In questa lezione

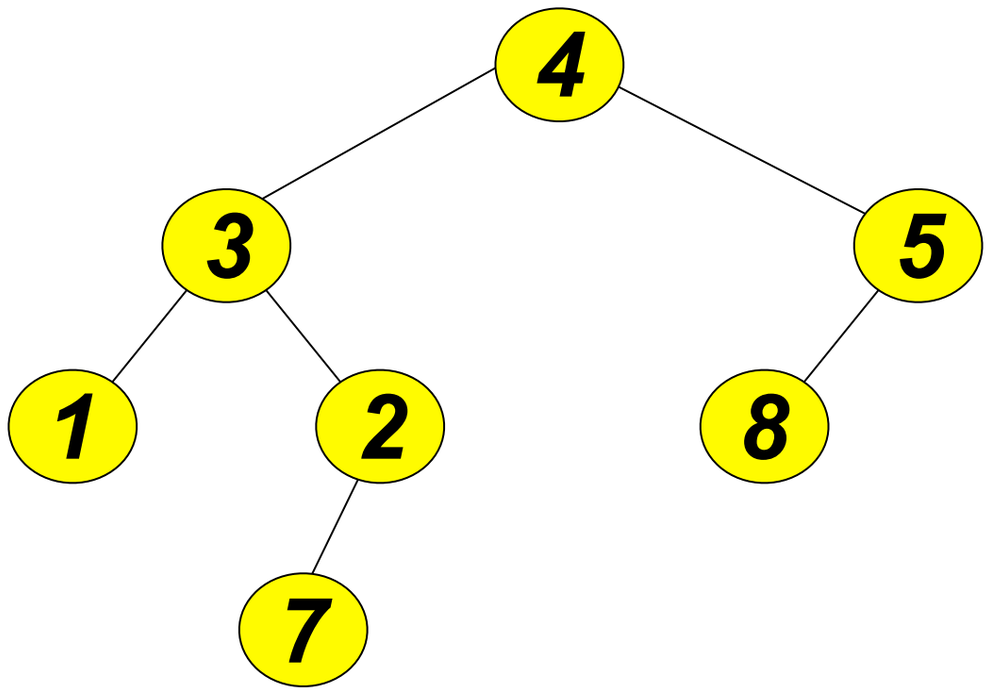
- **Alberi binari: visite**
- **Alberi binari di ricerca**

Visita inorder di un albero binario

visita inorder(x)
se l'albero x non è nullo allora
visita inorder il sottoalbero sinistro di x
visita x
visita inorder il sottoalbero destro di x

La radice di ogni sotto albero deve essere visitata dopo il suo sotto albero sinistro e prima di quello destro.

Input:



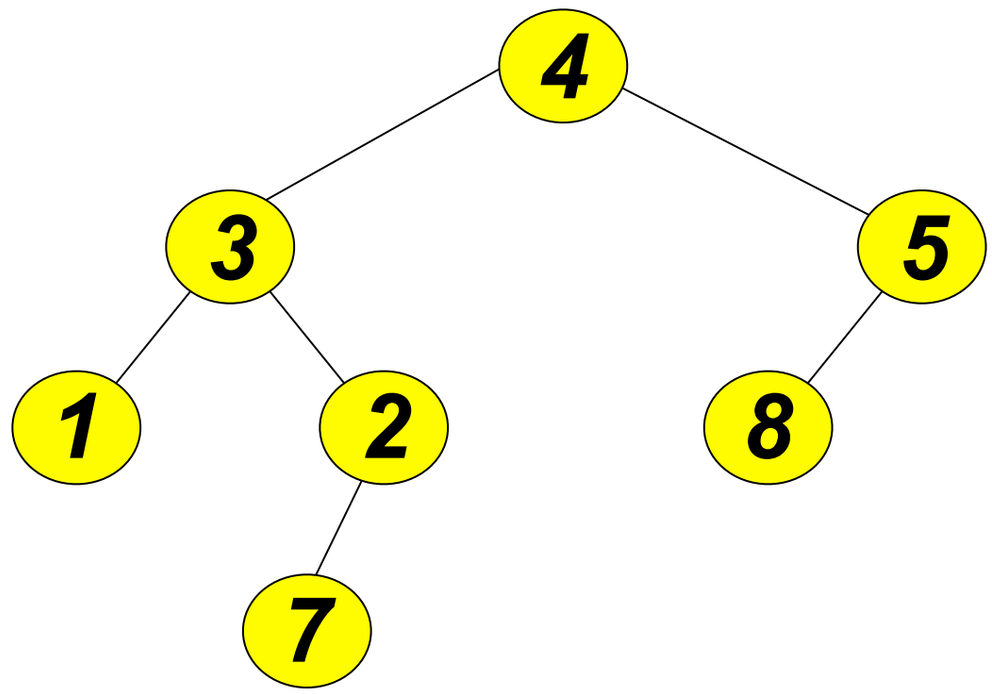
Output:
1
3
7
2
4
8
5

Visita inorder di un albero binario

Il nome ricorda l'ordine di visita della radice rispetto ai sotto alberi: in, cioè tra i due.

```
Inorder(x)
 if x ≠ nil then
      Inorder(x.left)
      print key[x]
      Inorder(x.right)
```

Input:



Inorder(4)

Inorder(3)

Inorder(1)

Inorder(nil)

print key(1)

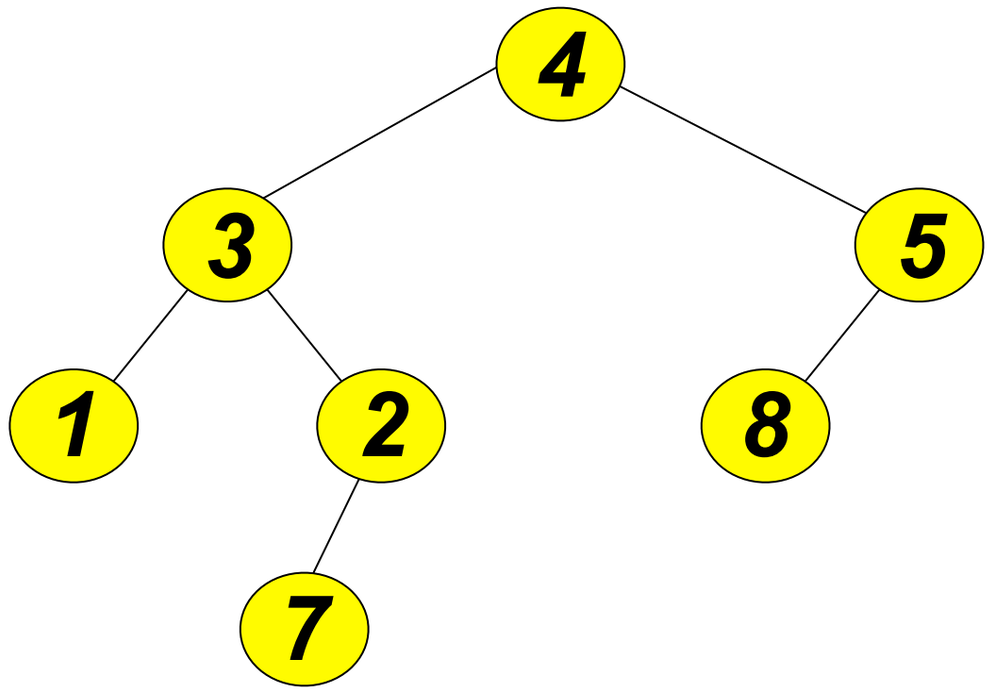
print key(3)

Output: 1 3

Visita inorder

```
Inorder(x)
 if x ≠ nil then
      Inorder(x.left)
      print key[x]
      Inorder(x.right)
```

Input:



Inorder(4)

Inorder(3)

Inorder(2)

Inorder(7)

print key(7)

print key(2)

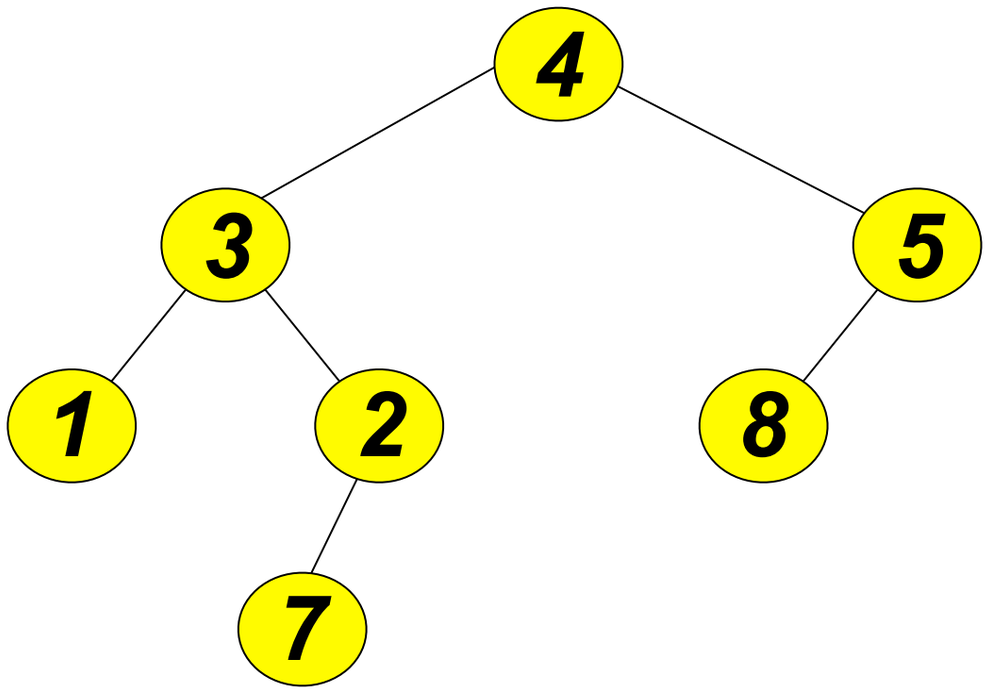
print key(4)

Output: 1 3 7 2 4

Visita inorder

```
Inorder(x)
 if x ≠ nil then
      Inorder(x.left)
      print key[x]
      Inorder(x.right)
```

Input:




```
Inorder(4)
     Inorder(5)
     Inorder(8)
     print key(8)
     print key(5)
```

Output: 1 3 7 2 4 8 5

Tempo di esecuzione inorder

Sia n il numero dei nodi dell'albero binario e k quello del suo sottoalbero sinistro allora la relazione di ricorrenza è

$T(0) = c$
$T(n) = T(k) + T(n-k-1) + d \quad (c,d >0)$

```
Inorder(x)
 if x ≠ nil then
      Inorder(x.left)
      print key[x]
      Inorder(x.right)
```

Ipotizziamo che T(n) = O(n).
Verifichiamo che T(n) ≤ An, per una costante A>0, a partire da un certo n in poi.

$T(n) = T(k) + T(n-k-1) + d$
$\leq Ak + A(n-k-1) + d$
$= Ak + An - Ak - A + d = An - A + d$

si vuole che An - A + d ≤ An questo è vero per A≥d.
Guardiamo al caso base T(1) = T(0) + T(0) + d = 2c+d e scegliamo quindi A = 2c+d, per concludere che T(n) = O(n) perché abbiamo trovato una costante A tale che T(n) ≤ An per ogni n≥0.

Quindi T(n) = O(n)

Limite inferiore inorder

```
Inorder(x)
 if x ≠ nil then
      Inorder(x.left)
      print key[x]
      Inorder(x.right)
```

Poichè T(n) = Ω(n), perchè tutti i nodi sono visitati.

$$T(n) = \Theta(n)$$

Altre visite

```
Inorder(x)
 if x ≠ nil then
      Inorder(x.left)
      print key[x]
      Inorder(x.right)
```

```
Postorder(x)
 if x ≠ nil then
   Postorder(x.left)
   Postorder(x.right)
   print key[x]
```

definiamo altre due visite modificando l'ordine di visita di della radice (di ogni sotto albero) rispetto ai suoi sotto alberi.

Visita postOrder :
visita il sottoalbero sinistro
visita il sottoalbero destro
visita la radice

Visita preOrder :
visita la radice
visita il sottoalbero sinistro
visita il sottoalbero destro

```
Preorder(x)
if x ≠ nil then
  print key[x]
  Preorder(x.left)
  Preorder(x.right)
```

$T(n) = \Theta(n)$

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?

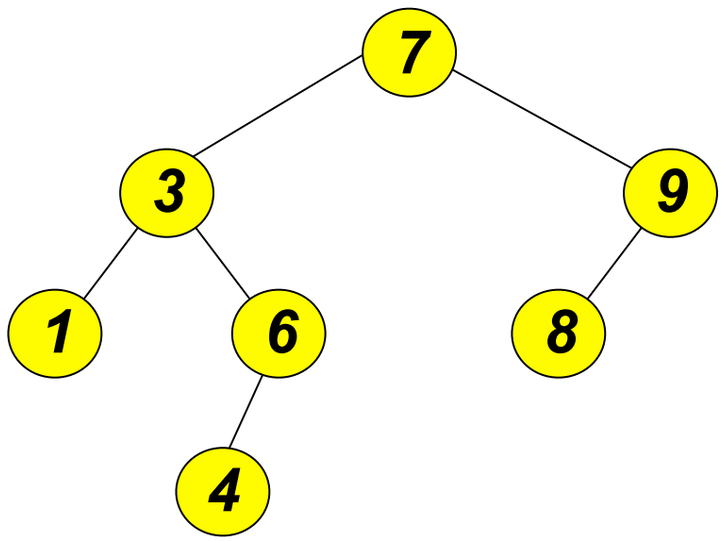
Alberi binari di ricerca

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

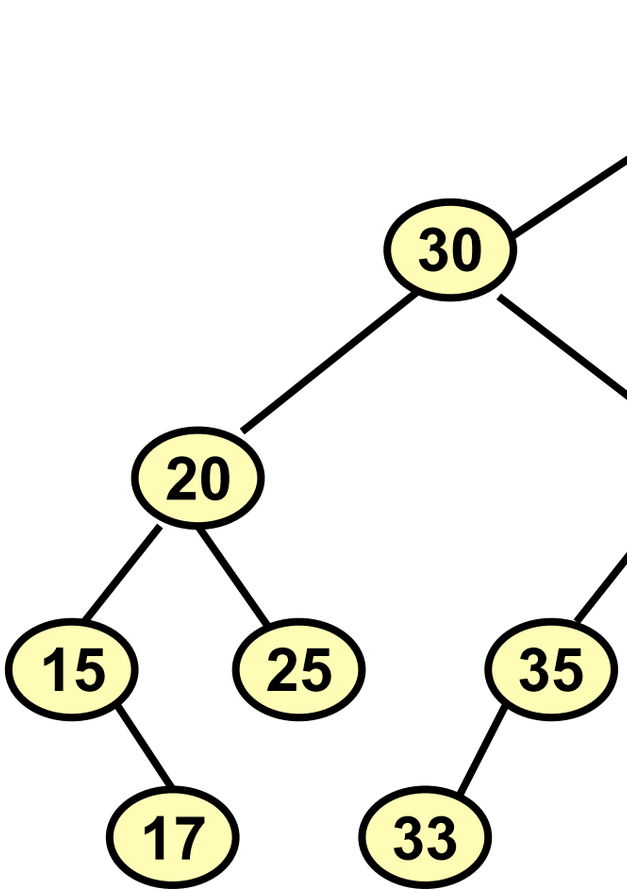
- **le chiavi dei nodi nel sottoalbero sinistro di v sono minori o uguali alla chiave di v e**
- **le chiavi dei nodi nel sottoalbero destro di v sono maggiori o uguali alla chiave di v.**

Esempio:

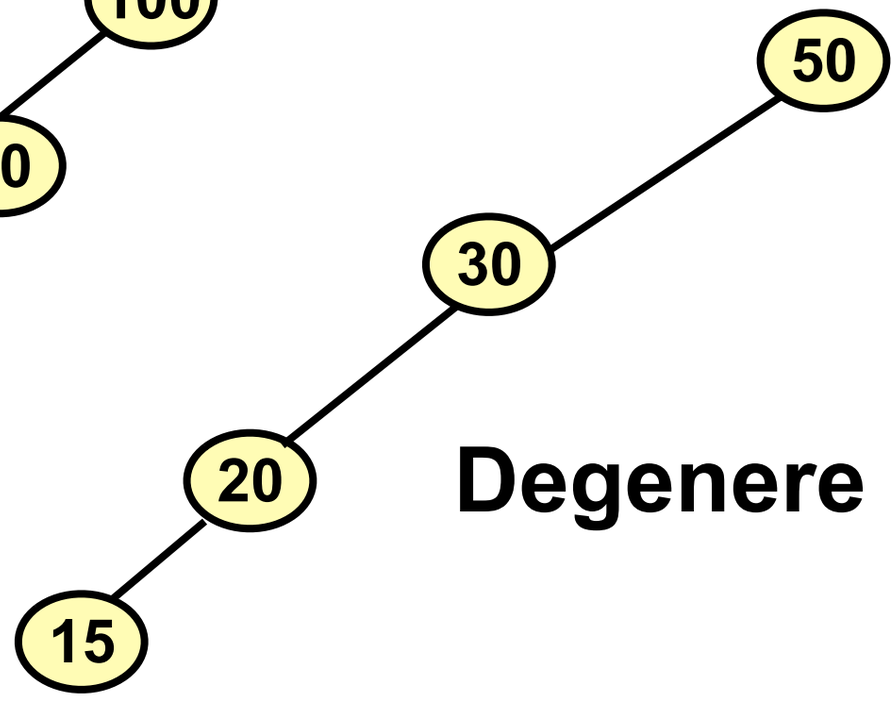
Nel seguito supponiamo che le chiavi siano tutte distinte



ABR: esempi



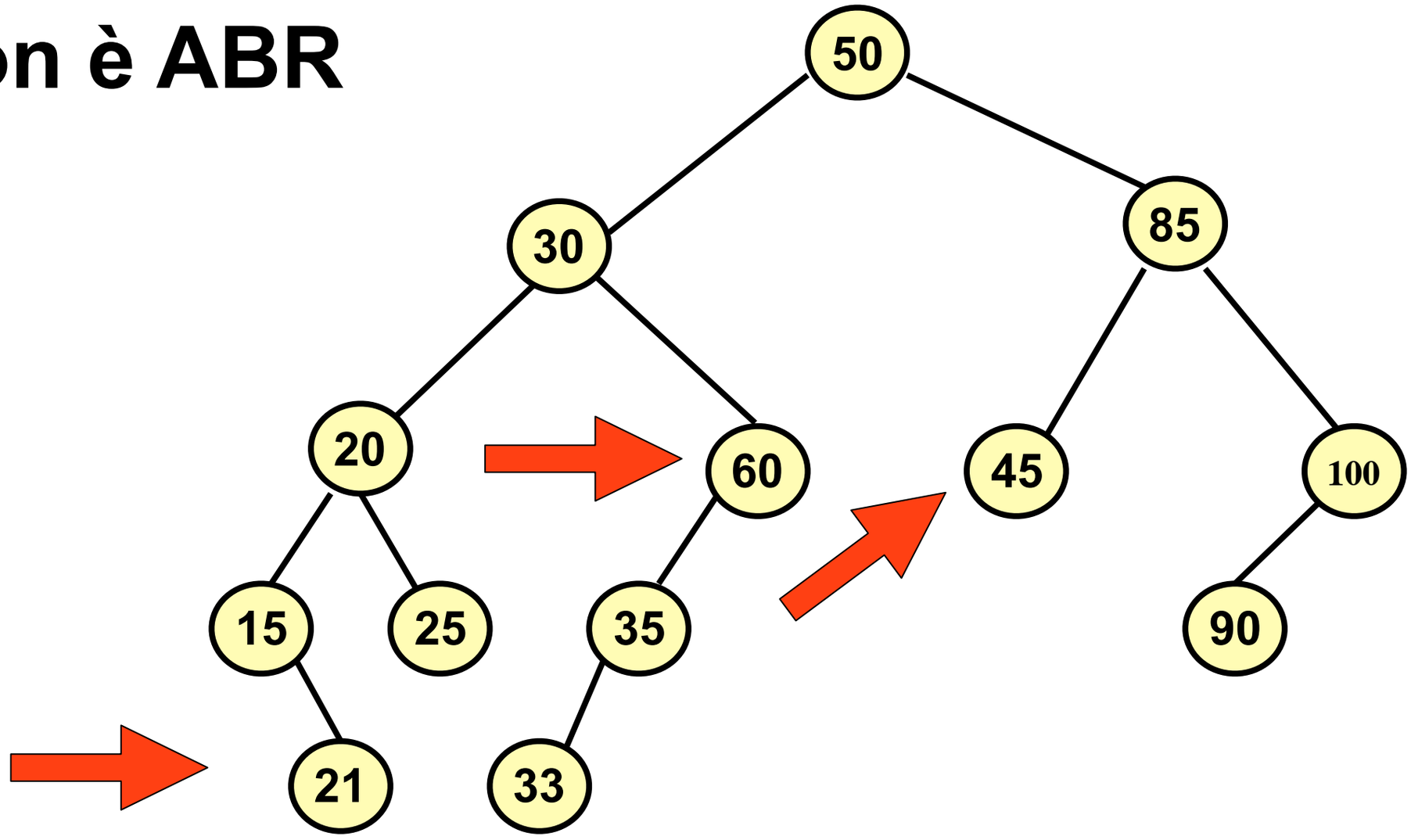
Tipico



Degenere

ABR: esempi

non è ABR



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

Rappresentazione in memoria alberi binari

Un abero binario di ricerca è innanzi tutto un albero binario t.

Nella rappresentazione in memoria assumiamo che

ogni nodo di t abbia

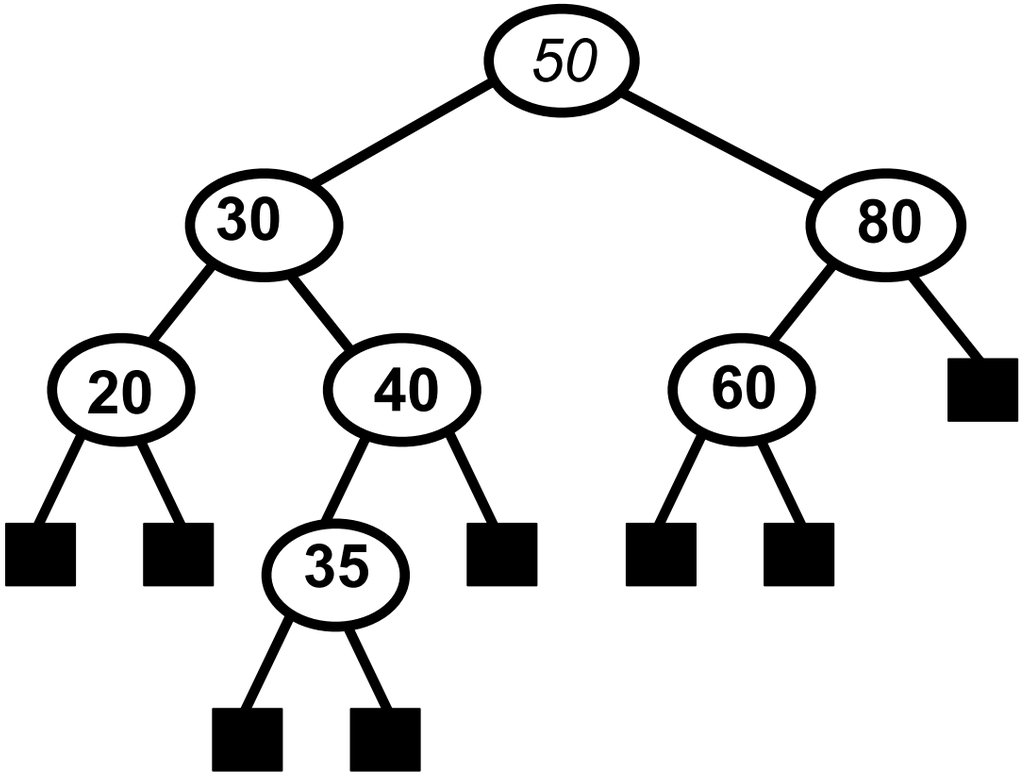
un campo key, che contiene la chiave del dato

un campo left, con un puntatore al figlio sinistro,

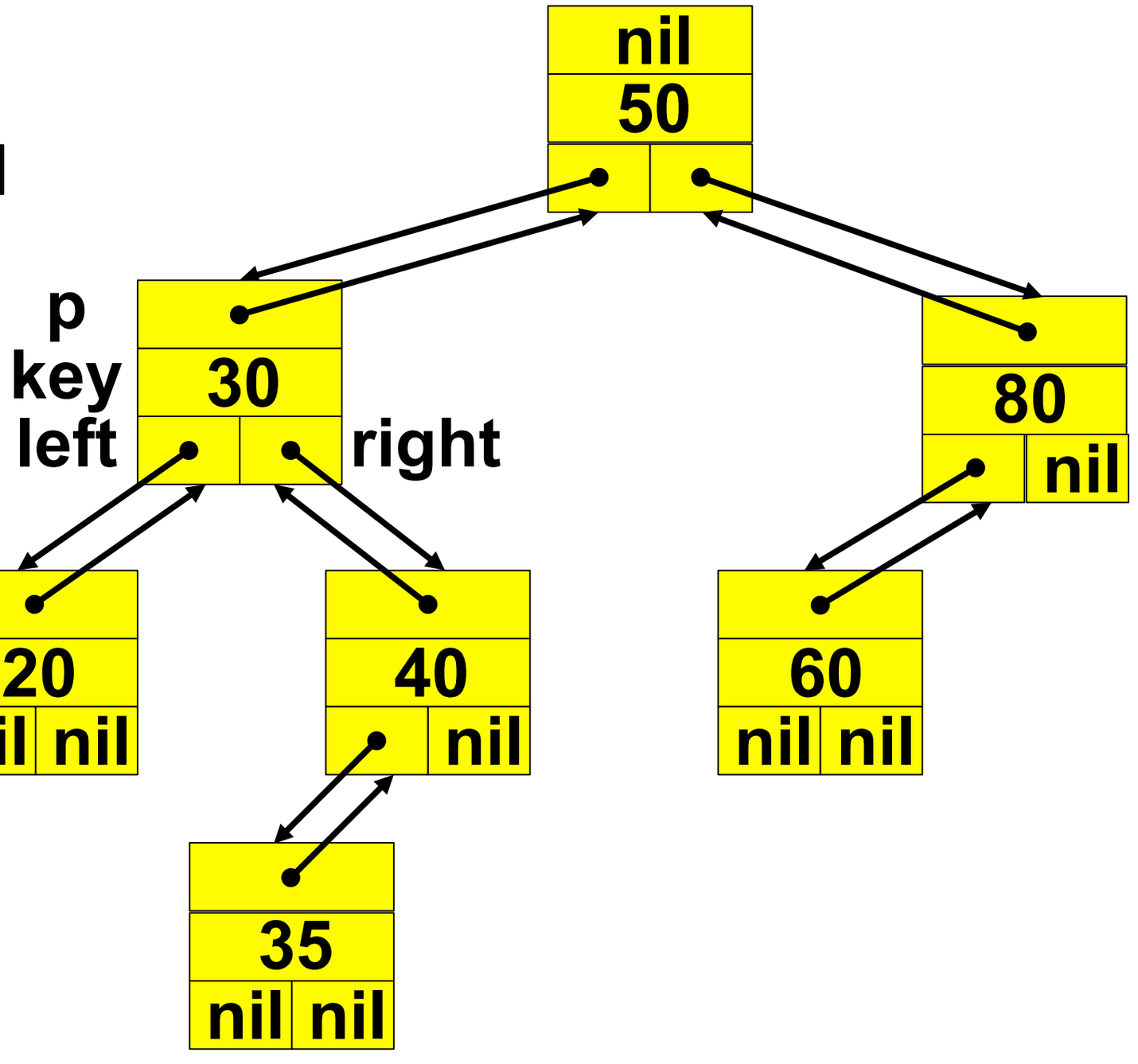
un campo right, con un puntatore al figlio destro

un campo p, con un puntatore al padre

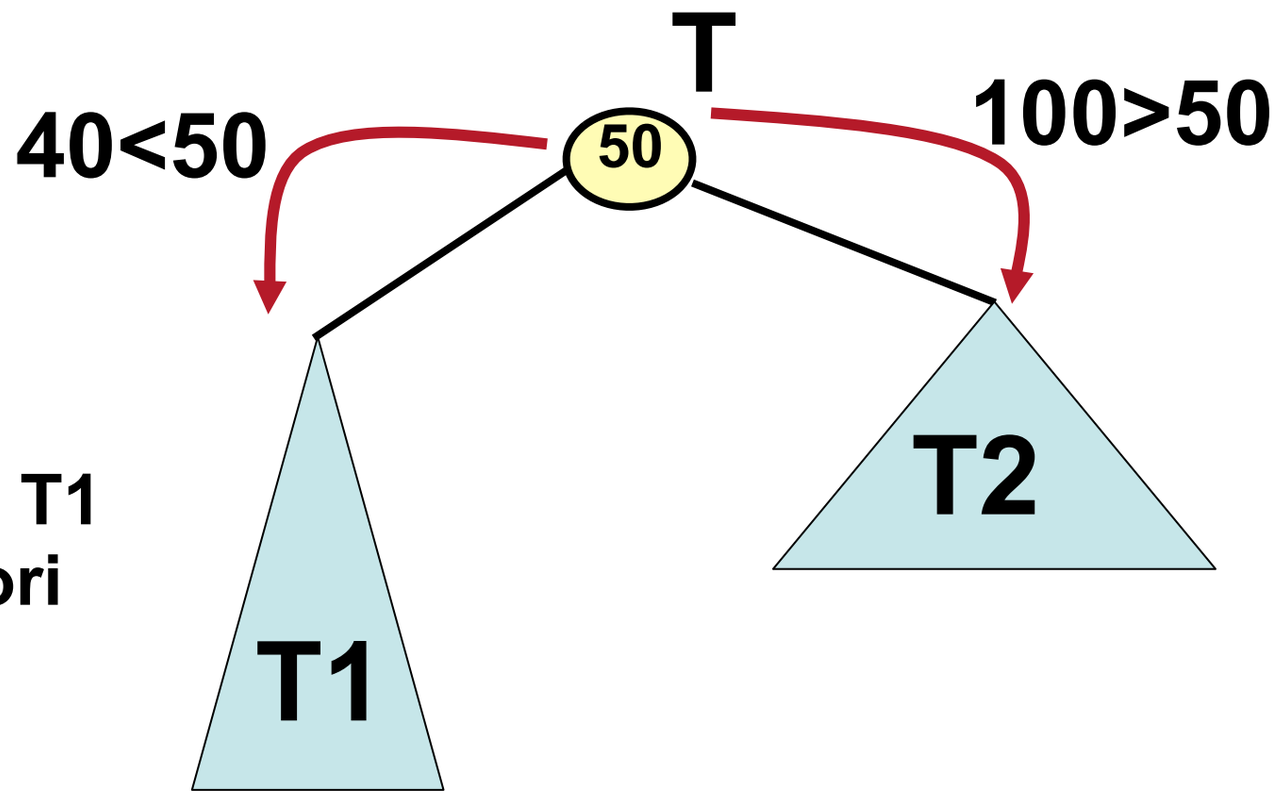
Rappresentazione in memoria



rappresentazione



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!

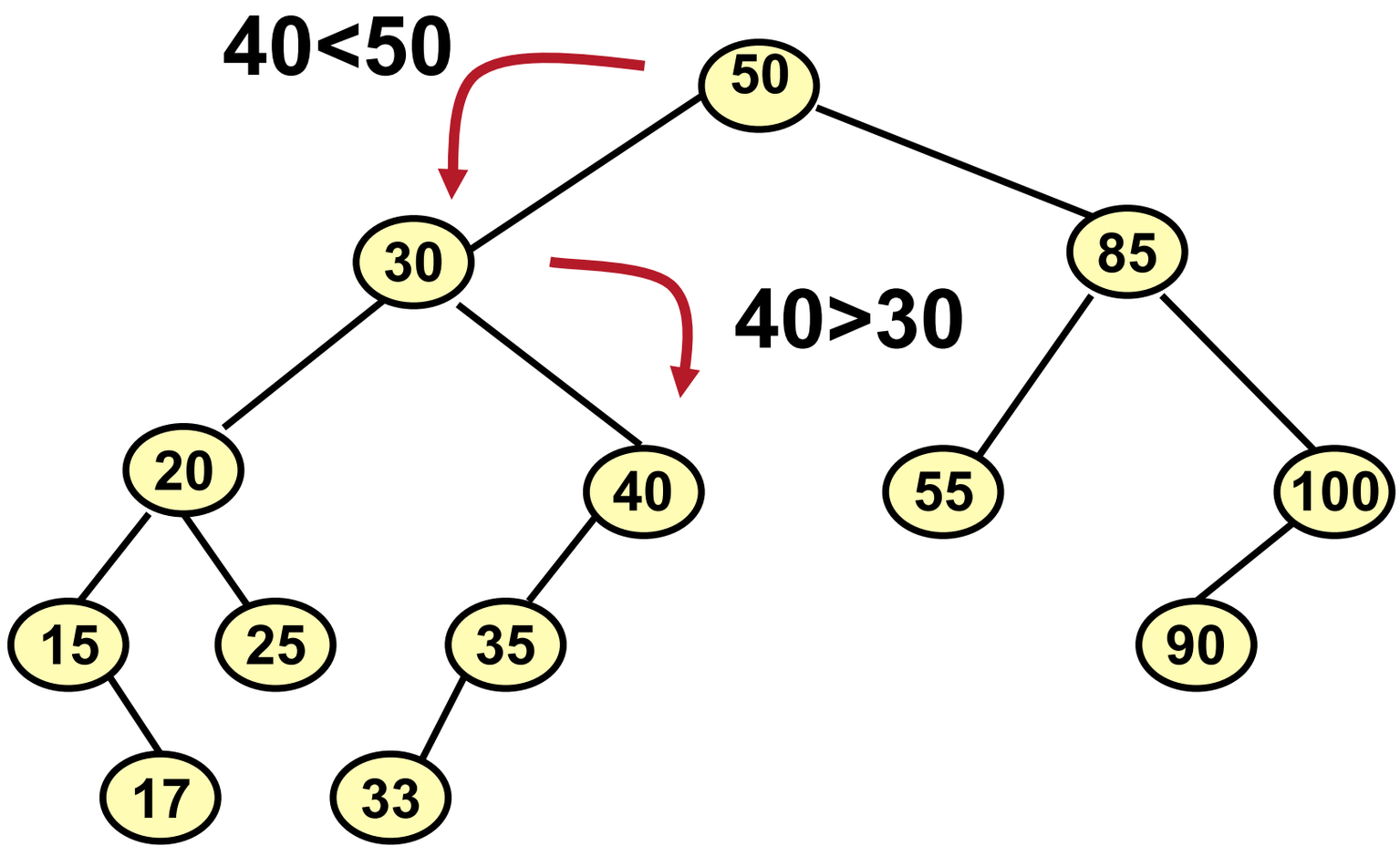
La ricerca di 40 in T deve proseguire nel sotto albero sinistro T1

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.

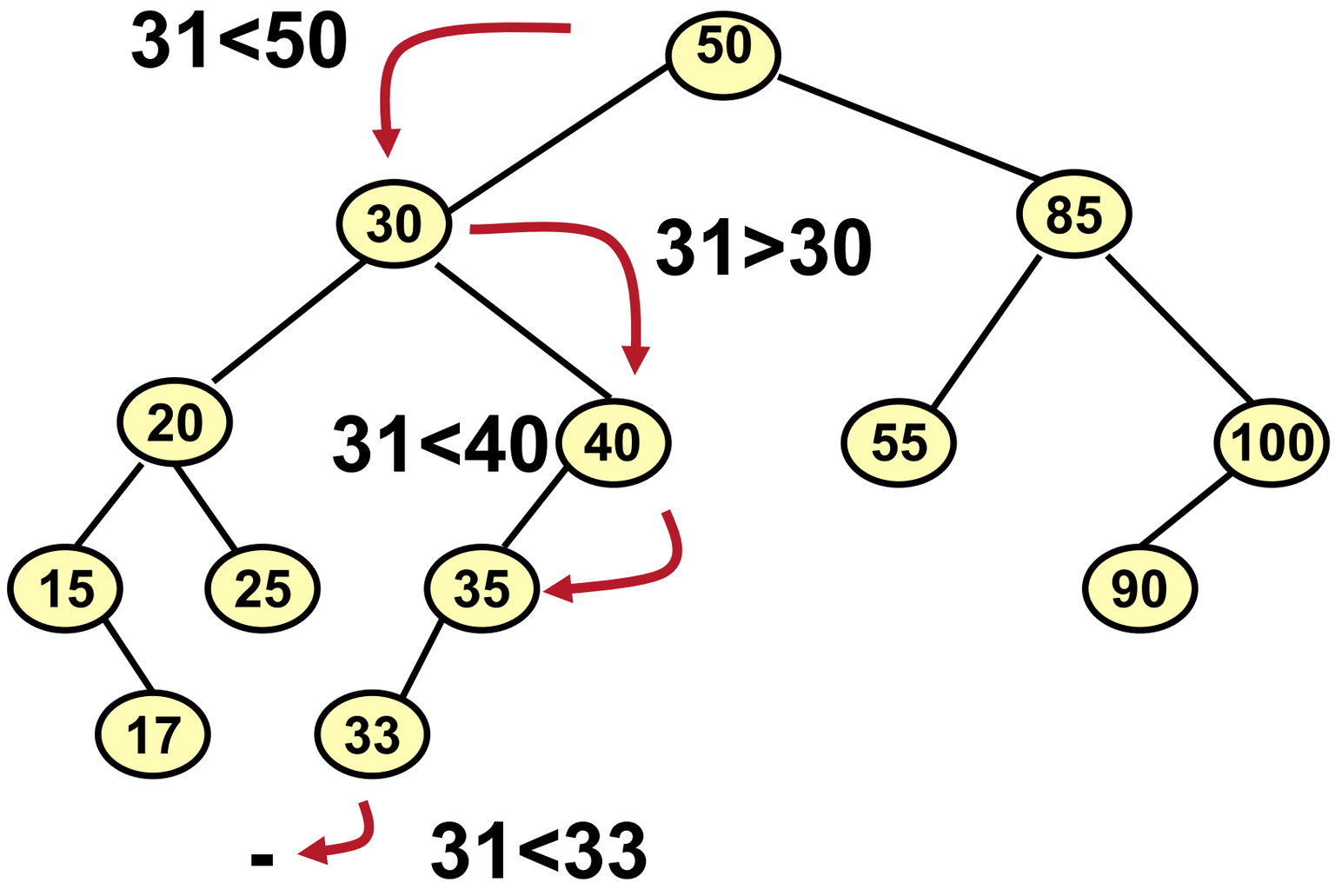
La ricerca con successo

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

postc: restituisce 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) + O(1)

La cui soluzione è Θ(h).

In generale non è O(lg n)!!

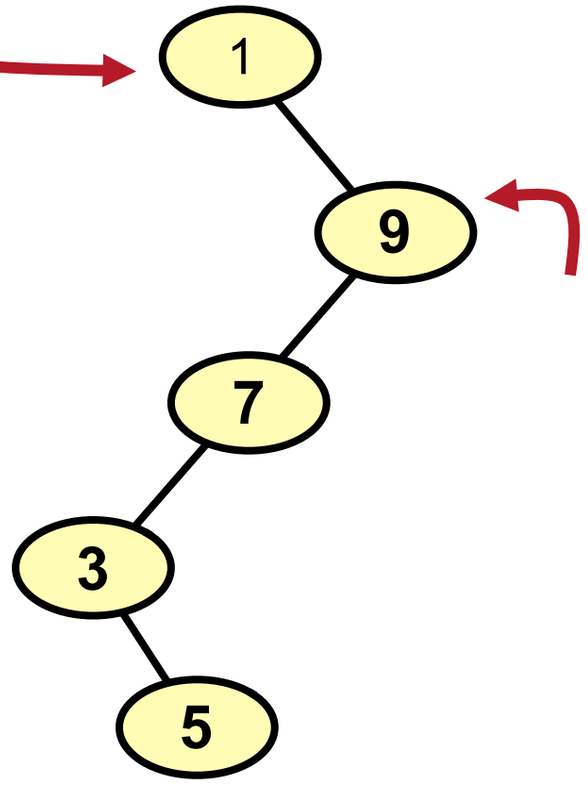
Ricordiamo che se n è il numero degli elementi di t vale
lg n ≤ h ≤ n - 1

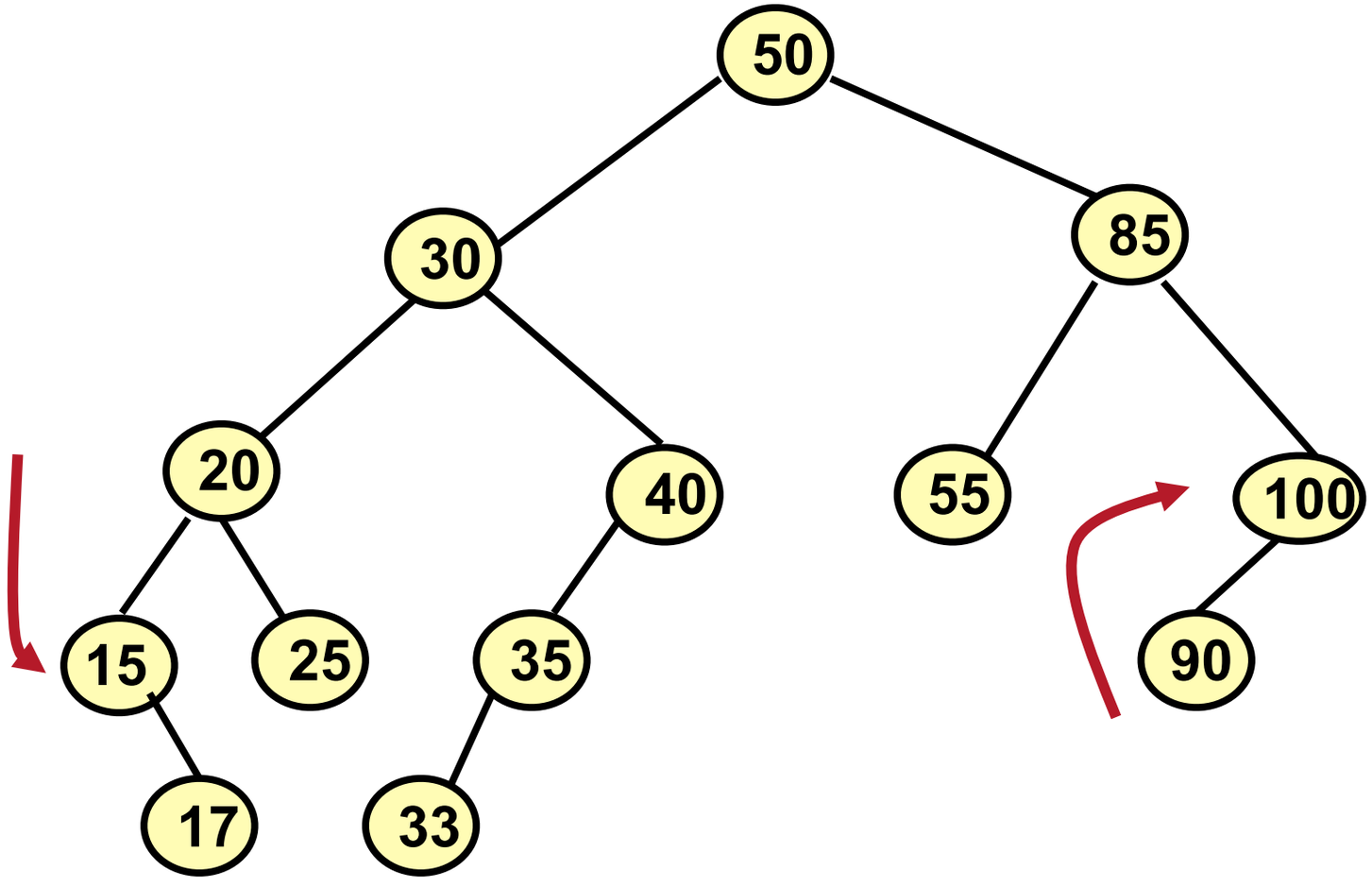
Quindi tempo di esecuzione nel caso peggiore potrebbe essere O(n)!
Ma è più informativo dire Θ(h) nel caso peggiore o O(h) in tutti i casi, per due motivi:

1. **si sa che h ≤ 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





Il più grande elemento

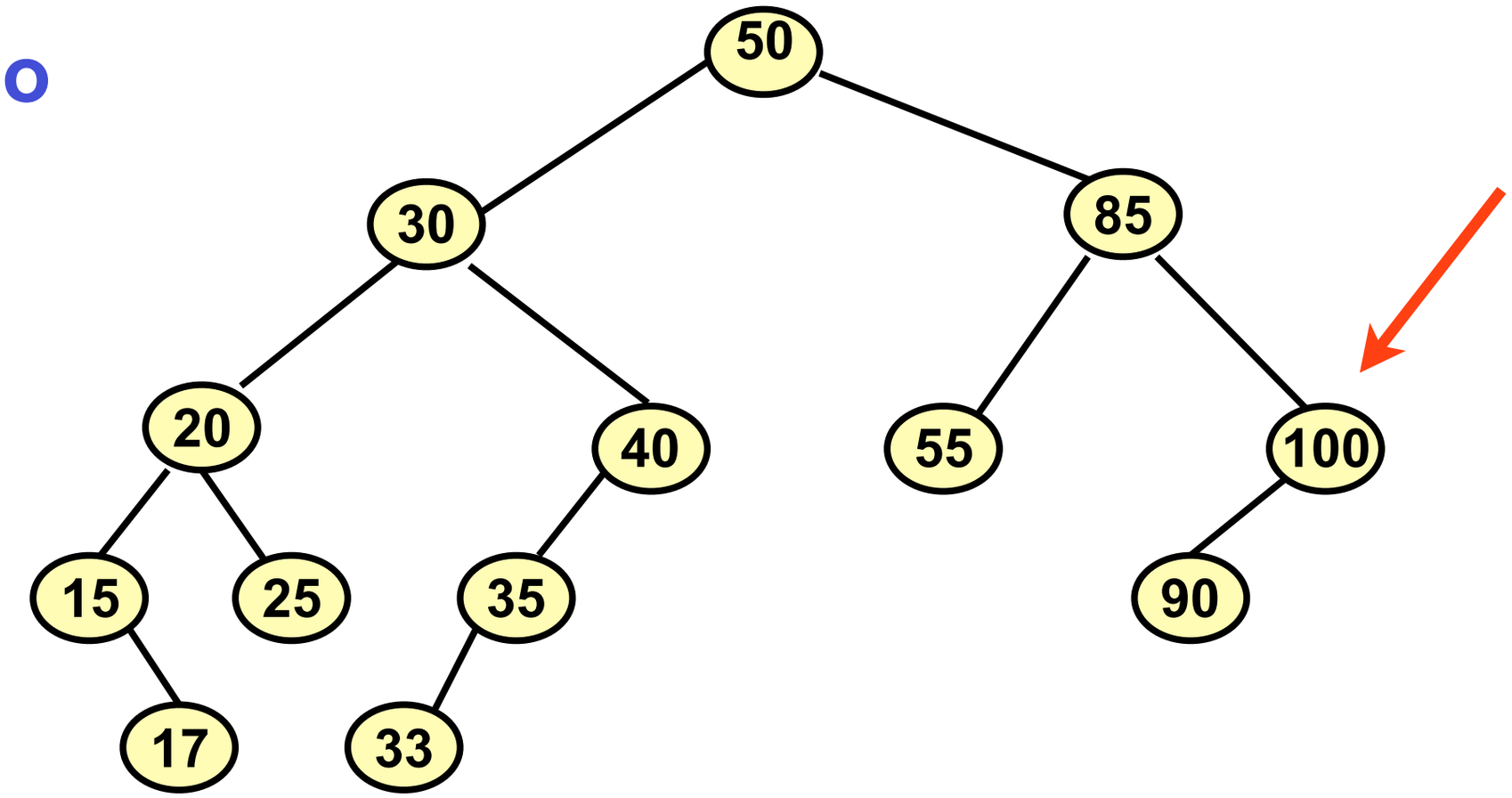
Il massimo

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)

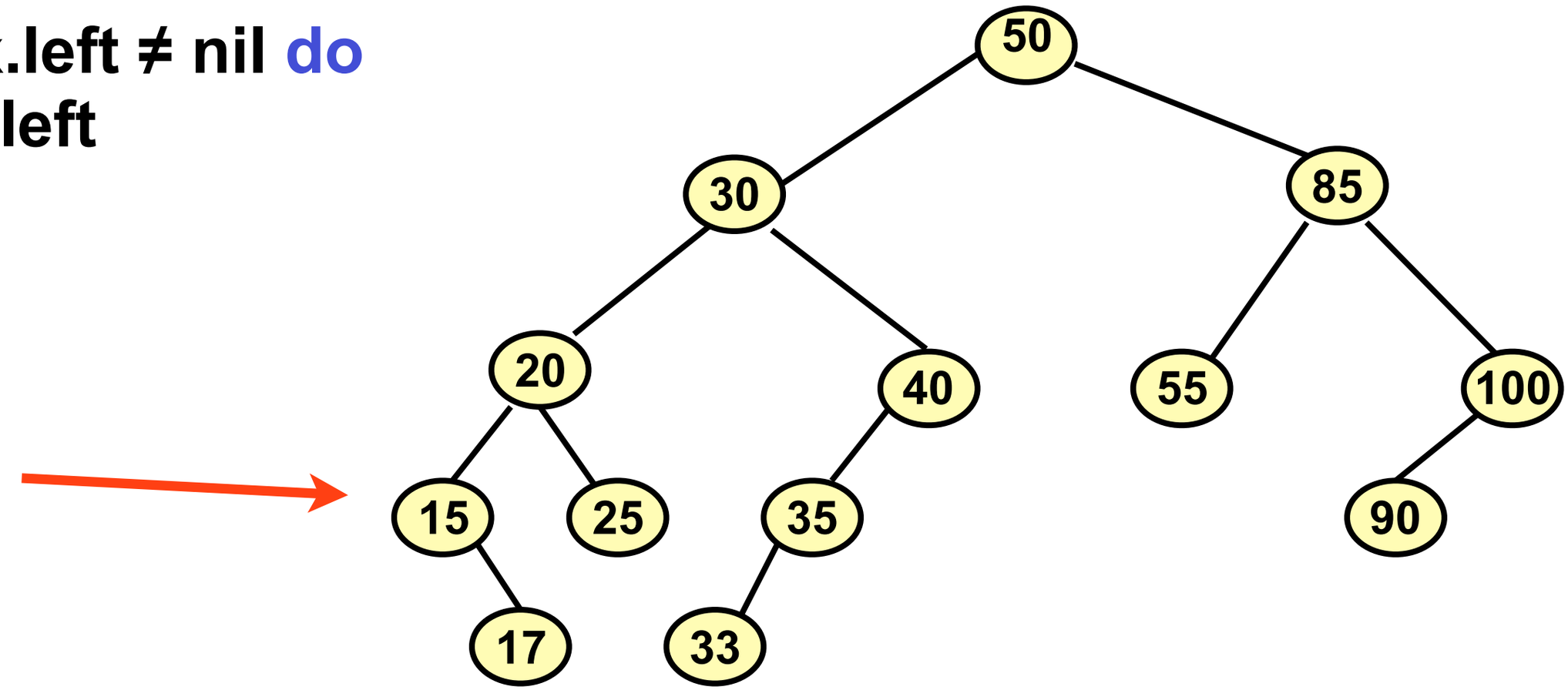
Il minimo

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

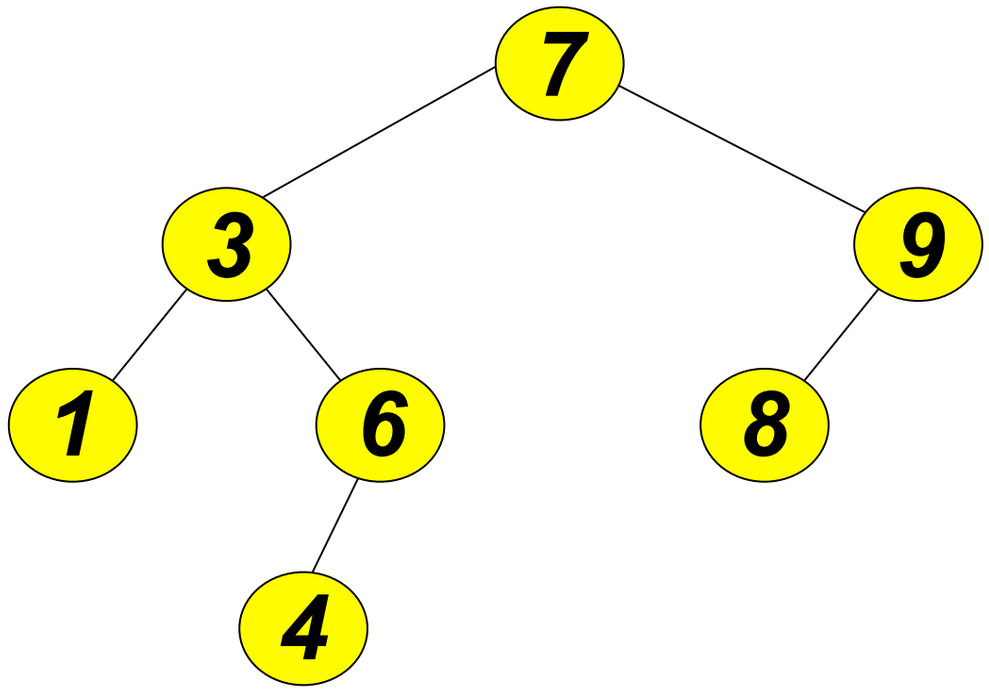
```
MinimumRic(x)
precond:x ≠ nil e x è un ABR
postcond:restituisce la chiave minima in x
 if x.left == NIL
        then return x.key
        else return MinimumRic(x.left)
```

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

L'inserimento

Insert(T,z)

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

postcond: inserisce x 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

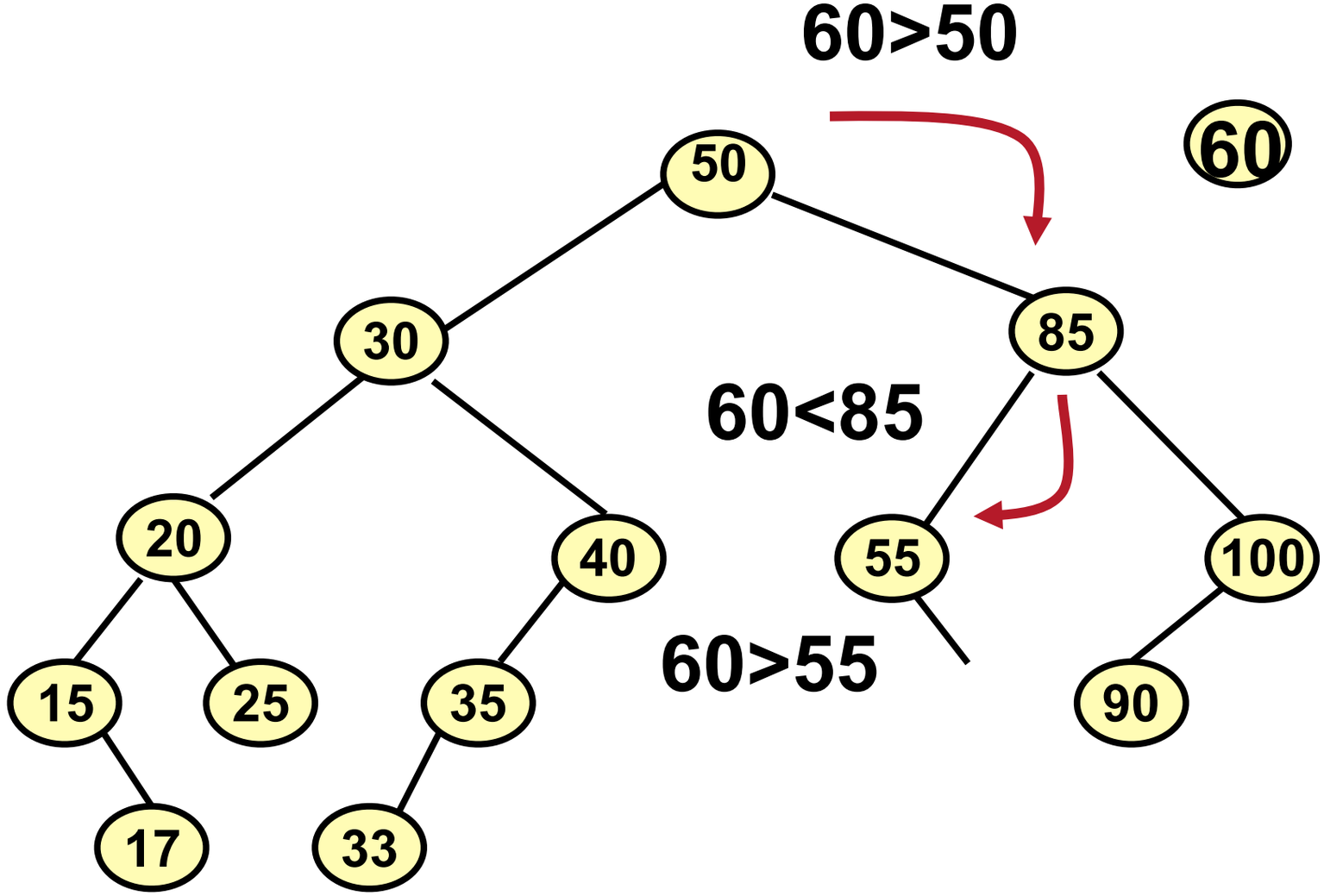
InsertRic(T,z) **Tempo di esecuzione? O(h)**

precond: T è un ABR, T ≠ NIL e z ≠ nil

postcond:inserisce z nell'ABR T

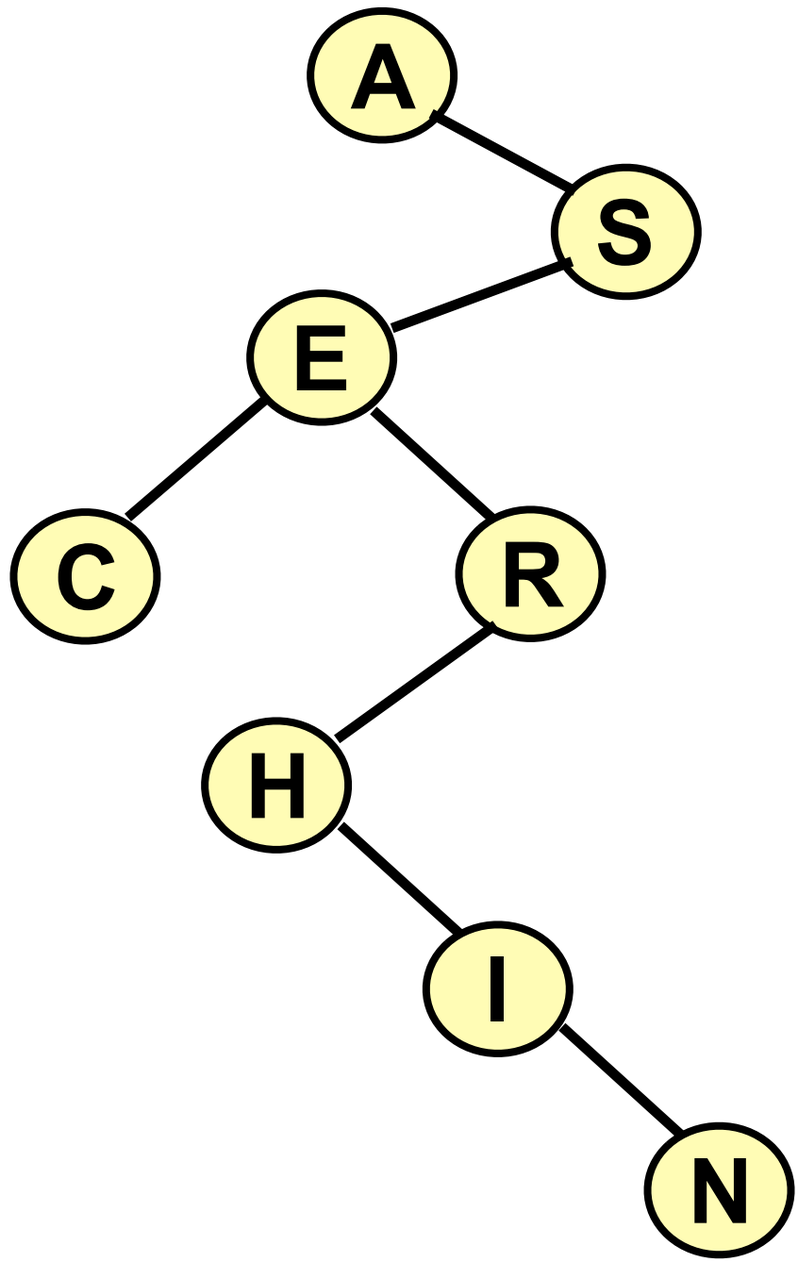
```
if z.key < T.key
     then
     if T.left == NIL
        then T.left = z
                z.p = T
     else InsertRic(T.left,z)
else
if T.right == NIL
     then T.right = z
                z.p = T
     else InsertRic(T.right,z)
```

Inseriamo z = 60



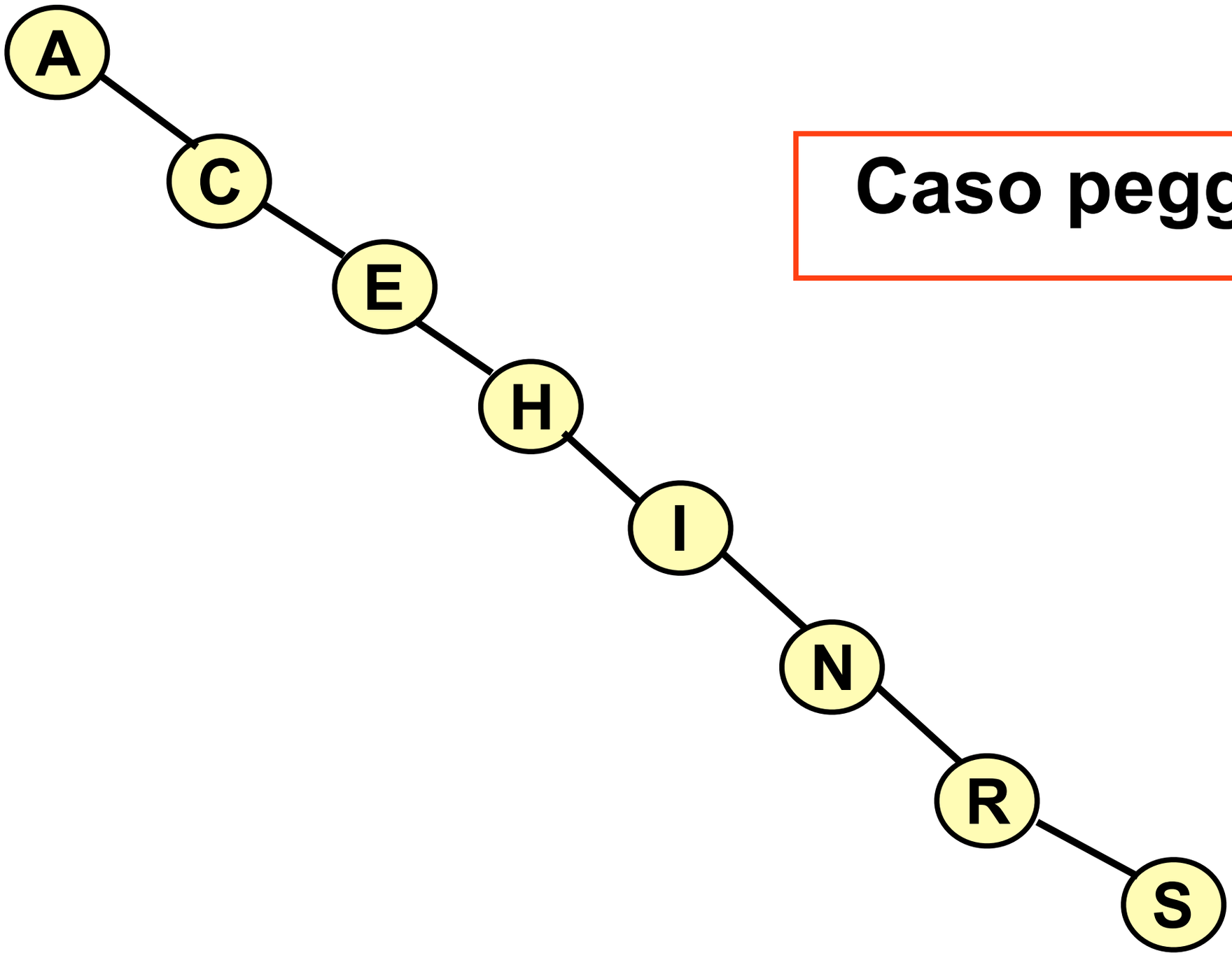
Forma dell'ABR

Inseriamo A S E R C H I N



Forma dell'ABR

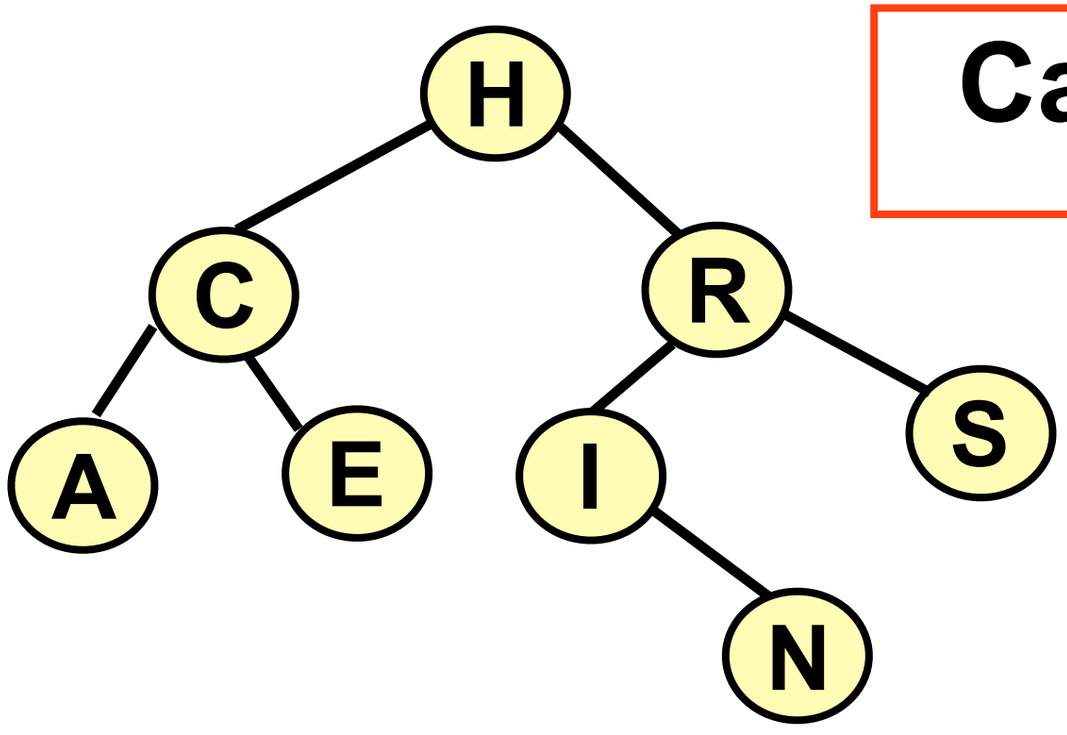
Inseriamo A C E H I N R S



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

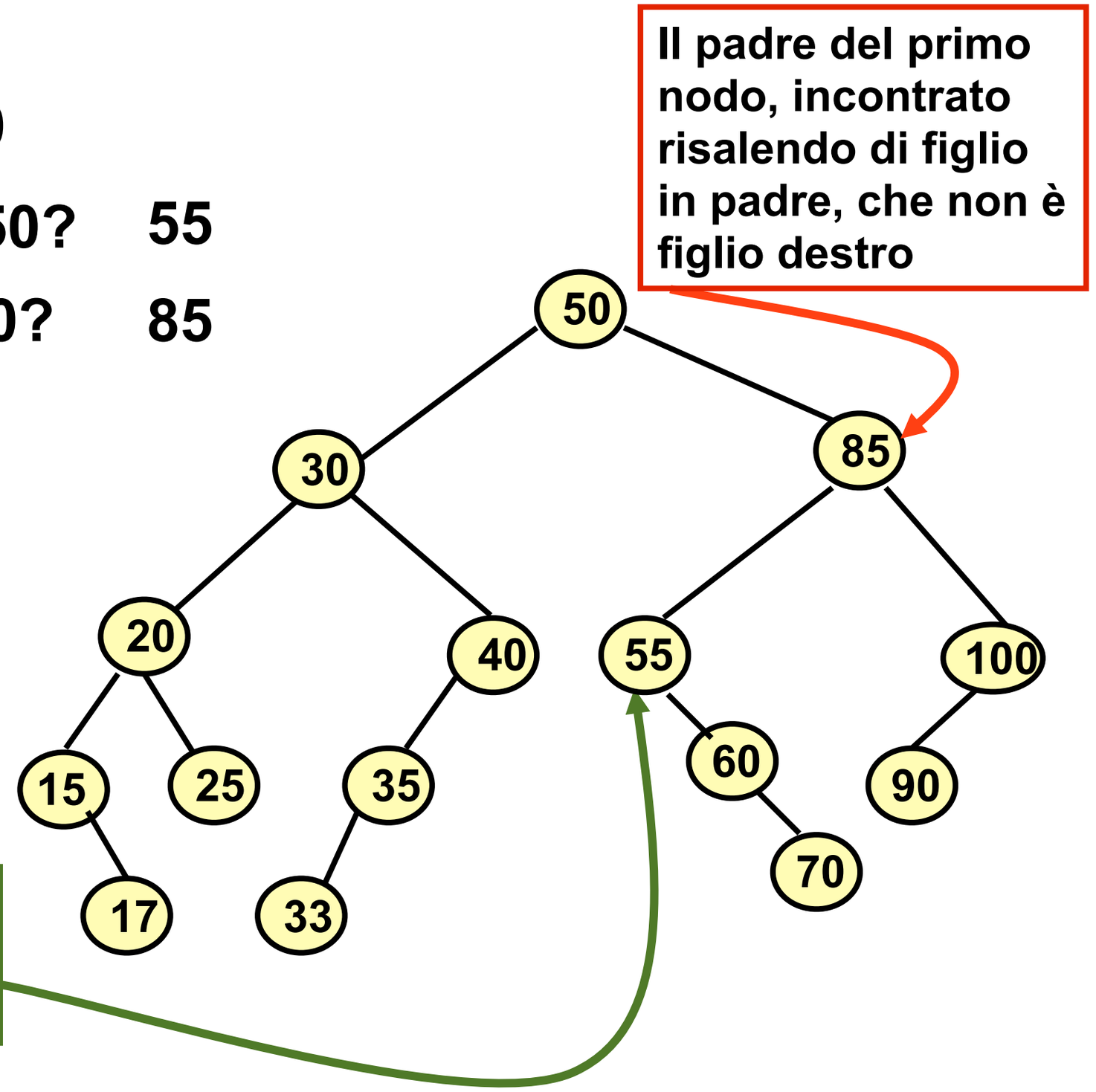
La chiave successiva a una data nell'albero, se c'è, dove può essere?

25 è il successivo di 20

Chi è il successivo di 50? 55

Chi è il successivo di 70? 85

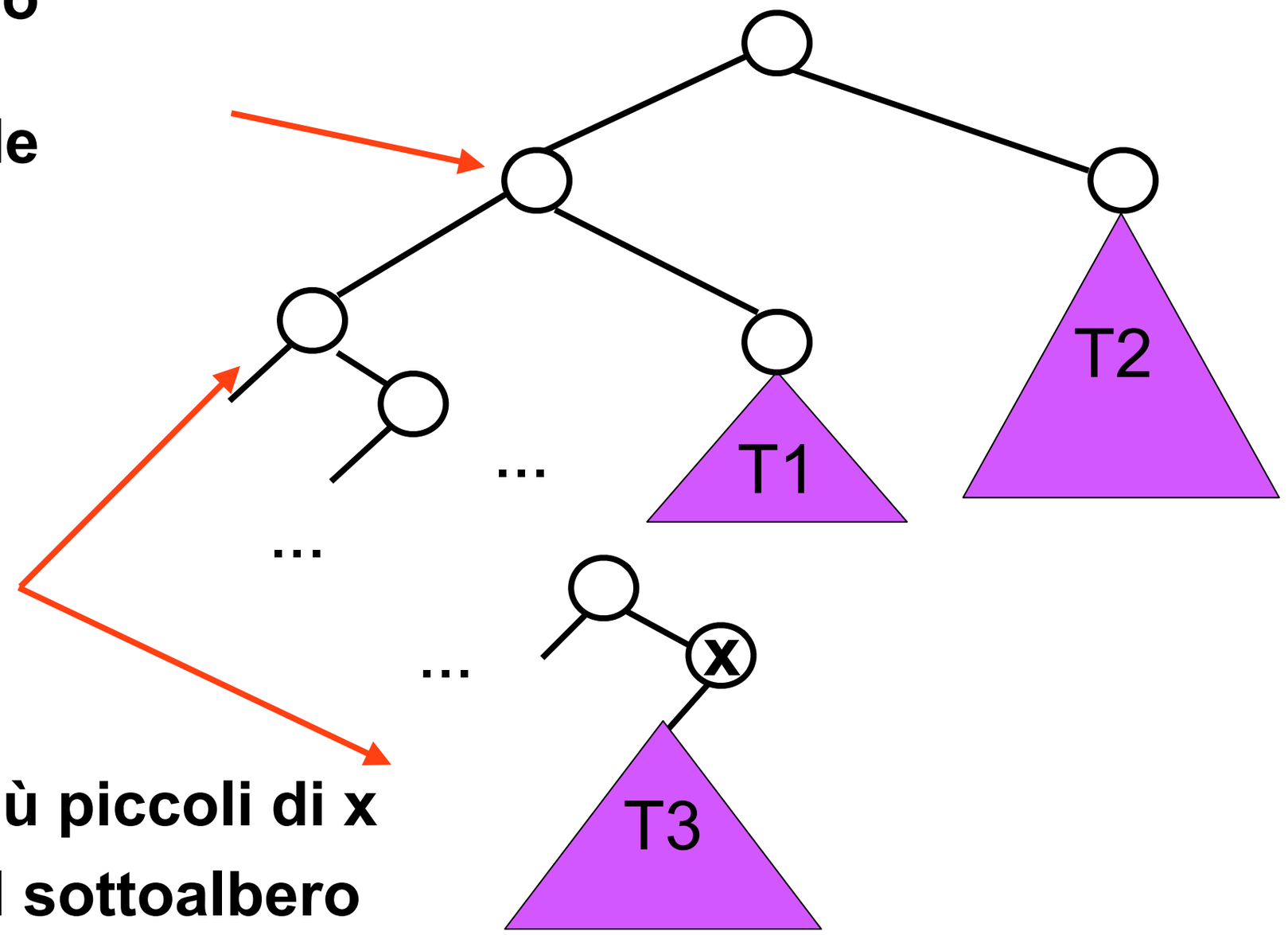
La chiave successiva di una data chiave x, in un nodo senza figli destri, nell'albero è quella del primo nodo, risalendo dal nodo verso la radice, che ha x nel sotto albero sinistro.



Successivo di un nodo di chiave x senza figlio destro

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

che ha x nel sottoalbero sinistro!



tutti più piccoli di x

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
```

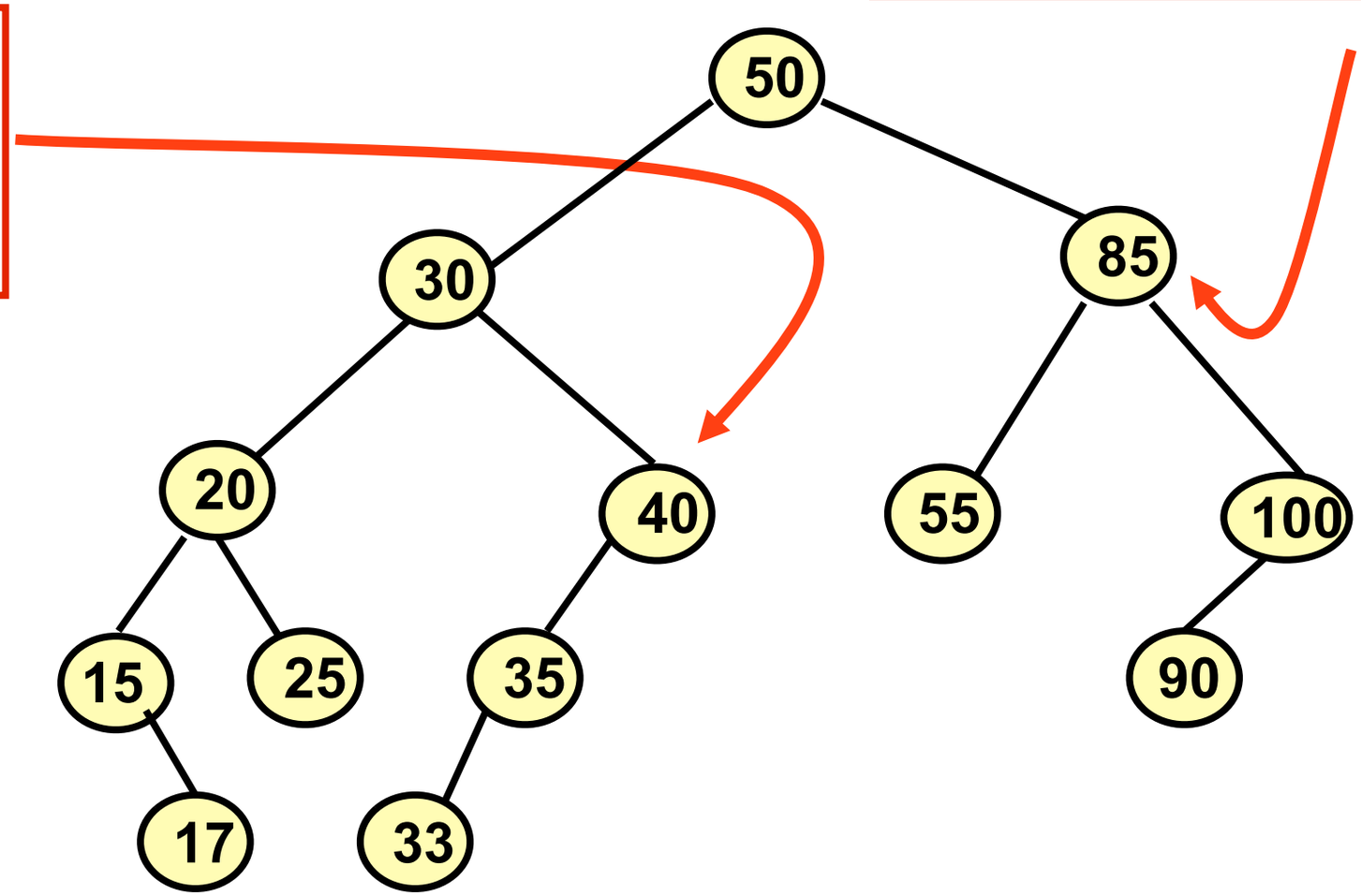
Precedente

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**

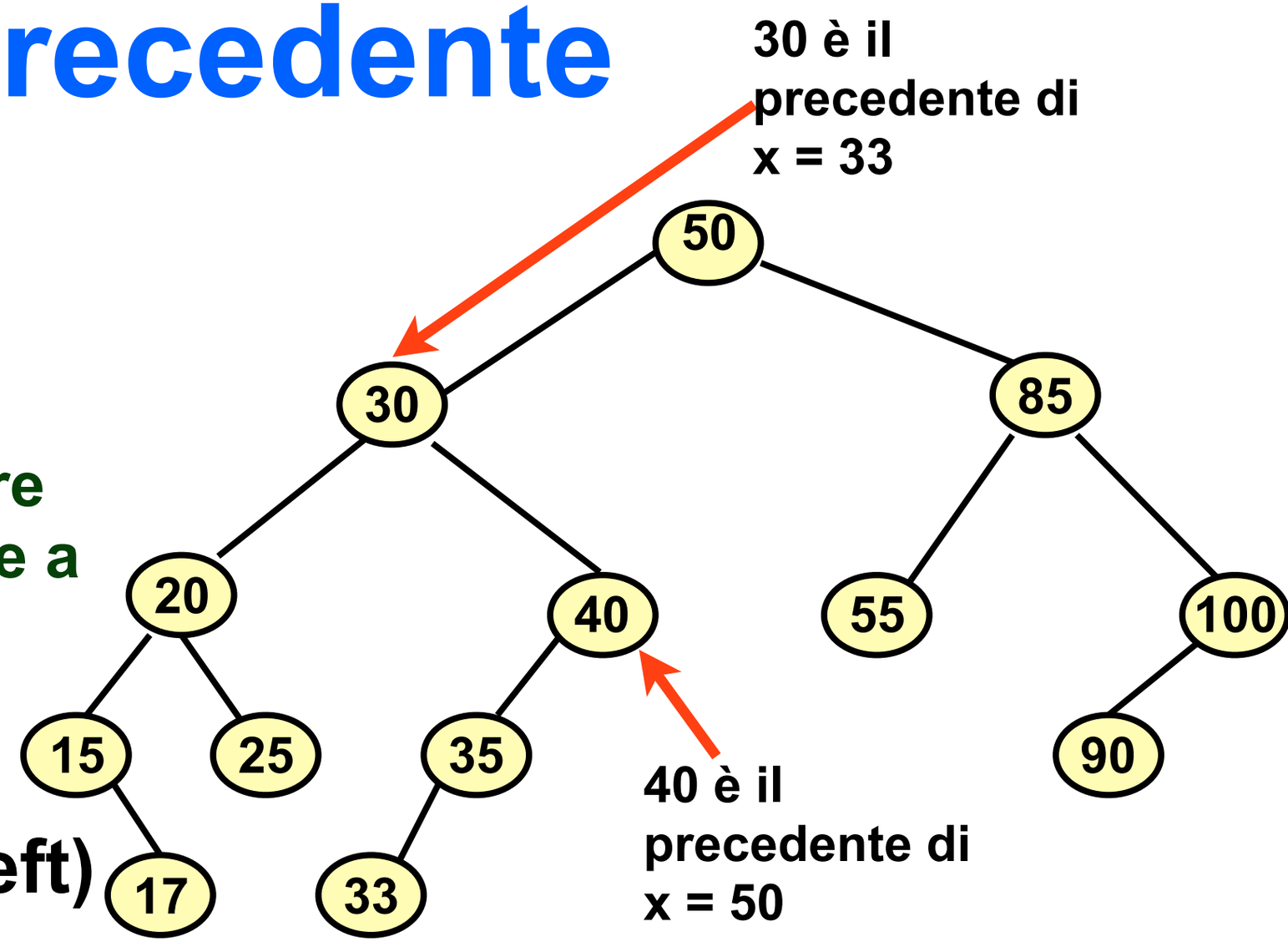
Il precedente

Predecessor(x)

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

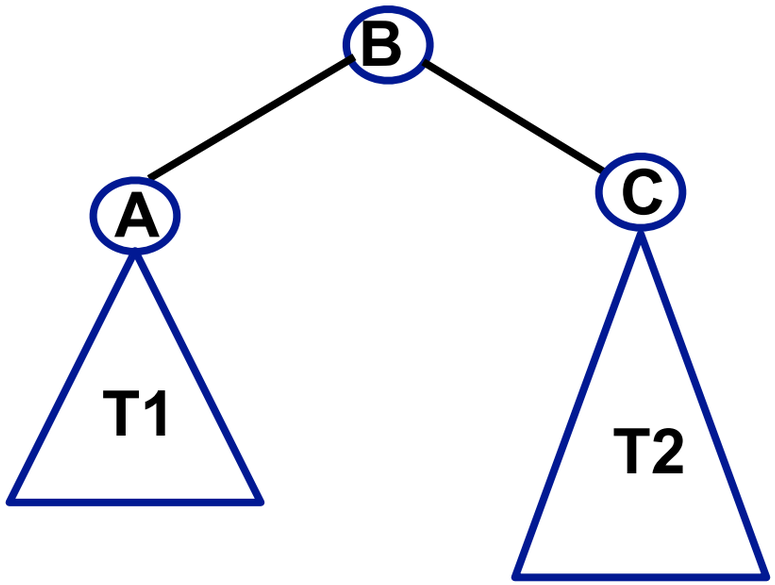
postc: restituisce il puntatore al nodo di chiave precedente a quella di x, se c'è, NIL altrimenti

```
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
```

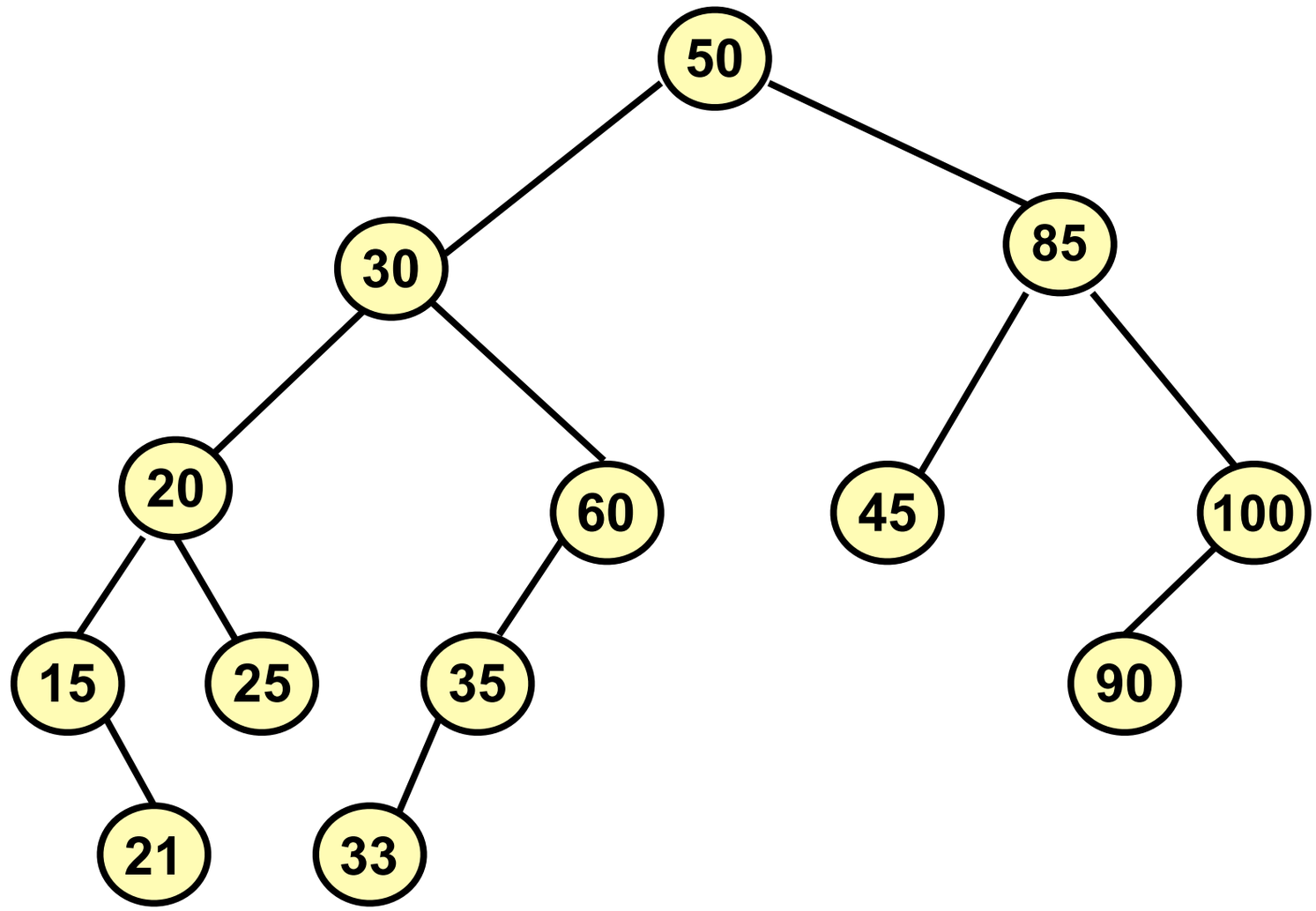


Complessità O(h)

Definizione ricorsiva di ABR

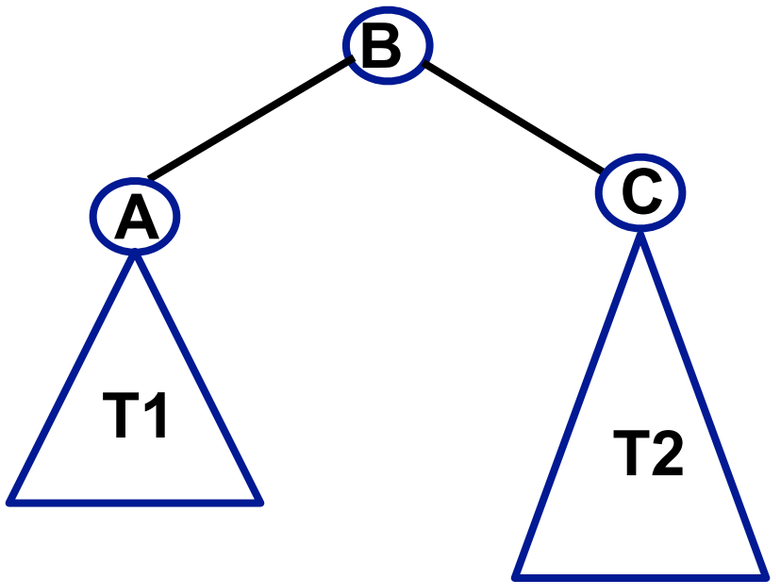


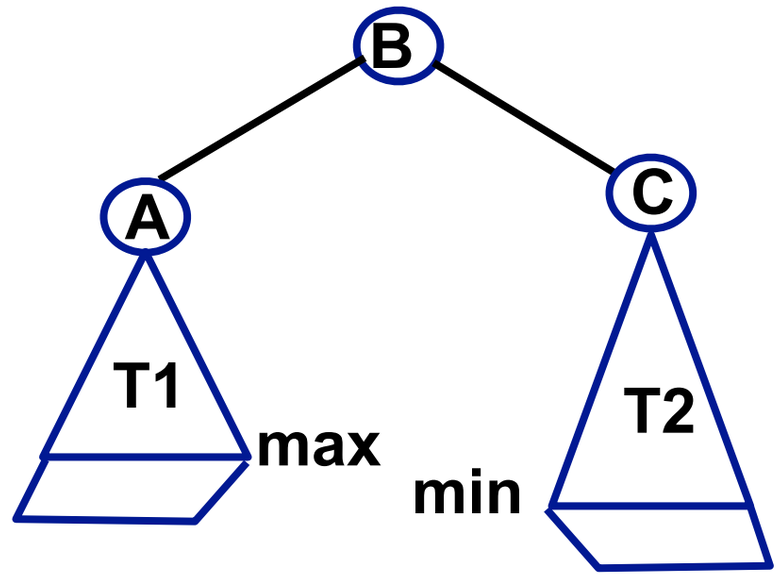
Un albero binario è un ABR se
il suo sotto albero sinistro, T1, è un ABR
il suo sotto albero destro, T2, è un ABR
e A < B < C



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.