In questa lezione

- **Alberi binari di ricerca**

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

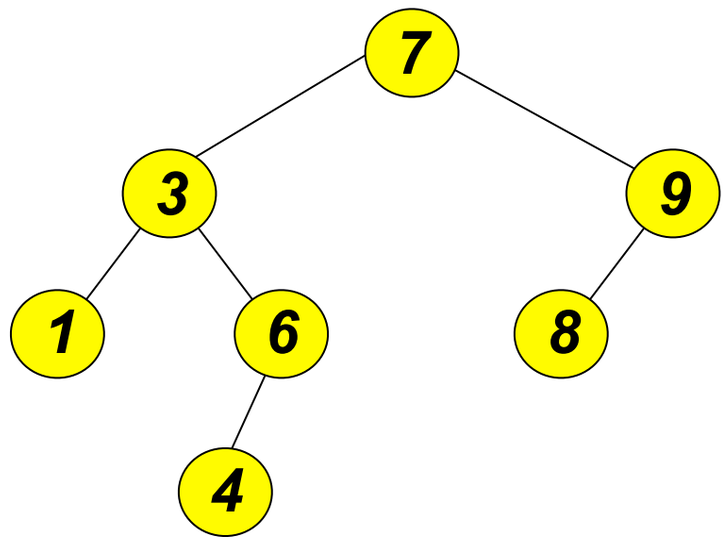
Definizione 1. Un ABR è un albero binario che soddisfa la proprietà della ricerca:

per ogni nodo v

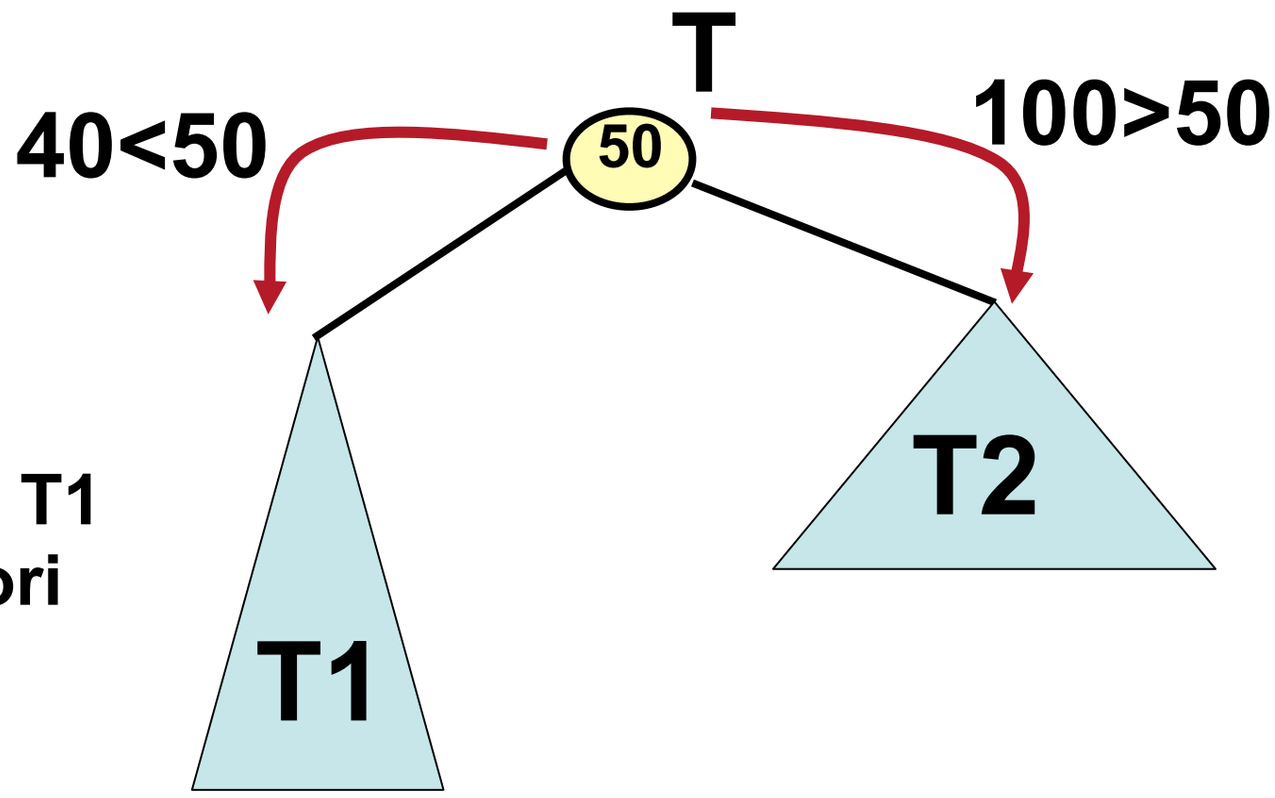
- **le chiavi dei nodi nel sottoalbero sinistro di v sono minori o uguali alla chiave di v e**
- **le chiavi dei nodi nel sottoalbero destro di v sono maggiori o uguali alla chiave di v.**

Esempio:

Nel seguito supponiamo che le chiavi siano tutte distinte



La definizione consente una ricerca veloce!



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

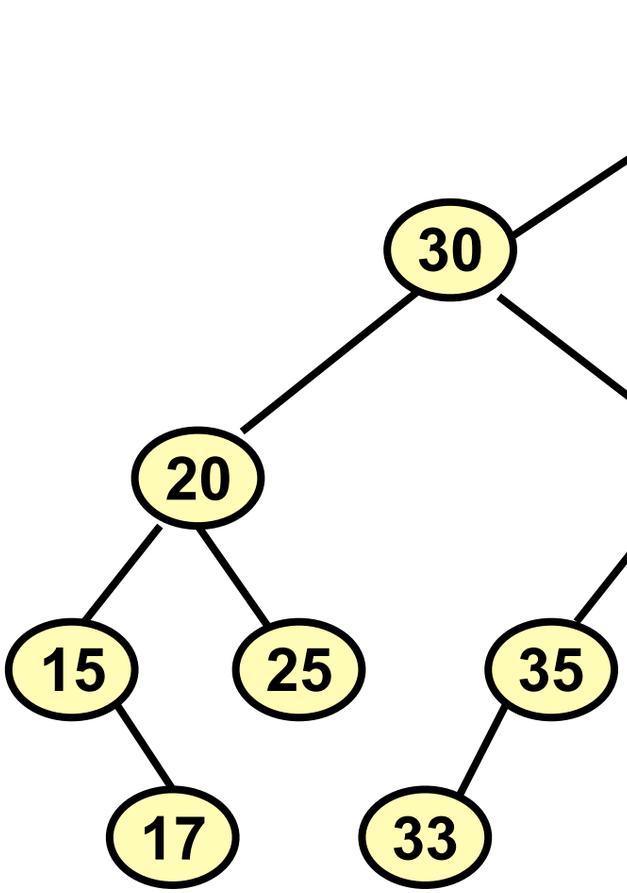
Tutti nel sotto albero destro T2 di T sono maggiori di 50!

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

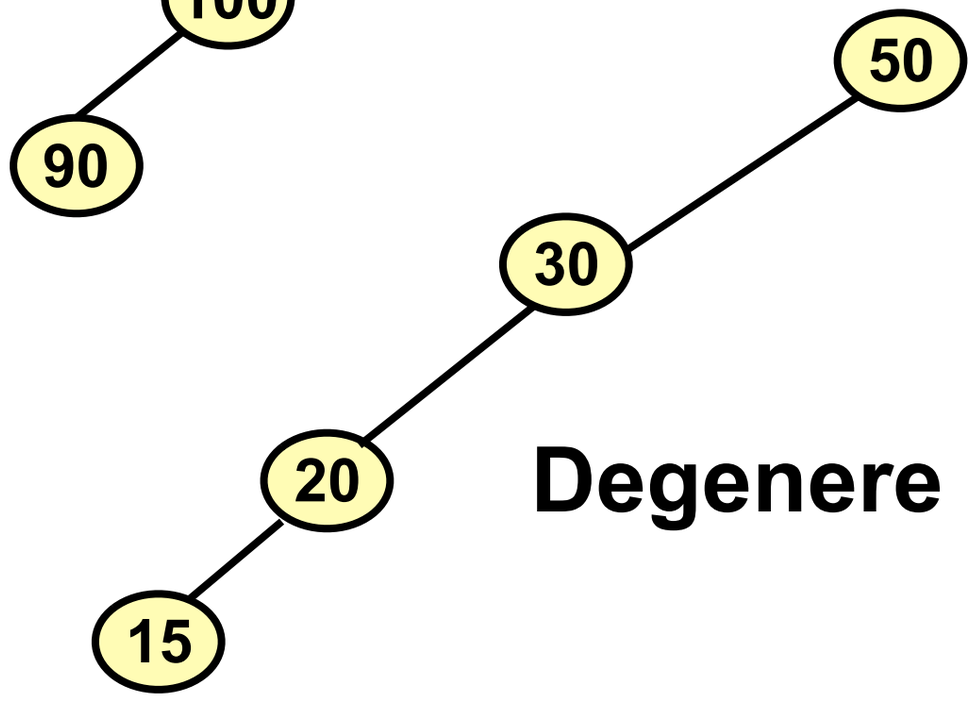
La ricerca di 100 in T deve proseguire nel sotto albero destro T2

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

ABR: esempi



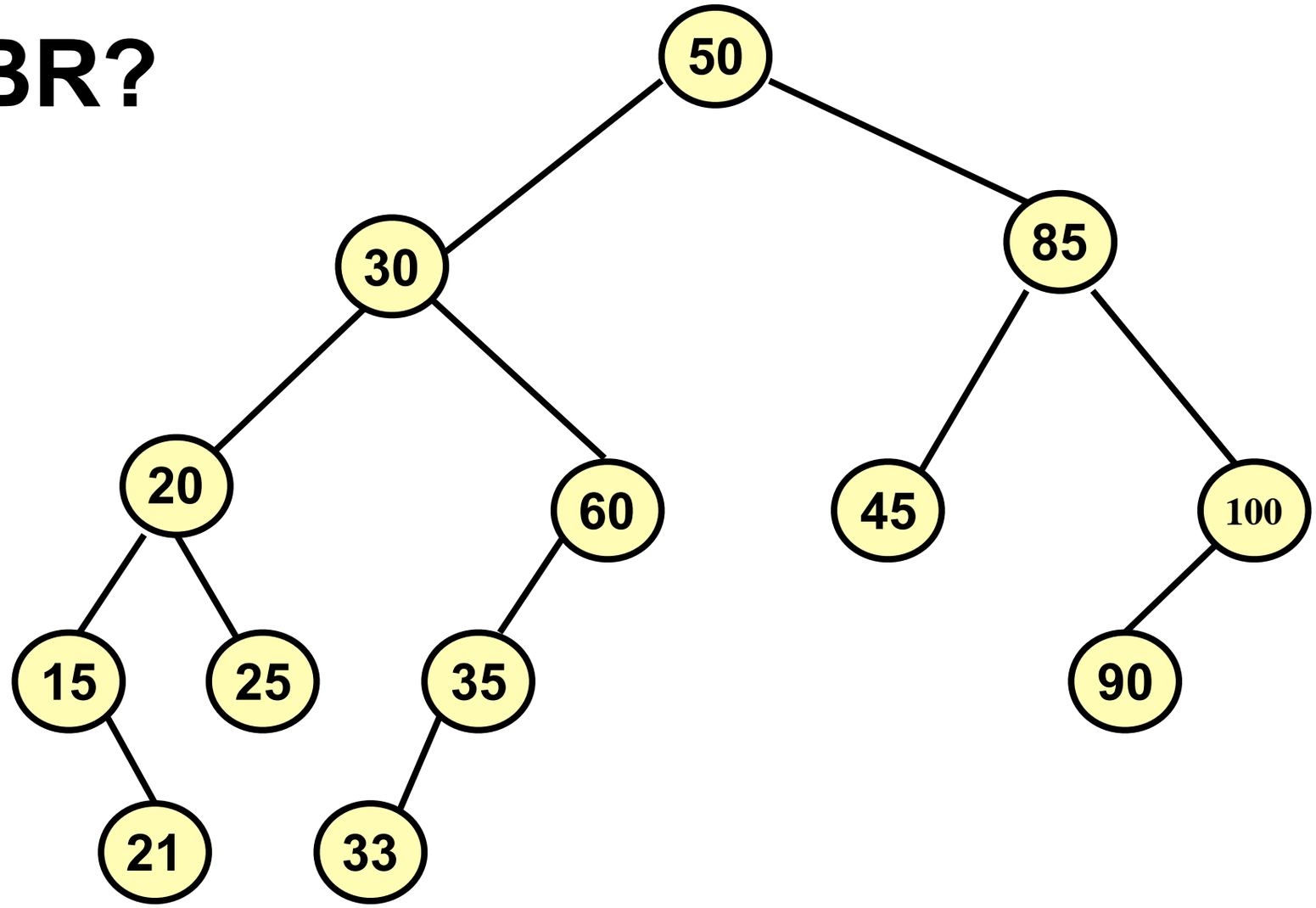
Tipico



Degenere

ABR: esempi

è un ABR?



Capire ABR e maxHeap

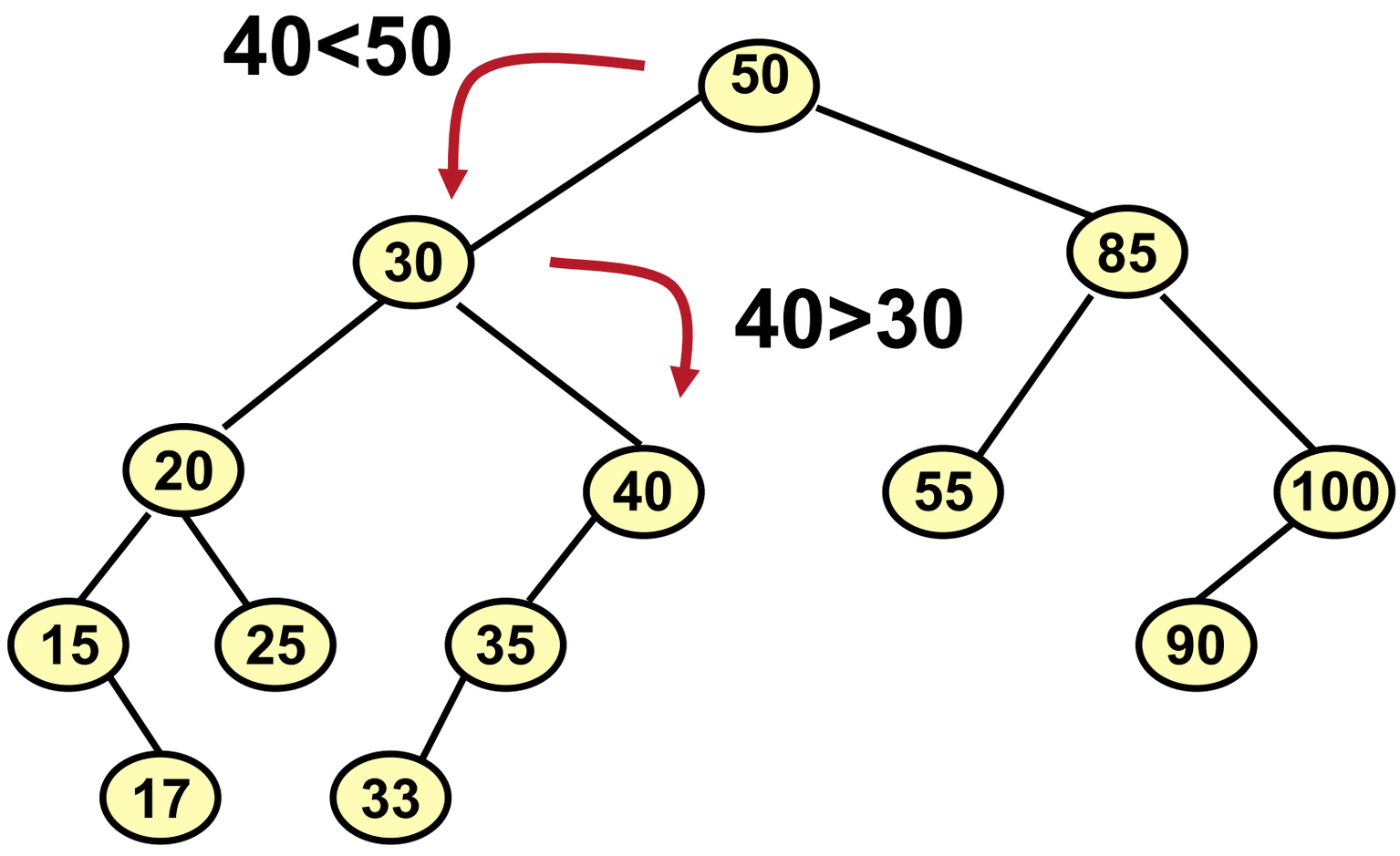
Rispondere alle seguenti domande:

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.**

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.**

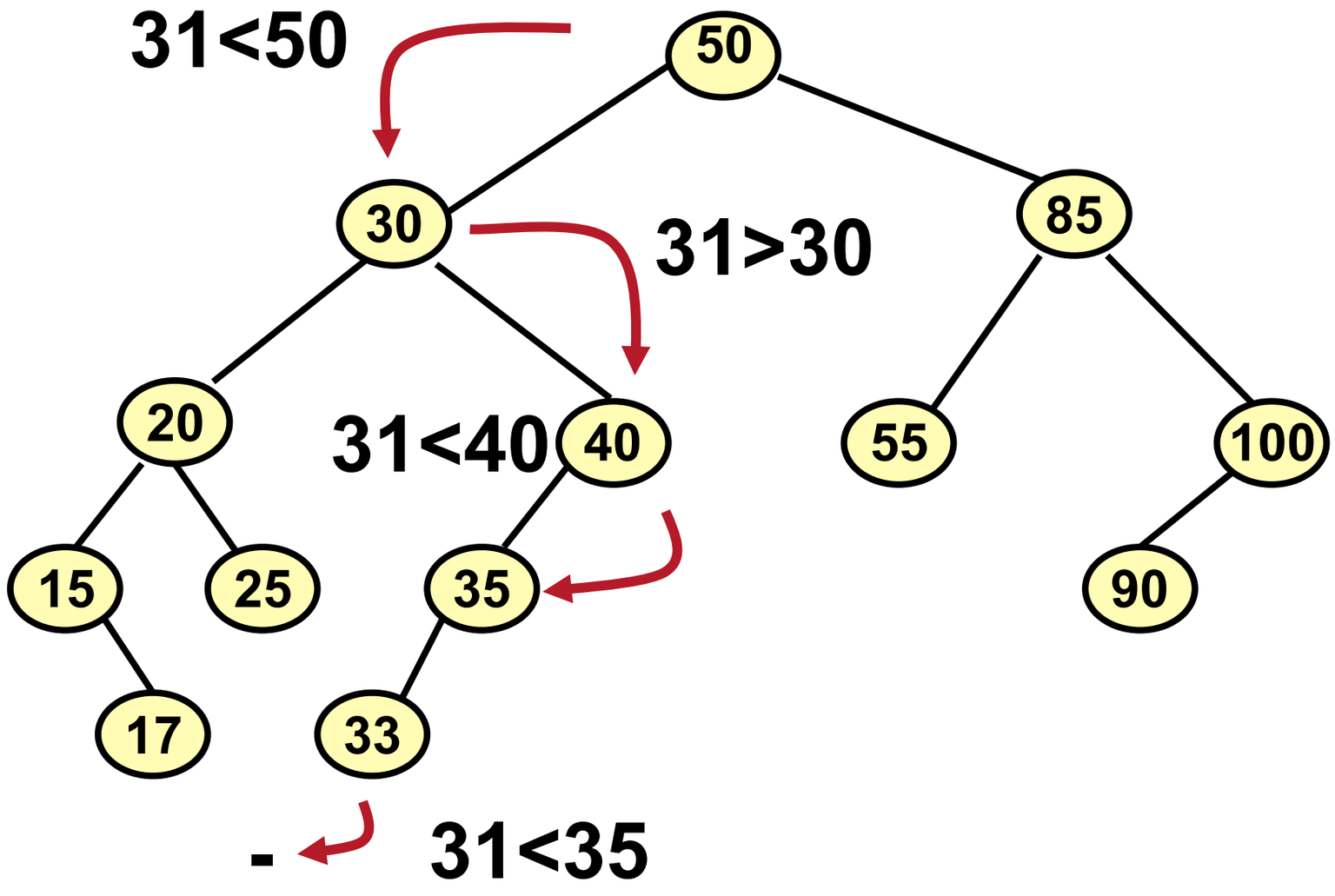
La ricerca con successo

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 al un albero binario e una chiave k

prec: x è un ABR

output: il puntatore 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 la complessità nel caso peggiore si ricava dalla seguente relazione di ricorrenza

$$T(h) = T(h-1) + O(1)$$

La cui soluzione è Θ(h).

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

Quindi complessità Θ(h) o O(n)!
Ma è più informativo dire Θ(h) nel caso peggiore o O(h) in tutti i casi.
In generale non è O(lg n)!!

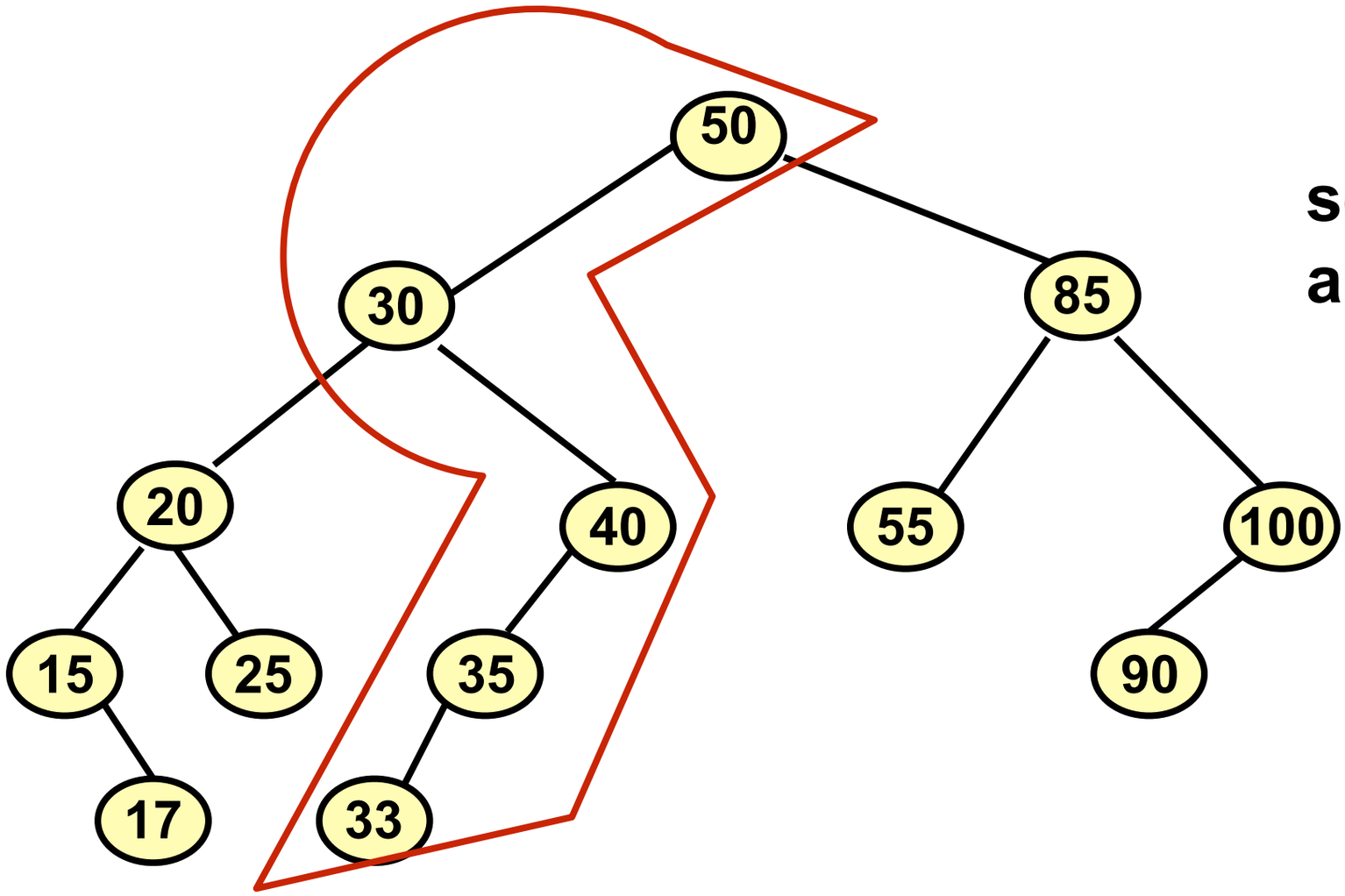
Capire un ABR 2

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;**
- **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.
Il cammino della ricerca è il cammino determinato dai confronti necessari a trovare k nell'albero.

Capire un ABR

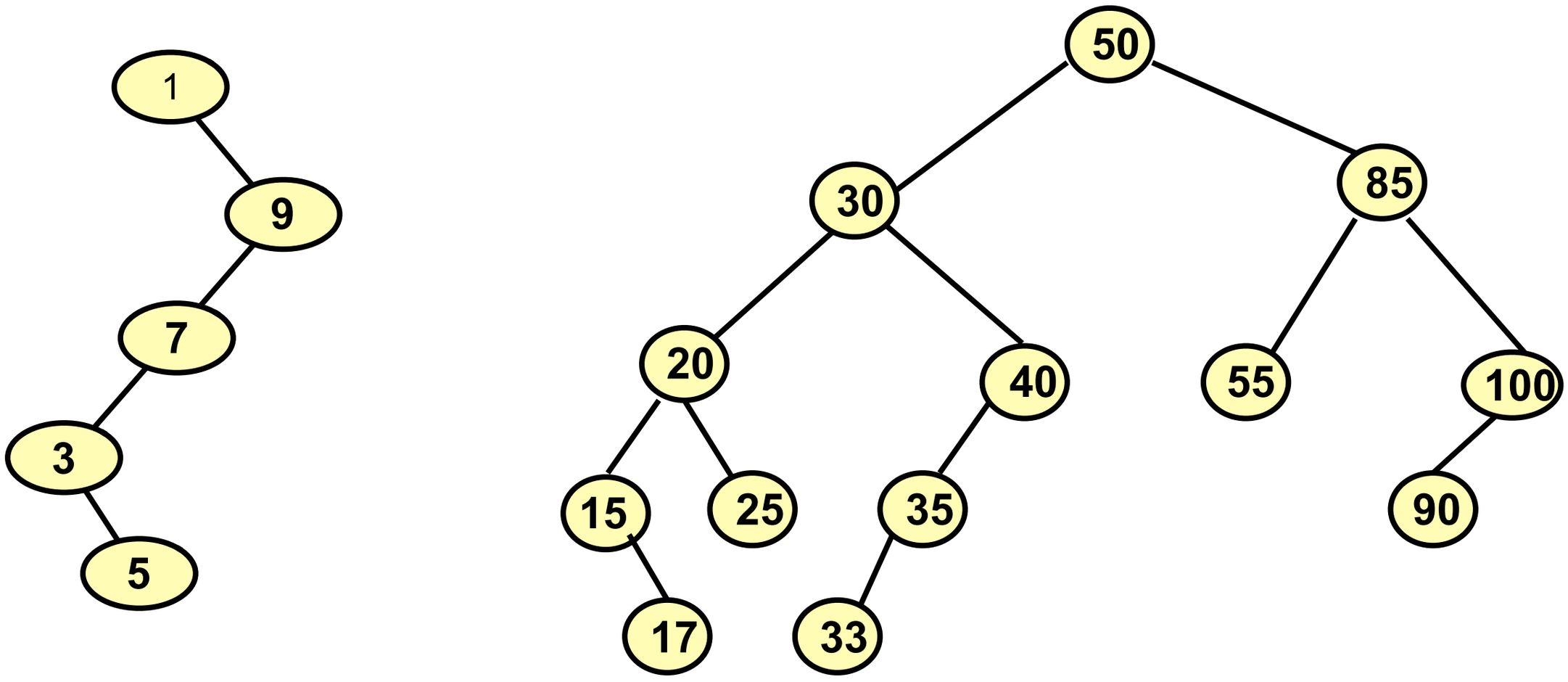


se a ∈ A, b ∈ B e c ∈ C, allora a ≤ b ≤ c.

Ma non è sempre vero! Si produca un esempio che contraddice questa affermazione, specificando gli insiemi A, B e C.

l'insieme delle chiavi alla sinistra del cammino di ricerca
A= {20,15,17,25}
l'insieme delle chiavi del cammino di ricerca
B={50,30,40,35,33}
l'insieme delle chiavi alla destra del cammino di ricerca
A= {85,55,90,100}

ABR: minimo e massimo



Il massimo

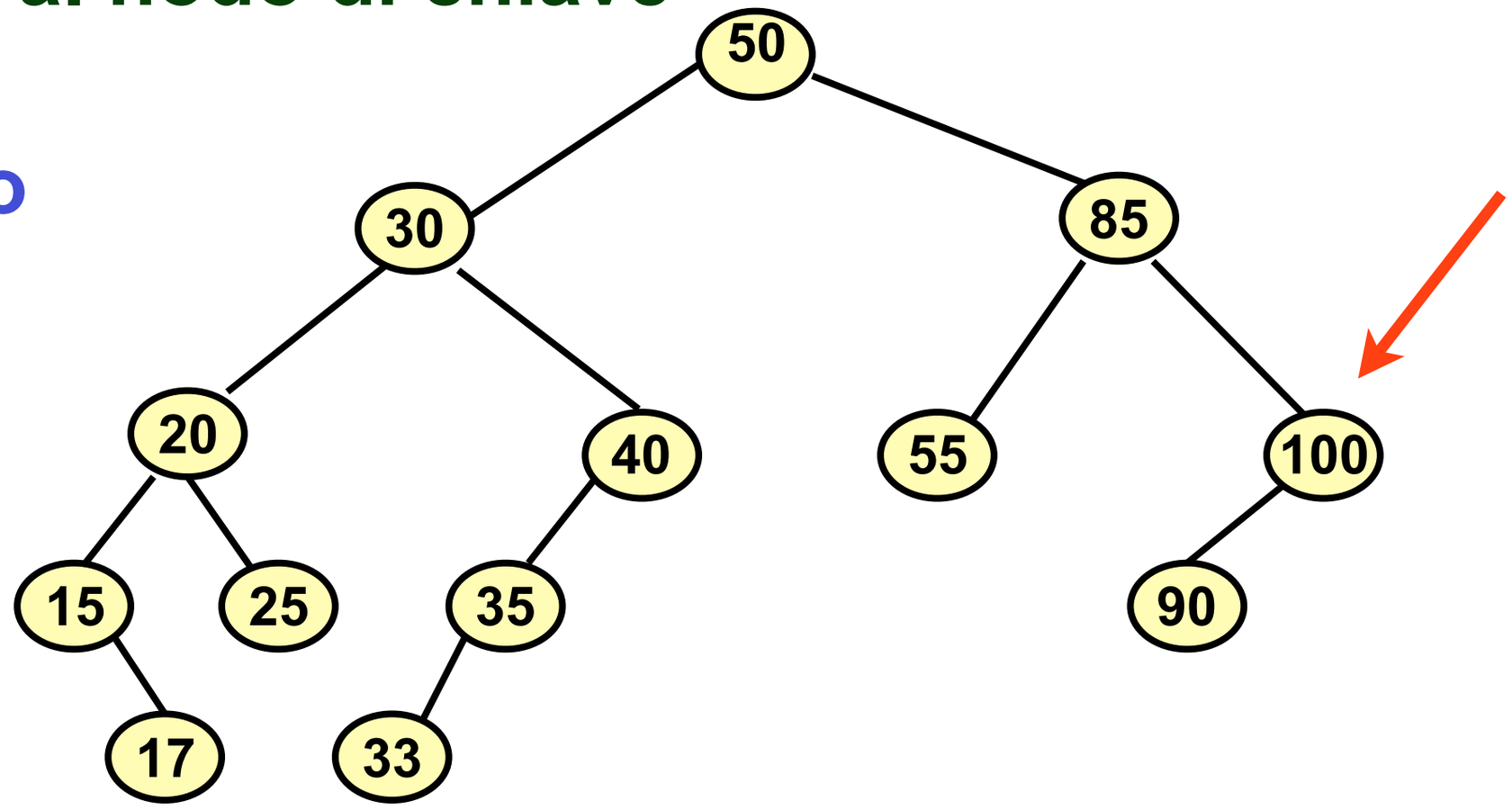
Maximum(x)

input: il puntatore a un albero binario

prec: x ≠ nil e è un ABR

output: 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

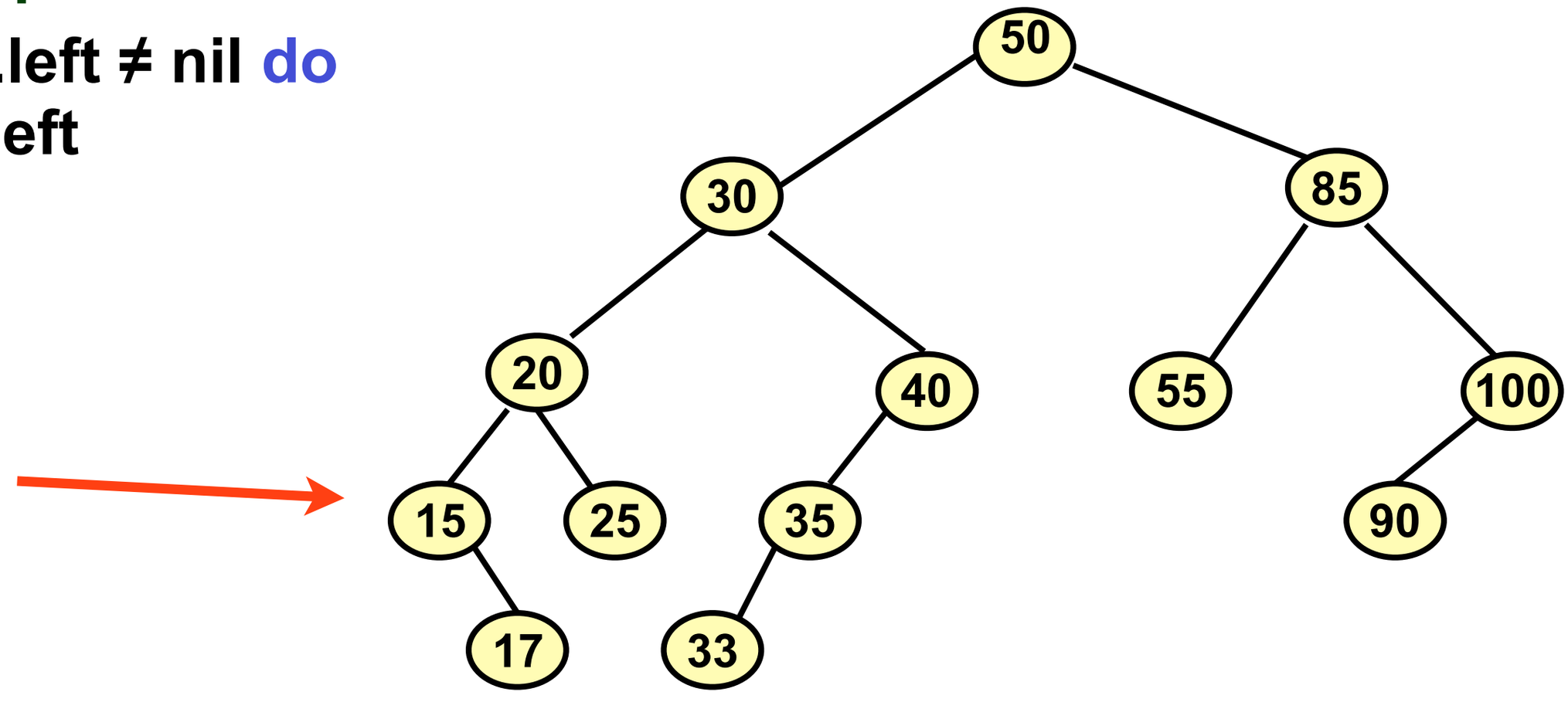
Minimum(x)

input: il puntatore a un albero binario

prec: x ≠ nil e è un ABR

output: 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)

input: il puntatore a un albero binario

prec: x ≠ nil e è un ABR

output: la chiave minima in x

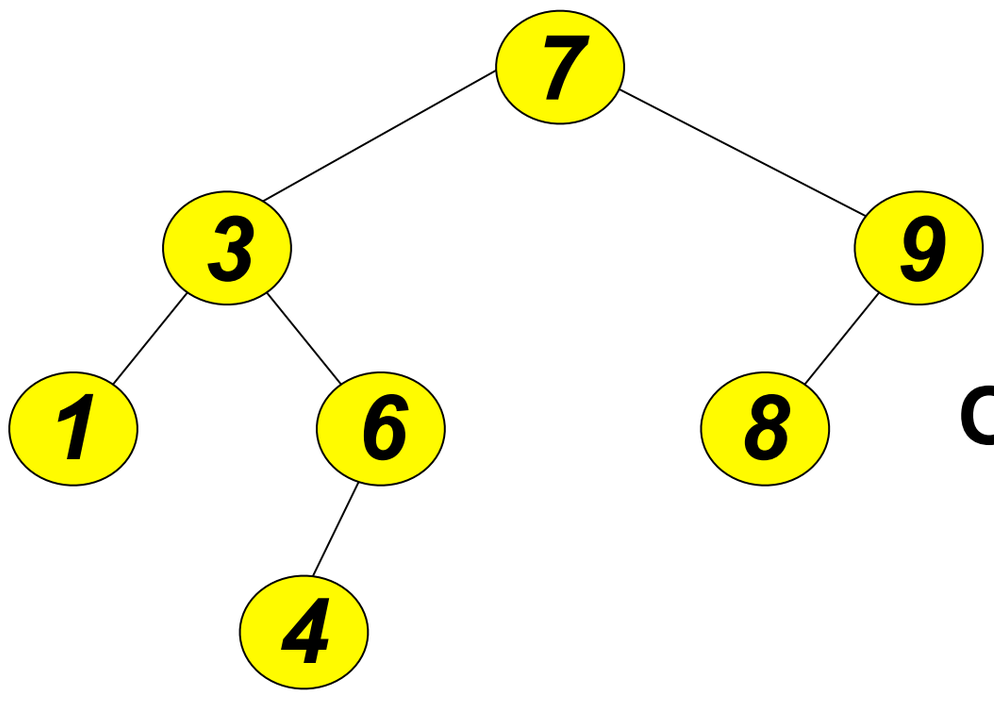
```
if x.left == NIL
        then return x.key
         else return MinimumRic(x.left)
```

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

Visita inorder di un ABR

```
Inorder-Tree-Walk(x)
input: il puntatore a un albero binario
prec: x è un ABR
output: stampa le chiavi in x in ordine crescente
 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

L'inserimento

Insert(T,z)

input: T è unpuntatore a un albero binario e z a un nuovo nodo da inserire

prec: z ≠ nil, è un ABR e z è un nodo non nullo e foglia, cioè tale che z.left=z.right=nil, non presente in T.

output: 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)

input: T è un puntatore a un albero binario e z a un nuovo nodo da inserire

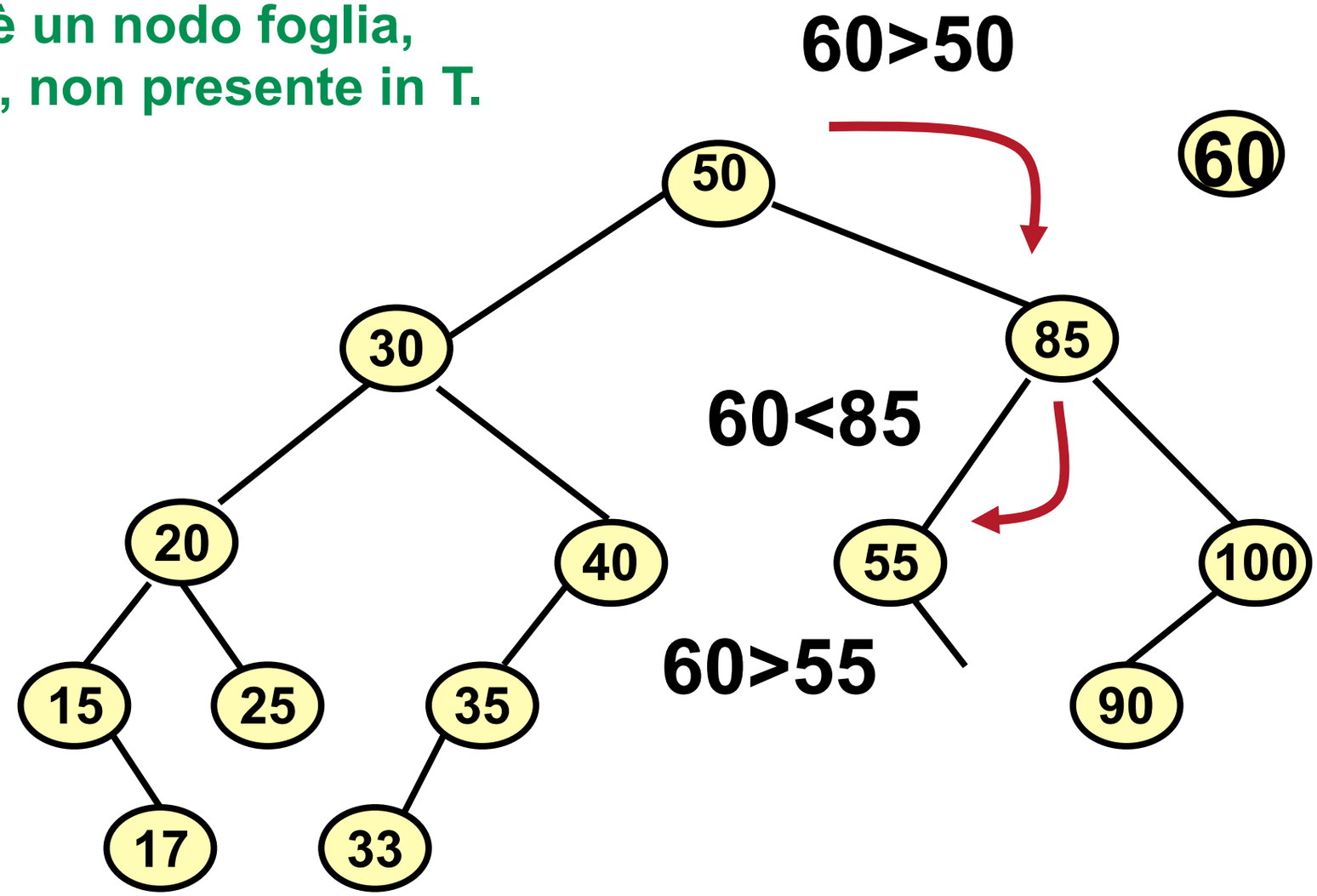
prec: T ≠ nil, T è un ABR e z è un nodo foglia, cioè tale che z.left=z.right=nil, non presente in T.

T ≠ NIL e z ≠ nil

output: 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)
```

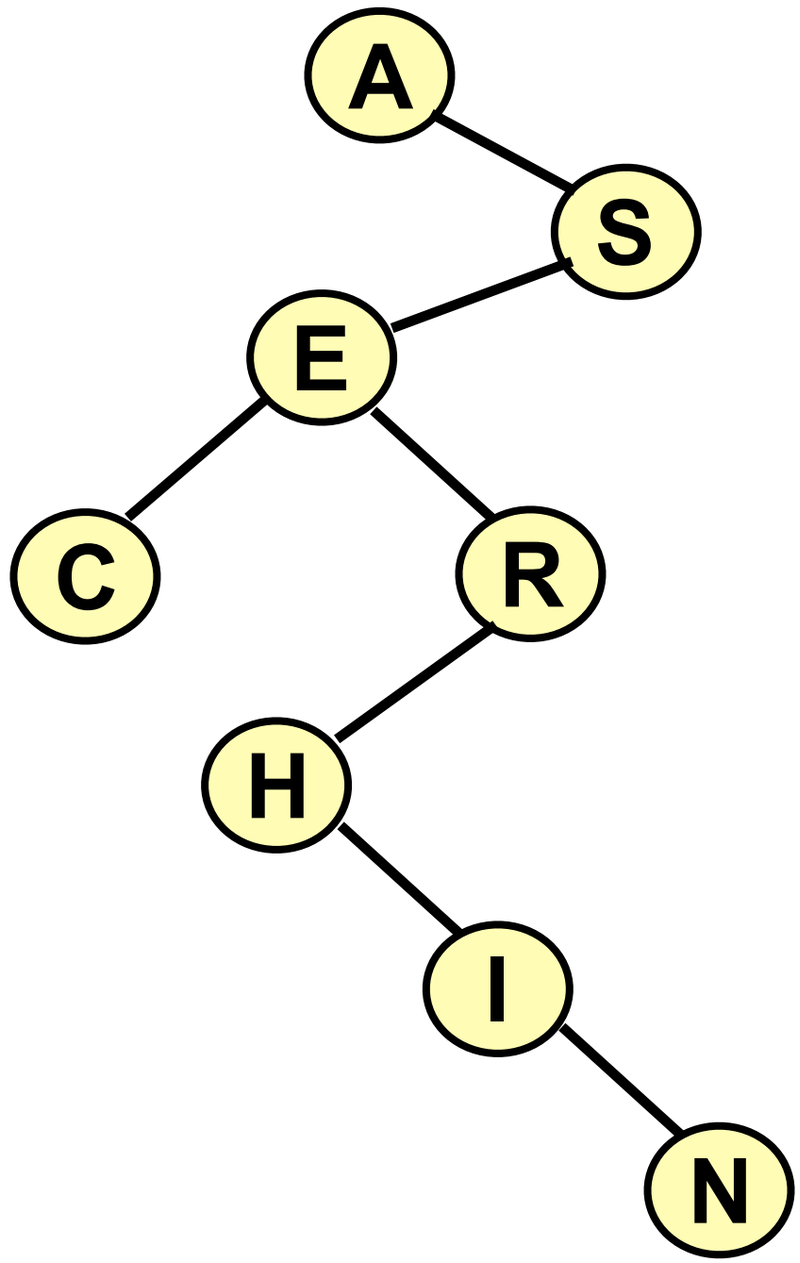
Inseriamo z = 60



Complessità? O(h)

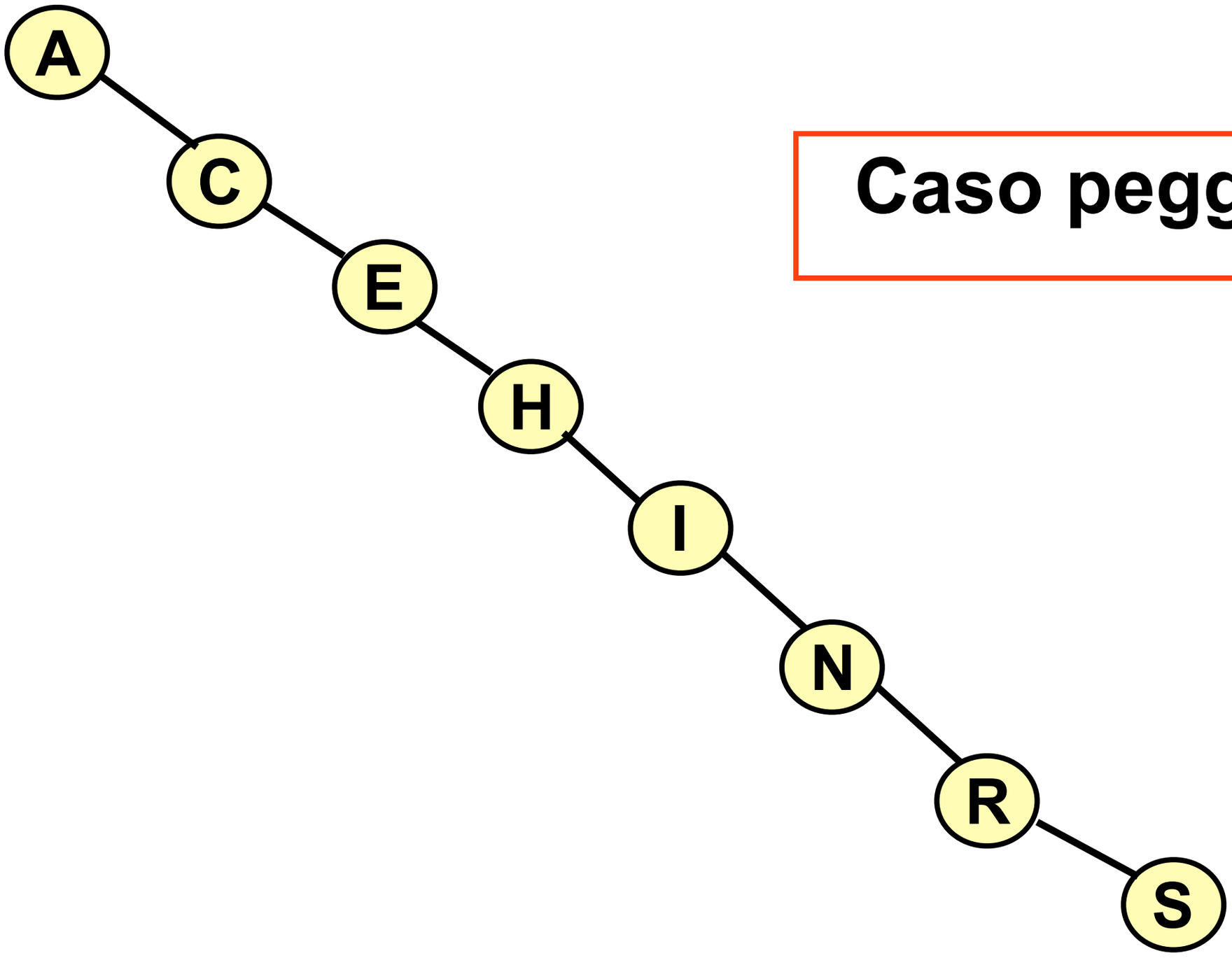
Forma dell'ABR

Inseriamo A S E R C H I N



Forma dell'ABR

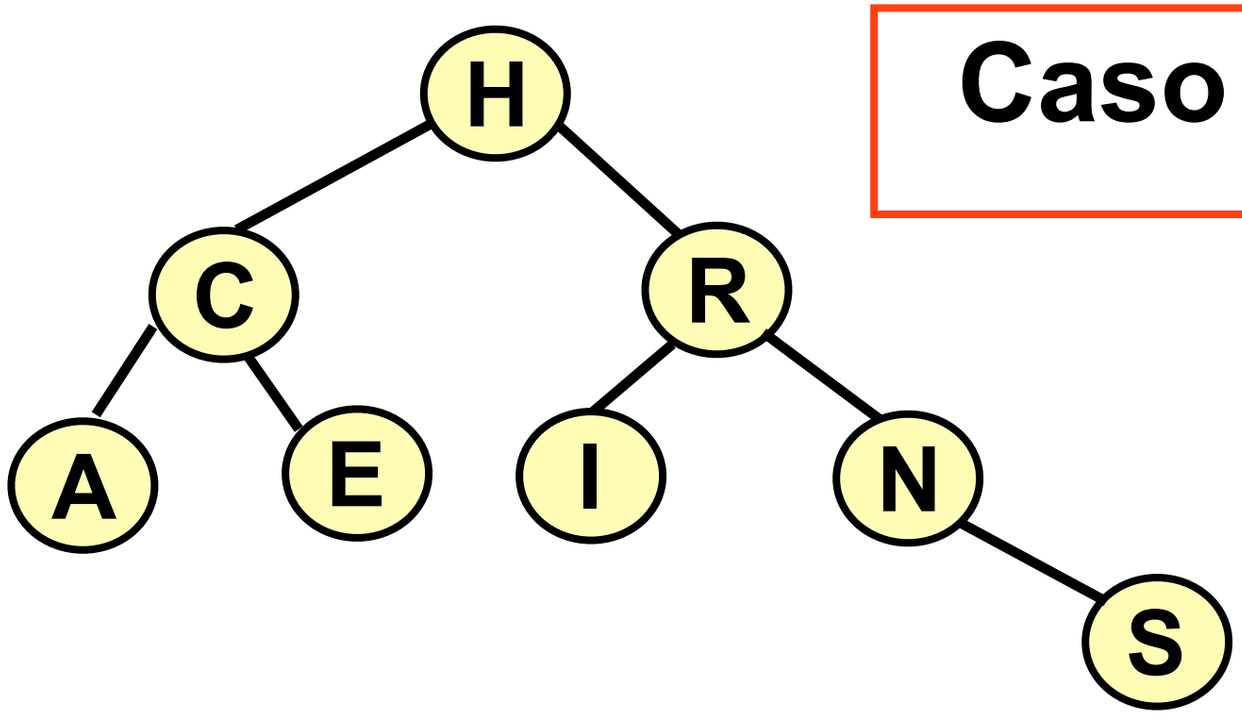
Inseriamo A C E H I N R S



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

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

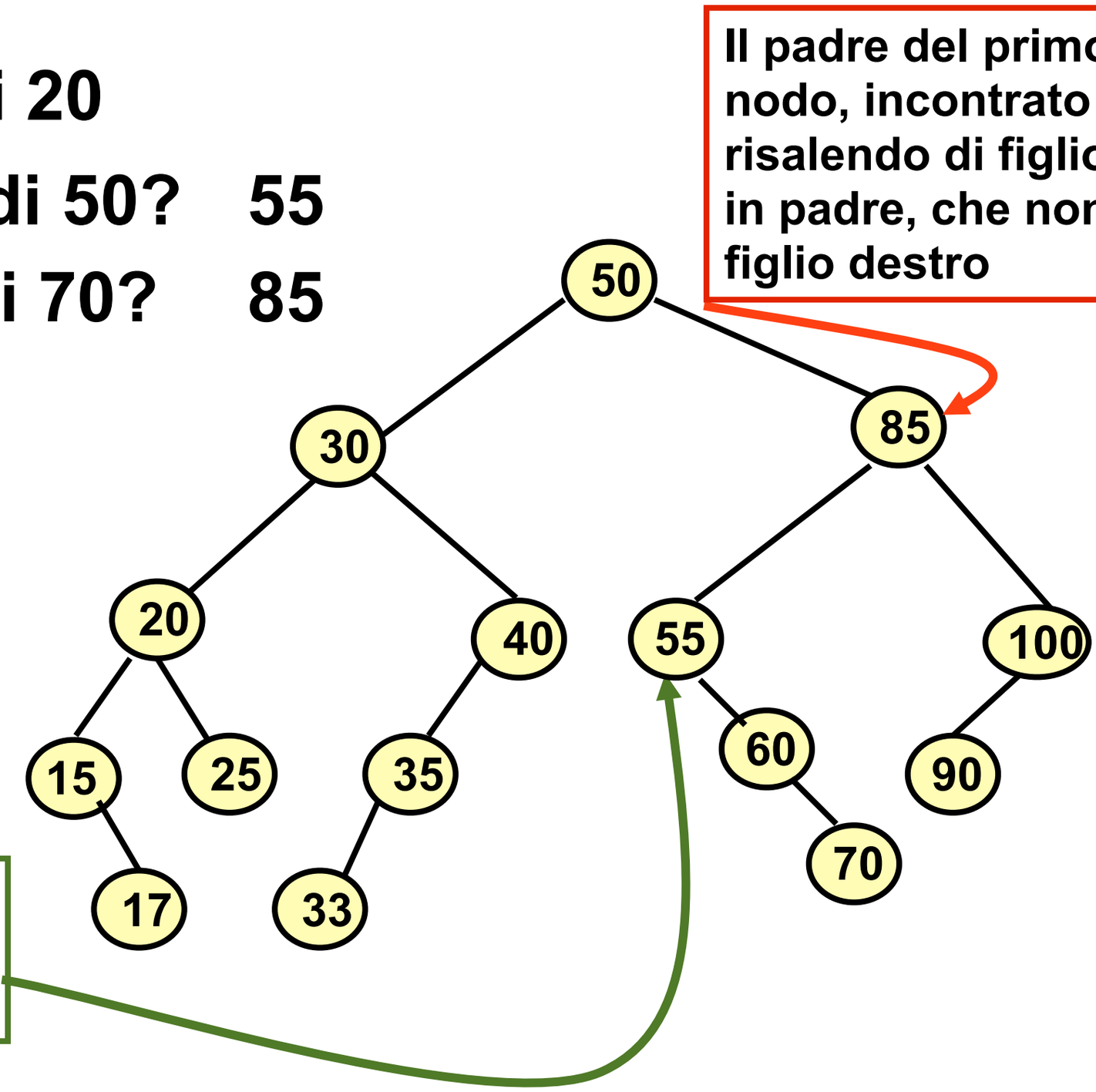
25 è il successivo di 20

Chi è il successivo di 50? 55

Chi è il successivo di 70? 85

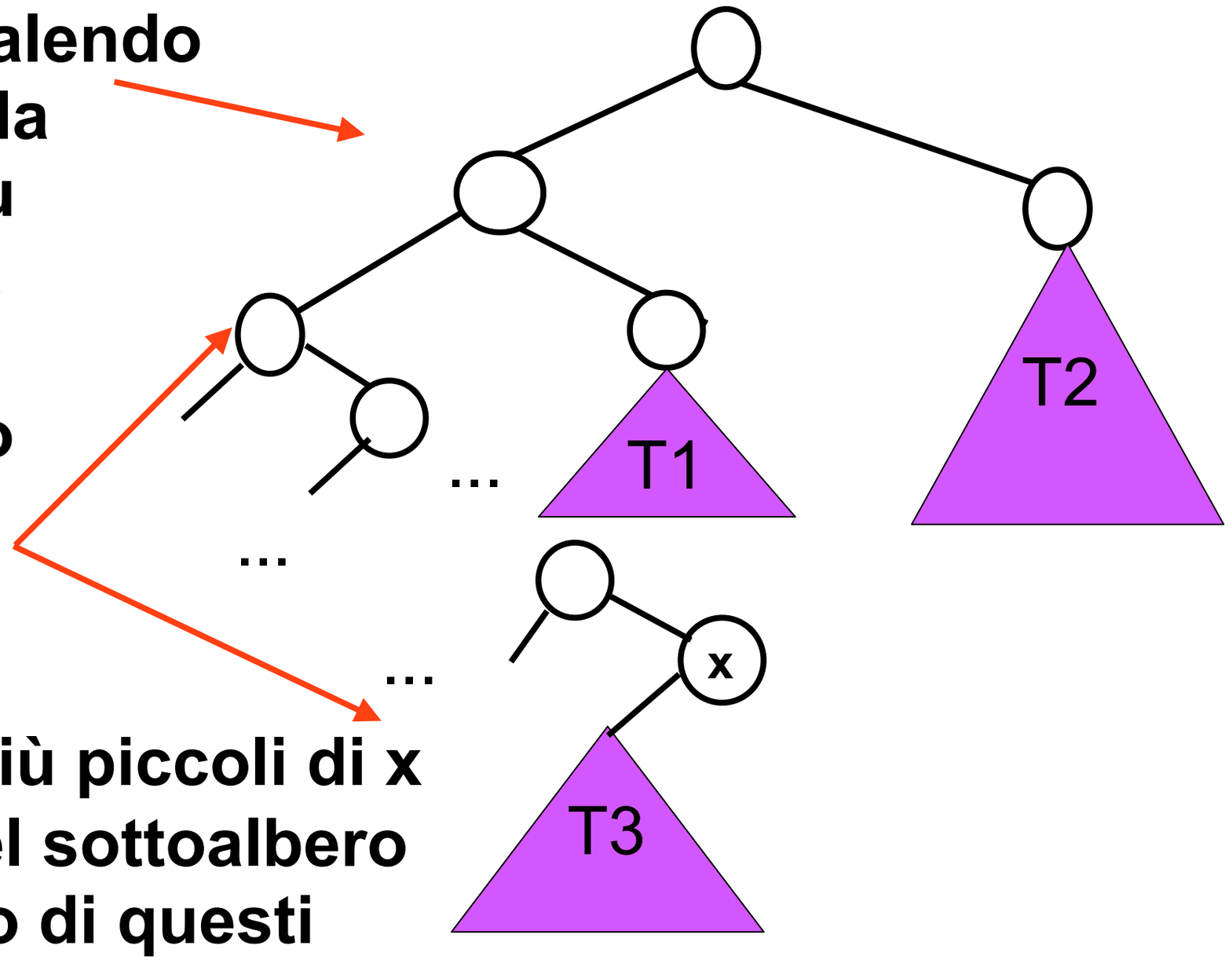
La chiave successiva di una data chiave x nell'albero è

quella del primo nodo, risalendo dal nodo verso la radice, che ha x nel sotto albero sinistro.



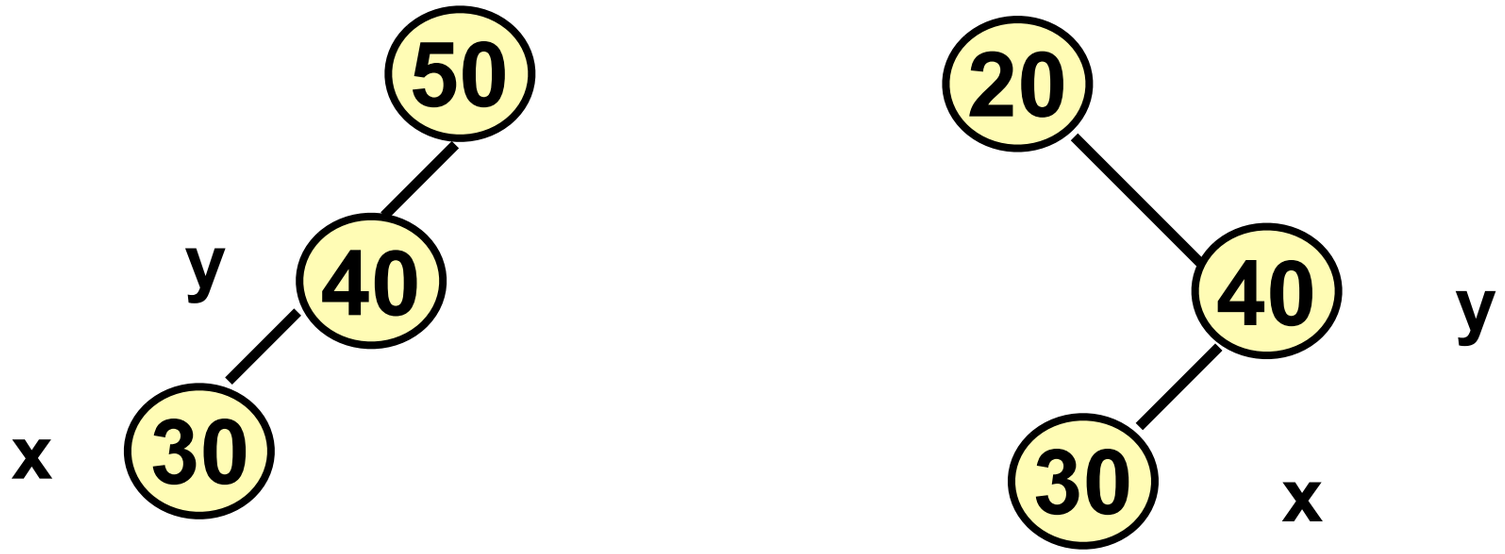
Successivo di un nodo di chiave x senza figlio destro

il primo risalendo da x verso la radice è più grande di x ha x nel sottoalbero sinistro!



tutti più piccoli di x x è nel sottoalbero destro di questi nodi

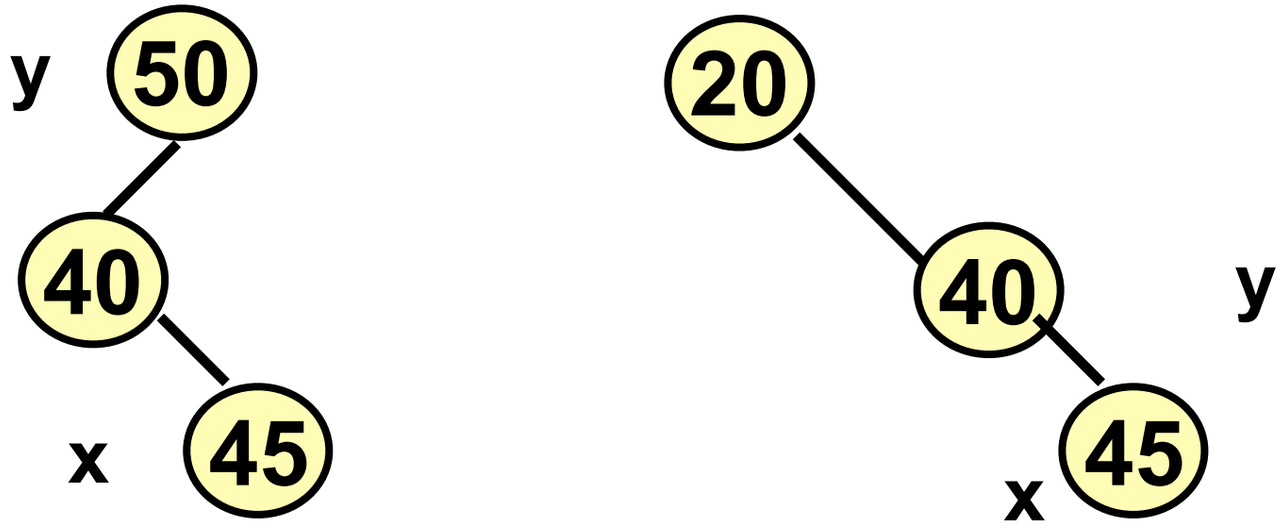
E’ sempre vero che il successivo di una foglia è suo padre?



Se x è figlio sinistro di y, vuol dire che la sua chiave è minore di quella del padre, d’altro canto il padre è il primo nodo risalendo verso la radice nel cui sottoalbero sinistro si trova x, quindi è il minore tra i più grandi, cioè il successivo come illustrato in figura.

Quindi il successivo di una foglia figlio sinistro è il padre.

E’ sempre vero che il successivo di una foglia è suo padre?



Se x è figlio destro di y, vuol dire che la sua chiave è maggiore di quella del padre, quindi il successivo di x si trova tra gli antenati, in particolare il primo che ha x nel sotto albero sinistro, risalendo da x verso la radice, come per tutti i nodi senza figlio destro.

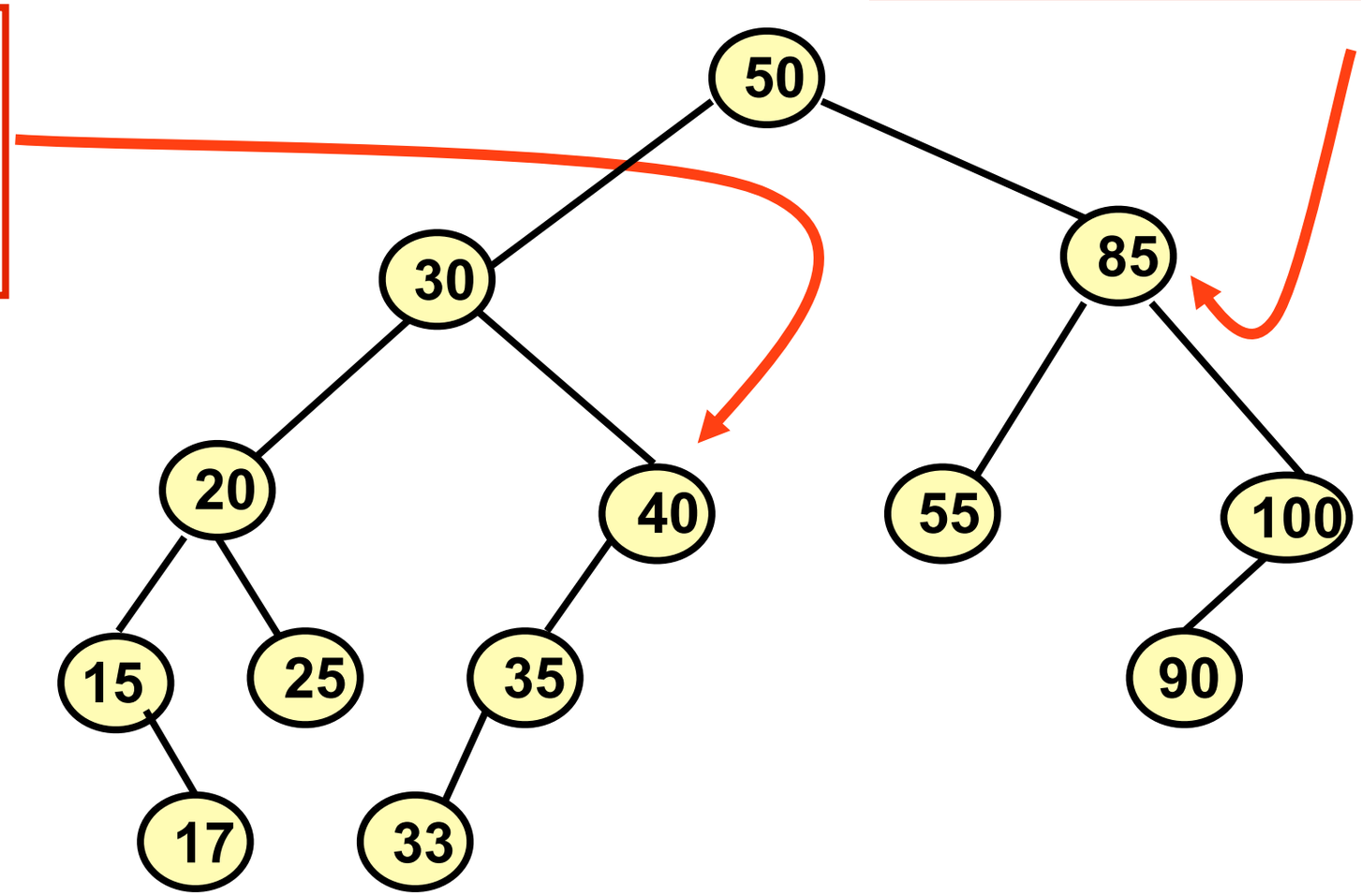
Precedente

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

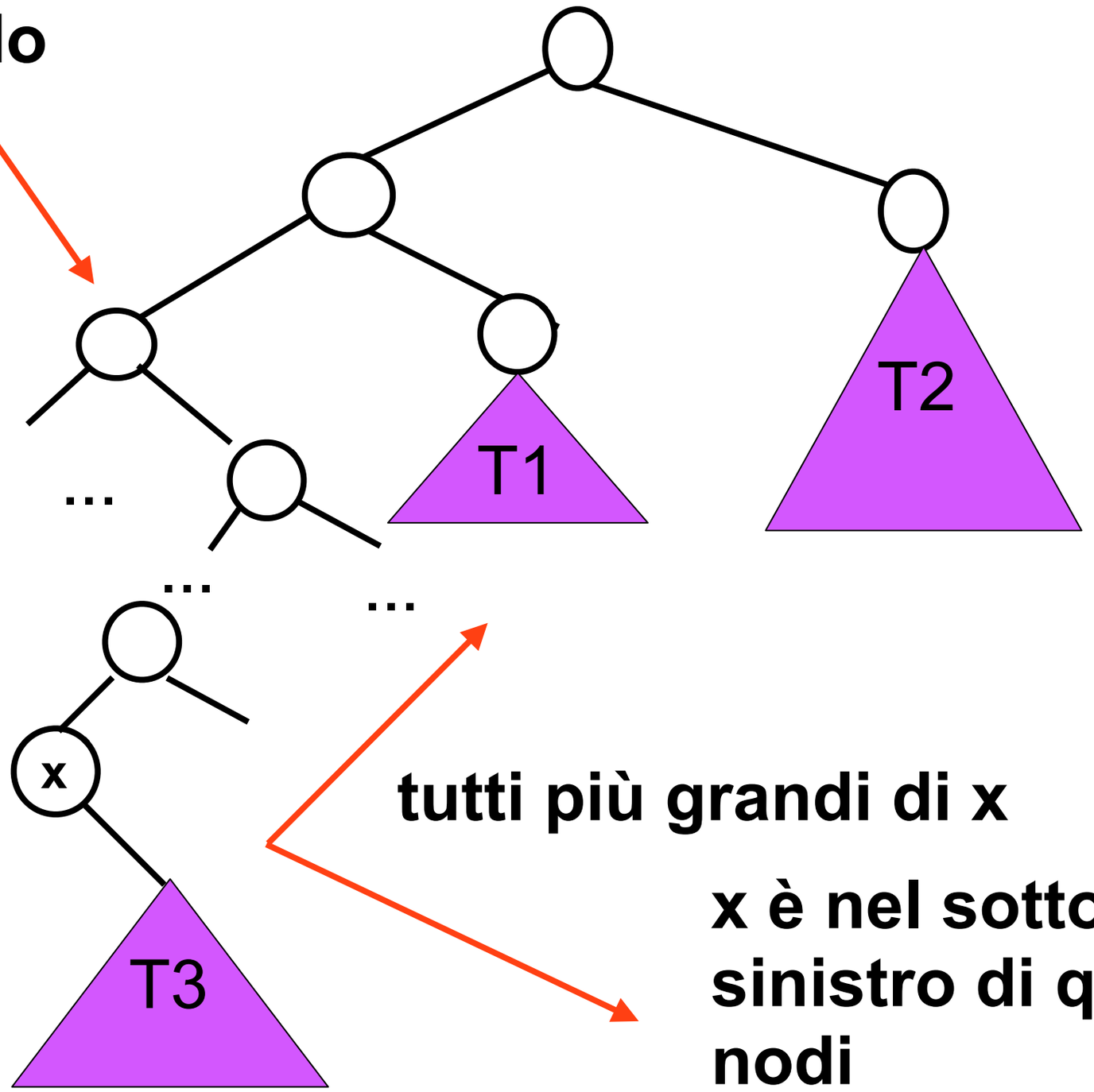


Chi è il precedente di 90? **85**

Precedente di un nodo di chiave x senza figlio sinistro

il primo risalendo da x verso la radice e più piccolo di x

il primo ad avere x nel sottoalbero destro!



tutti più grandi di x

x è nel sottoalbero sinistro di questi nodi

precedente di una foglia

Se x è figlio destro di y, vuol dire che la sua chiave è maggiore di quella del padre, d'altro canto il padre è il primo nodo risalendo verso la radice nel cui sottoalbero destro si trova x, quindi è il maggiore tra i più piccoli, cioè il precedente come illustrato in figura.

Quindi il precedente di una foglia figlio destro è il padre

Il precedente

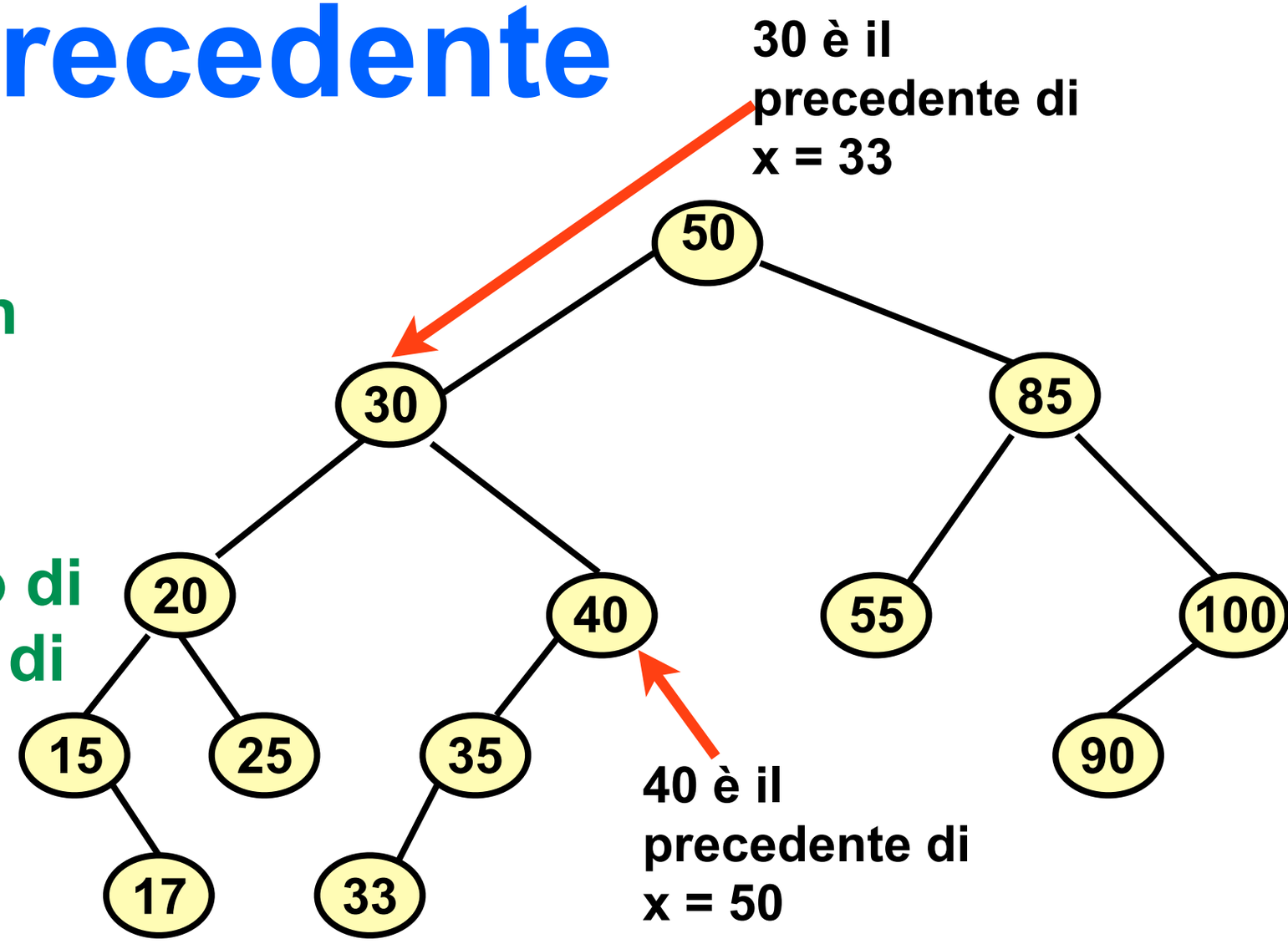
Predecessor(x)

input: x è un puntatore a un nodo in albero binario

prec: l'albero è un ABR

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

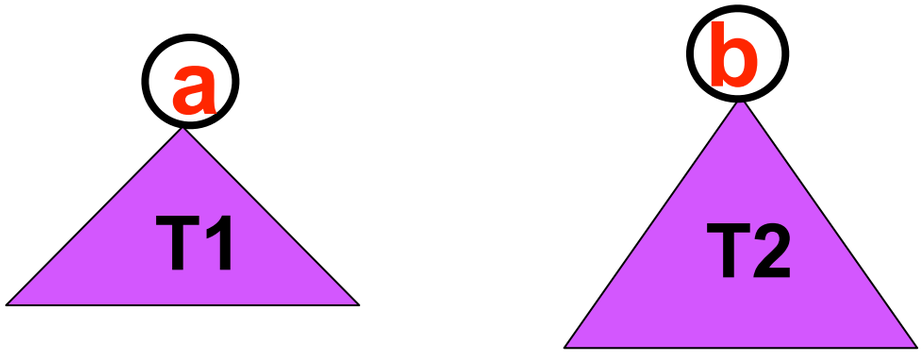
```
if x.left ≠ nil then
    return Maximum(x.left)
y = x.p
  while y ≠ nil and x == y.left
       do x = y
          y = x.p
  return y
```



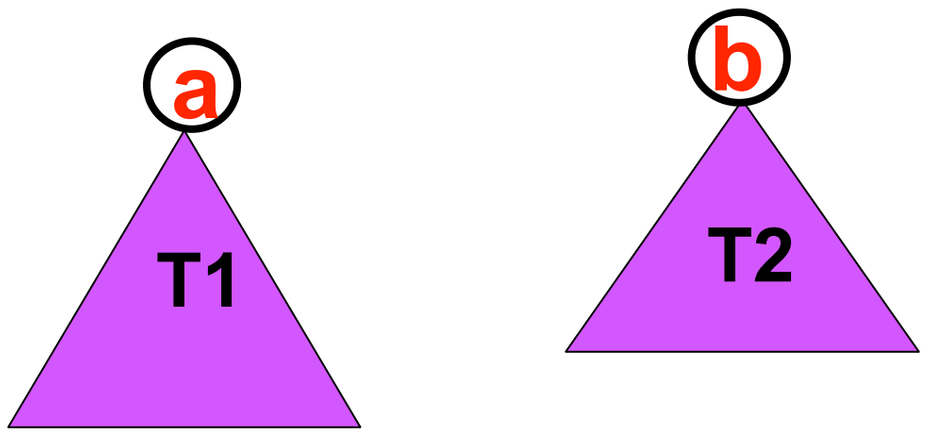
Complessità O(h)

Capire un ABR

Dati due alberi binari di ricerca T1 e T2, supponiamo che le chiavi in T1 siano tutte minori di quelle in T2 e che la radice contenga un campo aggiuntivo con l'altezza dell'albero. Si scriva un algoritmo per ottenere un albero binario di ricerca su tutte le chiavi in T1 e in T2 in O(h), dove h è l' altezza minima tra quelle dei due alberi. L'albero ottenuto deve avere altezza pari a quella del sotto albero di altezza massima +1.

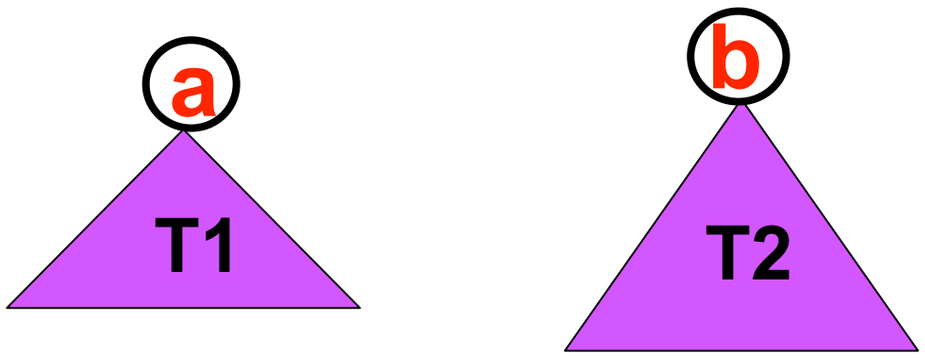


Qui h1, l'altezza di T1, è minore di h2, l'altezza di T2.



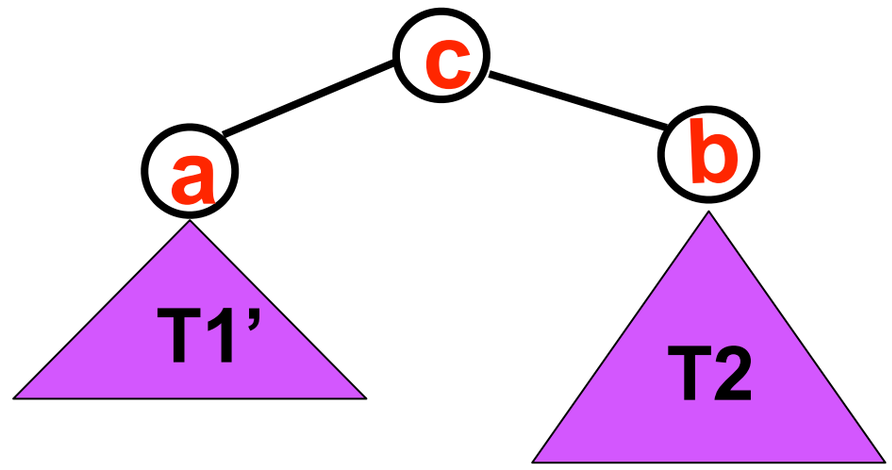
Qui h2, l'altezza di T2, è minore di h1, l'altezza di T1.

Capire un ABR - sol

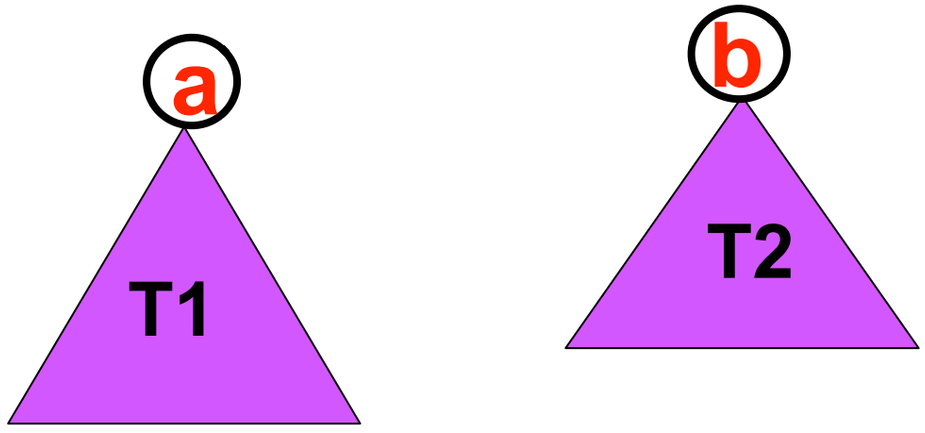


Sappiamo che le chiavi in T1 sono tutte minori di quelle in T2. Se h1< h2:

prendiamo il massimo in T1, c, e lo togliamo da T1. Sia T1' l'albero ottenuto da T eliminando c.
Creiamo un nodo radice con chiave c.
Le due radici di T1 e T2 diventano i figli sinistro e destro, rispettivamente, della nuova radice.
Infatti tutte le chiavi di T1' sono minori di c e per ipotesi tutte le chiavi di T2 sono maggiori di c.
Tempo di esecuzione O(h1), per il minimo, più O(1) per le altre operazioni.



Capire un ABR - sol



Se invece h2, l'altezza di T2 è minore di h1, l'altezza di T1, procediamo diversamente.

Prendiamo il minimo in T2, c, e lo togliamo da T2. Sia T2' l'albero ottenuto da T eliminando c. Creiamo un nodo radice con chiave c.
Le due radici di T1 e T2 diventano i figli sinistro e destro, rispettivamente, della nuova radice.
Infatti tutte le chiavi di T1 sono minori di c per ipotesi e tutte le chiavi di T2 sono maggiori di c.
Tempo di esecuzione O(h2), per il minimo, più O(1) per le altre operazioni.