

# *Algoritmi e Strutture Dati*

## **Alberi Binari di Ricerca**

# *Alberi binari di ricerca*

## **Motivazioni**

- gestione e ricerche in grosse quantità di dati
- *liste* ed *array non* sono *adeguati* perché inefficienti in tempo  $O(n)$  o in spazio

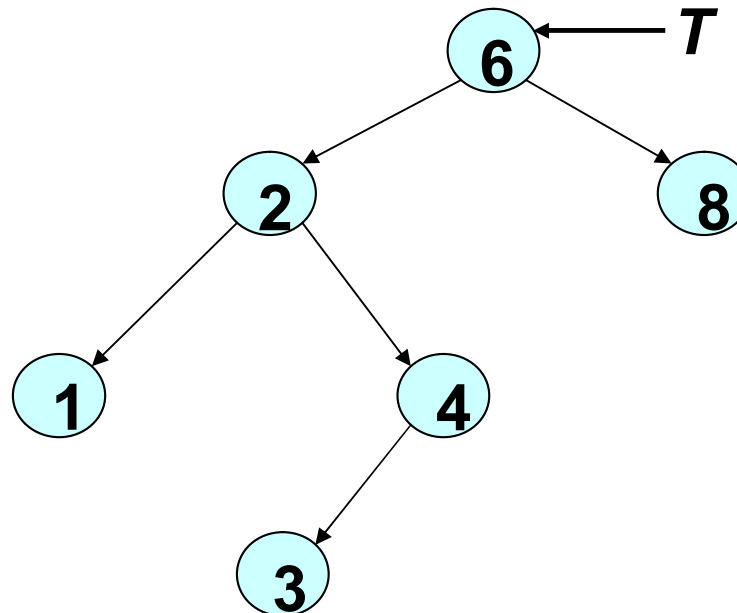
## **Esempi:**

- Mantenimento di archivi (*DataBase*)
- In generale, mantenimento e gestione di corpi di *dati* su cui si effettuano *molte ricerche*.

## Alberi binari di ricerca

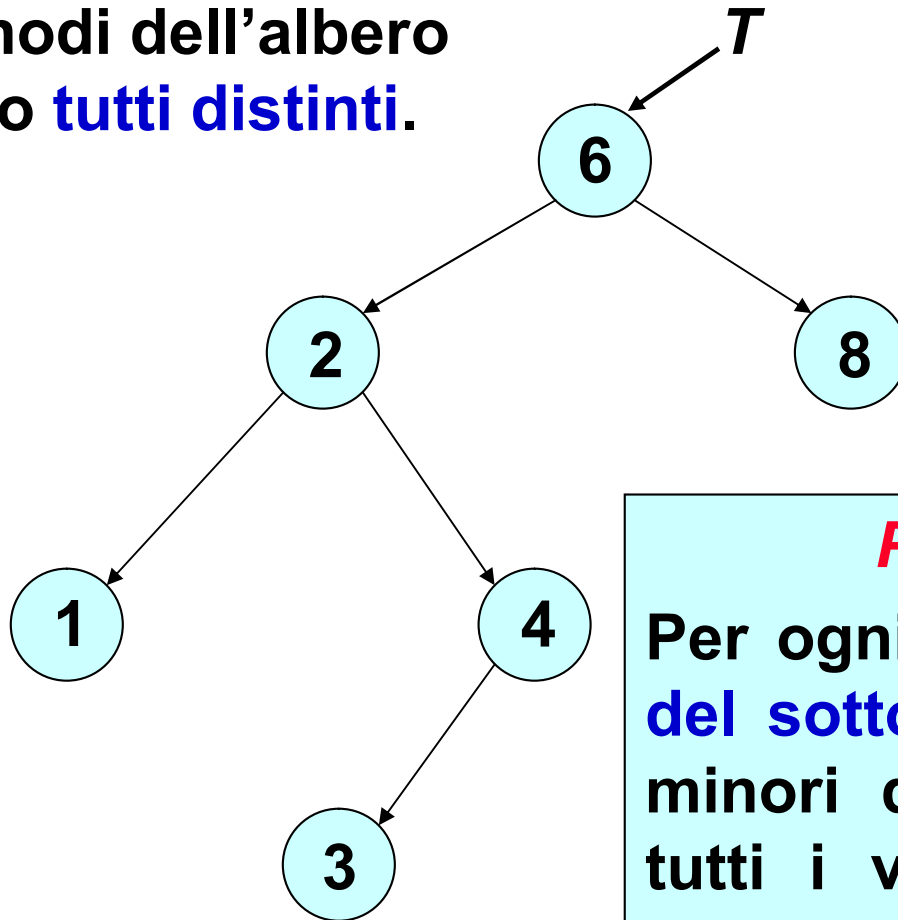
**Definizione:** Un albero binario di ricerca è un albero binario che soddisfa la seguente proprietà:

se  $X$  è un nodo e  $Y$  è un nodo nel **sottoalbero sinistro** di  $X$ , allora  $key[Y] \leq key[X]$ ; se  $Y$  è un nodo nel **sottoalbero destro** di  $X$  allora  $key[Y] \geq key[X]$



# Alberi binari di ricerca

**Assumiamo** che i **valori** nei nodi dell'albero siano **tutti distinti**.



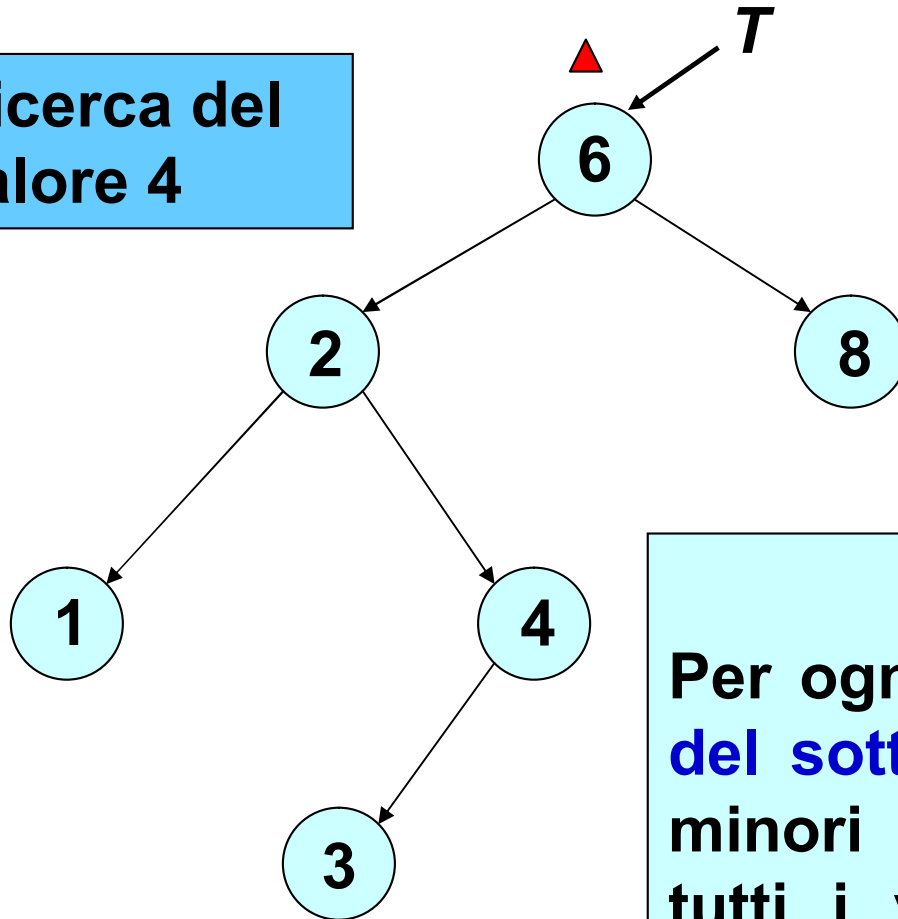
**Assumiamo** che i **valori** nei nodi (le chiavi) **possono** essere **ordinati**.

## **Proprietà degli ABR**

Per ogni nodo  $X$ , i valori nei **nodi del sottoalbero sinistro** sono tutti minori del valore nel nodo  $X$ , e tutti i valori nei **nodi del sottoalbero destro** sono maggiori del valore di  $X$

# Alberi binari di ricerca: esempio

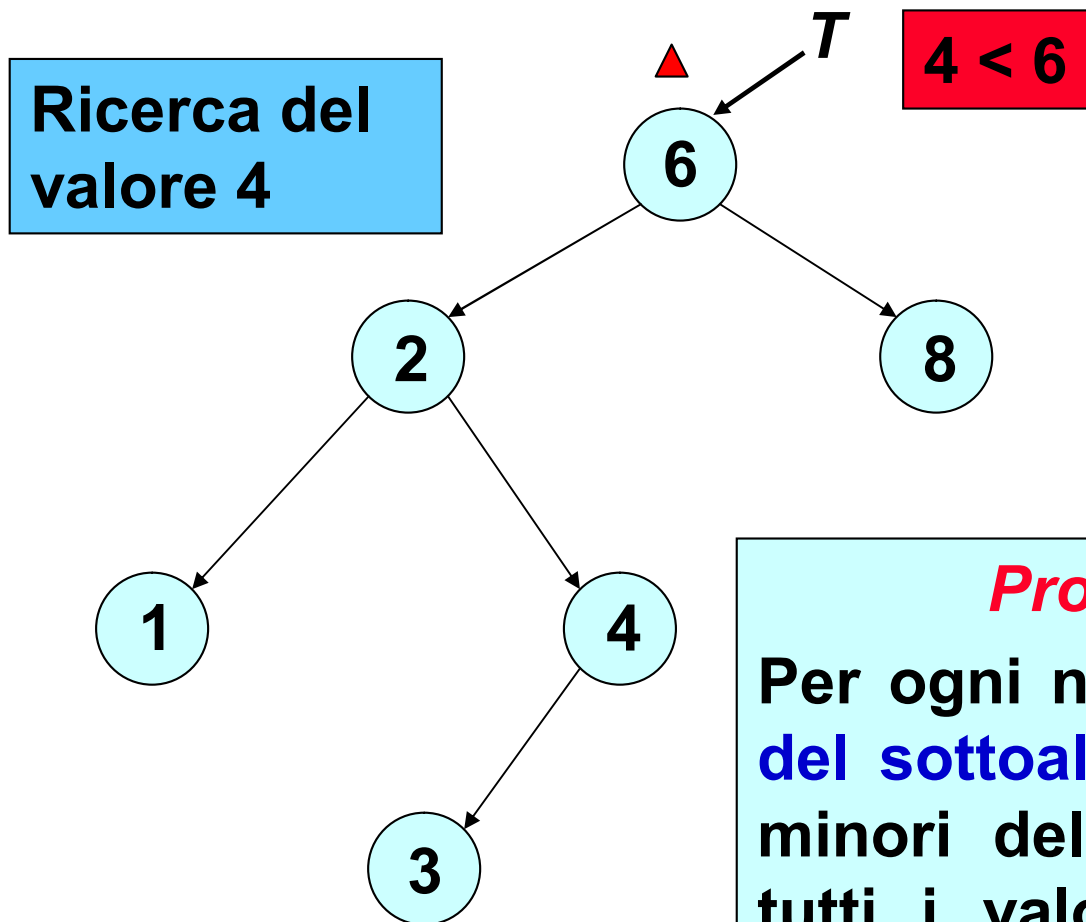
Ricerca del  
valore 4



## *Proprietà degli ABR*

Per ogni nodo  $X$ , i valori nei **nodi del sottoalbero sinistro** sono tutti minori del valore nel nodo  $X$ , e tutti i valori nei **nodi del sottoalbero destro** sono maggiori del valore di  $X$

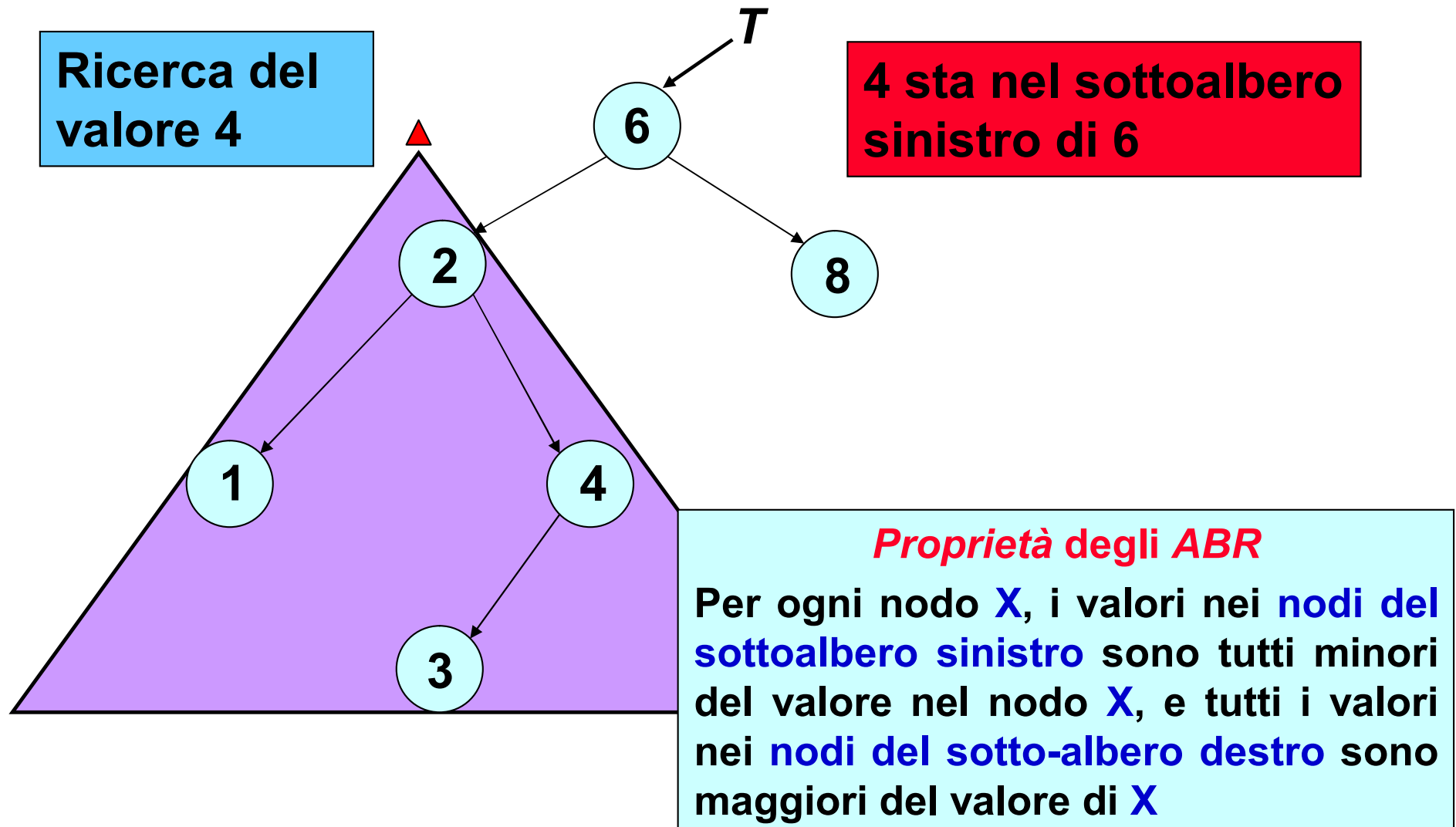
# Alberi binari di ricerca: esempio



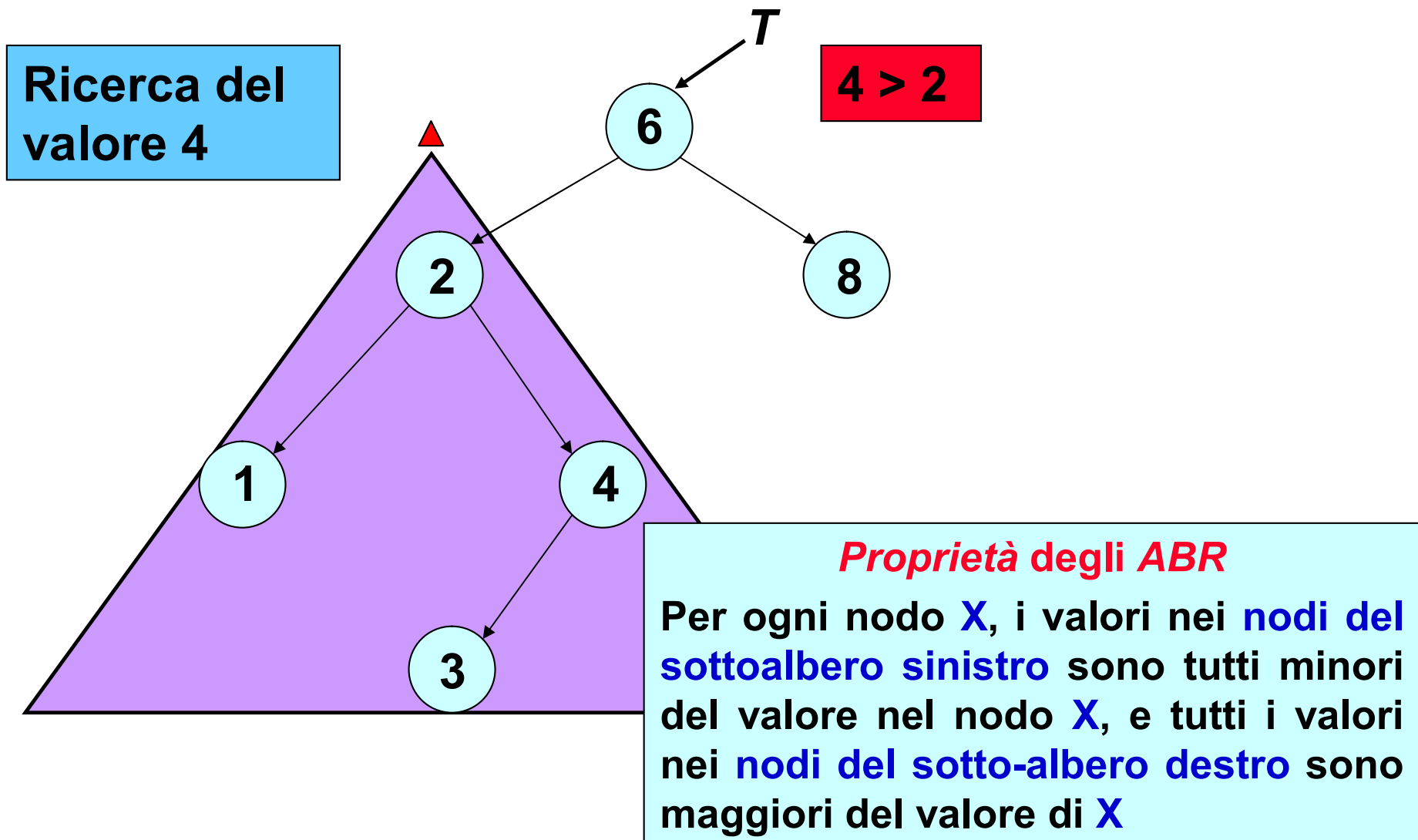
## Proprietà degli ABR

Per ogni nodo  $X$ , i valori nei **nodi del sottoalbero sinistro** sono tutti minori del valore nel nodo  $X$ , e tutti i valori nei **nodi del sottoalbero destro** sono maggiori del valore di  $X$

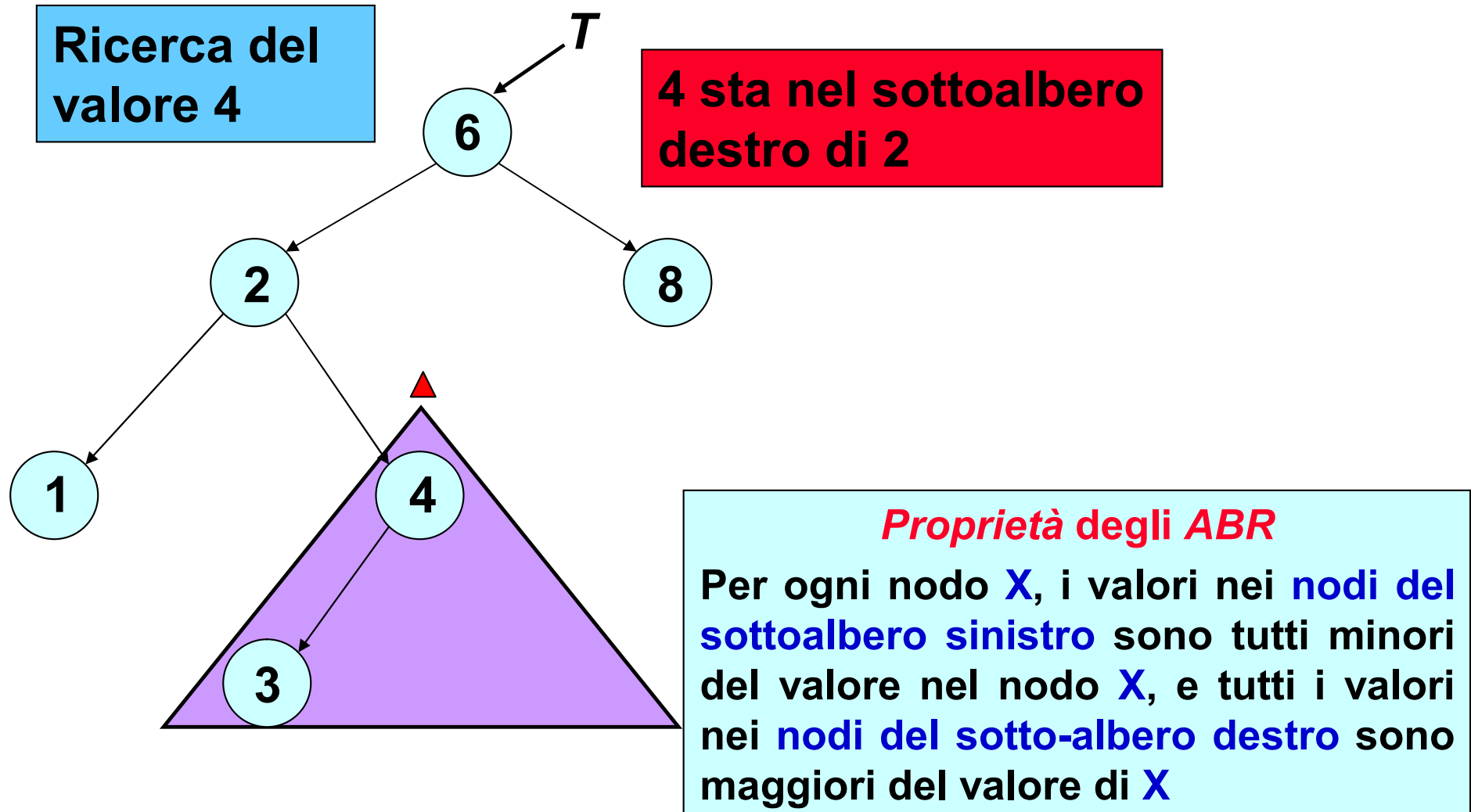
# Alberi binari di ricerca: esempio



# Alberi binari di ricerca: esempio

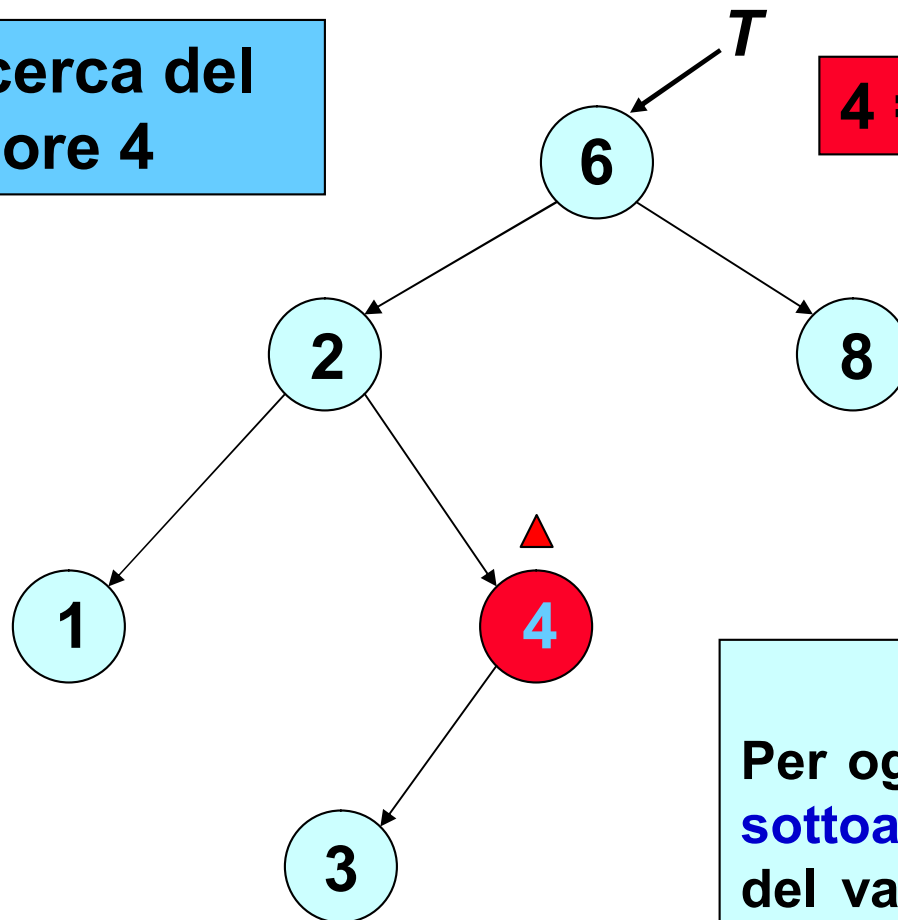


# Alberi binari di ricerca: esempio



# Alberi binari di ricerca: esempio

Ricerca del  
valore 4



4 = 4 : *Trovato!*

## *Proprietà degli ABR*

Per ogni nodo  $X$ , i valori nei **nodi del sottoalbero sinistro** sono tutti minori del valore nel nodo  $X$ , e tutti i valori nei **nodi del sotto-albero destro** sono maggiori del valore di  $X$

# Alberi binari di ricerca

In generale, la **ricerca** è confinata ai **nodi** posizionati **lungo un singolo percorso** (path) dalla radice ad una foglia

Tempo di ricerca =  $O(h)$



## *L'ADT albero binario di ricerca*

- *È una specializzazione dell'ADT albero binario*
- *Gli **elementi statici** sono essenzialmente gli stessi, l'unica differenza è che si assume che i dati contenuti (le **chiavi**) siano ordinabili secondo qualche **relazione d'ordine**.*

```
typedef *nodo ARB;  
struct {  
  ARB padre;  
  elemento key;  
  ARB figlio_destro, figlio_sinsitro;  
} nodo;
```



# L'ADT albero binario di ricerca

## ☆ Selettori:

- *root(T)*
- *figlio\_destro(T)*
- *figlio\_sinistro(T)*
- *key(T)*

## 🕒 Costruttori/Distruttori:

- *crea\_albero()*
- *ARB\_inserisci(T,x)*
- *ARB\_cancella (T,x)*

## 🕒 Proprietà:

- *vuoto(T)*

## 🕒 Operazioni di Ricerca

- *ARB\_ricerca(T,k)*
- *ARB\_minimo(T)*
- *ARB\_massimo(T)*
- *ARB\_successore(T,x)*
- *ARB\_predecessore(T,x)*

## *Ricerca in Alberi binari di ricerca*

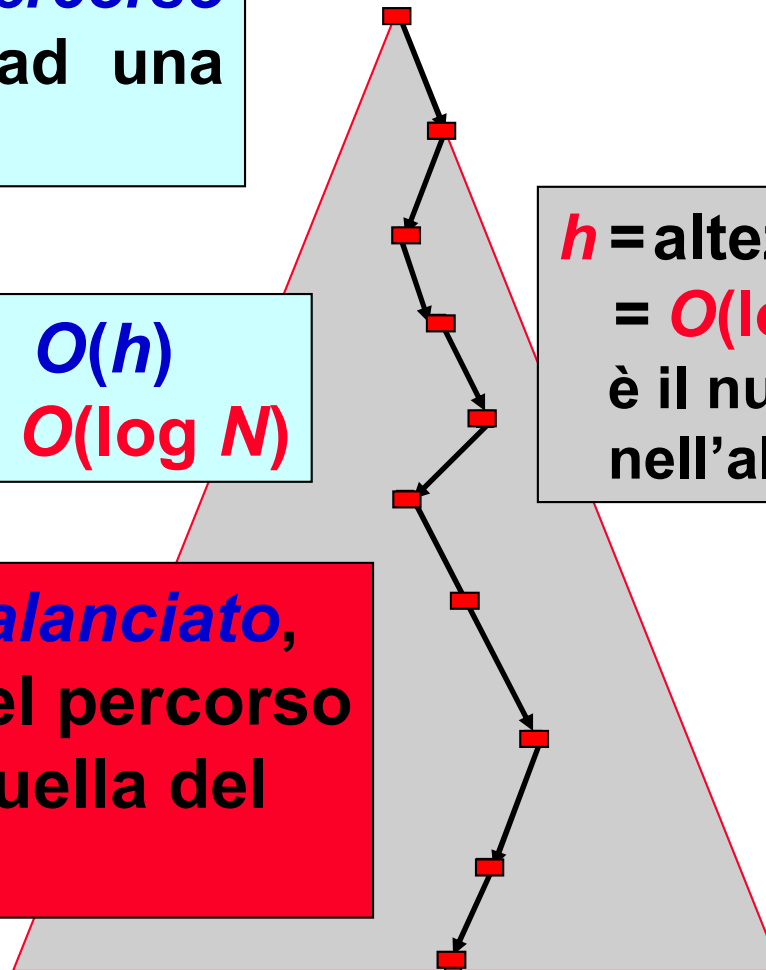
```
ARB_ricerca(T, k)
  IF T = NIL OR k = Key[T]
    THEN return T
  IF k < Key[T]
    THEN
      ARB_ricerca(figlio_sinistro[T], k)
  ELSE
      ARB_ricerca(figlio_destro[T], k)
```

# Ricerca in Alberi binari di ricerca

In generale, la **ricerca** è confinata ai **nodi** posizionati **lungo un singolo percorso** (**path**) dalla radice ad una foglia

Tempo di ricerca =  $O(h)$   
=  $O(\log N)$

Solo se l'albero è **balanciato**, cioè la lunghezza del percorso minimo è vicino a quella del percorso massimo

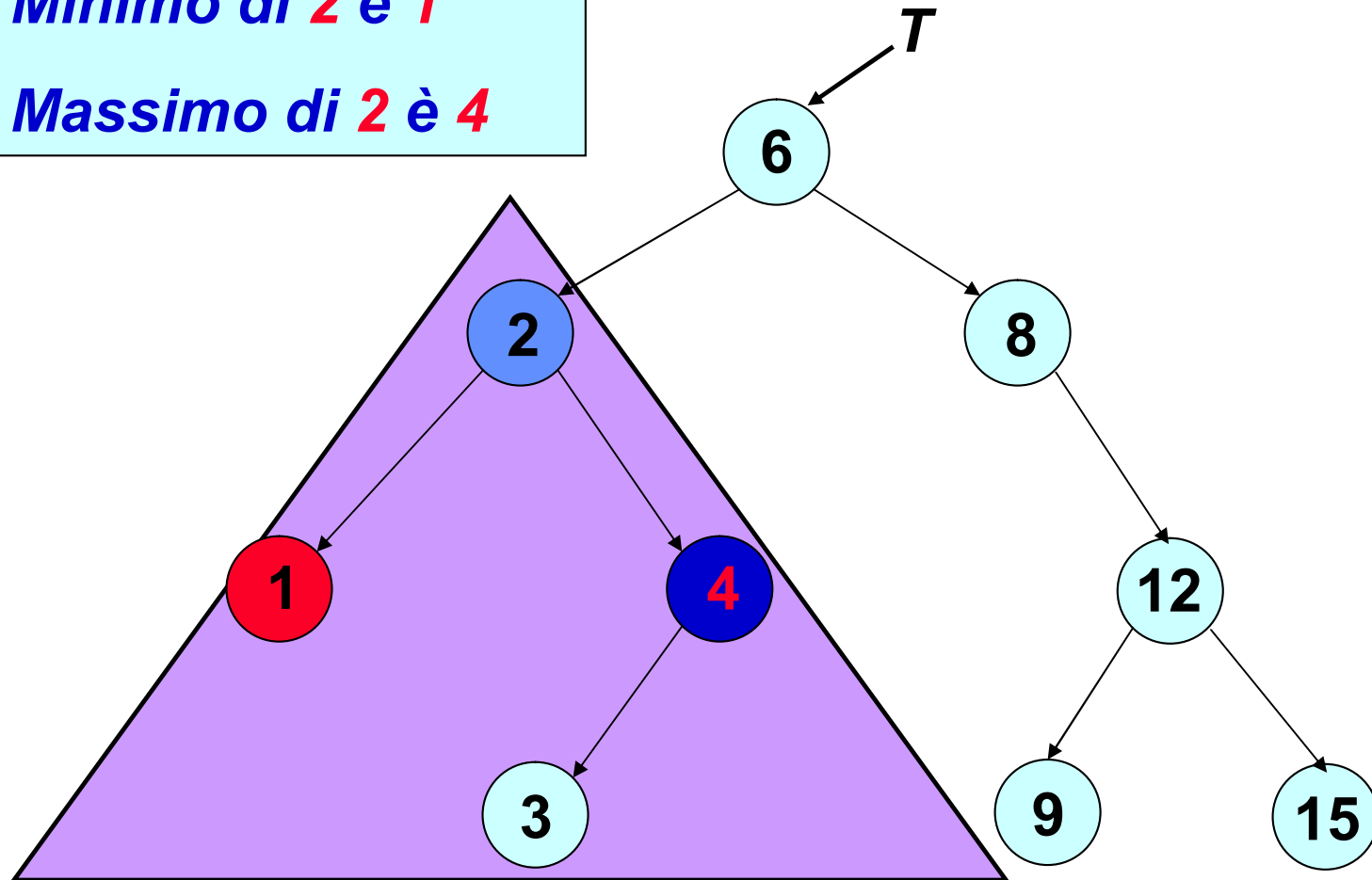


$h$  = altezza dell'albero  
=  $O(\log N)$  dove  $N$   
è il numero di nodi  
nell'albero

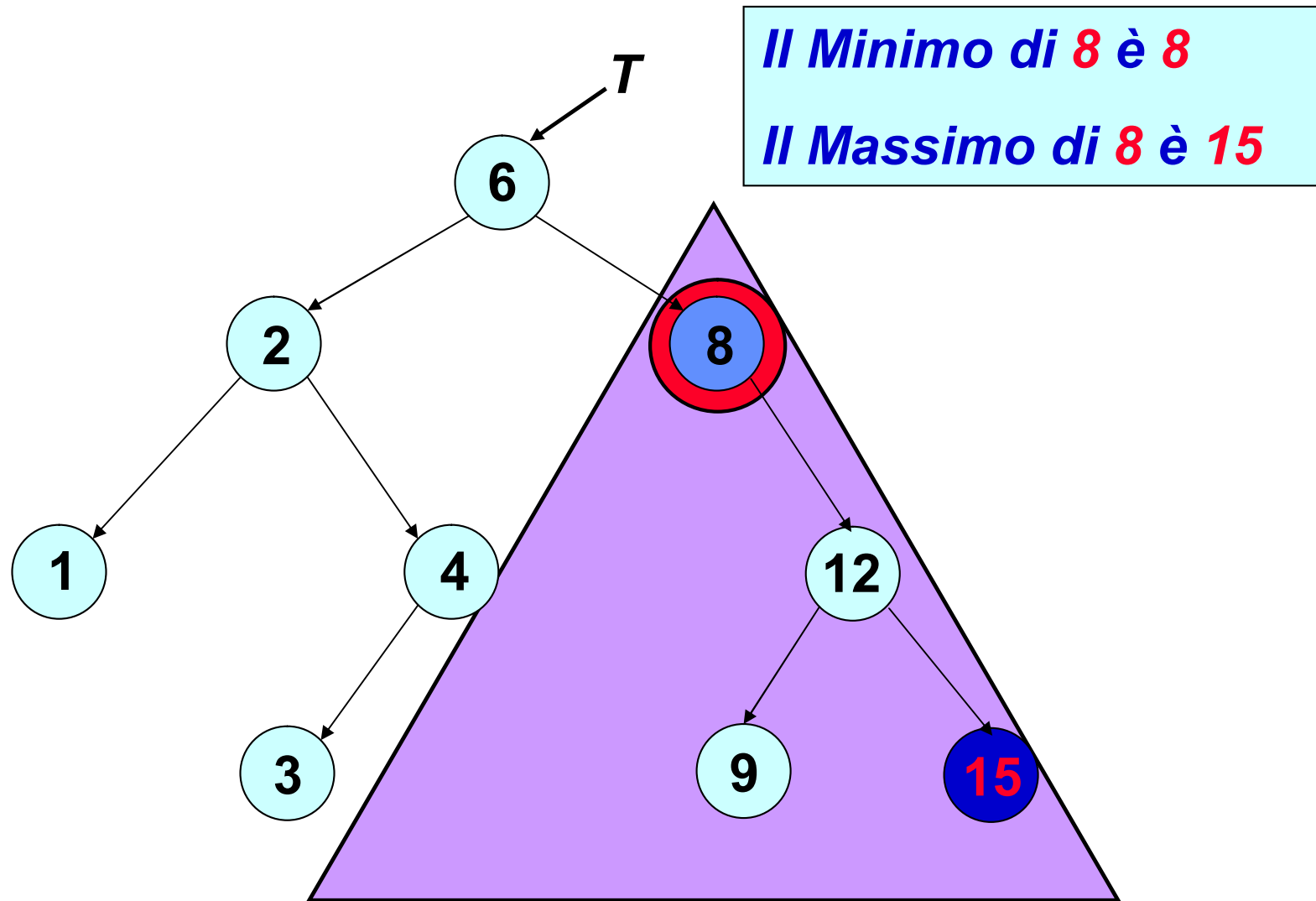
# ARB: ricerca del minimo e massimo

*Il Minimo di 2 è 1*

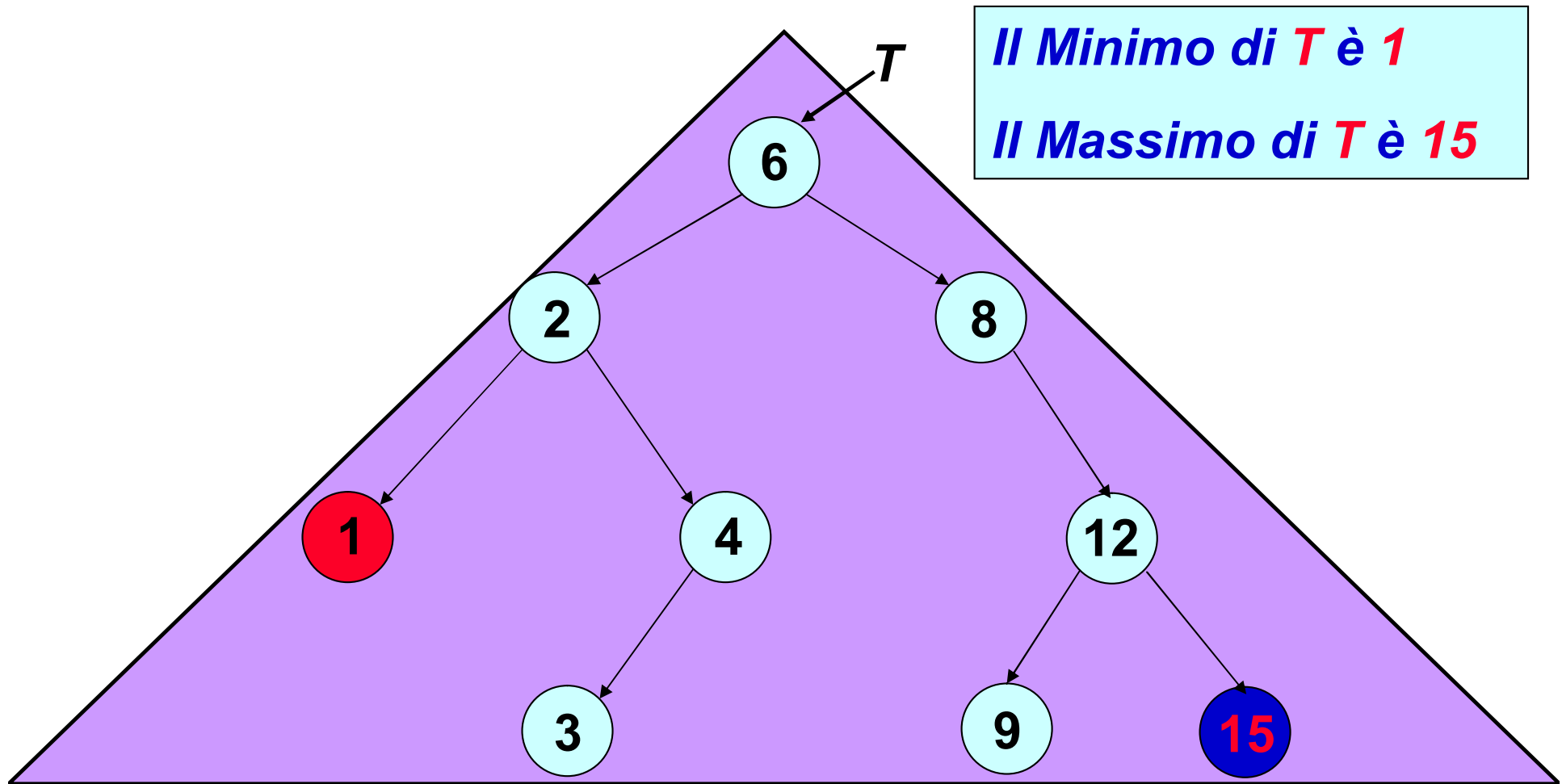
*Il Massimo di 2 è 4*



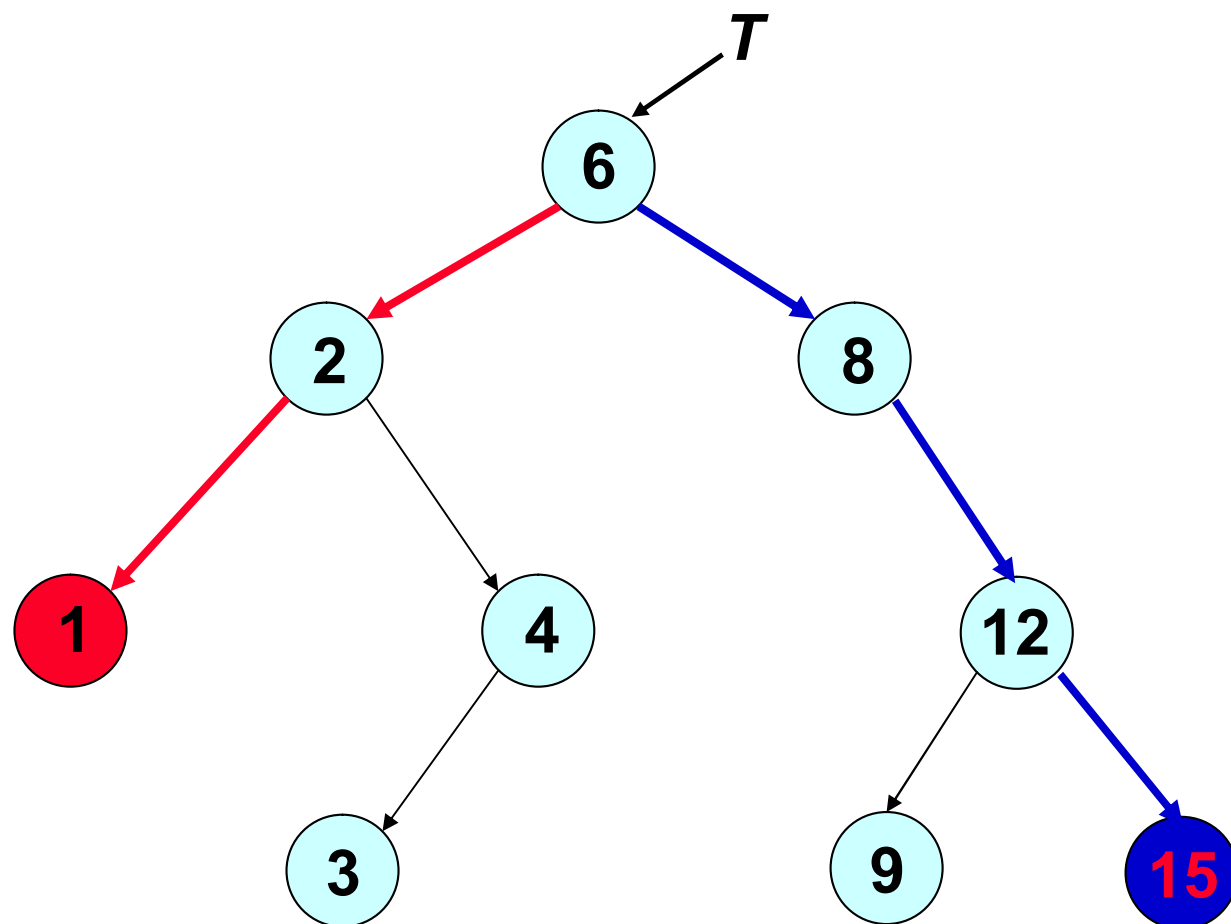
# ARB: ricerca del minimo e massimo



# ARB: ricerca del minimo e massimo



## *ARB: ricerca del minimo e massimo*



## *ARB: ricerca del minimo e massimo*

```
ARB ABR-Minimo(x:ARB)
  WHILE figlio-sinistro[x] ≠ NIL
    DO x = figlio-sinistro[x]
  return x
```

```
ARB ABR-Massimo(x: ARB)
  WHILE figlio-destro[x] ≠ NIL
    DO x = figlio-destro[x]
  return x
```

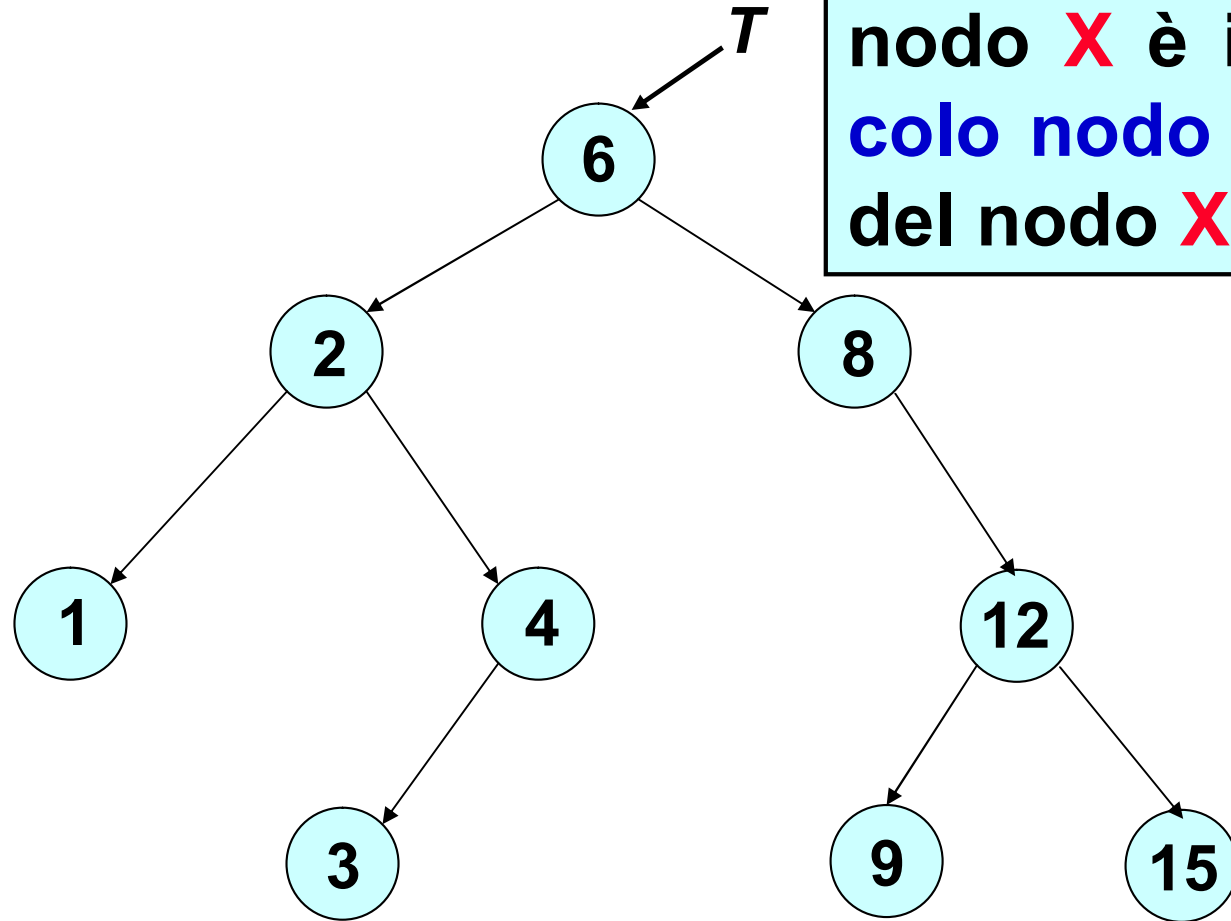
## *ARB: ricerca del minimo e massimo*

```
ARB ABR-Minimo(x:ARB)
  WHILE figlio-sinistro[x] ≠ NIL
    DO x = figlio-sinistro[x]
  return x
```

```
ARB ABR-Massimo(x: ARB)
  WHILE figlio-destro[x] ≠ NIL
    DO x = figlio-destro[x]
  return x
```

```
ARB ARB_Minimo(x:ARB)
  IF vuoto(figlio_sinistro[x]) Then
    return x
  Else ARB_Minimo(figlio_sinistro[x])
```

## ARB: ricerca del successore

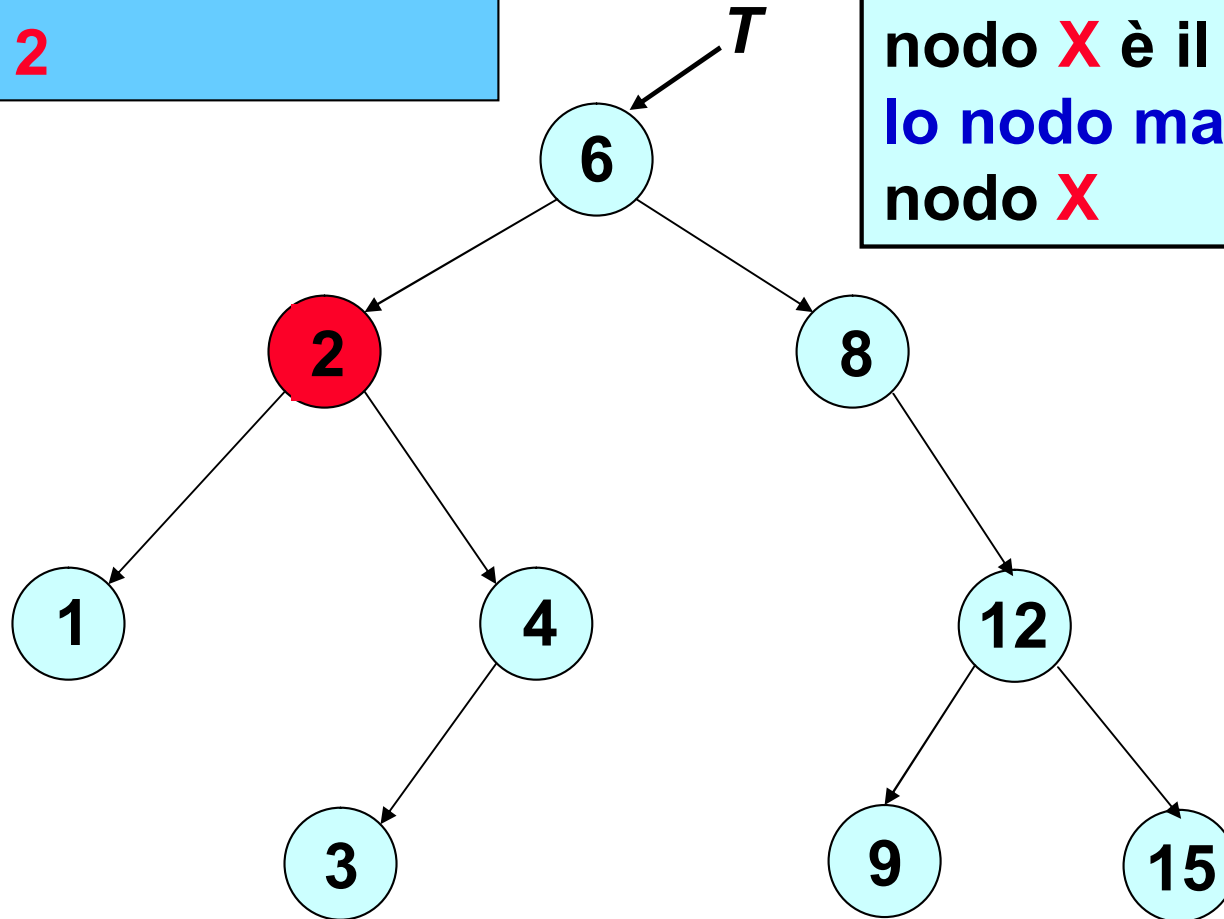


Il **successore** di un nodo **X** è il più piccolo nodo maggiore del nodo **X**

# ARB: ricerca del successore

**Ricerca** del successore  
del nodo **2**

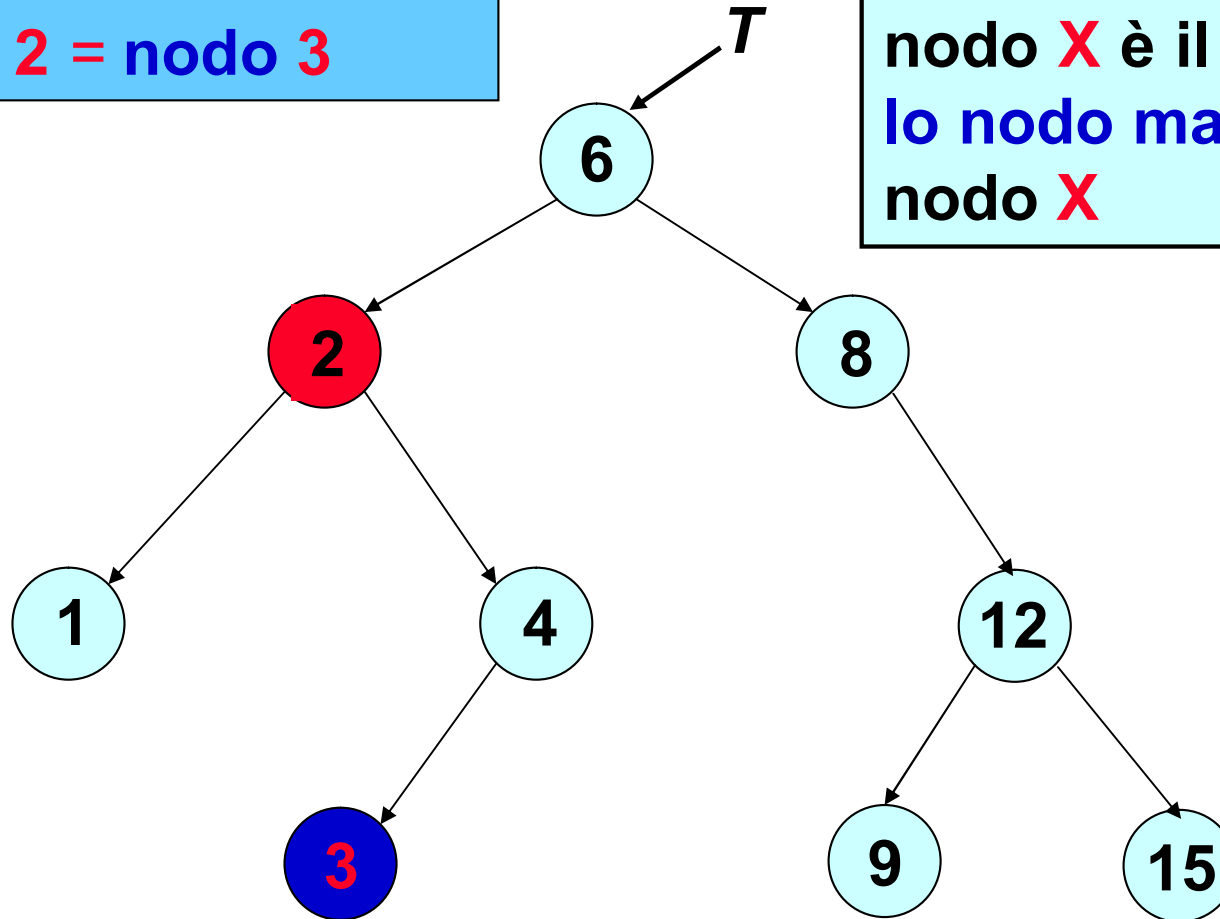
Il **successore** di un  
nodo **X** è il **più piccolo**  
nodo **maggiore** del  
nodo **X**



# ARB: ricerca del successore

**Ricerca** del successore  
del nodo **2** = nodo **3**

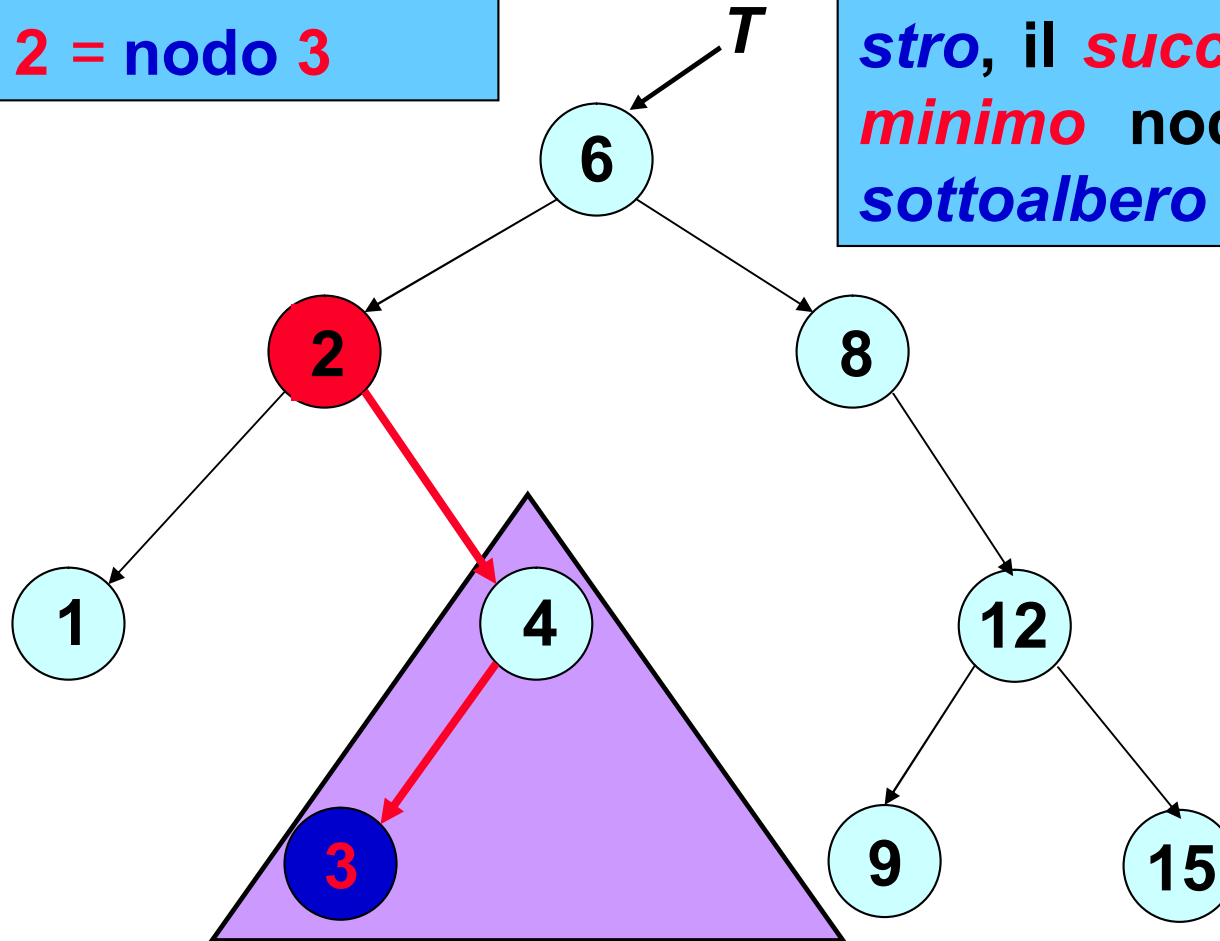
Il **successore** di un  
nodo **X** è il più picco-  
lo nodo maggiore del  
nodo **X**



# ARB: ricerca del successore

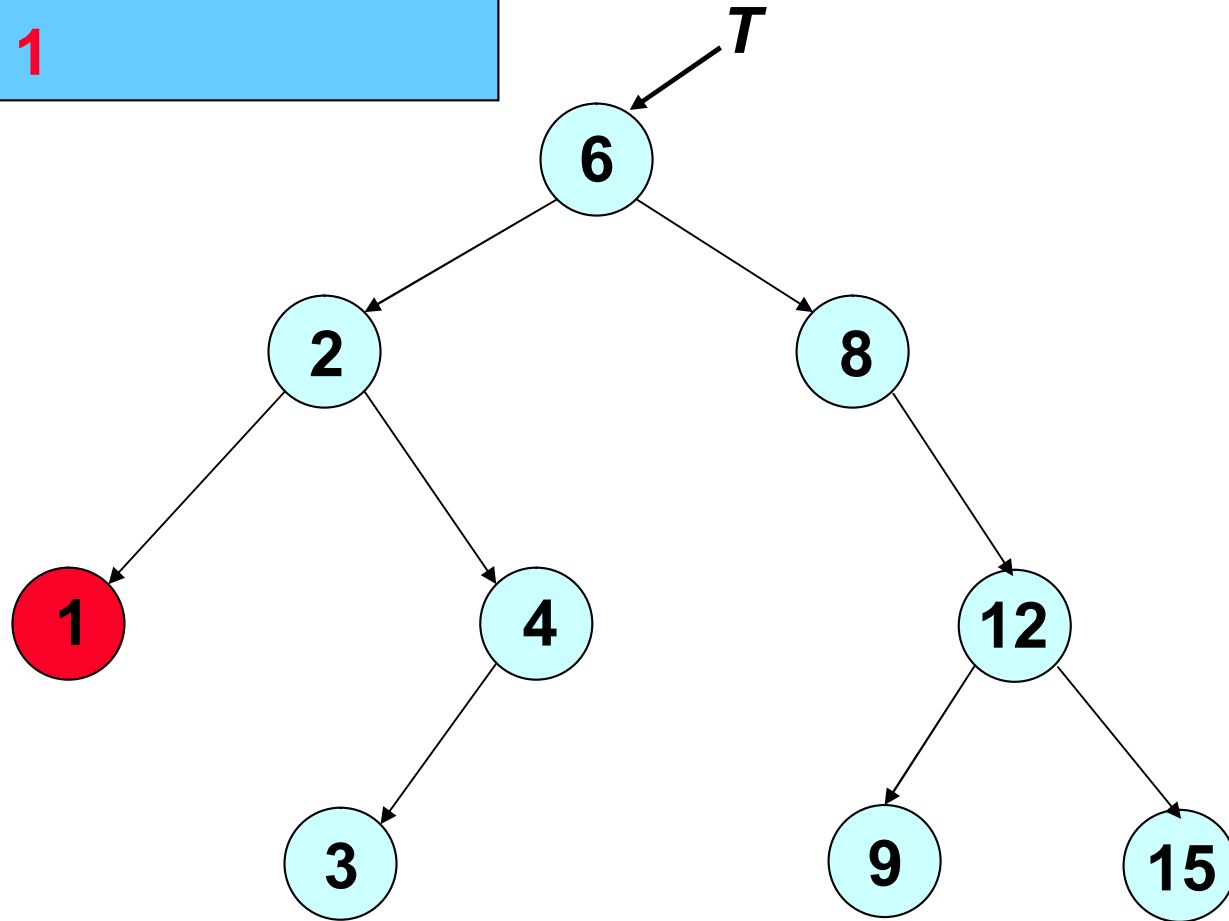
**Ricerca** del successore  
del nodo **2** = nodo **3**

Se **x** ha un **figlio de-**  
**stro**, il **successore** è il  
**minimo** nodo di quel  
**sottoalbero**



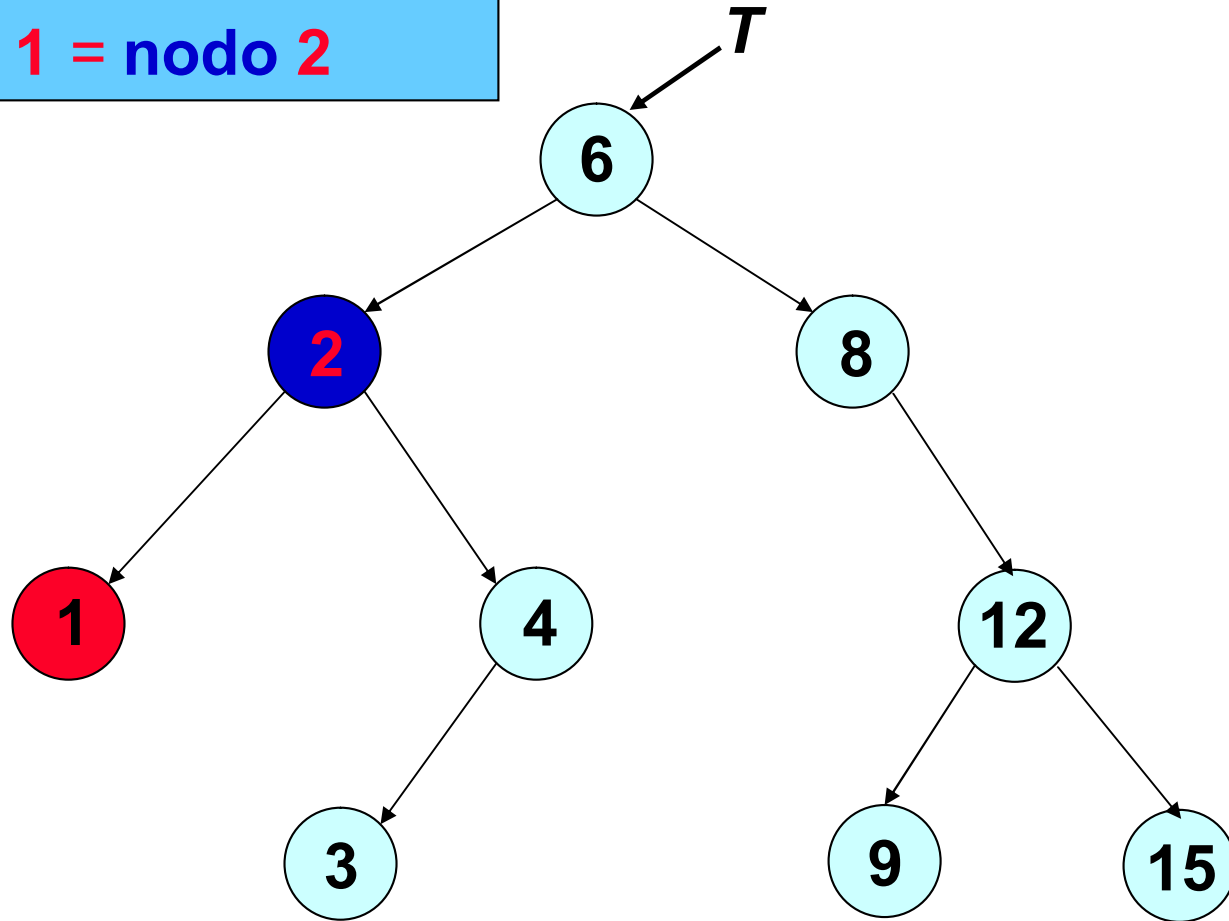
# ARB: ricerca del successore

**Ricerca** del successore  
del nodo **1**



# ARB: ricerca del successore

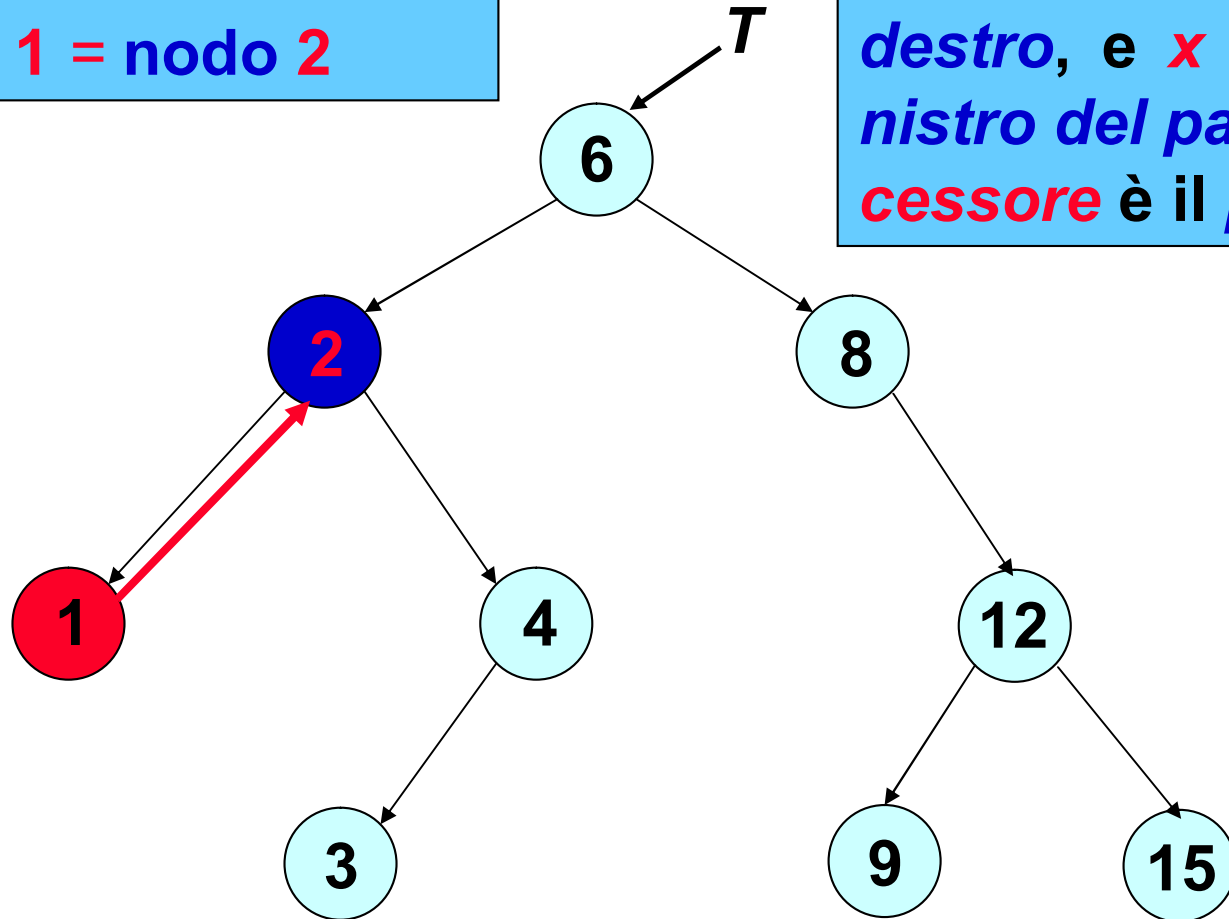
**Ricerca** del successore  
del nodo **1** = nodo **2**



# ARB: ricerca del successore

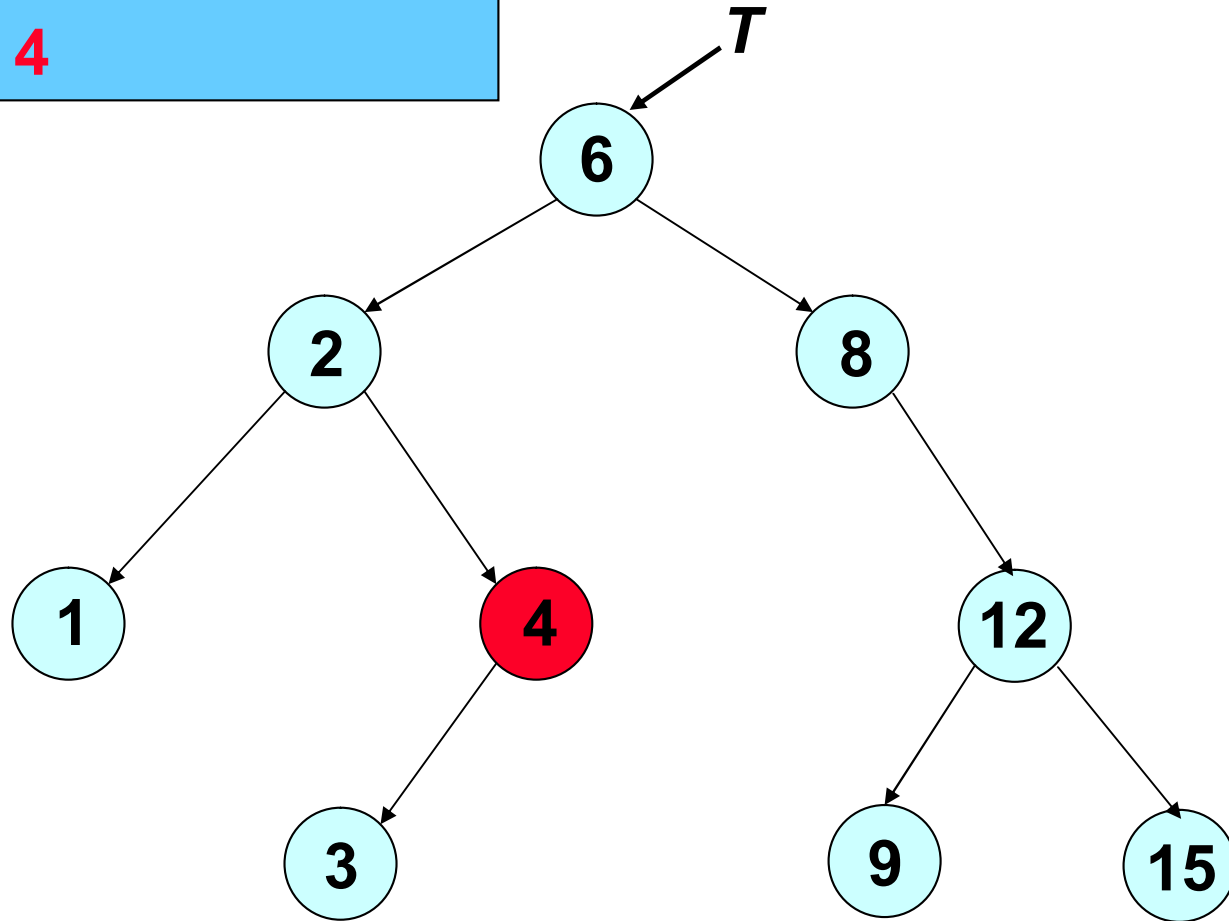
*Ricerca* del successore  
del nodo 1 = nodo 2

Se  $x$  NON ha un *figlio destro*, e  $x$  è *figlio sinistro del padre*, il *successore* è il *padre*.



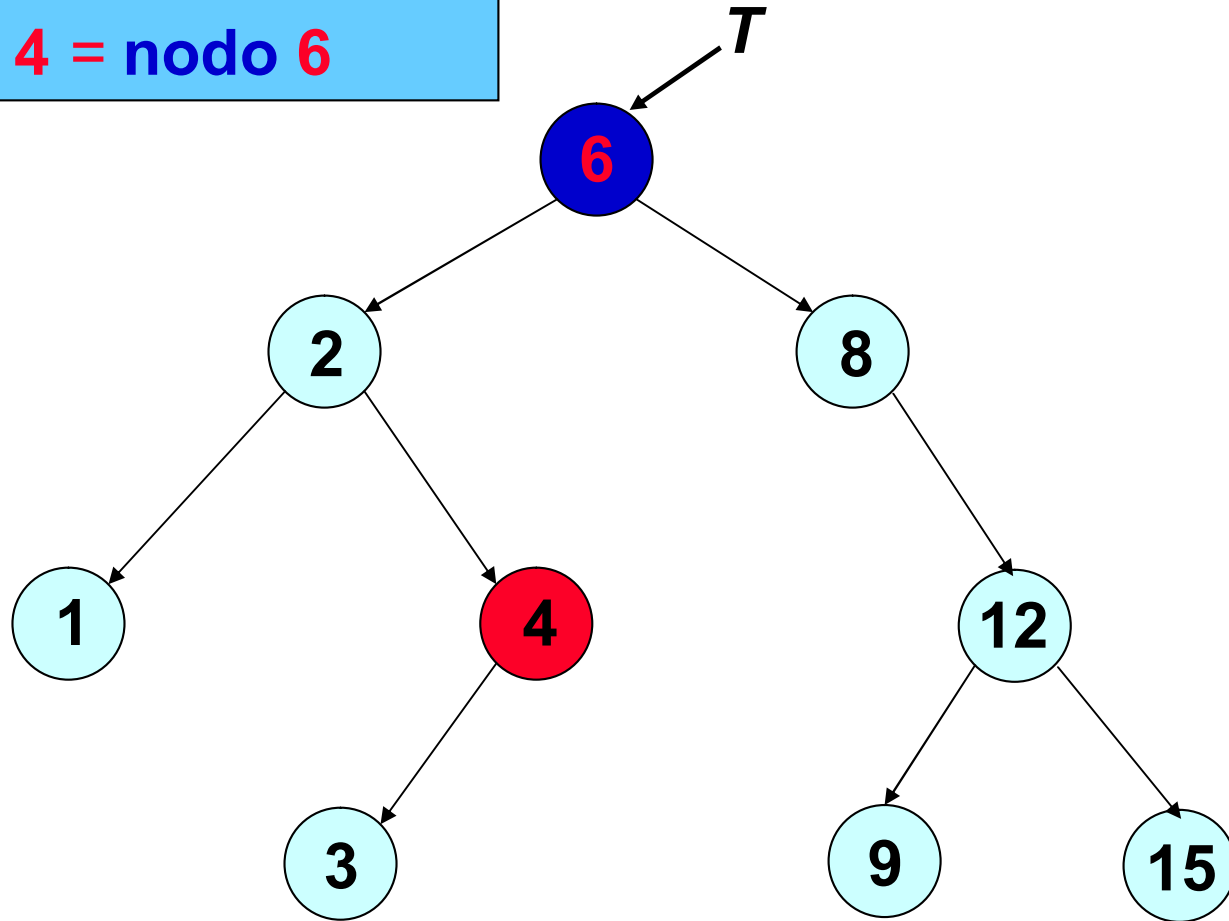
# ARB: ricerca del successore

**Ricerca** del successore  
del nodo **4**



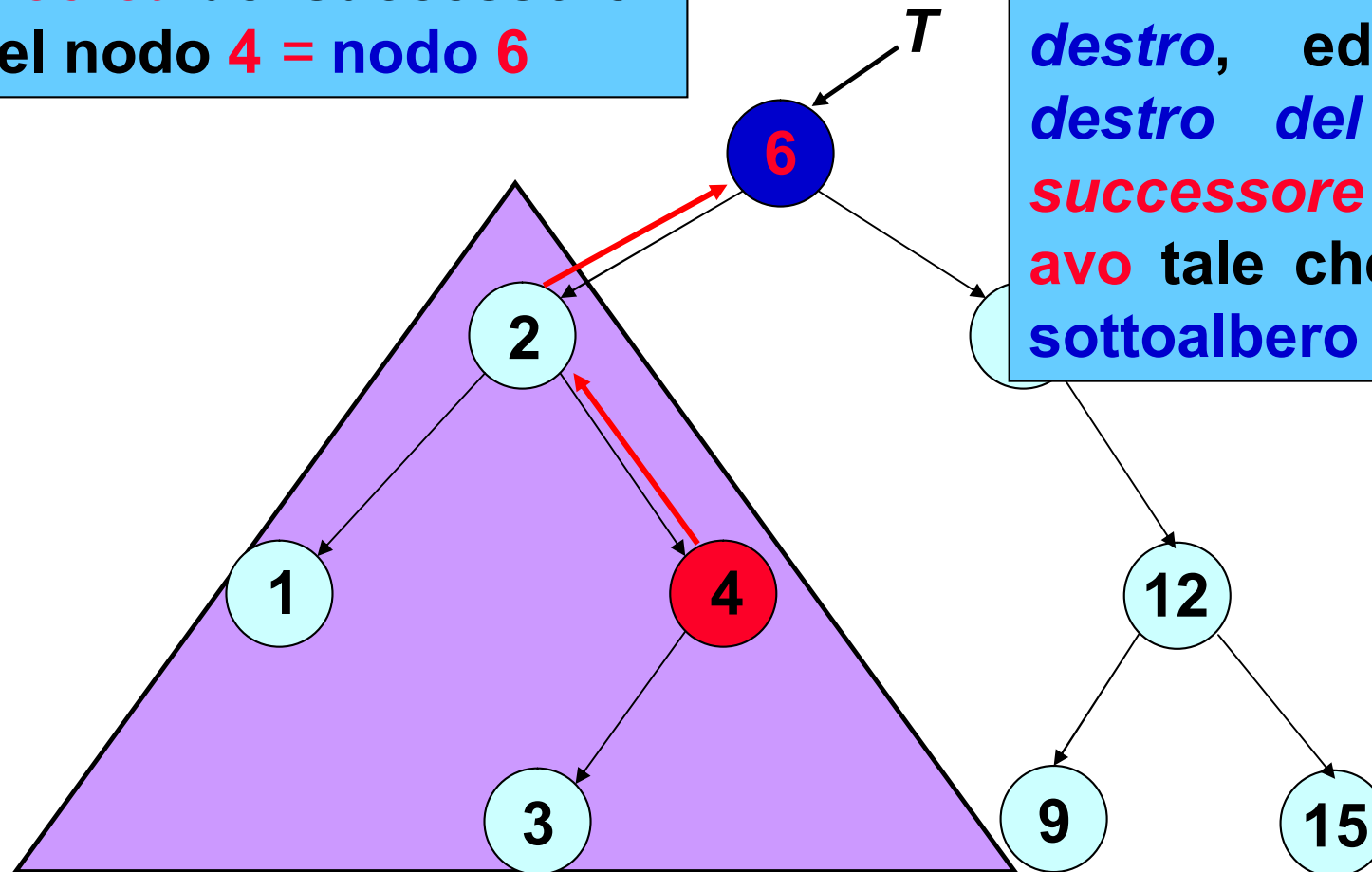
# ARB: ricerca del successore

**Ricerca** del successore  
del nodo **4** = nodo **6**



# ARB: ricerca del successore

**Ricerca** del successore  
del nodo **4** = nodo **6**



Se **x** NON ha un *figlio destro*, ed è *figlio destro del padre*, il **successore** è il primo **avo** tale che **x** sta nel *sottoalbero sinistro*.

## *ARB: ricerca del successore*

```
ABR-Successore(x)
```

```
  IF figlio-destro[x] ≠ NIL
```

```
    THEN return ABR-Minimo(figlio-destro[x])
```

```
  y = padre[x]
```

```
  WHILE y ≠ NIL AND x = figlio-destro[y]
```

```
    DO x = y
```

```
      y = padre[y]
```

```
  return y
```

## *ARB: ricerca del successore*

```
ABR-Successore(x)
```

```
  IF figlio-destro[x] ≠ NIL
```

```
    THEN return ABR-Minimo(figlio-destro[x])
```

```
  y = padre[x]
```

```
  WHILE y ≠ NIL AND x = figlio-destro[y]
```

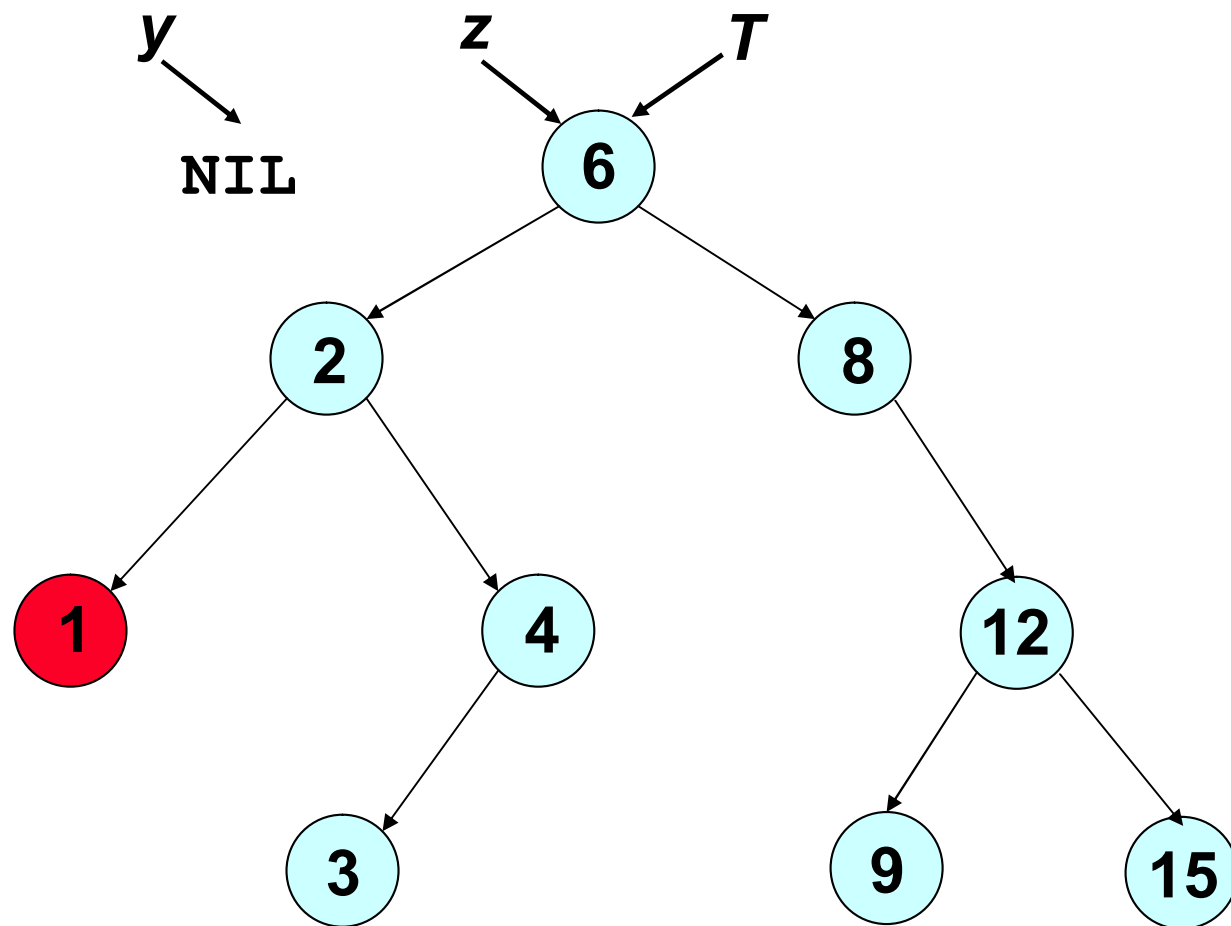
```
    DO x = y
```

```
      y = padre[y]
```

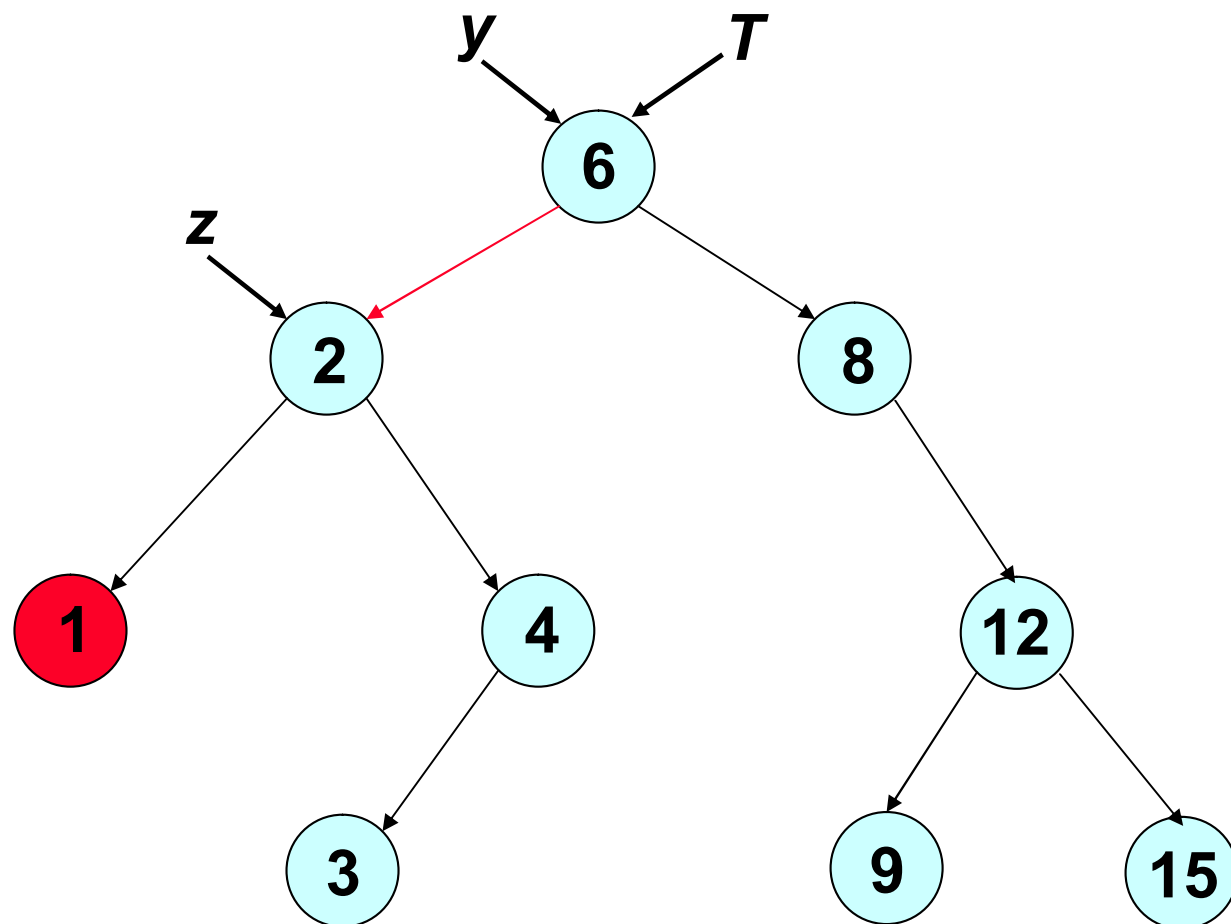
```
  return y
```

Questo algoritmo **assume** che ogni nodo abbia il **puntatore al padre**

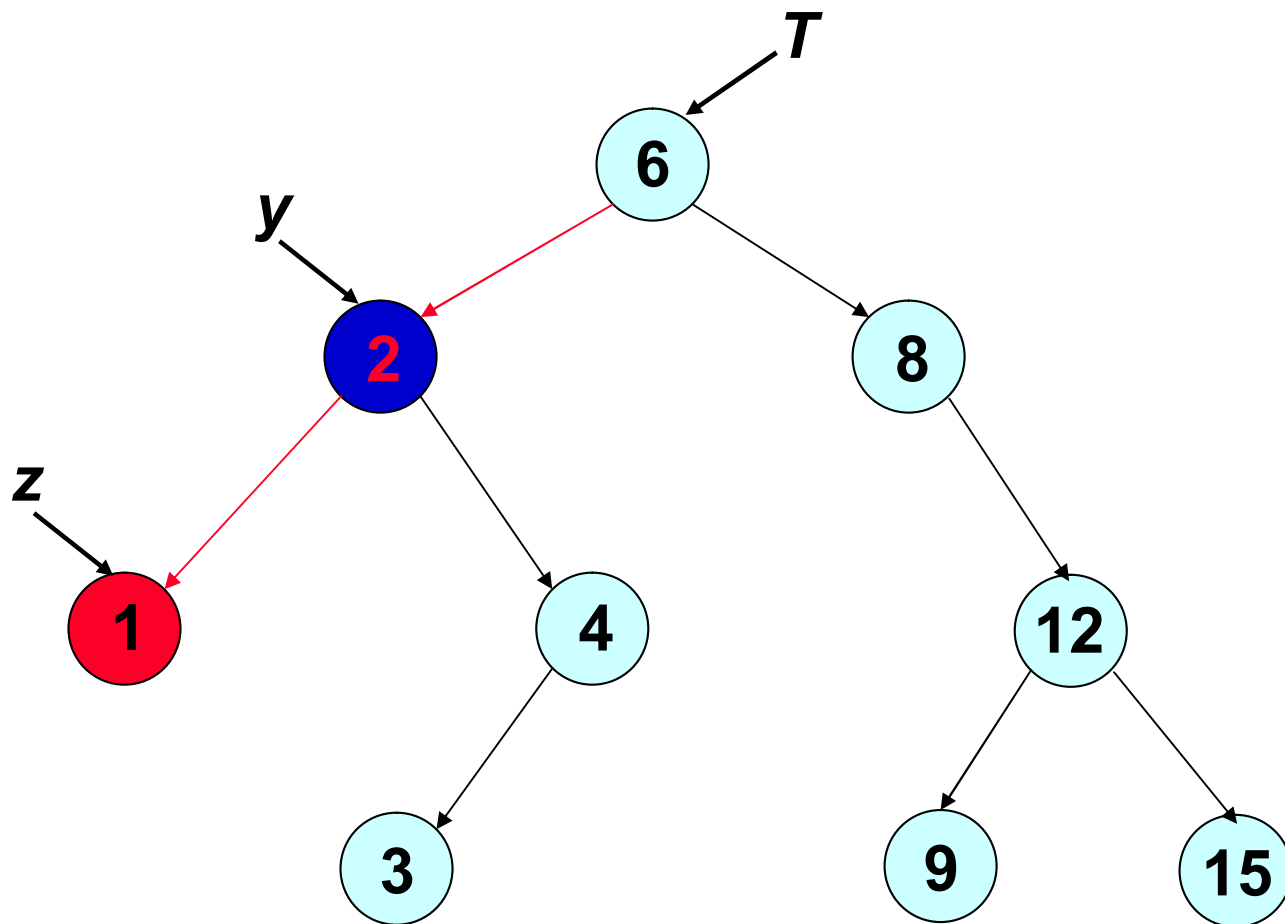
## *ARB: ricerca del successore II*



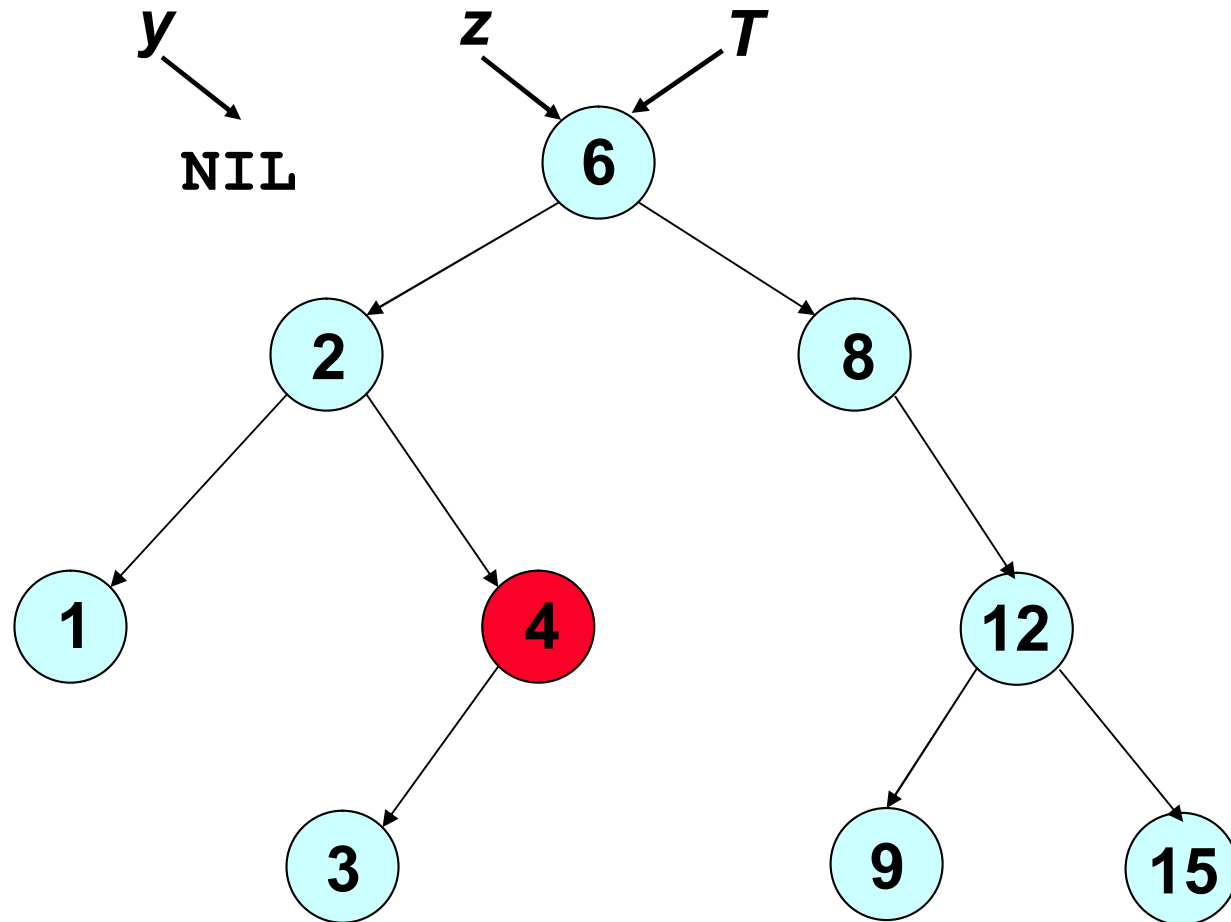
## *ARB: ricerca del successore II*



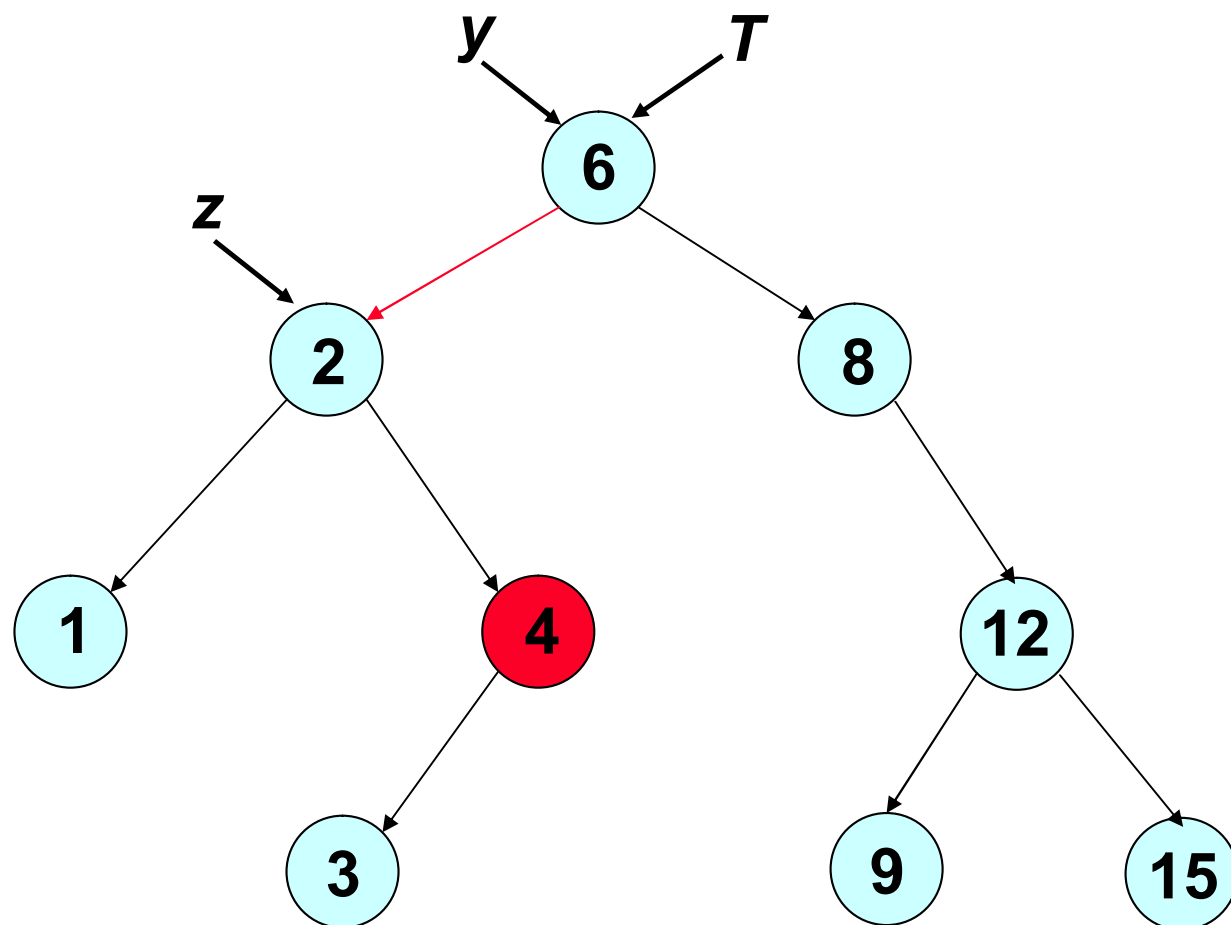
## *ARB: ricerca del successore II*



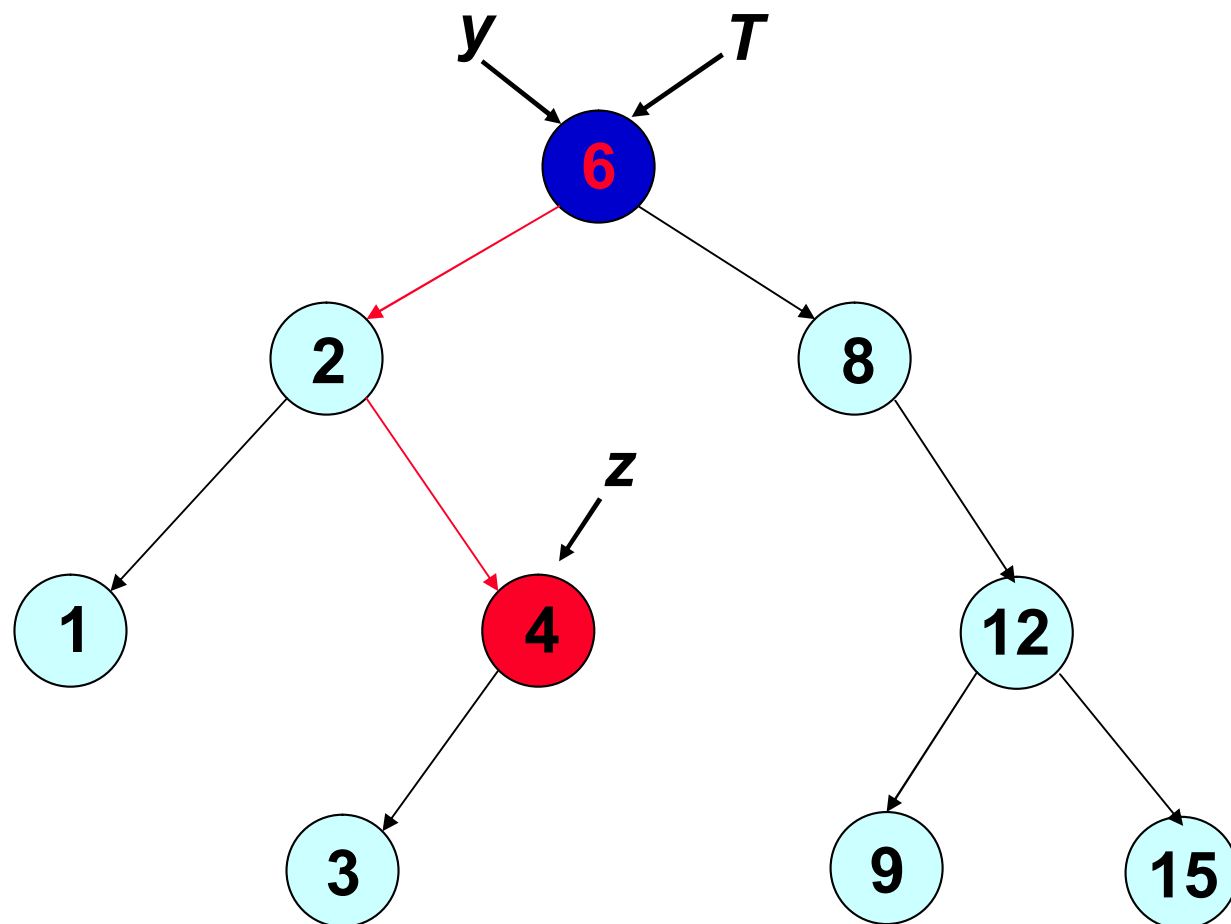
## *ARB: ricerca del successore II*



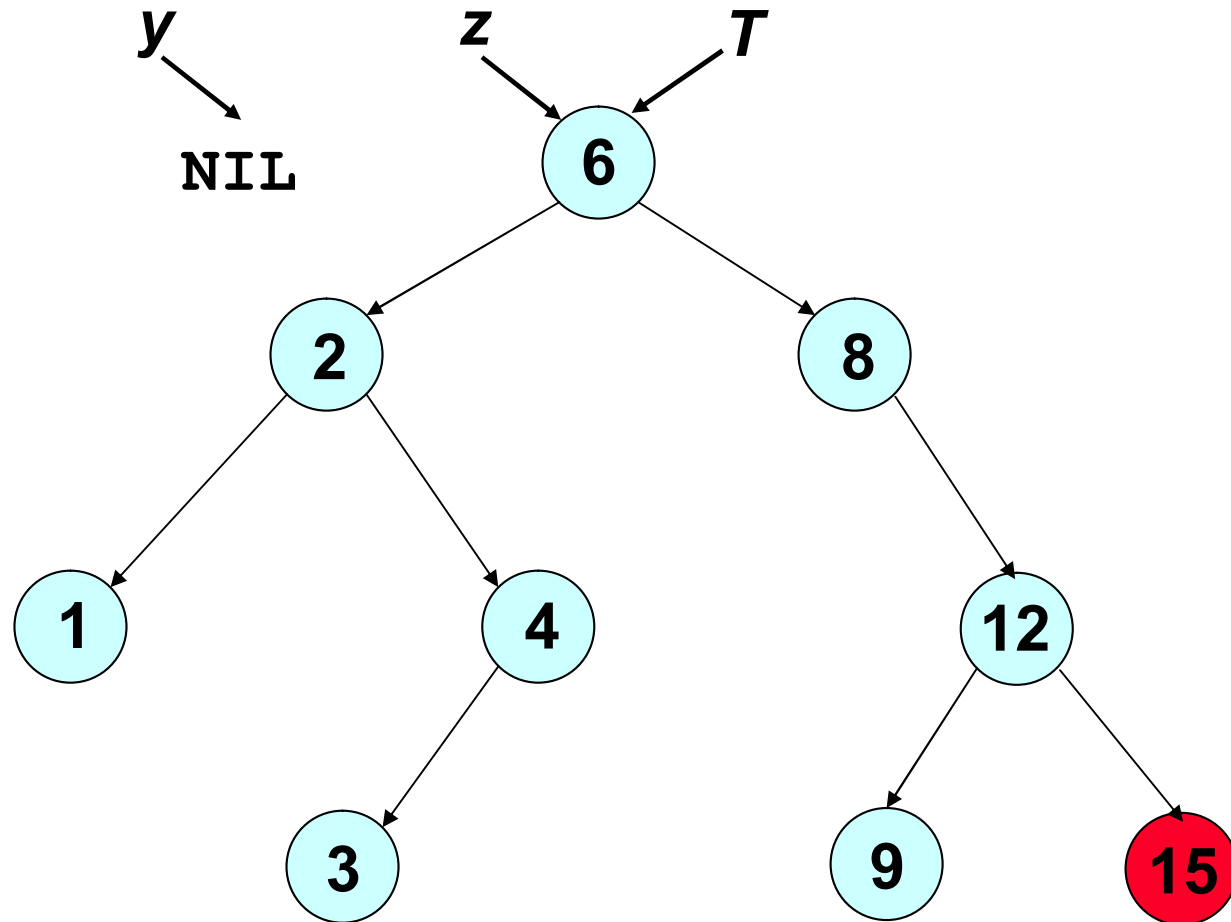
## *ARB: ricerca del successore II*



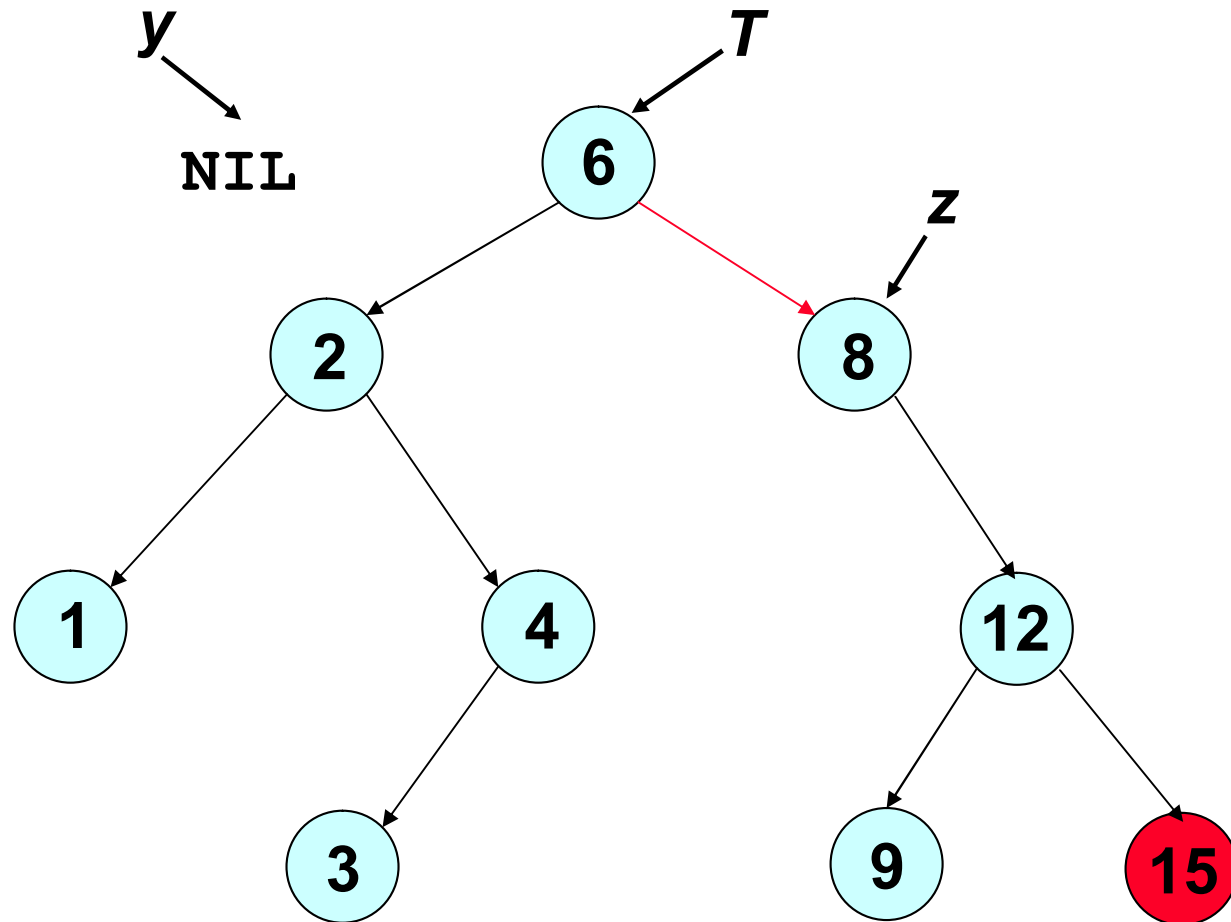
## *ARB: ricerca del successore II*



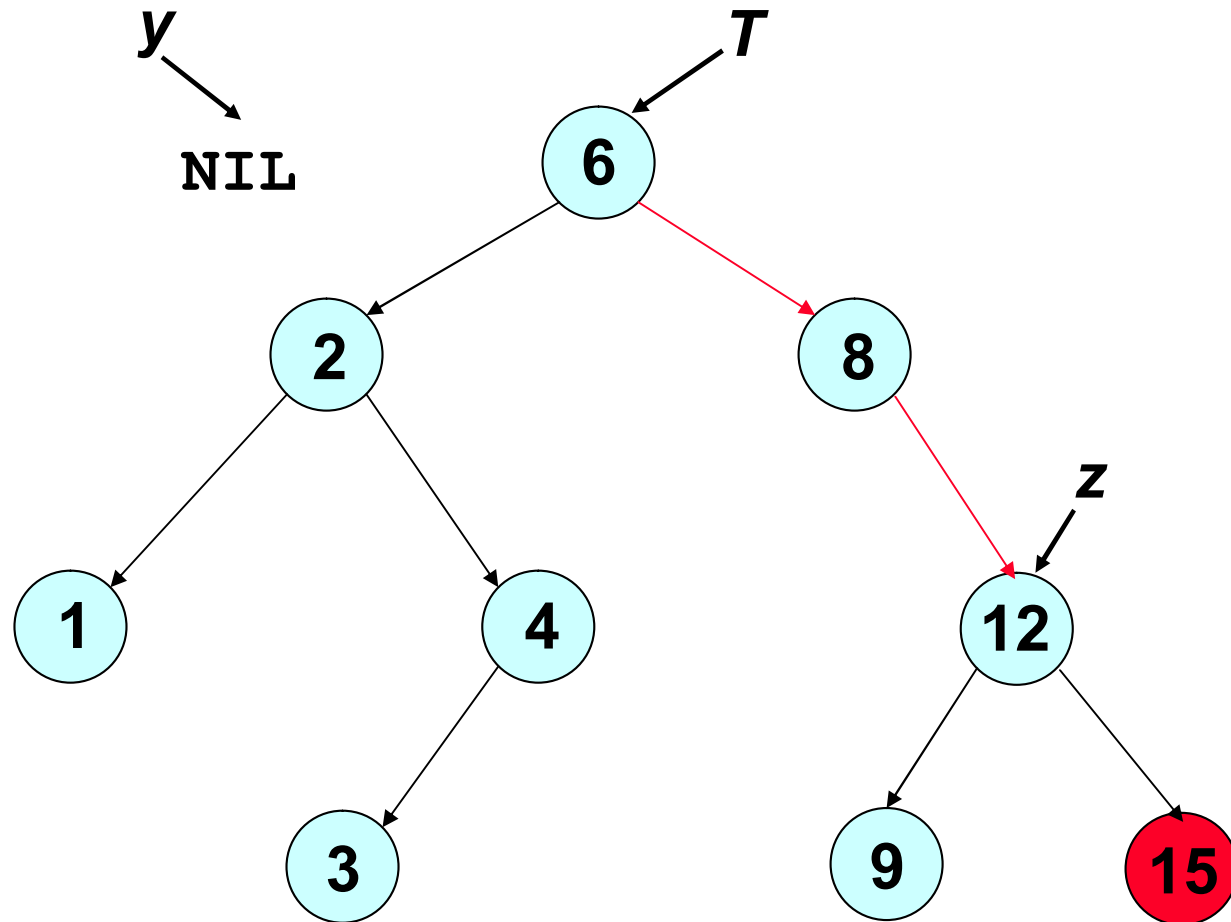
## *ARB: ricerca del successore II*



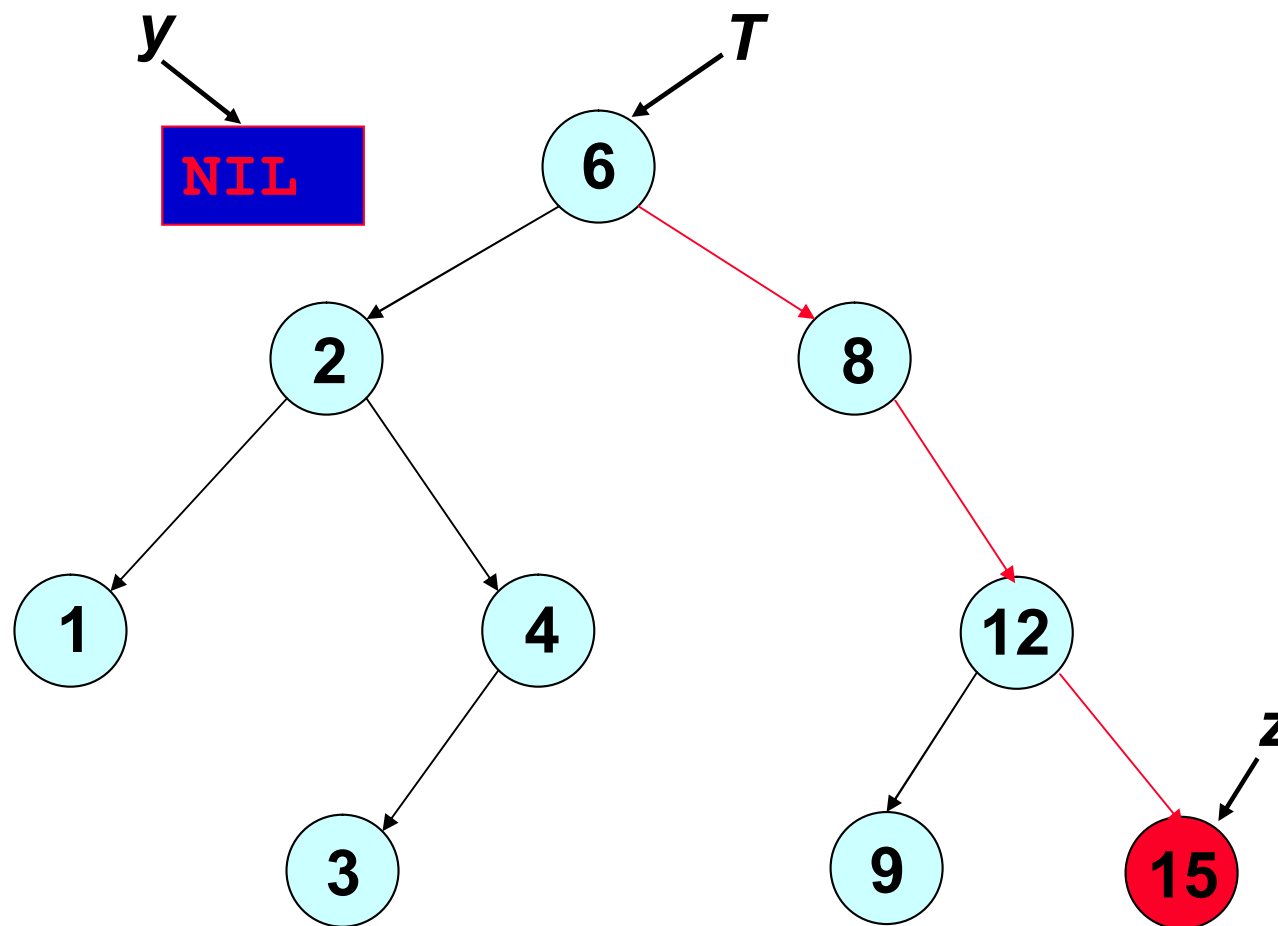
## ARB: ricerca del successore II



## *ARB: ricerca del successore II*



## ARB: ricerca del successore II



## *ARB: ricerca del successore II*

- Inizializzo il **successore** a **NIL**
- **Partendo dalla radice dell'albero:**
  - ogni volta che seguo un **ramo sinistro** per raggiungere il nodo, **aggiorno il successore al nodo padre;**
  - ogni volta che seguo un **ramo destro** per raggiungere il nodo, **NON** aggiorno il successore al **nodo padre;**

## *ARB: ricerca del successore*

```
ARB ABR-Successore' (x: ARB)
  IF figlio-destro[x] ≠ NIL
    THEN return ABR-Minimo(figlio-destro[x])
  z = Root[T]
  y = NIL
  WHILE z ≠ x
    DO IF key[z] < key[x]
      THEN z = figlio-destro[z]
      ELSE y = z
         z = figlio-sinistro[z]
  return y
```

## ARB: ricerca del successore ricorsiva

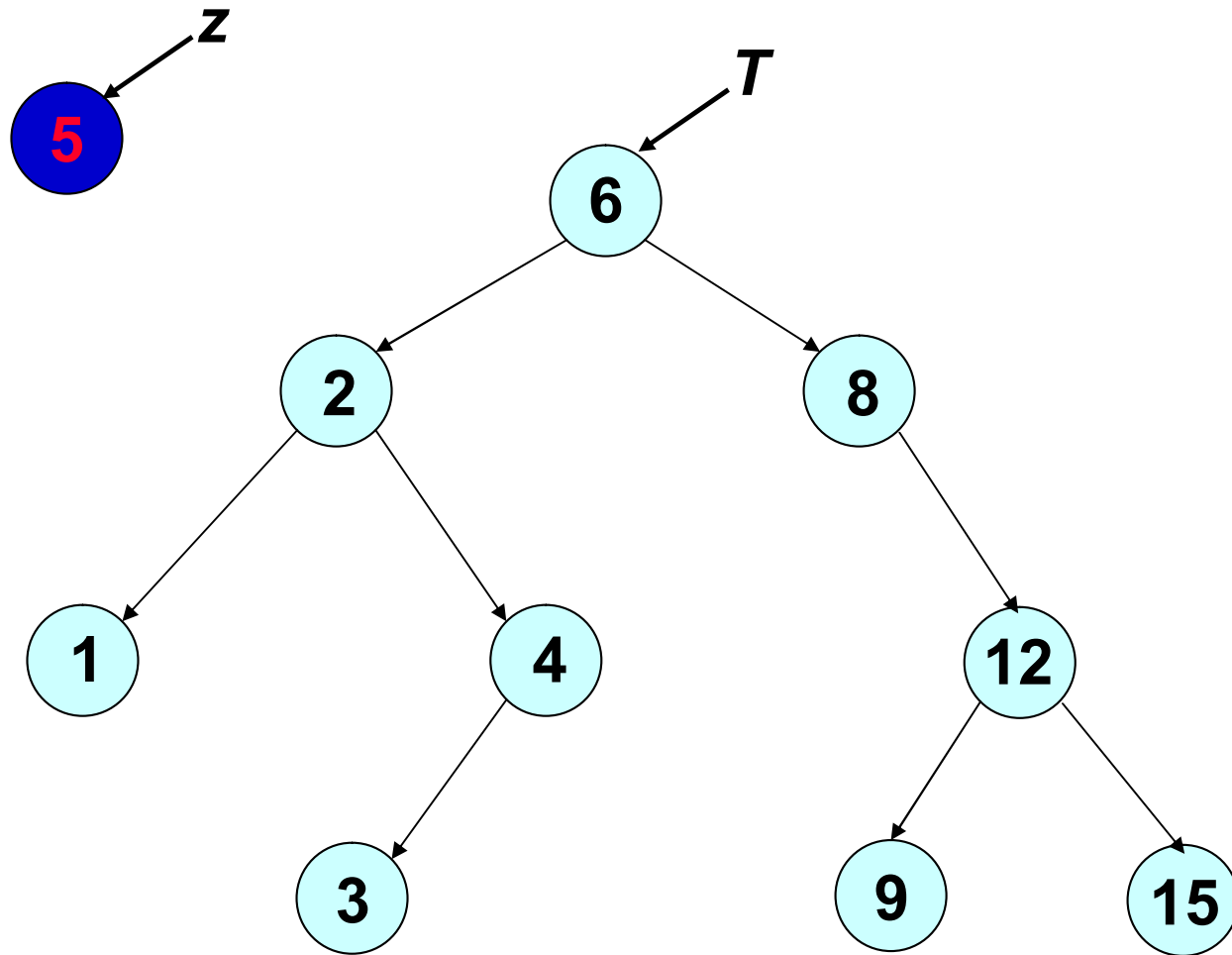
```
ARB ABR-Successore (x: ARB)
  return ABR-Successore_ric (key[x], root[x], NIL)
```

```
ABR-Successore_ric (key, T, P_T)
  IF T ≠ NIL THEN
    IF key > key[T] THEN
      return ABR-Successore_ric (key,
                                  figlio-destro[T], P_T)
    ELSE IF key < key[T] THEN
      return ABR-Successore_ric (key,
                                  figlio-sinistro[T], T)
    ELSE IF figlio-destro[T] ≠ NIL THEN
      return ABR-Minimo (figlio-destro[x])
  return P_T
```

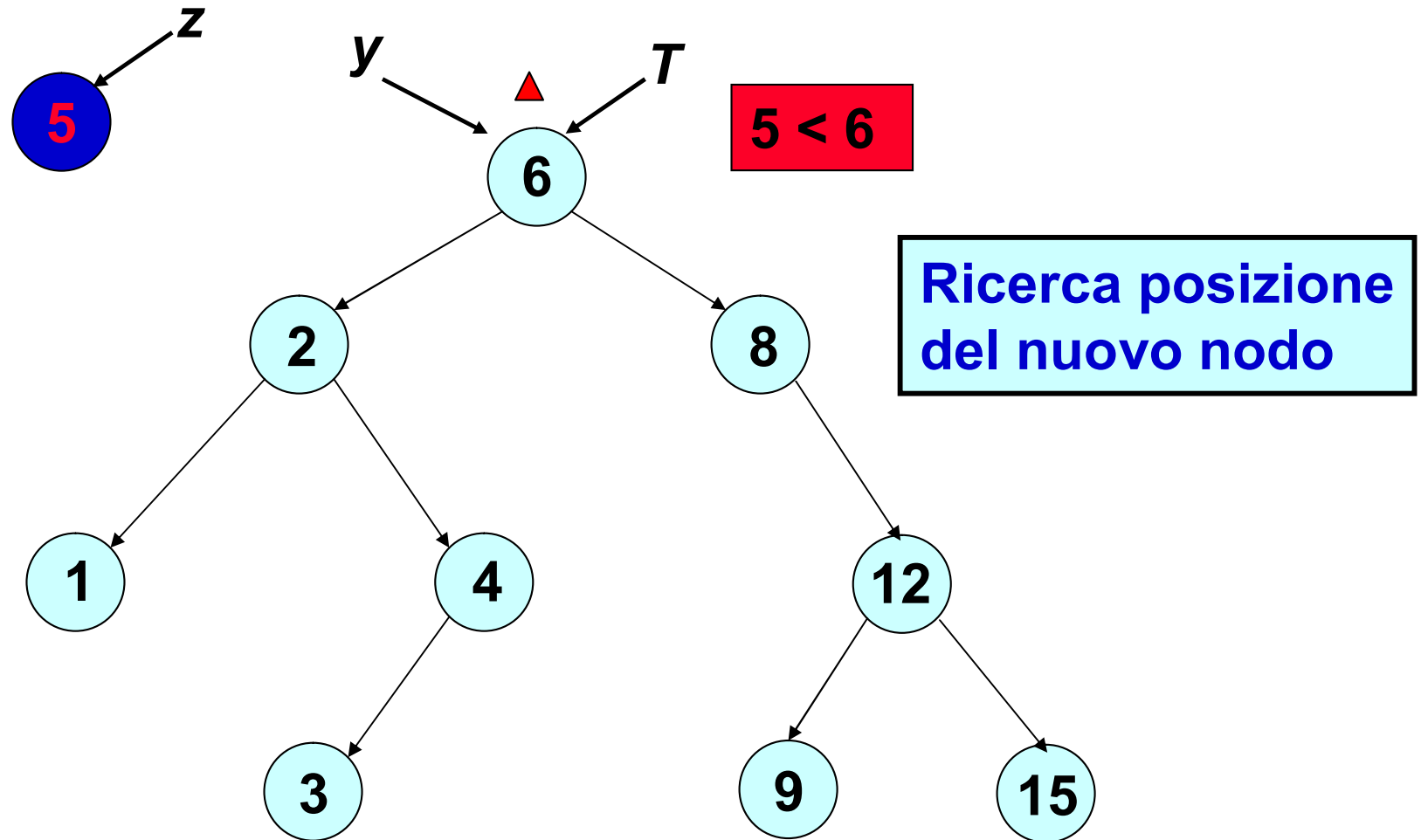
## *ARB: costo delle operazioni*

***Teorema.*** *Le operazioni di Ricerca, Minimo, Massimo, Successore e Predecessore sull'insieme dinamico Albero Binario di Ricerca possono essere eseguite in tempo  $O(h)$  dove  $h$  è l'altezza dell'albero*

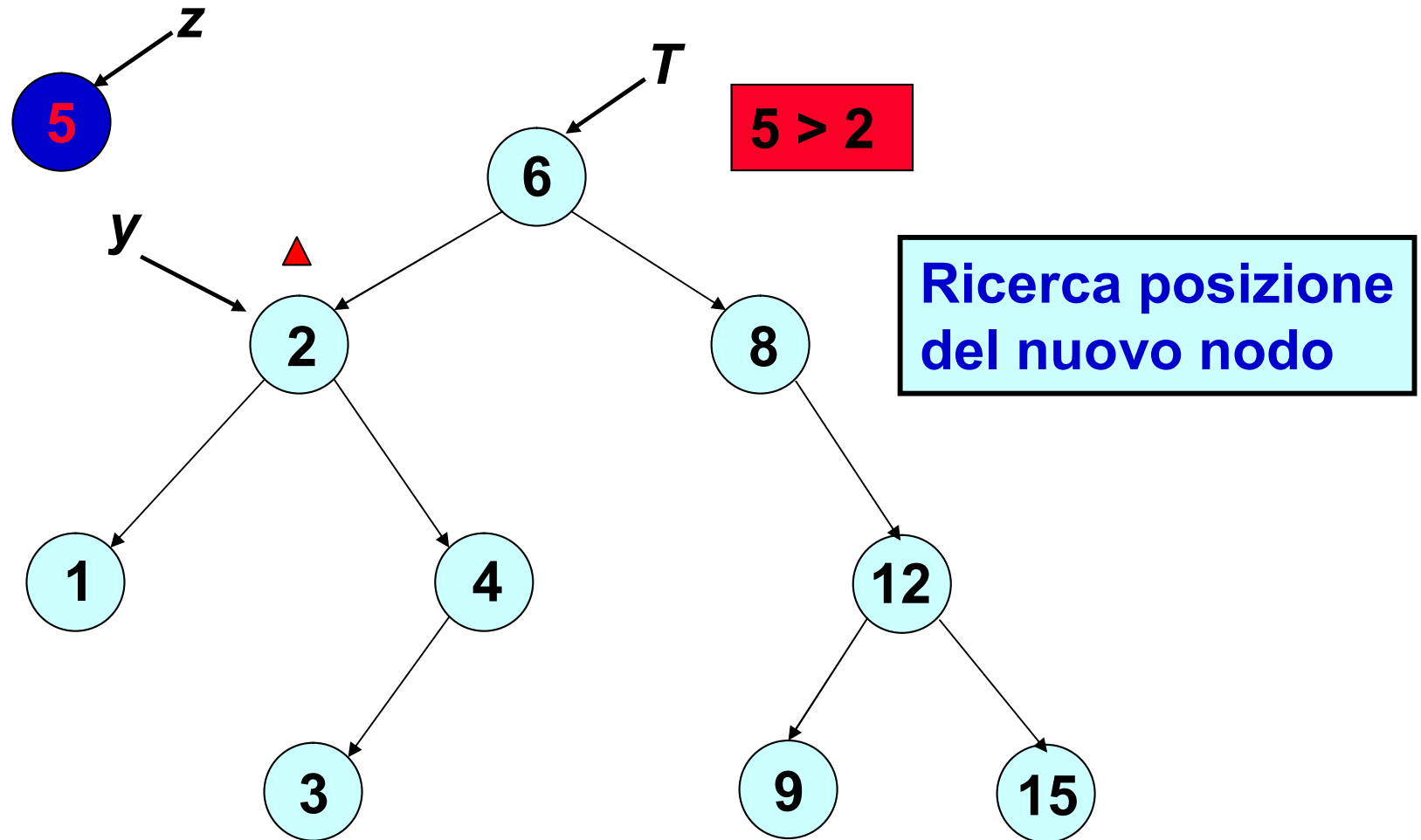
## *ARB: Inserimento di un nodo (caso I)*



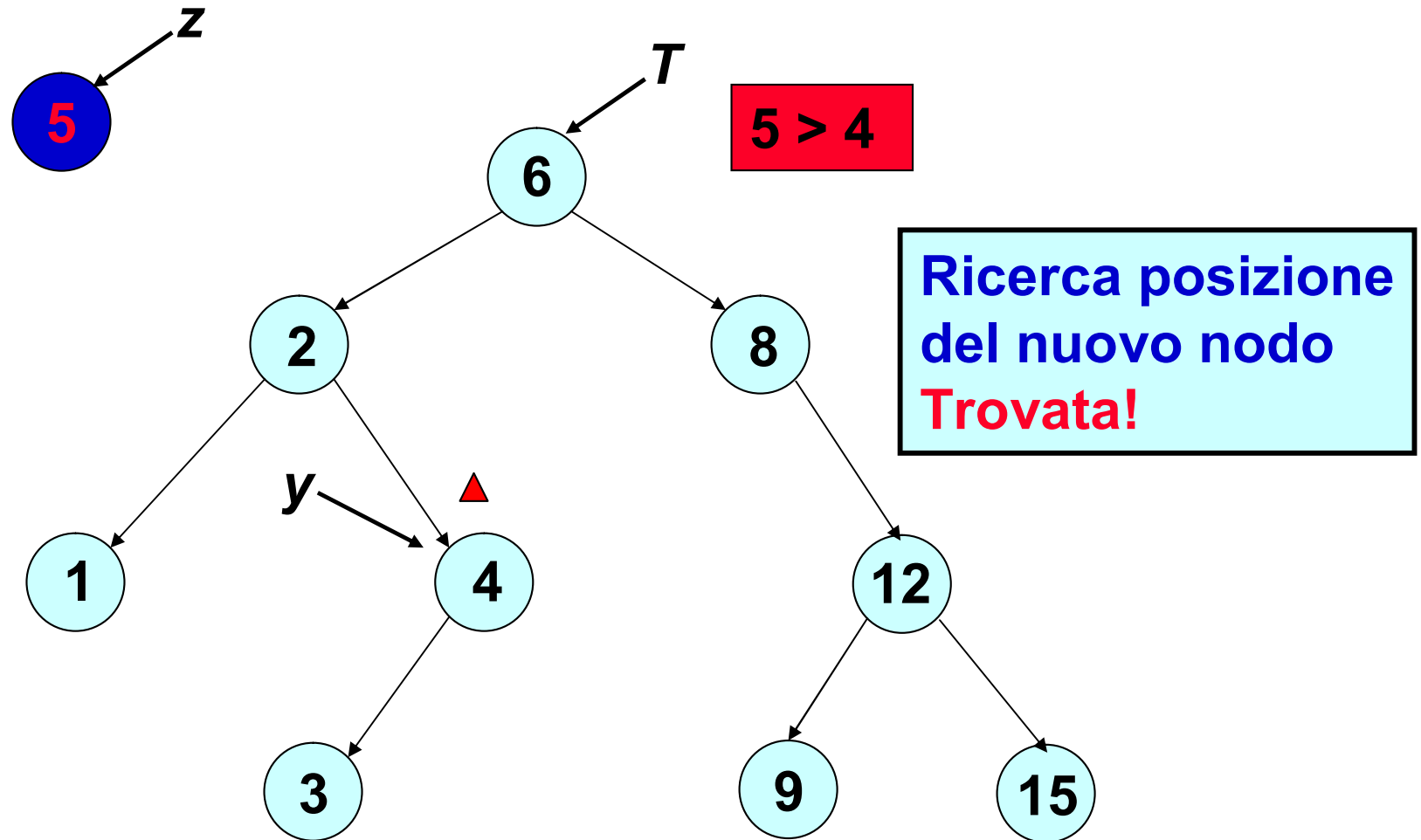
## ARB: Inserimento di un nodo (caso I)



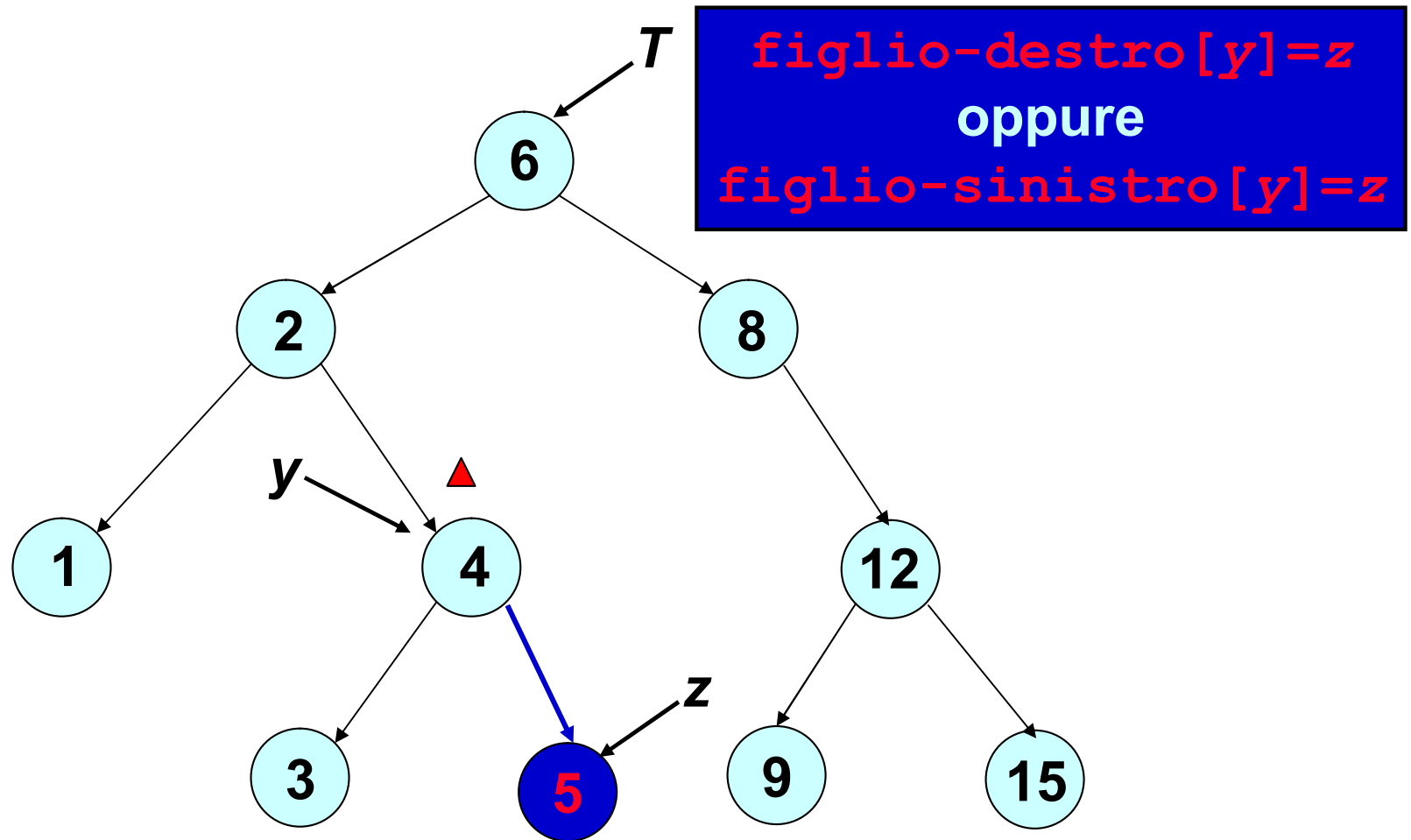
## ARB: Inserimento di un nodo (caso I)



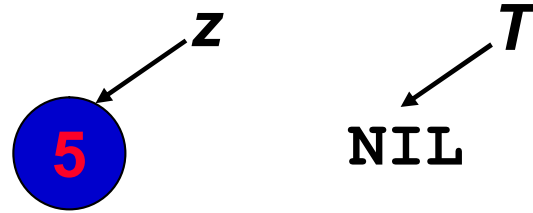
# ARB: Inserimento di un nodo (caso I)



# ARB: Inserimento di un nodo (caso I)

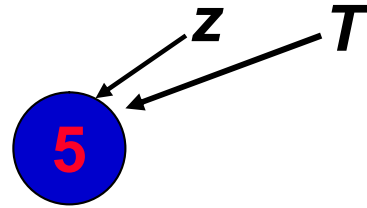


## *ARB: Inserimento di un nodo (caso II)*



**Albero è vuoto**

## ARB: Inserimento di un nodo (caso II)



**Root[T] = z**

**Albero è vuoto  
Il nuovo nodo da  
inserire diviene  
la radice**

## *ARB: Inserimento di un nodo*

```
ABR-inserisci (T, z)
  y = NIL
  x = Root[T]
  WHILE x ≠ NIL
    DO y = x
      IF key[z] < key[x]
        THEN x = figlio-sinistro[x]
        ELSE x = figlio-destro[x]
  padre[z] = y
  IF y = NIL
    THEN Root[T] = z
    ELSE IF key[z] < key[y]
      THEN figlio-sinistro[y] = z
      ELSE figlio-destro[y] = z
```

# ARB: Inserimento di un nodo

```
ABR-inserisci (T, z)
```

```
  y = NIL
```

```
  x = Root[T]
```

```
  WHILE x ≠ NIL
```

```
    DO y = x
```

```
      IF key[z] < key[x]
```

```
        THEN x = figlio-sinistro[x]
```

```
        ELSE x = figlio-destro[x]
```

```
  padre[z] = y
```

```
  IF y = NIL
```

```
    THEN Root[T] = z
```

```
    ELSE IF key[z] < key[y]
```

```
      THEN figlio-sinistro[y] = z
```

```
      ELSE figlio-destro[y] = z
```

Ricerca posizione  
del nuovo nodo

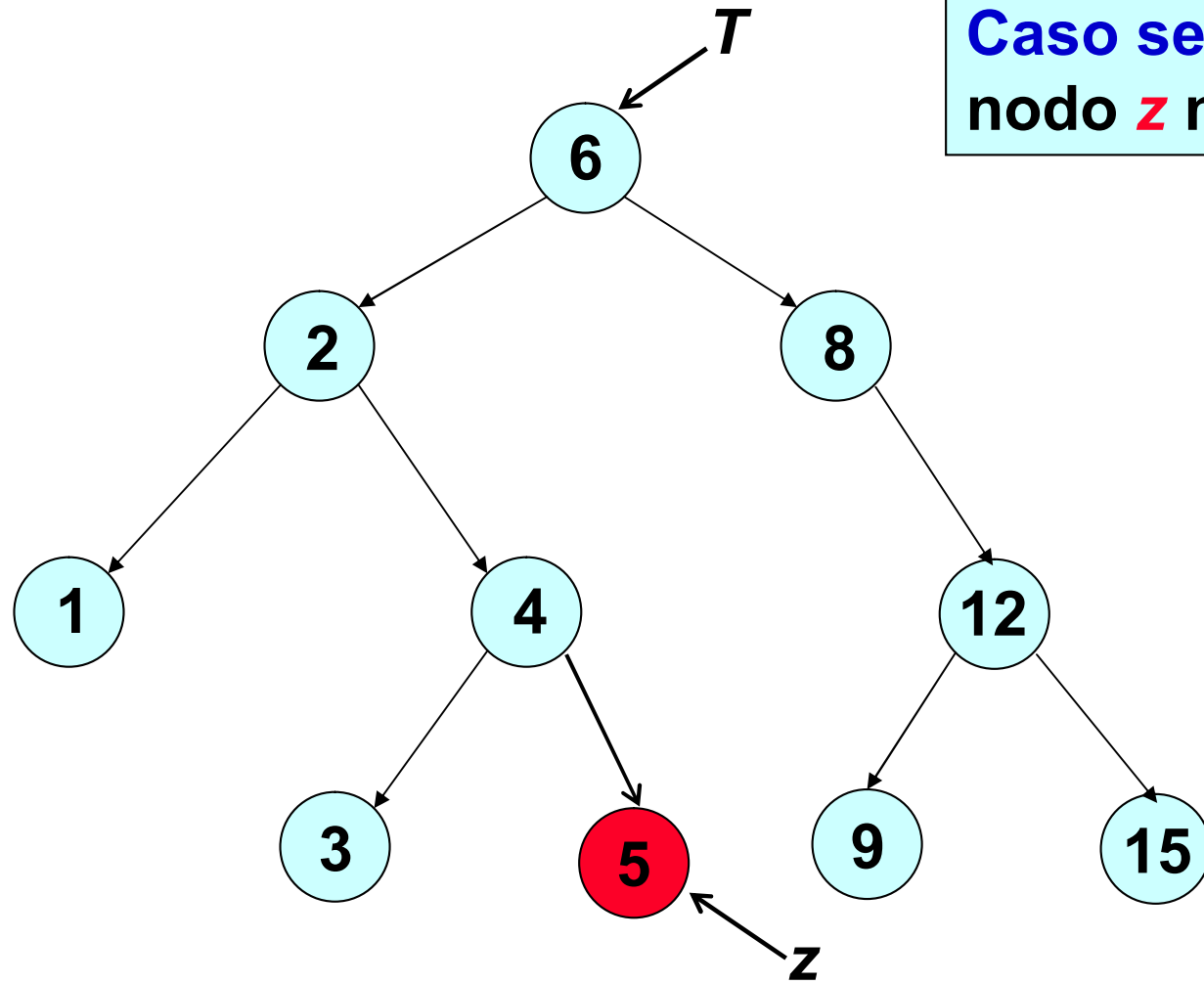
(caso II)

(caso I)

## *ARB: Inserimento di un nodo*

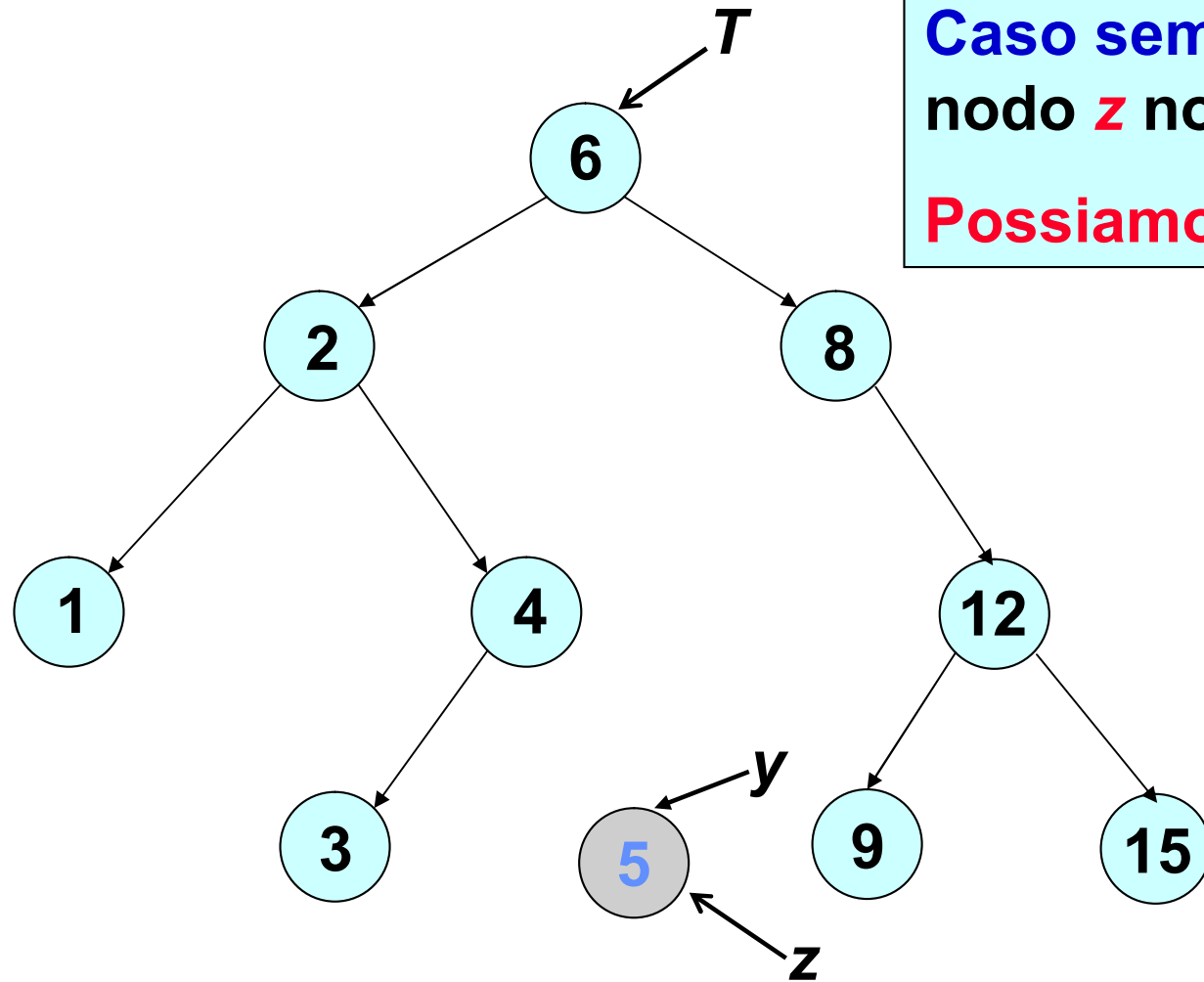
```
ABR-insert_ric(T,z)
  IF T ≠ NIL THEN
    IF key[z] < key[T] THEN
      sinistro[T]= ABR-insert_ric(sinistro[T],z)
    ELSE
      destro[T]= ABR-insert_ric(destro[T],z)
  ELSE
    T=z
  return T
```

# ARB: Cancellazione di un nodo (caso I)

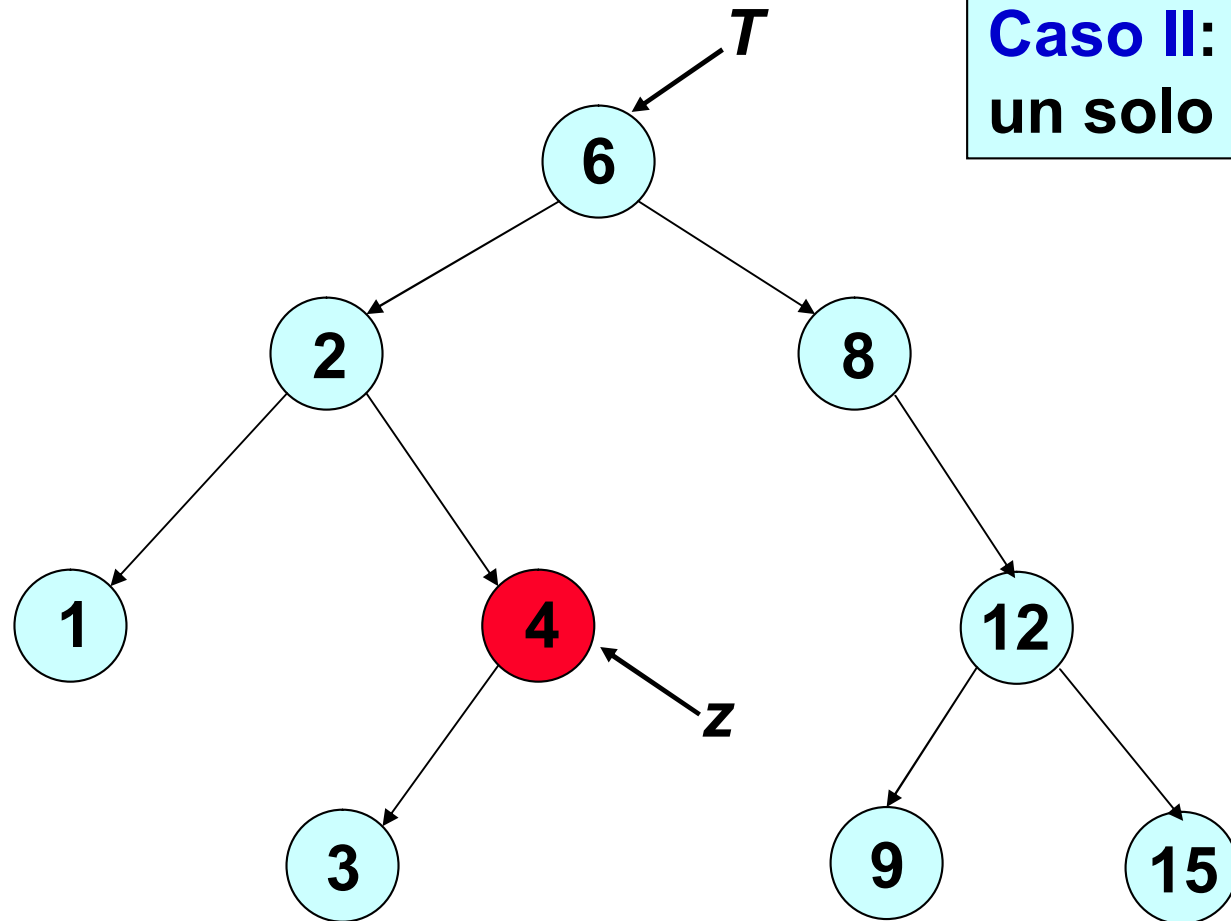


**Caso semplice: il nodo **z** non ha figli**

# ARB: Cancellazione di un nodo (caso I)

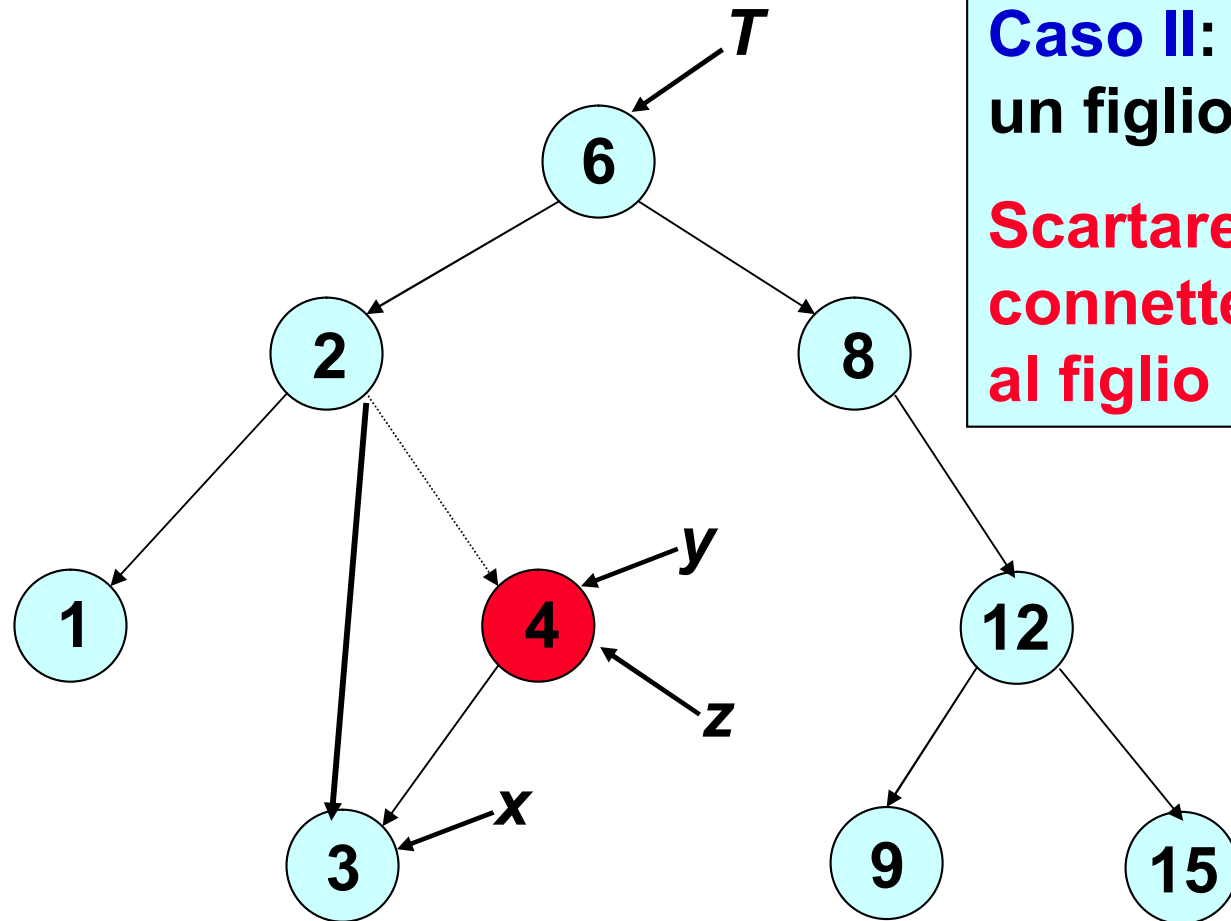


# ARB: Cancellazione di un nodo (caso II)



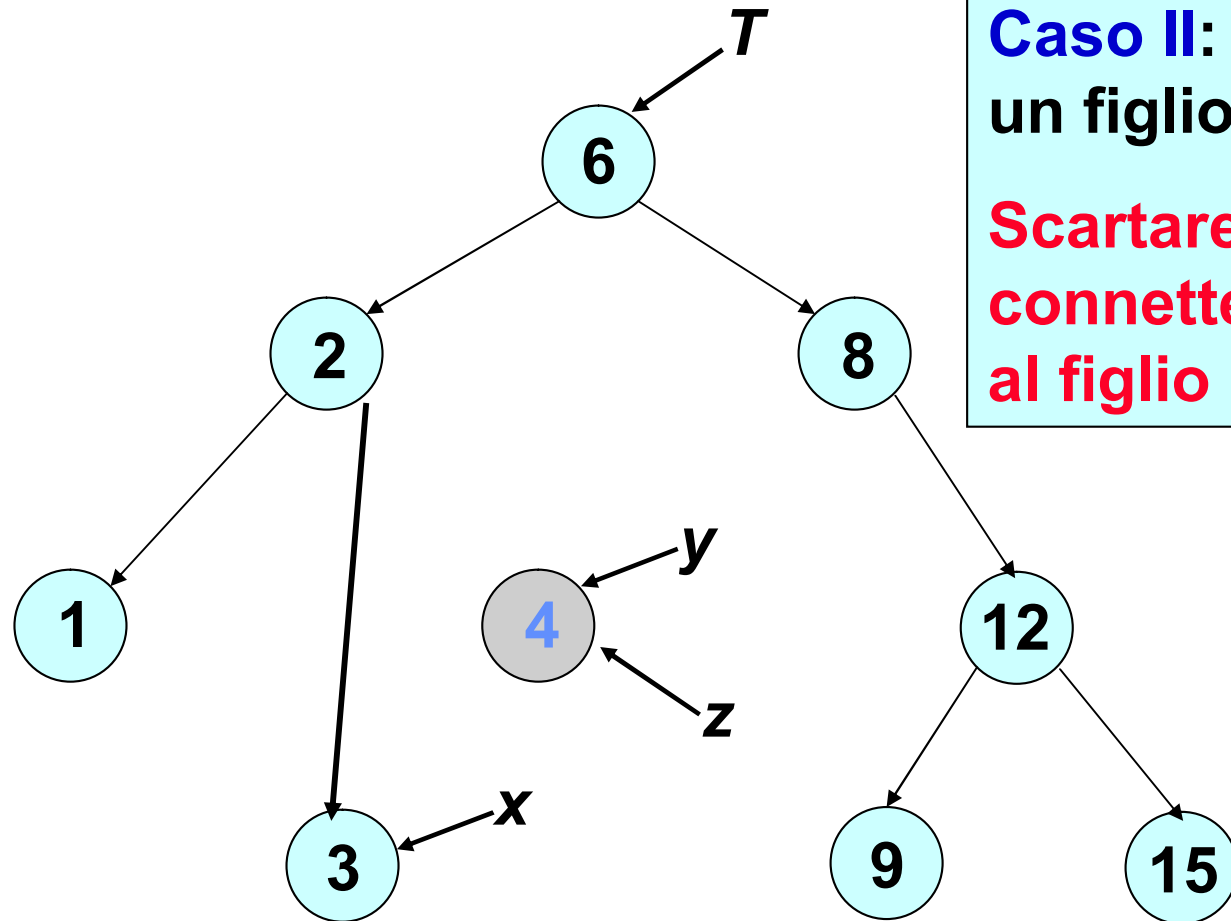
**Caso II: il nodo ha un solo figlio**

# ARB: Cancellazione di un nodo (caso II)



**Caso II: il nodo ha un figlio**  
**Scartare il nodo e connettere il padre al figlio**

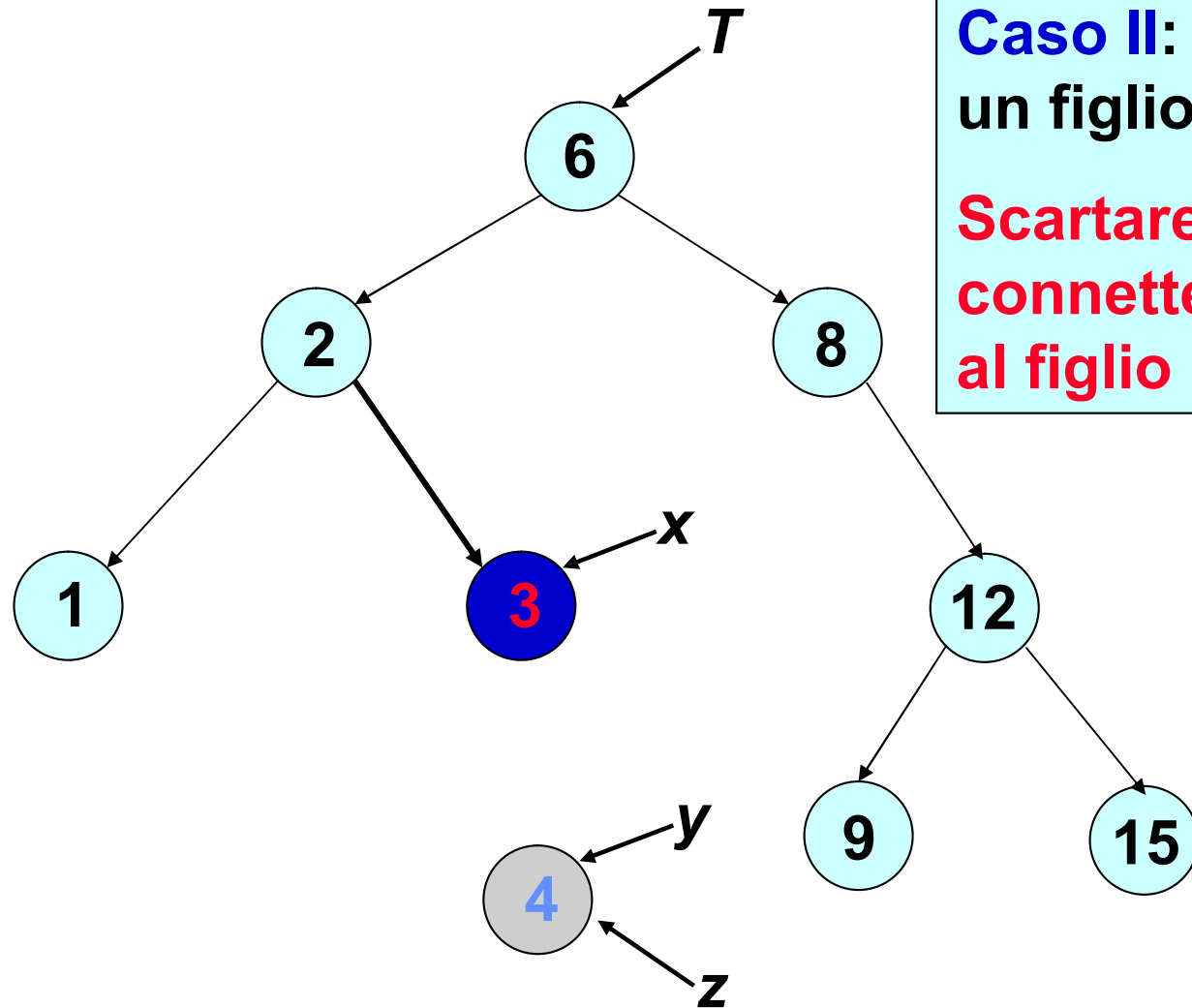
# ARB: Cancellazione di un nodo (caso II)



**Caso II: il nodo ha un figlio**

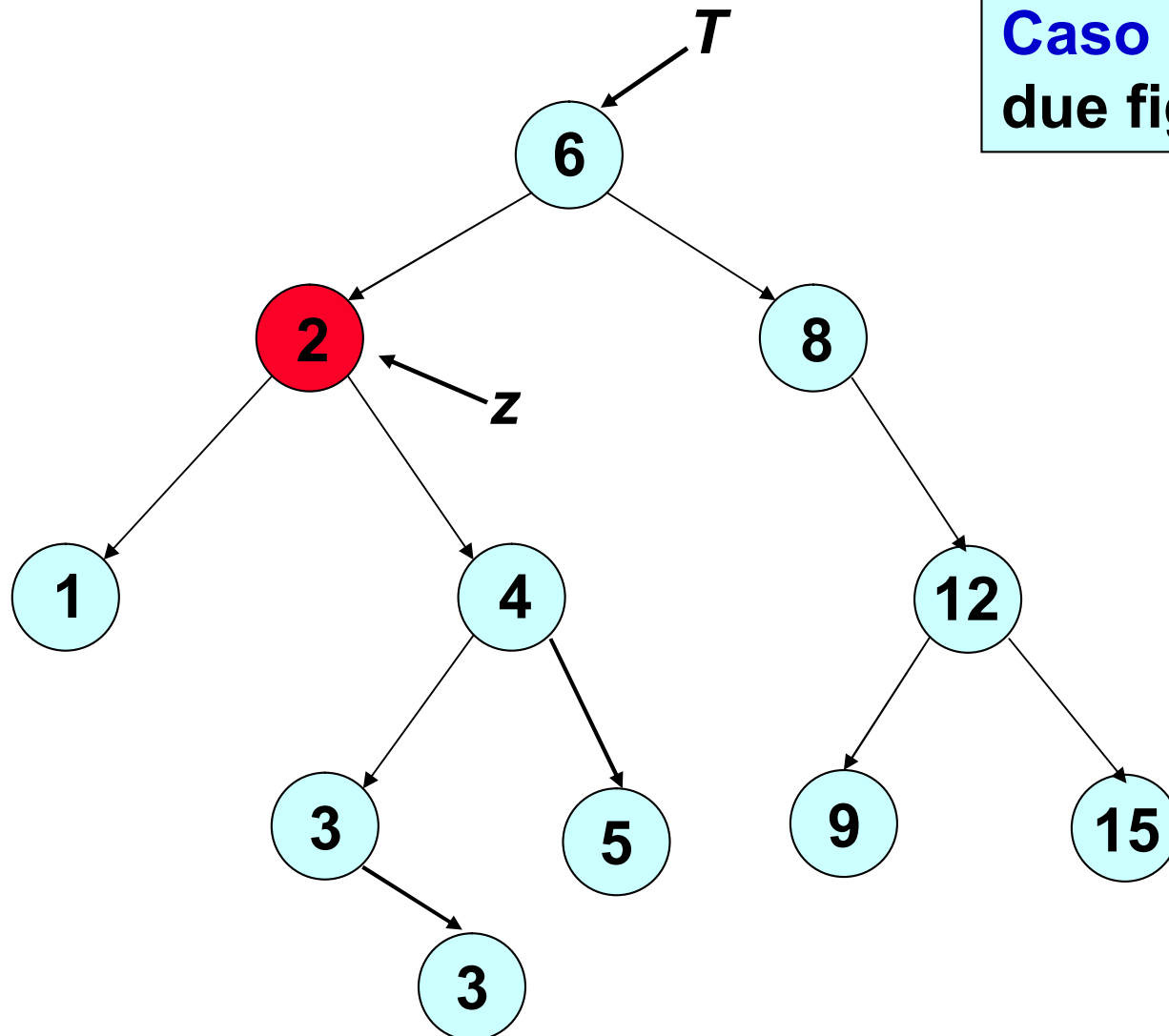
**Scartare il nodo e connettere il padre al figlio**

# ARB: Cancellazione di un nodo (caso II)



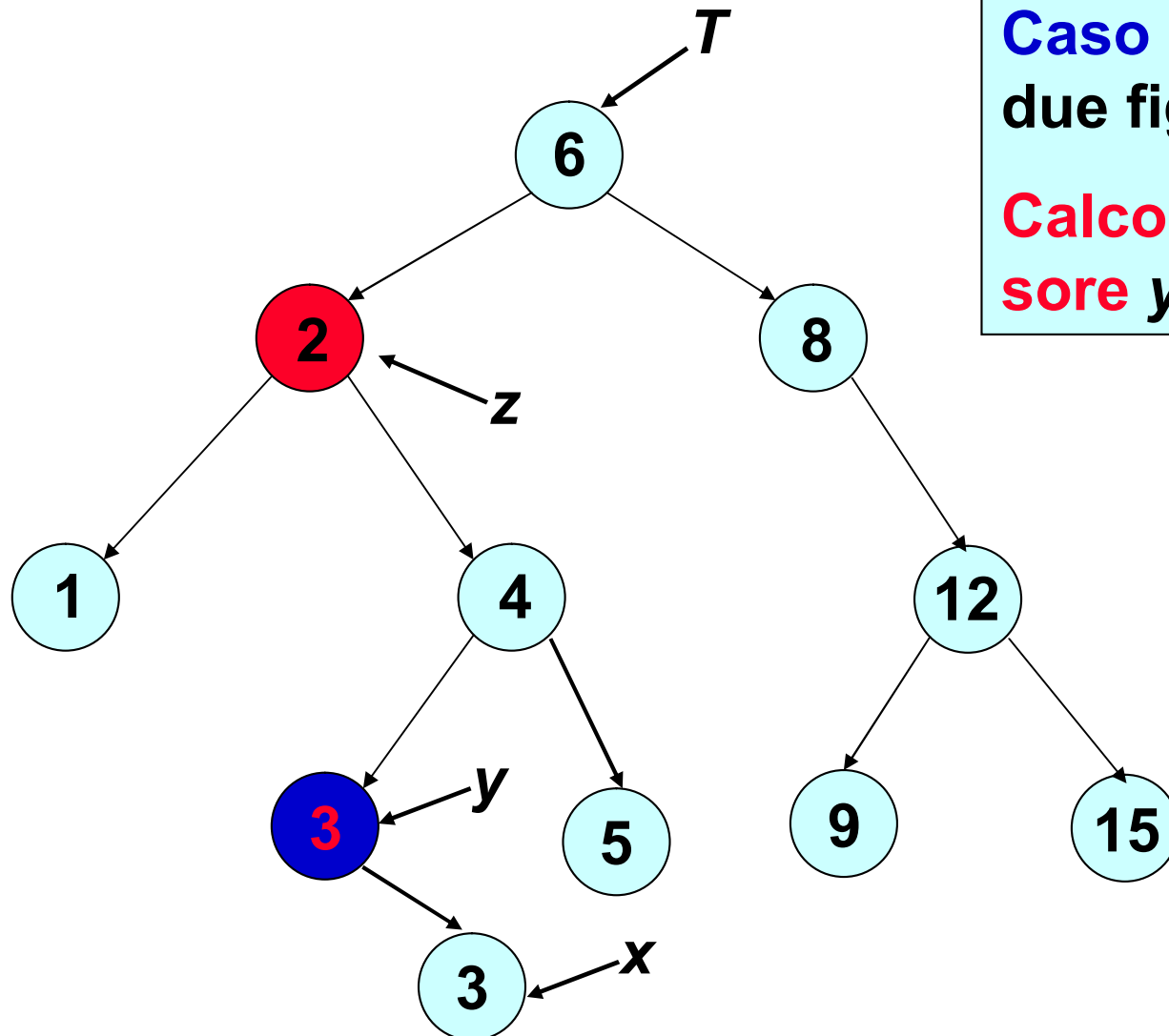
**Caso II: il nodo ha un figlio**  
**Scartare il nodo e connettere il padre al figlio**

# ARB: Cancellazione di un nodo (caso III)



**Caso III:** il nodo ha due figli

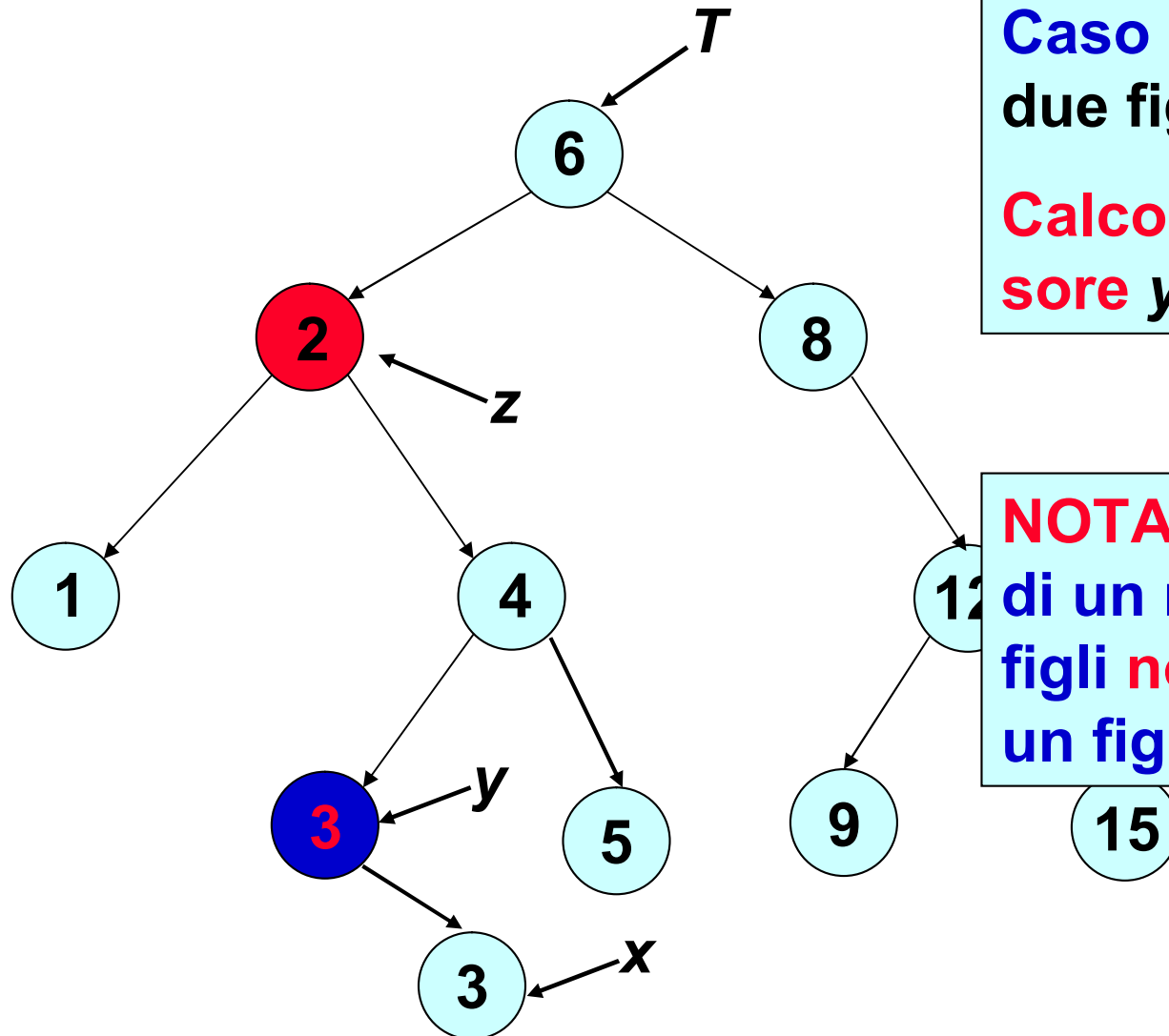
# ARB: Cancellazione di un nodo (caso III)



**Caso III:** il nodo ha due figli

**Calcolare il successore  $y$**

# ARB: Cancellazione di un nodo (caso III)

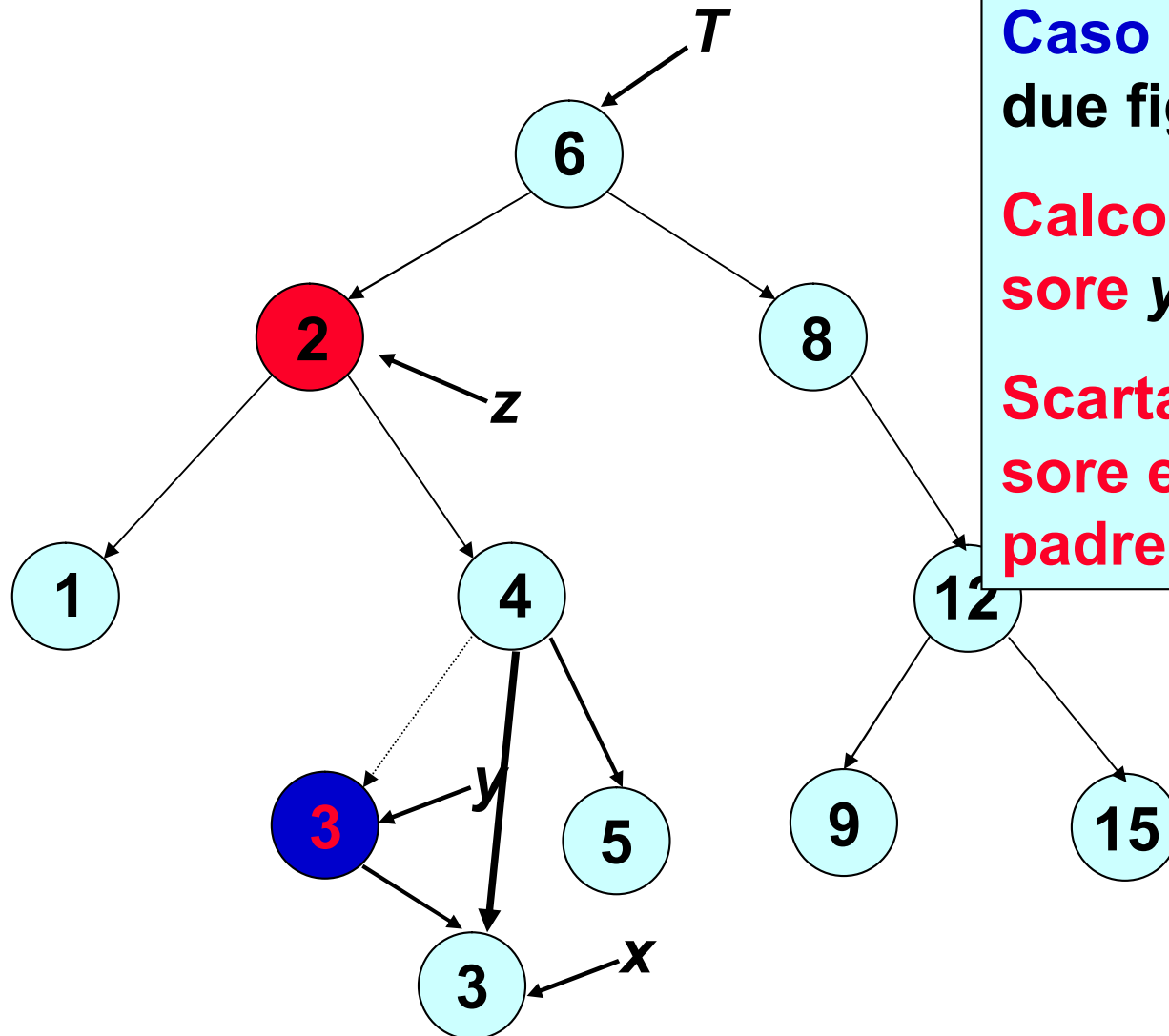


**Caso III:** il nodo ha due figli

**Calcolare il successore  $y$**

**NOTA:** Il successore di un nodo con due figli **non** può avere un figlio sinistro

# ARB: Cancellazione di un nodo (caso III)

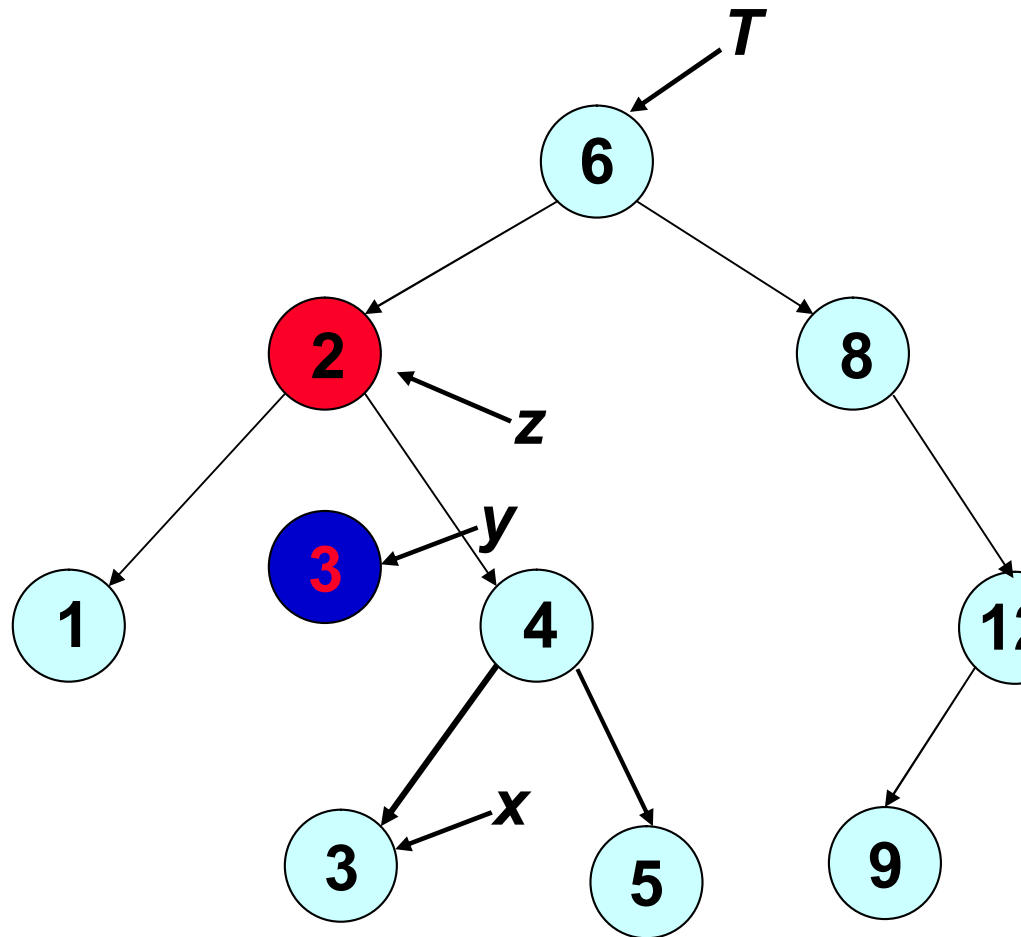


**Caso III:** il nodo ha due figli

**Calcolare il successore  $y$**

**Scartare il successore e connettere il padre al figlio destro**

# ARB: Cancellazione di un nodo (caso III)



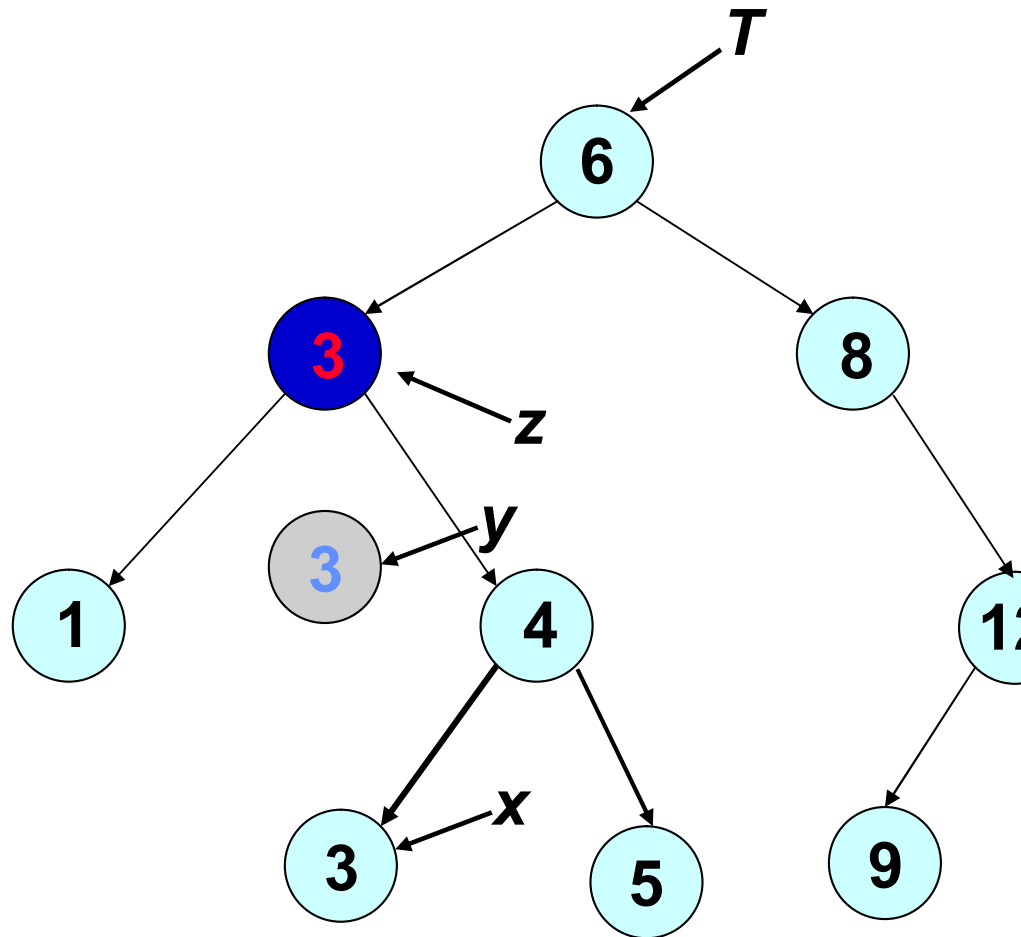
**Caso III:** il nodo ha due figli

**Calcolare il successore  $y$**

**Scartare il successore e connettere il padre al figlio destro**

**Copia il contenuto del successore nel nodo da cancellare**

# ARB: Cancellazione di un nodo (caso III)



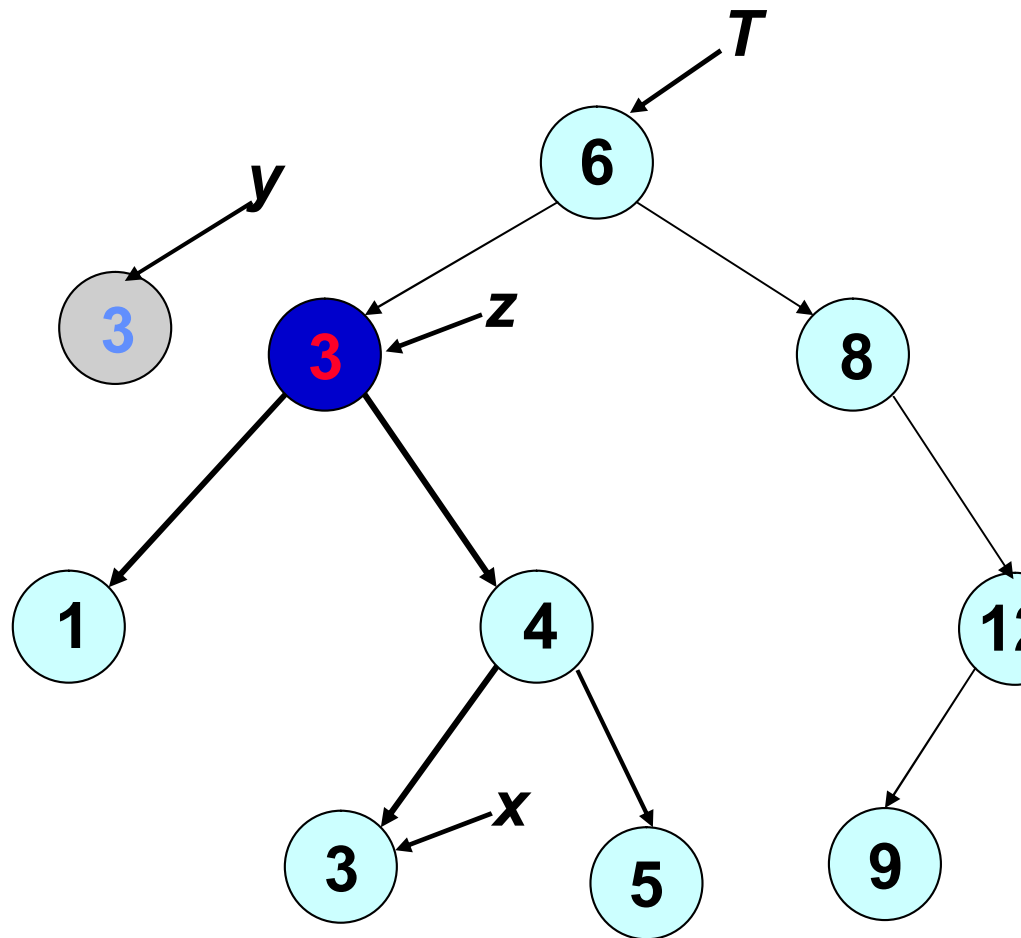
**Caso III:** il nodo ha due figli

**Calcolare il successore  $y$**

**Scartare il successore e connettere il padre al figlio destro**

**Copia il contenuto del successore nel nodo da cancellare**

## ARB: Cancellazione di un nodo (caso III)



**Caso III:** il nodo ha due figli

**Calcolare il successore  $y$**

**Scartare il successore e connettere il padre al figlio destro**

**Copia il contenuto del successore nel nodo da cancellare**

## ***ARB: Cancellazione di un nodo***

- ***Caso I:*** Il nodo **non ha figli**. Semplicemente si elimina.
- ***Caso II:*** Il nodo ha **un solo figlio**. Si **collega il padre del nodo al figlio** e si elimina il nodo.
- ***Caso III:*** Il nodo ha **due figli**.
  - ☆ si cerca **il suo successore** (che ha **un solo figlio destro**);
  - 🕒 si **elimina il successore** (come in ***Caso II***);
  - 🕒 si **copiano** i campi **valore** del successore **nel nodo** da eliminare.

## ARB: Cancellazione di un nodo

```
ABR-Cancella(T, z)
  IF (figlio-sinistro[z] = NIL OR
      figlio-destro[z] = NIL)
    THEN y = z
    ELSE y = ARB-Successore(z)
  IF figlio-sinistro[y] ≠ NIL
    THEN x = figlio-sinistro[y]
    ELSE x = figlio-destro[y]
  IF x ≠ NIL THEN padre[x] = padre[y]
  IF padre[y] = NIL
    THEN Root[T] = x
    ELSE IF y = figlio-sinistro[padre[y]]
          THEN figlio-sinistro[padre[y]] = x
          ELSE figlio-destro[padre[y]] = x
  IF y ≠ z THEN copia i campi di y in z
  return y
```

# ARB: Cancellazione di un nodo

```
ABR-Cancella(T, z)
```

```
IF (figlio-sinistro[z] = NIL OR  
    figlio-destro[z] = NIL)
```

```
THEN y = z
```

```
ELSE y = ABR-Successore(z)
```

```
IF figlio-sinistro[y] ≠ NIL
```

```
THEN x = figlio-sinistro[y]
```

```
ELSE x = figlio-destro[y]
```

```
IF x ≠ NIL THEN padre[x] = padre[y]
```

```
IF padre[y] = NIL
```

```
THEN Root[T] = x
```

```
ELSE IF y = figlio-sinistro[padre[y]]
```

```
THEN figlio-sinistro[padre[y]] = x
```

```
ELSE figlio-destro[padre[y]] = x
```

```
IF y ≠ z THEN "copia i campi di y in z"
```

```
return y
```

**casi I e II**

***y* è il nodo da eliminare**

**caso III**

# ARB: Cancellazione di un nodo

ABR-Cancella( $T, z$ )

IF (figlio-sinistro[ $z$ ] = NIL OR  
figlio-destro[ $z$ ] = NIL)

THEN  $y = z$

ELSE  $y = \text{ARB-Successore}(z)$

IF figlio-sinistro[ $y$ ]  $\neq$  NIL

THEN  $x = \text{figlio-sinistro}[y]$

ELSE  $x = \text{figlio-destro}[y]$

IF  $x \neq \text{NIL}$  THEN padre[ $x$ ] = padre[ $y$ ]

IF padre[ $y$ ] = NIL

THEN Root[ $T$ ] =  $x$

ELSE IF  $y = \text{figlio-sinistro}[\text{padre}[y]]$

THEN figlio-sinistro[padre[ $y$ ]] =  $x$

ELSE figlio-destro[padre[ $y$ ]] =  $x$

IF  $y \neq z$  THEN "copia i campi di  $y$  in  $z$ "

return  $y$

**casi I e II**

$y$  è il nodo da eliminare e  $x$  è il suo sostituto

**caso III**

# ARB: Cancellazione di un nodo

ABR-Cancella( $T, z$ )

IF (figlio-sinistro[ $z$ ] = NIL OR  
figlio-destro[ $z$ ] = NIL)

THEN  $y = z$

ELSE  $y = \text{ARB-Successore}(z)$

IF figlio-sinistro[ $y$ ]  $\neq$  NIL

THEN  $x = \text{figlio-sinistro}[y]$

ELSE  $x = \text{figlio-destro}[y]$

IF  $x \neq \text{NIL}$  THEN padre[ $x$ ] = padre[ $y$ ]

IF padre[ $y$ ] = NIL

THEN Root[ $T$ ] =  $x$

ELSE IF  $y = \text{figlio-sinistro}[\text{padre}[y]]$

THEN figlio-sinistro[padre[ $y$ ]] =  $x$

ELSE figlio-destro[padre[ $y$ ]] =  $x$

IF  $y \neq z$  THEN "copia i campi di  $y$  in  $z$ "

return  $y$

**casi I e II**

$y$  è il nodo da eliminare e  $x$  è il suo sostituto

$y$  è sostituito da  $x$

**caso III**

# ARB: Cancellazione di un nodo

ABR-Cancella( $T, z$ )

IF (figlio-sinistro[ $z$ ] = NIL OR  
figlio-destro[ $z$ ] = NIL)

THEN  $y = z$

ELSE  $y = \text{ARB-Successore}(z)$

IF figlio-sinistro[ $y$ ]  $\neq$  NIL

THEN  $x = \text{figlio-sinistro}[y]$

ELSE  $x = \text{figlio-destro}[y]$

IF  $x \neq \text{NIL}$  THEN padre[ $x$ ] = padre[ $y$ ]

IF padre[ $y$ ] = NIL

THEN Root[ $T$ ] =  $x$

ELSE IF  $y = \text{figlio-sinistro}[\text{padre}[y]]$

THEN figlio-sinistro[padre[ $y$ ]] =  $x$

ELSE figlio-destro[padre[ $y$ ]] =  $x$

IF  $y \neq z$  THEN "copia i dati"

return  $y$

**casi I e II**

$y$  è il nodo da eliminare e  $x$  è il suo sostituto

$y$  è sostituito da  $x$

**caso III**

il nodo eliminato  $y$  è ritornato per deallocazione

# ARB: Cancellazione ricorsiva

```
ABR-Cancella-ric(k, T)
```

```
IF T ≠ NIL THEN
```

```
IF k < key[T] THEN
```

```
sin[T] = ABR-Cancella-ric(k, sin[T])
```

```
ELSE IF k > key[T] THEN
```

```
des[T] = ABR-Cancella-ric(k, des[T])
```

```
ELSE IF (sin[T] = NIL OR des[T] = NIL) THEN
```

```
nodo = T
```

```
IF sin[nodo] ≠ NIL THEN
```

```
T = sin[nodo]
```

```
ELSE
```

```
T = des[nodo]
```

```
ELSE
```

```
nodo = ABR-Stacca-Succ(des[T], T)
```

```
"copia nodo in T"
```

```
dealloca(nodo)
```

```
return T
```

caso I e II

caso III

# ARB: Cancellazione ricorsiva

Il parametro **P** serve a ricordarsi il padre di un nodo durante la discesa

```
ABR-Stacca-Succ(T, P)
  IF T ≠ NIL THEN
    IF sin[T] ≠ NIL THEN
      return ABR-Stacca-Succ(sin[T], T)
    ELSE /* successore trovato */
      IF T = sin[P]
        sin[P] = des[T]
      ELSE /* il succ è il primo nodo passato */
        des[P] = des[T]
  return T
```

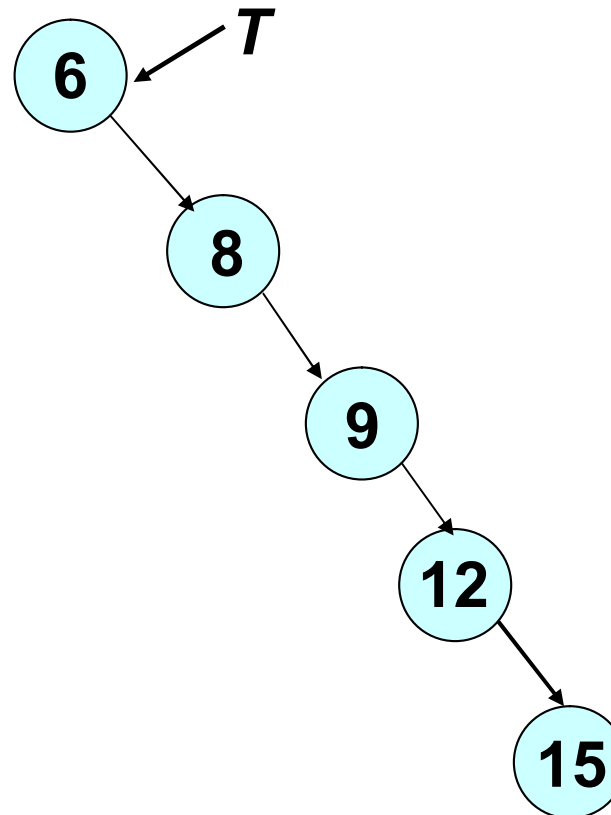
**NOTA.** Questo algoritmo **stacca il successore dell'albero T e ne ritorna il puntatore**. Può anche ritornare **NIL** in caso non esista un successore. Il valore di ritorno dovrebbe essere quindi verificato al prima dell'uso del chiamante. **Nel caso della cancellazione ricorsiva però siamo sicuri che il successore esista sempre e quindi non è necessario eseguire alcun controllo!**

## ***ARB: costo di Inserimento e Cancellazione***

***Teorema.*** ***Le operazioni di Inserimento e Cancellazione sull'insieme dinamico Albero Binario di Ricerca possono essere eseguite in tempo  $O(h)$  dove  $h$  è l'altezza dell'albero***

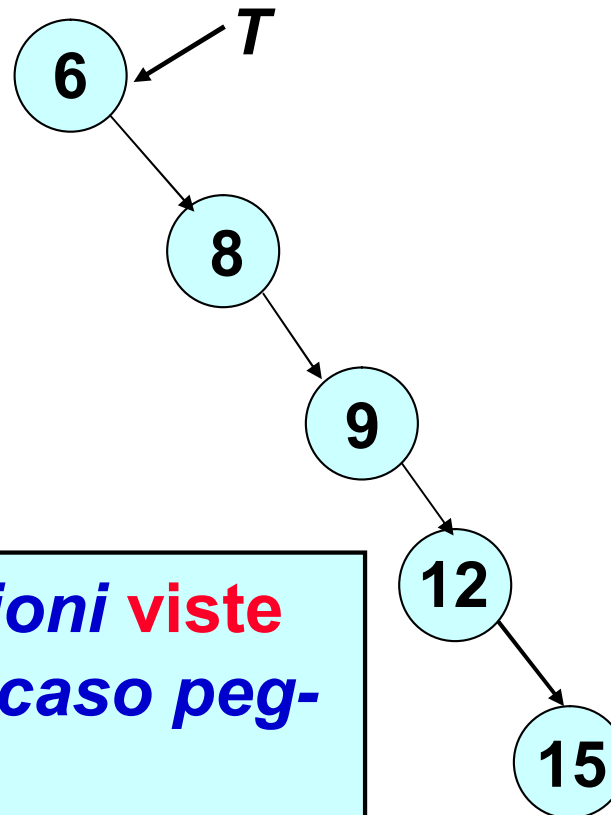
## Costo delle operazioni su ABR

**L'algoritmo di inserimento NON garantisce che l'albero risultante sia bilanciato. Nel caso peggiore l'altezza  $h$  può essere pari ad  $N$  (numero dei nodi)**



## Costo delle operazioni su ABR

**L'algoritmo di inserimento NON garantisce che l'albero risultante sia bilanciato. Nel caso peggiore l'altezza  $h$  può essere pari ad  $N$  (numero dei nodi)**



**Quindi tutte le operazioni viste hanno costo  $O(N)$  nel caso peggiore**

## Costo medio delle operazioni su ABR

Dobbiamo calcolare la **lunghezza media**  $a_n$  del **percorso di ricerca**.

- Assumiamo che le chiavi arrivino in ordine casuale (e che tutte abbiano **uguale probabilità** di presentarsi)
- La probabilità che la chiave  $i$  sia la radice è allora  $1/n$

$$a_n = \frac{1}{n} \sum_{i=1}^n p_i$$

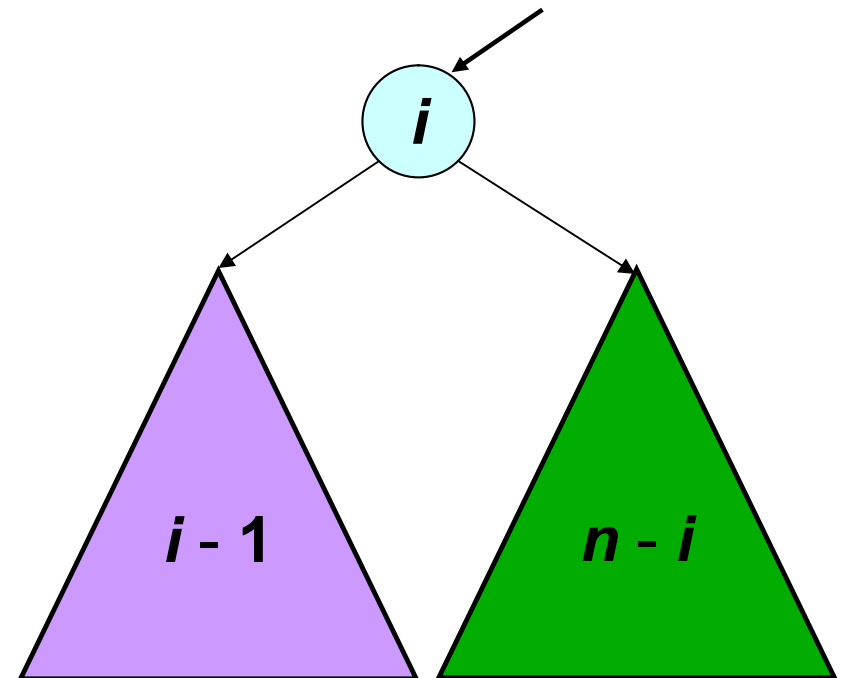
$p_i$  è la lunghezza del percorso al nodo  $i$

# Costo delle operazioni su ABR

Se  $i$  è la radice, allora

- il sottoalbero sinistro avrà  $i - 1$  nodi e
- il sottoalbero destro avrà  $n - i$  nodi

$$a_n = \frac{1}{n} \sum_{i=1}^n p_i$$

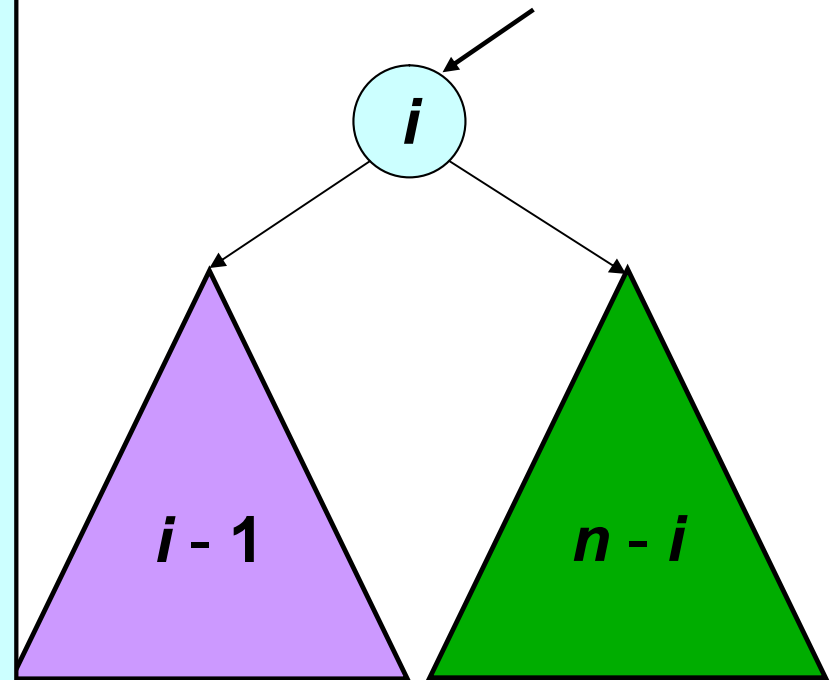


# Costo delle operazioni su ABR

Se  $i$  è la radice, allora

- il sottoalbero sinistro avrà  $i - 1$  nodi e
- il sottoalbero destro avrà  $n - i$  nodi
- ★ gli  $i - 1$  nodi a sinistra hanno lunghezza del percorso  $a_{i-1} + 1$
- 🕒 la radice ha lunghezza del percorso pari ad  $1$
- 🕒 gli  $n - i$  nodi a sinistra hanno lunghezza del percorso  $a_{n-i} + 1$

$$a_n = \frac{1}{n} \sum_{i=1}^n p_i$$



## Costo delle operazioni su ABR

$$a_n^{(i)} = (a_{i-1} + 1) \frac{i-1}{n} + 1 \frac{1}{n} + (a_{n-i} + 1) \frac{n-i}{n}$$

$a_n^{(i)}$  è la lunghezza media del percorso di ricerca con  $n$  chiavi quando la radice è la chiave  $i$

## Costo delle operazioni su ABR

$$a_n^{(i)} = (a_{i-1} + 1) \frac{i-1}{n} + 1 \frac{1}{n} + (a_{n-i} + 1) \frac{n-i}{n}$$

$a_n^{(i)}$  è la lunghezza media del percorso di ricerca con  $n$  chiavi quando la radice è la chiave  $i$

$$a_n = \frac{1}{n} \sum_{i=1}^n \left[ (a_{i-1} + 1) \frac{i-1}{n} + \frac{1}{n} + (a_{n-i} + 1) \frac{n-i}{n} \right]$$

$a_n$  è la media degli  $a_n^{(i)}$ , dove ciascun  $a_n^{(i)}$  a probabilità  $1/n$

## Costo delle operazioni su ABR

$$a_n = \frac{1}{n} \sum_{i=1}^n \left[ (a_{i-1} + 1) \frac{i-1}{n} + \frac{1}{n} + (a_{n-i} + 1) \frac{n-i}{n} \right]$$

$$= 1 + \frac{1}{n^2} \sum_{i=1}^n [a_{i-1}(i-1) + a_{n-i}(n-i)]$$

$$= 1 + \frac{2}{n^2} \sum_{i=1}^n a_{i-1}(i-1)$$

$$= 1 + \frac{2}{n^2} \sum_{i=1}^{n-1} i a_i$$

## Costo delle operazioni su ABR

$$a_n = 1 + \frac{2}{n^2} \sum_{i=1}^{n-1} ia_i$$

$$= 1 + \frac{2}{n^2} (n-1)a_{n-1} + \frac{2}{n^2} \sum_{i=1}^{n-2} ia_i$$

## Costo delle operazioni su ABR

$$a_n = 1 + \frac{2}{n^2} \sum_{i=1}^{n-1} ia_i$$

$$= 1 + \frac{2}{n^2} (n-1)a_{n-1} + \frac{2}{n^2} \sum_{i=1}^{n-2} ia_i$$

$$a_{n-1} = 1 + \frac{2}{(n-1)^2} \sum_{i=1}^{n-2} ia_i$$

## Costo delle operazioni su ABR

$$a_n = 1 + \frac{2}{n^2} \sum_{i=1}^{n-1} ia_i$$

$$= 1 + \frac{2}{n^2} (n-1)a_{n-1} + \frac{2}{n^2} \sum_{i=1}^{n-2} ia_i$$

$$\frac{2}{n^2} \sum_{i=1}^{n-2} ia_i = \frac{(n-1)^2}{n^2} (a_{n-1} - 1)$$

$$a_{n-1} = 1 + \frac{2}{(n-1)^2} \sum_{i=1}^{n-2} ia_i$$

## Costo delle operazioni su ABR

$$a_n = 1 + \frac{2}{n^2} (n-1)a_{n-1} + \frac{2}{n^2} \sum_{i=1}^{n-2} ia_i$$

$$\frac{2}{n^2} \sum_{i=1}^{n-2} ia_i = \frac{(n-1)^2}{n^2} (a_{n-1} - 1)$$

$$a_n = \frac{1}{n^2} ((n^2 - 1)a_{n-1} + 2n - 1)$$

## Costo delle operazioni su ABR

$$a_n = \frac{1}{n^2} ((n^2 - 1)a_{n-1} + 2n - 1)$$

$$a_n = 2 \frac{n+1}{n} H_n - 3$$

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

**Funzione armonica**

# Costo delle operazioni su ABR

$$a_n = \frac{1}{n^2} ((n^2 - 1)a_{n-1} + 2n - 1)$$

**Dimostrare per induzione**

$$a_n = 2 \frac{n+1}{n} H_n - 3$$

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

**Funzione armonica**

## Costo delle operazioni su ABR

$$a_n = 2 \frac{n+1}{n} H_n - 3$$

$$a_n = 2(\ln n + \gamma) - 3 = 2 \ln n - c$$

**Formula di Eulero**

$$H_n = \gamma + \ln n + \frac{1}{2n} + \frac{1}{12n^2} + \dots$$

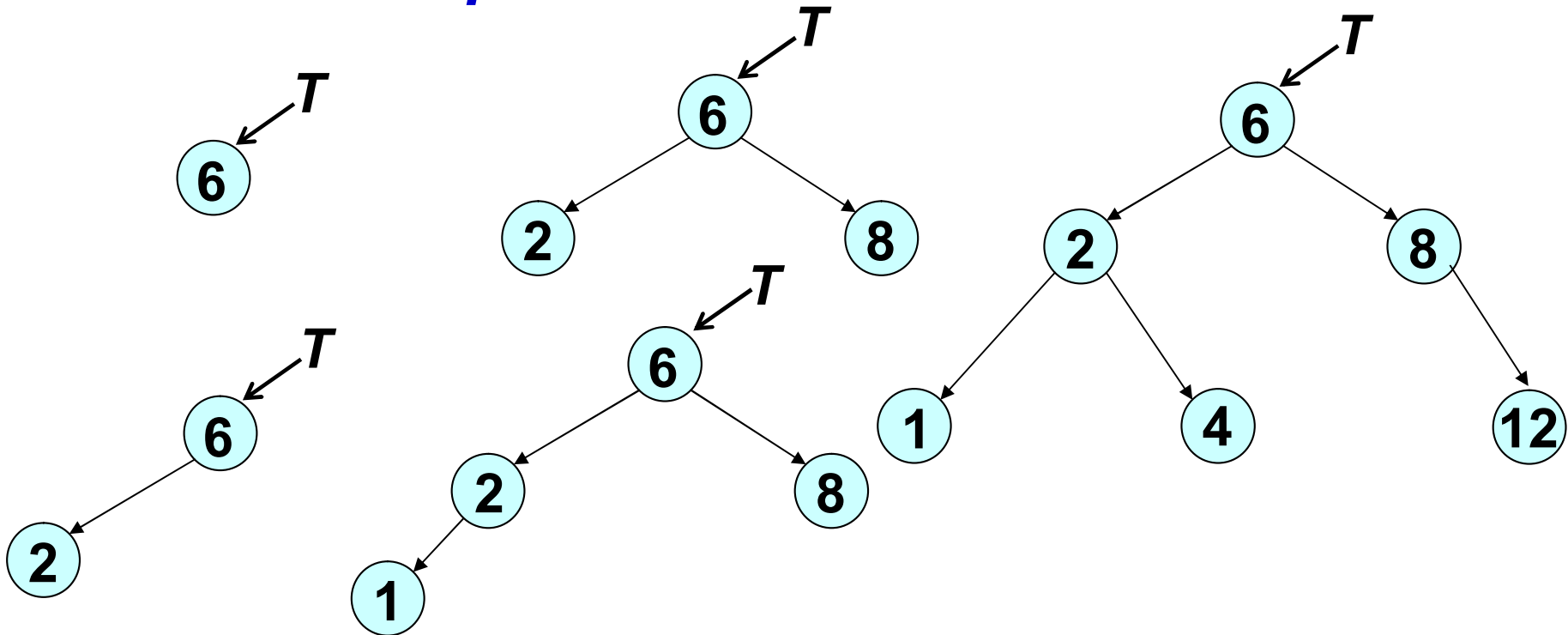
**dove  $\gamma \star 0.577$**

## *Alberi perfettamente bilanciati*

***Definizione:*** Un albero binario si dice **Perfettamente Bilanciato** se, per ogni nodo  $i$ , il **numero dei nodi** nel suo **sottoalbero sinistro** e il **numero dei nodi** del suo **sottoalbero destro** **differiscono al più di 1**

# Alberi perfettamente bilanciati

**Definizione:** Un albero binario si dice Perfettamente Bilanciato se, per ogni nodo  $i$ , il **numero dei nodi** nel suo **sottoalbero sinistro** e il **numero dei nodi** del suo **sottoalbero destro** **differiscono al più di 1**



## *Alberi perfettamente bilanciati*

**Definizione:** Un albero binario si dice **Perfettamente Bilanciato** se, per ogni nodo  $i$ , il **numero dei nodi** nel suo **sottoalbero sinistro** e il **numero dei nodi** del suo **sottoalbero destro** **differiscono al più di 1**

La **lunghezza media del percorso** in un **albero perfettamente bilanciato (APB)** è approssimativamente

$$a'_n = \log n - 1$$

## Confronto tra ABR e APB

Trascurando i termini costanti, otteniamo quindi che il **rappporto** tra la **lunghezza media del percorso** in un **albero di ricerca** e quella nell'**albero perfettamente bilanciato** è (per  $n$  grande)

$$\frac{a_n}{a'_n} = \frac{2 \ln n - c}{\log n - 1} = \frac{2 \ln n}{\log n} = 2 \ln 2 \cong 1.386$$

## Confronto tra ABR e APB

Ciò significa che, se anche **bilanciassimo** perfettamente l'albero **dopo ogni inserimento** il **guadagno sul percorso medio** che otterremmo **NON supererebbe il 39%**.

$$\frac{a_n}{a'_n} = \frac{2 \ln n - c}{\log n - 1} = \frac{2 \ln n}{\log n} = 2 \ln 2 \cong 1.386$$

**Sconsigliabile** nella maggior parte dei casi, **a meno che** il **numero dei nodi e il rapporto tra ricerche e inserimenti siano molto grandi**.