

## RICERCA MINIMO IN ABR

MIN(T)

IF T ≠ NIL THEN

IF T->S<sub>x</sub> ≠ NIL THEN

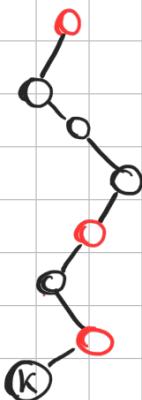
RETURN MIN(T->S<sub>x</sub>)

RETURN T

## RICERCA SUCCESSORE DI K IN ABR

Il successore di un numero in una sequenza è il più piccolo elemento tra i maggiori di K, quindi dovranno prendere il minimo del sottoalbero destro (maggiori di) del nodo di valore K.

Se K non ha sottosalberi destri dobbiamo considerare tutti i nodi del percorso di ricerca e tenere conto **Solo** di quelli che sviluppano il percorso a sinistra verso K. Tra questi dovranno prendere l'ultimo (che sarà il più piccolo tra questi.)



Se  $K$  **NON** e' presente lo gestiremo come quest'ultimo caso

SUCCESSORE ( $T, K$ )

IF  $T \neq \text{NIL}$  THEN

IF  $T \rightarrow \text{Key} < K$  THEN

Continuiamo a cercare  $K$

RETURN SUCCESSORE ( $T \rightarrow dx, K$ )

se vado solo a dx ritorna NIL

ELSE IF  $T \rightarrow \text{Key} > K$  THEN

$\text{RET} = \text{SUCCESSORE } (T \rightarrow s_x, K)$

"Salviamo" il risultato della visita  
a  $s_x$

IF  $\text{RET} = \text{NIL}$  THEN

$\text{RET} = T$

Se la visita e' arrivata alle foglie

ritorna l'ultimo nodo  $> K$

RETURN RET

ELSE

RETURN MINIMO ( $T \rightarrow dx$ )  $T \rightarrow \text{key} = K$

RETURN T

## CANCELLAZIONE IN ABR

A differenza delle liste qui abbiamo un problema in quanto  
il nodo da eliminare e' puntato da un solo nodo ma punta  
ai suoi due figli, che non possono perci' essere puntati assieme

del solo puntatore del padre di K.

Il trucco sarà non eliminare il nodo dove si trova K, bensì un modo più facile da eliminare e pure il suo contenuto nel nodo che contiene K. Vediamo a seconda dei casi quale modo scegliere:

Cerchiamo anzitutto nodi profici oppure che hanno un solo figlio così da gestire più facilmente l'eliminazione.

Se K appartiene a T prendiamo il massimo del sottoalbero sinistro oppure il minimo elemento del sottoalbero destro.

Questi sono per definizione predecessore o successore di K quindi scambiandolo con il nodo contenente K non abbiamo bisogno di "aggiustare" tutto l'albero in quanto non disturba il padre di K perché rispetta le proprietà proprio come K

CANCELLA(T,K)

IF  $T \neq \text{NIL}$

IF  $\text{key} < K$  THEN

$T \rightarrow dx = \text{CANCELLA}(T \rightarrow dx, K)$

ELSE IF  $\text{key} > K$  THEN

$T \rightarrow S_x = \text{CANCELLA}(T \rightarrow S_x, K)$

ELSE

$T \rightarrow \text{Key} = K$

IF  $T \rightarrow S_x \neq \text{NIL}$   $\wedge T \rightarrow d_x \neq \text{NIL}$  THEN *Caso problematico*

$Y = \text{STACCA-MINIMO}(T \rightarrow d_x, T)$  *prendiamo il successore*

$T \rightarrow \text{Key} = Y \rightarrow \text{Key}$

*aggiorniamo T al successore*

ELSE

*che abbiamo eliminato*

IF  $T \rightarrow S_x = \text{NIL}$  THEN

*collega il padre di T all'unico*

$\text{RET} = T \rightarrow d_x$

*figlio di T*

ELSE

$\text{RET} = T \rightarrow S_x$

*dealloca T*

*dealloca il nodo*

RETURN RET

RETURN T

STACCA-MINIMO *scollega il minimo di quell' albero dall' albero e ritorna un puntatore a quel nodo (a cui si è collegato).*

Dobbiamo fare particolarmente attenzione alla struttura interna perché non possiamo banalmente definirlo ricorsivamente, dovranno "portare" sempre il nodo precedente per "attaccarlo" al successivo.

STACCA-MINIMO ( $T, p$ )

IF  $T \neq \text{NIL}$  THEN

IF  $T \rightarrow S_x \neq \text{NIL}$  THEN

CORRENTE PRECEDENTE



RETURN STACCA-MINIMO ( $T \rightarrow S_x, T$ )

ELSE

stacca il minimo ( $T \rightarrow S_x = \text{NIL}$ )

IF  $T = p \rightarrow S_x$  THEN

dobbiamo capire quale figlio stiamo

aggiornando

ELSE

$p \rightarrow S_x = T \rightarrow d_x$

RETURN  $T$

Il tempo di esecuzione di CANCELLA e' lineare sull'altezza  
perche' entrambi gli algoritmi utilizzati dipendono da h

## 'ORDINAMENTO' ABR

Stampiamo una sequenza ordinata dato un ABR sfruttando la DFS in ORDER

ORDER( $T$ )

IF  $T \neq \text{NIL}$  THEN

  ORDER( $T \rightarrow s_x$ )

  PRINT( $T \rightarrow \text{key}$ )

  ORDER( $T \rightarrow d_x$ )

Questo algoritmo impiega  $\Theta(n)$  per l'esecuzione e possiamo considerarlo una sorta di algoritmo di ordinamento, questo è possibile grazie alle proprietà dell'ABR.

Noi abbiamo dimostrato che un algoritmo di ordinamento ha tempo almeno  $n \lg n$ , generando un assurdo. Questo è dovuto al fatto che per inserire nodi in un ABR ci vuole un tempo  $\Omega(n \lg n)$ , ed è facilmente osservabile perché ogni nodo impiega  $\lg n$  (altezza), ad esempio gli ultimi  $\frac{n}{2}$  nodi (foglie) scorreranno tutta l'altezza quindi  $\frac{n}{2} \cdot \lg_2 n = \Omega(n \lg n)$  fino ad arrivare a  $O(n^2)$  (albero degenere)

## VERIFICA DI UN ALBERO HEAP

Verifichiamo se un albero è uno heap.

Ricordiamo che le proprietà dello heap:

$$\forall x \in T \quad \text{val}(x) \geq \text{val}(T \rightarrow s_x) \wedge \text{val}(x) \geq \text{val}(T \rightarrow d_x)$$

Questa proprietà implica solo 3 nodi quinelli non dipende dal numero di nodi dell'albero.

Possiamo verificare che ogni nodo abbia questa proprietà con una DFS pre-order dove la visita consiste nella verifica delle proprietà

ALBEROHEAP( $T$ )

IF  $T \neq \text{NIL}$  THEN

IF  $\neg \text{VISITAHEAP}(T)$  THEN

C'è stata una violazione

RETURN FALSE

ELSE

RETURN ( $\text{ALBEROHEAP}(T \rightarrow s_x)$  AND  $\text{ALBEROHEAP}(T \rightarrow d_x)$ )

RETURN TRUE

Definiamo l'algoritmo di visita

VISITA HEAP ( $T$ )

RET = TRUE

IF  $T \rightarrow s_x \neq \text{NIL}$  THEN

RET =  $(T \rightarrow \text{key} \geq T \rightarrow s_x \rightarrow \text{key})$

$$\text{Val}(x) \geq \text{Val}(s_x)$$

IF  $T \rightarrow d_x \neq \text{NIL}$  THEN

RET = RET  $\wedge$   $(T \rightarrow \text{key} \geq T \rightarrow d_x \rightarrow \text{key})$

$$\text{Val}(x) \leq \text{Val}(d_x)$$

RETURN RET

OSS

VISITA HEAP effettua un check locale perché i nodi non rispettano la proprietà rispetto tutto l'albero, al contrario la proprietà dell'ABR è globale quindi l'algoritmo effettua un check globale.

Queste osservazioni si riflettono sui tempi di esecuzione degli algoritmi: VISITA HEAP ha un tempo costante mentre VISITA ABR avrà tempo lineare (dipende dal numero di nodi perché ogni nodo va confrontato con tutti gli altri nodi)