Algoritmi e Strutture Dati

Alberi Binari di Ricerca

Alberi binari di ricerca

Motivazioni

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

Esempi:

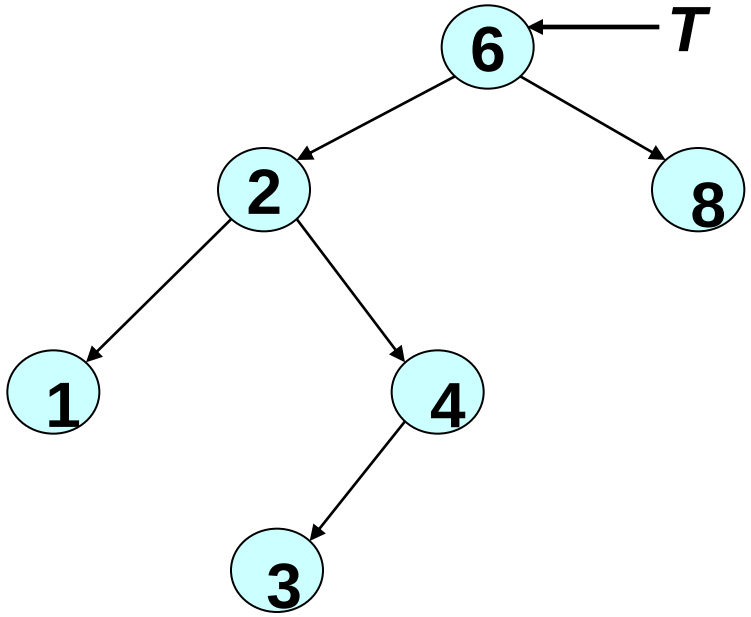
- **Mantenimento di archivi (*DataBase*)**
- **In generale, mantenimento e gestione di corpi di *dati* su cui si effettuano *molte ricerche*, eventualmente alternate a operazioni di inserimento e cancellazione.**

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 $key[Y] \leq key[X]$;

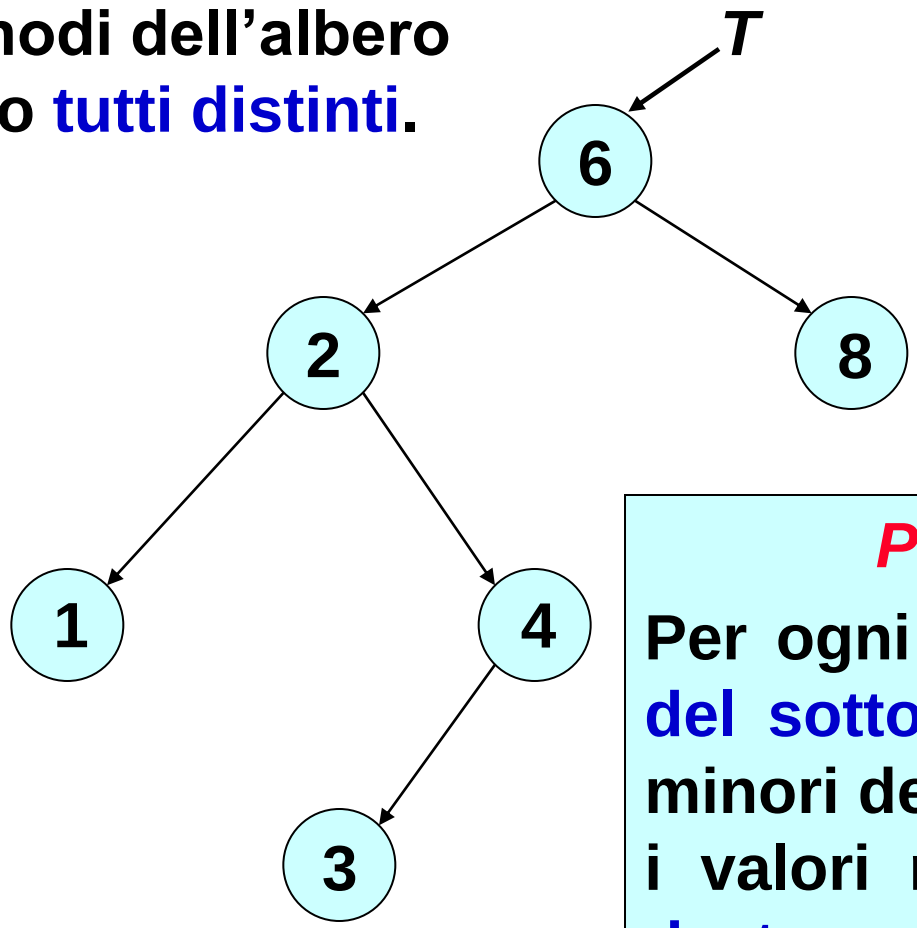
se *Y* è un nodo nel sottoablero destro di *X* allora $key[Y] \geq key[X]$



Alberi binari di ricerca

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

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

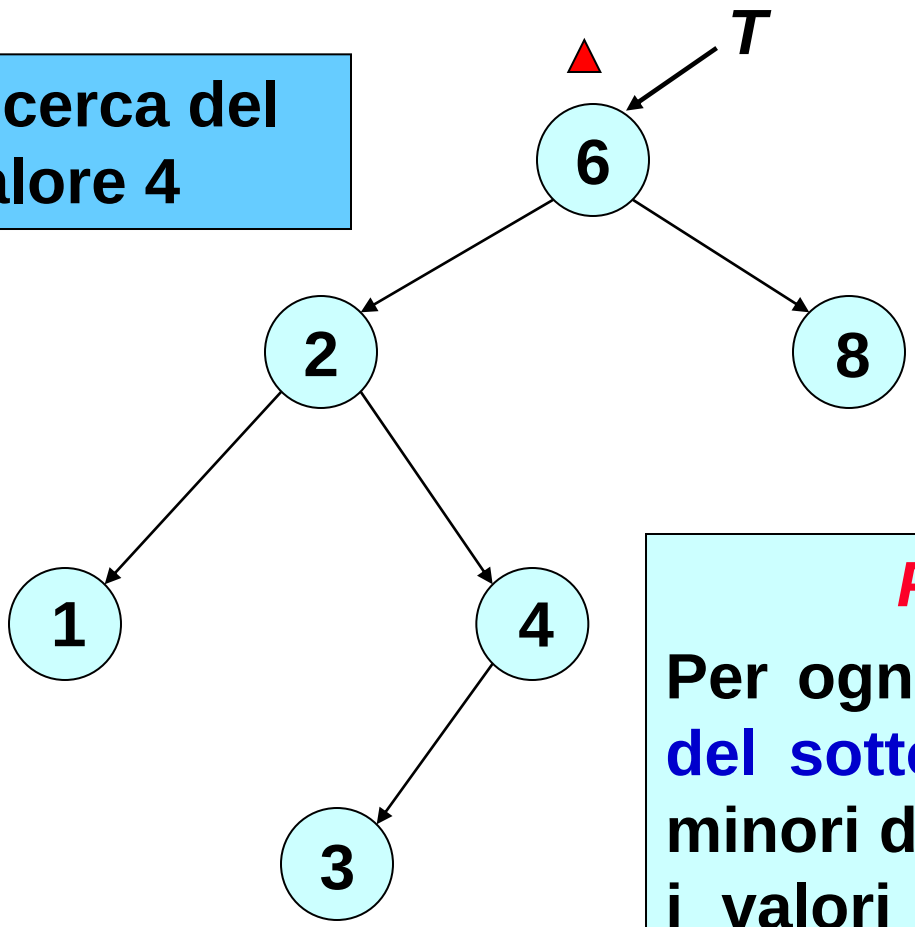


Proprietà degli *ABR*

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 sotto-albero destro sono maggiori del valore di X

Alberi binari di ricerca: esempio

Ricerca del valore 4

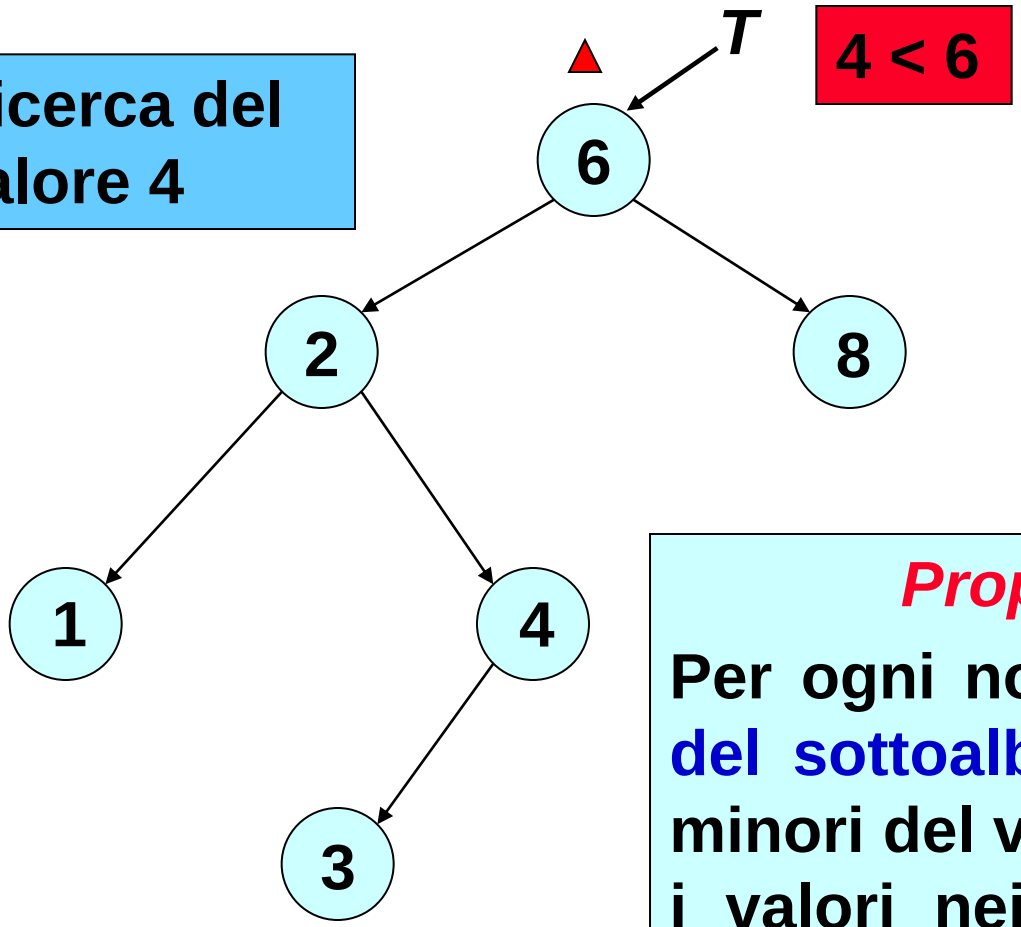


Proprietà* degli *ABR

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 sotto-albero destro sono maggiori del valore di X

Alberi binari di ricerca: esempio

Ricerca del valore 4

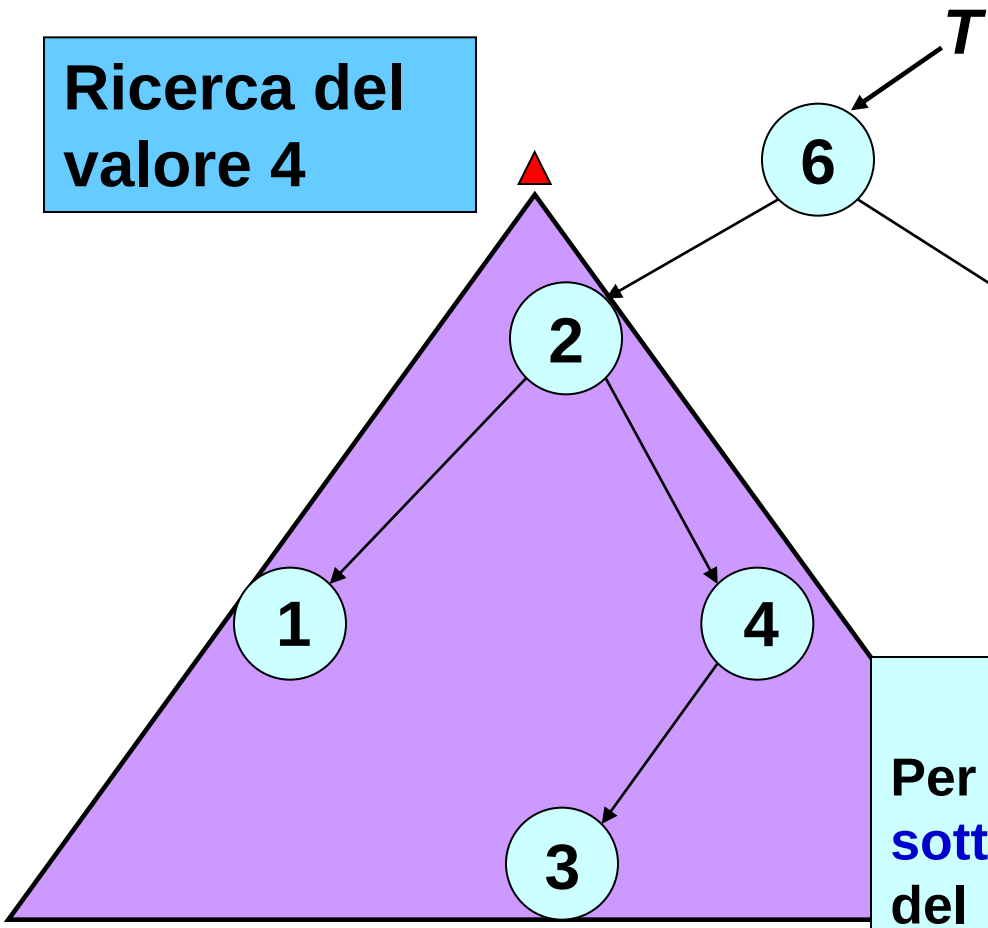


Proprietà* degli *ABR

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 sotto-albero destro sono maggiori del valore di X

Alberi binari di ricerca: esempio

Ricerca del valore 4



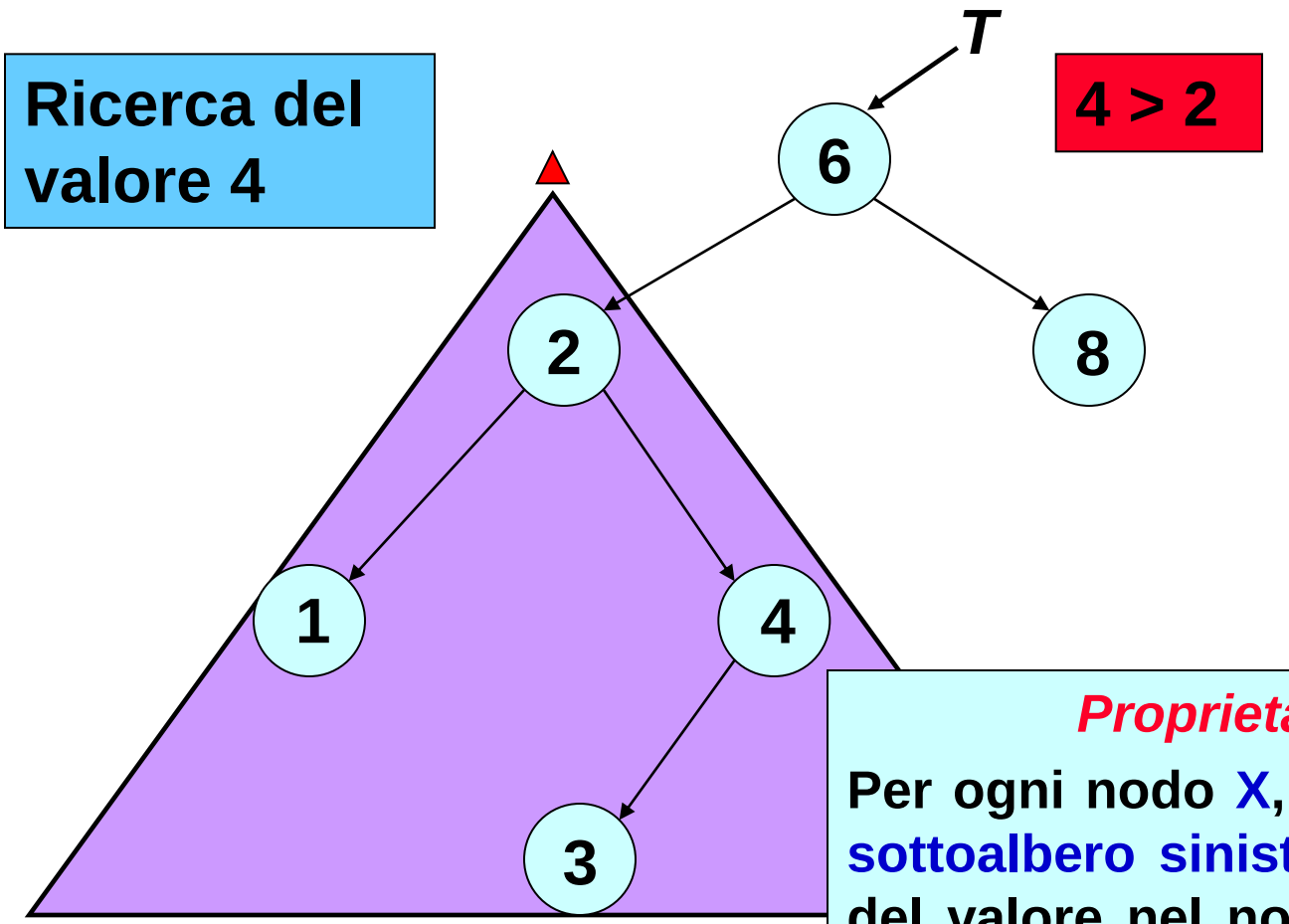
4 sta nel sottoalbero sinistro di 6

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: esempio

Ricerca del valore 4

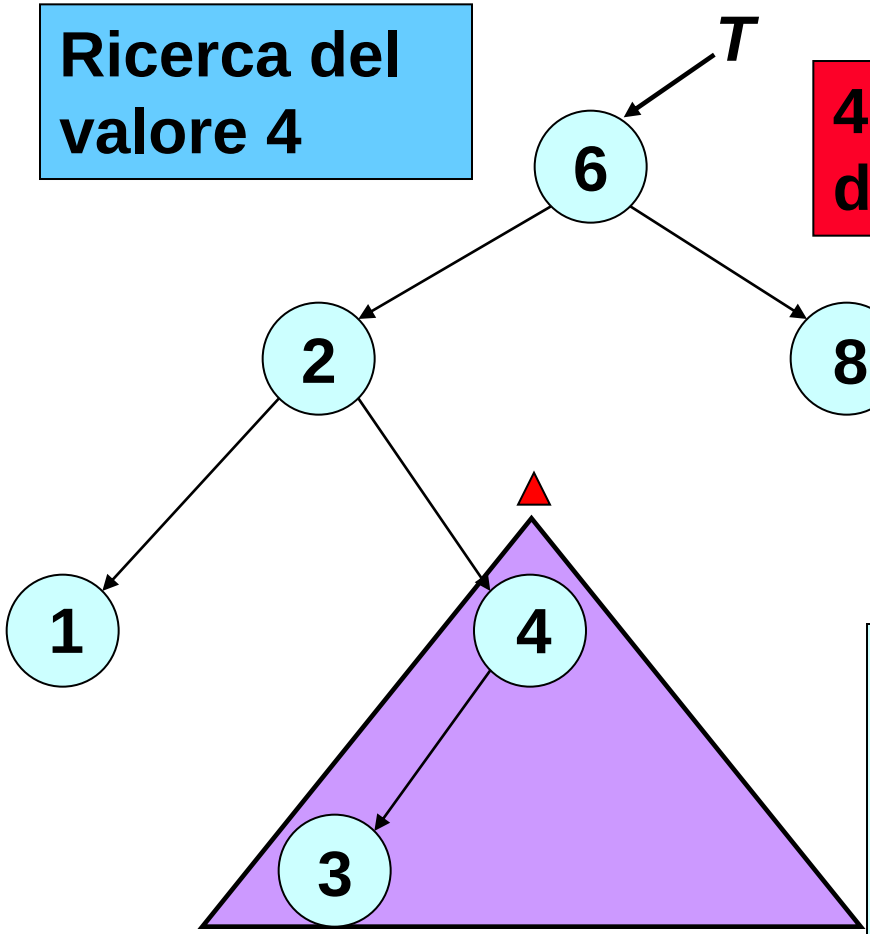


Proprietà* degli *ABR

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 sotto-albero destro sono maggiori del valore di X

Alberi binari di ricerca: esempio

Ricerca del valore 4



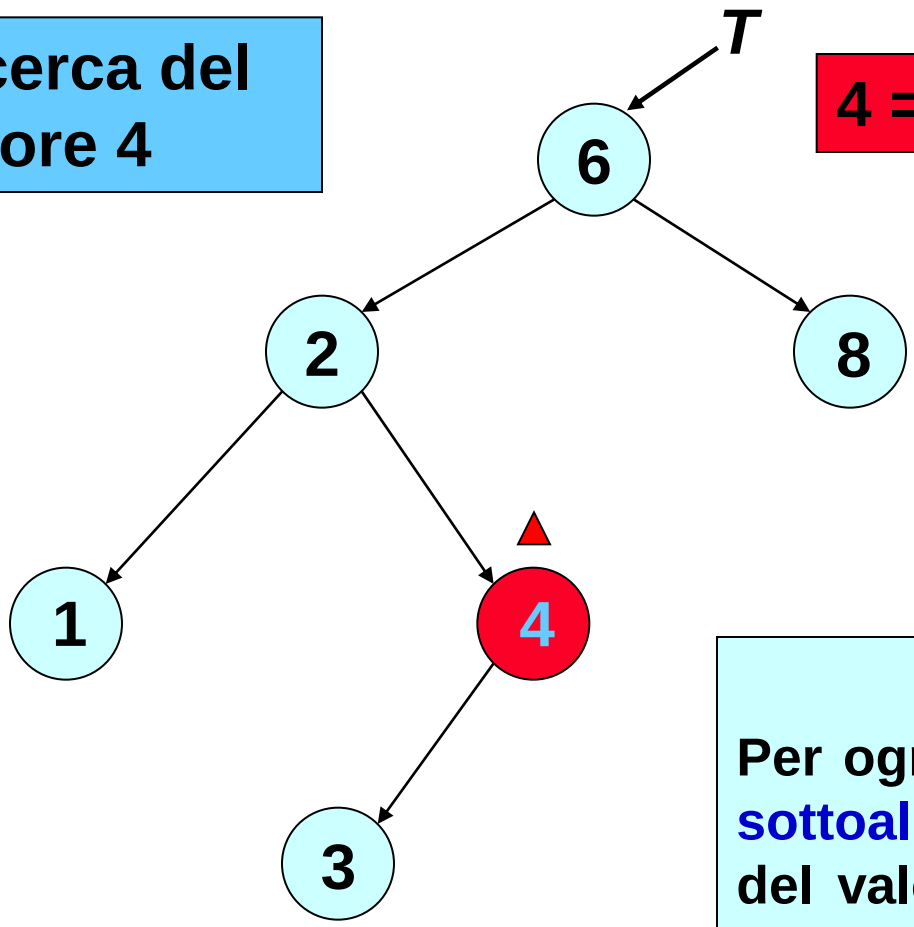
4 sta nel sottoalbero destro di 2

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: esempio

Ricerca del valore 4



4 = 4 : *Trovato*!

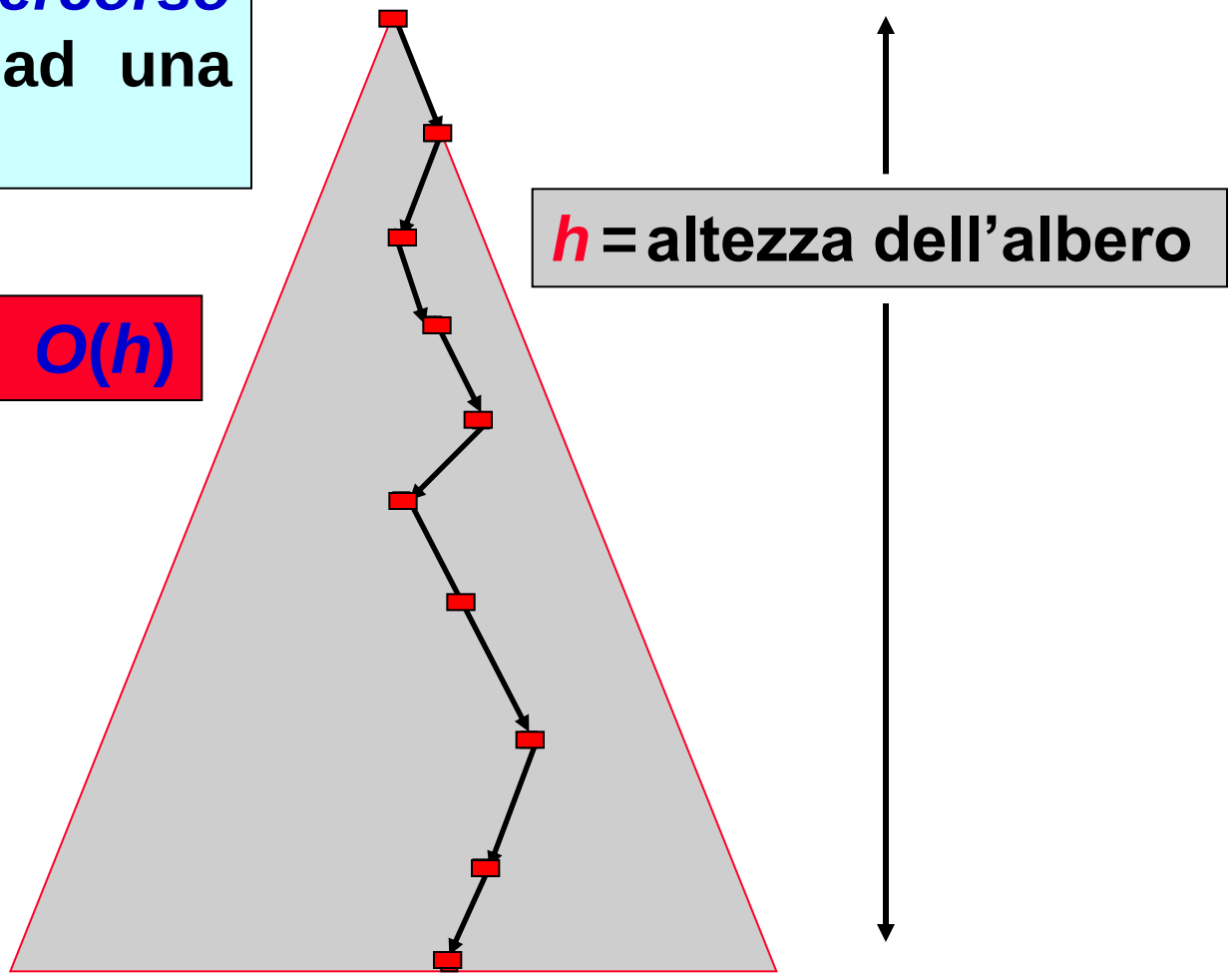
Proprietà* degli *ABR

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

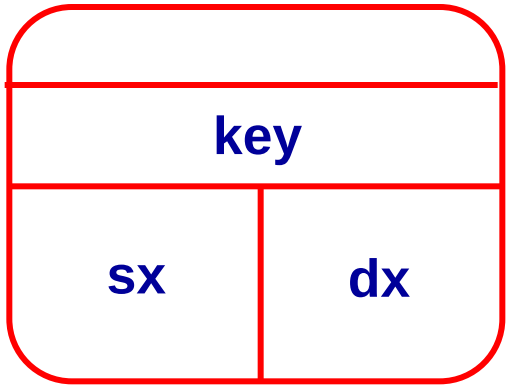
Tempo di ricerca = *O*(*h*)



ADT albero binario di ricerca: tipo di dato

- ***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 {
        elemento key;
        ARB dx, sx;
      } nodo;
```



ADT albero binario di ricerca: funzioni

- **Selettori:**
  - root(T)
  - figlio_dx(T)
  - figlio_sx(T)
  - key(T)
- **Costruttori/Distruttori:**
  - crea_albero()
  - ARB_inserisci(T,x)
  - ARB_cancella (T,x)
- **Proprietà:**
  - vuoto(T)

= return (T=Nil)

- **Operazioni di Ricerca**
  - ARB_ricerca(T,k)
  - ARB_minimo(T)
  - ARB_massimo(T)
  - ARB_successore(T,x)
  - ARB_predecessore(T,x)

Ritorna il valore del test di uguaglianza

Ricerca in Alberi binari di ricerca

```
ARB_ricerca(T,k)
    IF T ≠ NIL THEN
        IF k ≠ Key[T] THEN
            IF k < Key[T] THEN
                return ARB_ricerca(sx[T],k)
            ELSE
                return ARB_ricerca(dx[T],k)
        ELSE
            return T
    ELSE
        return T
```

NOTA: Questo algoritmo cerca il nodo con chiave k nell'albero T e ne ritorna il puntatore. Ritorna NIL nel caso non esista alcun nodo con chiave k.

Ricerca in Alberi binari di ricerca

```
ARB_ricerca'(T,k)
    IF T = NIL OR k = Key[T] THEN
          return T
    ELSE IF k < Key[T] THEN
              return ARB_ricerca'(sx[T],k)
          ELSE
              return ARB_ricerca'(dx[T],k)
```

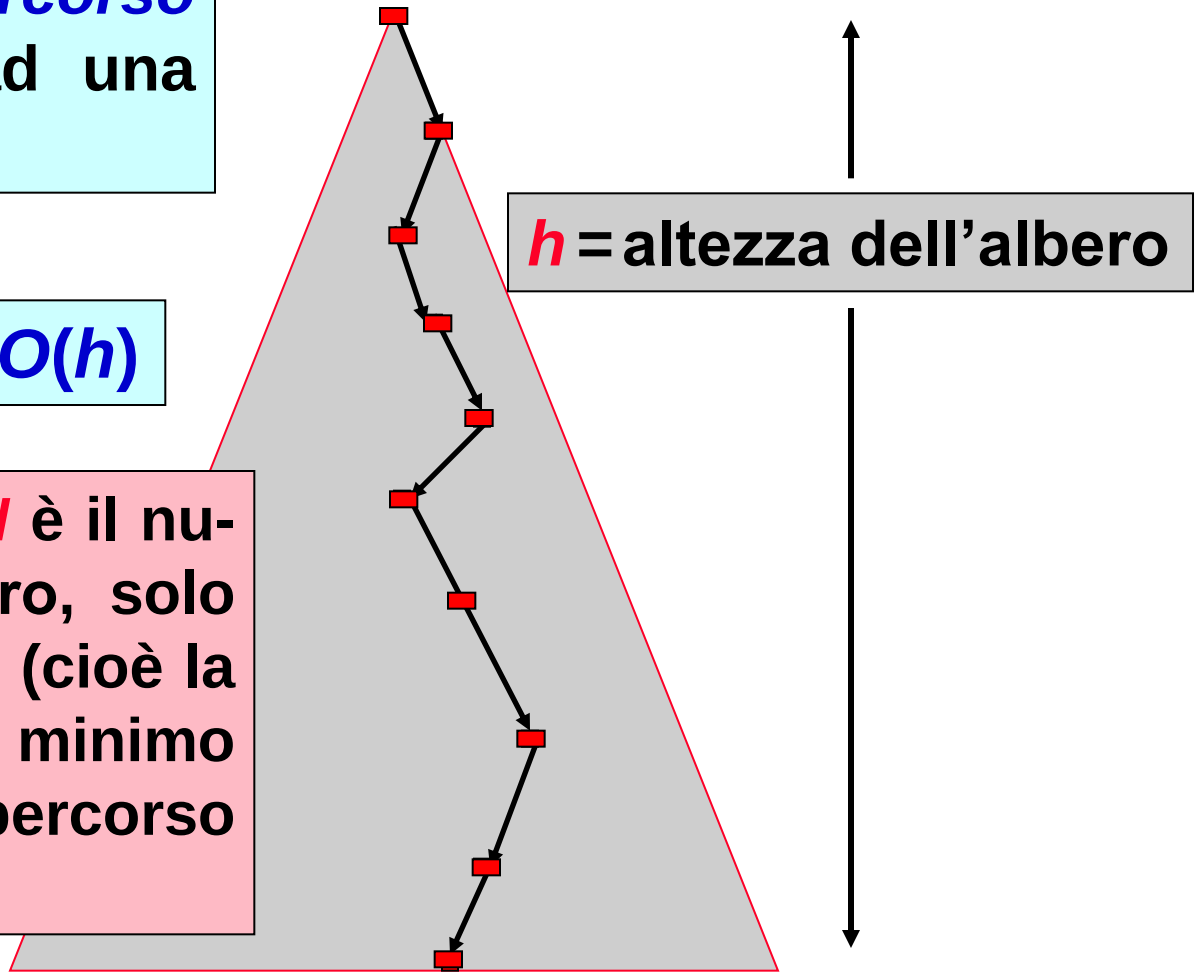
***NOTA**: Variante sintattica del precedente algoritmo!*

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*)

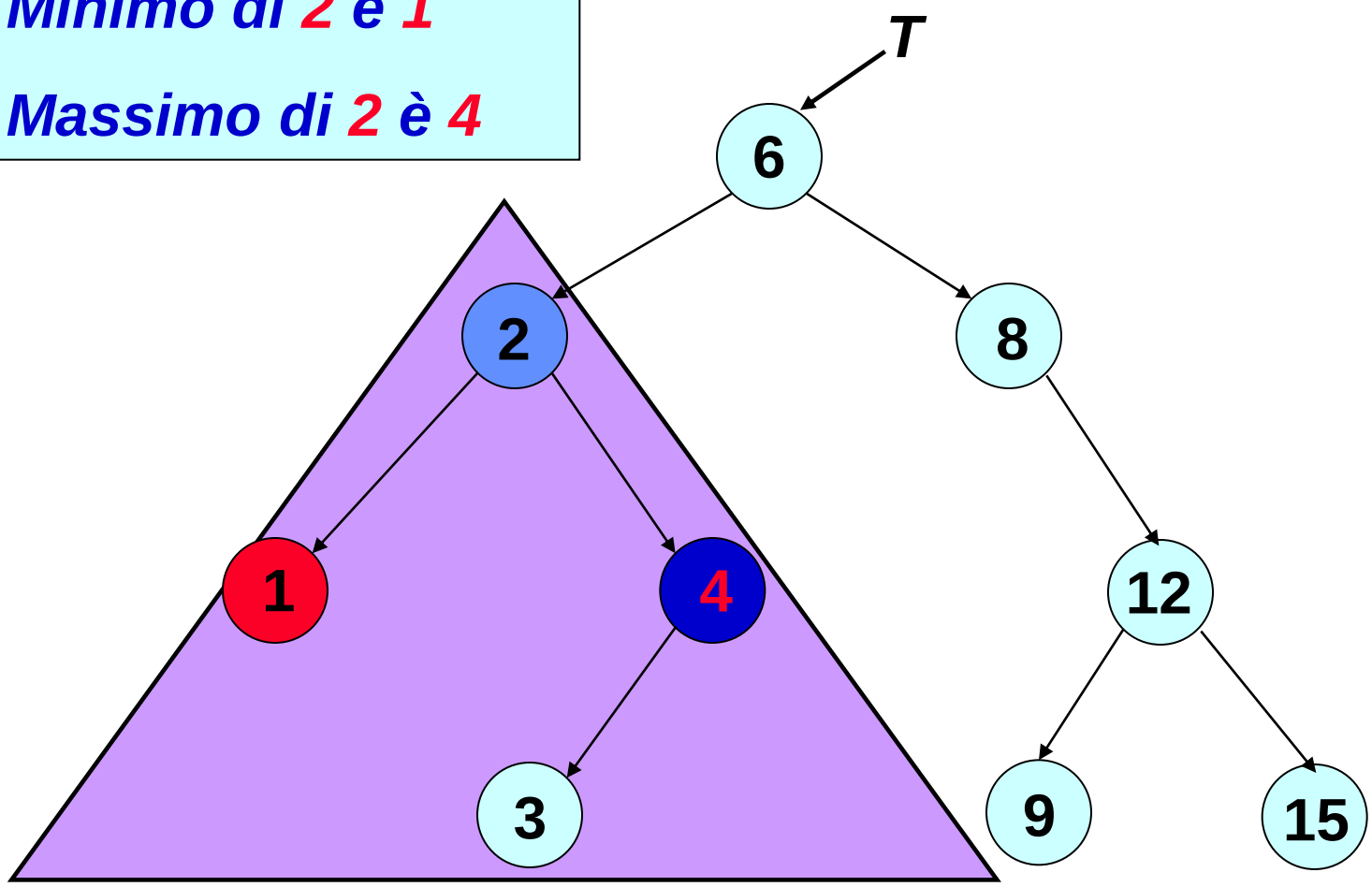
***O*(*h*) = *O*(log *N*), dove *N* è il numero di nodi nell'albero, solo se l'albero è *balanciato* (cioè la lunghezza del percorso minimo è vicino a quella del percorso massimo).**



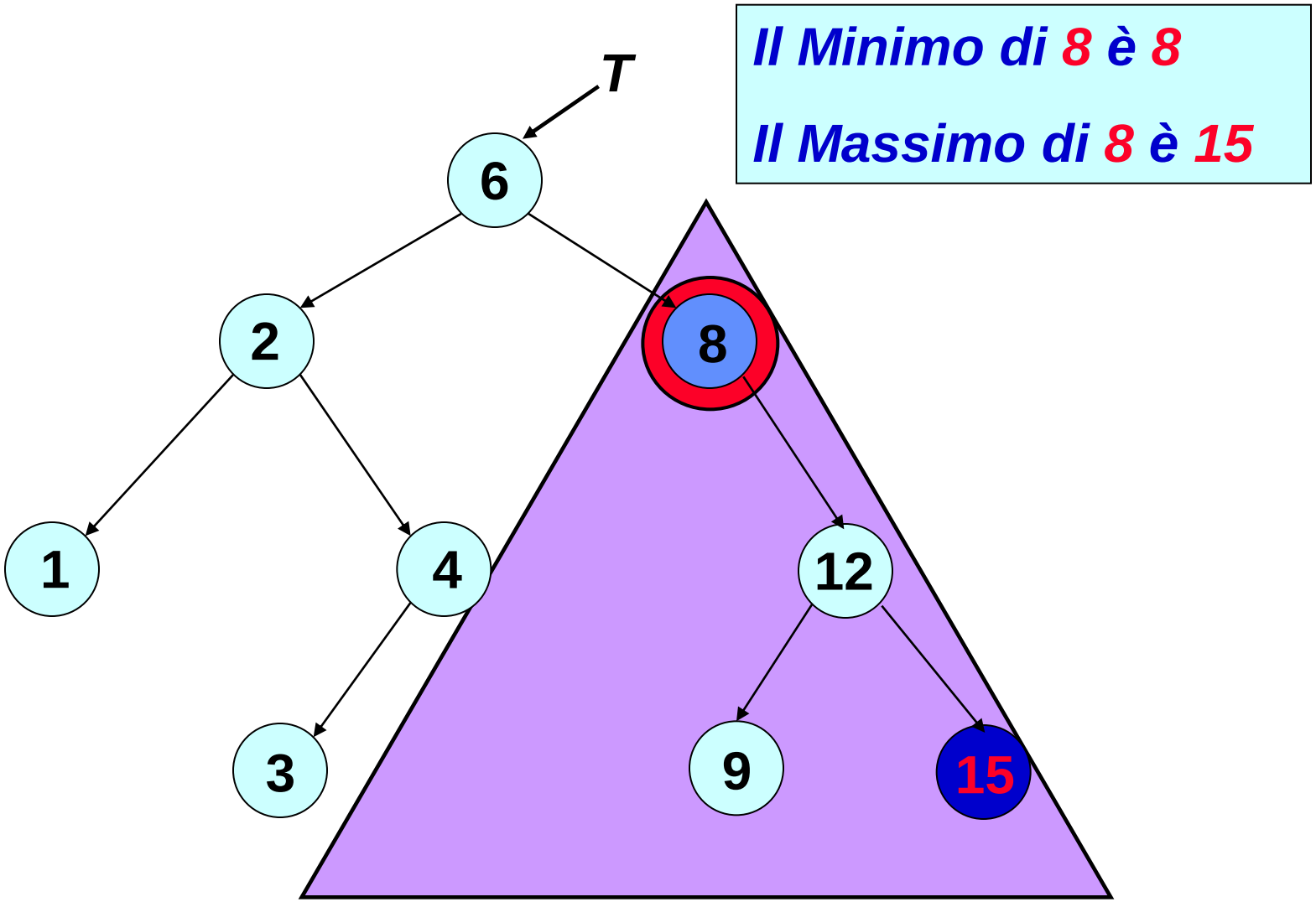
ARB: ricerca del minimo e massimo

Il Minimo di 2 è 1

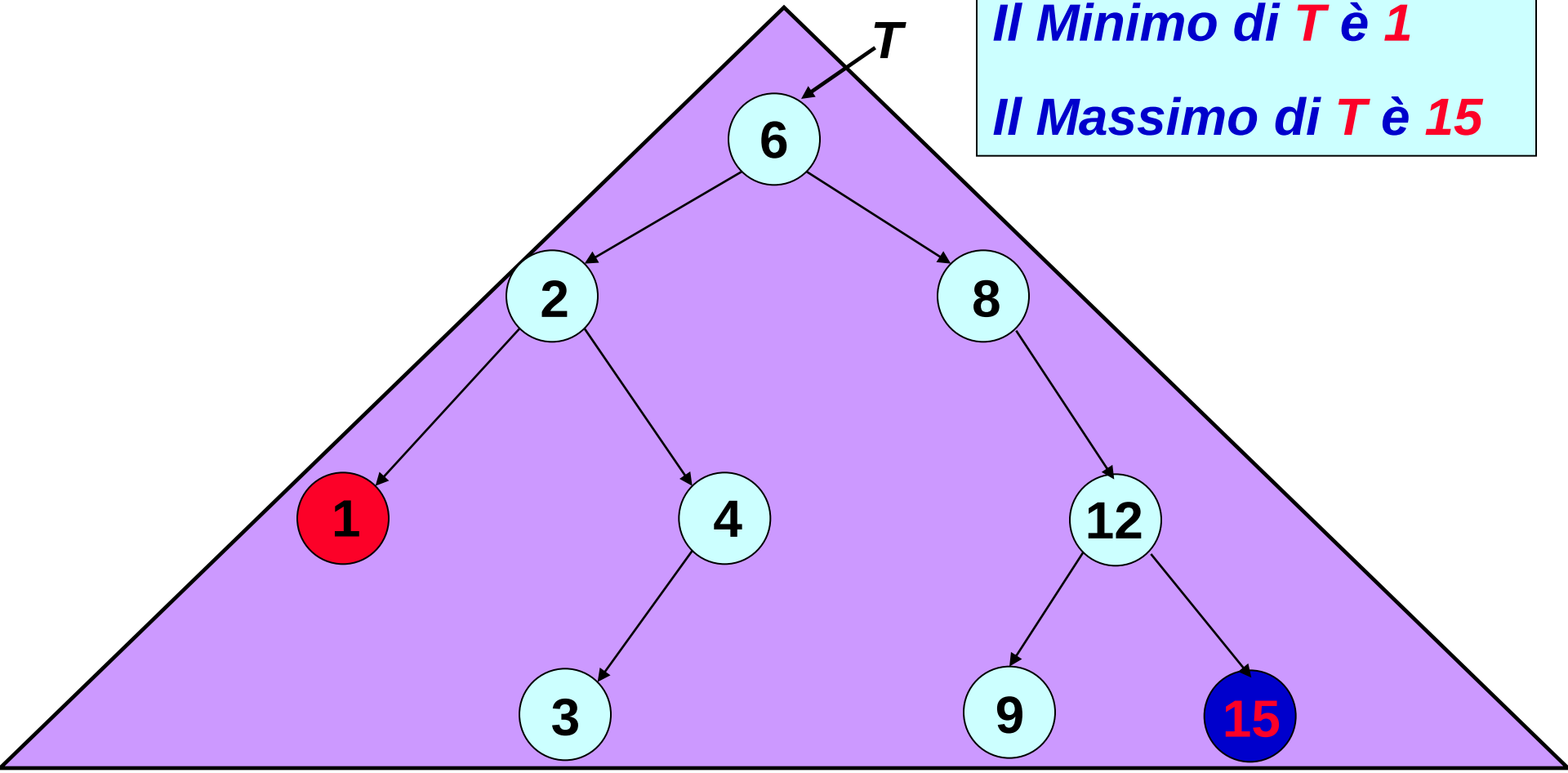
Il Massimo di 2 è 4



ARB: ricerca del minimo e massimo



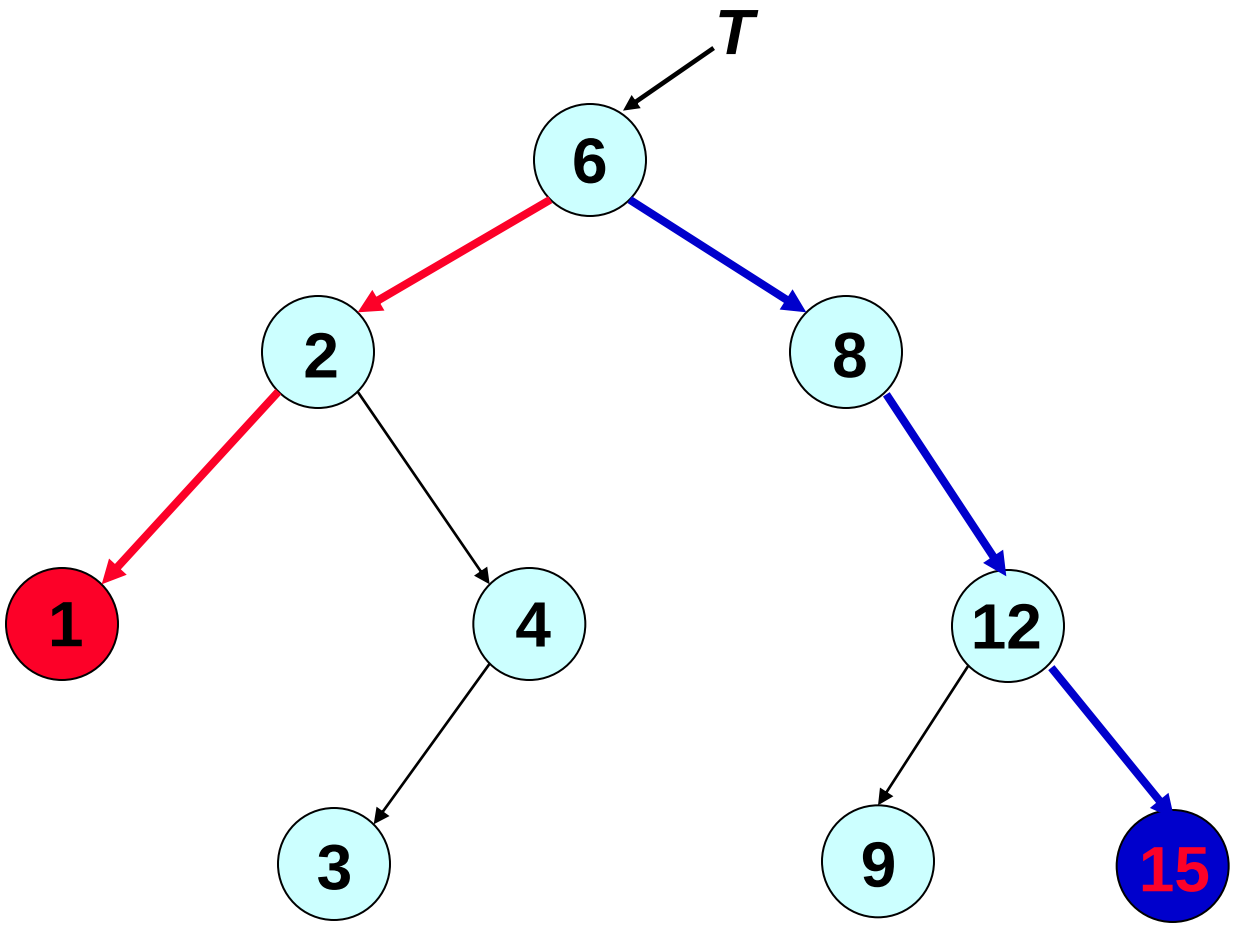
ARB: ricerca del minimo e massimo



Il Minimo di T è 1

Il Massimo di T è 15

ARB: ricerca del minimo e massimo



ARB: ricerca del minimo e massimo

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

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

ARB: ricerca del minimo e massimo

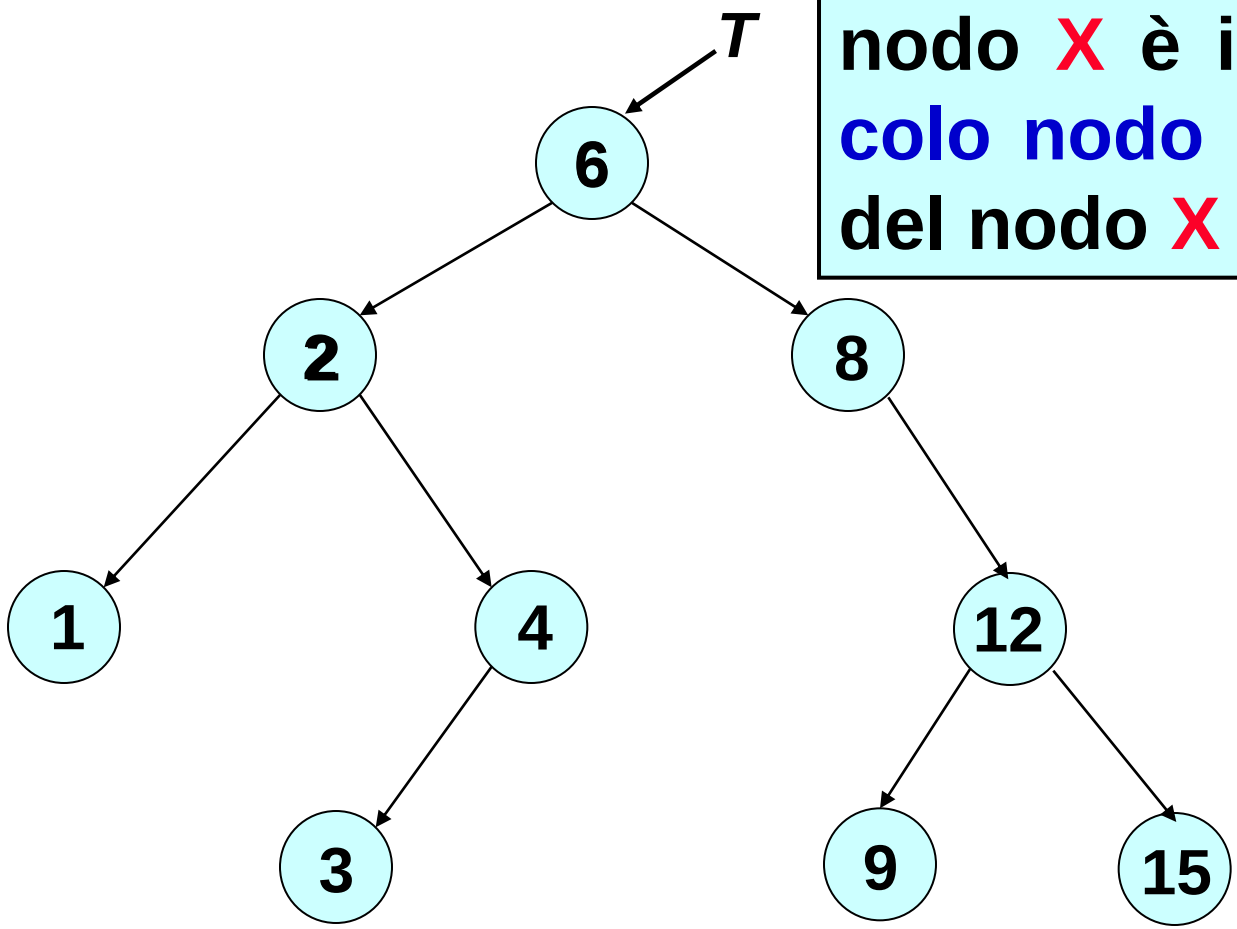
```
ARB ABR-Minimo(x:ARB)
    WHILE x ≠ NIL && sx[x] ≠ NIL DO
          x = sx[x]
    return x
```

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

```
ARB ARB_Minimo(x:ARB)
   IF x ≠ NIL AND sx[x] ≠ NIL THEN
        return ARB_Minimo(sx[x])
   return x
```

ARB: ricerca del successore

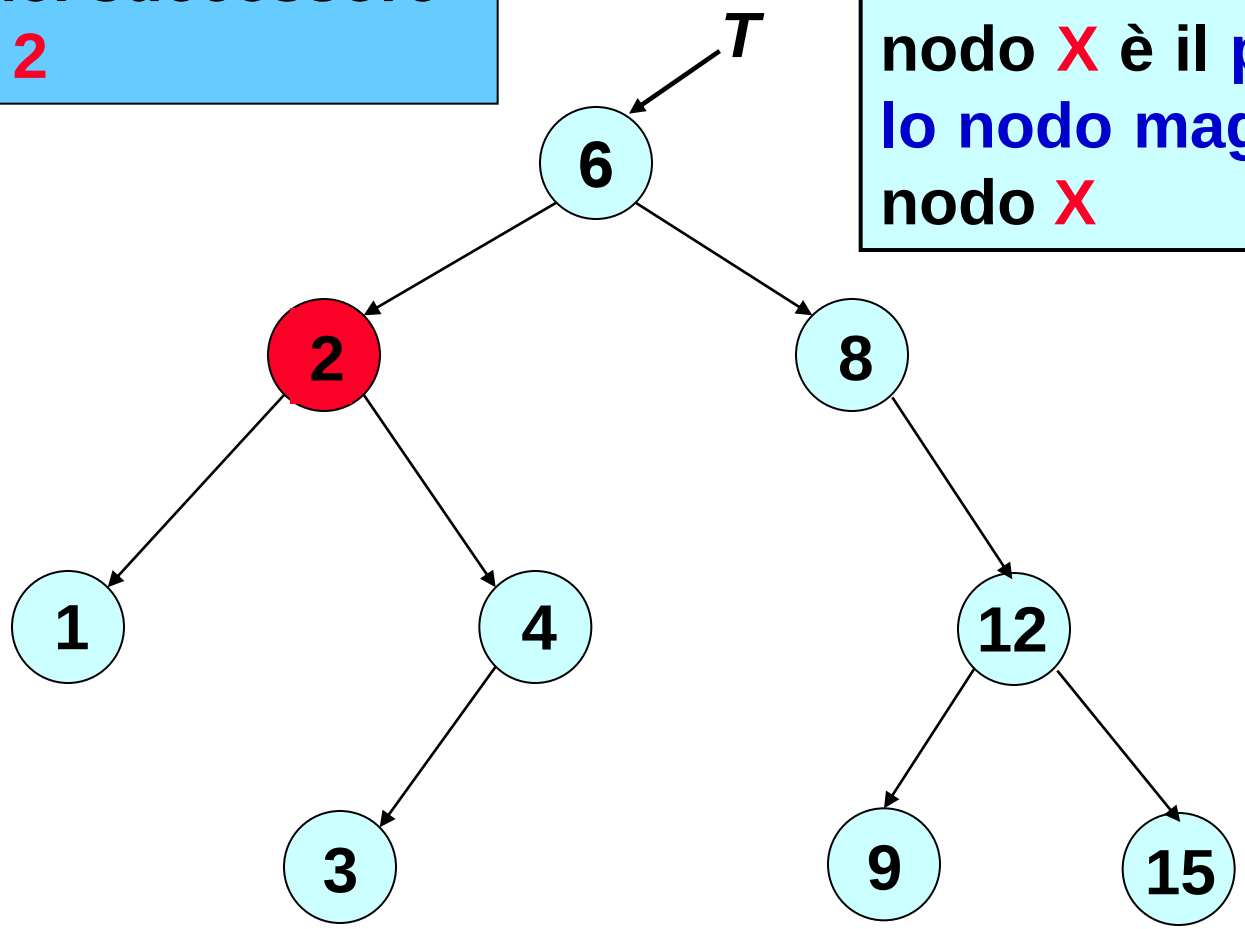
Il successore di un nodo X è il più piccolo nodo maggiore del nodo X



ARB: ricerca del successore

Ricerca **del successore del nodo 2**

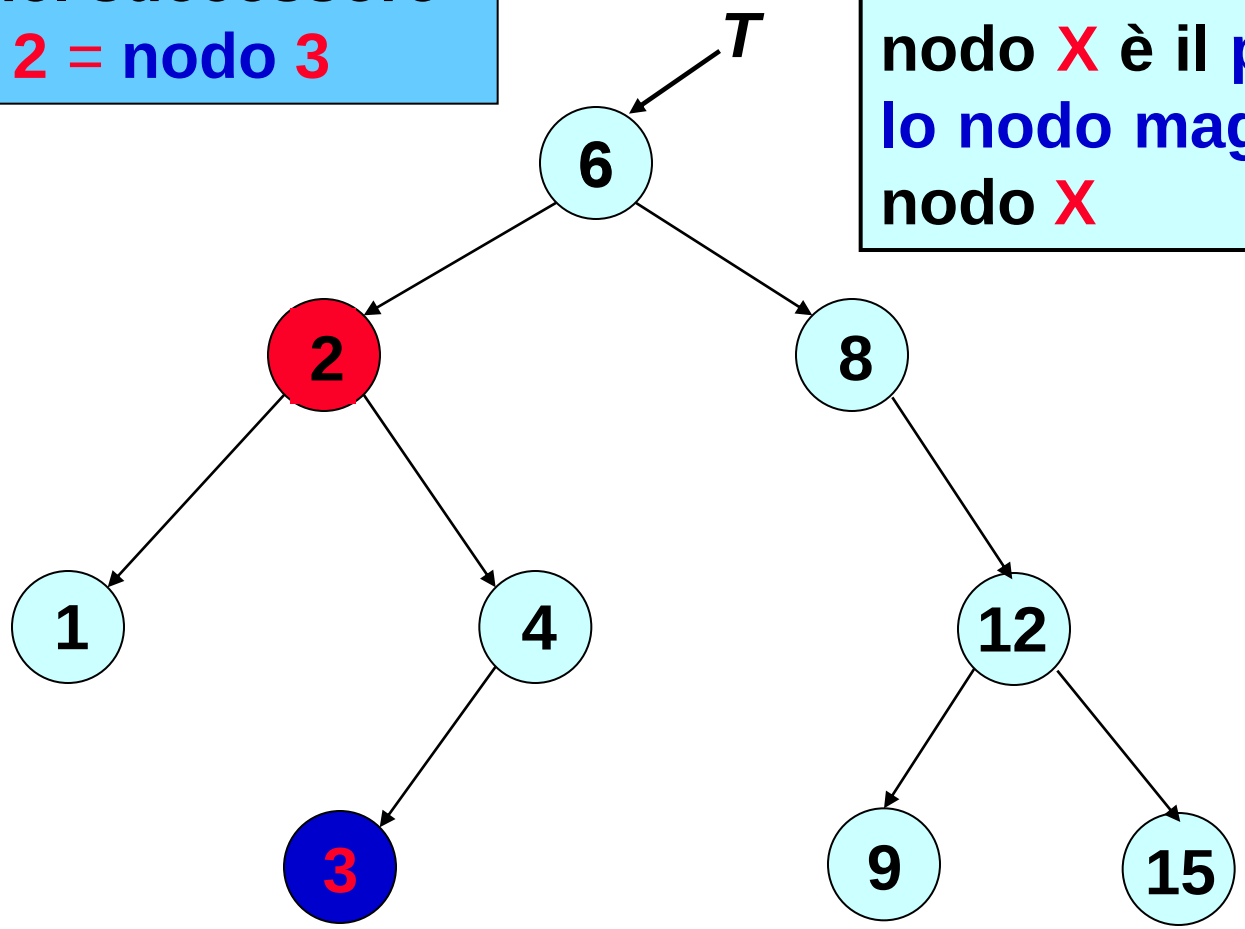
Il successore di un nodo X è il più piccolo nodo maggiore del nodo X



ARB: ricerca del successore

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

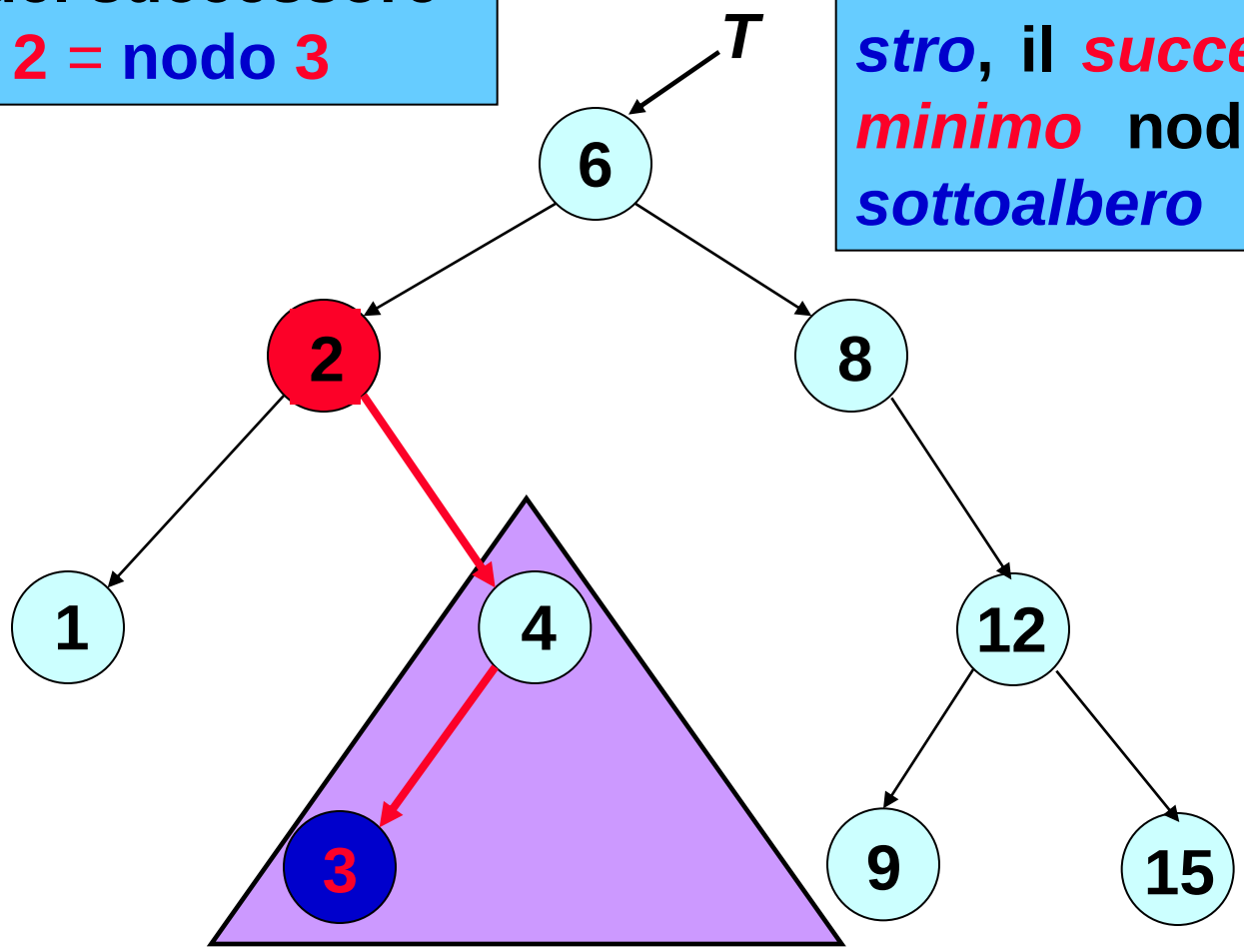
Il successore di un nodo X è il più piccolo nodo maggiore del nodo X



ARB: ricerca del successore

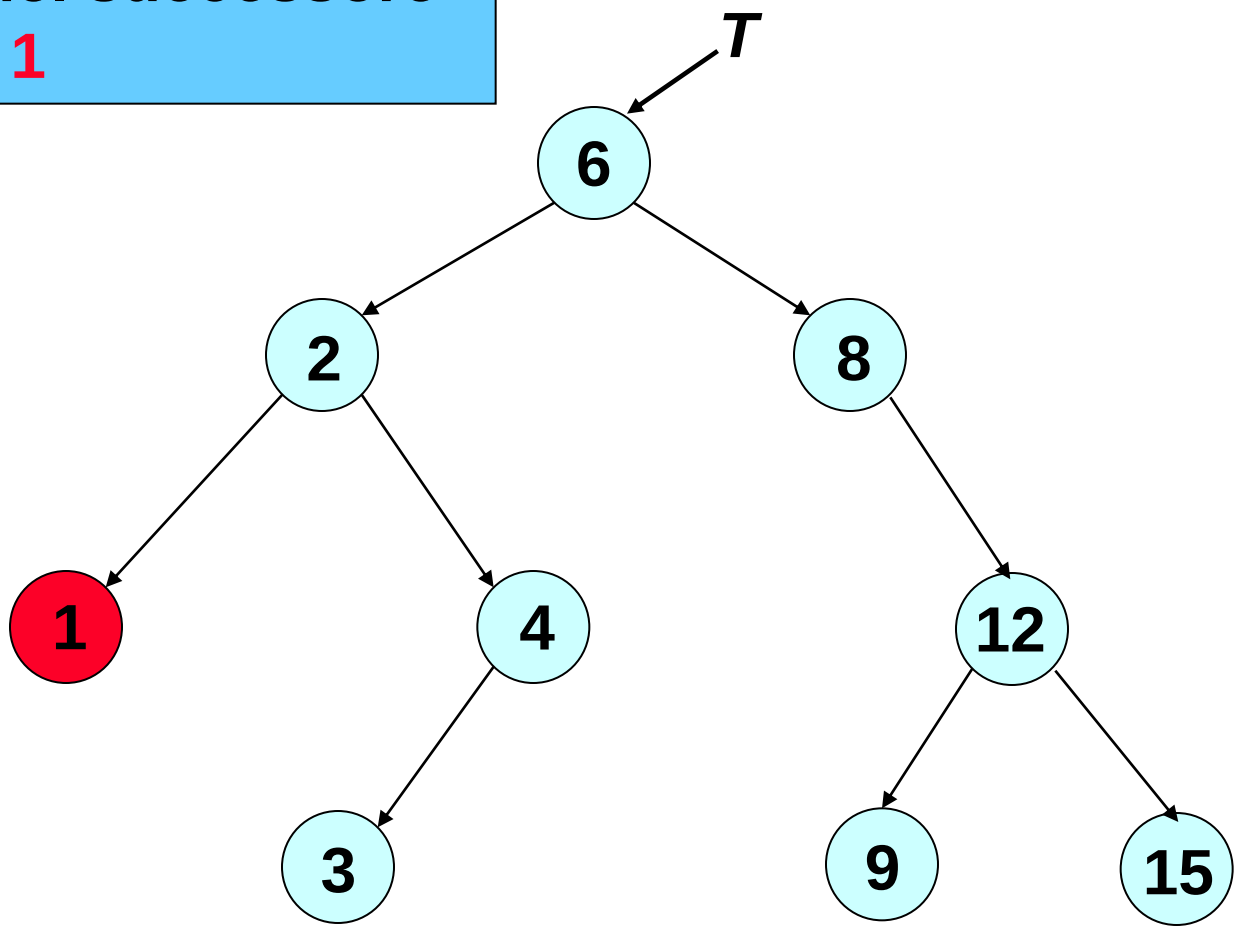
Ricerca **del successore del nodo 2 = nodo 3**

Se *x* ha un *figlio destro*, il *successore* è il *minimo* nodo di quel *sottoalbero*



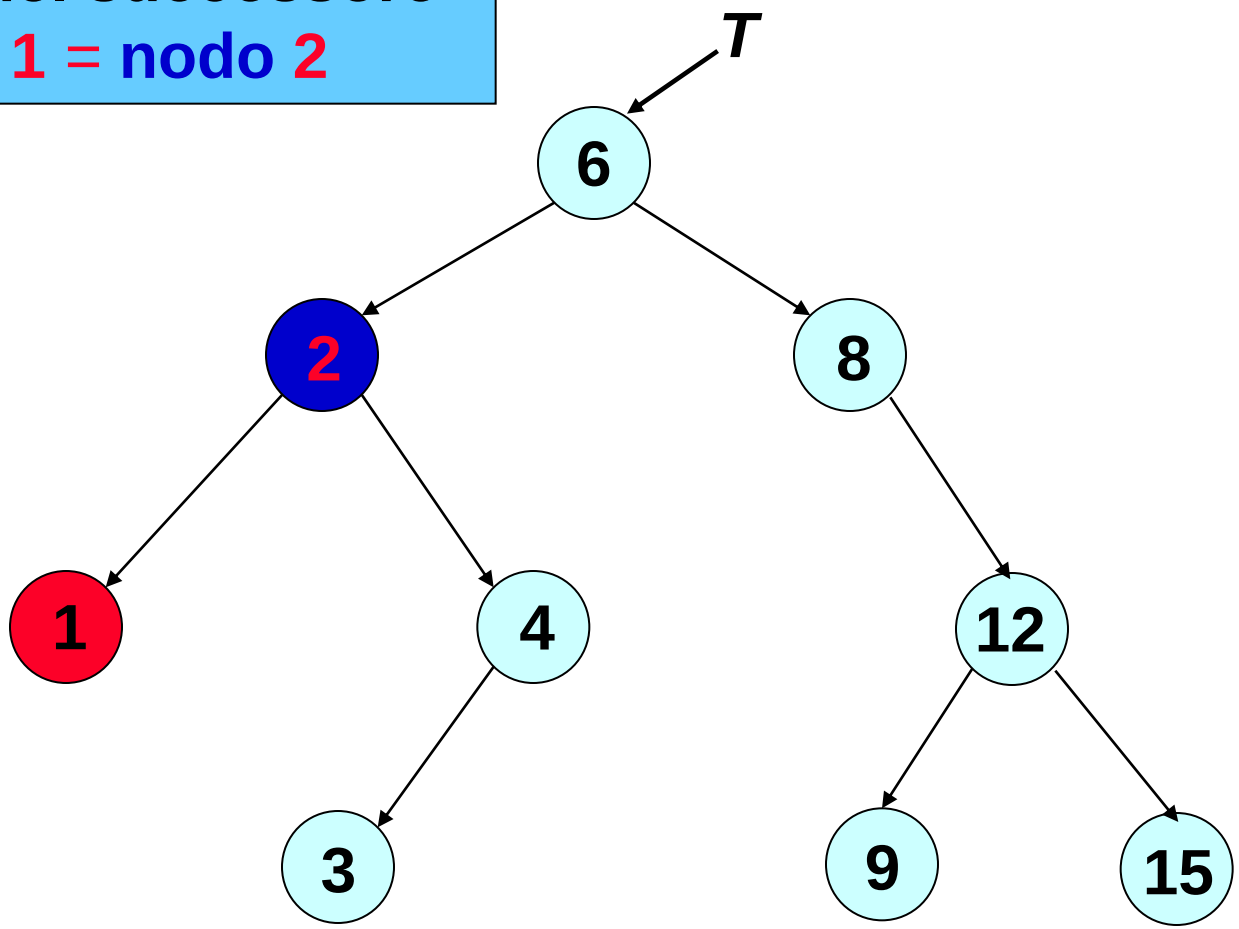
ARB: ricerca del successore

Ricerca **del successore del nodo 1**



ARB: ricerca del successore

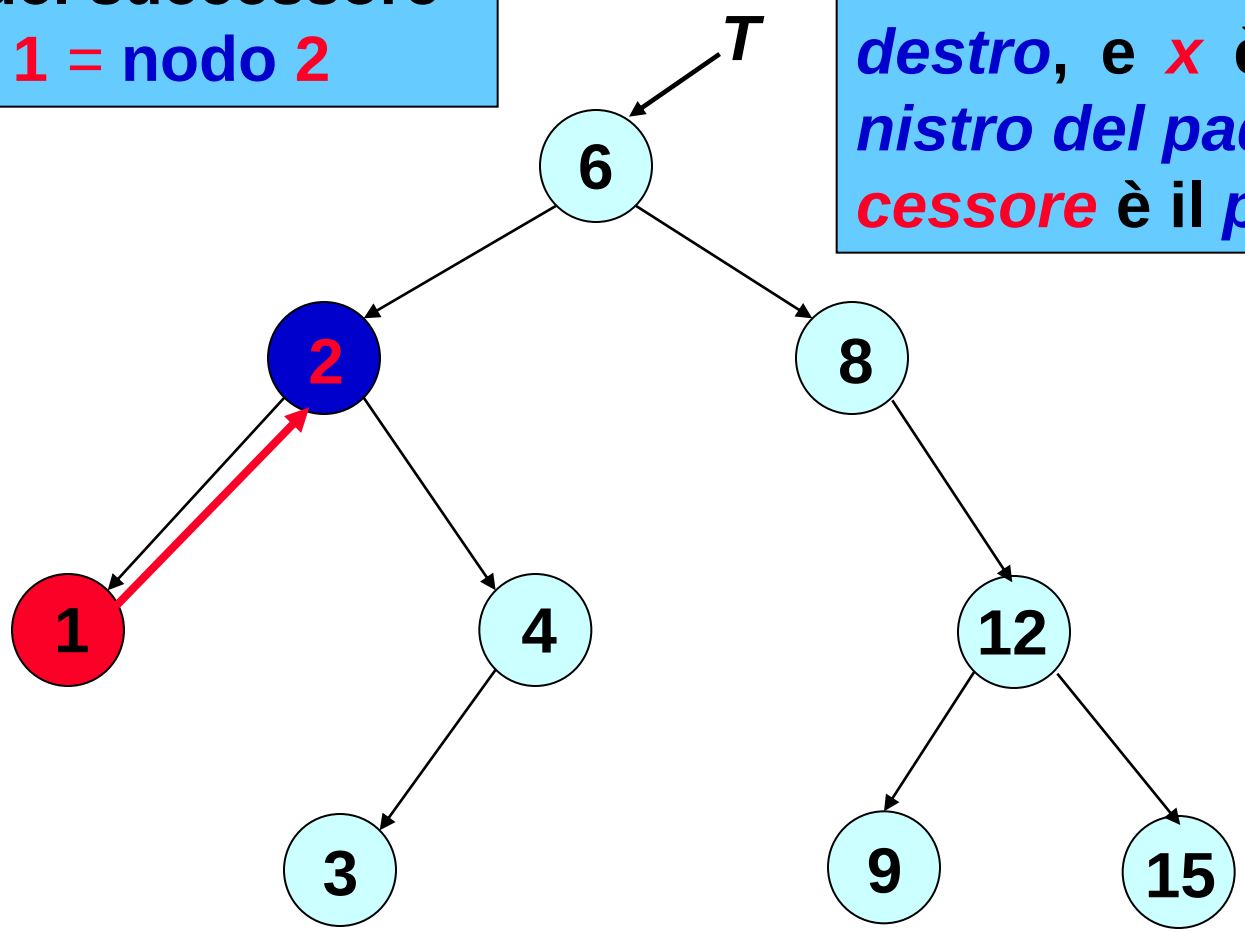
Ricerca **del successore del nodo 1 = nodo 2**



ARB: ricerca del successore

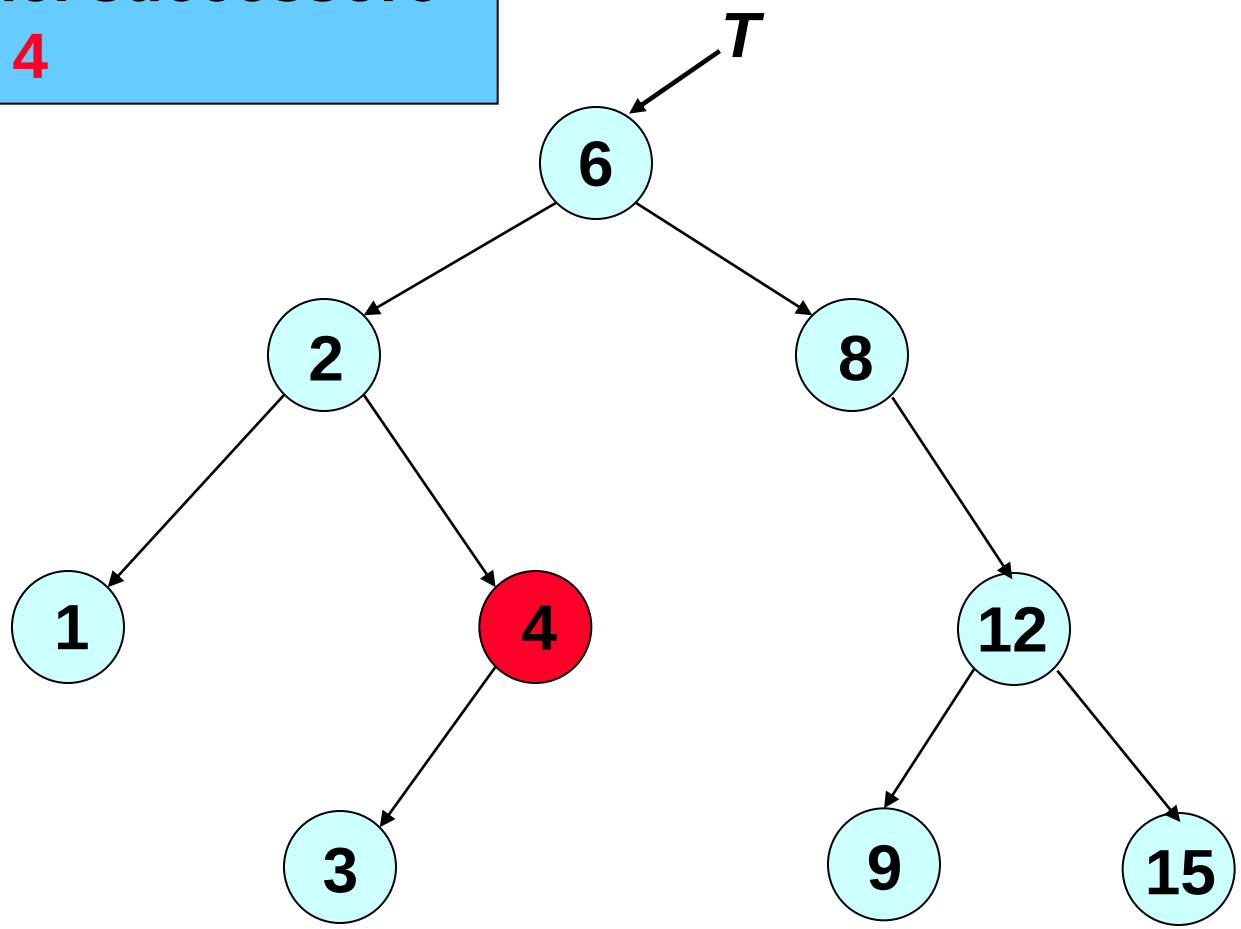
***Ricerca* del successore del nodo 1 = nodo 2**

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



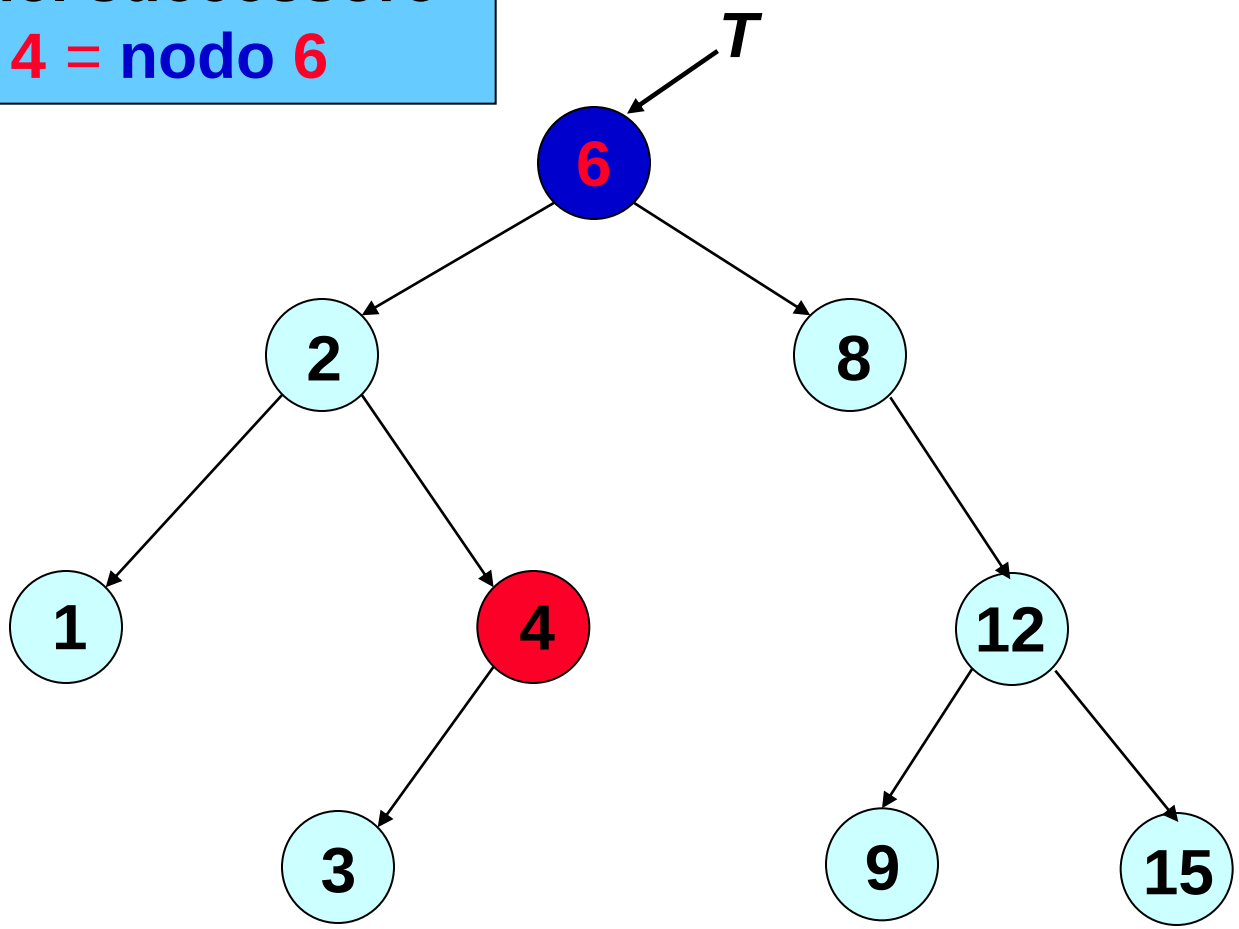
ARB: ricerca del successore

Ricerca **del successore del nodo 4**



ARB: ricerca del successore

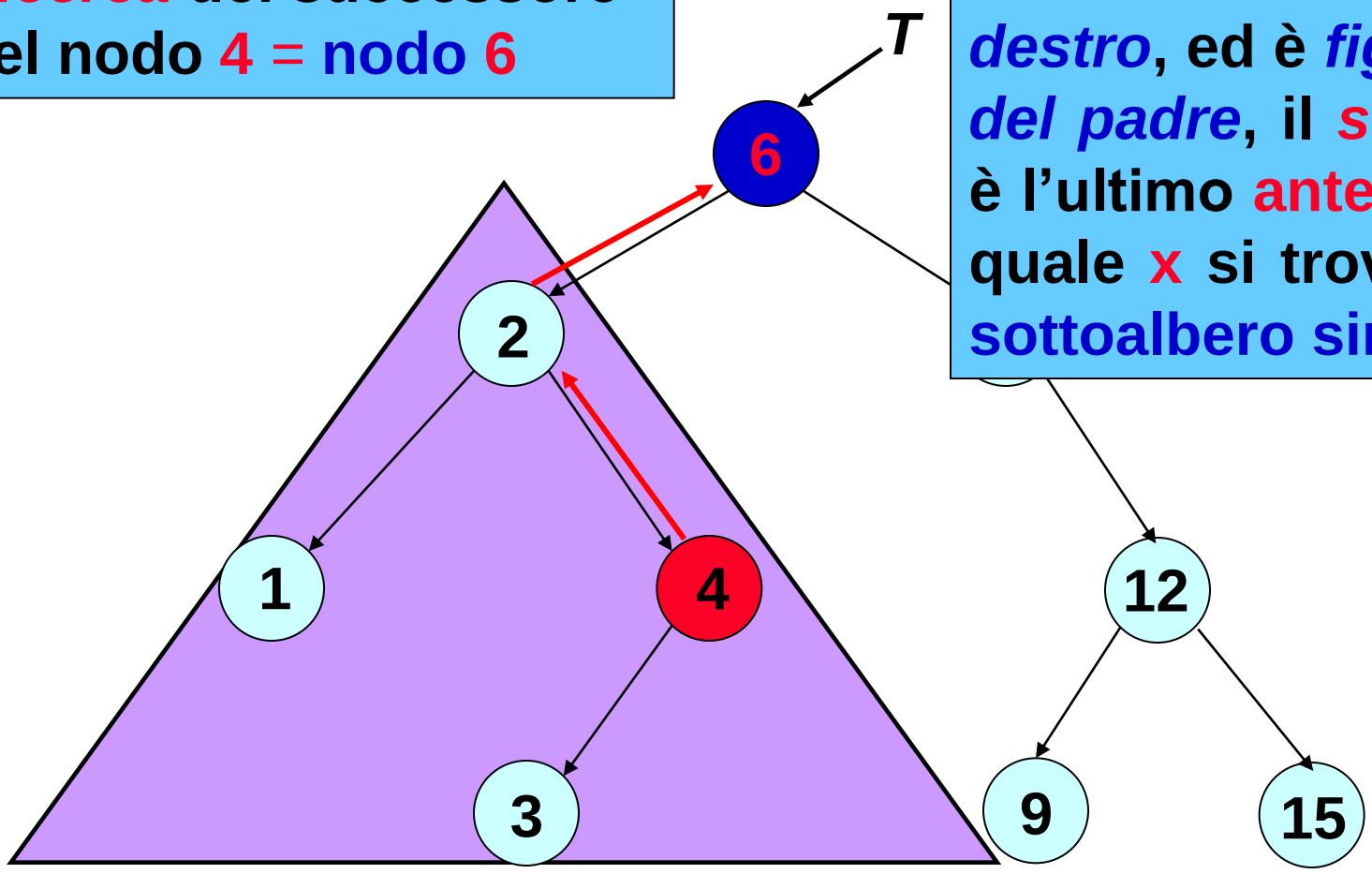
Ricerca **del successore del nodo 4 = nodo 6**



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 l'ultimo antenato per il quale x si trova nel suo sottoalbero sinistro.



ARB: ricerca del successore

```
ABR-Successore(T,k)
   Z = T
   Y = NIL
   WHILE (Z!=NIL && key[Z]!=k)
      Y = Z
      IF (key[Z] < k) THEN
          Z = dx[Y]
      ELSE IF (key[Z] > k) THEN
          Z = sx[Y]
    IF (Z != NIL && dx[Z]!=NIL) THEN
          return ABR-Minimo(dx[Z])
    ELSE
      WHILE (Y != NIL AND Z = dx[Y]) DO
            Z = Y
            Y = padre[Z]
      return Y
```

Se *x* NON ha un *figlio destro*, ed è *figlio destro del padre*, il *successore* è l'ultimo antenato per il quale x si trova nel suo sottoalbero sinistro.

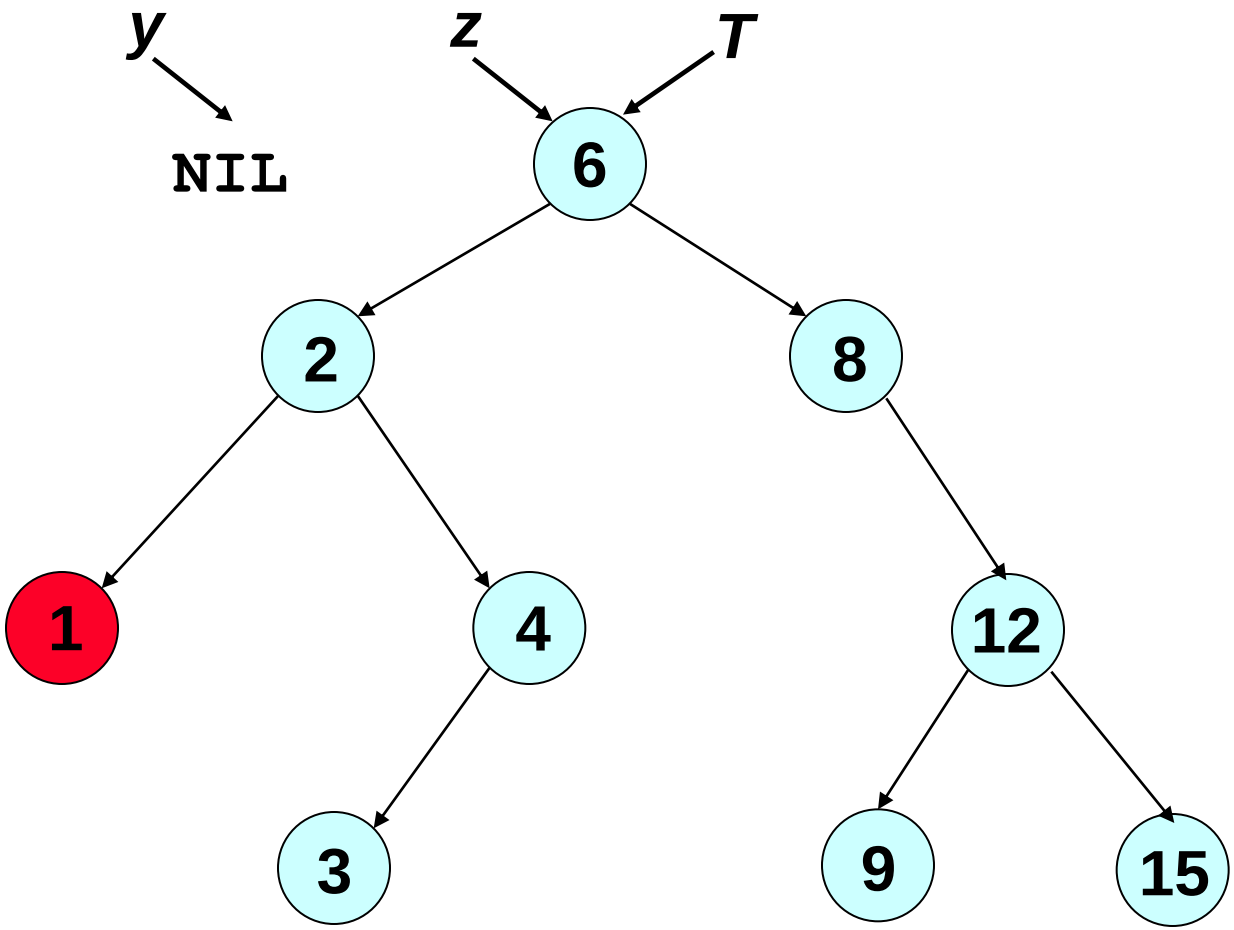
ARB: ricerca del successore

```
ABR-Successore(T,k)
  Z = T
  Y = NIL
  WHILE (Z!=NIL && key[Z]!=k)
     Y = Z
     IF (key[Z] < k) THEN
        Z = f_dx[Y]
     ELSE IF (key[Z] > k) THEN
        Z = f_sx[Y]
```

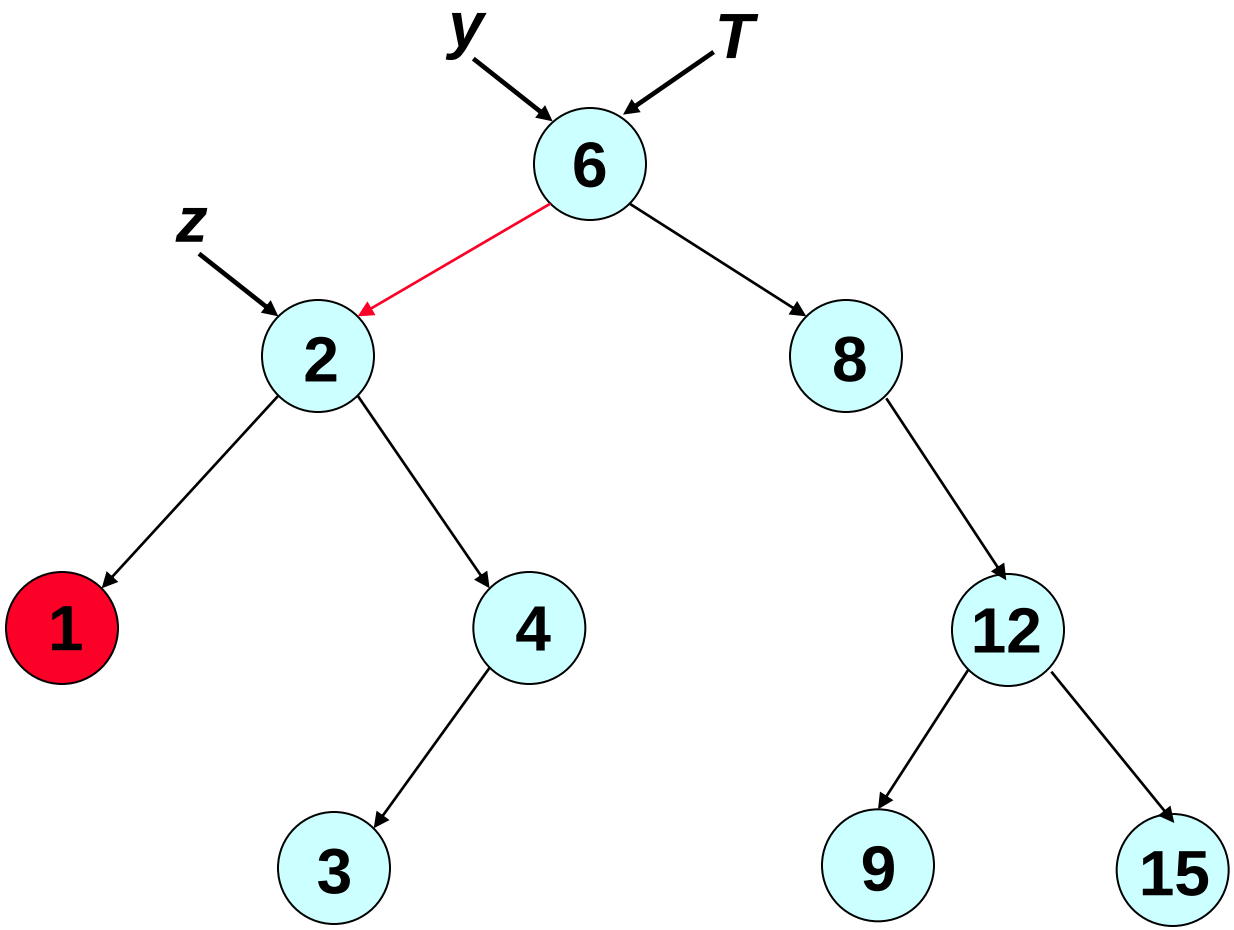
Questo algoritmo assume che ogni nodo abbia il puntatore al padre

```
      WHILE (Y != NIL AND Z = f_dx[Y]) DO
          Z = Y
          Y = padre[Z]
      return Y
```

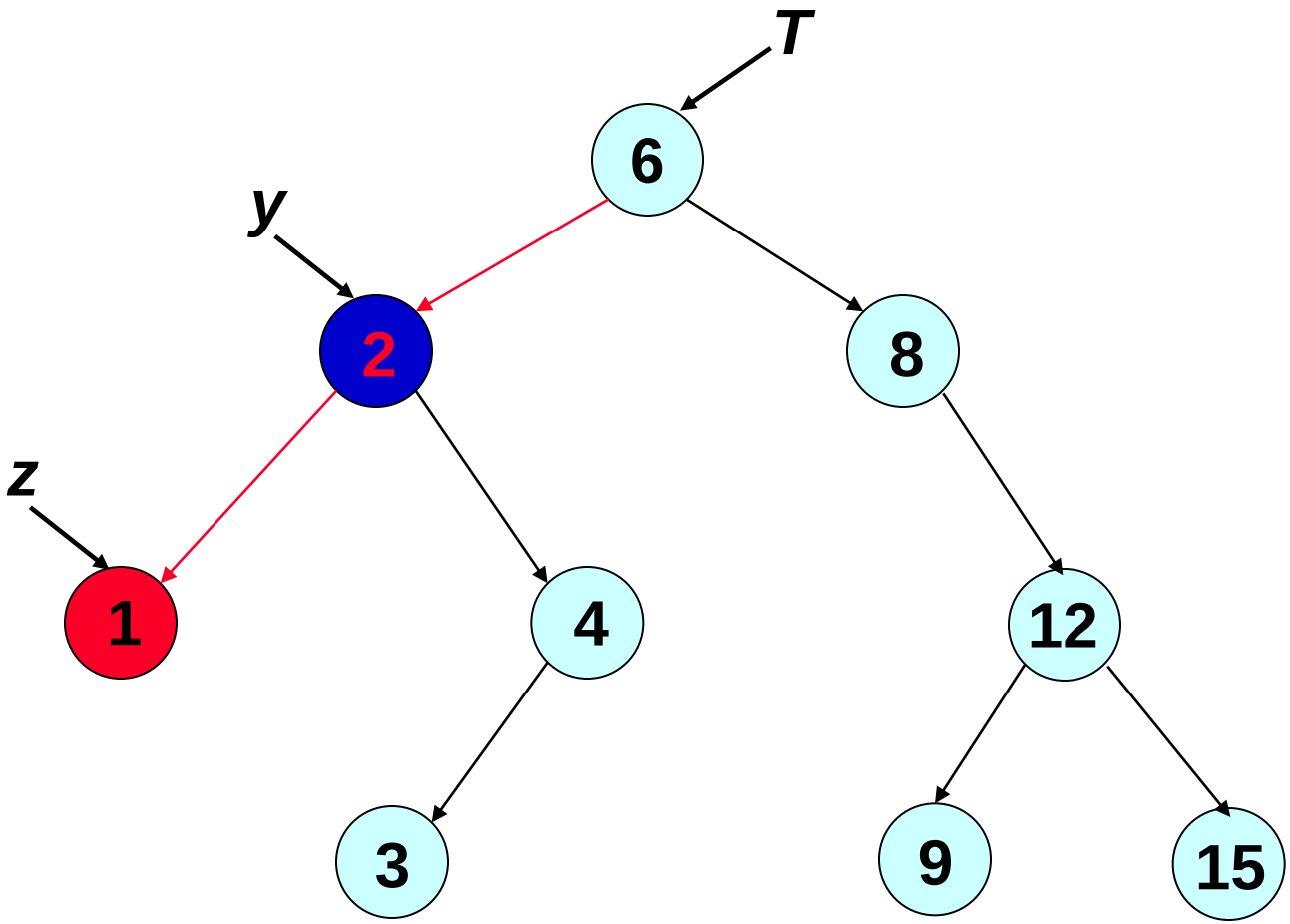
ARB: ricerca del successore II



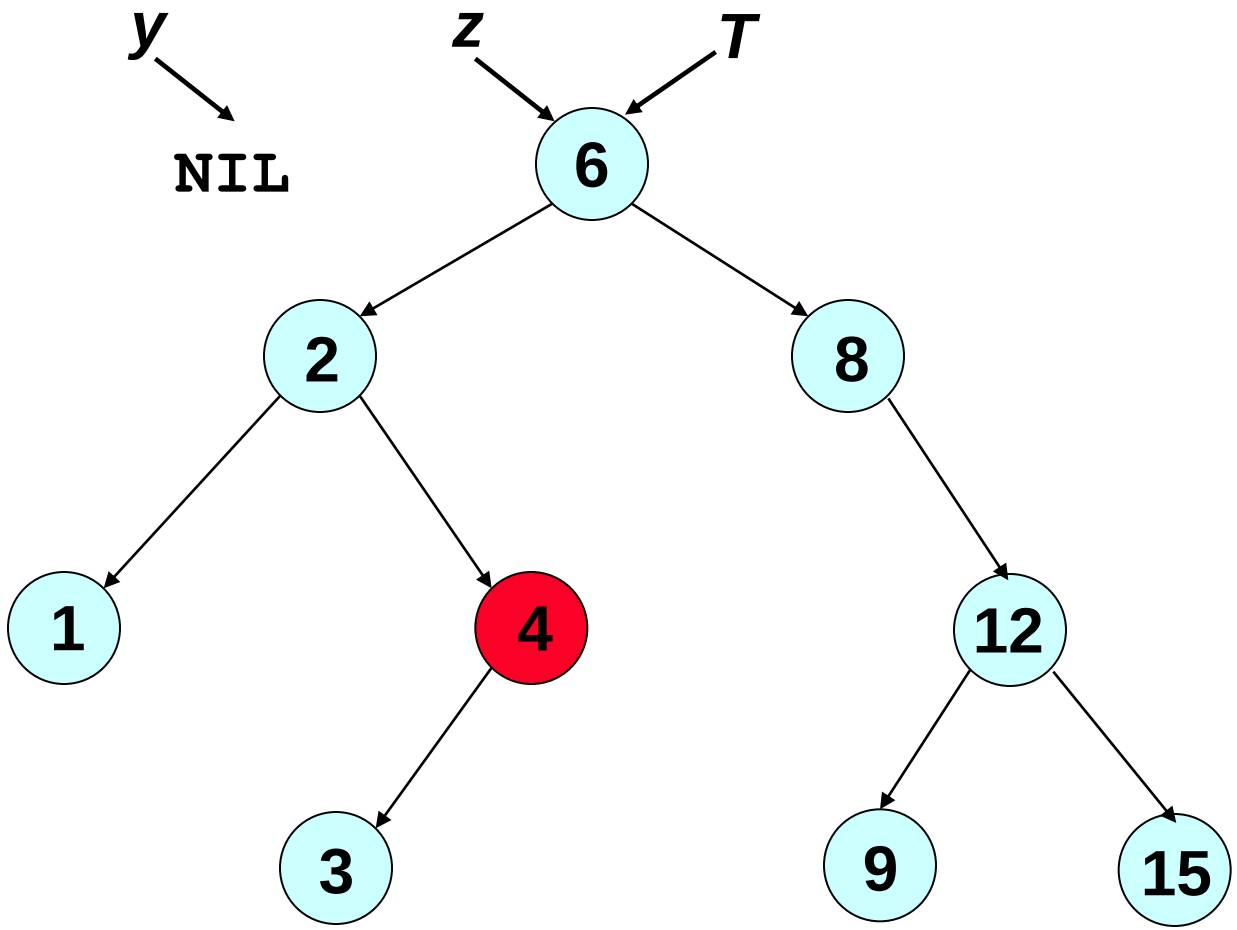
ARB: ricerca del successore II



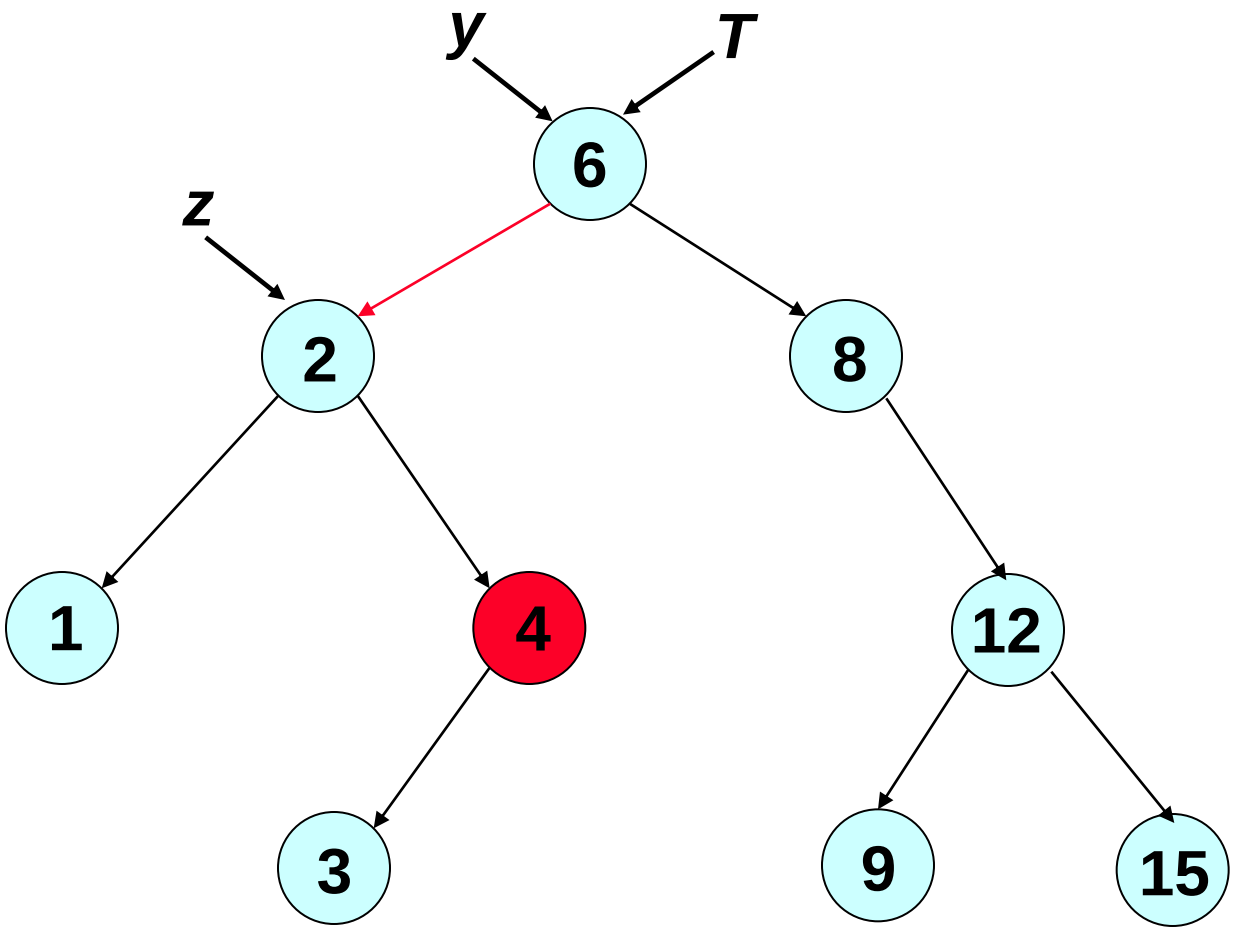
ARB: ricerca del successore II



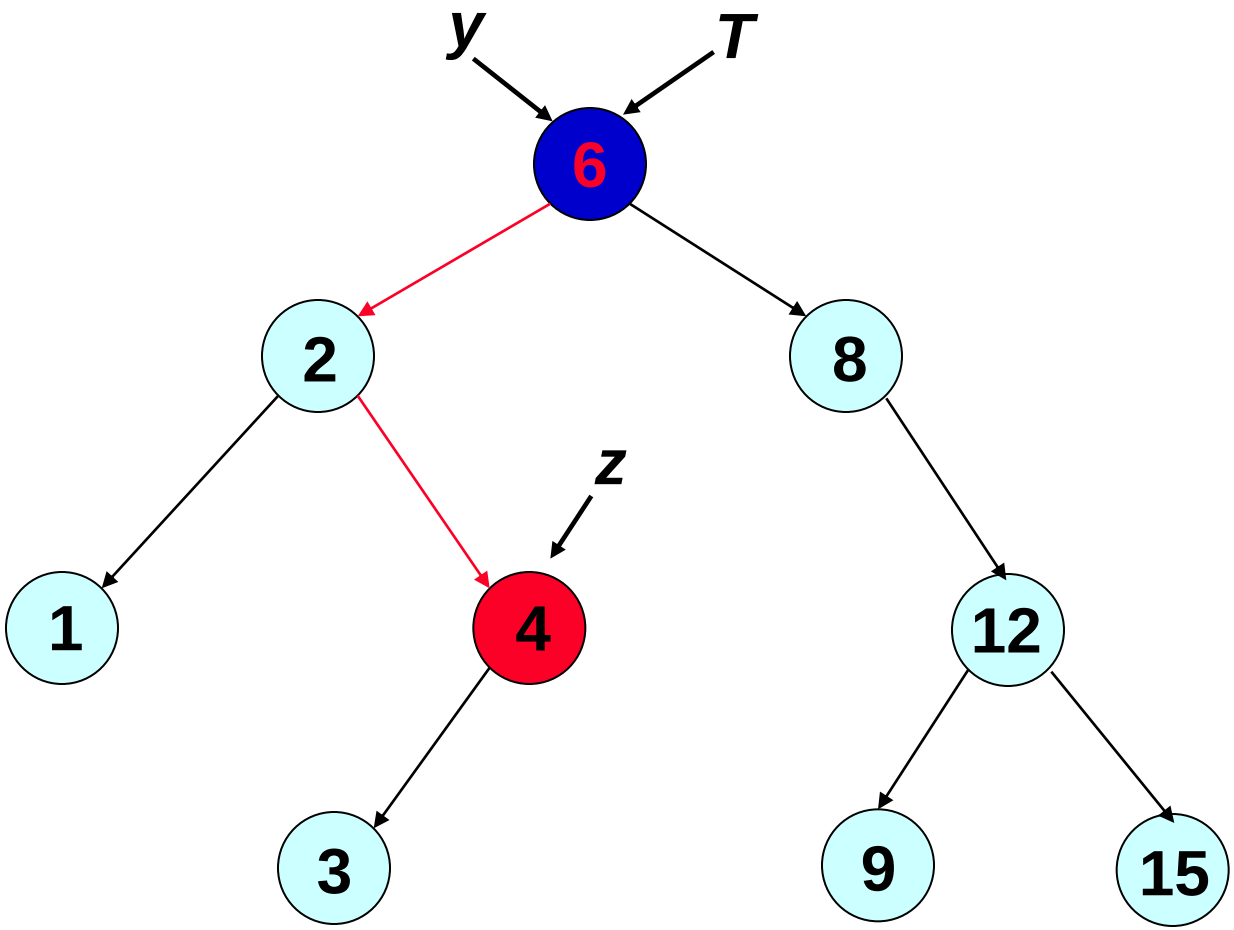
ARB: ricerca del successore II



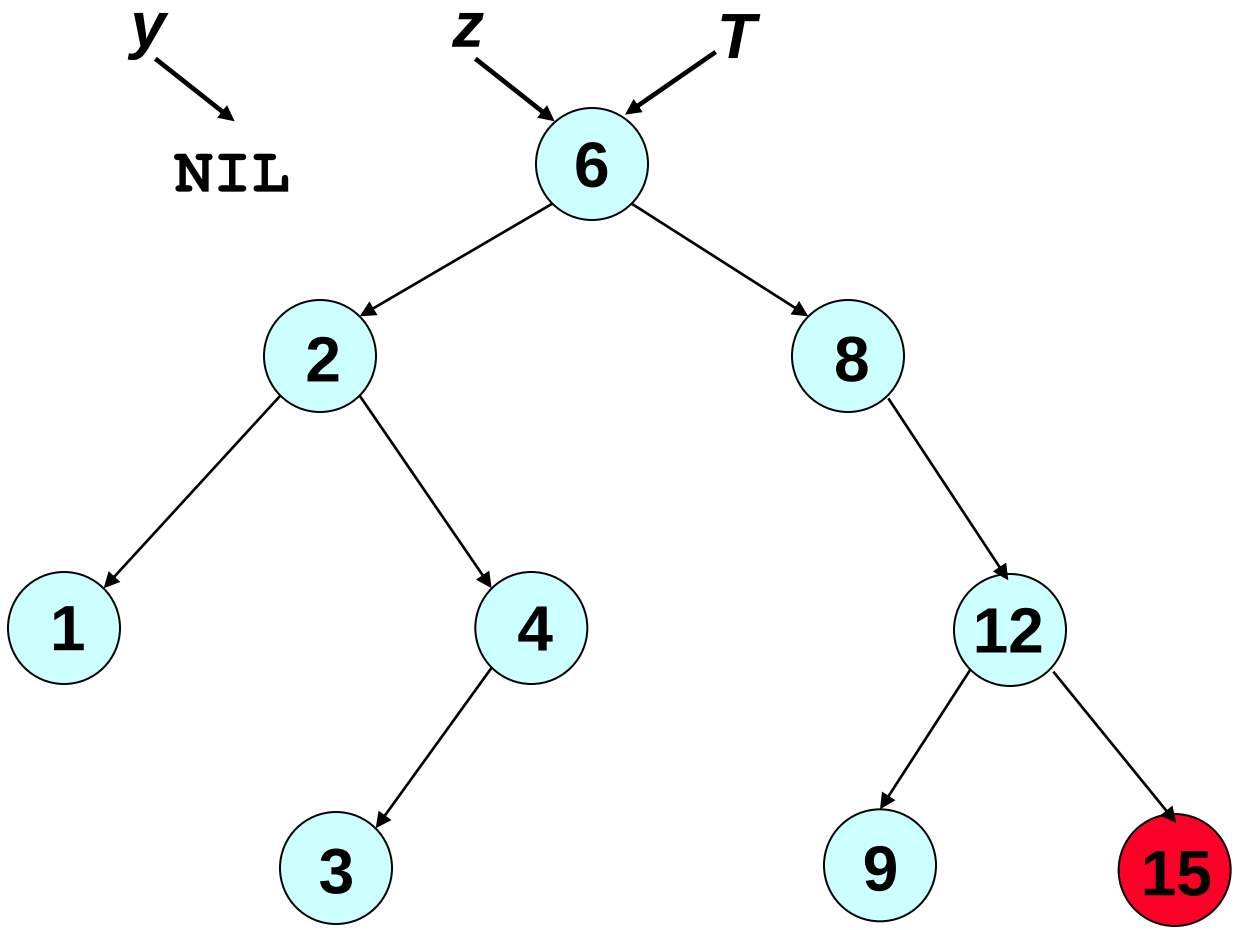
ARB: ricerca del successore II



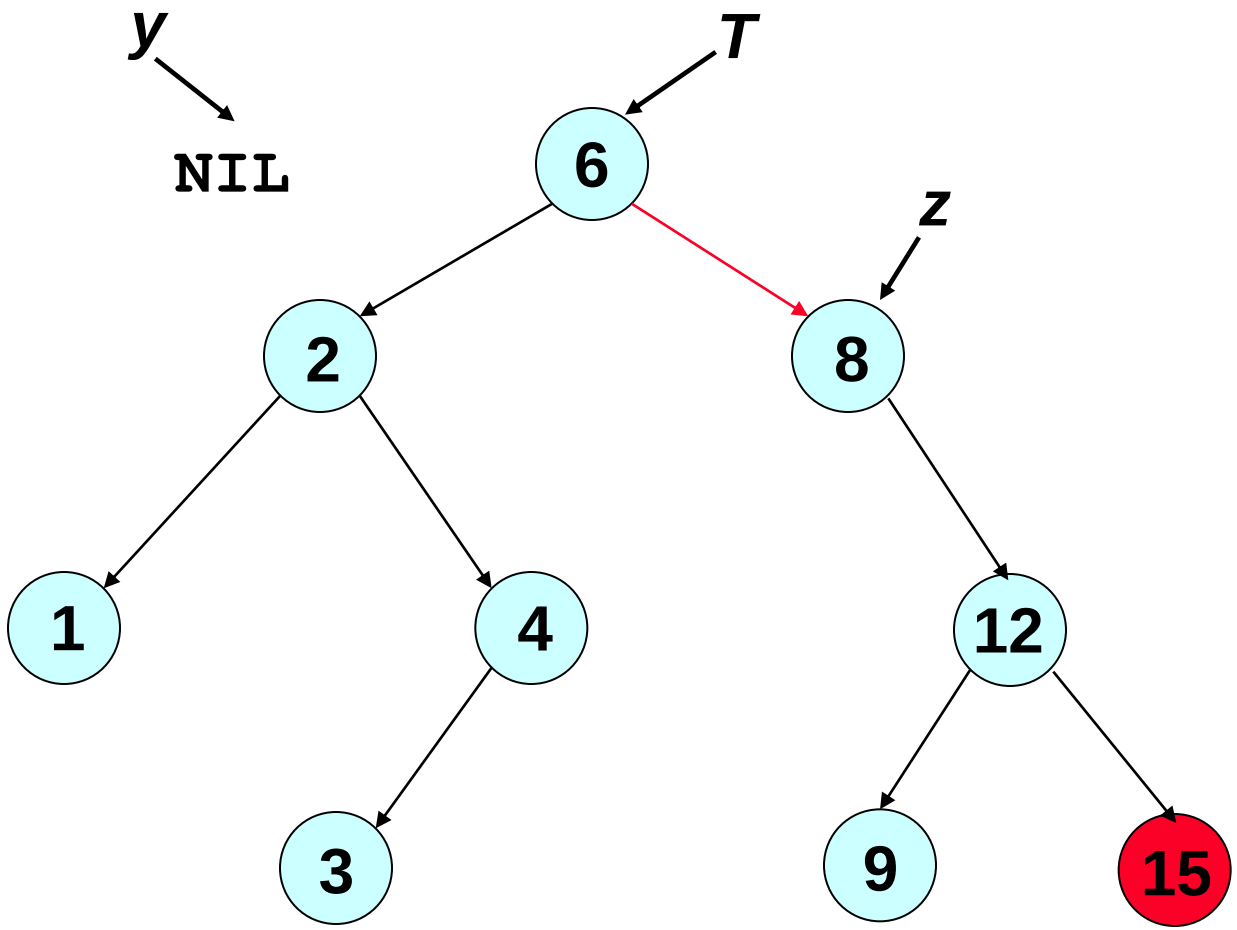
ARB: ricerca del successore II



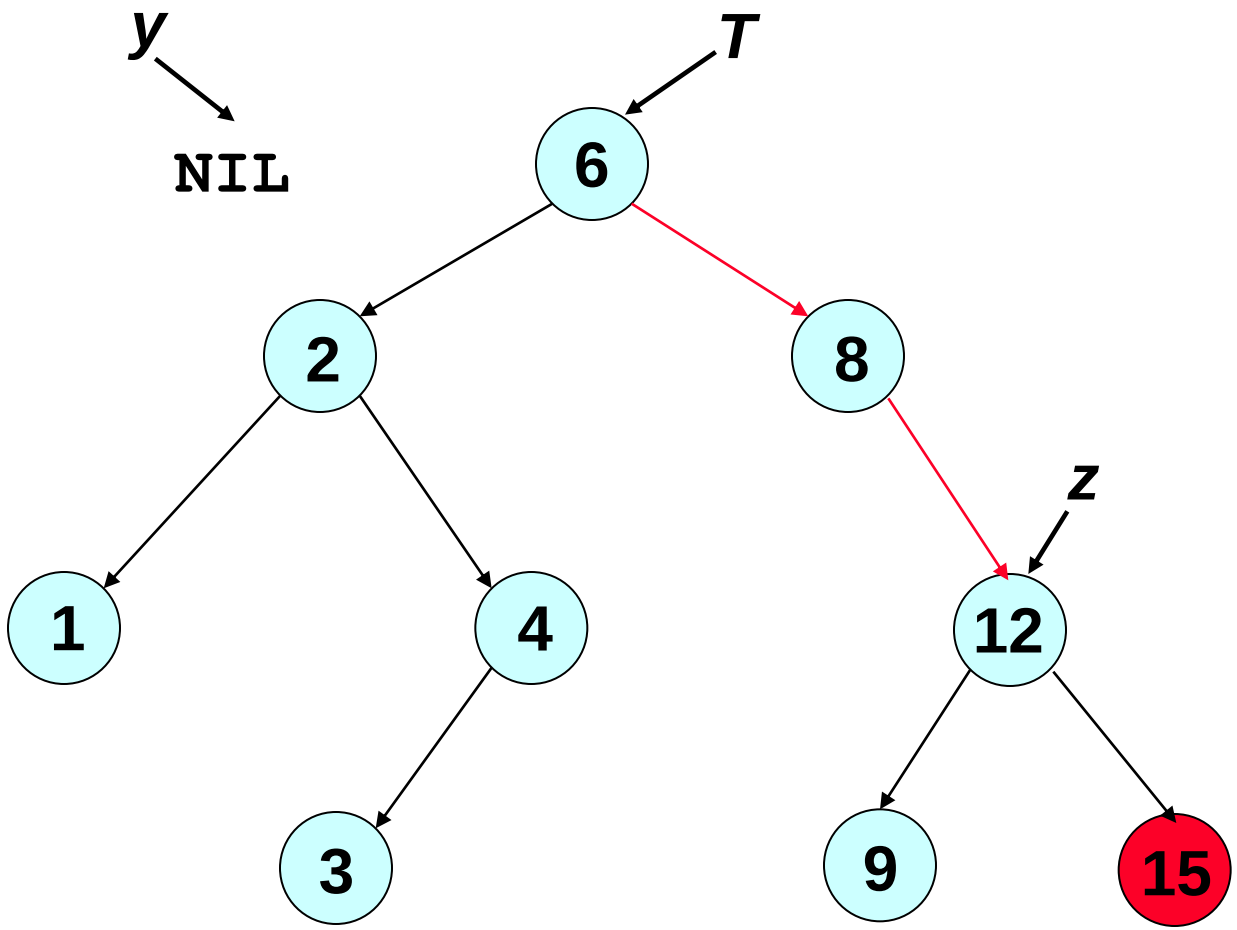
ARB: ricerca del successore II



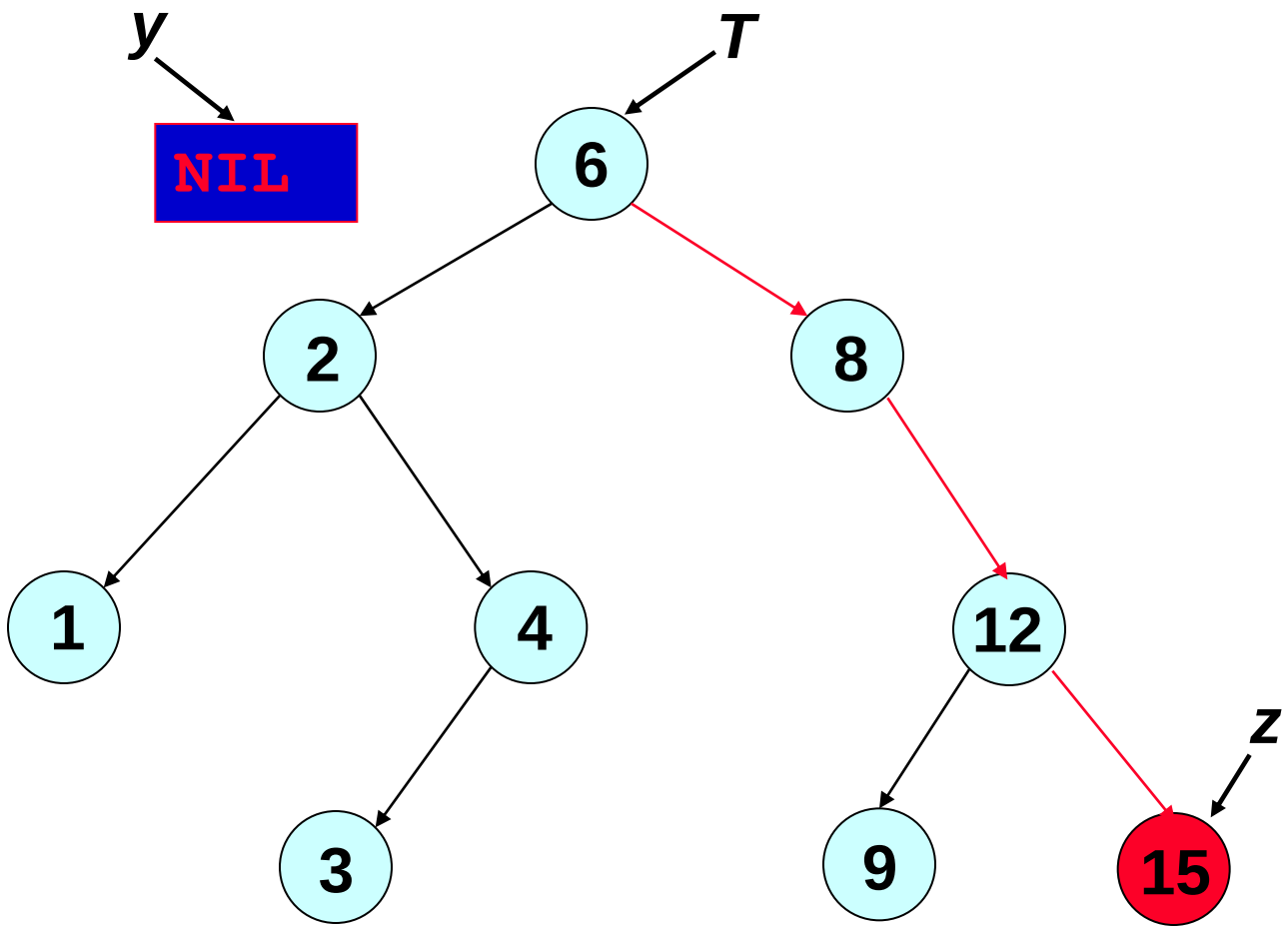
ARB: ricerca del successore II



ARB: ricerca del successore II



ARB: ricerca del successore II



ARB: ricerca del successore II

- **Inizializziamo il *successore* a *NIL***

- **Partendo dalla radice dell'albero:**
  - **ogni volta che si segue il *ramo sinistro* per raggiungere il nodo, si aggiorna il successore al nodo padre;**
  - **ogni volta che si segue un *ramo destro* per raggiungere il nodo, NON si aggiorna il successore al nodo padre;**

ARB: ricerca del successore

y punta sempre al nodo candidato a essere successore

```
ARB ABR-Successore'(T, k)
   z = T
   y = NIL
   WHILE (z != NIL && key[z] != k)
       IF (key[z] < k)
                z = dx[z]
       ELSE IF (key[z] > k)
                y = z
                z = sx[z]
   IF (z != NIL && dx[z] != NIL) THEN
          y = ABR-Minimo(dx[z])
   return y
```

ARB: ricerca del successore ricorsiva

```
ABR-Successore_ric'(T,k,Y)
  IF (T != NIL) THEN
    IF (k > key[T]) THEN
       return ABR-Successore_ric'(dx[T],k,Y)
    ELSE IF (k < key[T]) THEN
       return ABR-Successore_ric'(sx[T],k,T)
    ELSE /* k = key[T] */
       IF (dx[T] != NIL) THEN
            return ABR-Minimo(dx[T])
  return Y
```

```
ABR-Successore'(T,k)
   return ABR-Successore_ric'(T,k,NIL)
```

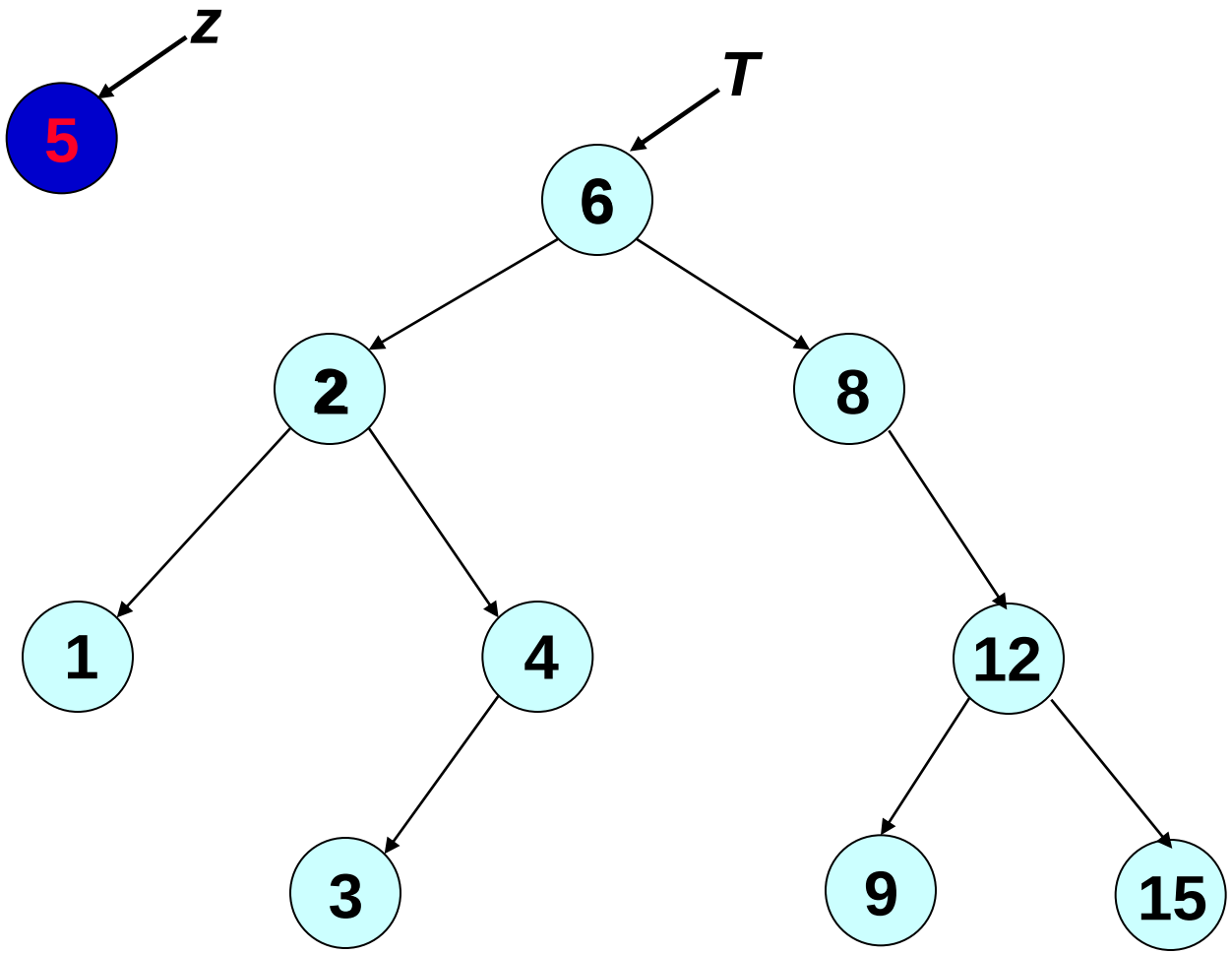
ARB: ricerca del successore ricorsiva

```
ABR-Successore_ric(T,k)
  IF (T != NIL) THEN
    IF (key[T] < k) THEN
          return ABR-Successore_ric(dx[T],k)
    ELSE IF (key[T] = k)
          return ABR-Minimo(dx[T])
    ELSE /* key[T] > key */
          succ = ABR-Successore_ric(sx[T],k)
          IF (succ != NIL) THEN
              return succ
          ELSE
              return T
  ELSE
     return NIL
```

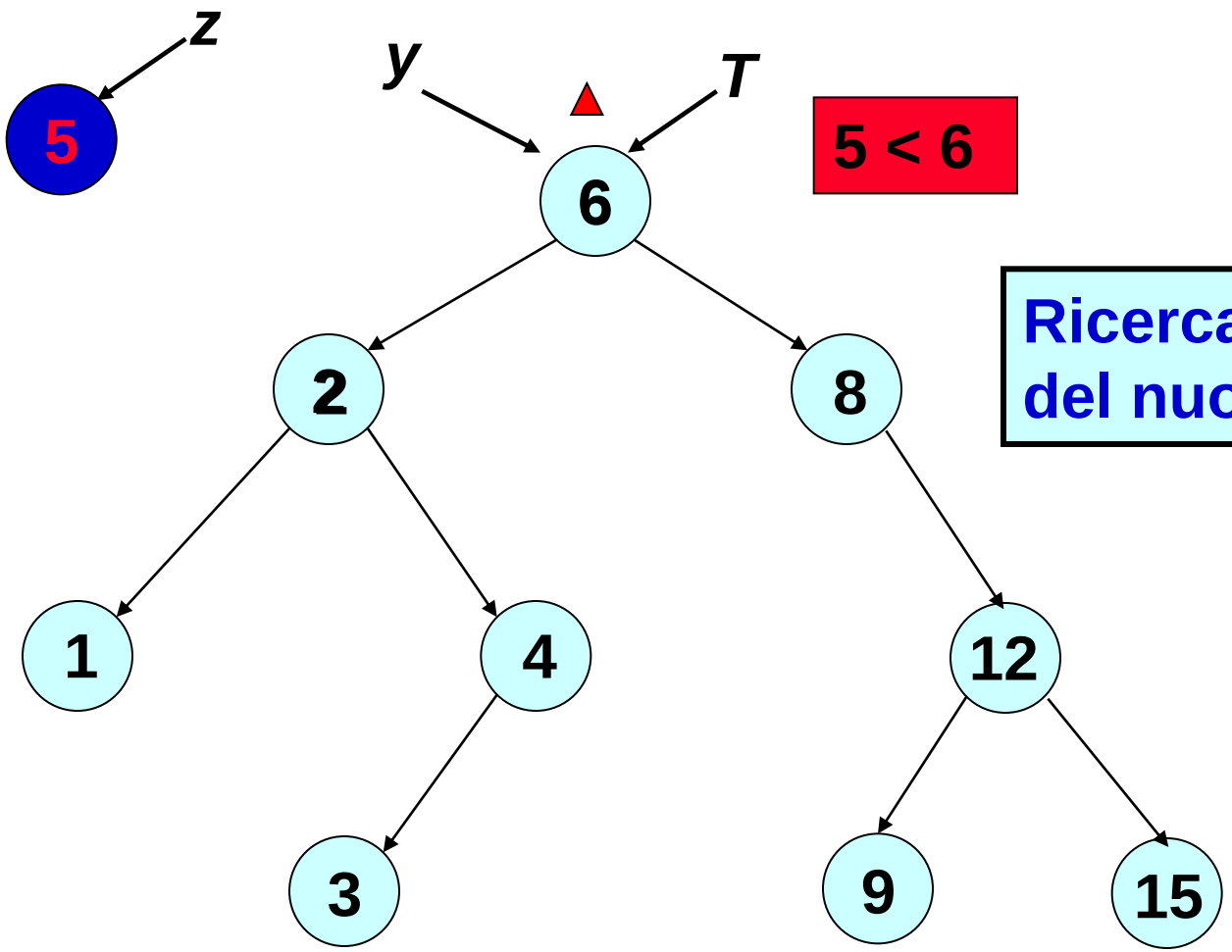
ARB: costo delle operazioni

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

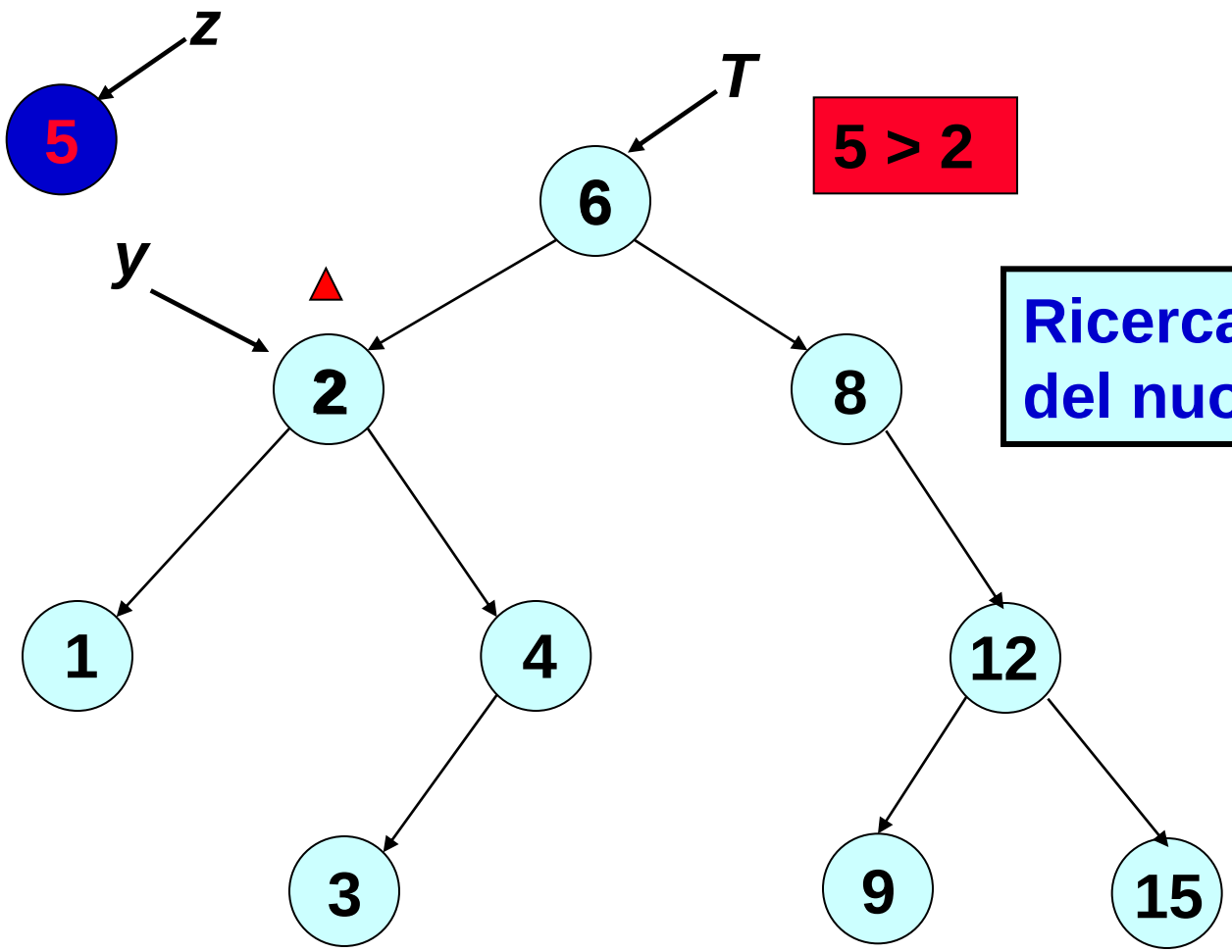
ARB: Inserimento di un nodo (caso I)



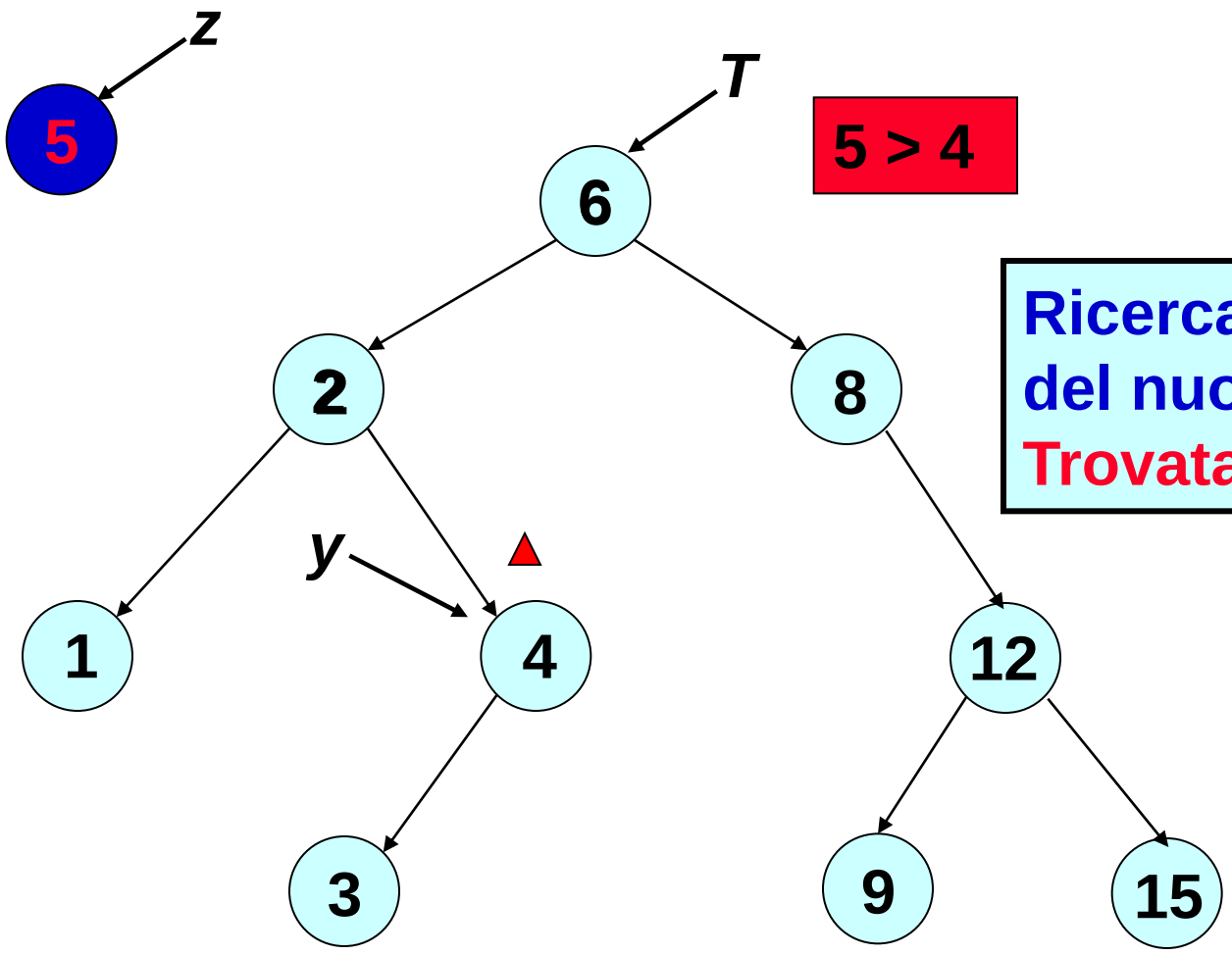
ARB: Inserimento di un nodo (caso I)



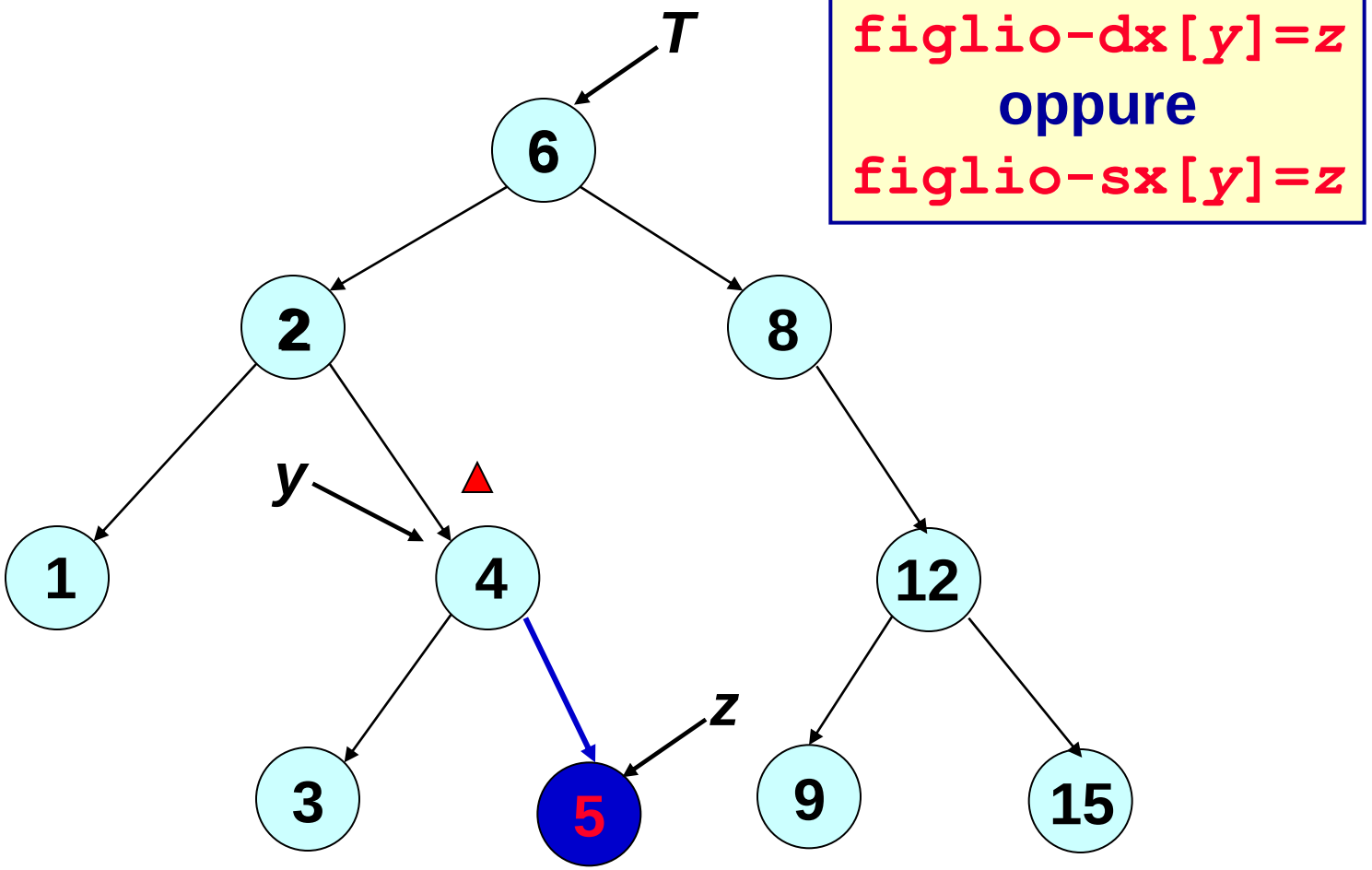
ARB: Inserimento di un nodo (caso I)



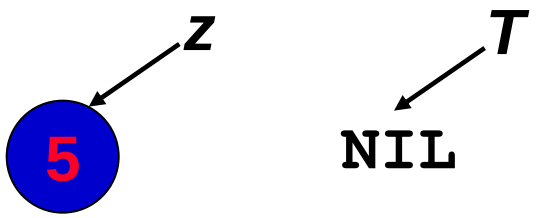
ARB: Inserimento di un nodo (caso I)



ARB: Inserimento di un nodo (caso I)

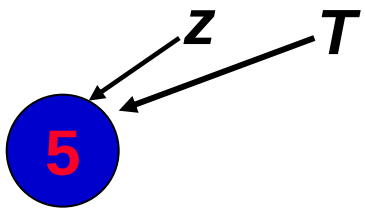


ARB: Inserimento di un nodo (caso II)



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,k)
    z = alloca nodo ARB
    key[z] = k
    y = NIL
    x = T
    WHILE (x != NIL)
      DO y = x
         IF (key[z] < key[x])
           THEN x = sx[x]
           ELSE x = dx[x]
    IF y = NIL THEN
           T = z
    ELSE IF key[z] < key[y] THEN
              sx[y] = z
         ELSE
              dx[y] = z
```

ARB: Inserimento di un nodo

```
ABR-inserisci(T, k)
    z = alloca nodo ARB
    key[z] = k
    y = NIL
    x = T
    WHILE (x != NIL)
      DO y = x
         IF (k < key[x])
           THEN x = sx[x]
           ELSE x = dx[x]
    IF y = NIL THEN
         T = z
    ELSE IF k < key[y] THEN
              sx[y] = z
         ELSE
              dx[y] = z
```

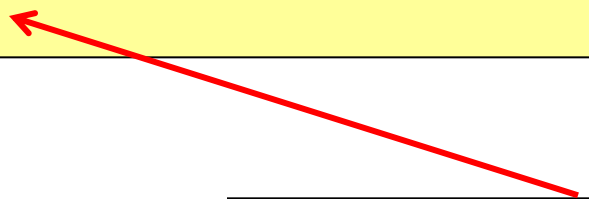
Ricerca posizione del nuovo nodo

(caso II)

(caso I)

ARB: Inserimento di un nodo

```
ABR-insert_ric(T,z)
  IF T != NIL THEN
    IF key[z] < key[T] THEN
        sx[T]= ABR-insert_ric(sx[T],z)
    ELSE
        dx[T]= ABR-insert_ric(dx[T],z)
    return T
  ELSE
      return z
```



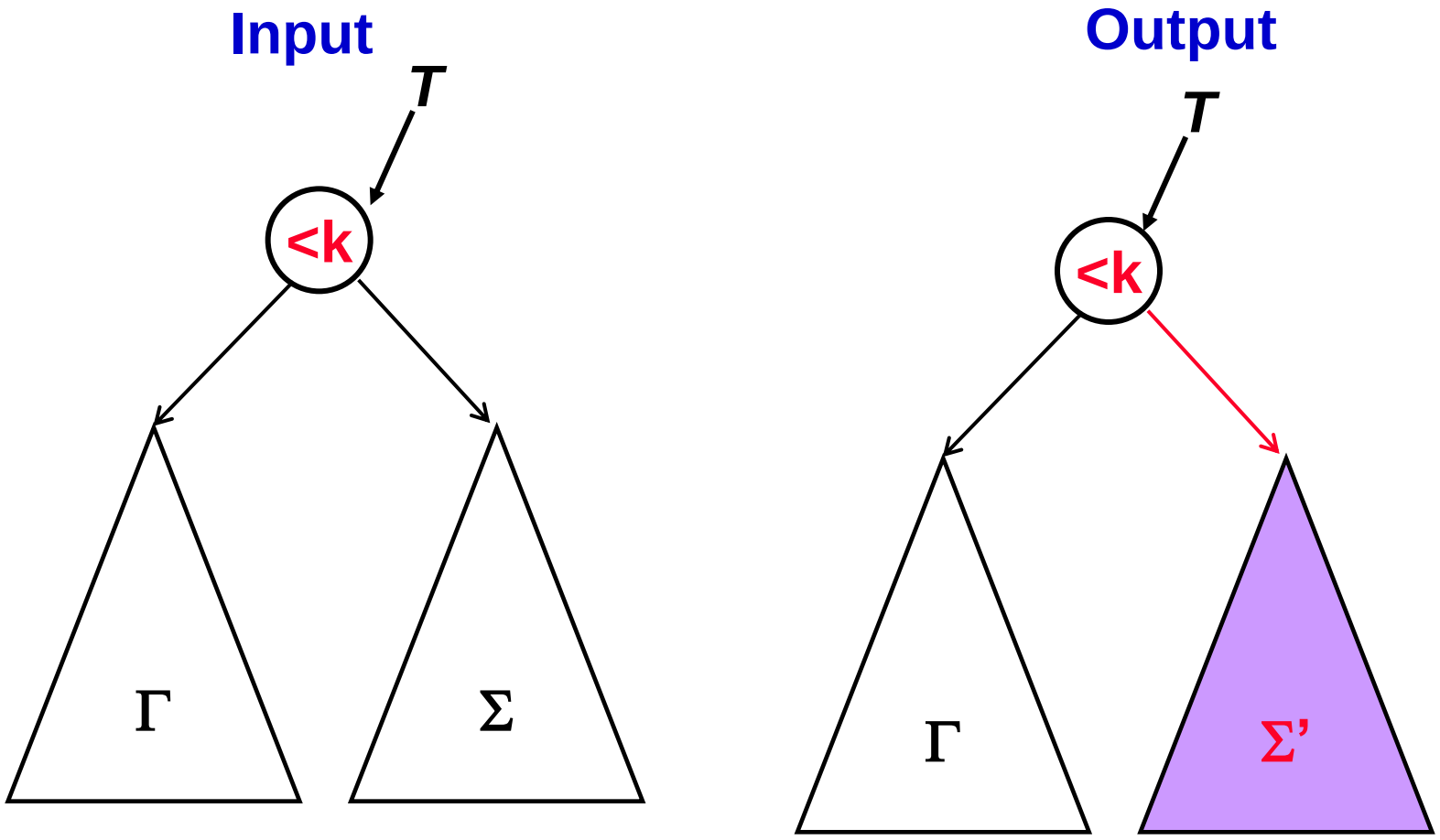
Ricordate che qui z è il nodo che contiene la chiave da inserire

ARB: Inserimento di un nodo

```
ABR-insert_ric(T,k)
  IF T != NIL THEN
    IF k < key[T] THEN
          sx[T]= ABR-insert_ric(sx[T],k)
    ELSE
          dx[T]= ABR-insert_ric(dx[T],k)
    return T
  ELSE
       z = alloca nodo ARB
       key[z] = k
       return z
```

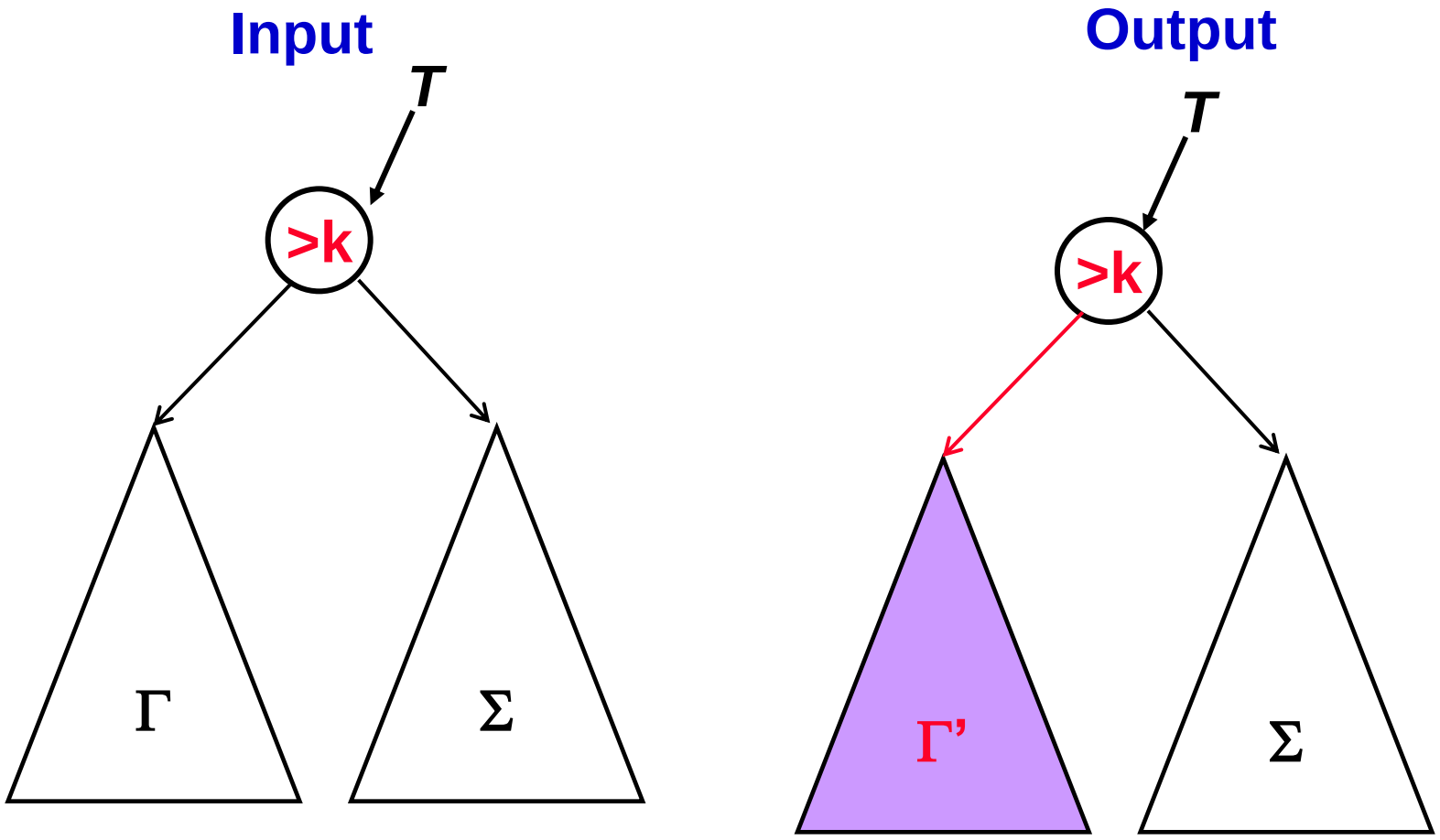
Qui invece k è la chiave da inserire. Si deve quindi allocare il nodo!

Cancellazione ricorsiva



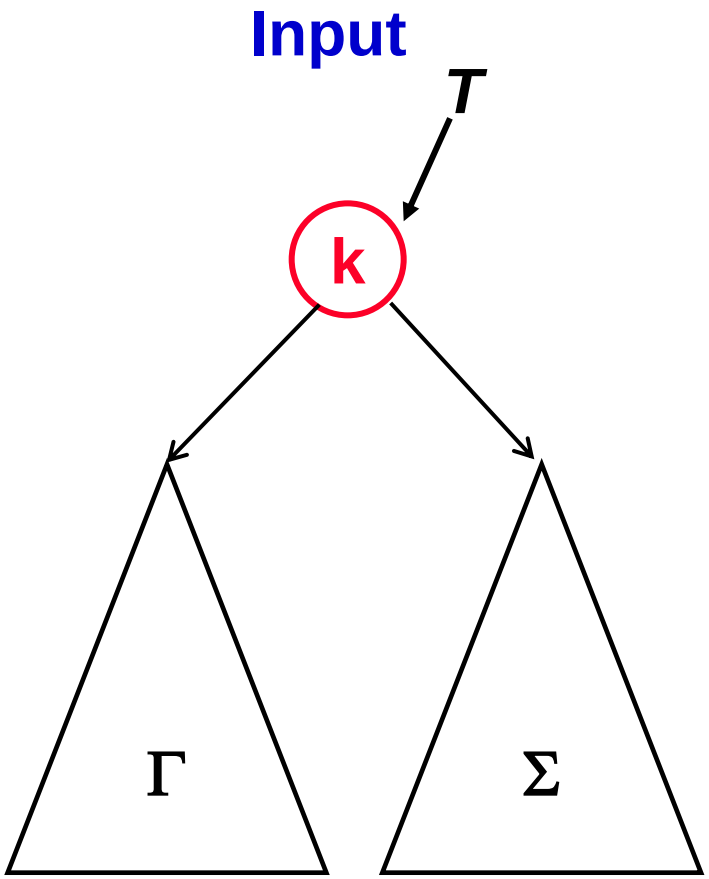
Σ' = cancella(Σ,k)

Cancellazione ricorsiva



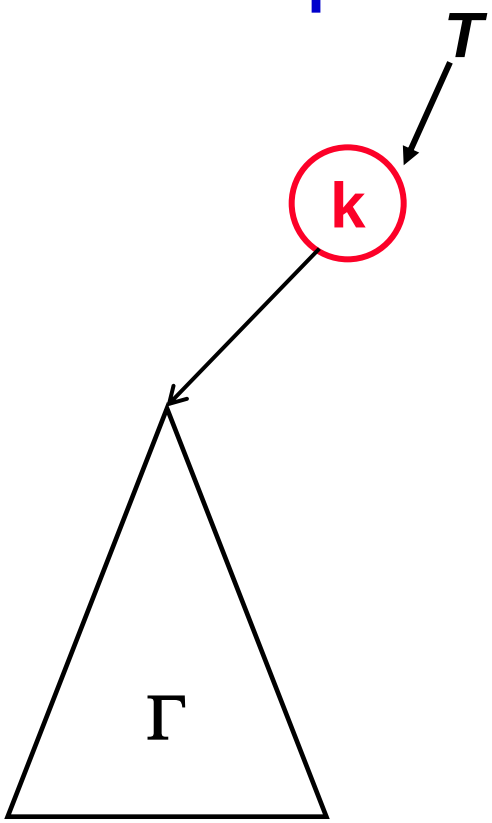
Γ' = cancella(Γ,k)

Cancellazione ricorsiva

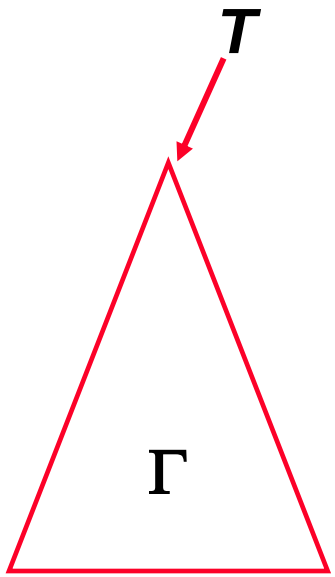


Cancellazione ricorsiva (caso I)

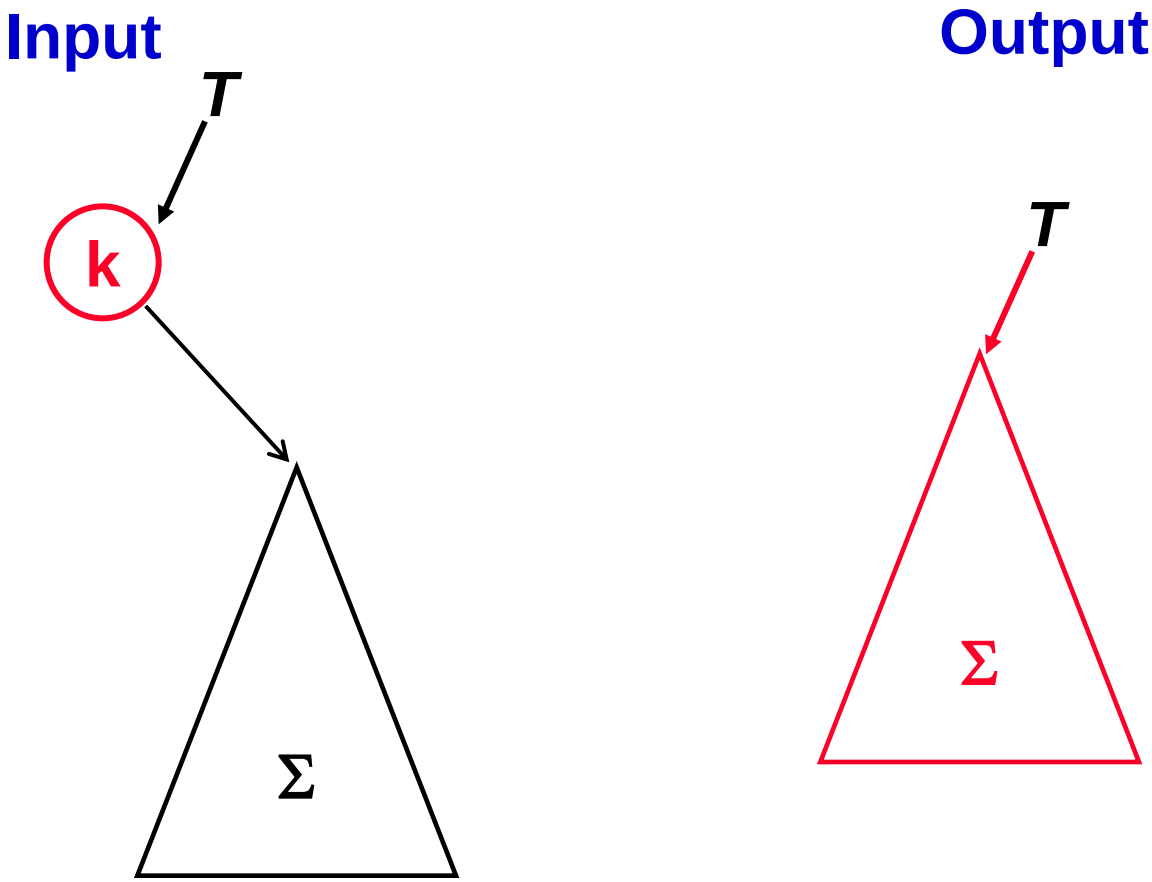
Input



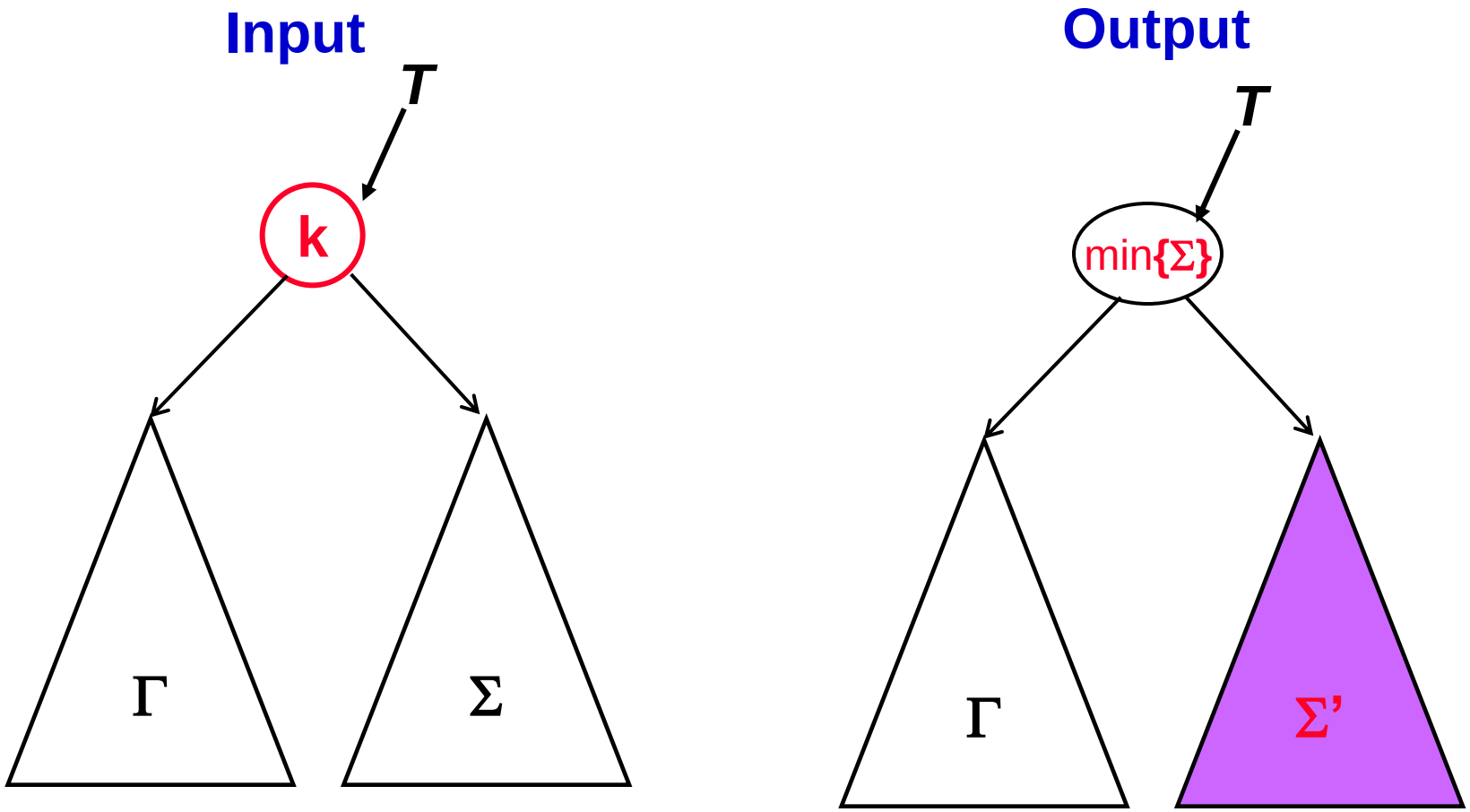
Output



Cancellazione ricorsiva (caso II)



Cancellazione ricorsiva (caso III)



Σ' = stacca-minimo(Σ)

ARB: Cancellazione ricorsiva

```
ABR-Cancella-ric(k,T)
   IF T != NIL THEN
     IF k < key[T] THEN
          sx[T]=ARB-Cancella-ric(k,sx[T])
     ELSE IF k > key[T] THEN
          dx[T]=ARB-Cancella-ric(k,dx[T])
     ELSE /* k = key[T] */
         nodo = T
         IF dx[nodo] = NIL THEN
              T = sx[nodo]
         ELSE IF sx[nodo] = NIL THEN
              T = dx[nodo]
         ELSE
            nodo = Stacca-min(dx[T],T)
            "copia nodo in T"
        dealloca(nodo)
   return T
```

casi I e II

caso III

ARB: Cancellazione ricorsova

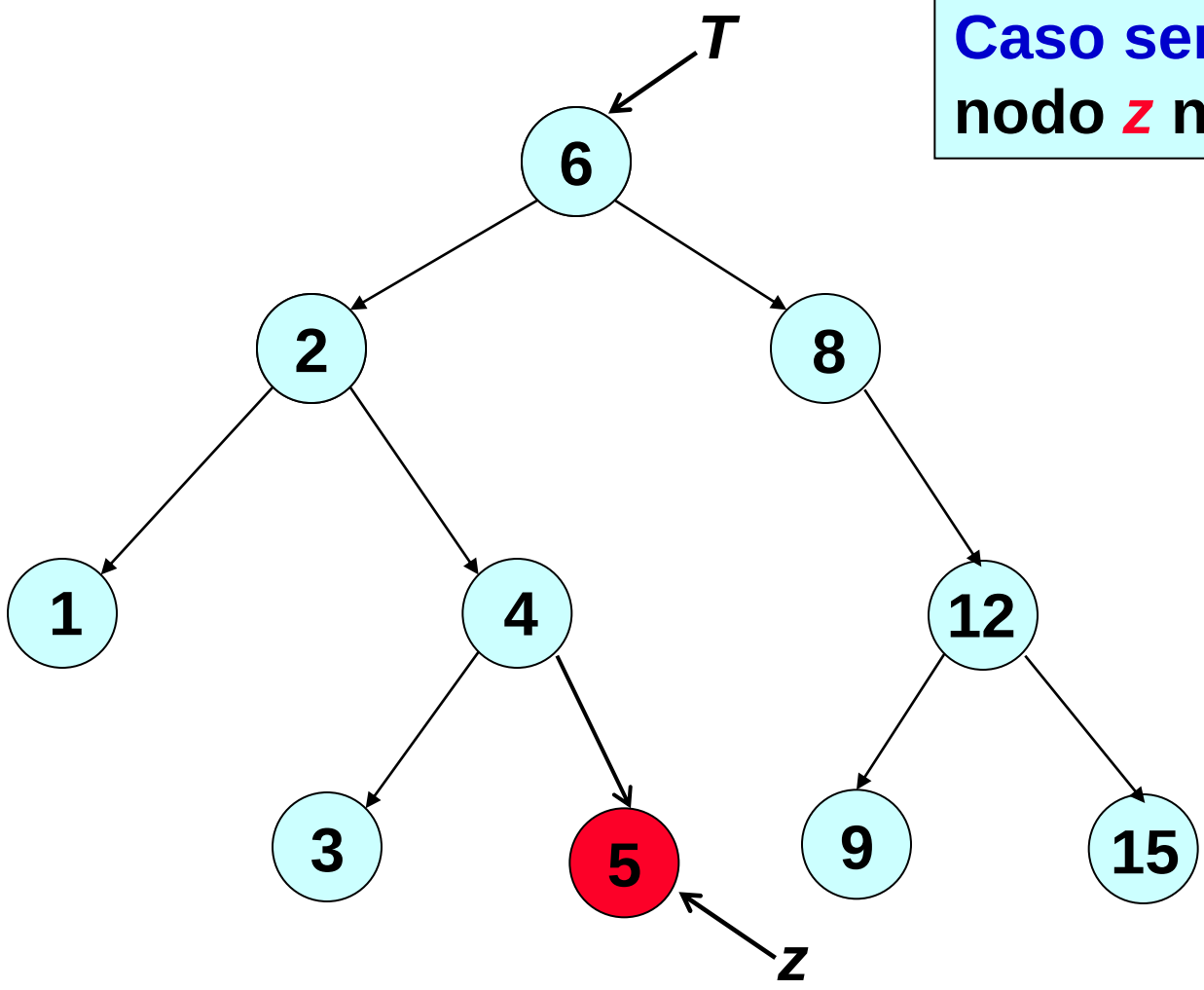
Il parametro P denota il padre di T in ogni chiamata.

```
Stacca-min(T,P)
   IF T ≠ NIL THEN
     IF sx[T] ≠ NIL THEN
          return Stacca-min(sx[T],T)
     ELSE /* successore trovato */
        IF T = sx[P]
           sx[P] = dx[T]
        ELSE /* min è il primo nodo passato */
           dx[P] = dx[T]
   return T
```

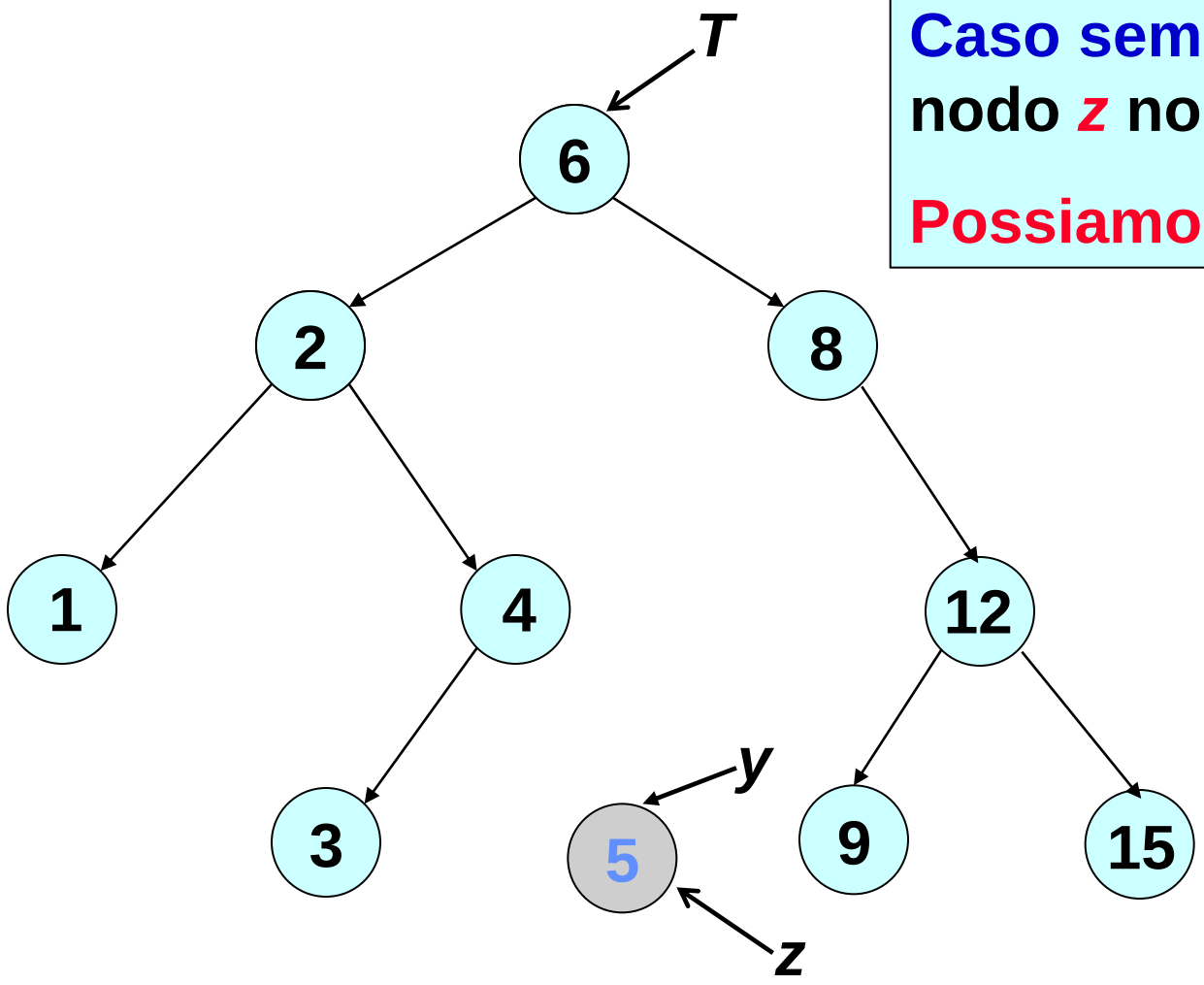
***NOTA**. L'algoritmo **stacca il nodo minimo dell'albero T e ne ritorna il puntatore**. Può anche ritornare **NIL** in caso non esista un minimo (**T** è vuoto). Il valore di ritorno dovrebbe essere quindi verificato dal chiamante prima dell'uso.*
*Nel caso della **cancellazione ricorsiva** però siamo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

ARB: Cancellazione di un nodo (caso I)

Caso semplice: il nodo *z* non ha figli



ARB: Cancellazione di un nodo (caso I)

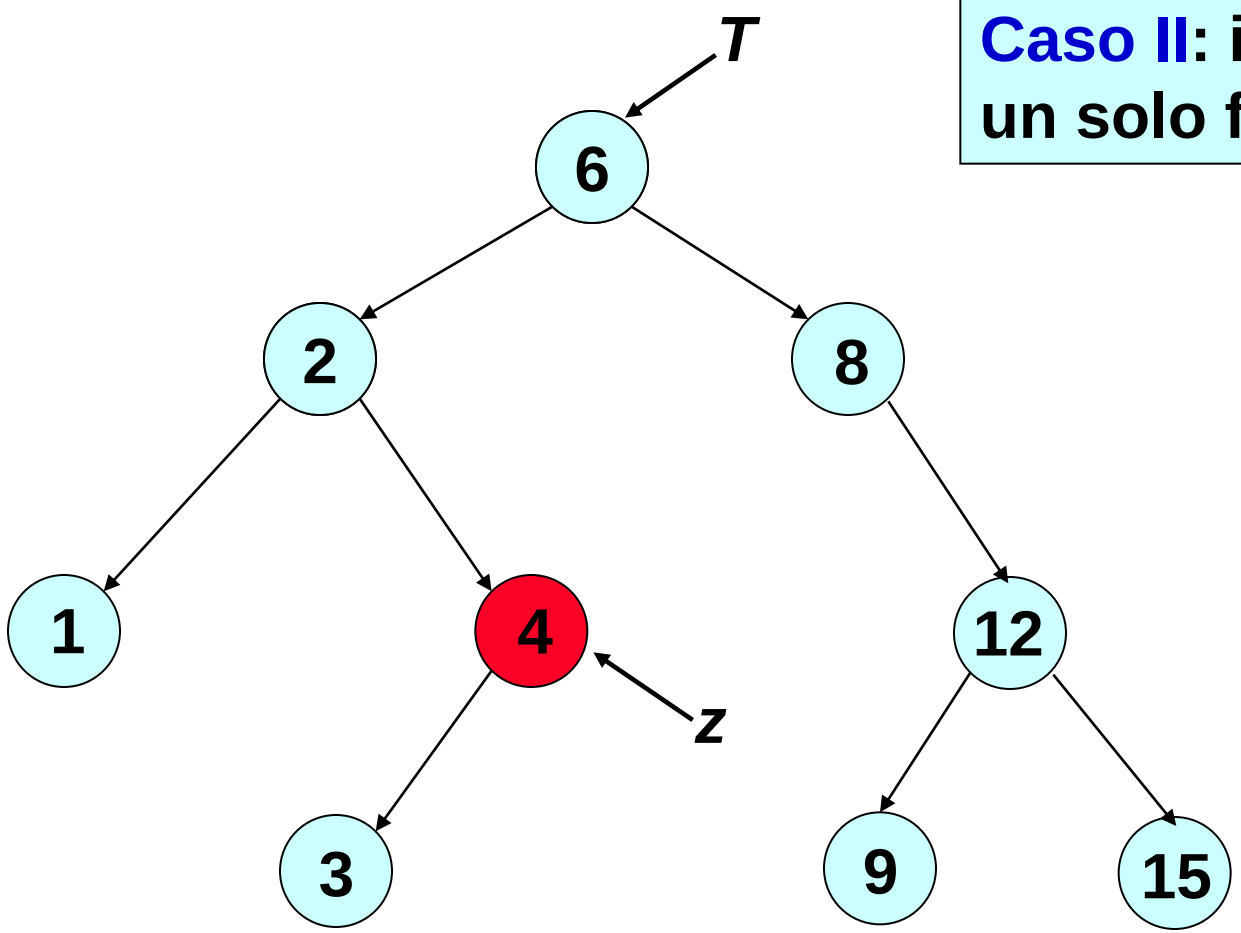


Caso semplice: il nodo *z* non ha figli

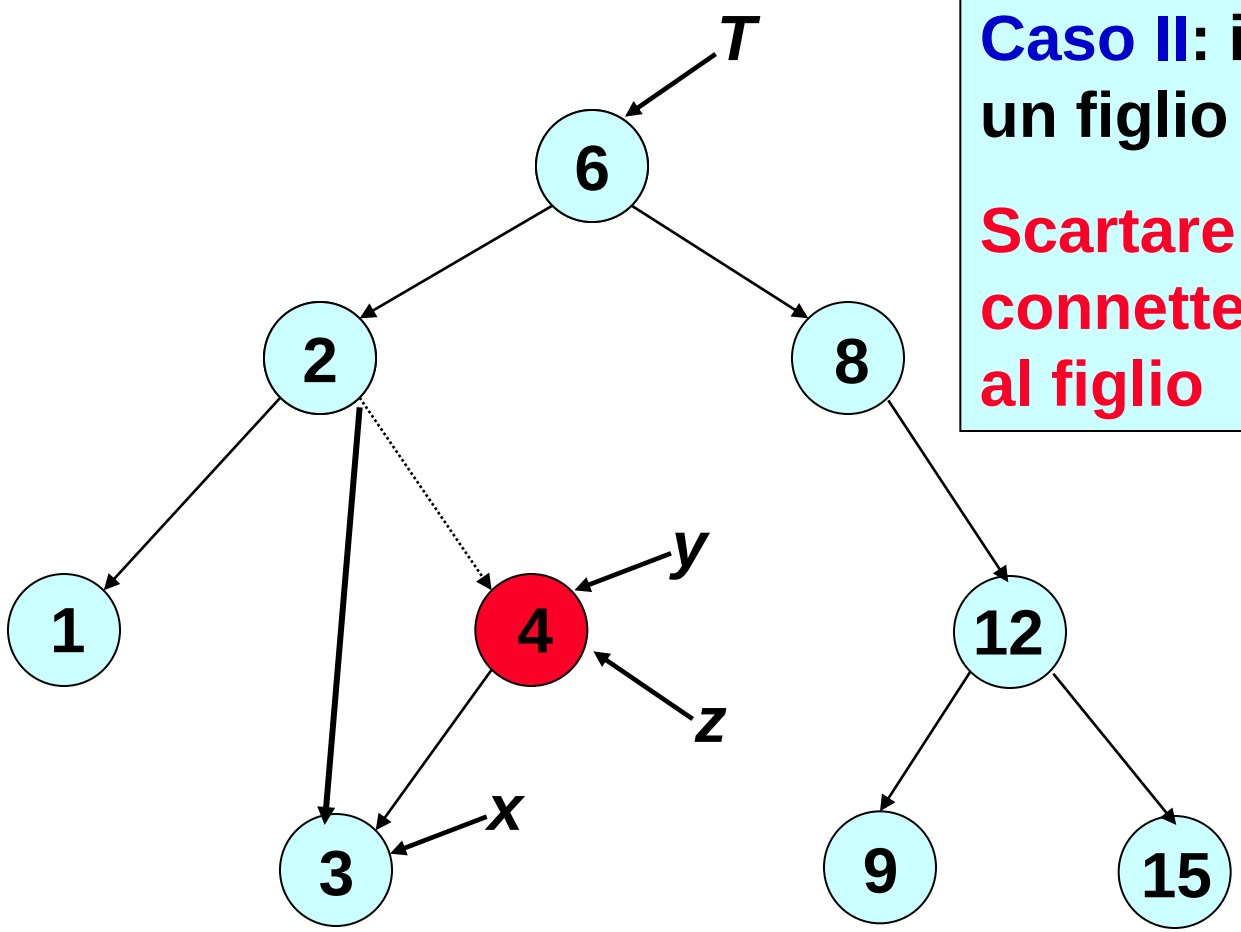
Possiamo eliminarlo

ARB: Cancellazione di un nodo (caso II)

Caso II: il nodo ha un solo figlio



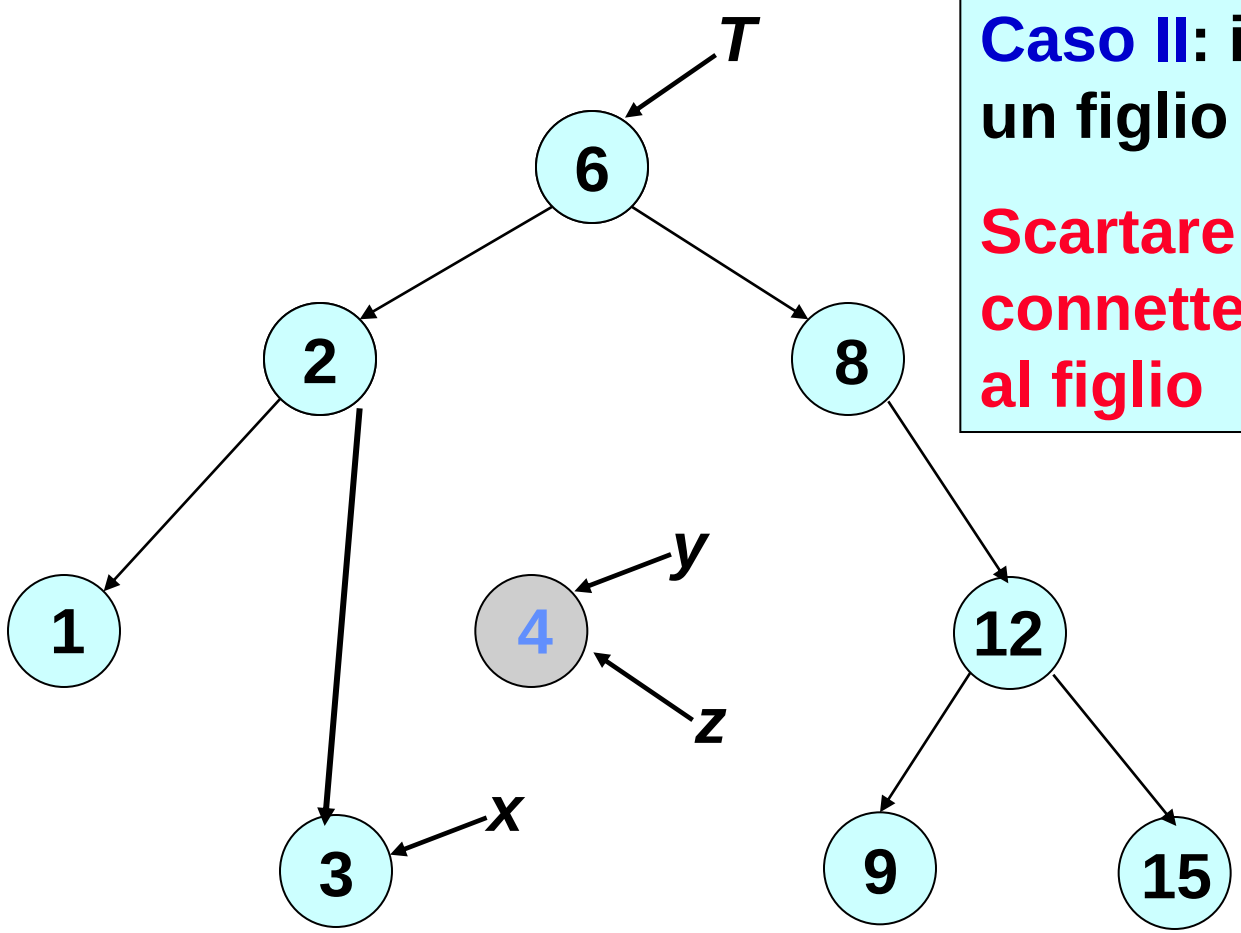
ARB: Cancellazione di un nodo (caso II)



Caso II: il nodo ha un figlio

Scartare il nodo e connettere il padre al figlio

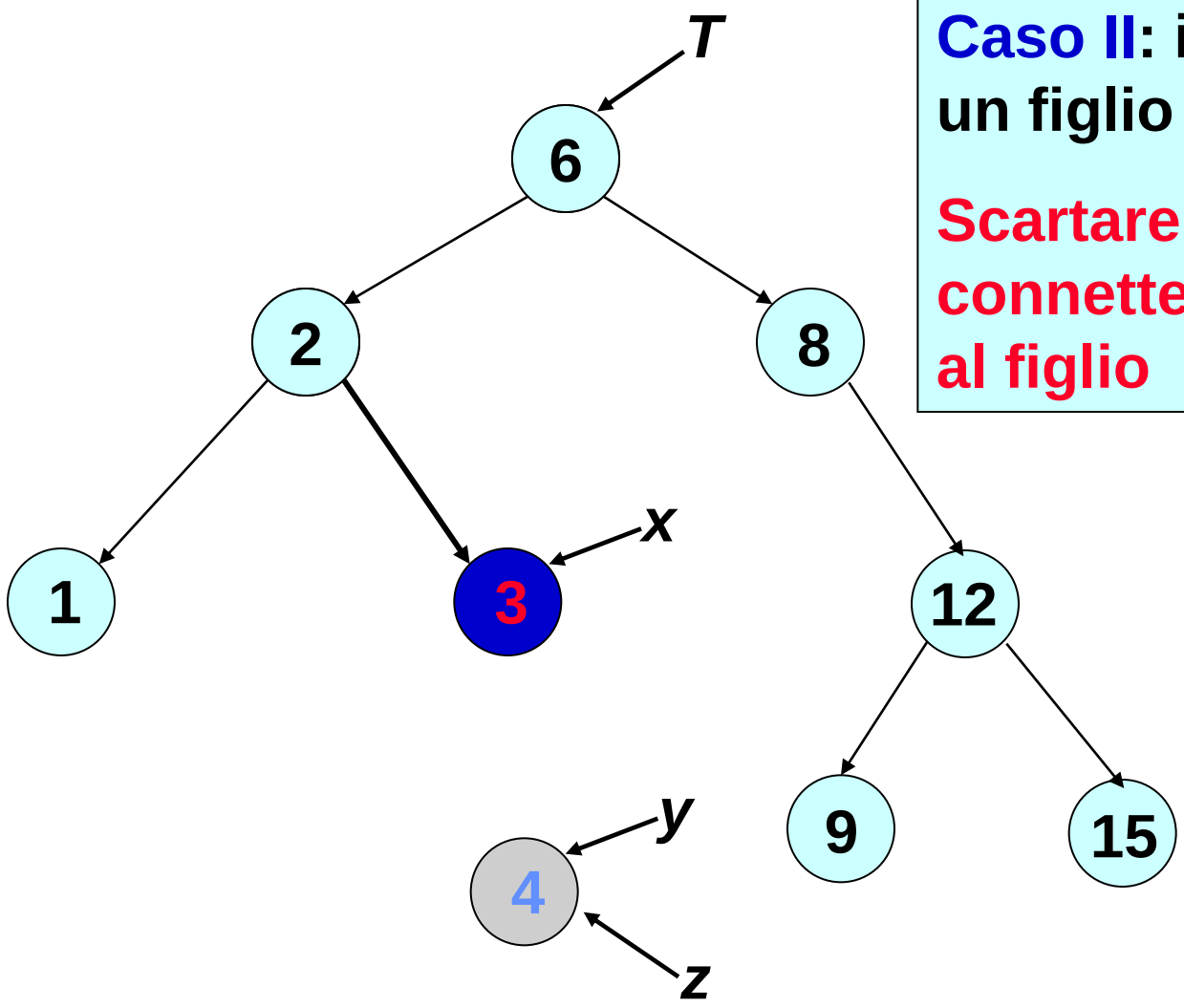
ARB: Cancellazione di un nodo (caso II)



Caso II: il nodo ha un figlio

Scartare il nodo e connettere il padre al figlio

ARB: Cancellazione di un nodo (caso II)

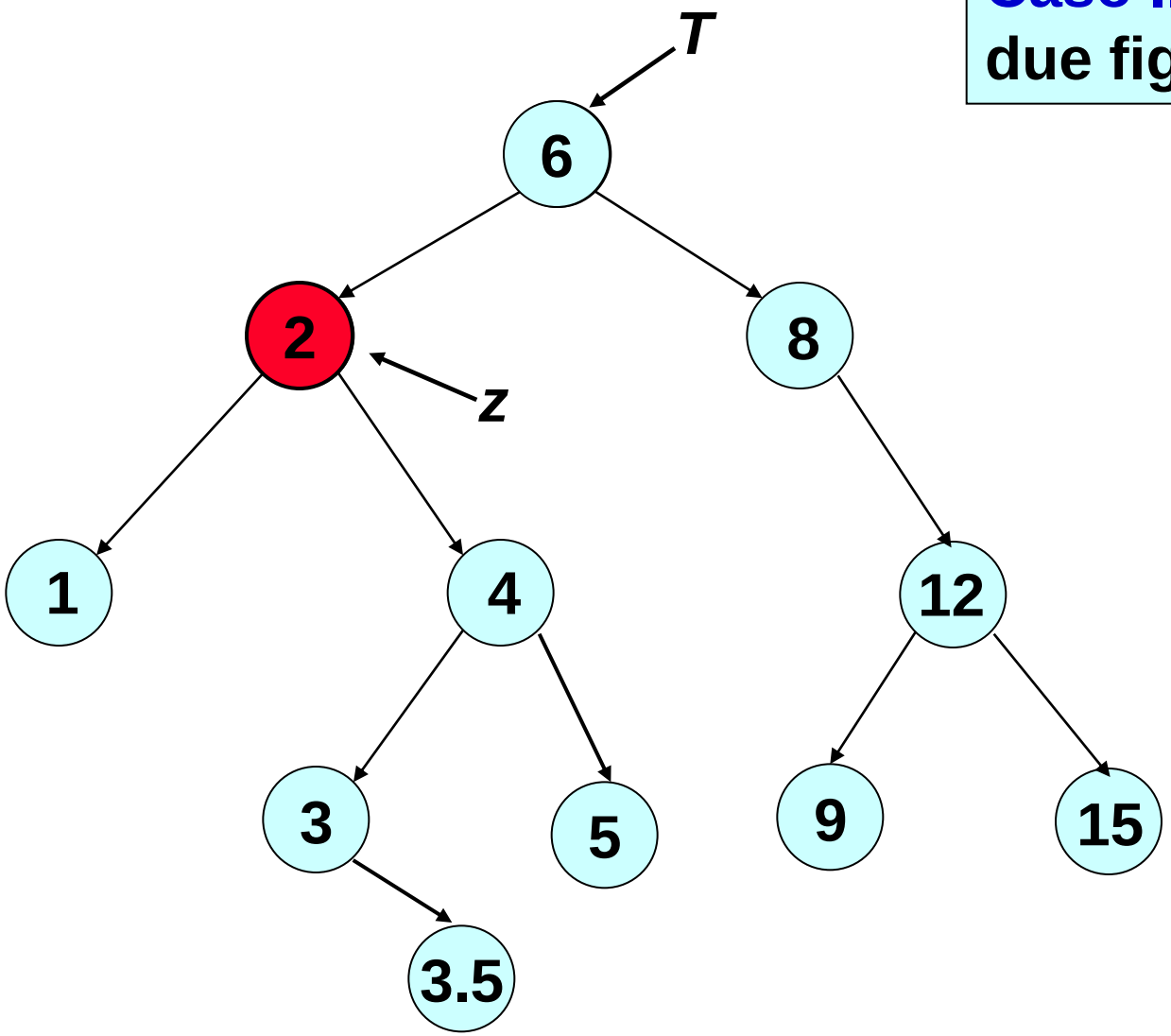


Caso II: il nodo ha un figlio

Scartare il nodo e connettere il padre al figlio

ARB: Cancellazione di un nodo (caso III)

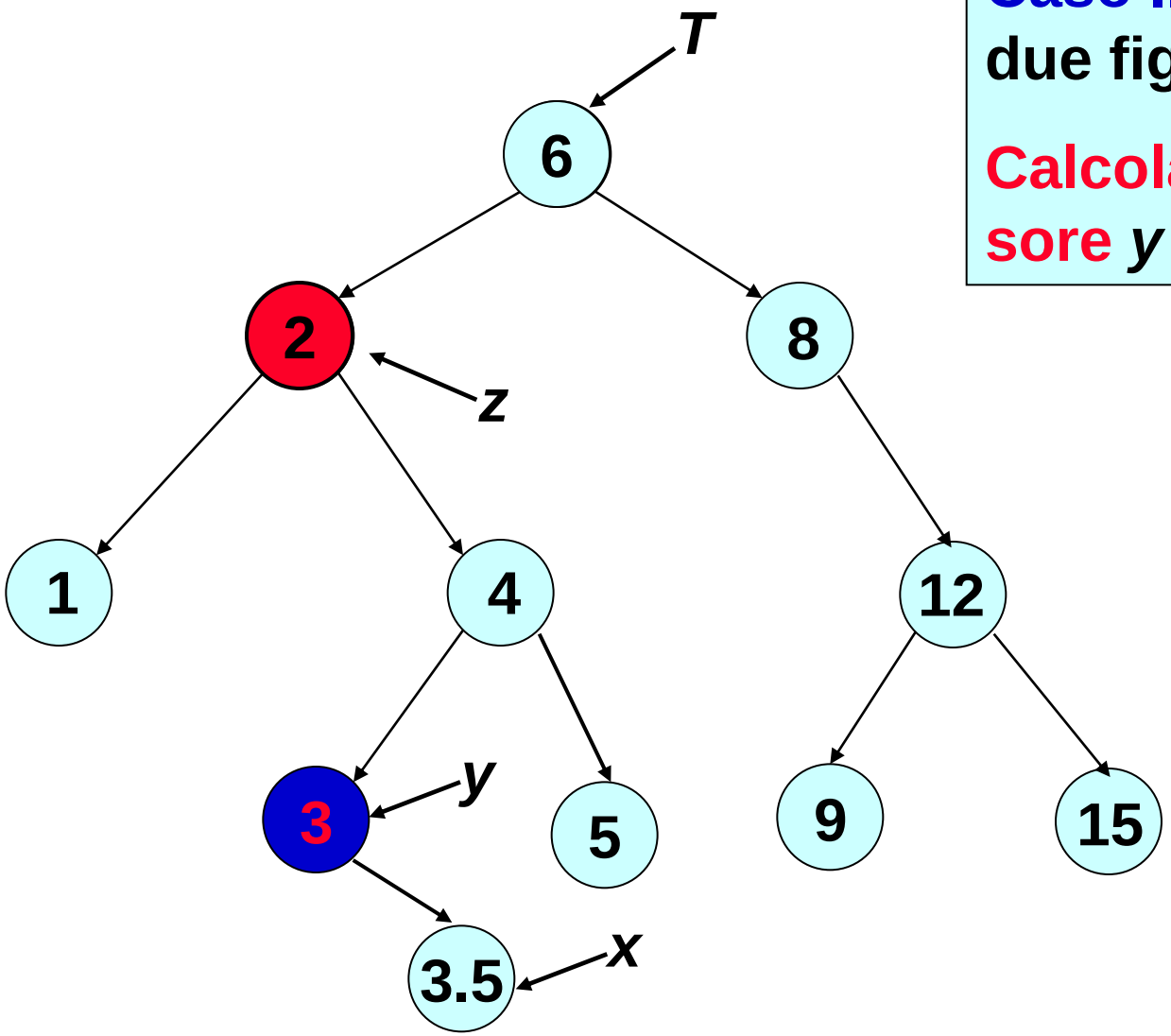
Caso III: il nodo ha due figli



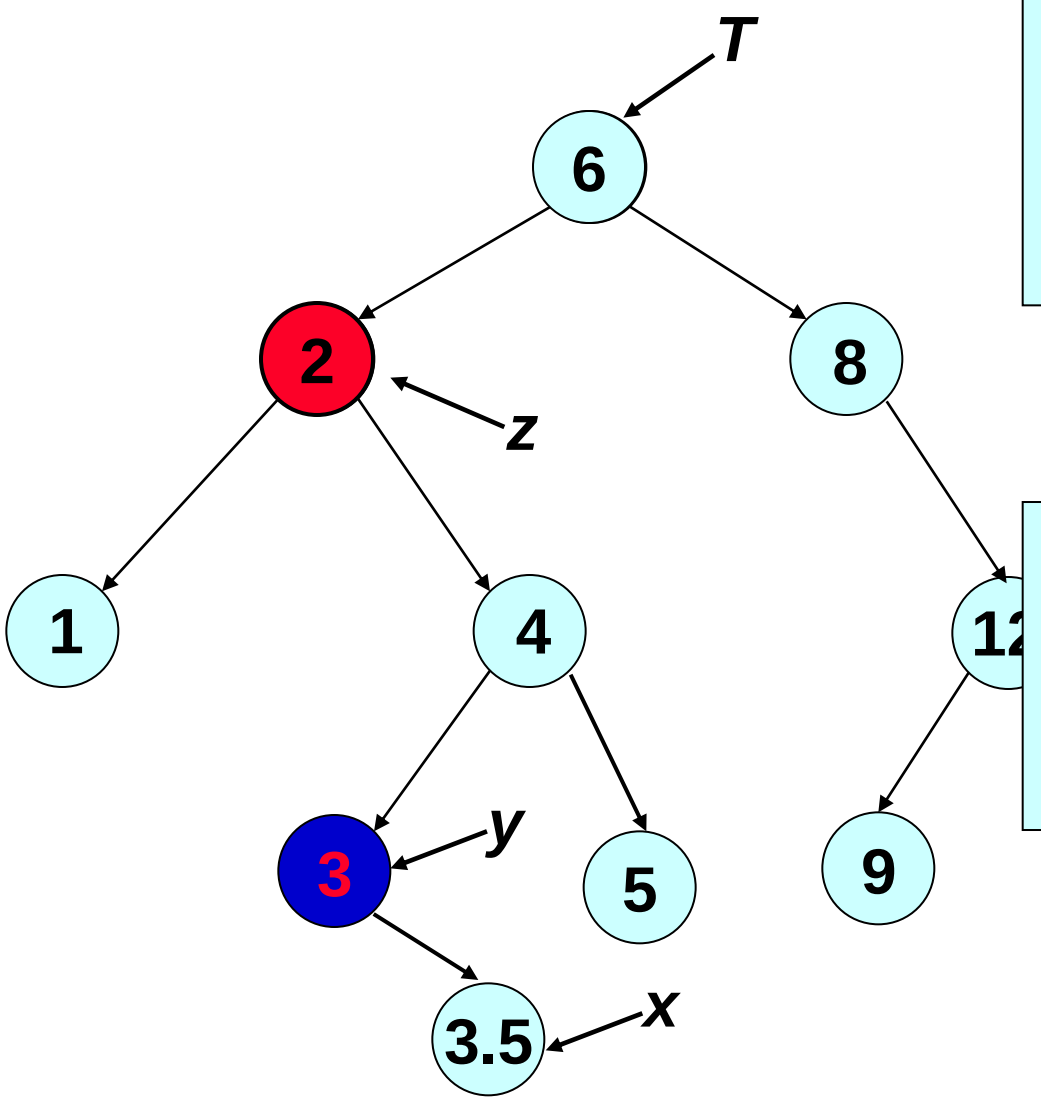
ARB: Cancellazione di un nodo (caso III)

Caso III: il nodo ha due figli

Calcolare il successore *y*



ARB: Cancellazione di un nodo (caso III)

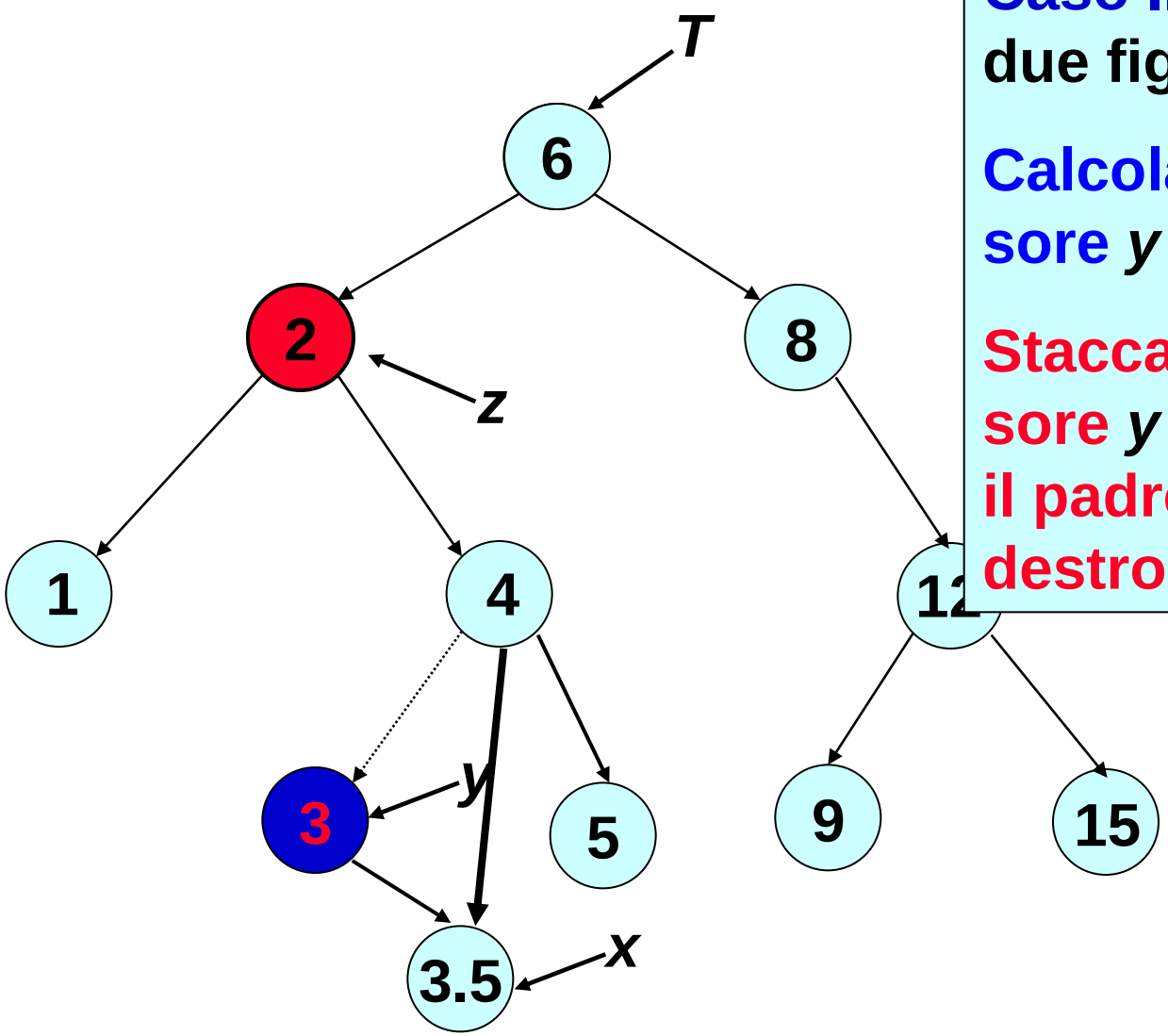


Caso III: il nodo ha due figli

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)

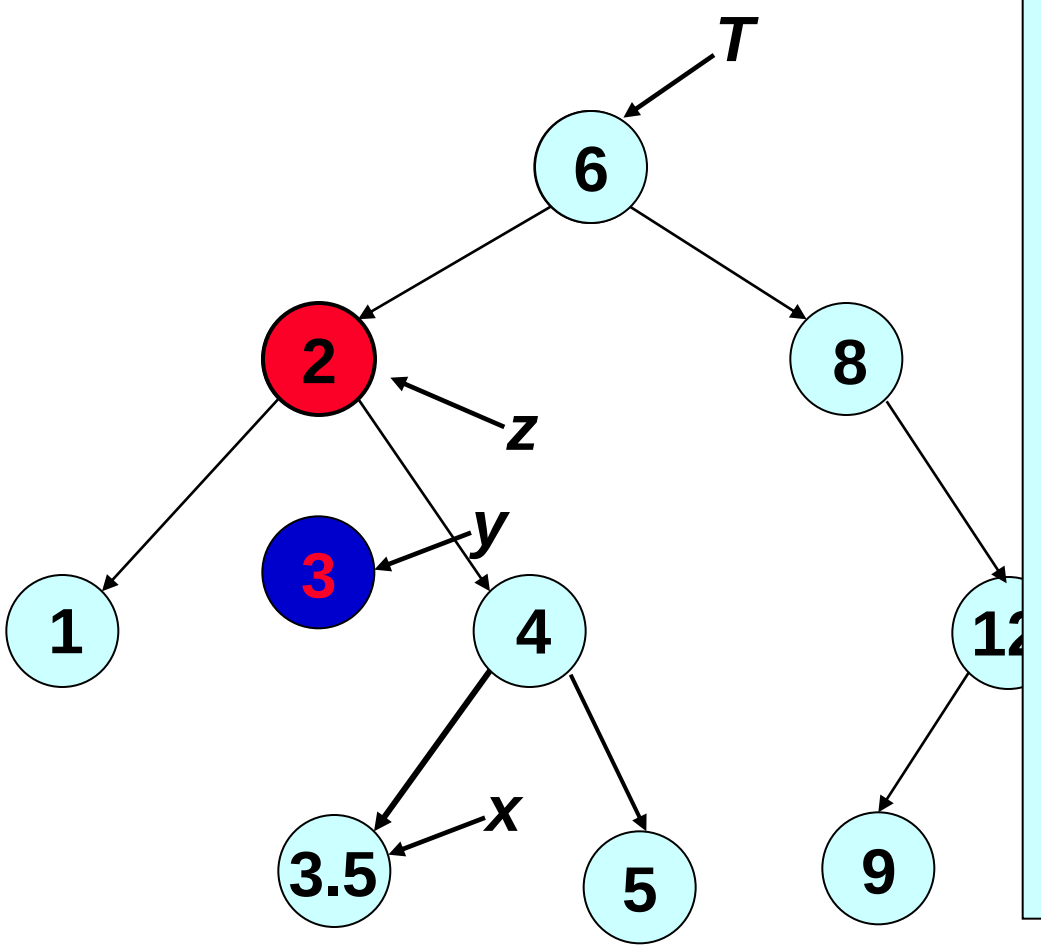


Caso III: il nodo ha due figli

Calcolare il successore *y*

Staccare il successore *y* e connettere il padre al figlio destro

ARB: Cancellazione di un nodo (caso III)



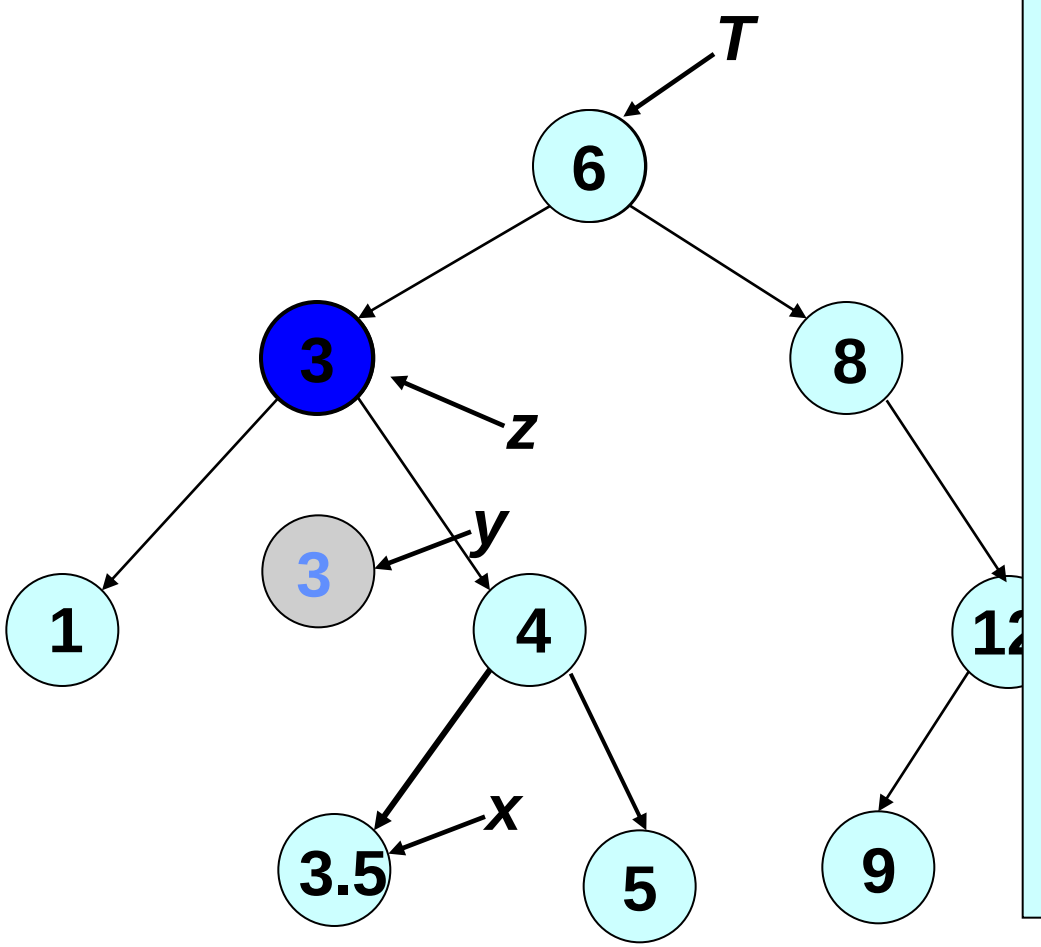
Caso III: il nodo ha due figli

Calcolare il successore *y*

Staccare il successore *y* e connettere il padre al figlio destro

Copia il contenuto del successore nel nodo da cancellare

ARB: Cancellazione di un nodo (caso III)



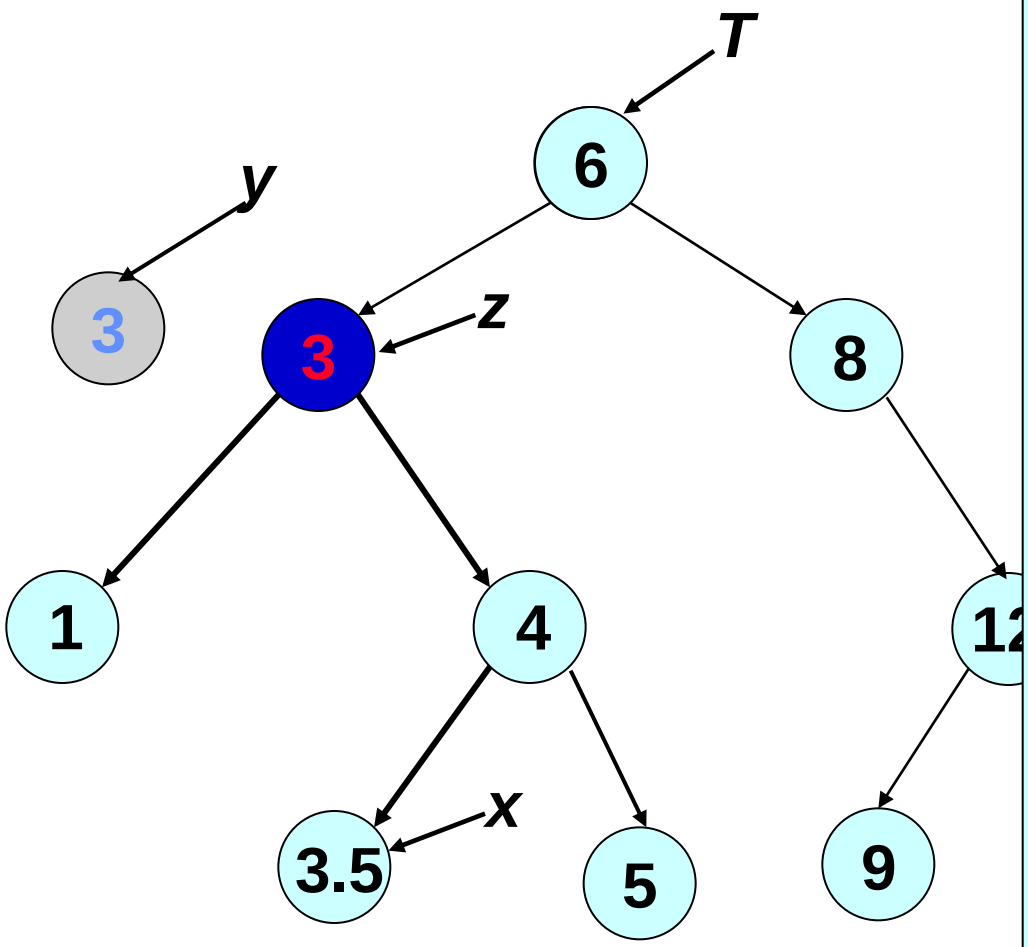
Caso III: il nodo ha due figli

Calcolare il successore *y*

Staccare il successore *y* 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*

Staccare il successore *y* e connettere il padre al figlio destro

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

Deallocare il nodo staccato *y*

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 (sx[z]=NIL OR
          dx[z]=NIL) THEN
          y = z
    ELSE y = ARB-Successore(z)
    IF sx[y] != NIL THEN
          x = sx[y]
    ELSE x = dx[y]
    IF x!=NIL THEN padre[x]=padre[y]
    IF padre[y]=NIL THEN T=x
    ELSE IF y=sx[padre[y]] THEN
                sx[padre[y]]=x
          ELSE dx[padre[y]]=x
    IF y!=z THEN "copia i campi di y in z"
    dealloca y
    return T
```

ARB: Cancellazione di un nodo

```
ABR-Cancella(T,z)
    IF (sx[z]=NIL OR
          dx[z]=NIL) THEN
          y = z
    ELSE y = ARB-Successore(z)
    IF sx[y] != NIL THEN
          x = sx[y]
    ELSE x = dx[y]
    IF x!=NIL THEN padre[x]=padre[y]
    IF padre[y]=NIL THEN T=x
    ELSE IF y=sx[padre[y]] THEN
                sx[padre[y]]=x
          ELSE dx[padre[y]]=x
    IF y!=z THEN "copia i campi di y in z"
    dealloca y
    return T
```

casi I e II

y è il nodo da eliminare

caso III

ARB: Cancellazione di un nodo

```
ABR-Cancella(T,z)
    IF (sx[z]=NIL OR
          dx[z]=NIL) THEN
          y = z
    ELSE y = ARB-Successore(z)
    IF sx[y] != NIL THEN
          x = sx[y]
    ELSE x = dx[y]
    IF x!=NIL THEN padre[x]=padre[y]
    IF padre[y]=NIL THEN T=x
    ELSE IF y=sx[padre[y]] THEN
              sx[padre[y]]=x
         ELSE dx[padre[y]]=x
    IF y!=z THEN "copia i campi di y in z"
    dealloca y
    return T
```

casi I e II

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

y è sostituito da x

caso III

```
ABR-Cancella-iter(T,k)
   p = NIL
   z = T
   WHILE (z != NIL && key[z]!=k) DO
         p = z
         IF (key[z] > k) THEN z = sx[z]
                         ELSE z = dx[z]
   IF (z = NIL) THEN return T  /* nulla da cancellare */
   IF (sx[z] = NIL OR dx[z] = NIL)
     THEN y = z
     ELSE /* z ha 2 figli: si cerca il successore */
          y = dx[z]; p = z
          WHILE (sx[y] ≠ NIL) DO
              p = y
              y = sx[y]
   IF (sx[y] ≠ NIL) THEN x = sx[y]
                    ELSE x = dx[y]
   IF (p = NIL) THEN T = x /* si sta cancellando la radice */
   ELSE IF (y = sx[p]) THEN sx[p]=x
                       ELSE dx[p]=x
   IF (y != z) THEN /* z ha due figli */
       "copia i campi di y in z"
   dealloca y
   return T
```

y è il nodo da eliminare
p è il padre di y
x è il sostituto di y in T

```
ABR-Cancella-iter(T,k)
    p = NIL
    z = T
    WHILE (z != NIL && key[z]!=k) DO
          p = z
          IF (key[z] > k) THEN z = sx[z]
                          ELSE z = dx[z]
    IF (z = NIL) THEN return T  /* nulla da cancellare */
    IF (sx[z] = NIL OR dx[z] = NIL)
      THEN y = z
      ELSE /* z ha 2 figli: si cerca il successore */
           y = dx[z]; p = z
           WHILE (sx[y] ≠ NIL) DO
              p = y
              y = sx[y]
    IF (sx[y] ≠ NIL) THEN x = sx[y]
                     ELSE x = dx[y]
    IF (p = NIL) THEN T = x /* si sta canc
    ELSE IF (y = sx[p]) THEN sx[p]=x
                        ELSE dx[p]=x
    IF (y != z) THEN /* z ha due figli */
        "copia i campi di y in z"
    dealloca y
    return T
```

y è il nodo da eliminare
p è il padre di *y*
x è il sostituto di *y* in *T*

Ricerca di k

Casi I e II

Ricerca successore
Caso III

Distacco nodo y da eliminare e aggior-namento del padre

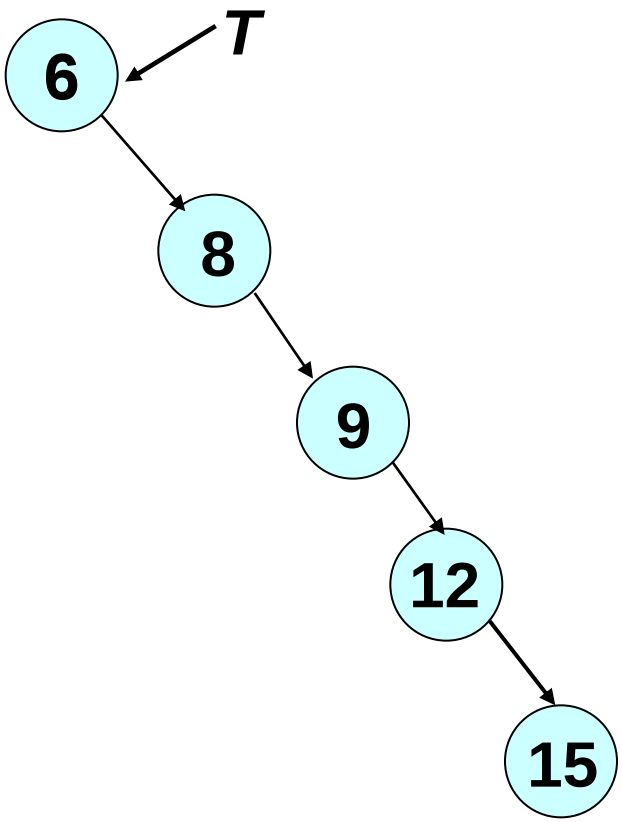
Copia successore
Caso III

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*

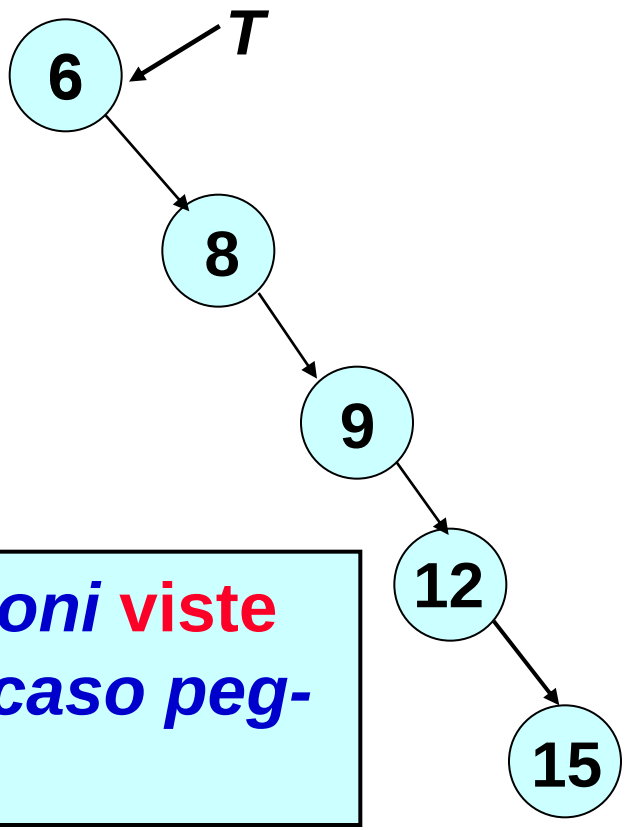
Costo delle operazioni su ABR

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



Costo delle operazioni su ABR

L'algortimo di inserimento NON garantisce che l'albero risultante sia bilaciato. 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 (tutte abbiano *uguale probabilità* di presentarsi).**
- **Allora la probabilità che una chiave *i* sia la radice dell'albero è 1/*n*;**
- **Assumiamo, inoltre, che la probabilità che una chiave *i venga cercata sia* 1/*n*.**

$$a(n)=\frac{1}{n}\sum_{i=1}^{n} p_i$$

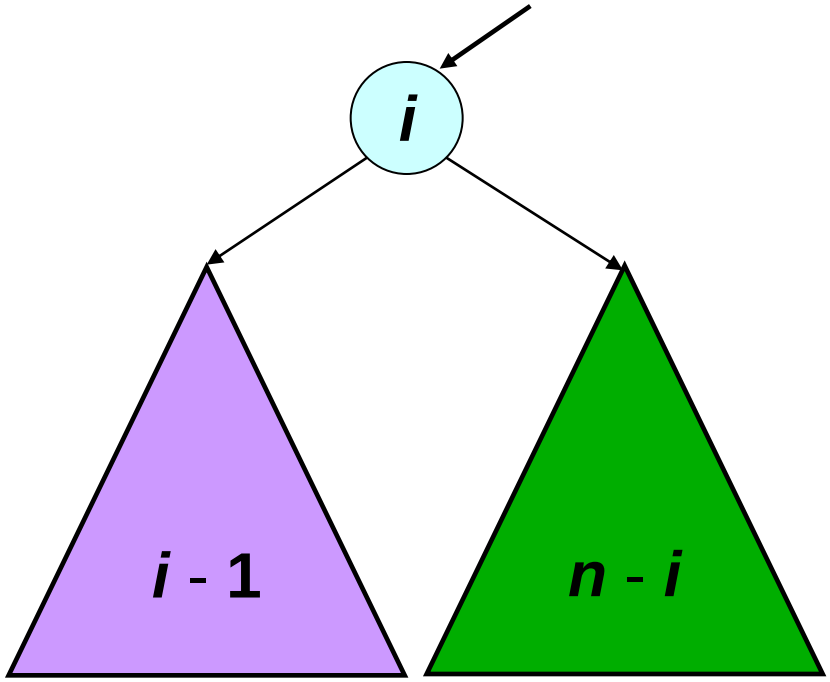
dove p_i è la lunghezza media (su tutti i possibili ABR) del percorso al nodo *i*

Costo delle operazioni su ABR

Se la chiave *i* è la radice dell'albero, allora

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

$$a(n) = \frac{1}{n} \sum_{i=1}^{n} p_i$$

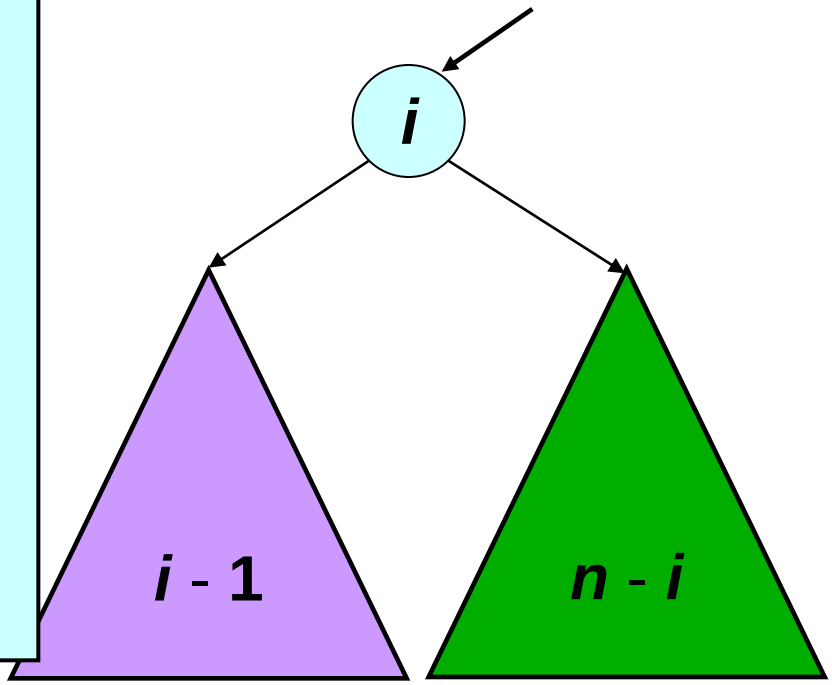


Costo delle operazioni su ABR

Se la chiave *i* è la radice dell'albero, allora

- **il sottoalbero sinistro avrà *i* - 1 nodi e**
- **il sottoalbero destro avrà *n* - *i* nodi**
- **ciascuno degli *i* - 1 nodi a sinistra hanno lunghezza media del percorso *a(i-1)+1***
- **la radice ha lunghezza del percorso pari ad 1**
- **ciascuno degli *n* - *i* nodi a sinistra hanno lunghezza media del percorso *a(n-i)+1***

$$a(n)=\frac{1}{n}\sum_{i=1}^{n} p_i$$



Costo delle operazioni su ABR

$a^i(n)$ sia la lunghezza media del percorso di ricerca con *n* chiavi quando la radice è la chiave *i*

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

$a(i-1)$ è la lunghezza media del percorso di ricerca con *i-1* chiavi

$a(n-i)$ è la lunghezza media del percorso di ricerca con *n-i* chiavi

Costo delle operazioni su ABR

$a^i(n)$ sia la lunghezza media del percorso di ricerca con n chiavi quando la radice è la chiave i

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

allora

$$a(n) = \frac{1}{n}\sum_{i=1}^{n} a^i(n)$$

$a(n)$ è la media degli $a^i(n)$, dove ciascun $a^i(n)$ ha probabilità $1/n$, cioè la probabilità che proprio la chiave i sia la radice dell'albero.

Costo delle operazioni su ABR

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

allora $a(n) = \frac{1}{n}\sum_{i=1}^{n} a^i(n) =$

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

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

Costo delle operazioni su ABR

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

$$= 1 + \frac{1}{n^2}\sum_{i=1}^{n}\left[a(i-1)\cdot(i-1) + a(n-i)\cdot(n-i)\right]$$

$$= 1 + \frac{2}{n^2}\sum_{i=1}^{n}\left[a(i-1)\cdot(i-1)\right]$$

$$= 1 + \frac{2}{n^2}\sum_{i=0}^{n-1} i\,a(i)$$

Costo delle operazioni su ABR

$$a(n) = 1 + \frac{2}{n^2} \sum_{i=0}^{n-1} i \cdot a(i)$$

$$= 1 + \frac{2}{n^2}(n-1) \cdot a(n-1) + \frac{2}{n^2} \sum_{i=0}^{n-2} i \cdot a(i)$$

Costo delle operazioni su ABR

$$a(n) = 1 + \frac{2}{n^2} \sum_{i=0}^{n-1} i \cdot a(i)$$

$$= 1 + \frac{2}{n^2}(n-1) \cdot a(n-1) + \frac{2}{n^2} \sum_{i=0}^{n-2} i \cdot a(i)$$

$$a(n-1) = 1 + \frac{2}{(n-1)^2} \sum_{i=0}^{n-2} i \cdot a(i)$$

Costo delle operazioni su ABR

$$a(n) = 1 + \frac{2}{n^2} \sum_{i=0}^{n-1} i \cdot a(i)$$

$$= 1 + \frac{2}{n^2}(n-1) \cdot a(n-1) + \frac{2}{n^2} \sum_{i=0}^{n-2} i \cdot a(i)$$

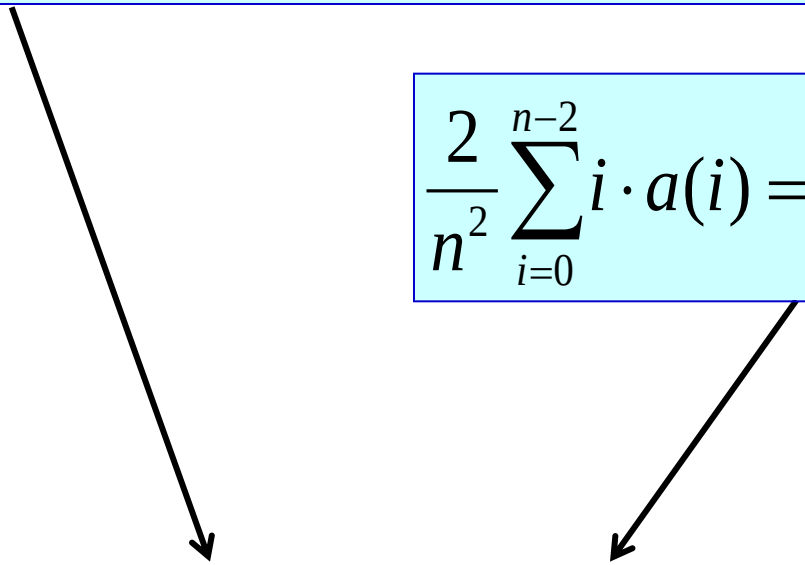
$$\frac{2}{n^2} \sum_{i=0}^{n-2} i \cdot a(i) = \frac{(n-1)^2}{n^2}\left(a(n-1) - 1\right)$$

$$a(n-1) = 1 + \frac{2}{(n-1)^2} \sum_{i=0}^{n-2} i \cdot a(i)$$

Costo delle operazioni su ABR

$$a(n) = 1 + \frac{2}{n^2}(n-1)\cdot a(n-1) + \frac{2}{n^2}\sum_{i=0}^{n-2} i \cdot a(i)$$

$$\frac{2}{n^2}\sum_{i=0}^{n-2} i \cdot a(i) = \frac{(n-1)^2}{n^2}\left(a(n-1)-1\right)$$



$$a(n) = \frac{1}{n^2}\left[(n^2-1)\cdot a(n-1) + 2n - 1\right]$$

Costo delle operazioni su ABR

$$a(n) = \frac{1}{n^2}\left[(n^2 - 1)\cdot a(n-1) + 2n - 1\right]$$

Dimostrare per induzione

$$a(n) = 2\frac{n+1}{n}H(n) - 3$$

$$H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

Funzione armonica

Costo delle operazioni su ABR

$$a(n) = 2\frac{n+1}{n}H(n) - 3$$

Dimostrare per induzione

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

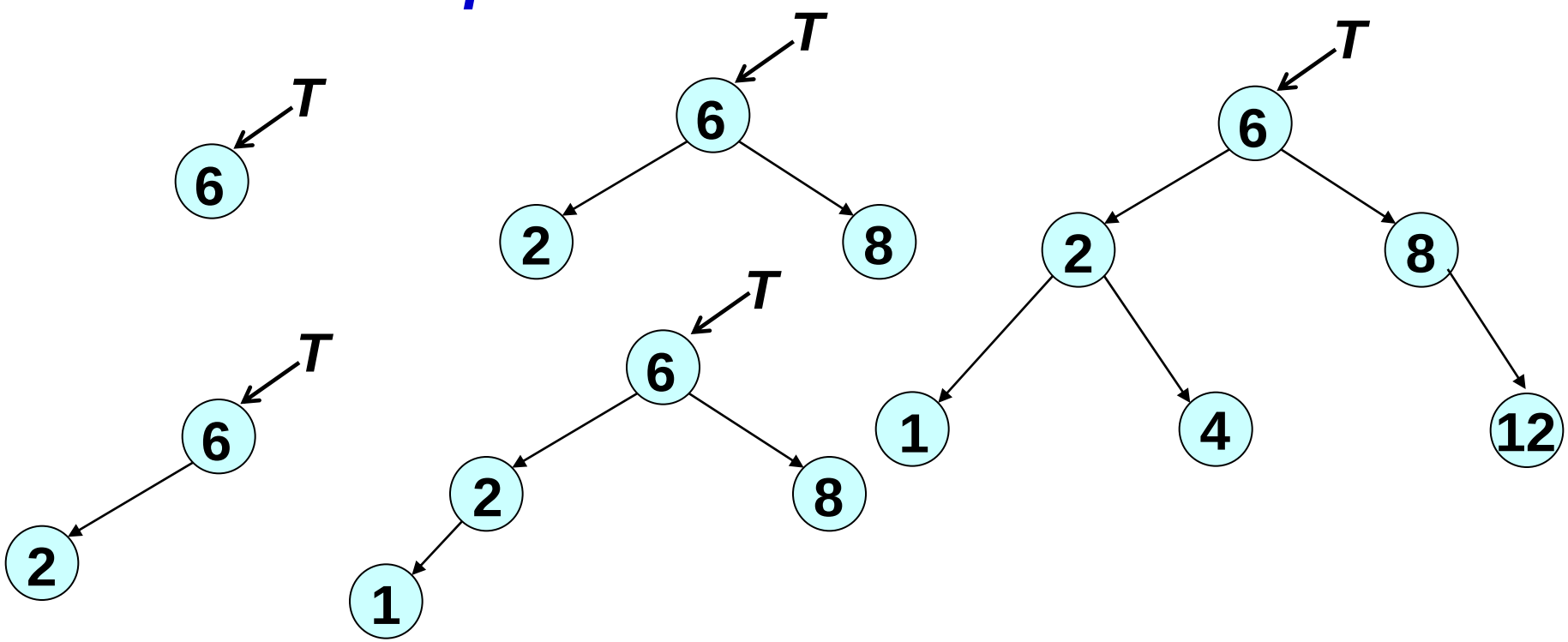
Formula di Eulero

$$H(n) = \gamma + \ln n + \frac{1}{2n} + \frac{1}{12n^2} + \ldots$$

dove $\gamma \approx 0.577$

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**

La *lunghezza media a'(n) del percorso* in un *albero perfettamente bilanciato (APB)* con *n* nodi è approssimativamente

$$a'(n) = \log n - 1$$

Confronto tra ABR e APB

Il *rapporto* tra la *lunghezza media* $a(n)$ *del percorso* in un *albero di ricerca* e la *lunghezza media* $a'(n)$ nell'*albero perfettamente bilanciato* è (per n sufficientemente grande) è approssimativamente

$$\frac{a(n)}{a'(n)} = \frac{2\ln n - c}{\log n - 1} \cong \frac{2\ln n}{\log n} = 2\ln 2 \cong 1{,}386$$

(trascurando i termini costanti)

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*.**