ALBERI AVL: INSERIMENTO E CANCELLAZIONE
ALBERI ROSSO NERI
DYNAMIC SELECT

ALGORITMI E STRUTTURE DATI

ALBERI ROSSO-NERI

ALTRE TIPOLOGIE DI ALBERI BINARI DI RICERCA BILANCIATI

ALTRE TIPOLOGIE DI ALBERI BINARI DI RICERCA BILANCIATI

- Gli alberi AVL non sono l'unico tipo albero binario di ricerca bilanciato

ALTRE TIPOLOGIE DI ALBERI BINARI DI RICERCA BILANCIATI

- Gli alberi AVL non sono l'unico tipo albero binario di ricerca bilanciato
- Un altro tipo di albero che si incontra spesso "in the wild" sono i **red-black tree** o alberi rosso-neri.

ALTRE TIPOLOGIE DI ALBERI BINARI DI RICERCA BILANCIATI

- Gli alberi AVL non sono l'unico tipo albero binario di ricerca bilanciato
- Un altro tipo di albero che si incontra spesso "in the wild" sono i **red-black tree** o alberi rosso-neri.
- Idea: ogni nodo contiene un bit di informazione aggiuntivo che dice se il nodo è colorato di nero o di rosso

PROPRIETÀ DEGLI ALBERI ROSSO-NERI

PROPRIETÀ DEGLI ALBERI ROSSO-NERI

- Ogni nodo è colorato di rosso o di nero

PROPRIETÀ DEGLI ALBERI ROSSO-NERI

- Ogni nodo è colorato di rosso o di nero
- La radice deve essere colorata di nero

PROPRIETÀ DEGLI ALBERI ROSSO-NERI

- Ogni nodo è colorato di rosso o di nero
- La radice deve essere colorata di nero
- I valori None si considerano colorati di nero

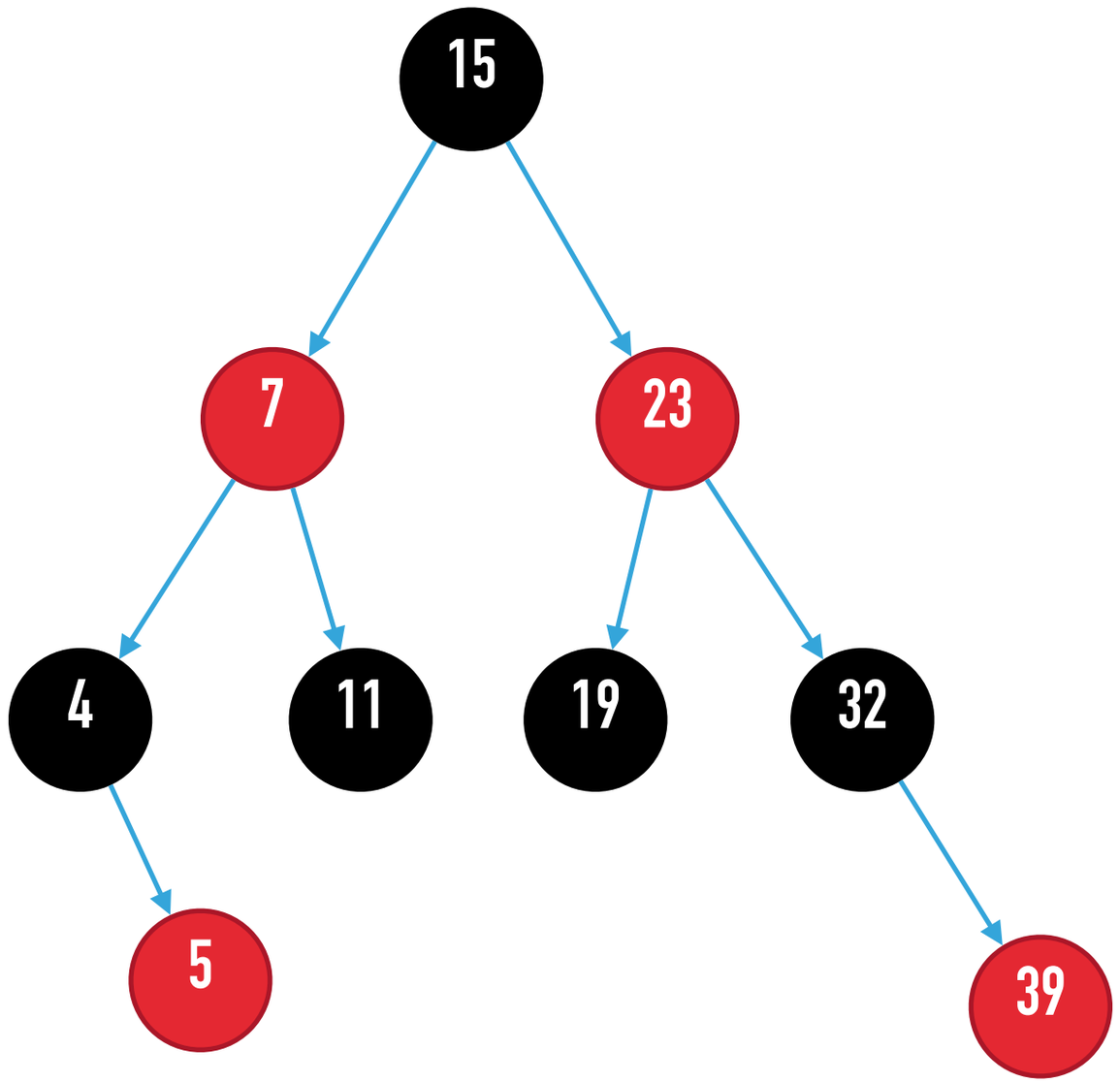
PROPRIETÀ DEGLI ALBERI ROSSO-NERI

- Ogni nodo è colorato di rosso o di nero
- La radice deve essere colorata di nero
- I valori None si considerano colorati di nero
- I figli di un nodo rosso devono essere nodi neri

PROPRIETÀ DEGLI ALBERI ROSSO-NERI

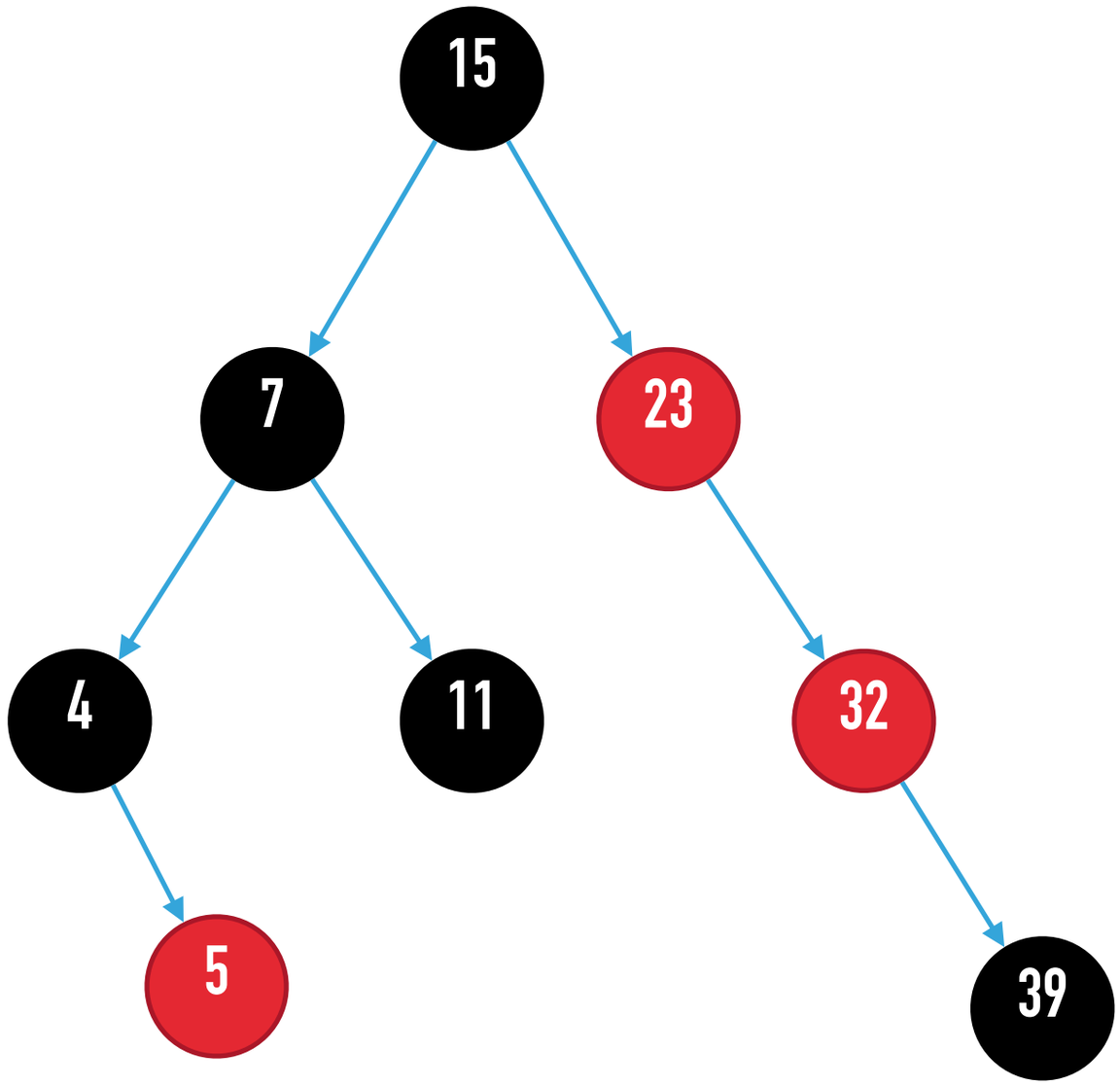
- Ogni nodo è colorato di rosso o di nero
- La radice deve essere colorata di nero
- I valori None si considerano colorati di nero
- I figli di un nodo rosso devono essere nodi neri
- Ogni percorso da un qualsiasi nodo ad un valore None nel sottoalbero del nodo (equiv. un percorso per arrivare alle foglie o ai nodi con reference a None) attraversa lo stesso numero di nodi neri

ALBERO ROSSO-NERO



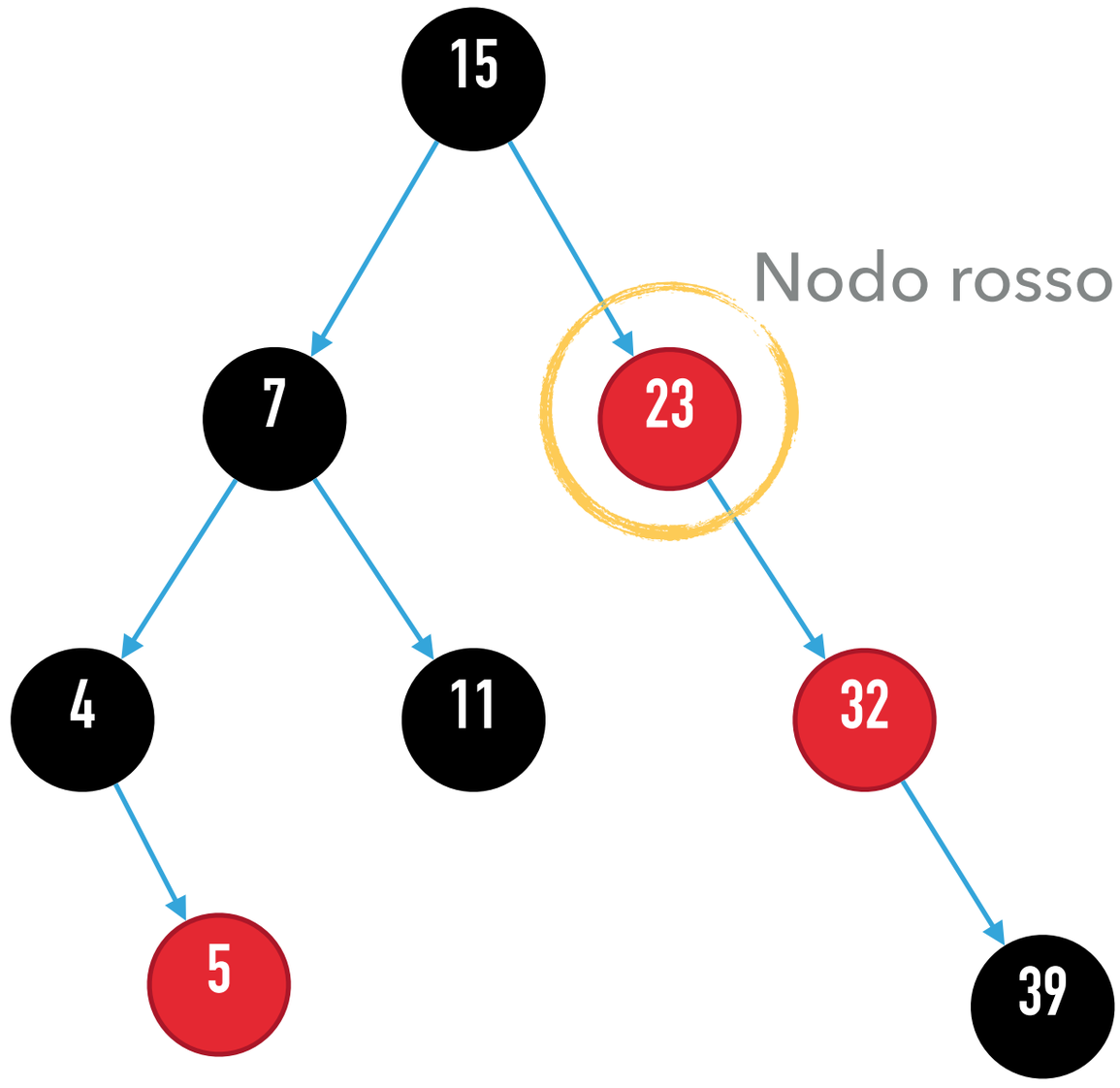
Questo albero rispetta tutte le proprietà di albero rosso-nero

ALBERO ROSSO-NERO



Questo albero NON rispetta
tutte le proprietà di albero rosso-nero

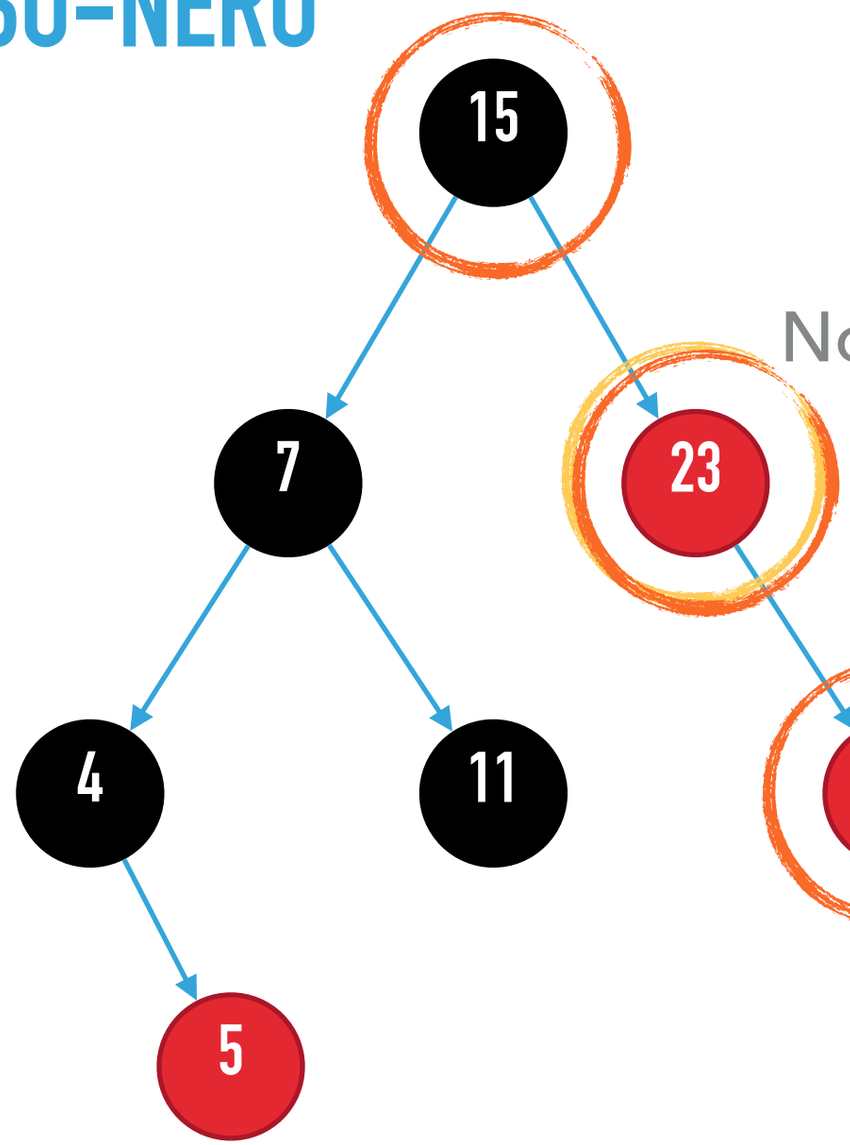
ALBERO ROSSO-NERO



Nodo rosso con figlio rosso

Questo albero NON rispetta
tutte le proprietà di albero rosso-nero

ALBERO ROSSO-NERO



Nodo rosso con figlio rosso

Numero differente di nodi neri per arrivare alle foglie o ai nodi con reference a None

Questo albero NON rispetta
tutte le proprietà di albero rosso-nero

ALBERI ROSSO-NERI: INSERIMENTO

ALBERI ROSSO-NERI: INSERIMENTO

- L'inserimento negli alberi rosso-neri inizia come per gli alberi binari di ricerca normali

ALBERI ROSSO-NERI: INSERIMENTO

- L'inserimento negli alberi rosso-neri inizia come per gli alberi binari di ricerca normali
- Il nodo da inserire viene colorato di rosso

ALBERI ROSSO-NERI: INSERIMENTO

- L'inserimento negli alberi rosso-neri inizia come per gli alberi binari di ricerca normali
- Il nodo da inserire viene colorato di rosso
- Se una proprietà degli alberi rosso-neri viene violata si eseguono una serie di ricolorazioni e rotazioni per ripristinarla (sempre in tempo $O(\log n)$)

ALBERI ROSSO-NERI: NODI CONTENUTI

ALBERI ROSSO-NERI: NODI CONTENUTI

- Mostriamo ora che un albero rosso-nero con n nodi ha altezza $O(\log n)$

ALBERI ROSSO-NERI: NODI CONTENUTI

- Mostriamo ora che un albero rosso-nero con n nodi ha altezza $O(\log n)$
- Dato che ogni nodo negli alberi rosso-neri ha lo stesso numero di nodi neri da lui stesso alle foglie e ai nodi con reference a None, possiamo definire la black height del nodo

ALBERI ROSSO-NERI: NODI CONTENUTI

- Mostriamo ora che un albero rosso-nero con n nodi ha altezza $O(\log n)$
- Dato che ogni nodo negli alberi rosso-neri ha lo stesso numero di nodi neri da lui stesso alle foglie e ai nodi con reference a None, possiamo definire la black height del nodo
- Dato un nodo x, definiamo $bh(x)$ il numero di nodi neri che si devono attraversare per arrivare da x a una qualsiasi foglia e nodo con reference a None

ALBERI ROSSO-NERI: NODI CONTENUTI

ALBERI ROSSO-NERI: NODI CONTENUTI

Mostriamo per induzione che ogni sottoalbero avente il nodo x come radice contiene almeno $2^{bh(x)} - 1$ nodi interni (i.e., foglie escluse).

ALBERI ROSSO-NERI: NODI CONTENUTI

Mostriamo per induzione che ogni sottoalbero avente il nodo x come radice contiene almeno $2^{bh(x)} - 1$ nodi interni (i.e., foglie escluse).

La proprietà è vera è vero se x è una foglia: $bh(x) = 0$, e contiene $2^{bh(x)} - 1 = 2^0 - 1 = 0$ nodi interni.

ALBERI ROSSO-NERI: NODI CONTENUTI

Mostriamo per induzione che ogni sottoalbero avente il nodo x come radice contiene almeno $2^{bh(x)} - 1$ nodi interni (i.e., foglie escluse).

La proprietà è vera è vero se x è una foglia: $bh(x) = 0$, e contiene $2^{bh(x)} - 1 = 2^0 - 1 = 0$ nodi interni.

Sia x un nodo con $bh(x) > 0$. Questo significa che ha due figli di altezza $bh(x)$ o $bh(x) - 1$ a seconda del colore.

ALBERI ROSSO-NERI: NODI CONTENUTI

ALBERI ROSSO-NERI: NODI CONTENUTI

Per ipotesi induttiva ciascuno dei sottoalberi ha almeno $2^{bh(x)-1} - 1$ nodi interni.

ALBERI ROSSO-NERI: NODI CONTENUTI

Per ipotesi induttiva ciascuno dei sottoalberi ha almeno $2^{bh(x)-1} - 1$ nodi interni.

Quindi x ha almeno $(2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1$, ovvero $2^{bh(x)} - 1$ nodi interni, che è quello che volevamo provare.

ALBERI ROSSO-NERI: NODI CONTENUTI

Per ipotesi induttiva ciascuno dei sottoalberi ha almeno $2^{bh(x)-1} - 1$ nodi interni.

Quindi x ha almeno $(2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1$, ovvero $2^{bh(x)} - 1$ nodi interni, che è quello che volevamo provare.

Consideriamo ora l'altezza h della radice r. Dato che la non possono esserci due nodi consecutivi rossi e che la radice è nera, essa ha $bh(r) \geq h/2$

ALBERI ROSSO-NERI: NODI CONTENUTI

ALBERI ROSSO-NERI: NODI CONTENUTI

Ma, per la proprietà precedente, l'intero albero può contenere al più $2^{bh(r)} - 1 = 2^{h/2} - 1$ nodi.

ALBERI ROSSO-NERI: NODI CONTENUTI

Ma, per la proprietà precedente, l'intero albero può contenere al più $2^{bh(r)} - 1 = 2^{h/2} - 1$ nodi.

Da questo deriviamo

$$n \geq 2^{h/2} - 1$$
$$\log_2(n-1) \geq \log_2(2^{h/2})$$
$$2\log_2(n-1) \geq h$$

ALBERI ROSSO-NERI: NODI CONTENUTI

Ma, per la proprietà precedente, l'intero albero può contenere al più $2^{bh(r)} - 1 = 2^{h/2} - 1$ nodi.

Da questo deriviamo

$$n \geq 2^{h/2} - 1$$
$$\log_2(n-1) \geq \log_2(2^{h/2})$$
$$2\log_2(n-1) \geq h$$

E quindi $h \leq 2\log_2(n-1)$, quindi l'altezza dell'albero è al più due volte $\log_2(n-1)$, quindi $O(\log n)$.

DYNAMIC SELECT

CALCOLARE I PERCENTILI IN UN INSIEME DINAMICO

CALCOLARE I PERCENTILI IN UN INSIEME DINAMICO

Supponiamo di avere un albero binario di ricerca. Come facciamo a trovare il p-simo percentile, e.g. la mediana?

CALCOLARE I PERCENTILI IN UN INSIEME DINAMICO

Supponiamo di avere un albero binario di ricerca. Come facciamo a trovare il p-simo percentile, e.g. la mediana?

Partiamo dalla radice. Ci sono tre casi:

CALCOLARE I PERCENTILI IN UN INSIEME DINAMICO

Supponiamo di avere un albero binario di ricerca. Come facciamo a trovare il p-simo percentile, e.g. la mediana?

Partiamo dalla radice. Ci sono tre casi:

- a sx della radice *r* ci sono $m = \left\lfloor \frac{n}{2} \right\rfloor$ nodi: *r* è la mediana

CALCOLARE I PERCENTILI IN UN INSIEME DINAMICO

Supponiamo di avere un albero binario di ricerca. Come facciamo a trovare il p-simo percentile, e.g. la mediana?

Partiamo dalla radice. Ci sono tre casi:

- a sx della radice *r* ci sono $m = \left\lfloor \frac{n}{2} \right\rfloor$ nodi: *r* è la mediana

- a sx della radice *r* ci sono $<m$ nodi: la mediana è a dx

CALCOLARE I PERCENTILI IN UN INSIEME DINAMICO

Supponiamo di avere un albero binario di ricerca. Come facciamo a trovare il p-simo percentile, e.g. la mediana?

Partiamo dalla radice. Ci sono tre casi:

- a sx della radice *r* ci sono $m = \left\lfloor \frac{n}{2} \right\rfloor$ nodi: *r* è la mediana

- a sx della radice *r* ci sono <*m* nodi: la mediana è a dx

- a sx della radice *r* ci sono >*m* nodi: la mediana è a sx

CALCOLARE I PERCENTILI IN UN INSIEME DINAMICO

Scriviamo lo pseudocodice

AUMENTARE UNA STRUTTURA DATI

AUMENTARE UNA STRUTTURA DATI

Per poter risolvere il problema di select efficientemente, devo quindi sapere quanti nodi contiene un sottoalbero.

AUMENTARE UNA STRUTTURA DATI

Per poter risolvere il problema di select efficientemente, devo quindi sapere quanti nodi contiene un sottoalbero.

Aggiungo ad ogni nodo *x* una variabile *size*, che conta il numero di nodi nel sottoalbero radicato in *x*. Vale

x.size = x.left.size + x.right.size +1

AUMENTARE UNA STRUTTURA DATI

Per poter risolvere il problema di select efficientemente, devo quindi sapere quanti nodi contiene un sottoalbero.

Aggiungo ad ogni nodo *x* una variabile *size*, che conta il numero di nodi nel sottoalbero radicato in *x*. Vale
x.size = x.left.size + x.right.size +1

Devo estendere le operazioni su un BST per mantenere aggiornato il campo *size*. In particolare per INSERT e DELETE.

INSERT PER DYNAMIC SELECT

INSERT PER DYNAMIC SELECT

Inserisco il nodo normalmente come foglia. Quali nodi avranno un campo *size* diverso?

INSERT PER DYNAMIC SELECT

Inserisco il nodo normalmente come foglia. Quali nodi avranno un campo *size* diverso?

Risalgo dal nodo inserito alla radice dell'albero ed incremento *size* di 1 per ogni nodo.

INSERT PER DYNAMIC SELECT

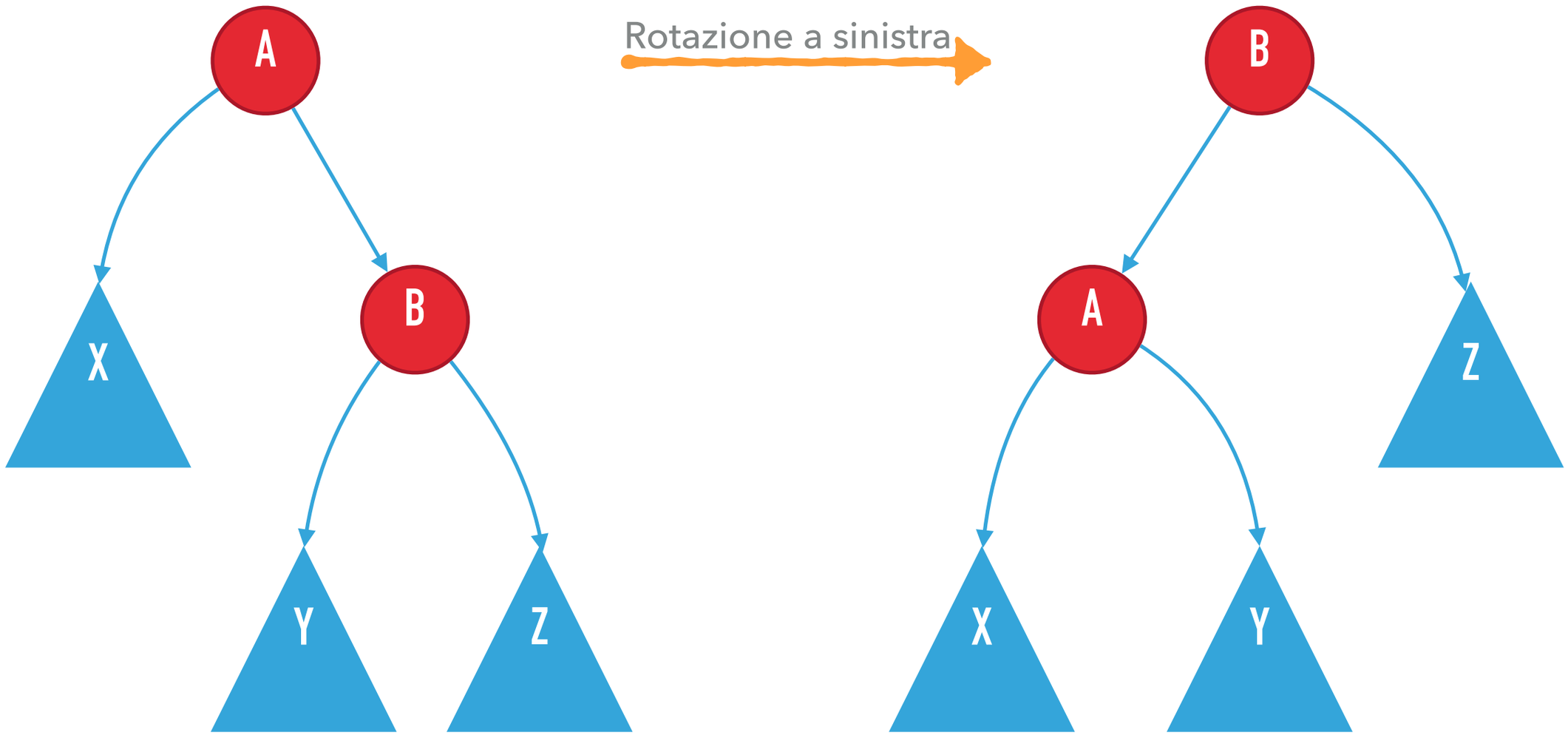
Inserisco il nodo normalmente come foglia. Quali nodi avranno un campo *size* diverso?

Risalgo dal nodo inserito alla radice dell'albero ed incremento *size* di 1 per ogni nodo.

Nel caso di un albero AVL, in cui posso fare delle rotazioni per bilanciare l'albero, devo modificare le rotazioni in modo che preservino il valore corretto di *size*.

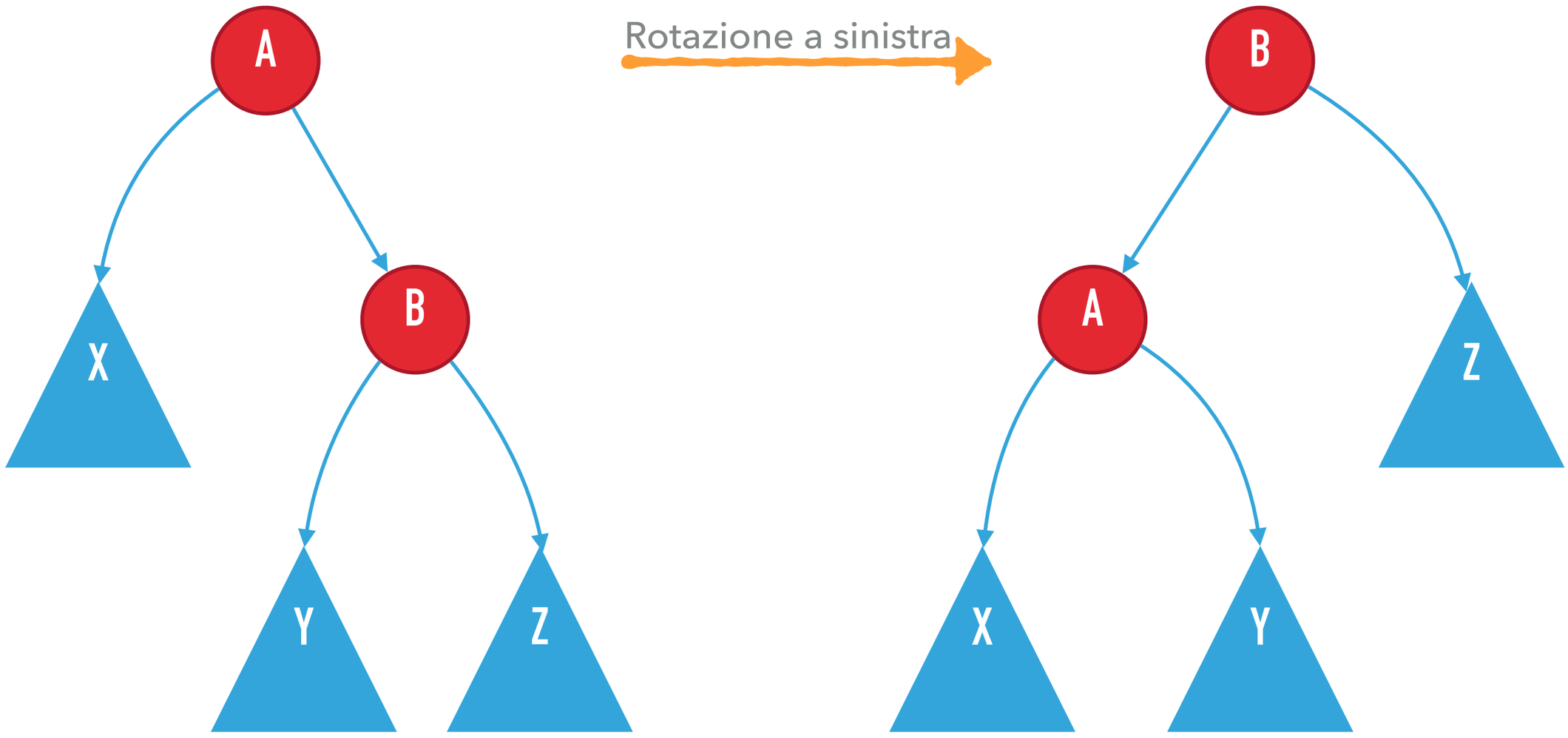
ROTAZIONI PER DYNAMIC SELECT

Cambiano solo i valori di size in A e B



ROTAZIONI PER DYNAMIC SELECT

Cambiano solo i valori di size in A e B



size(A) = size(A) - size(B.right) - 1
size(B) = size(B) + size(A.left) + 1

ROTAZIONE SINISTRA PER DYNAMIC SELECT

- Parametri: nodo a
- b = a.right # nodo che prenderà il posto di a
- **b.size = b.size + a.left.size + 1 # aggiorno size b**
- **a.size = a.size - b.right.size - 1 # aggiorno size a**
- a.right = b.left
- if a.right is not None:
    - a.right.parent = a
- b.left = a
- # con queste operazioni abbiamo messo il figlio sinistro di b come figlio destro di a, dobbiamo ancora sistemare i genitori di a e b
- if a.parent is not None: # sistemiamo il genitore di a
    - if a.parent.left is a: # nel caso a fosse figlio sinistro
        - a.parent.left = b
    - else # nel caso a fosse figlio destro
        - a.parent.right = b
- b.parent = a.parent
- a.parent = b