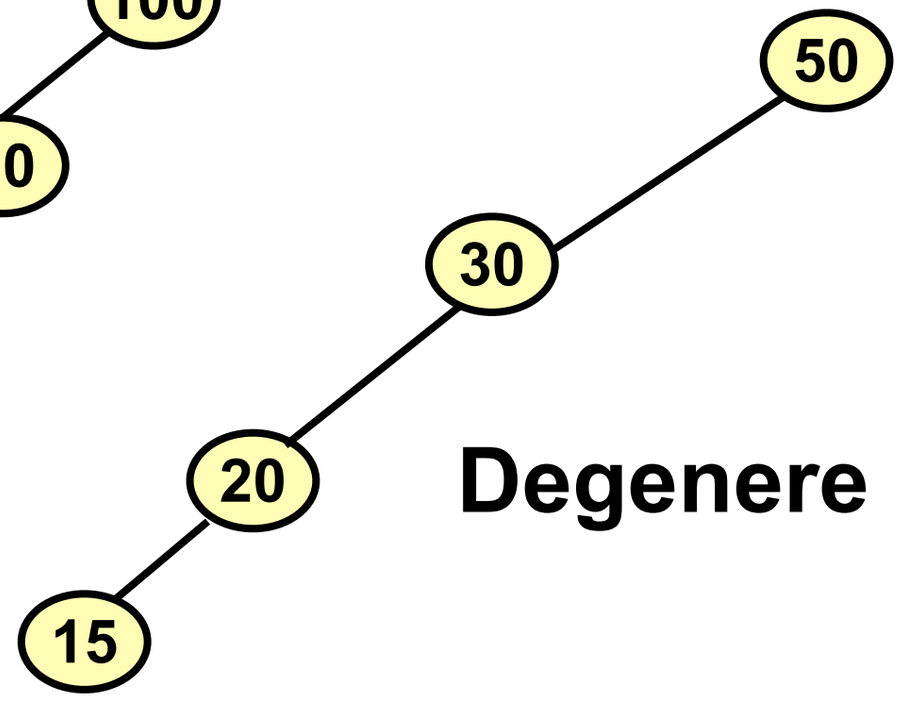
Nel seguito supponiamo che le chiavi siano tutte distinte



ABR: esempi



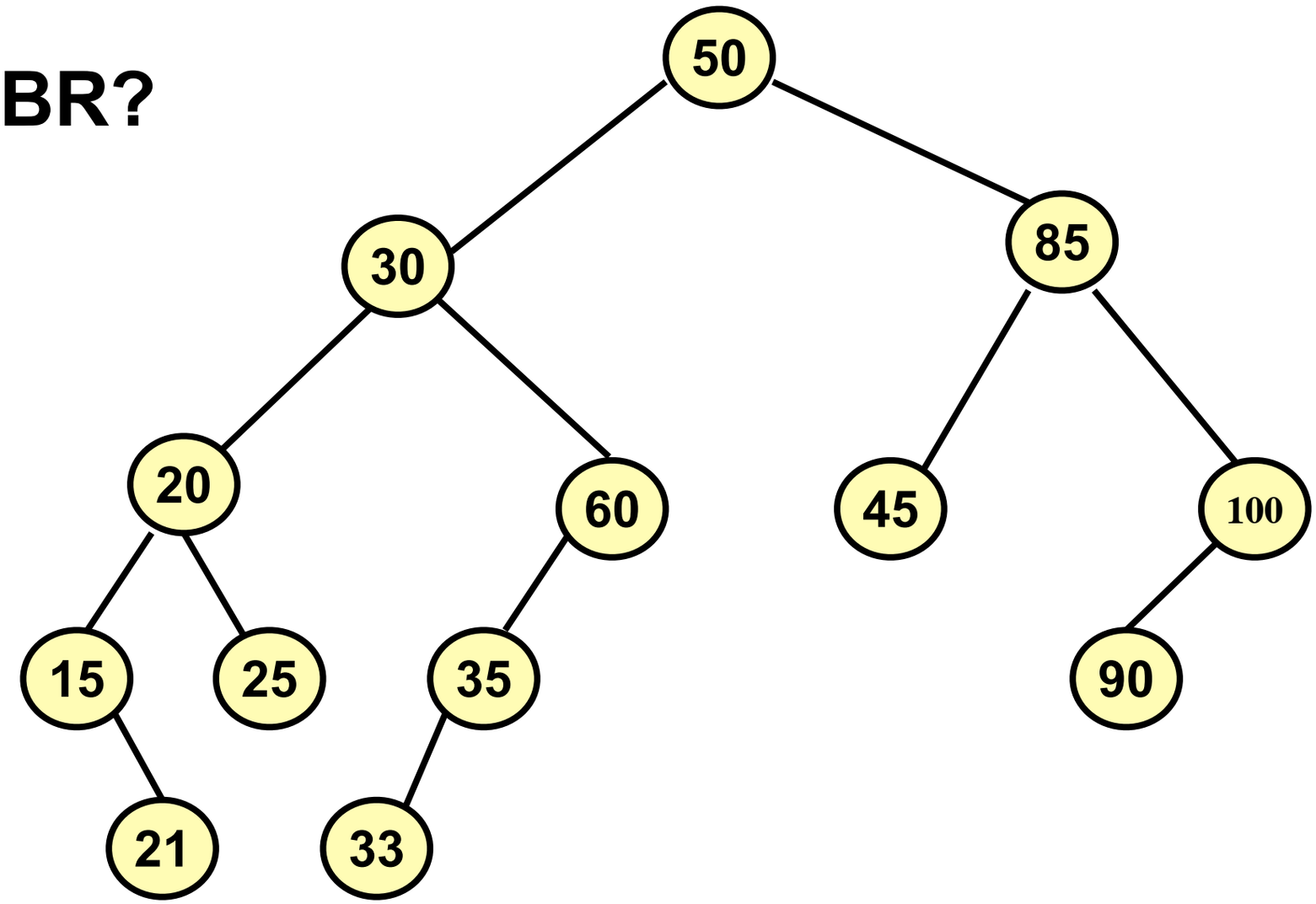
Tipico



Degenere

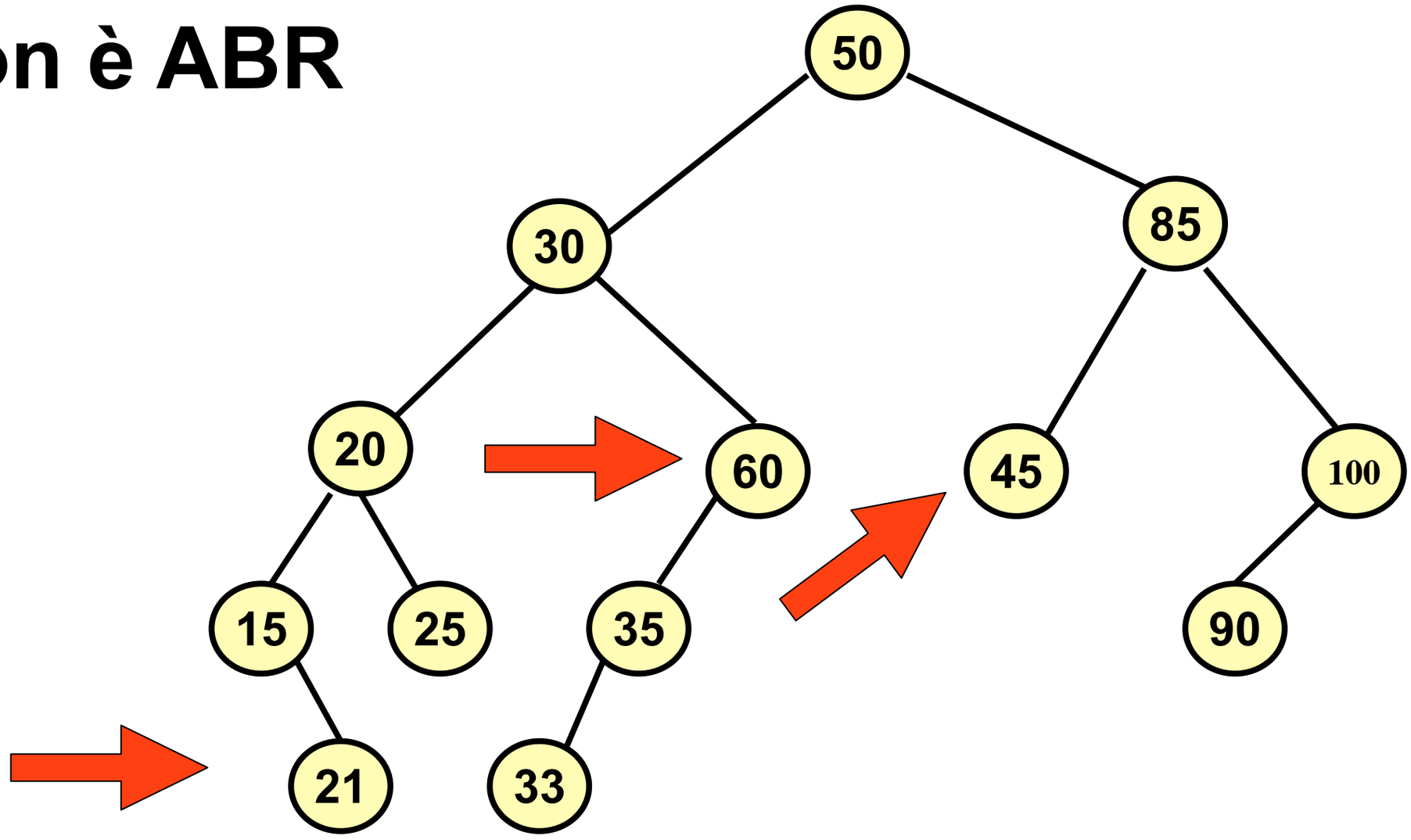
ABR: esempi

E’ un ABR?



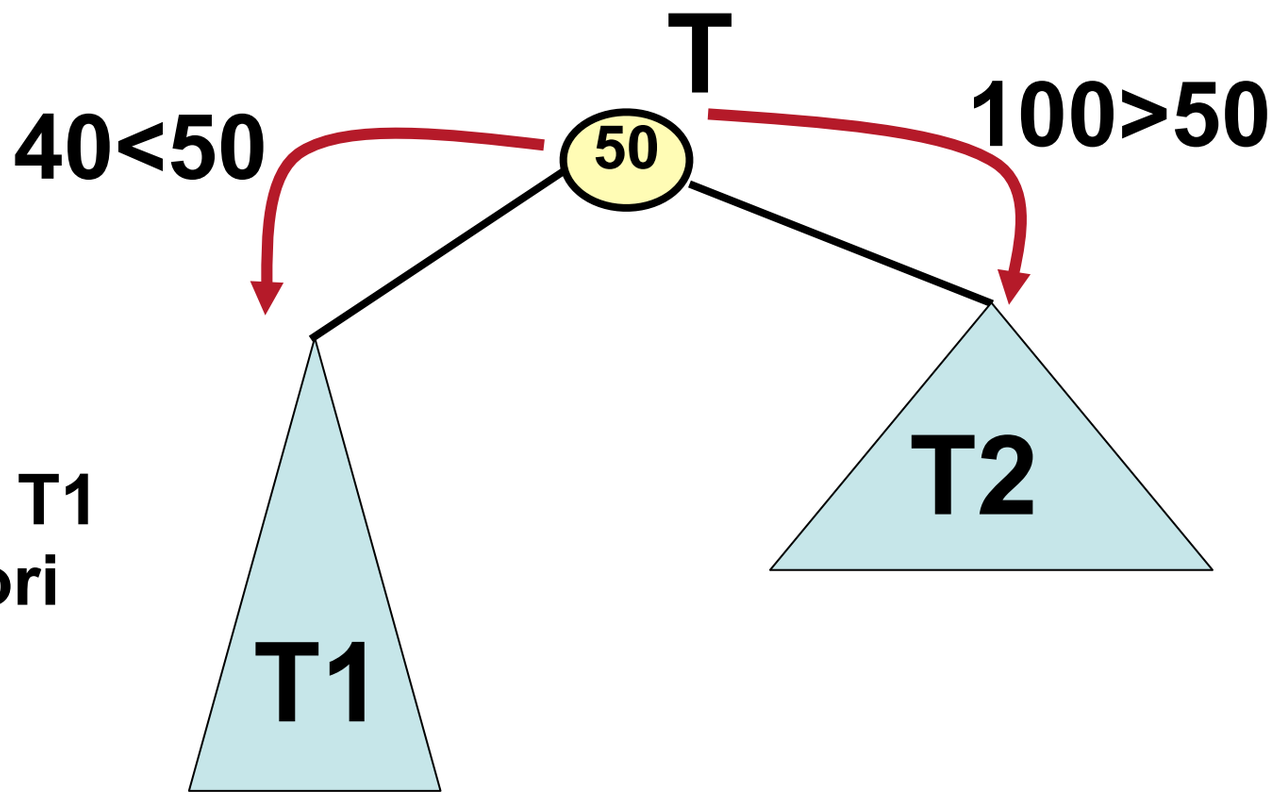
ABR: esempi

non è ABR



soddisfa la proprietà che per ogni nodo il figlio sinistro è minore del padre e il destro maggiore del padre:

La definizione consente una ricerca veloce!



Tutti nel sotto albero sinistro T1 di T sono minori di 50!

Tutti nel sotto albero destro T2 di T sono maggiori di 50!

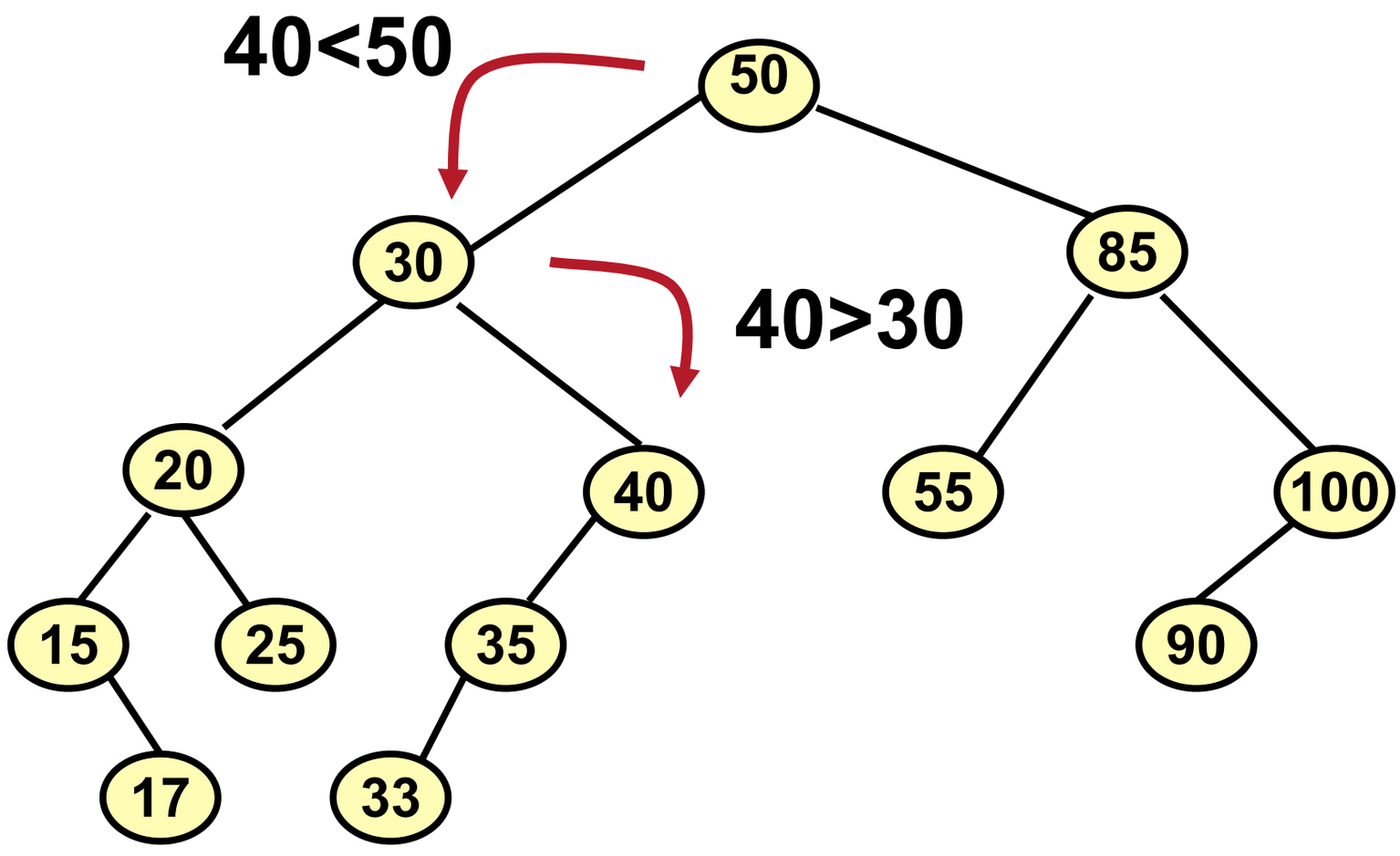
La ricerca di 40 in T deve proseguire nel sotto albero sinistro T1

La ricerca di 100 in T deve proseguire nel sotto albero destro T2

Ricorsivamente, in funzione del confronto, la ricerca deve proseguire in uno dei due sottoalberi.

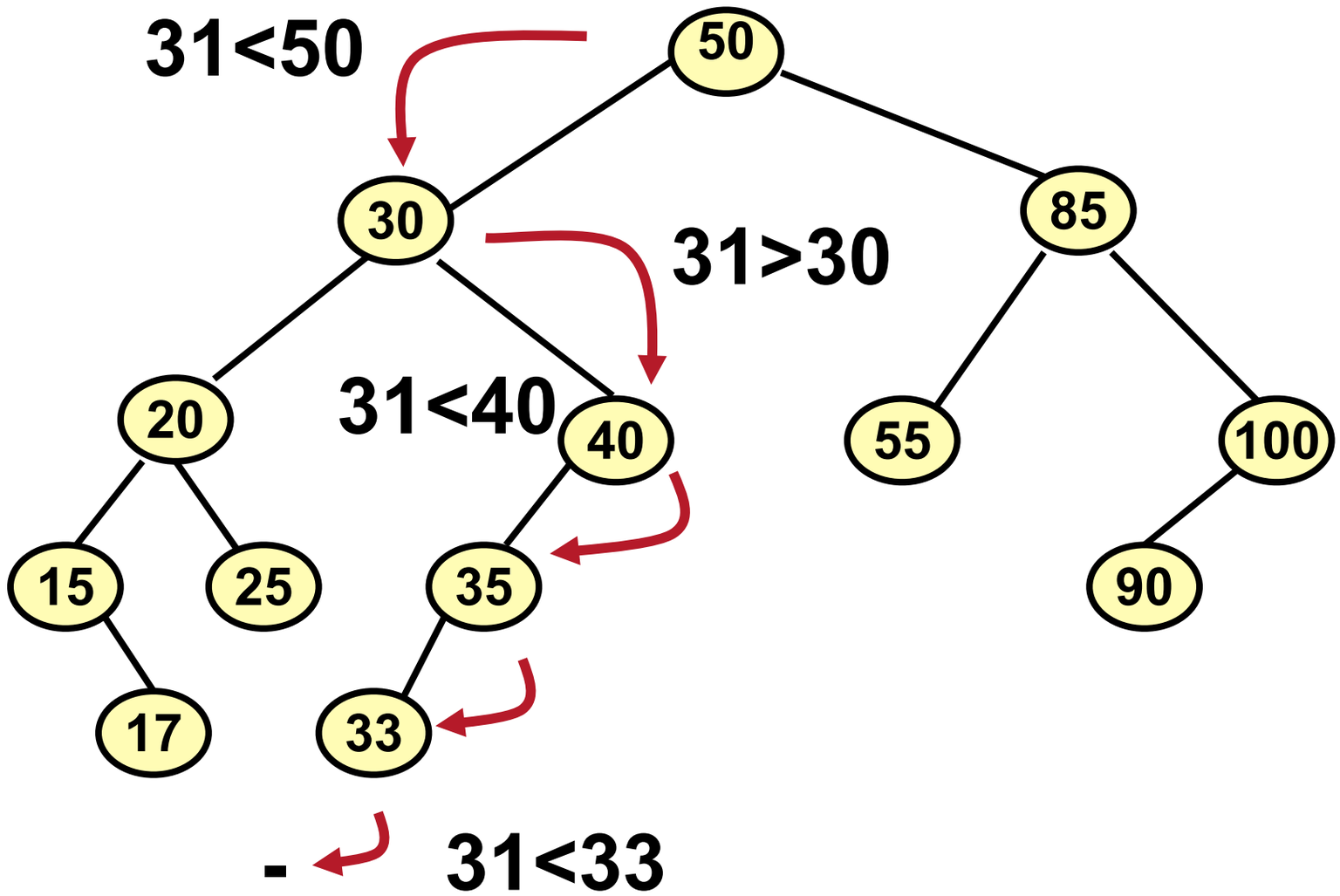
La ricerca con successo

La ricerca di 40 in



La ricerca con insuccesso

La ricerca di 31 in



L'algoritmo per la ricerca

Tree-Search(x, k)

Input: un puntatore x e una chiave k

prec: x è un ABR

output: il puntatore (riferimento) al nodo di chiave k, se presente, nil altrimenti

```
  if x == NIL or k == x.key then
    return x
if k < x.key then
    return Tree-Search(x.left, k)
else
    return Tree-Search(x.right, k)
```

la ricerca: analisi

Detta h l'altezza di un ABR t il tempo di esecuzione nel caso peggiore si ricava dalla seguente relazione di ricorrenza

$$T(h) = T(h-1) + \Theta(1)$$

La cui soluzione è Θ(h).

In generale NON è O(lg n)!!

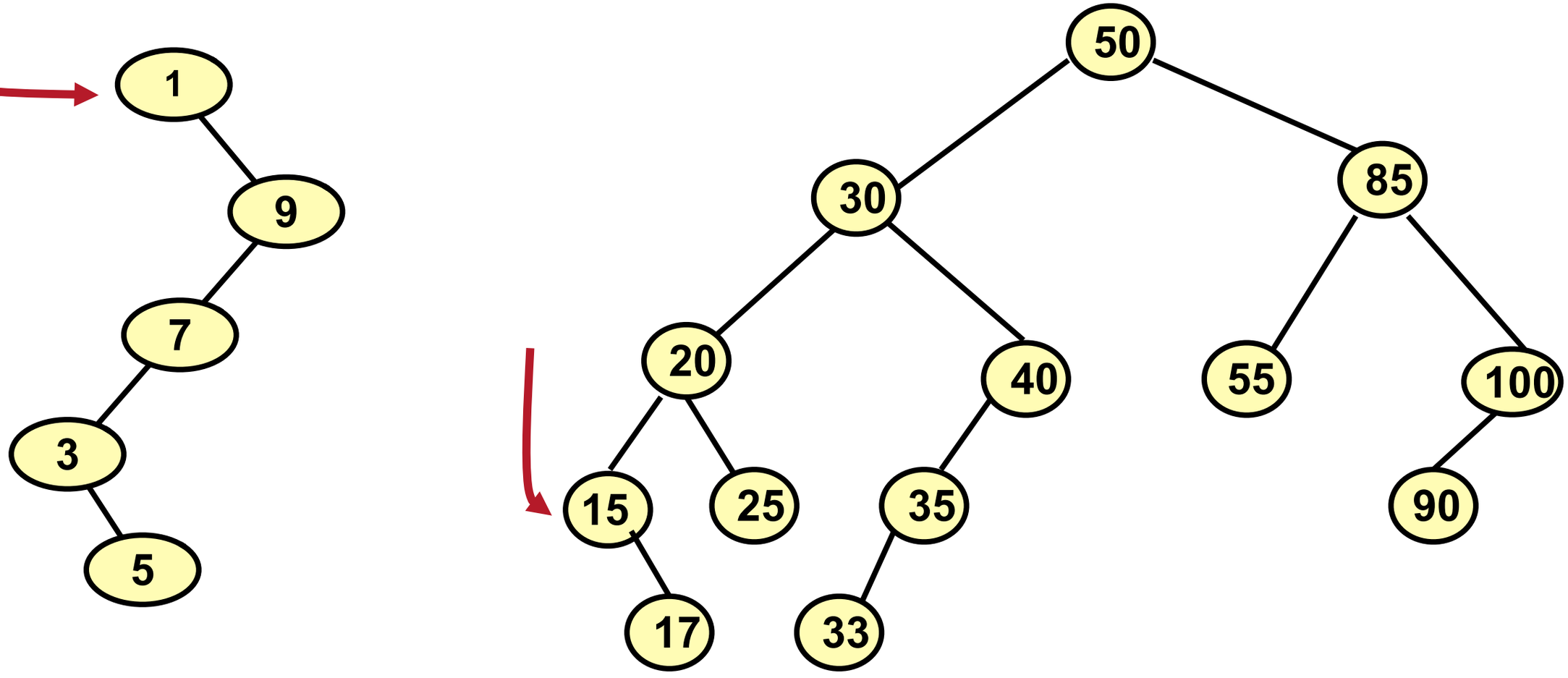
Ricordiamo che se n è il numero degli elementi di t vale
$$\lg n \leq h \leq n - 1$$

Quindi tempo di esecuzione nel caso peggiore potrebbe essere O(n)!
Ma è più informativo dire Θ(h) nel caso peggiore o O(h) in tutti i casi, per due motivi:

1. **si sa che $h \leq n - 1$, quindi questa possibilità è presa in conto quando si dice O(h) o Θ(h) nel caso peggiore**
2. **Si potrebbe pensare che sia un O(n) perché tutti i nodi son visitati, ma questo è vero solo nel caso degenere.**

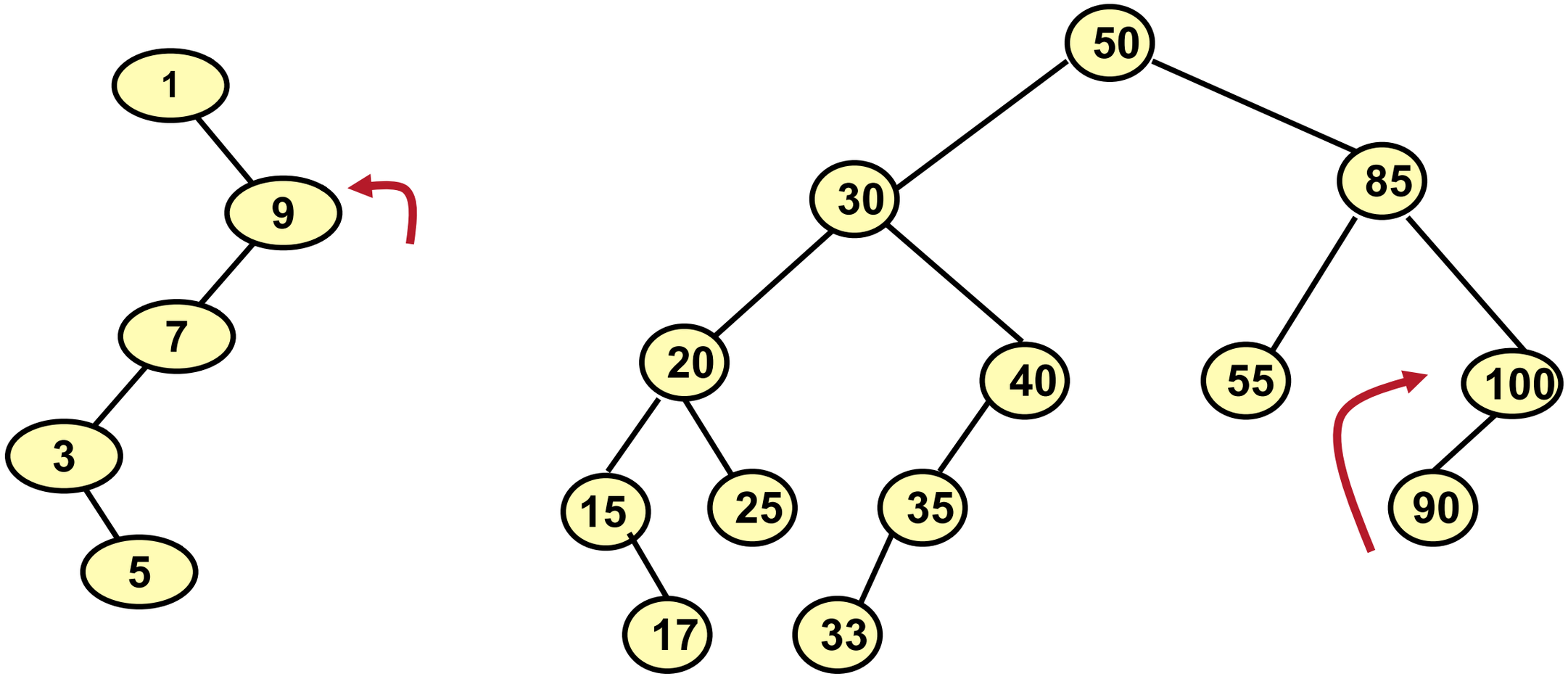
ABR: minimo e massimo

Il più piccolo elemento



ABR: minimo e massimo

Il più grande elemento



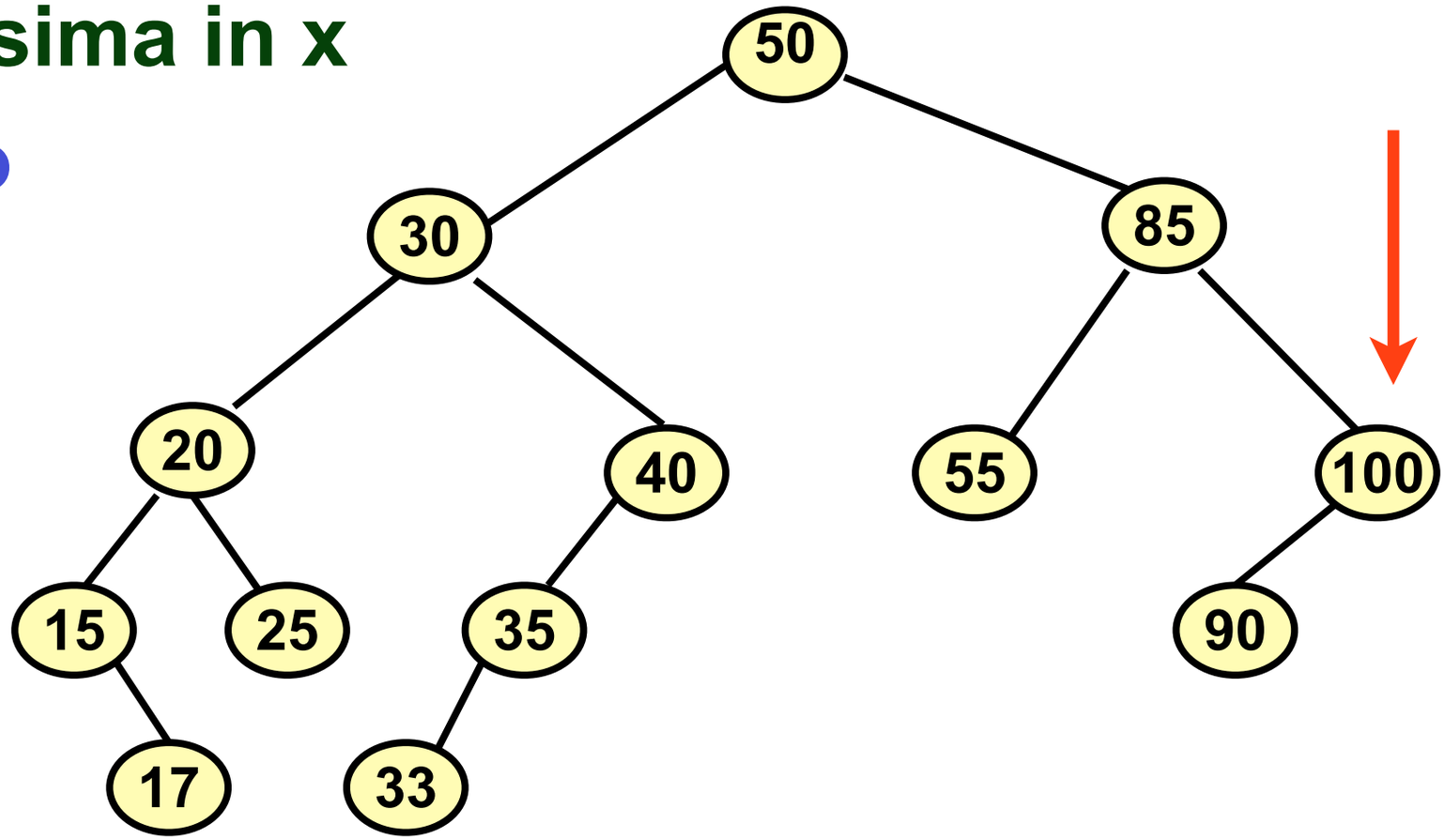
Il massimo

Maximum(x)

precond: x ≠ nil e x è un ABR

postcond: restituisce il puntatore al nodo di chiave massima in x

```
while x.right ≠ nil do
      x = x.right
return x
```



Complessità nel caso peggiore Θ(h), in tutti i casi O(h)

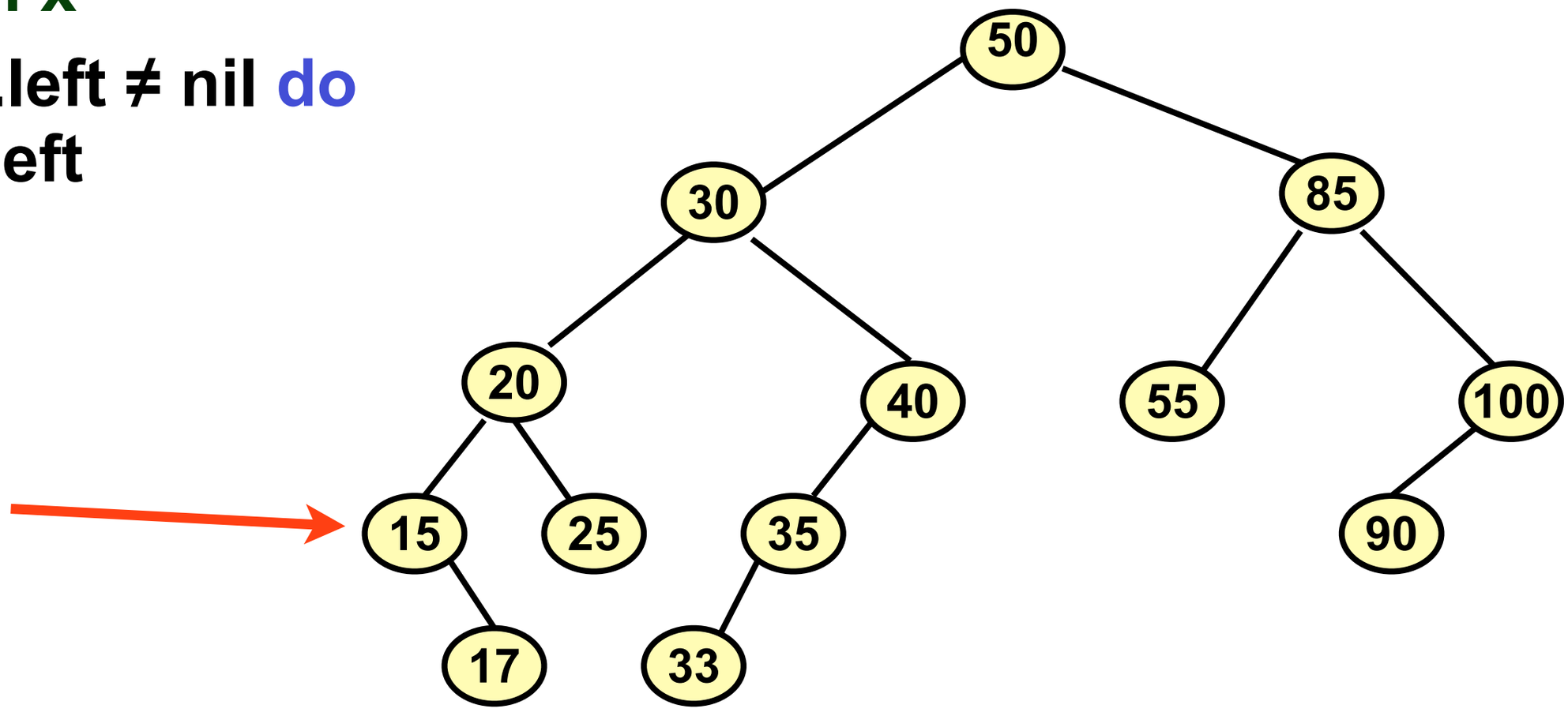
Il minimo

Minimum(x)

precond:x ≠ nil e x è un ABR

postcond:restituisce il puntatore al nodo di chiave minima in x

```
while x.left ≠ nil do
    x = x.left
return x
```



Complessità nel caso peggiore Θ(h), in tutti i casi O(h)

Il minimo: versione ricorsiva

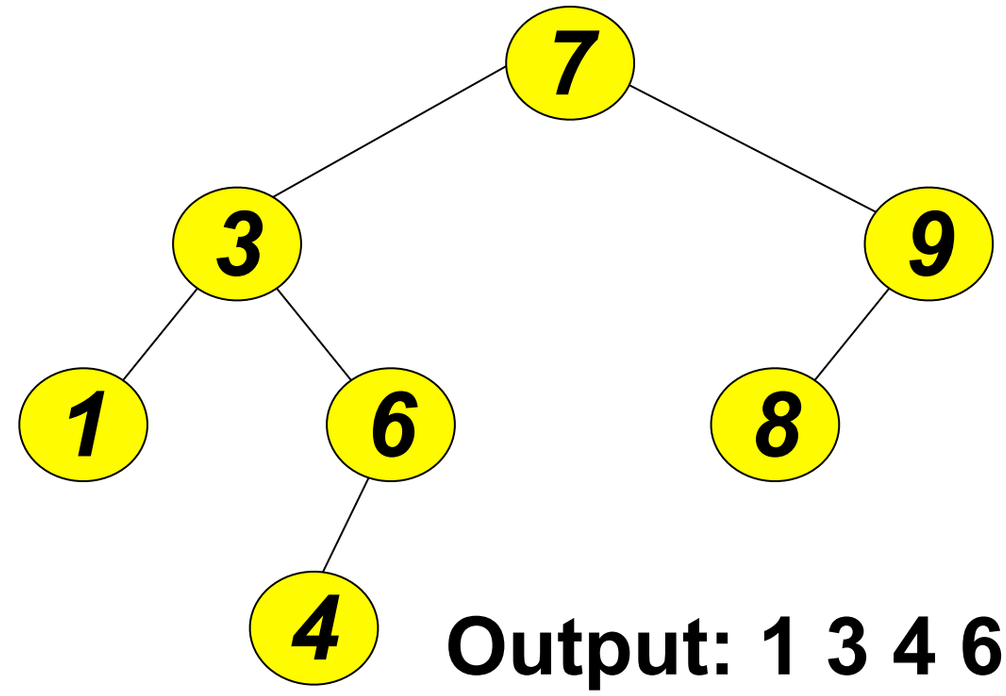
```
MinimumRic(x)
precond: x ≠ nil e x è un ABR
postcond:restituisce la chiave minima in x
 if x.left == NIL
        then return x.key
        else return MinimumRic(x.left)
```

Complessità nel caso peggiore Θ(h), in tutti i casi O(h)

Visita inorder di un ABR

```
Inorder-Tree-Walk(x)
 if x ≠ nil then
      Inorder-Tree-Walk(x.left)
      print   x.key
      Inorder-Tree-Walk(x.right)
```

Input:

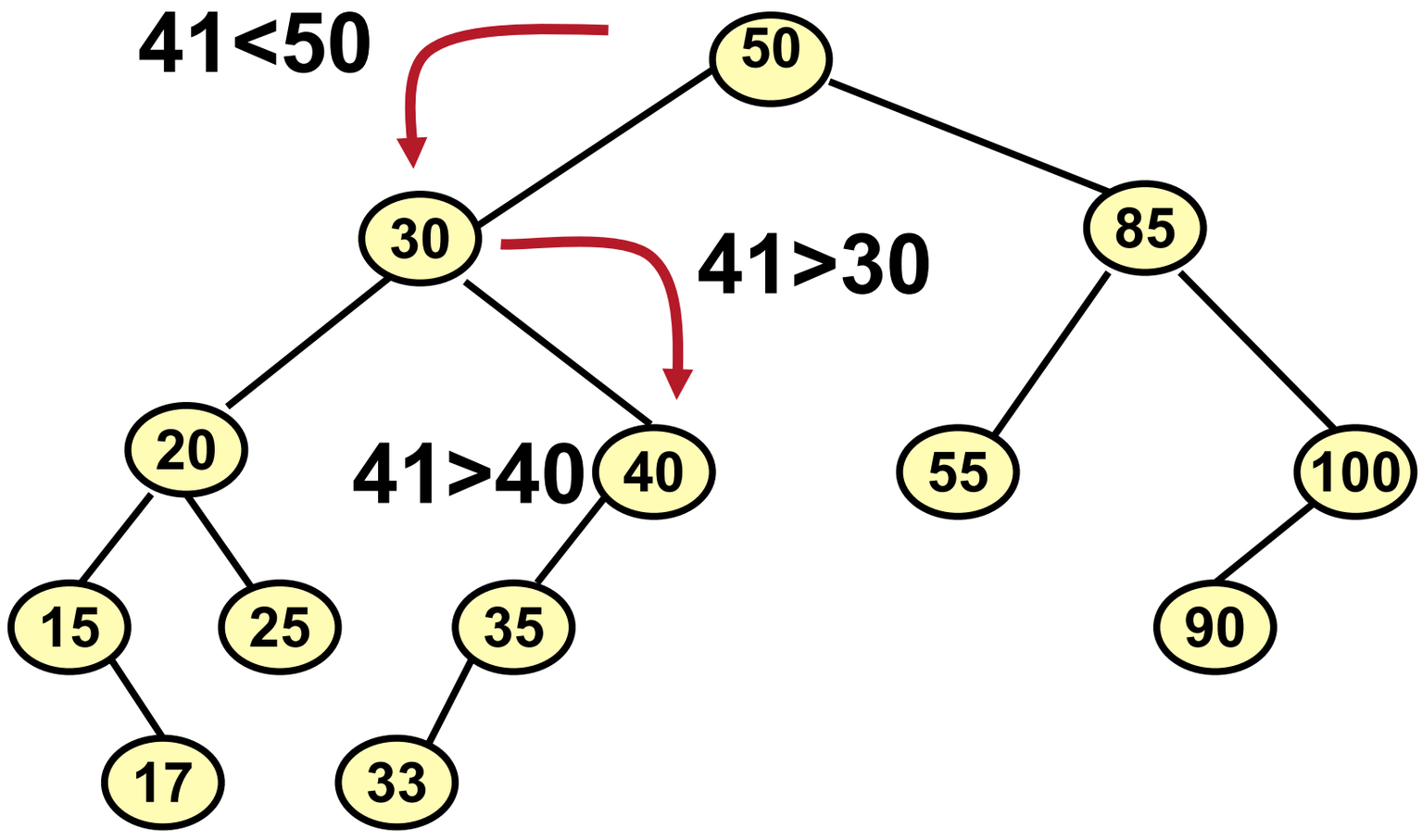


Output: 1 3 4 6 7 8 9

Le chiavi sono ordinate!

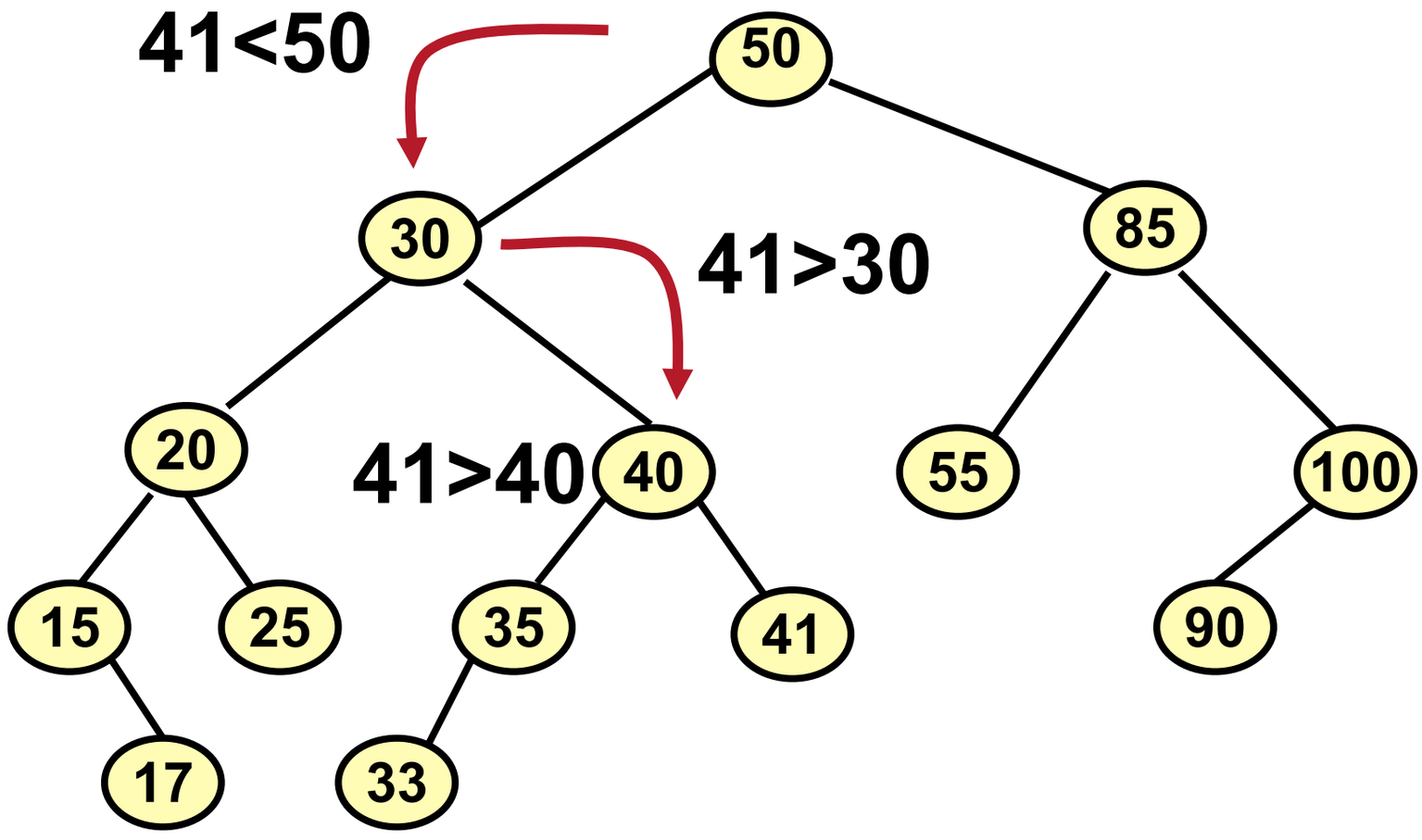
L’inserimento

Inserimento di 41 in



L’inserimento

Inserimento di di 41 in



L'inserimento

```
Insert(T,z)
```

precond: T è un ABR e z è un nodo, con un campo chiave e z.p=z.left=z.right=nil

postcond: inserisce z in T

```
if  T == nil
   then  T.root = z
         z.p == NIL
  else InsertRic(T,z)
```

Inserimento in un albero vuoto z = 60

L'inserimento

```
InsertRic(T,z)
precond:  T è un ABR, T ≠ NIL e z ≠ nil e
z.key non è in T
postcond:inserisce z nell'ABR T
if z.key < T.key
      then
      if T.left == NIL
         then T.left = z
              z.p = T
   else InsertRic(T.left,z)
else
if T.right == NIL
   then T.right = z
              z.p = T
   else InsertRic(T.right,z)
```

Tempo di esecuzione?

L'inserimento

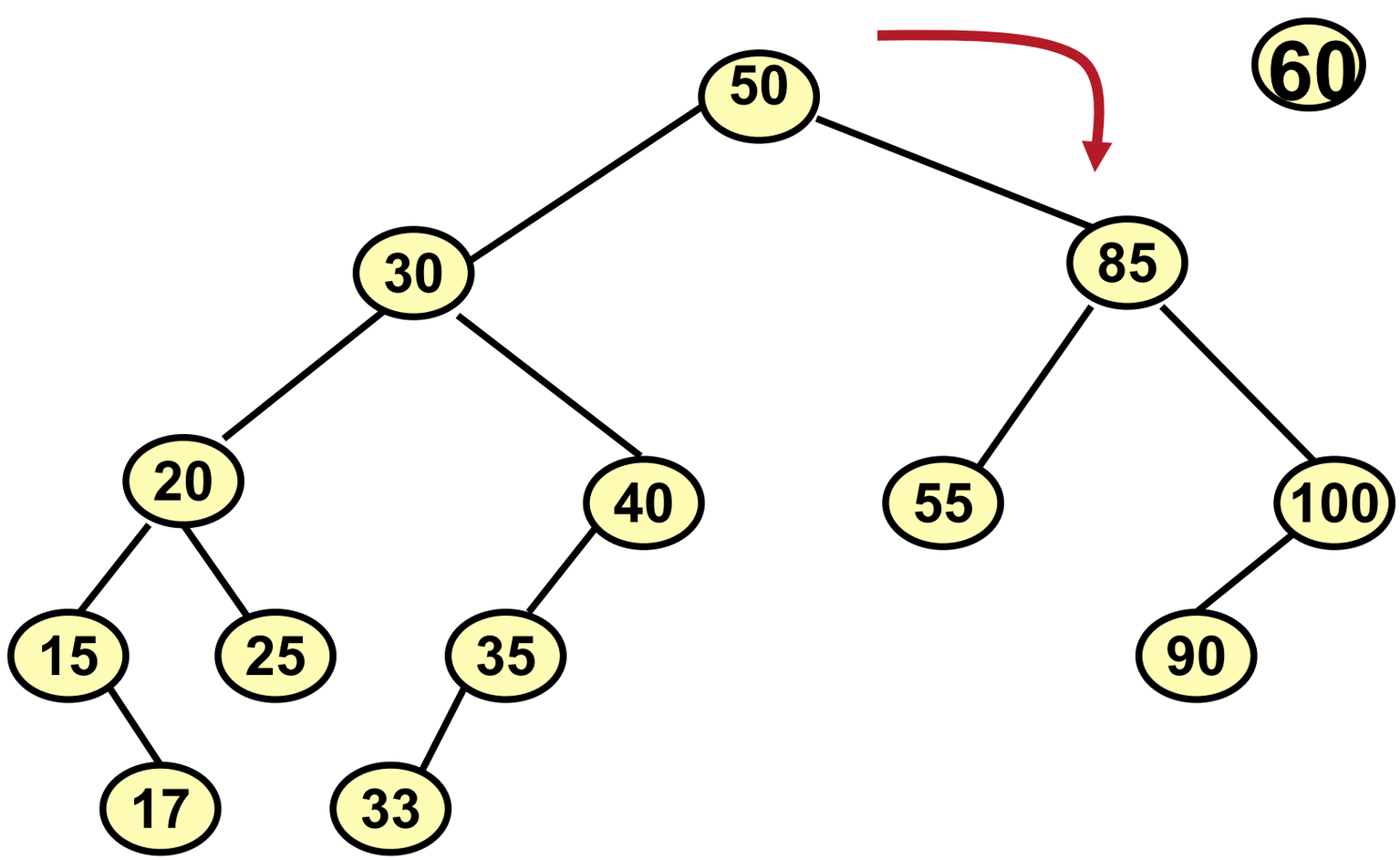
```
InsertRic(T,z)
precond:  T è un ABR, T ≠ NIL e z ≠ nil
postcond:inserisce z nell'ABR T
if z.key < T.key
     then
     if T.left == NIL
        then T.left = z
                z.p = T
        else InsertRic(T.left,z)
else
if T.right == NIL
   then T.right = z
           z.p = T
   else InsertRic(T.right,z)
```

Tempo di esecuzione in O(h)

L'inserimento: esempio

```
InsertRic(T,z)
precond:  T è un ABR, T ≠ NIL e z ≠ nil
postcond:inserisce z nell'ABR T
if z.key < T.key
     then
     if T.left == NIL
        then T.left = z
             z.p = T
     else InsertRic(T.left,z)
else
if T.right == NIL
     then T.right = z
             z.p = T
     else InsertRic(T.right,z)
```

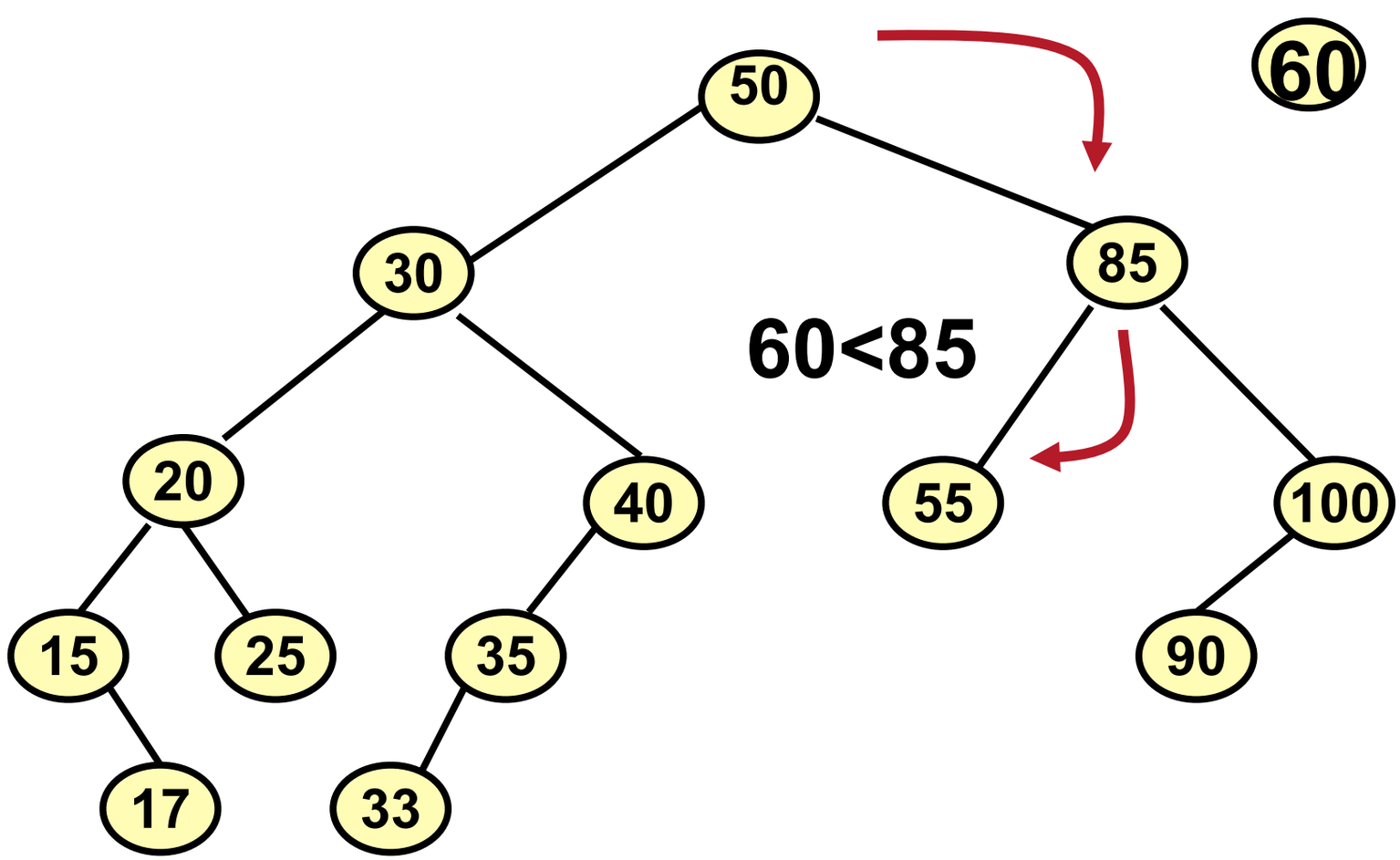
Inseriamo z = 60



L'inserimento: esempio

```
InsertRic(T,z)
precond:  T è un ABR, T ≠ NIL e z ≠ nil
postcond:inserisce z nell'ABR T
if z.key < T.key
     then
     if T.left == NIL
        then T.left = z
             z.p = T
     else InsertRic(T.left,z)
else
if T.right == NIL
     then T.right = z
             z.p = T
     else InsertRic(T.right,z)
```

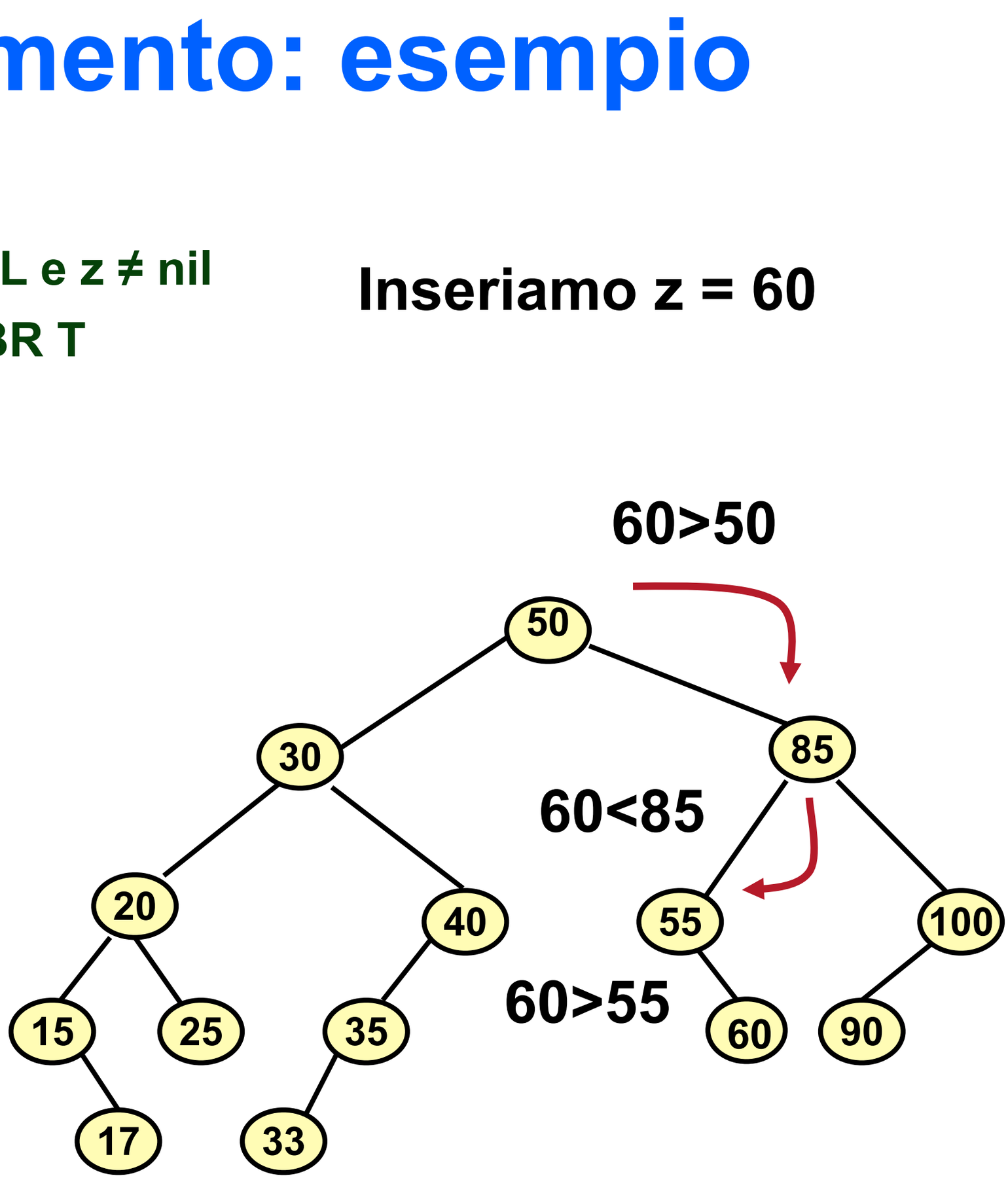
Inseriamo z = 60



L'inserimento: esempio

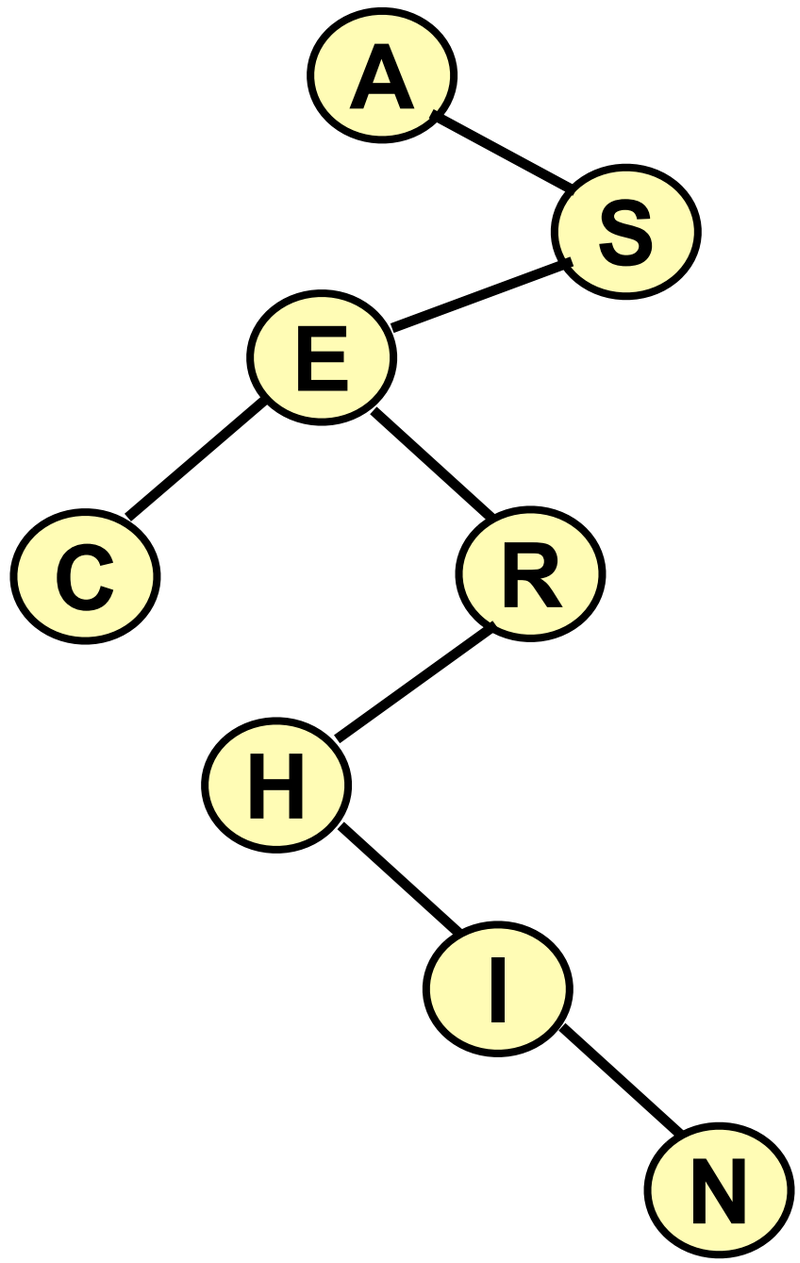
```
InsertRic(T,z)
precond:  T è un ABR, T ≠ NIL e z ≠ nil
postcond:inserisce z nell'ABR T
if z.key < T.key
     then
     if T.left == NIL
        then T.left = z
                z.p = T
     else InsertRic(T.left,z)
else
if T.right == NIL
     then T.right = z
                z.p = T
     else InsertRic(T.right,z)
```

Inseriamo z = 60



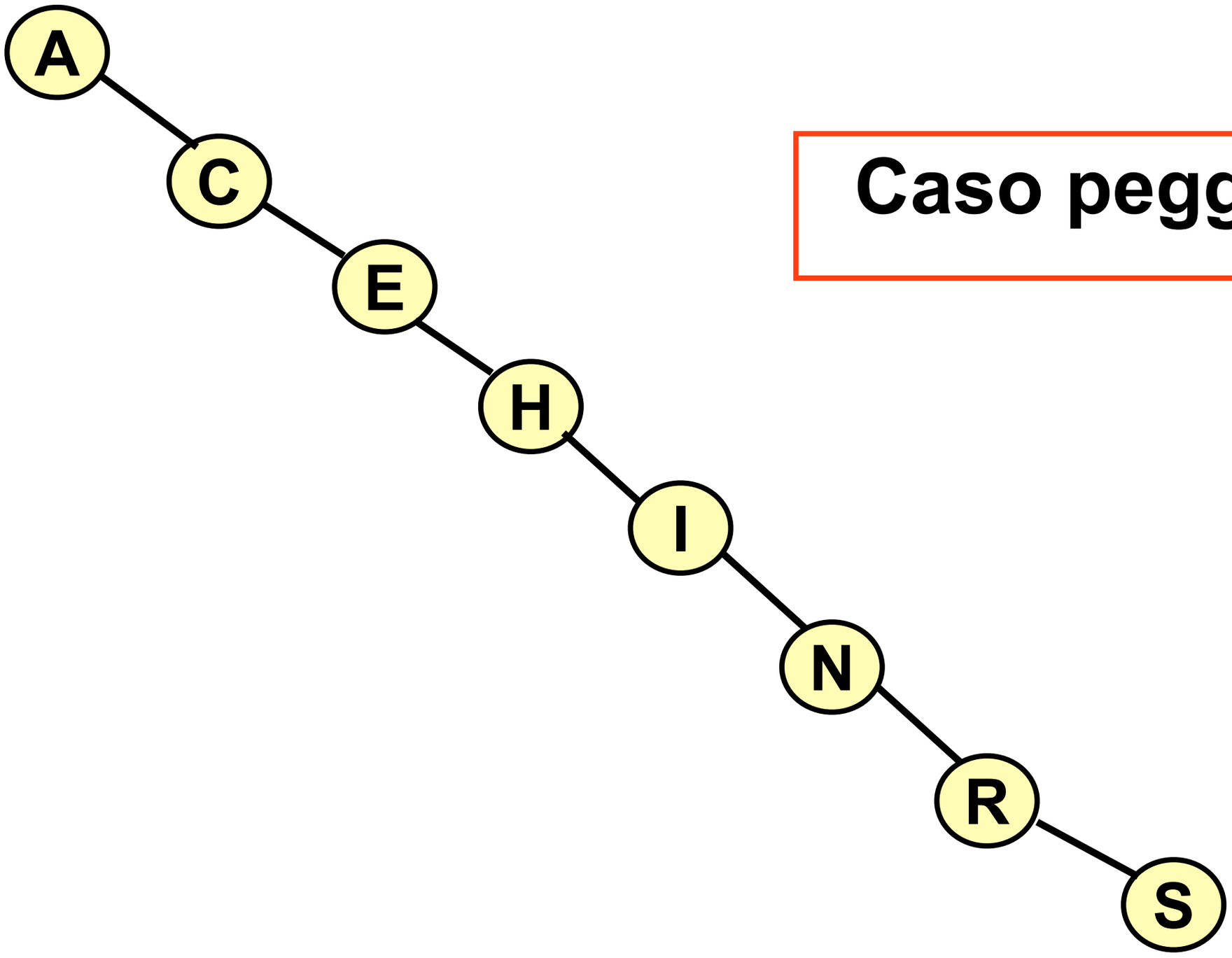
Forma dell'ABR

Inseriamo nell'ordine A S E R C H I N



Forma dell'ABR

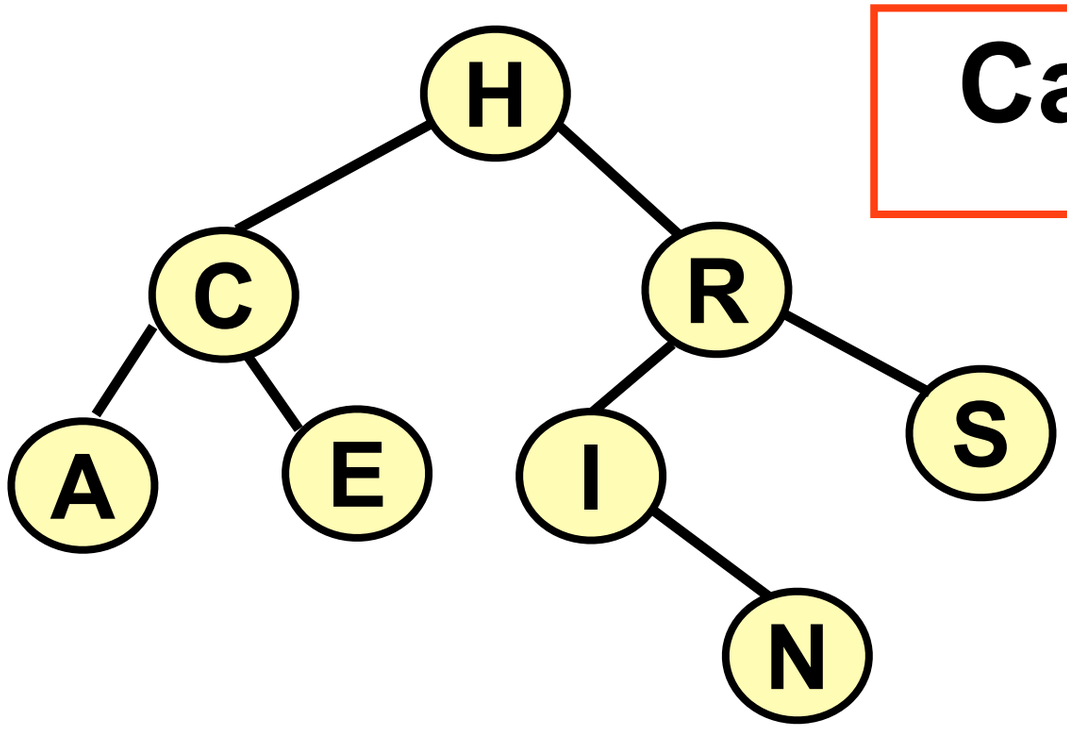
Inseriamo A C E H I N R S



Caso peggiore

Forma dell'ABR

Inseriamo H C A R E I N S



Caso migliore

La forma dell'albero dipende dall'ordine di inserimento

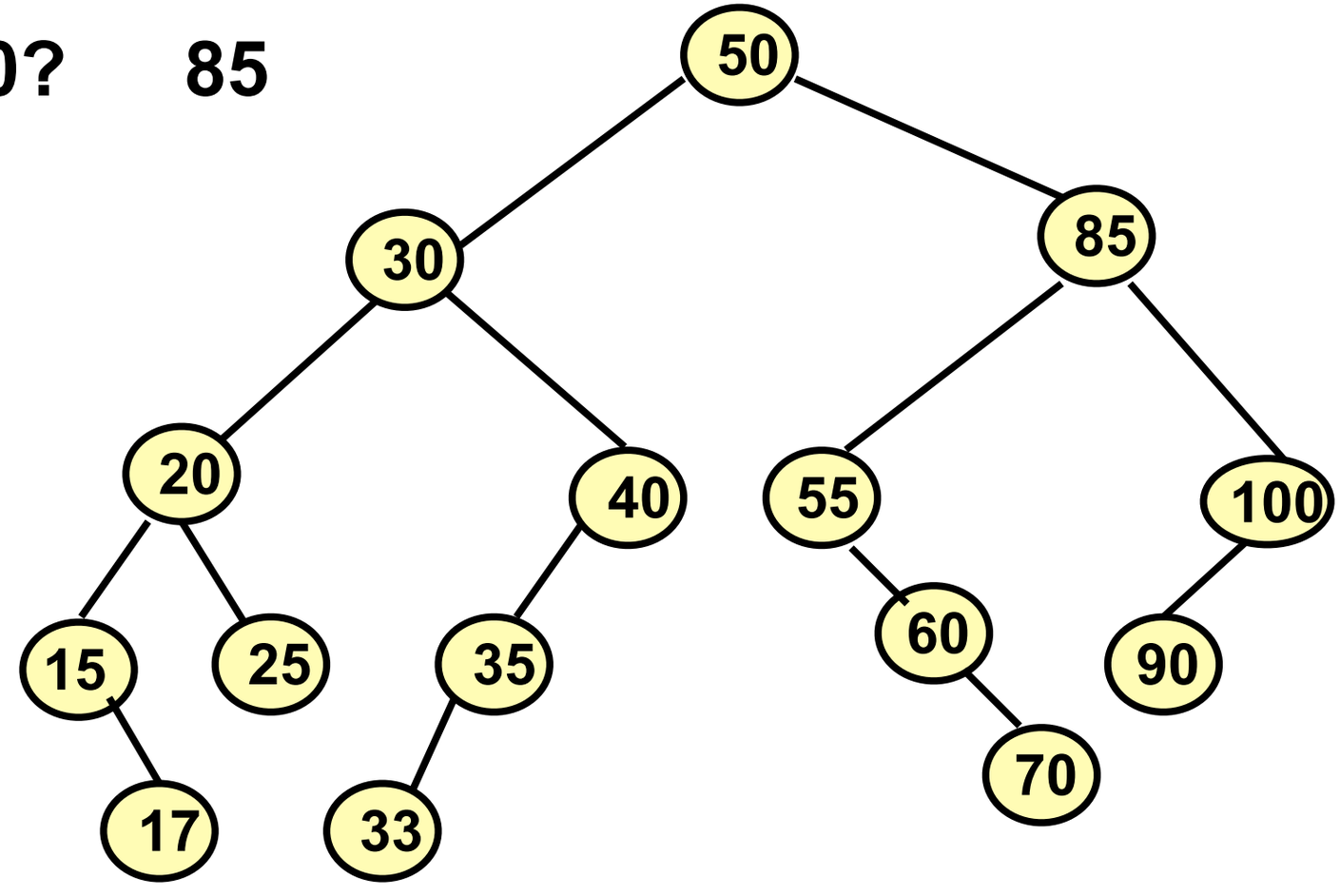
Successivo di un nodo in un ABR

La chiave successiva a una data nell'albero, se c'è, dove può essere?

25 è il successivo di 20

Chi è il successivo di 50? **55**

Chi è il successivo di 70? **85**



Successivo di un nodo in un ABR

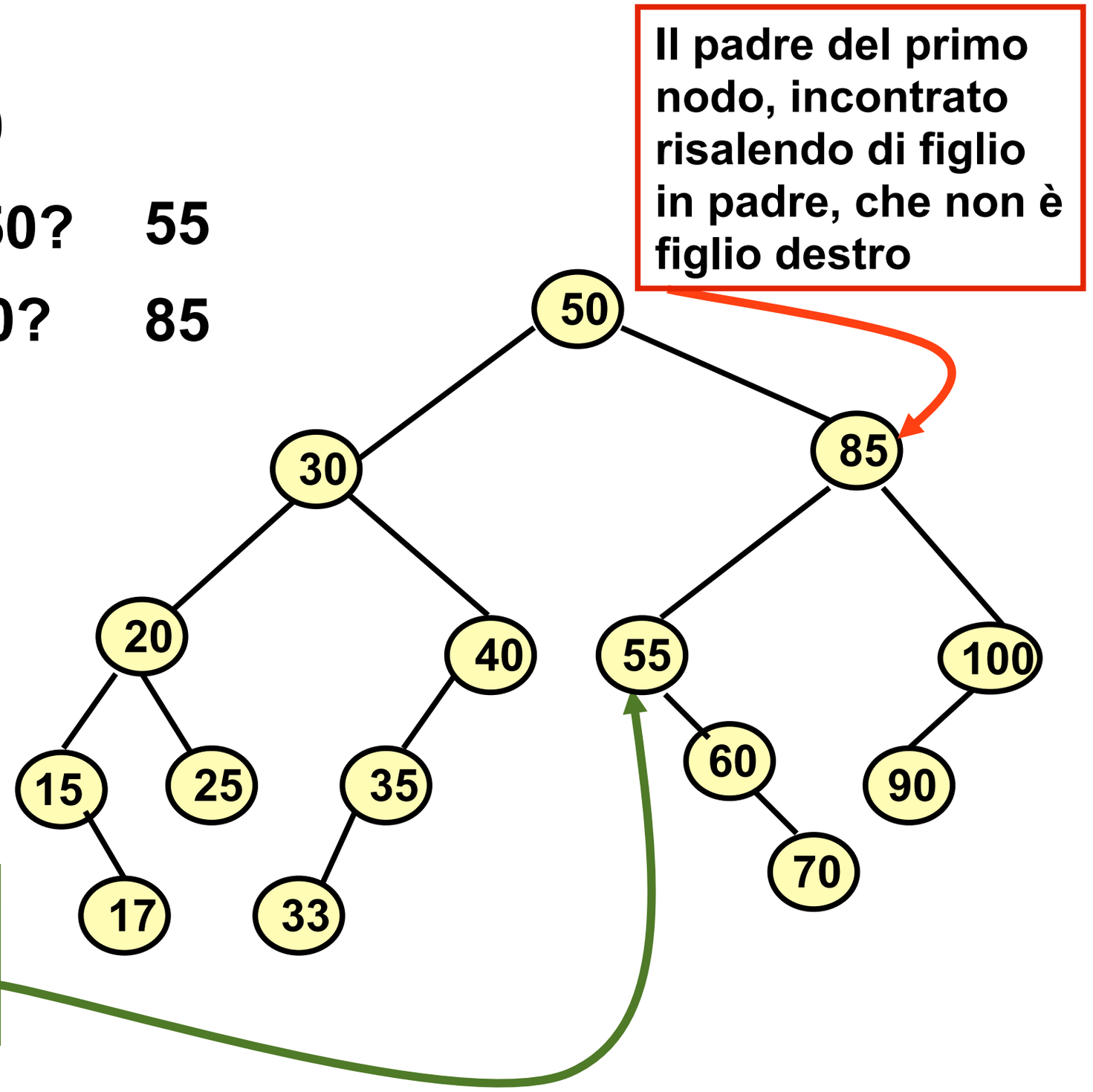
La chiave successiva a una data nell'albero, se c'è, dove può essere?

25 è il successivo di 20

Chi è il successivo di 50? **55**

Chi è il successivo di 70? **85**

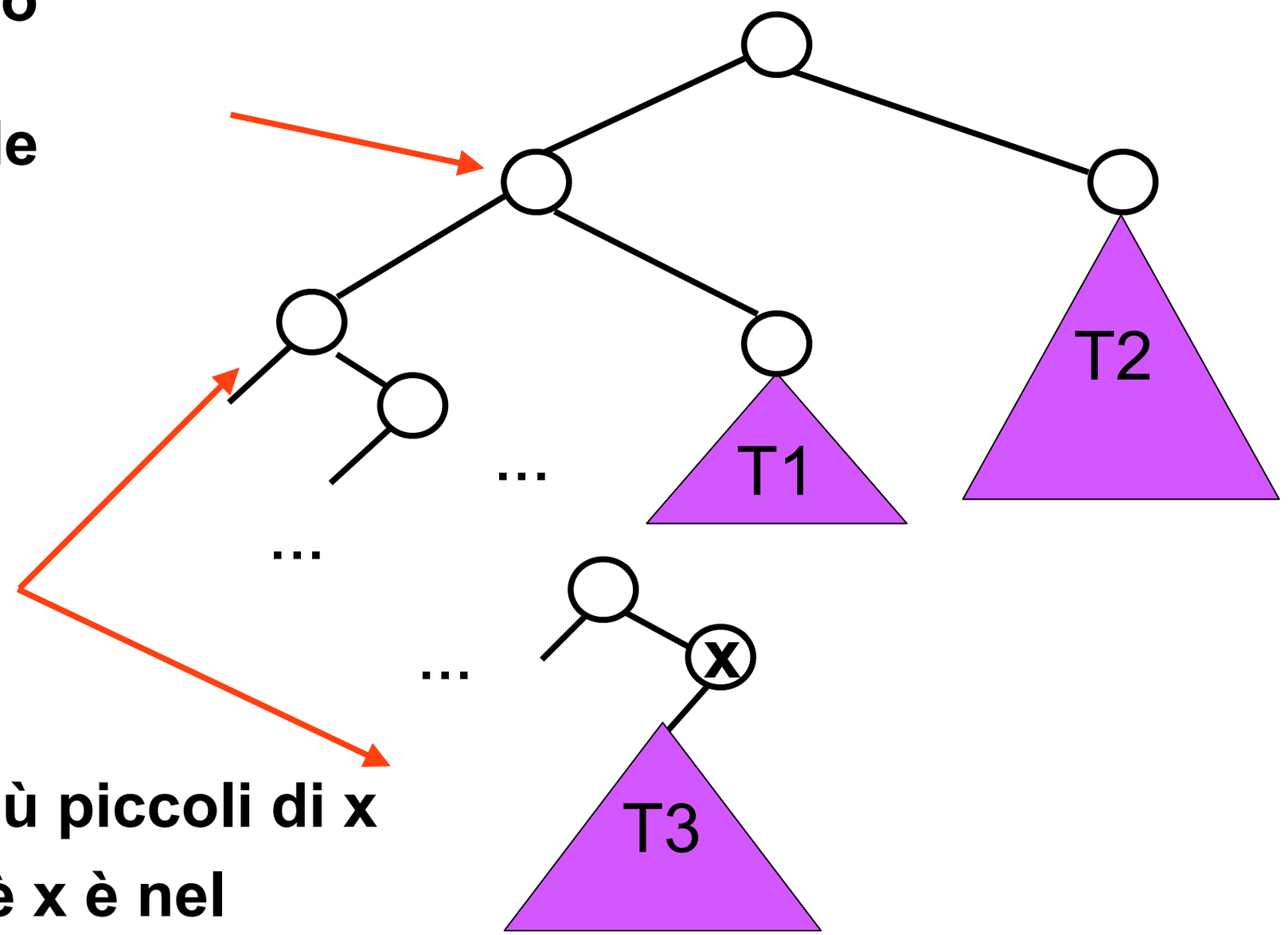
La chiave successiva di una data chiave x, in un nodo senza figli destri, nell'albero è quella del primo nodo, risalendo dal nodo verso la radice, che ha x nel sotto albero sinistro.



Successivo di un nodo di chiave x senza figlio destro

il primo risalendo da x verso la radice più grande di x

e cioè che ha x nel sottoalbero sinistro!

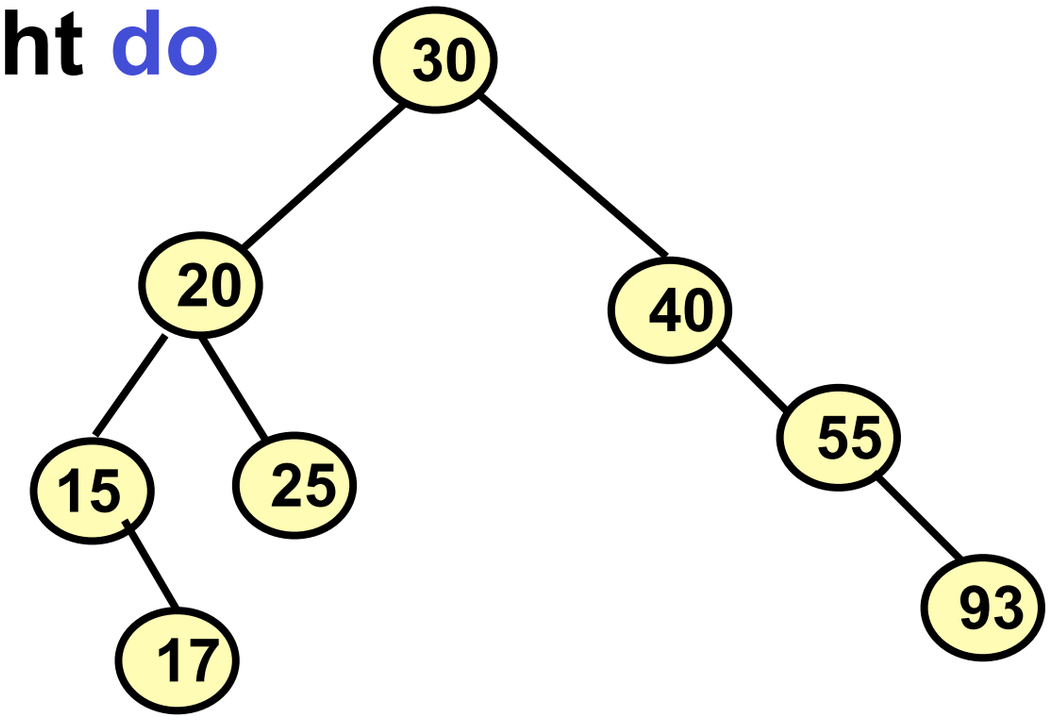


tutti più piccoli di x

perchè x è nel sottoalbero destro di questi nodi

Successivo: pseudocodice

```
Successor(x)
prec: x è un puntatore a un nodo in un ABR
postc: restituisce il puntatore al nodo di chiave
successivo a quella di x, se c'è, NIL altrimenti
if x.right ≠ nil then
      return Minimum(x.right)
y = x.p
   while y ≠ nil and x == y.right do
            x = y
            y = x.p
   return y
```



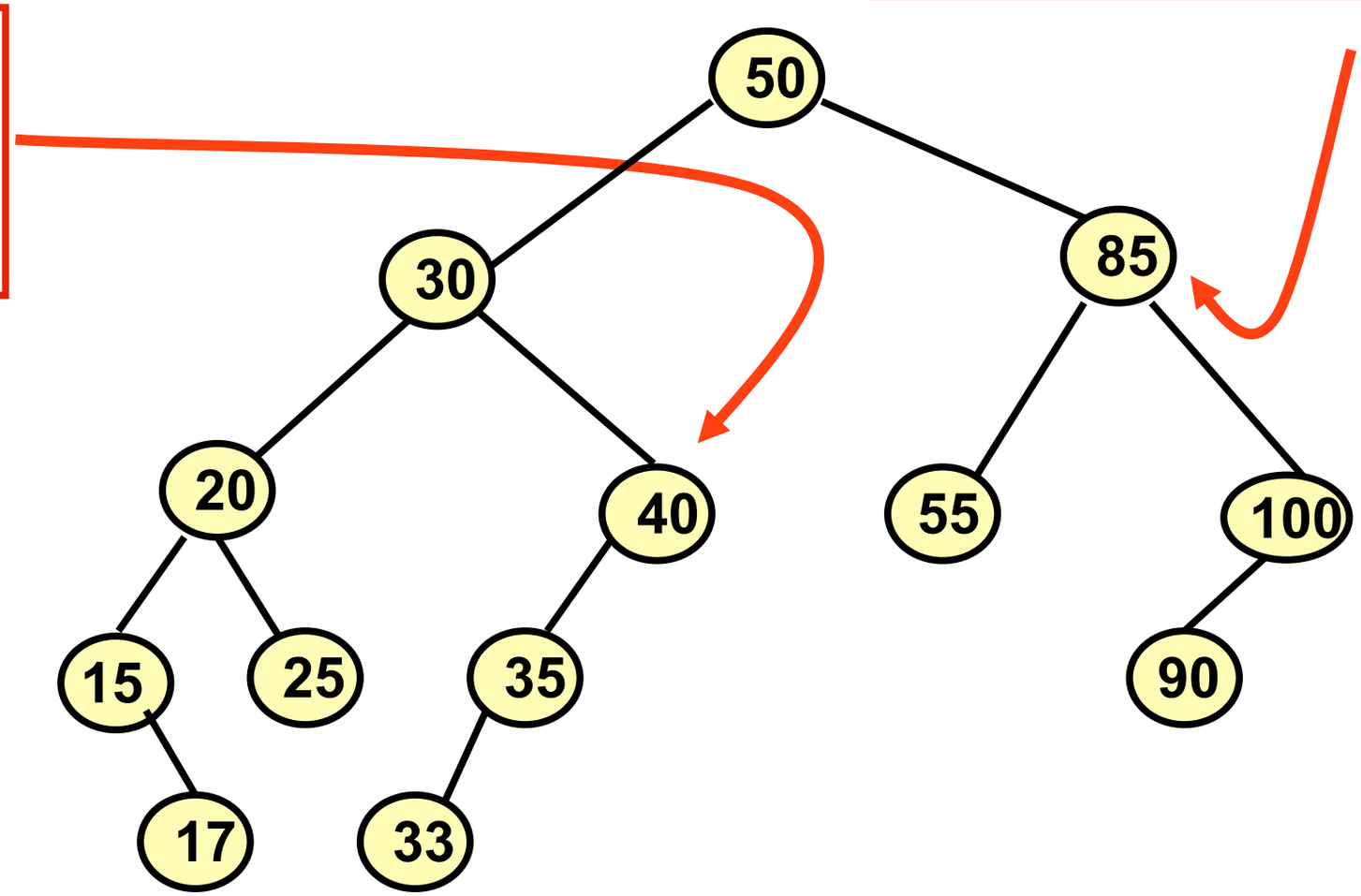
Precedente

35 è il precedente di 40

Chi è il precedente di 50? **40**

Il padre del primo nodo, incontrato risalendo di figlio in padre che non è figlio sinistro

Il più grande elemento nel sottoalbero sinistro



Chi è il precedente di 90? **85**

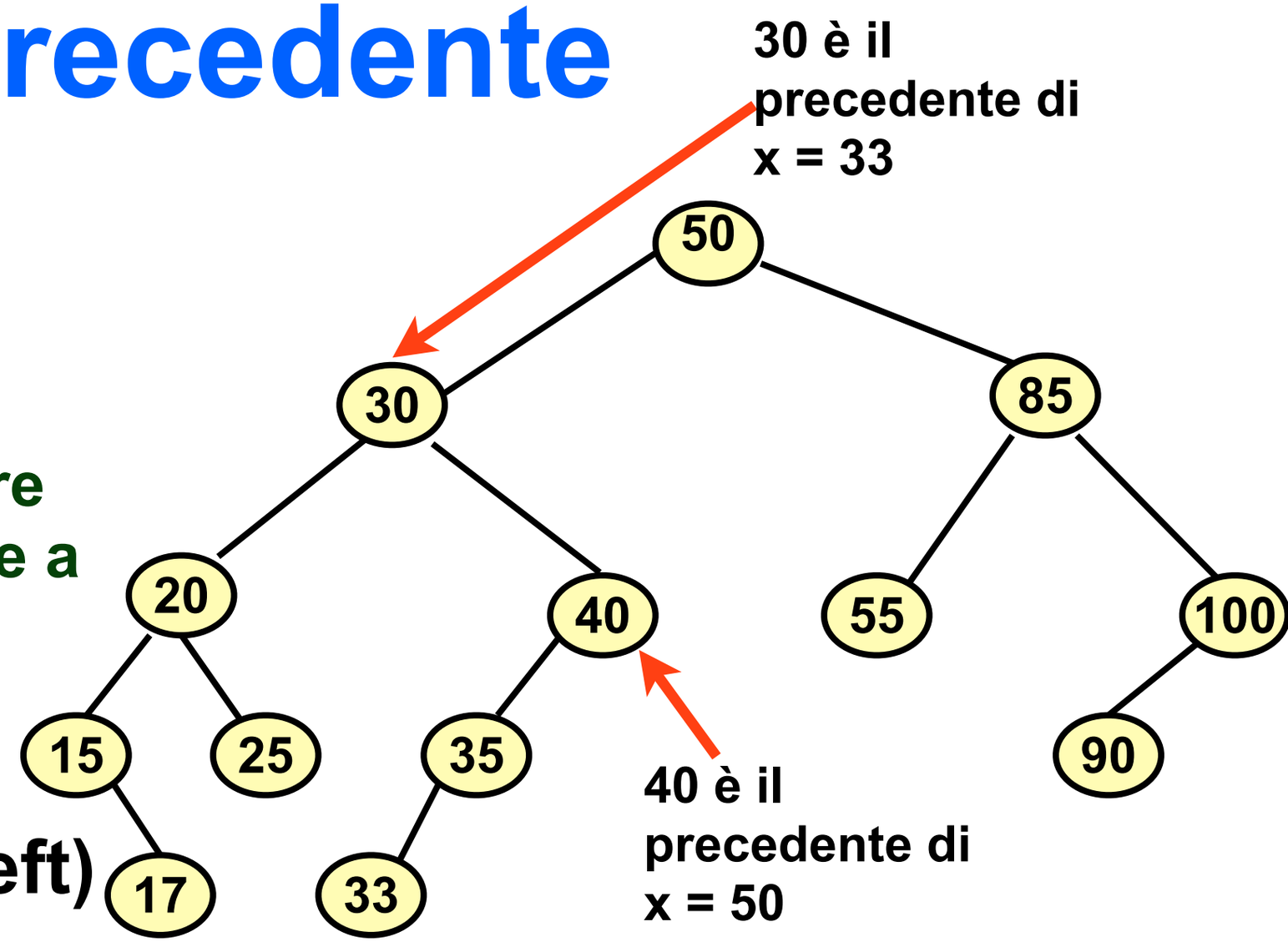
Il precedente

Predecessor(x)

prec: x è un puntatore a un nodo in un ABR

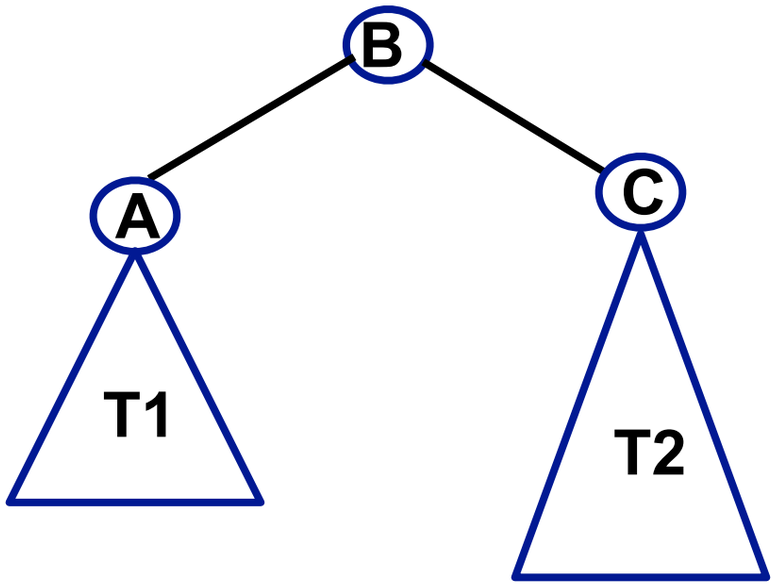
postc: restituisce il puntatore al nodo di chiave precedente a quella di x, se c'è, NIL altrimenti

```
if x.left ≠ nil then
    return Maximum(x.left)
y = x.p
  while y ≠ nil and x == y.left do
          x = y
          y = x.p
  return y
```

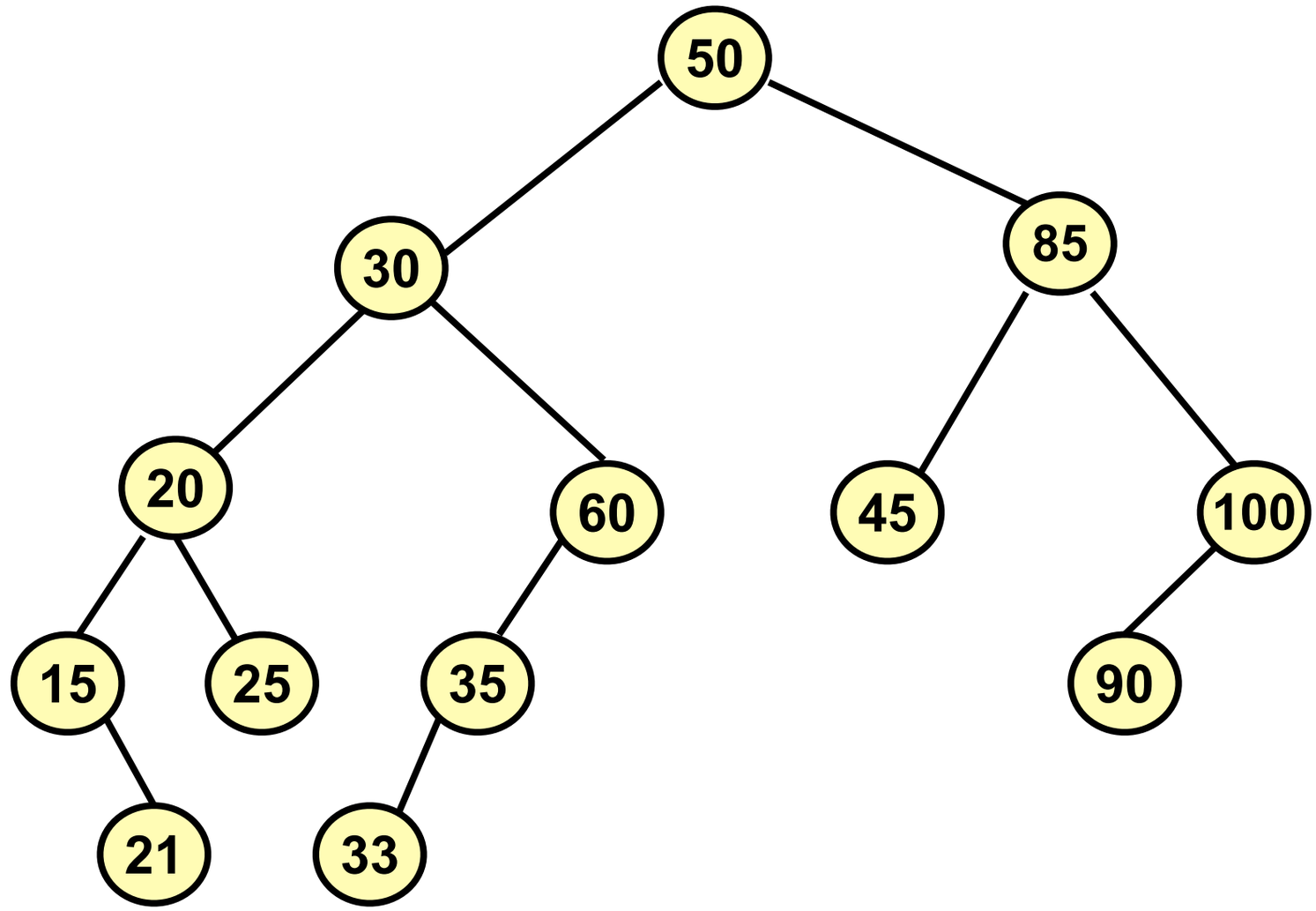


Complessità O(h)

Definizione ricorsiva di ABR

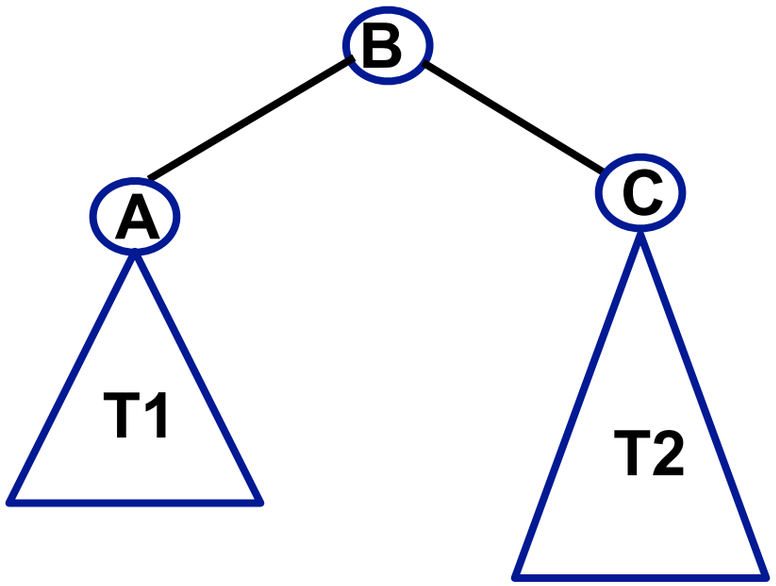


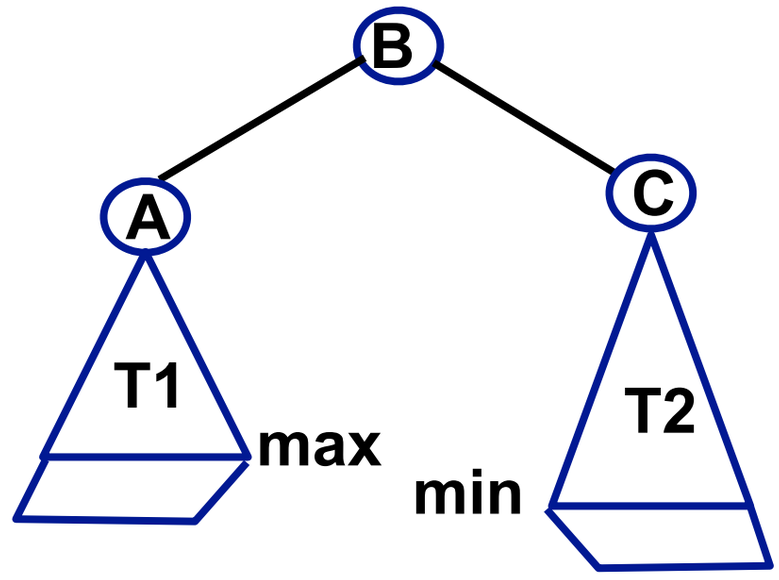
Un albero binario è un ABR se
il suo sotto albero sinistro, T1, è un ABR
il suo sotto albero destro, T2, è un ABR
e A < B < C



NO! Questo definisce un albero come quello sopra disegnato che non è un ABR!

Definizione ricorsiva di ABR





Un albero binario è un ABR se
il suo sotto albero sinistro, T1, è un ABR
il suo sotto albero destro, T2, è un ABR
e la sua radice B è maggiore del massimo nel sotto albero sinistro e minore del minimo nel sotto albero destro.
La base della definizione ricorsiva è l'albero vuoto.

Algoritmo di Verifica basato sulla definizione

ABR2(T)
input: un albero binario T
postc: restituisce vero se T è ABR, con chiavi distinte

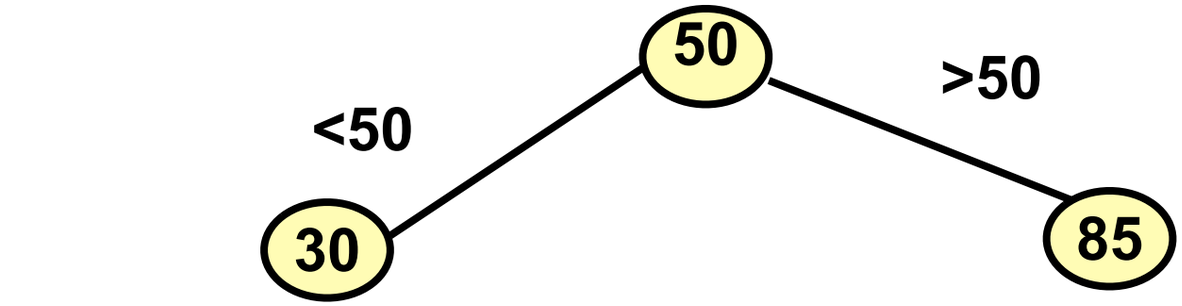
```
if (T == NIL) return 1

if (T.left != NIL and MaximumRic(T.left) > T.key) return 0

if (T.right != NIL and MinimumRic(T.rigth) < T.key) return 0

return (ABR2(T.left) and ABR2(T.right))
/*vero se , recursivamente, il sottoalbero sinistro e il destro sono  ABR */
```

Qui MaximumRic(T) e MinimumRic(T) danno in output il valore della chiave massima rispettivamente minima in T

Tempo di esecuzione?

O(nh), se h è l'altezza e n il numero dei nodi dell'albero binario T.
Troppo.

Proprietà dell'ABR

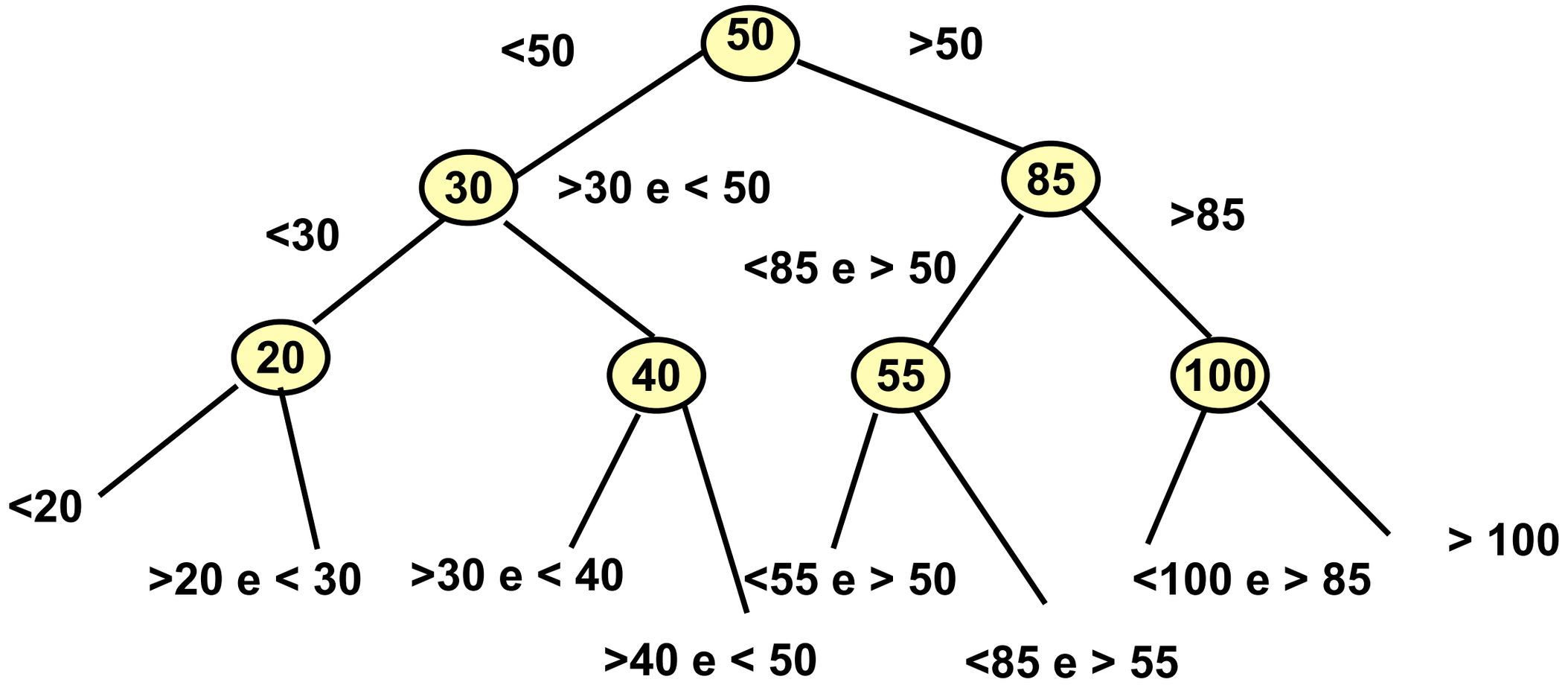


Ogni volta che scendiamo di padre in figlio:

se il figlio è un figlio destro posso aggiornare il limite inferiore ai valori che possono trovarsi nel sotto albero destro, usando la chiave del padre

se il figlio è un figlio sinistro posso aggiornare il limite superiore ai valori che possono trovarsi nel sotto albero sinistro, usando la chiave del padre.

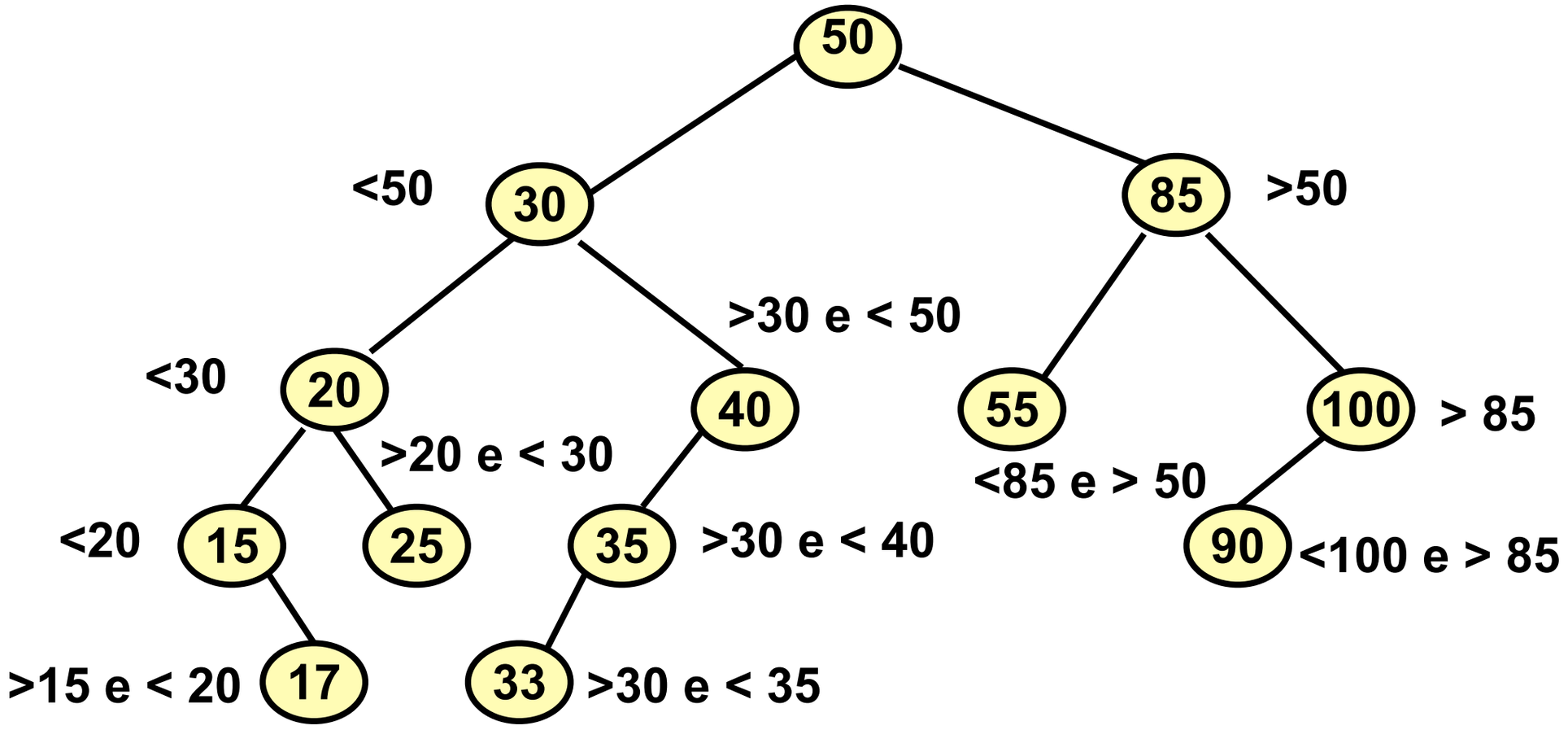
Proprietà dell'ABR



Ogni volta che scendiamo di padre in figlio:
se il figlio è un figlio destro posso aggiornare il limite inferiore ai valori che possono trovarsi nel sotto albero destro, usando la chiave del padre

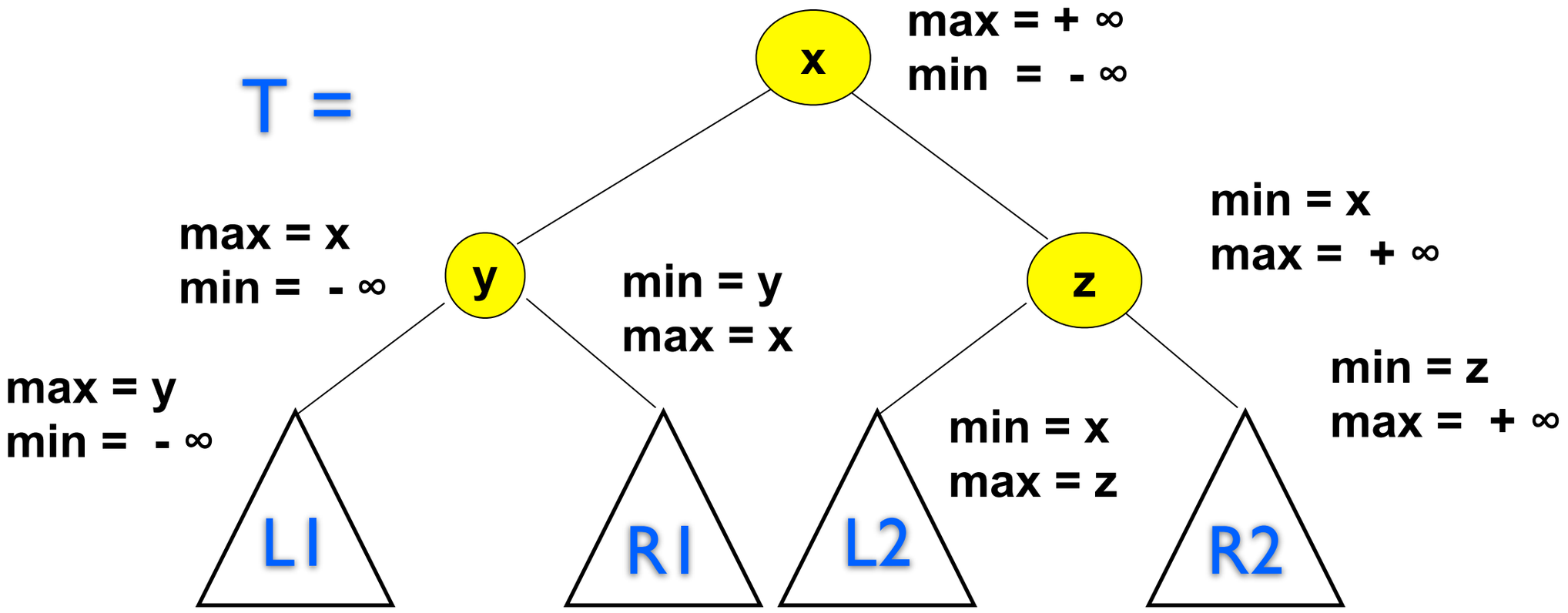
se il figlio è un figlio sinistro posso aggiornare il limite superiore ai valori che possono trovarsi nel sotto albero sinistro, usando la chiave del padre.

Algoritmo di verifica se un albero binario è un ABR



Per verificare se si tratta di un ABR:
si controlla se scendendo di padre in figlio la chiave del nodo si trova nell'intervallo di valori determinati dal limite inferiore corrente e il limite superiore corrente
Se questi vincoli sono rispettati per tutti i nodi, l'albero è un albero binario di ricerca

Algoritmo di verifica efficiente



Si modifica la visita preorder passando in due parametri i valori min e max da considerare:
se si scende al figlio sinistro, la chiave del padre è il nuovo valore di max,
se si scende a destra la chiave del padre è il nuovo valore di min.

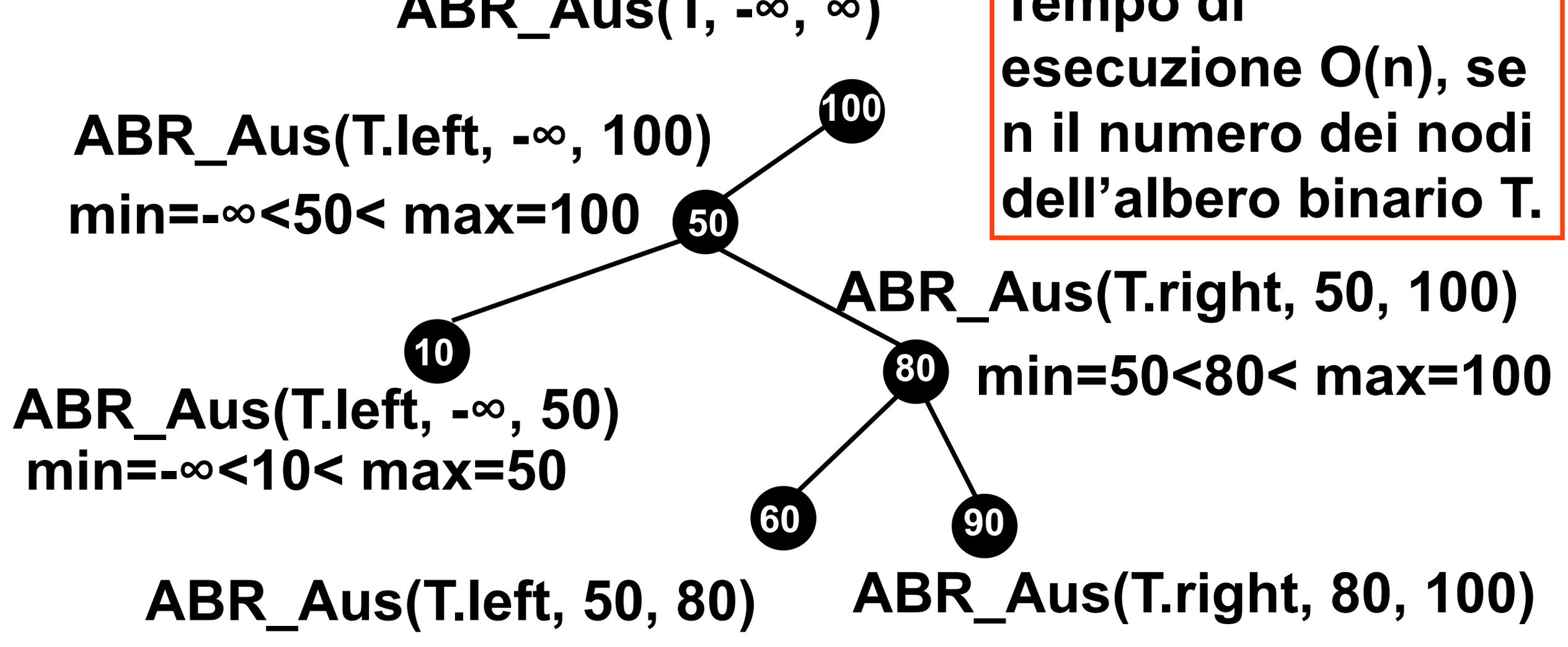
Pseudocodice

```
ABR3(T)
input: T è un albero binario
postc:restituisce vero se T è ABR, con chiavi distinte
min = - ∞; max = ∞
return(ABR_Aus(T, min,max))

ABR_Aus(T,min,max)
input: T è un albero binario, min  e max due interi
output: vero se T è un ABR
if (T == NIL) return 1
if (T.key ≤ min || T.key ≥ max ) return 0
return (ABR_Aus(T.left, min,T.key)  and
  ABR_Aus(T.right,T.key,max))
/*vero se, ricorsivamente, i nodi nel sottoalbero
  sinistro
sono tra min e T.key e quelli del  destro sono  tra T.key
  e max */
```

Verificare che un albero binario è un ABR

```
ABR_Aus(T,min,max)
input: T è un albero binario, min  e max due interi
output: restituisce vero se T è un ABR
if (T == NIL) return 1
if (T.key ≤ min || T.key ≥ max ) return 0
return (ABR_Aus(T.left, min,T.key)  and    ABR_Aus(T.right,T.key,max))
/*vero se, ricorsivamente, i nodi nel sottoalbero sinistro
sono tra min e T.key e quelli del  destro sono  tra T.key e max */
```

Tempo di esecuzione O(n), se n il numero dei nodi dell'albero binario T.



Capire ABR e maxHeap

Rispondere alle seguenti domande:

1. **E' possibile che un ABR T, <u>completo almeno fino al penultimo livello</u>, contenente almeno tre nodi ed avente chiavi distinte soddisfi anche le proprietà di ordinamento tra padri e figli di un Max-heap? Se si, mostrare un esempio di tale albero. Se no, dimostrarne l'impossibilità.**

Capire ABR e maxHeap

Rispondere alle seguenti domande:

1. **E' possibile che un ABR T, <u>completo almeno fino al penultimo livello</u>, contenente almeno tre nodi ed avente chiavi distinte soddisfi anche le proprietà di ordinamento tra padri e figli di un Max-heap? Se si, mostrare un esempio di tale albero. Se no, dimostrarne l'impossibilità.**

La risposta è no, il figlio destro deve avere una chiave maggiore di quella della radice in un ABR, mentre deve essere minore in un Max-Heap.

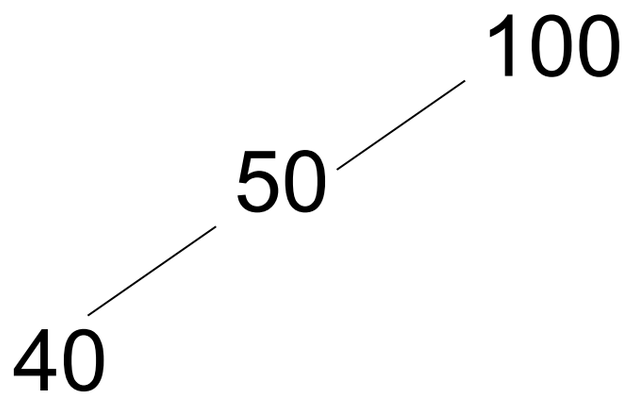
Capire ABR e maxHeap

2. E’ possibile che un ABR T, contenente almeno tre nodi ed avente chiavi distinte soddisfi anche le proprietà di ordinamento tra padri e figli di un Max-heap?
Se si, mostrare un esempio di tale albero.
Se no, dimostrarne l’impossibilità.

Capire ABR e maxHeap

2. E’ possibile che un ABR T, contenente almeno tre nodi ed avente chiavi distinte soddisfi anche le proprietà di ordinamento tra padri e figli di un Max-heap? Se si, mostrare un esempio di tale albero. Se no, dimostrarne l’impossibilità.

La risposta è sì, basta prendere un albero degenere a sinistra, allora ogni figlio ha chiave minore del padre.



Capire un ABR - 1

Supponiamo che l'operazione di ricerca di una chiave k in un albero binario di ricerca termini su di una foglia. Consideriamo tre insiemi:

- **A, l'insieme delle chiavi alla sinistra del cammino di ricerca;**
- **B, l'insieme delle chiavi del cammino di ricerca; cioè il cammino determinato dai confronti necessari a trovare k nell'albero.**
- **C, l'insieme delle chiavi alla destra del cammino di ricerca.**

Si potrebbe credere che se $a \in A$, $b \in B$ e $c \in C$, allora $a \leq b \leq c$. Si produca un esempio che contraddice questa affermazione, specificando gli insieme A,B e C.