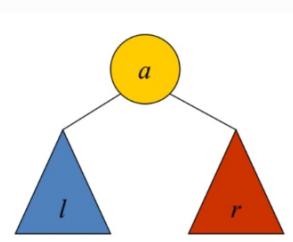


DEFINIZIONE

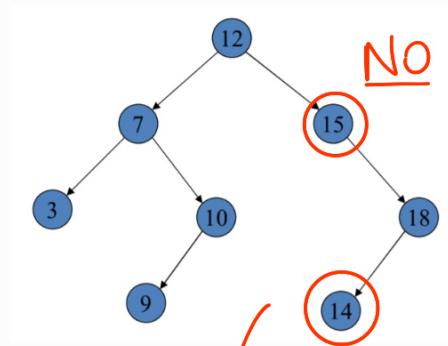
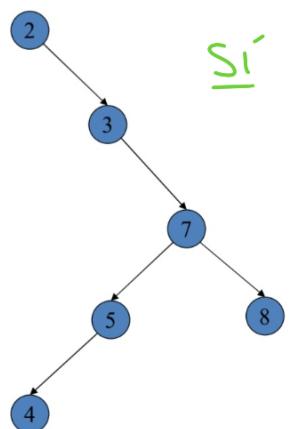
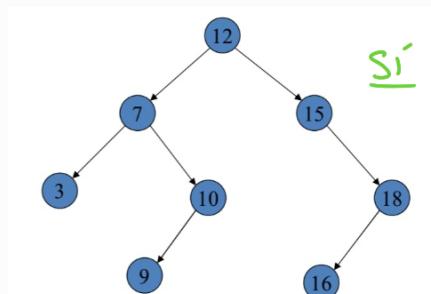
Sia A un insieme ordinato.

$BRT(A)$ = insieme di alberi binari di ricerca su A



- a) $\emptyset \in BRT(A)$
- b) $a \in A \wedge l \in BRT(A) \wedge r \in BRT(A) \wedge \forall c \in keys(l). c < a \wedge \forall c \in keys(r). a < c$
 $\Rightarrow \{a, l, r\} \in BRT(A)$

Esempi:



$14 < 15 \Rightarrow 14$ non può essere nel sottoalbero dx di 15

REALIZZAZIONE CON PUNTATORI

<i>parent</i>	
<i>key</i>	
<i>left</i>	<i>right</i>

ALGORITMI

PRINT-INORDER(T)

▷ pre: T binario di ricerca
▷ post: stampate le chiavi in T in ordine

if $T = \text{nil}$ **then**

return

end if

PRINT-INORDER($T.\text{left}$)

print $T.\text{key}$

PRINT-INORDER($T.\text{right}$)

} prime l , poi a , poi r

RIC-SEARCH(x, T)

▷ pre: x chiave, T binario di ricerca
▷ post: il nodo $S \in T$ con $S.\text{key} = x$ se esiste, nil altrimenti

if $T = \text{nil}$ **then return** nil

else

if $x = T.\text{key}$ **then return** T

else

if $x < T.\text{key}$ **then return** SEARCH($x, T.\text{left}$)

else ▷ $x > T.\text{key}$

return SEARCH($x, T.\text{right}$)

end if

end if

end if

$O(h)$

h : altezza di T

IT-SEARCH(x, T)

▷ pre: x chiave, T binario di ricerca
▷ post: il nodo $S \in T$ con $S.\text{key} = x$ se esiste, nil altrimenti

while $T \neq \text{nil}$ **and** $x \neq T.\text{key}$ **do**

if $x < T.\text{key}$ **then**

$T \leftarrow T.\text{left}$

else

$T \leftarrow T.\text{right}$

end if

end while

return T

$O(h)$

TREE-MIN(T)

▷ pre: T binario di ricerca non vuoto

▷ post: il nodo $S \in T$ con $S.\text{key}$ minimo

$S \leftarrow T$

while $S.\text{left} \neq \text{nil}$ **do**

$S \leftarrow S.\text{left}$

end while

return S

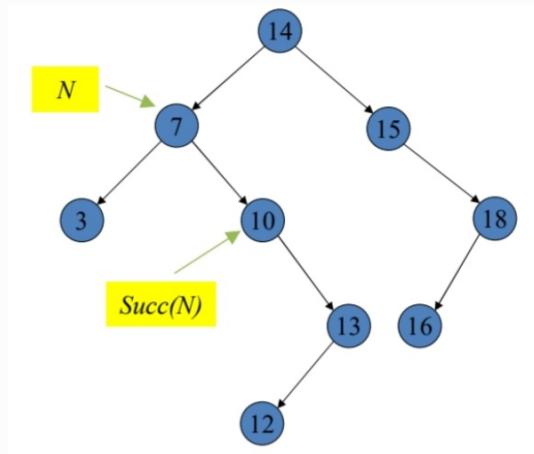
il minimo si trova scendendo verso sinistra

$O(h)$

Per il massimo, e' analogo ma

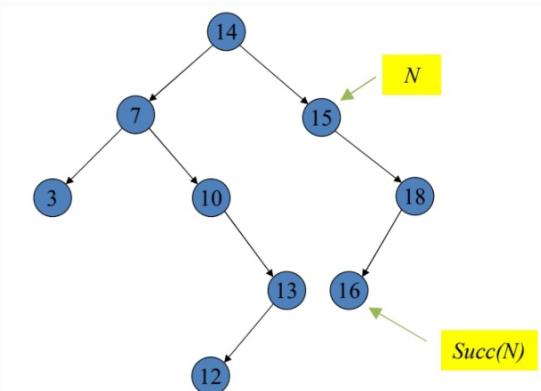
bisogna scendere lungo il ramo dx

SUCCESSORE di un nodo N = il nodo con etichetta minima tra quelle maggiori di $N.key$



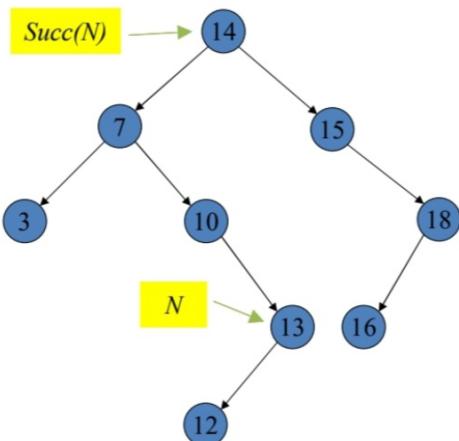
Se $N.right \neq \text{nil}$, allora
Succ(N) è il minimo del
suo sottoalbero destro

→



Se $N.right = \text{nil}$, allora
il suo successore è il suo
avo A più vicino t.c. $N.key < A.key$

→



TREE-SUCC(N)

▷ pre: N nodo di un albero bin. di ricerca

▷ post: il successore di N se esiste, nil altrimenti

if $N.right \neq \text{nil}$ **then**

return TREE-MIN($N.right$)

else ▷ il successore è l'avo più vicino con etichetta maggiore

$P \leftarrow N.parent$

while $P \neq \text{nil}$ **and** $N = P.right$ **do**

$N \leftarrow P$

$P \leftarrow N.parent$

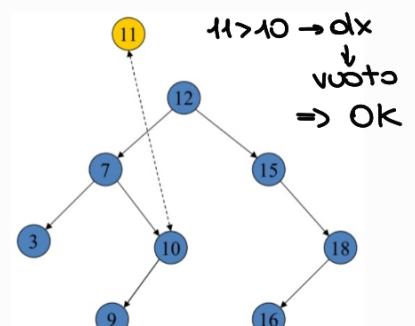
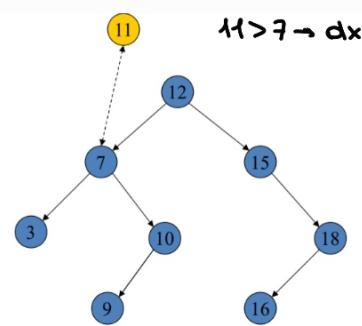
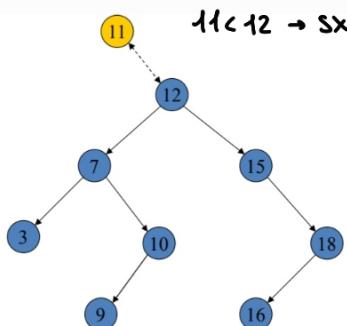
end while

return P

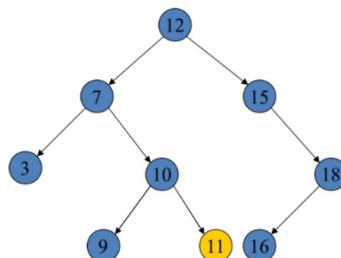
end if

INSERIMENTO → avviene al livello delle foglie, sostituendo un sottoalbero vuoto in modo che l'albero rimanga albero di ricerca.

esempio:



Risultato:



TREE-INSERT(N, T)

▷ pre: N nuovo nodo con $N.left = N.right = nil$, T è un albero binario di ricerca
▷ post: N è un nodo di T , T è un albero binario di ricerca

$P \leftarrow nil$

$S \leftarrow T$

while $S \neq nil$ **do** ▷ inv: se $P \neq nil$ allora P è il padre di S

$P \leftarrow S$

if $N.key = S.key$ **then**) non inserisco una chiave già presente
 return

else

if $N.key < S.key$ **then**

$S \leftarrow S.left$

else

$S \leftarrow S.right$

end if

end if

end while

$N.parent \leftarrow P$

if $P = nil$ **then**

$T \leftarrow N$

else

if $N.key < P.key$ **then**

$P.left \leftarrow N$

else

$P.right \leftarrow N$

end if

end if

CANCELLAZIONE → Supponiamo di voler eliminare il nodo Z

Distinguiamo 3 casi :

- Z non ha figli (foglia) → setto a nil il riferimento a Z di Z.parent
- Z ha esattamente 1 figlio → aggancio il figlio a Z.parent
- Z ha due figli

1-DELETE(Z, T)

▷ pre: Z nodo di T con esattamente un figlio
 ▷ post: Z non è più un nodo di T

if $Z = T$ then

```
if  $Z.left \neq \text{nil}$  then
     $T \leftarrow Z.left$ 
else
     $T \leftarrow Z.right$ 
end if
 $T.parent \leftarrow \text{nil}$ 
```

else

```
if  $Z.left \neq \text{nil}$  then
     $Z.left.parent \leftarrow Z.parent$ 
     $S \leftarrow Z.left$ 
else
     $Z.right.parent \leftarrow Z.parent$ 
     $S \leftarrow Z.right$ 
end if
if  $Z.parent.right = Z$  then
     $Z.parent.right \leftarrow S$ 
else
     $Z.parent.left \leftarrow S$ 
end if
end if
```

Z e' la radice

→ Questo non permette di cancellare qualunque nodo

↓
 Se Z ha due figli ?

PIÙ GENERALE

TREE-DELETE(Z, T)

▷ pre: Z nodo di T

▷ post: Z non è più un nodo di T

if $Z.left = \text{nil} \wedge Z.right = \text{nil}$ then ▷ Z è una foglia

if $Z = T$ then

$T \leftarrow \text{nil}$

else

if $Z.parent.left = Z$ then ▷ Z è figlio sinistro

$Z.parent.left \leftarrow \text{nil}$

else ▷ Z è figlio destro

$Z.parent.right \leftarrow \text{nil}$

end if

end if

else

if $Z.left = \text{nil} \vee Z.right = \text{nil}$ then

1-DELETE(Z, T)

) come prima

|| ha 1 figlio

else ▷ Z ha due figli e dunque si può cercare il minimo in Z.right

$Y \leftarrow \text{TREE-MIN}(Z.right)$

$Z.key \leftarrow Y.key$

TREE-DELETE(Y, T)

quando arrivo qui, sono sicuro che Y
 abbia al max 1 figlio (quello dx)

end if

end if

end if