

Algoritmi e Strutture Dati

Alberi Bilanciati: Alberi Red-Black

Alberi bilanciati di ricerca

- Gli *alberi binari di ricerca* sono semplici da gestire (inserimenti e cancellazioni facili da implementare) ma hanno prestazioni poco prevedibili e potenzialmente basse
- Gli *alberi perfettamente bilanciati* hanno prestazioni ottimali ($\log n$ garantito) ma inserimenti e cancellazioni complesse (ri-bilanciamenti)
- **Alberi AVL** (*Adelson-Velskii e Landis*): *alberi quasi bilanciati*. Buone prestazioni e gestione relativamente semplice.

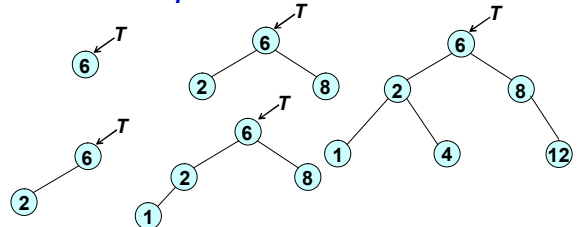
Alberi AVL: definizione

Definizione: Un albero binario di ricerca è un **Albero AVL** se per ogni nodo x :

- l'*altezza del sottoalbero sinistro di x* e quella del *sottoalbero destro di x* differiscono al più di uno, e
- *entrambi i sottoalberi* sinistro e destro di x sono *alberi AVL*.

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

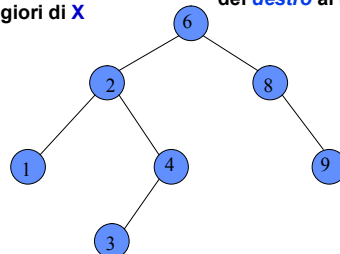
Definizione: Un albero binario si dice **Perfettamente Bilanciato** se, per ogni nodo i , il **numero dei nodi** nel suo **sottoalbero sinistro** e il **numero dei nodi** del suo **sottoalbero destro differiscono al più di 1**

L'**altezza** di un **albero perfettamente bilanciato (APB)** con n nodi è $h = \log_2 n$

Alberi AVL: alberi binari bilanciati

Proprietà degli ABR
Per ogni nodo X , i **nodi del sottoalbero sinistro** sono **minori del nodo X** , e i **nodi del sottoalbero destro** sono **maggiori di X**

Proprietà AVL
ABR dove per ogni nodo X , l'**altezza del sottoalbero sinistro differisce** da quella del **destro** al massimo di 1.



Alberi AVL: alberi binari bilanciati

Altezza(X) =
 $\max(\text{Altezza}(\text{sinistro}(X)), \text{Altezza}(\text{destra}(X))) + 1$

Proprietà AVL
 ABR dove per ogni nodo X, l'altezza del *suttoalbero sinistro* differisce da quella del *destra* al massimo di 1.

Alberi AVL: alberi binari bilanciati

Altezza(X) =
 $\max(\text{Altezza}(\text{sinistro}(X)), \text{Altezza}(\text{destra}(X))) + 1$
 $\text{Altezza}(\emptyset) = -1$

Proprietà AVL
 ABR dove per ogni nodo X, l'altezza del *suttoalbero sinistro* differisce da quella del *destra* al massimo di 1.

Questo è un albero AVL?

Alberi AVL: alberi binari bilanciati

Altezza(X) =
 $\max(\text{Altezza}(\text{sinistro}(X)), \text{Altezza}(\text{destra}(X))) + 1$
 $\text{Altezza}(\emptyset) = -1$

Proprietà AVL
 ABR dove per ogni nodo X, l'altezza del *suttoalbero sinistro* differisce da quella del *destra* al massimo di 1.

Alberi AVL: alberi binari bilanciati

Altezza(X) =
 $\max(\text{Altezza}(\text{sinistro}(X)), \text{Altezza}(\text{destra}(X))) + 1$

Proprietà AVL
 ABR dove per ogni nodo X, l'altezza del *suttoalbero sinistro* differisce da quella del *destra* al massimo di 1.

Alberi AVL: alberi binari bilanciati

Altezza(X) =
 $\max(\text{Altezza}(\text{sinistro}(X)), \text{Altezza}(\text{destra}(X))) + 1$

Proprietà AVL
 ABR dove per ogni nodo X, l'altezza del *suttoalbero sinistro* differisce da quella del *destra* al massimo di 1.

Alberi AVL: alberi binari bilanciati

Altezza(X) =
 $\max(\text{Altezza}(\text{sinistro}(X)), \text{Altezza}(\text{destra}(X))) + 1$

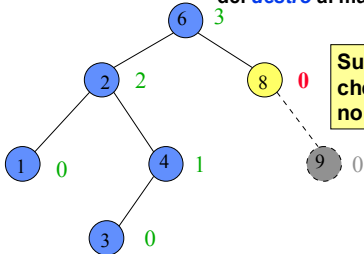
Proprietà AVL
 ABR dove per ogni nodo X, l'altezza del *suttoalbero sinistro* differisce da quella del *destra* al massimo di 1.

Si! Questo è un albero AVL.

Alberi AVL: alberi binari bilanciati

$$\text{Altezza}(X) = \max(\text{Altezza}(\text{sinistro}(X)), \text{Altezza}(\text{destra}(X))) + 1$$

Proprietà AVL
 ABR dove per ogni nodo X , l'altezza del *sottoalbero sinistro* differisce da quella del *destra* al massimo di 1.

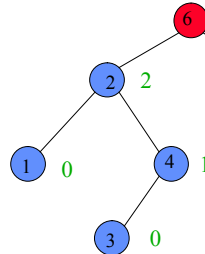


Supponiamo che il nodo 9 non ci sia.

Alberi AVL: alberi binari bilanciati

$$\text{Altezza}(X) = \max(\text{Altezza}(\text{sinistro}(X)), \text{Altezza}(\text{destra}(X))) + 1$$

Proprietà AVL
 ABR dove per ogni nodo X , l'altezza del *sottoalbero sinistro* differisce da quella del *destra* al massimo di 1.



La **Proprietà AVL** non è soddisfatta dal nodo 6.

No! Questo NON è un albero AVL.

Alberi AVL: definizione

Definizione: Un albero binario di ricerca è un **Albero AVL** se per ogni nodo x :

- l'altezza del *sottoalbero sinistro* di x e quella del *sottoalbero destro* di x differiscono al più di uno, e
- entrambi i *sottoalberi* sinistro e destro di x sono *alberi AVL*

Esempi di **alberi AVL minimi** di diverse altezze (cioè con il numero minimo di nodi)

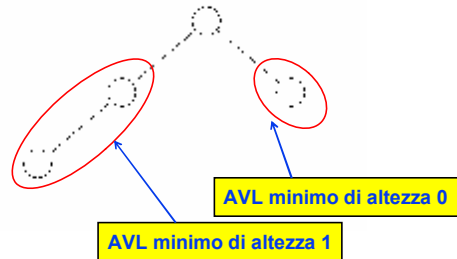
Albero AVL minimo di altezza 1



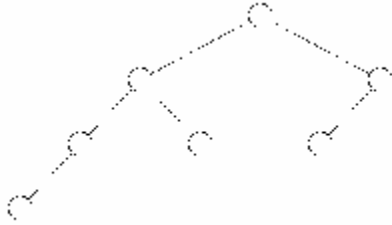
Albero AVL minimo di altezza 2



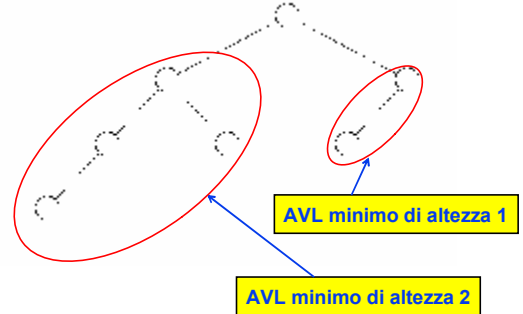
Albero AVL minimo di altezza 2



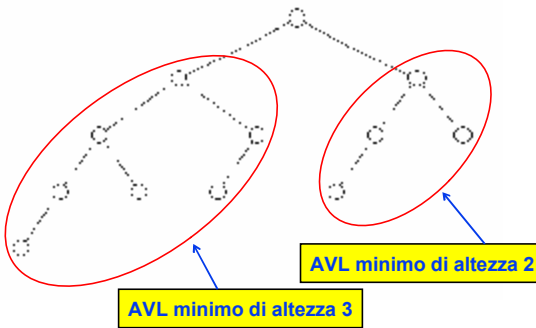
Albero AVL minimo di altezza 3



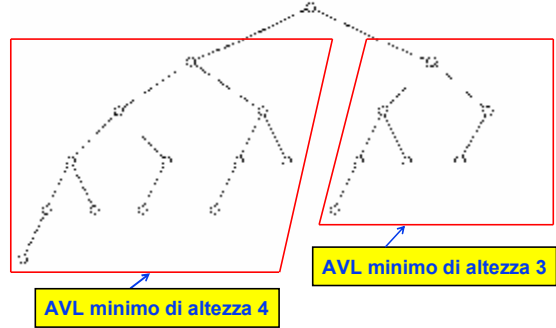
Albero AVL minimo di altezza 3



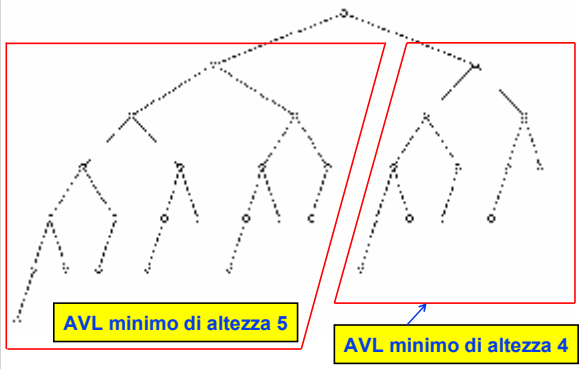
Albero AVL minimo di altezza 4



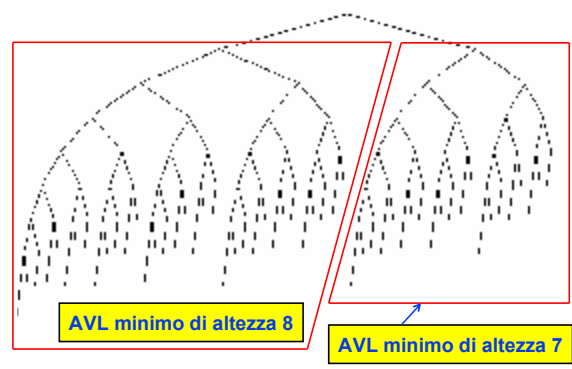
Albero AVL minimo di altezza 5



Albero AVL minimo di altezza 6



Albero AVL minimo di altezza 9



Alberi Red-Black (alberi rossi-neri)

Un **Albero Red-Black (rosso-nero)** è essenzialmente un albero binario di ricerca in cui:

- 1 Le **chiavi** vengono mantenute **solo nei nodi interni** dell'albero
- 2 Le foglie sono costituite da **nodi NIL**, cioè nodi **sentinella** il cui contenuto è **irrelevante** e che evitano di trattare diversamente i puntatori ai nodi dai puntatori **NIL**.
 - In altre parole, al posto di un puntatore **NIL** si usa un puntatore ad un nodo **NIL**.
 - Quando un nodo ha come figli nodi **NIL**, quel nodo sarebbe una foglia nell'albero binario di ricerca corrispondente.

Altezza di un albero AVL

Dato un qualsiasi **albero AVL** con n nodi, si può dimostrare che la sua **altezza** è determinata dalla seguente formula:

$$h \cong 1.44 \log(n + 2) - 0.328$$

Alberi Red-Black (alberi rossi-neri)

Un **Albero Red-Black (rosso-nero)** deve soddisfare le seguenti proprietà (vincoli):

- 1 **Ogni nodo** è colorato o di **rosso** o di **nero**;
- 2 Per convenzione, i **nodi NIL** si considerano **nodi neri**;
- 3 **Se un nodo è rosso**, allora **entrambi i suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** (figlio di una foglia) ha lo **stesso numero di nodi neri**;

Alberi Red-Black (alberi rossi-neri)

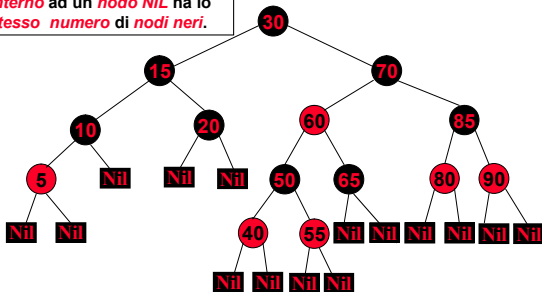
Un **Albero Red-Black (rosso-nero)** deve soddisfare le seguenti proprietà (vincoli):

- 1 **Ogni nodo** è colorato o di **rosso** o di **nero**;
- 2 Per convenzione, i **nodi NIL** si considerano **nodi neri**;
- 3 **Se un nodo è rosso**, allora **entrambi i suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** (figlio di una foglia) ha lo **stesso numero di nodi neri**;

Considereremo solo **alberi Red-Black** in cui la **radice è nera**.

Alberi Red-Black: esempio 1

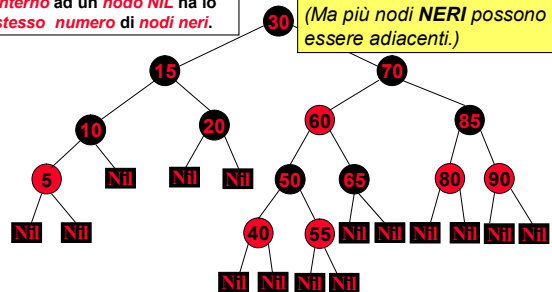
- 3 Se un **nodo è rosso**, allora **entrambi i suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo **stesso numero di nodi neri**.



Alberi Red-Black: esempio 1

- 3 Se un **nodo è rosso**, allora **entrambi i suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo **stesso numero di nodi neri**.

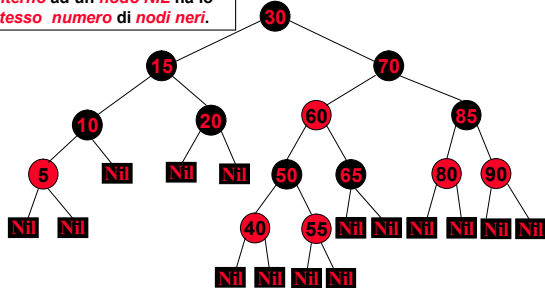
Vincolo 3 impone che non possano esserci due nodi ROSSI adiacenti.
(Ma più nodi NERI possono essere adiacenti.)



Alberi Red-Black: esempio I

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

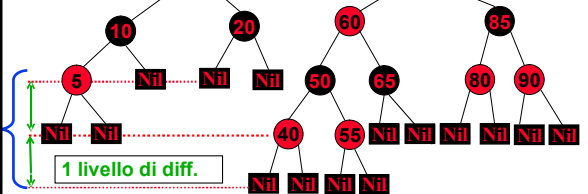
Ci sono **3 nodi neri** lungo ogni percorso dalla radice ad un **nodo NIL**.



Alberi Red-Black: esempio I

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

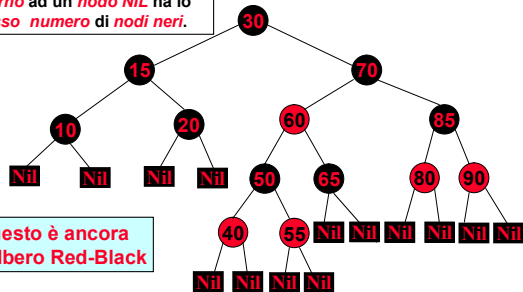
altezza 2 → 15 ← altezza 3



Alberi Red-Black: esempio II

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

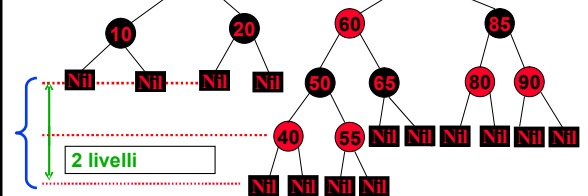
Questo è ancora un **albero Red-Black**



Alberi Red-Black: esempio II

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

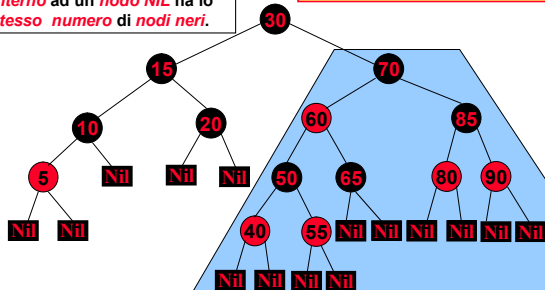
altezza 1 → 15 ← altezza 3



Alberi Red-Black: esempio I

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

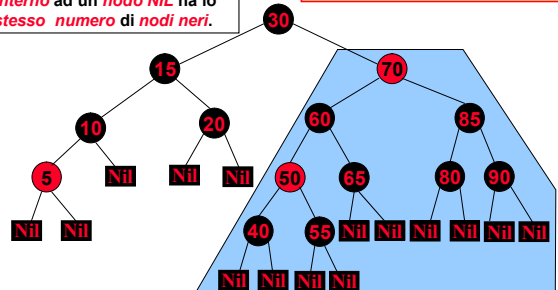
Ci sono **3 nodi neri** lungo ogni percorso dalla radice ad un **nodo NIL**.



Alberi Red-Black: esempio III

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

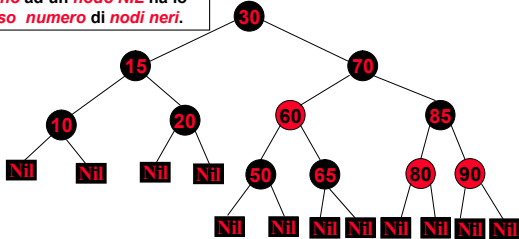
Ci sono **3 nodi neri** lungo ogni percorso dalla radice ad un **nodo NIL**.



Alberi Red-Black: esempio IV

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

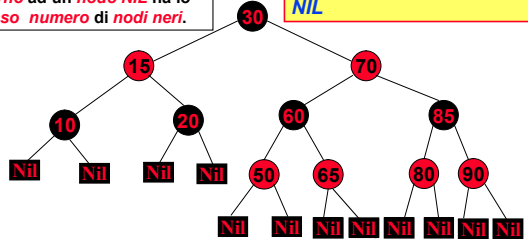
Albero RB con 3 nodi neri lungo ogni percorso dalla radice ad un nodo NIL.



Alberi Red-Black: esempio V

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

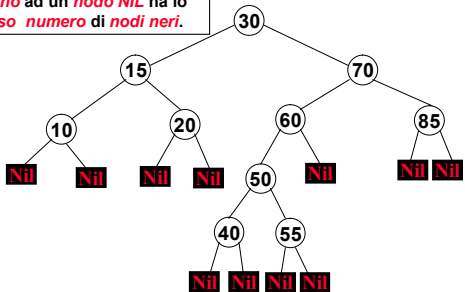
Stesso albero con 2 nodi neri lungo ogni percorso dalla radice ad un nodo NIL.



Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

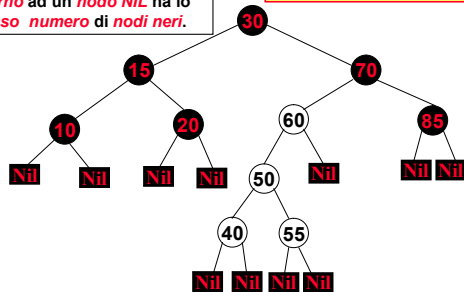
Questo albero può essere un albero Red-Black?



Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 ci possono essere al più 3 nodi neri lungo un percorso!

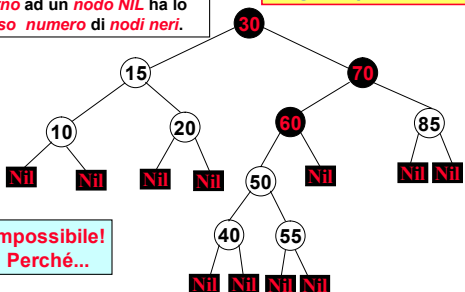


Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 ci possono essere al più 3 nodi neri lungo un percorso!

Impossibile! Perché...

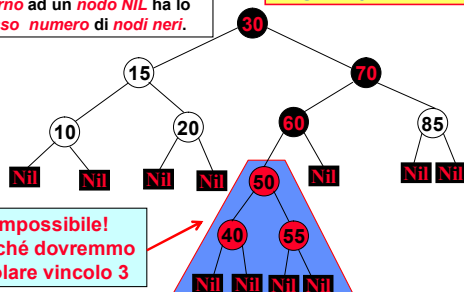


Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 ci possono essere al più 3 nodi neri lungo un percorso!

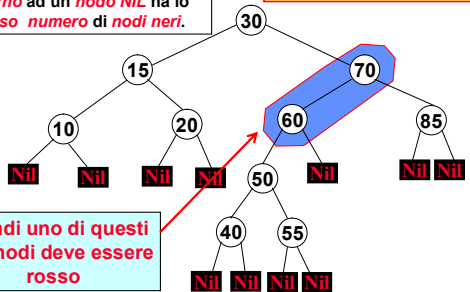
Impossibile! Perché dovremmo violare vincolo 3



Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 ci possono essere al più 3 nodi neri lungo un percorso!

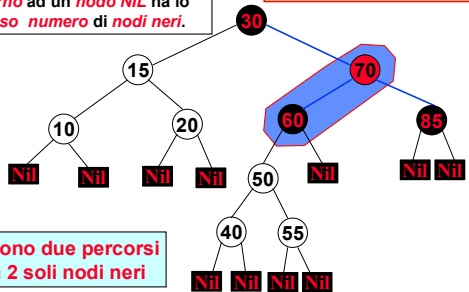


Quindi uno di questi due nodi deve essere rosso

Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 ci possono essere al più 3 nodi neri lungo un percorso!

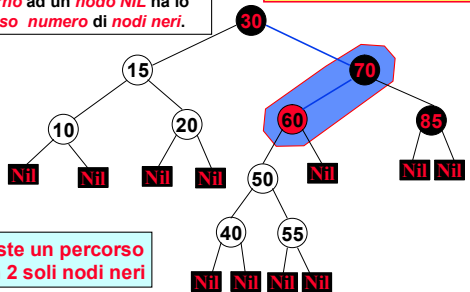


Esistono due percorsi con 2 soli nodi neri

Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 ci possono essere al più 3 nodi neri lungo un percorso!

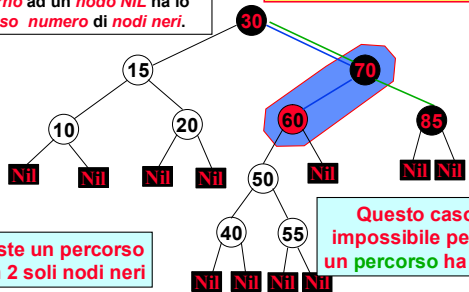


Esiste un percorso con 2 soli nodi neri

Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 ci possono essere al più 3 nodi neri lungo un percorso!



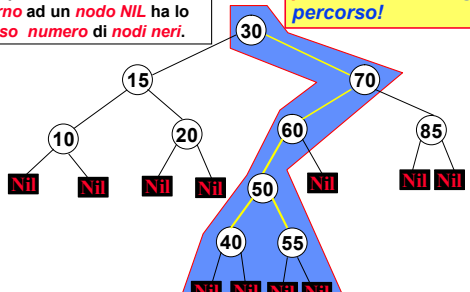
Esiste un percorso con 2 soli nodi neri

Questo caso è impossibile perché un percorso ha 3 neri

Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

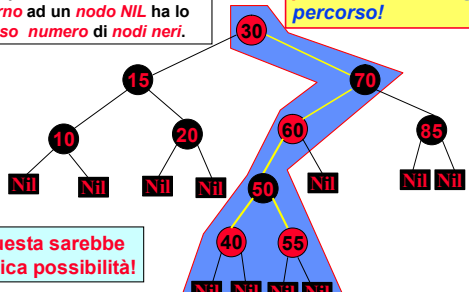
Per vincolo 4 e il vincolo 3, ci possono essere al più 2 nodi neri lungo un percorso!



Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 e il vincolo 3, ci possono essere al più 2 nodi neri lungo un percorso!

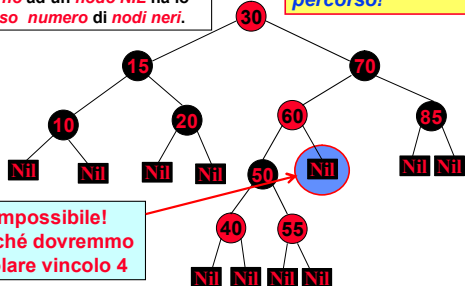


Questa sarebbe l'unica possibilità!

Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Per vincolo 4 e il vincolo 3, ci possono essere **al più 2 nodi neri lungo un percorso!**

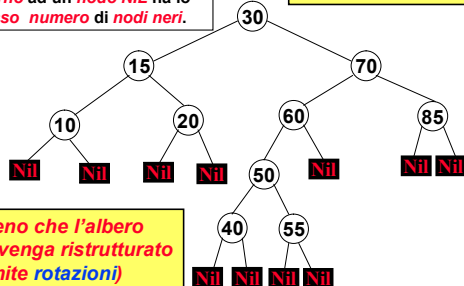


Impossibile! Perché dovremmo violare vincolo 4

Alberi Red-Black: esempio VI

- 3 Se un **nodo è rosso**, allora entrambi i **suoi figli sono neri**;
- 4 Ogni percorso da un **nodo interno** ad un **nodo NIL** ha lo stesso numero di **nodi neri**.

Questo albero **NON** può essere un albero Red-Black!



A meno che l'albero non venga ristrutturato (tramite rotazioni)

Percorso Nero in alberi Red-Black

Definizione: Il numero di **nodi neri** lungo ogni percorso da un **nodo x** (escluso) ad una **foglia** (nodo **NIL**) è detto **altezza nera di x**

Definizione: L'**altezza nera di un albero Red-Black** è l'**altezza nera della sua radice**.

Percorso massimo in alberi Red-Black

Lemma: Un albero Red-Black con **n** nodi ha altezza pari **al più a:**

$$2 \log(n + 1)$$

Dimostrazione: Iniziamo col dimostrare per **induzione** che il sottoalbero con radice **x** ha almeno

$$2^{bh(x)} - 1 \text{ nodi interni}$$

dove **bh(x)** è l'**altezza nera di x**.

L'**induzione** sarà sull'**altezza** del nodo **x**.

Percorso massimo in alberi Red-Black

Il sottoalbero con radice **x** ha al meno

$$2^{bh(x)} - 1 \text{ nodi interni}$$

dove **bh(x)** è l'**altezza nera di x**.

Passo Base:

Se **x** ha **altezza 0**: allora **x** è una **foglia NIL** e il suo sottoalbero contiene **0 nodi interni**.

La sua **altezza nera** è inoltre **bh(x) = 0**.

Otteniamo quindi:

$$0 \geq 2^{bh(x)} - 1 = 2^0 - 1 = 0$$

Percorso massimo in alberi Red-Black

Il sottoalbero con radice **x** ha al meno

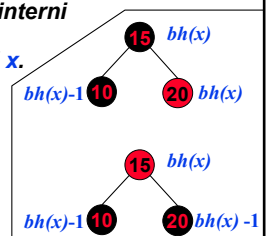
$$2^{bh(x)} - 1 \text{ nodi interni}$$

dove **bh(x)** è l'**altezza nera di x**.

Passo Induttivo:

Se **x** sia interno con 2 figli e abbia **altezza bh(x) > 1**

Entrambi i suoi figli avranno **altezza bh(x) o bh(x)-1**



Percorso massimo in alberi Red-Black

Il sottoalbero con radice x ha al meno

$$2^{bh(x)} - 1 \text{ nodi interni}$$

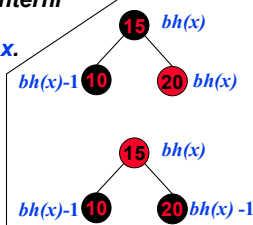
dove $bh(x)$ è l'altezza nera di x .

Passo Induttivo:

Sia x nodo interno con 2 figli e abbia altezza > 0

Entrambi i suoi figli avranno altezza nera $bh(x)$ o $bh(x)-1$

Entrambi i suoi figli avranno certamente altezza minore di x , quindi possiamo usare ipotesi induttiva



Percorso massimo in alberi Red-Black

Il sottoalbero con radice x ha al meno

$$2^{bh(x)} - 1 \text{ nodi interni}$$

dove $bh(x)$ è l'altezza nera di x .

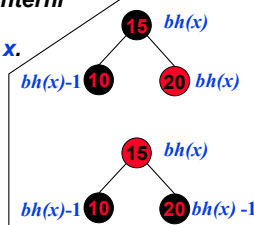
Passo Induttivo:

Se x sia interno con 2 figli e abbia altezza > 0

Entrambi i suoi figli avranno altezza nera $bh(x)$ o $bh(x)-1$

Ogni figlio avrà almeno

$$2^{bh(x)-1} - 1 \text{ nodi interni}$$



Percorso massimo in alberi Red-Black

Il sottoalbero con radice x ha al meno

$$2^{bh(x)} - 1 \text{ nodi interni}$$

dove $bh(x)$ è l'altezza nera di x .

Passo Induttivo: Sia x nodo interno con 2 figli e abbia altezza > 0 . Ogni figlio avrà almeno

$$2^{bh(x)-1} - 1 \text{ nodi interni}$$

Quindi il sottoalbero con radice x conterrà almeno

$$(2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1 = (2^{bh(x)} - 1) \text{ nodi interni}$$

Percorso massimo in alberi Red-Black

Il sottoalbero con radice x ha quindi almeno

$$2^{bh(x)} - 1 \text{ nodi interni}$$

dove $bh(x)$ è l'altezza nera di x .

Completiamo la dimostrazione del lemma.

Sia ora h l'altezza dell'albero.

Per il vincolo 3 almeno metà dei nodi lungo ogni percorso (esclusa la radice) sono neri.

Quindi l'altezza nera dell'albero dovrà essere almeno $h/2$ (cioè $bh \geq h/2$).

Percorso massimo in alberi Red-Black

Il sottoalbero con radice x ha al meno

$$2^{bh(x)} - 1 \text{ nodi interni}$$

dove $bh(x)$ è l'altezza nera di x .

Completiamo la dimostrazione del lemma.

Sia ora h l'altezza dell'albero.

Quindi l'altezza nera dell'albero dovrà essere almeno $h/2$ (cioè $bh \geq h/2$).

Ma allora, il numero di nodi dell'albero sarà

$$n \geq (2^{bh(x)} - 1) \geq 2^{h/2} - 1$$

Percorso massimo in alberi Red-Black

Lemma: Un albero Red-Black con n nodi ha altezza al più $2 \log(n+1)$

Dimostrazione:

Completiamo la dimostrazione del lemma.

Sia ora h l'altezza dell'albero.

Ma allora, il numero di nodi dell'albero sarà

$$n \geq (2^{bh(x)} - 1) \geq 2^{h/2} - 1$$

Portando 1 a sinistra e applicando il logaritmo:

$$\log(n+1) \geq h/2 \quad \text{cioè} \quad h \leq 2 \log(n+1)$$

Costo operazioni su alberi Red-Black

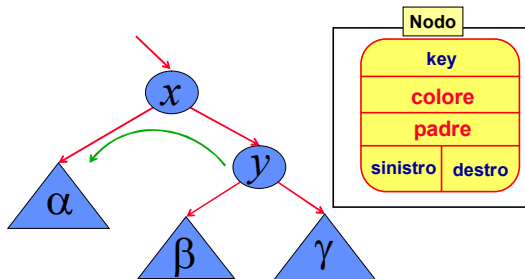
Teorema: In un *albero Red-Black* le operazioni di *ricerca*, *inserimento* e *cancellazione* hanno costo $O(\log(n))$.

Dimostrazione: Conseguenza diretta del **Lemma** precedente e dal fatto che tutte le operazioni hanno costo $O(h)$, con h l'altezza dell'albero.

Violazione delle proprietà per inserimento

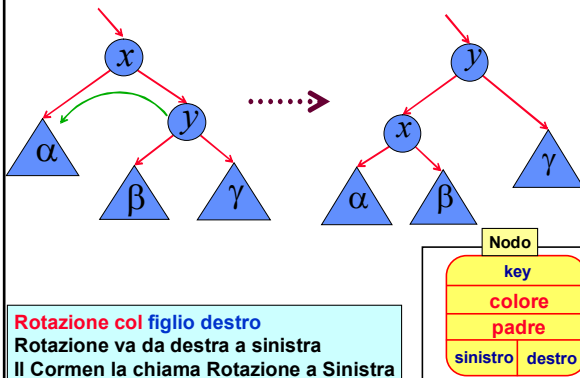
- Durante la *costruzione* di un *albero Red-Black*, quando un *nuovo nodo* viene *inserito* è possibile che le condizioni di bilanciamento risultino *violate*.
- Lo stesso accade per le *cancellazioni*.
- Quando le *proprietà Red-Black* vengono *violate* si può agire:
 - *modificando i colori* nella zona della violazione;
 - *operando dei ribilanciamenti dell'albero tramite rotazioni: Rotazione destra e Rotazione sinistra;*

Alberi Red-Black: Rotazione destra



Rotazione col figlio destro
Rotazione va da destra a sinistra
Il Cormen la chiama Rotazione a Sinistra

Alberi Red-Black: Rotazione destra



Rotazione col figlio destro
Rotazione va da destra a sinistra
Il Cormen la chiama Rotazione a Sinistra

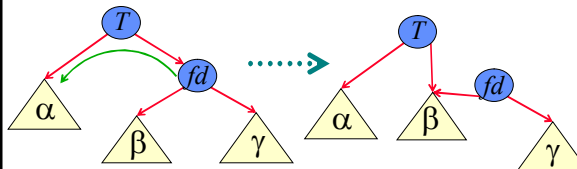
Alberi Red-Black: Rotazioni

```
albero-RB Ruota-destro (T:albero-RB)
  fd = figlio-dx[T]
  figlio-dx[T] = figlio-sx[fd]
  figlio-sx[fd] = T
  return fd
```

```
albero-RB Ruota-sinistro (T:albero-RB)
  fs = figlio-sx[T]
  figlio-sx[T] = figlio-dx[fs]
  figlio-dx[fs] = T
  return fs
```

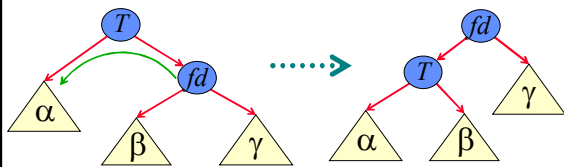
Alberi Red-Black: Rotazioni

```
albero-RB Ruota-destro (T:albero-RB)
  fd = figlio-dx[T]
  figlio-dx[T] = figlio-sx[fd]
  figlio-sx[fd] = T
  return fd
```



Alberi Red-Black: Rotazioni

```
albero-RB Ruota-destro(T:albero-RB)
  fd = figlio-dx[T]
  figlio-dx[T] = figlio-sx[fd]
  figlio-sx[fd] = T
  return fd
```



Inserimento in alberi Red-Black

- Inseriamo un nuovo nodo (chiave) usando la stessa procedura usata per gli ABR;
- Coloriamo il nuovo nodo di rosso;
- La ritorno dalle chiamate ricorsive di inserimento, ripristiniamo la proprietà Red-Black se è il caso:
 - spostiamo le violazioni verso l'alto rispettando sempre il vincolo 4 (mantenendo l'altezza nera dell'albero);
- Al termine dell'inserimento, coloriamo sempre la "radice di nero".

Le operazioni di ripristino sono necessarie solo quando due nodi consecutivi sono rossi!

Inserimento in alberi Red-Black

```
albero-RB INS_RB(k, T)
  IF T ≠ NIL THEN
    IF k < K[T] THEN
      figlio_sx[T] = INS_RB(k, figlio_sx[T]);
      T = Bilancia_sx(T);
    ELSE /* k ≥ K[T] */
      figlio_dx[T] = INS_RB(k, figlio_dx[T]);
      T = Bilancia_dx(T);
    ELSE
      T = alloca_nodo; K[T] = k;
      figlio_dx[T] = NIL;
      figlio_sx[T] = NIL;
      color[T] = red;
  return T;
```

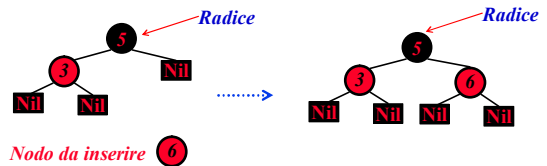
Al termine dell'inserimento, la proc. chiamante deve sempre colorare la "radice di nero".

Inserimento in alberi Red-Black

Le operazioni di ripristino sono necessarie solo quando due nodi consecutivi sono rossi!

Se la radice dell'albero è sempre nera, non si presenta mai la necessità di ribilanciare l'albero se la sua altezza è minore di 3!

Non si possono, infatti, verificare violazioni!

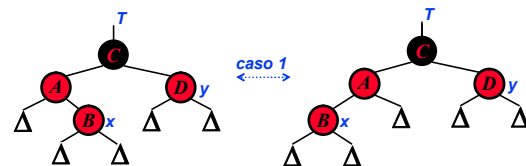


Bilanciamento del sottoalbero sinistro

```
albero-RB Bilancia_sx(T)
  IF ha_figlio(figlio_sx[T])
    v = Tipo_violazione(figlio_sx[T], figlio_dx[T])
    /* v = 0 nessuna violazione */
    CASE v OF
      1: T = Bilancia_sx_1(T);
      2: T = Bilancia_sx_2(T);
          T = Bilancia_sx_3(T);
      3: T = Bilancia_sx_3(T);
  return T;
```

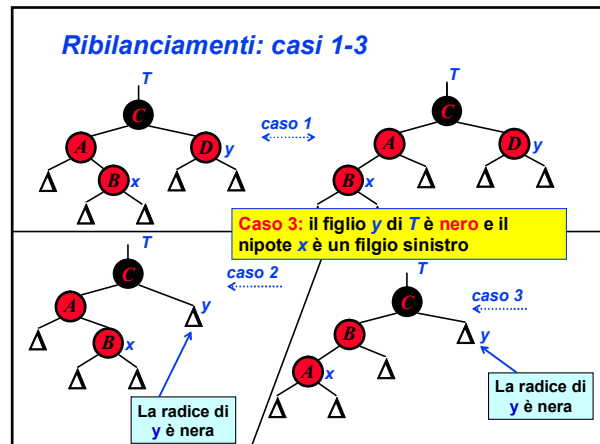
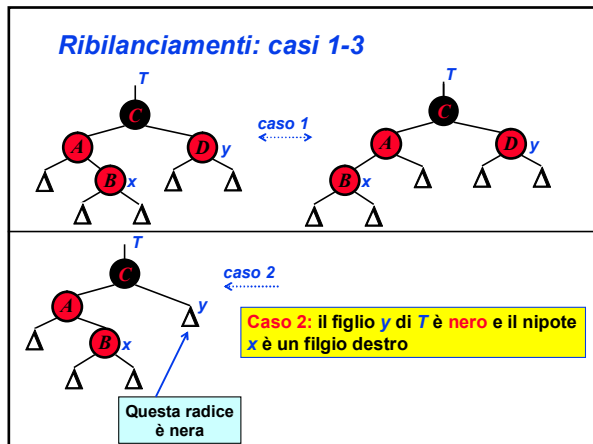
Verifica se l'altezza del sottoalbero sinistro di T è maggiore di 0

Ribilanciamenti: casi 1-3



Caso 1: il figlio y di T è rosso

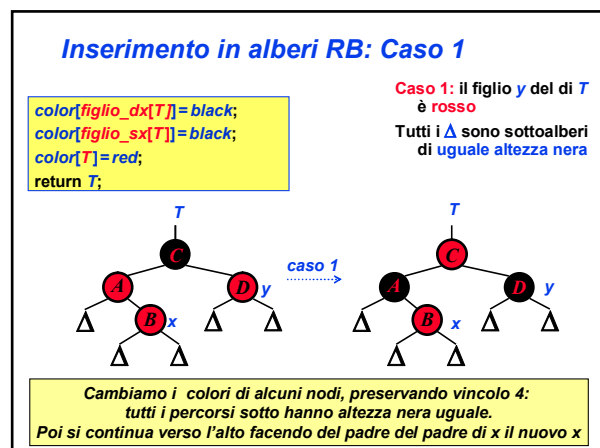
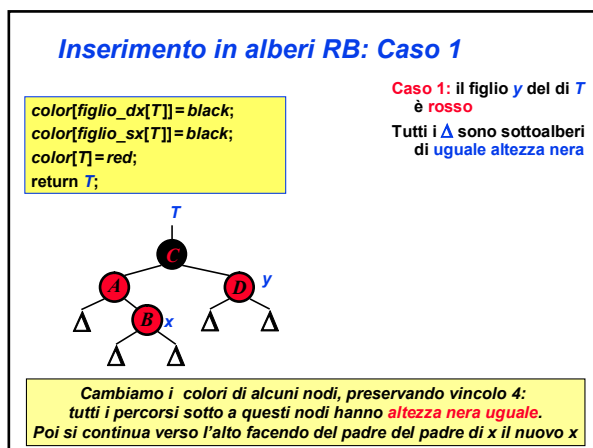
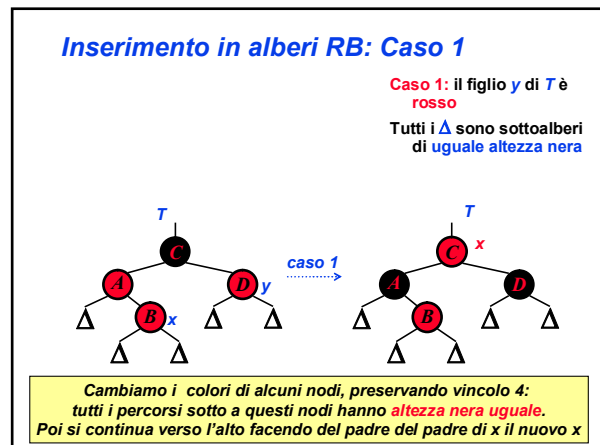
- x è il nodo modificato che provoca la violazione Red-Black
- y è il figlio destro di T



Calcolo del tipo di violazione

```

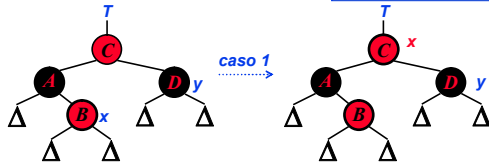
int Tipo_violazione(s,d)
violazione = 0
IF color[s] = red && color[d] = red THEN
  IF color[figlio_sx[s]] = red || color[figlio_dx[s]] = red
    violazione = 1;
  ELSE
    IF color[s] = red THEN
      IF color[figlio_dx[s]] = red
        violazione = 2;
      ELSE
        IF color[figlio_sx[s]] = red
          violazione = 3;
    return violazione;
  
```



Inserimento in alberi RB: Caso 1

```
colore[figlio_dx[T]] = black;
colore[figlio_sx[T]] = black;
colore[T] = red;
return T;
```

Poiché il padre di C può essere rosso, può essere necessario ripristinare la proprietà RB più in alto



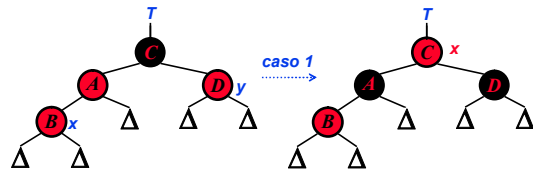
Cambiamo i colori di alcuni nodi, preservando vincolo 4: tutti i percorsi sotto hanno altezza nera uguale. Poi si continua verso l'alto facendo del padre del padre di x il nuovo x

Inserimento in alberi RB: Caso 1

```
colore[figlio_dx[T]] = black;
colore[figlio_sx[T]] = black;
colore[T] = red;
return T;
```

Caso 1: il figlio y del padre del padre di x è rosso

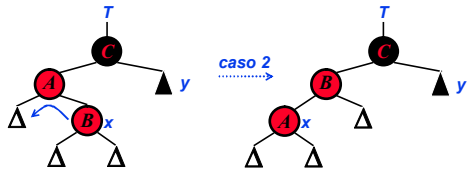
Tutti i Δ sono sottoalberi di uguale altezza nera



Si eseguono le stesse azioni sia quando x è un figlio sinistro o un figlio destro

Inserimento in alberi RB: Caso 2

- Caso 2:
- il figlio y di T è nero
 - x è un figlio destro
 - Trasformiamo nel caso 3 con rotazione destra

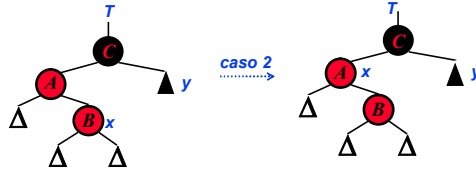


Trasformiamo il caso 2 nel caso 3 (x è figlio sinistro) con una rotazione destra. Ciò preserva il vincolo 4: tutti i percorsi sotto x contengono lo stesso numero di nodi neri

Inserimento in alberi RB: Caso 2

```
figlio_sx[T] = Ruota-dx(figlio_sx[T])
return T
// continua con caso 3
```

- Caso 2:
- il figlio y di T è nero
 - x è un figlio destro
 - Trasformiamo nel caso 3 con rotazione destra

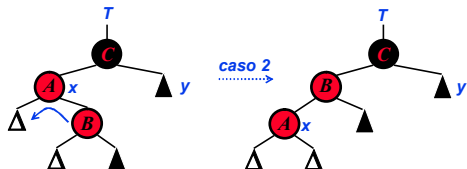


Trasformiamo il caso 2 nel caso 3 (x è figlio sinistro) con una rotazione destra. Ciò preserva il vincolo 4: tutti i percorsi sotto x contengono lo stesso numero di nodi neri

Inserimento in alberi RB: Caso 2

```
figlio_sx[T] = Ruota-dx(figlio_sx[T])
return T
// continua con caso 3
```

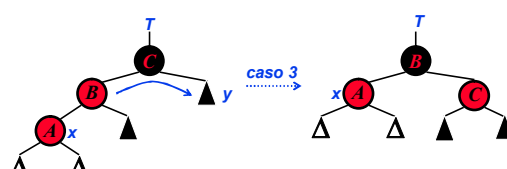
- Caso 2:
- il figlio y di T è nero
 - x è un figlio destro
 - Trasformiamo nel caso 3 con rotazione destra



Trasformiamo il caso 2 nel caso 3 (x è figlio sinistro) con una rotazione destra. Ciò preserva il vincolo 4: tutti i percorsi sotto x contengono lo stesso numero di nodi neri

Inserimento in alberi RB: Caso 3

- Caso 3:
- il figlio y di T è nero
 - x è un figlio sinistro
 - Cambiare colori e rotazione sinistra

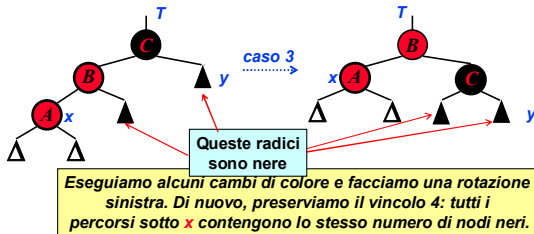


Eseguiamo alcuni cambi di colore e facciamo una rotazione sinistra. Di nuovo, preserviamo il vincolo 4: tutti i percorsi sotto x contengono lo stesso numero di nodi neri.

Inserimento in alberi RB: Caso 3

```
T = Ruota_sx(T);
colore[T] = black;
colore[figlio_dx[T]] = red;
return T;
```

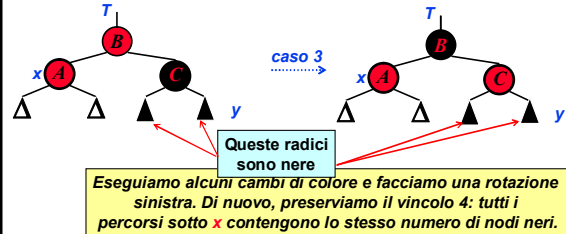
- Caso 3:**
- il figlio y del padre del padre di x è nero
 - x è un figlio sinistro
 - Cambiare colori e rotazione sinistra



Inserimento in alberi RB: Caso 3

```
T = Ruota_sx(T)
colore[T] = black;
colore[figlio_dx[T]] = red;
return T
```

- Caso 3:**
- il figlio y del padre del padre di x è nero
 - x è un figlio sinistro
 - Cambiare colori e rotazione sinistra



Bilanciamento in alberi Red-Black

```
albero-RB Bilancia_sx_1(T)
colore[T] = red;
colore[figlio_dx[T]] = black;
colore[figlio_sx[T]] = black;
return T;
```

```
albero-RB Bilancia_sx_2(T)
figlio_sx[T] = Ruota_dx(figlio_sx[T]);
return T;
```

```
albero-RB Bilancia_sx_3(T)
T = Ruota_sx(T)
colore[T] = black;
colore[figlio_dx[T]] = red;
return T;
```

Bilanciamento del sottoalbero sinistro

```
albero-RB Bilancia_sx(T)
IF ha_figlio(figlio_sx[T])
v = Tipo_violazione(figlio_sx[T],figlio_dx[T])
/* v = 0 nessuna violazione */
CASE v OF
1: T = Bilancia_sx_1(T);
2: T = Bilancia_sx_2(T);
   T = Bilancia_sx_3(T);
3: T = Bilancia_sx_3(T);
return T;
```

Inserimento in alberi RB: Casi 4-6

- I casi 1-3 assumono che il figlio di T , che viola con un nipote di T , sia un figlio sinistro.
- Se il figlio di T , che viola con nipote di T , è un figlio destro si applicano i casi 4-6, che sono simmetrici (dobbiamo solo scambiare *sinistro* con *destra* e viceversa).

L'estensione dell'algoritmo ai casi 4-6 è lasciato come **esercizio**

Inserimento in alberi Red-Black: I

Il padre è NERO, il nuovo nodo x diventa ROSSO

