

Algoritmer og Datastrukturer 1

Binære Søgetræer [CLRS, kapitel 12]



Gerth Stølting Brodal

Aarhus Universitet

Abstrakt Datastruktur: Ordbog

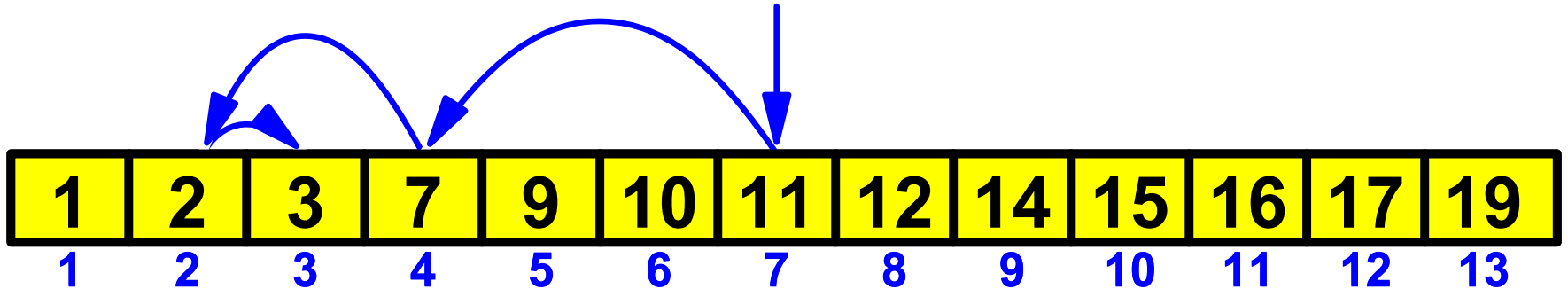
Search(S,x)

Insert(S,x)

Delete(S,x)

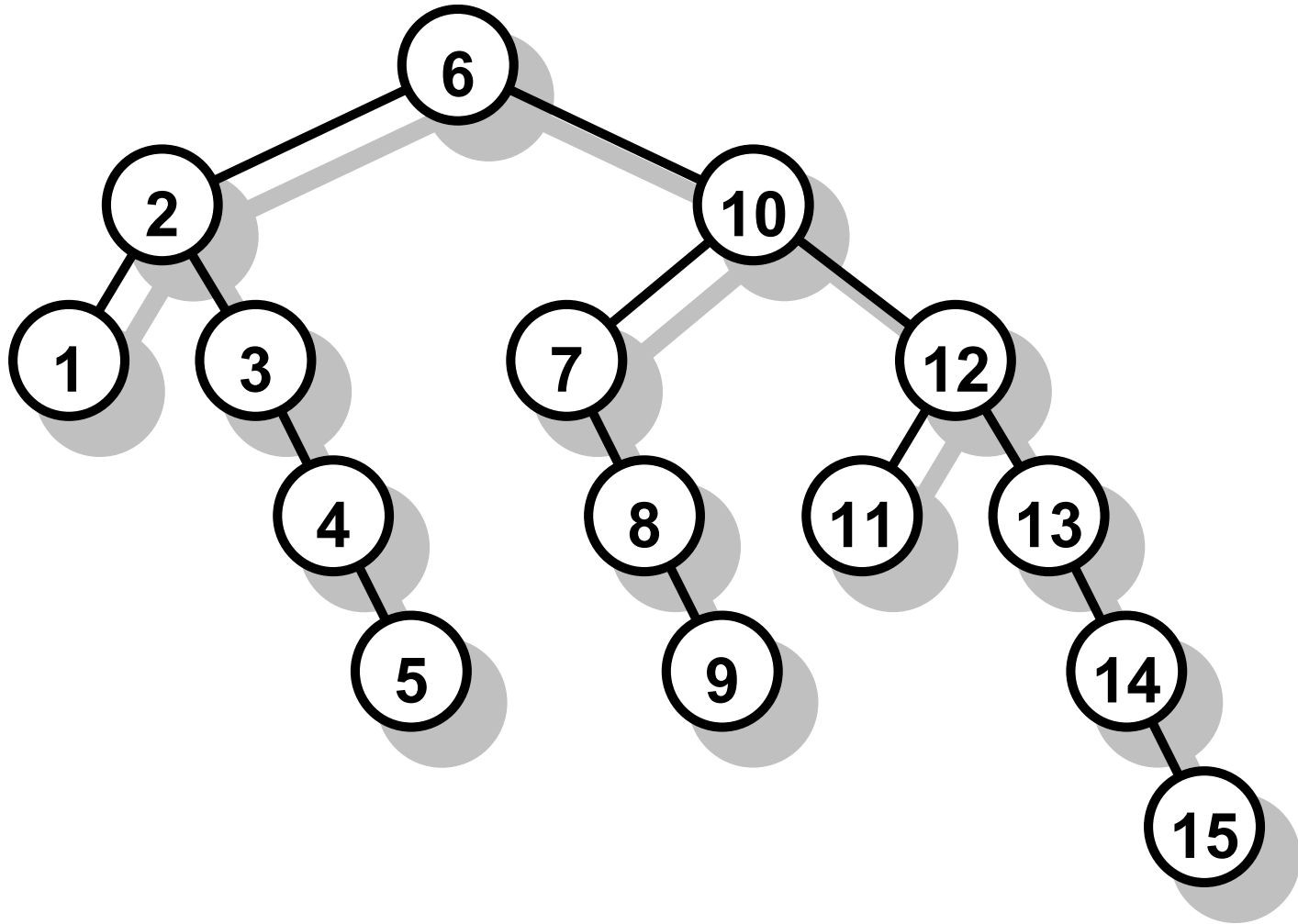
(Statisk) Ordbog : Sorteret Array

Search(4)



Search(S,x)	$O(\log n)$
Insert(S,x)	$O(n)$
Delete(S,x)	

Søgetræ



Invariant For alle knuder er elementerne i venstre (højre) undertræ mindre (større) end eller lig med elementet i knuden

Søgetræs søgninger

INORDER-TREE-WALK(x)

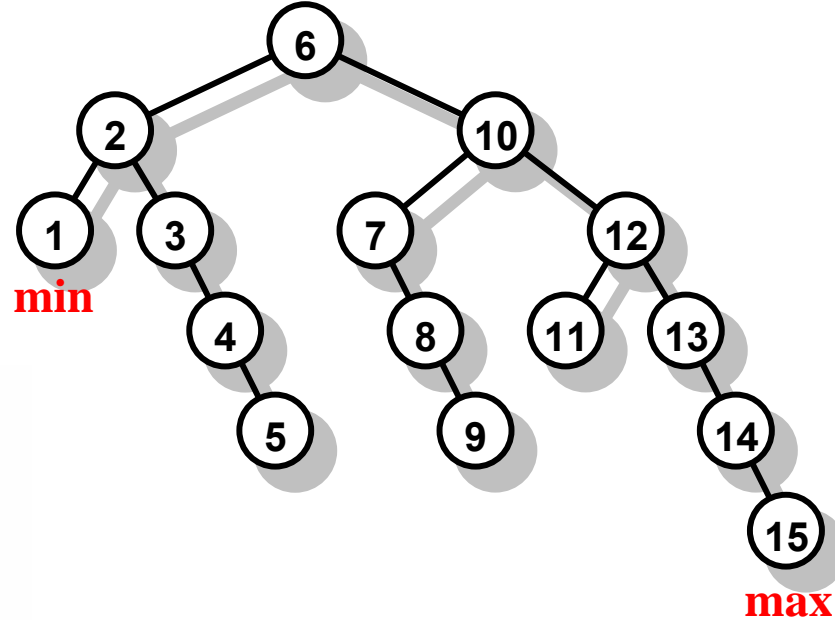
```
1  if  $x \neq \text{NIL}$ 
2    then INORDER-TREE-WALK( $\text{left}[x]$ )
3         print  $\text{key}[x]$ 
4         INORDER-TREE-WALK( $\text{right}[x]$ )
```

TREE-SEARCH(x, k)

```
1  if  $x = \text{NIL}$  or  $k = \text{key}[x]$ 
2    then return  $x$ 
3  if  $k < \text{key}[x]$ 
4    then return TREE-SEARCH( $\text{left}[x], k$ )
5  else return TREE-SEARCH( $\text{right}[x], k$ )
```

ITERATIVE-TREE-SEARCH(x, k)

```
1  while  $x \neq \text{NIL}$  and  $k \neq \text{key}[x]$ 
2    do if  $k < \text{key}[x]$ 
3         then  $x \leftarrow \text{left}[x]$ 
4         else  $x \leftarrow \text{right}[x]$ 
5  return  $x$ 
```



TREE-MINIMUM(x)

```
1  while  $\text{left}[x] \neq \text{NIL}$ 
2    do  $x \leftarrow \text{left}[x]$ 
3  return  $x$ 
```

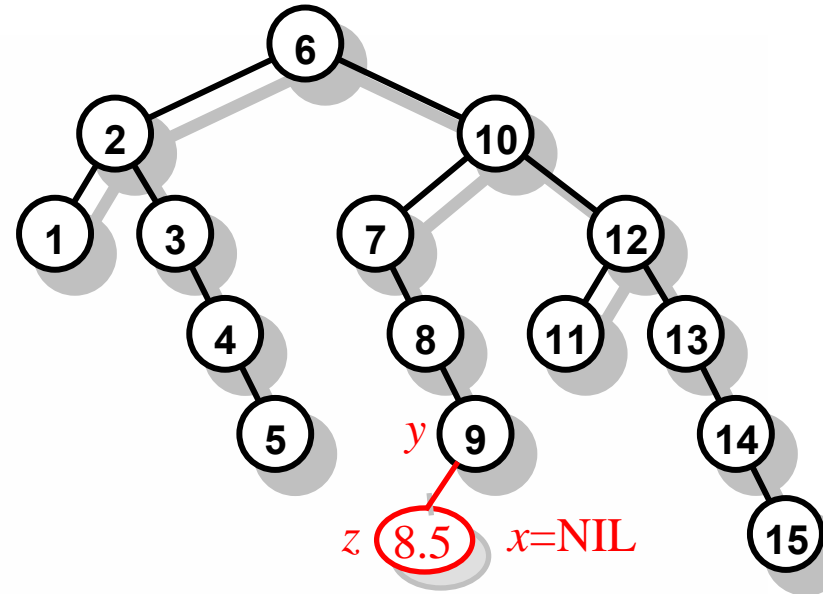
TREE-MAXIMUM(x)

```
1  while  $\text{right}[x] \neq \text{NIL}$ 
2    do  $x \leftarrow \text{right}[x]$ 
3  return  $x$ 
```

Indsættelse i Søgetræ

TREE-INSERT(T, z)

```
1   $y \leftarrow \text{NIL}$ 
2   $x \leftarrow \text{root}[T]$ 
3  while  $x \neq \text{NIL}$ 
4      do  $y \leftarrow x$ 
5          if  $\text{key}[z] < \text{key}[x]$ 
6              then  $x \leftarrow \text{left}[x]$ 
7              else  $x \leftarrow \text{right}[x]$ 
8   $p[z] \leftarrow y$ 
9  if  $y = \text{NIL}$ 
10     then  $\text{root}[T] \leftarrow z$ 
11     else if  $\text{key}[z] < \text{key}[y]$ 
12         then  $\text{left}[y] \leftarrow z$ 
13         else  $\text{right}[y] \leftarrow z$ 
```

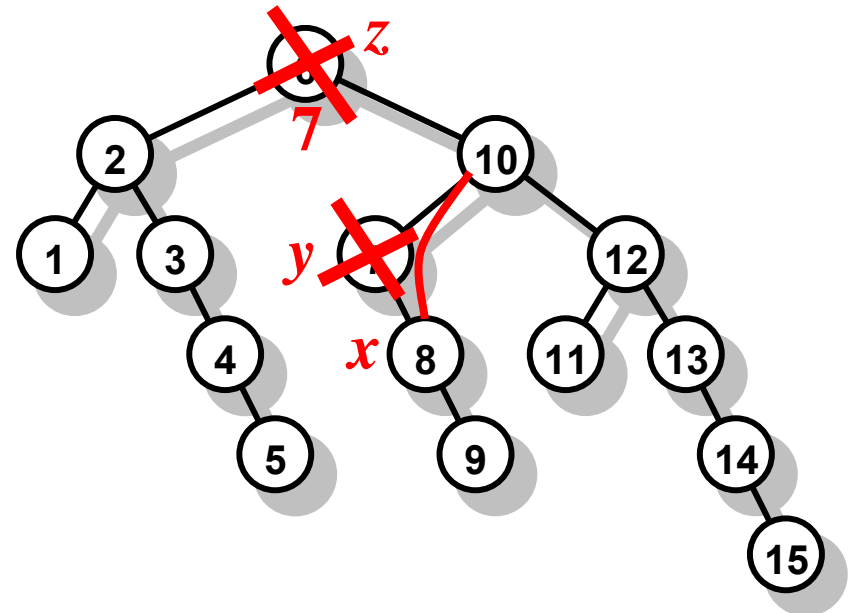


▷ Tree T was empty

Slettelse fra Søgetræ

TREE-DELETE(T, z)

```
1  if  $left[z] = NIL$  or  $right[z] = NIL$ 
2    then  $y \leftarrow z$ 
3    else  $y \leftarrow TREE-SUCCESSOR(z)$ 
4  if  $left[y] \neq NIL$ 
5    then  $x \leftarrow left[y]$ 
6    else  $x \leftarrow right[y]$ 
7  if  $x \neq NIL$ 
8    then  $p[x] \leftarrow p[y]$ 
9  if  $p[y] = NIL$ 
10   then  $root[T] \leftarrow x$ 
11   else if  $y = left[p[y]]$ 
12         then  $left[p[y]] \leftarrow x$ 
13         else  $right[p[y]] \leftarrow x$ 
14  if  $y \neq z$ 
15   then  $key[z] \leftarrow key[y]$ 
16         copy  $y$ 's satellite data into  $z$ 
17  return  $y$ 
```



TREE-SUCCESSOR(x)

```
1  if  $right[x] \neq NIL$ 
2    then return  $TREE-MINIMUM(right[x])$ 
3   $y \leftarrow p[x]$ 
4  while  $y \neq NIL$  and  $x = right[y]$ 
5    do  $x \leftarrow y$ 
6        $y \leftarrow p[y]$ 
7  return  $y$ 
```

Søgetræer

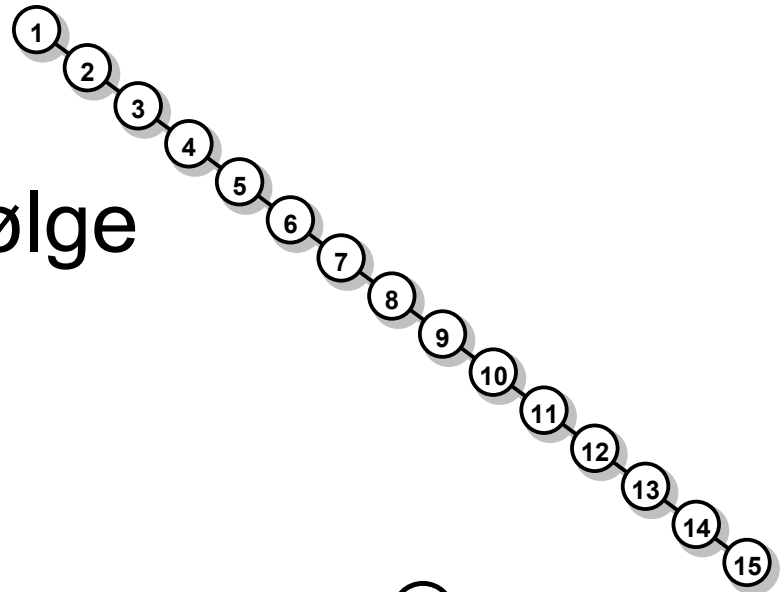
Inorder-Tree-Walk	$O(n)$
Tree-Search Iterative-Tree-Search	$O(h)$
Tree-Minimum Tree-Maximum	$O(h)$
Tree-Insert	$O(h)$
Tree-Delete	$O(h)$

h = højden af træet, **n** = antal elementer

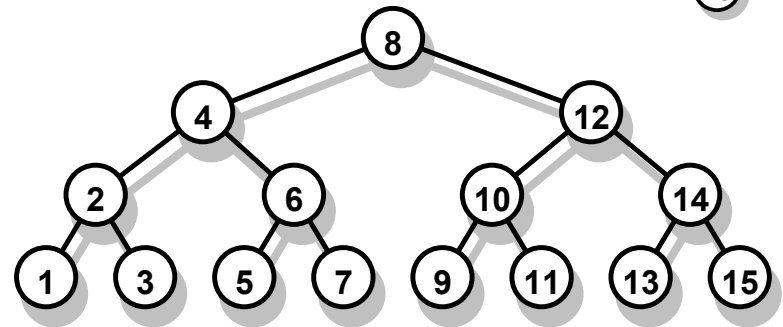
Højden af et Søgetræ ?

Største og Mindste Højde

Indsæt i stigende rækkefølge
- Højde n



Perfekt balanceret
(mindst mulig højde)
- Højde $1 + \lfloor \log n \rfloor$



Tilfældige Indsættelser i et Søgetræ

Algoritme:

Indsæt n elementer i tilfældige rækkefølge

- Ligesom ved QuickSort argumenteres at den **forventede dybde** af et element er $O(\log n)$
- Den forventede **højde af træet** er $O(\log n)$, dvs. *alle knuder* har forventet dybde $O(\log n)$ [CLRS, Kap. 12.4]

Balancerede Søgetræer

Ved **balancerede søgetræer** omstruktureres træet løbende så søgetiderne forbliver **$O(\log n)$**

- AVL-træer
- BB[α]-træer
- Splay træer
- Lokalt balancerede træer
- Rød-sorter træer
- Randomized trees
- (2,3)-træer
- (2,4)-træer
- B-træer
- Vægtbalancerede B-træer
- ...

Algoritmer og Datastrukturer 1

Rød-Sorte Søgetræer [CLRS, kapitel 13]



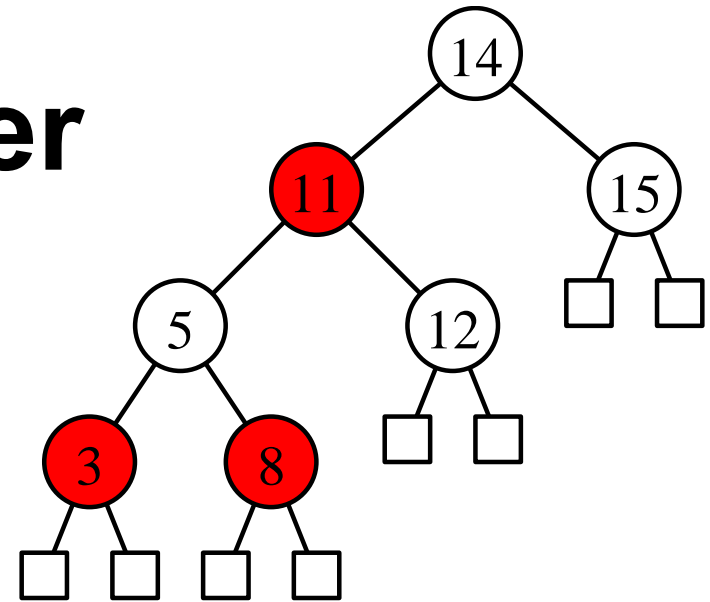
Gerth Stølting Brodal

Aarhus Universitet

Rød-Sorte Søgetræer

Mål

Søgetræer med dybde $O(\log n)$



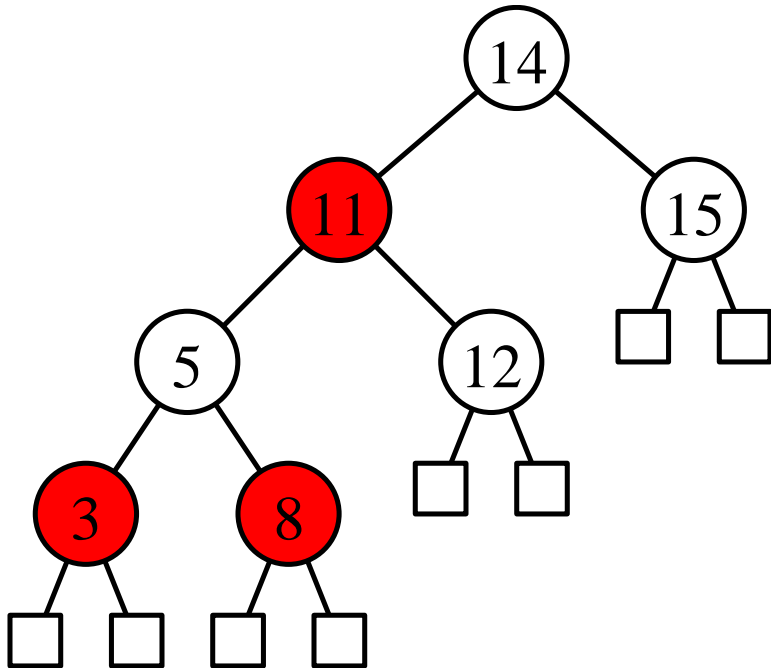
Invarianter

- Hver knude er enten **RØD** eller **SORT**
- Hver **RØD** knude har en **SORT** far
- Alle stier fra roden til et eksternt blad har **samme antal sorte knuder**

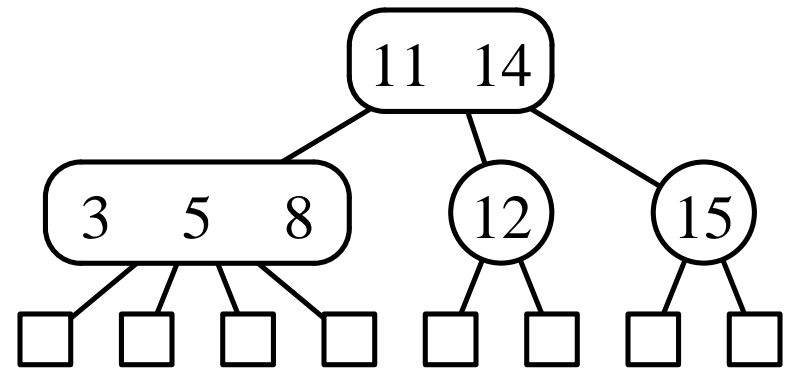
Sætning

Et rød-sort træ har højde $\leq 2 \cdot \log(n+1)$

Rød-Sorte vs (2-4)-træer



Rød-sort træ

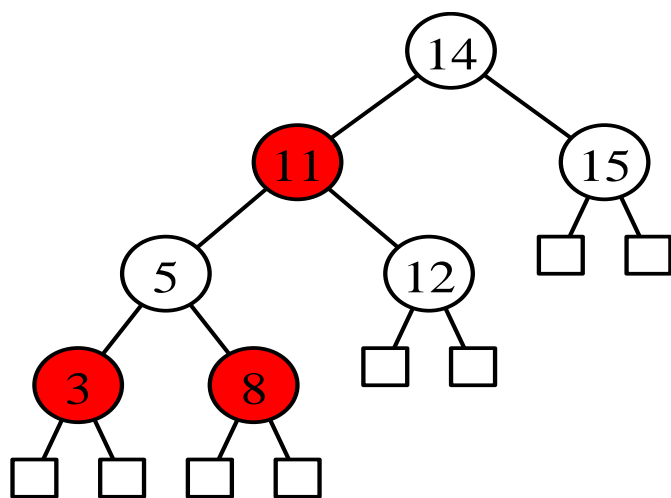


(2,4)-træ

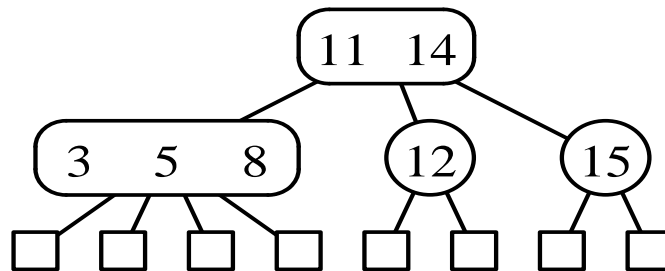
(2,4)-træ \equiv **rød-sort træ** hvor alle røde knuder er slået sammen med faderen

Egenskaber ved (2,4)-træer

- Alle interne knuder har mellem **2 og 4 børn**
- Knude med **i børn** gemmer **$i-1$ elementer**
- Stier fra roden til et eksternt blad er lige lange

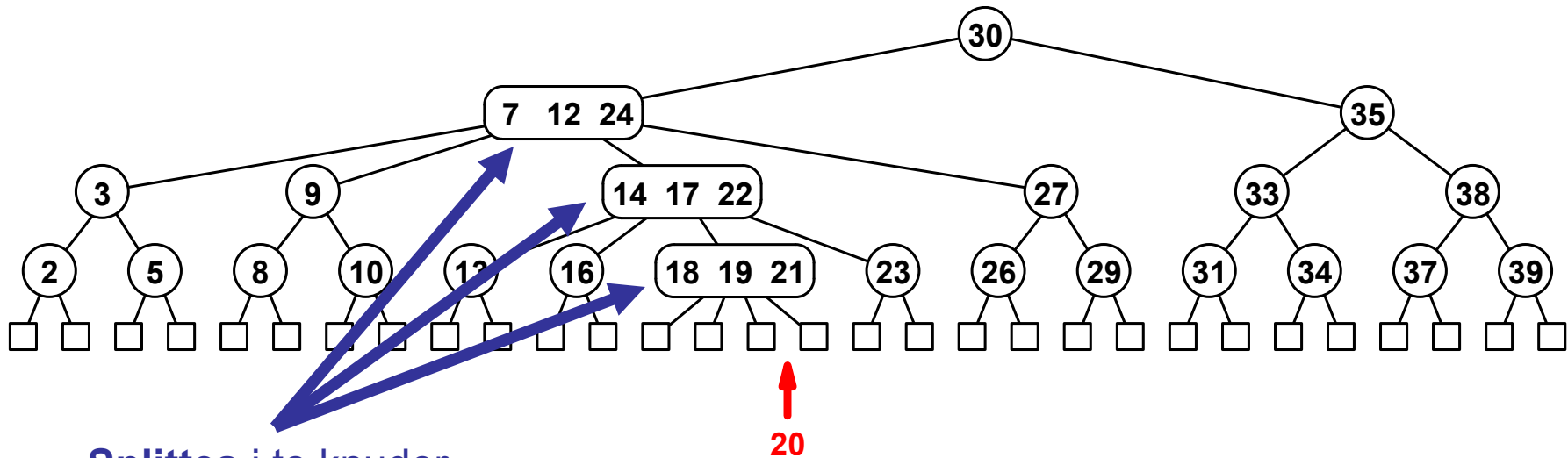


Rød-sort træ

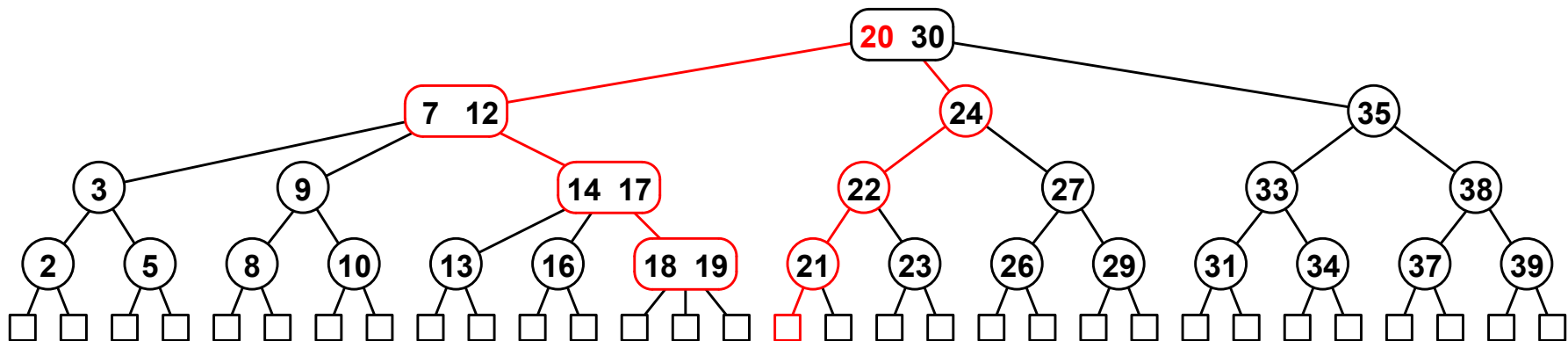


(2,4)-træ

Indsættelse i et (2,4)-træ

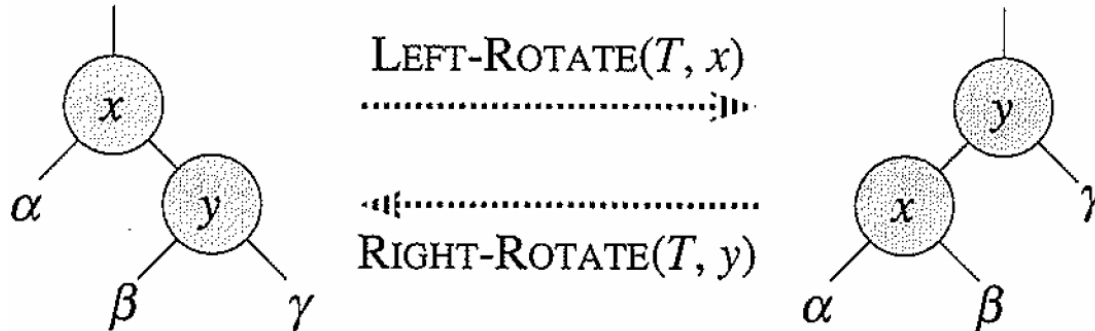


Splittes i to knuder
og midterste nøgle
flyttes et niveau op



Opdateringer af Rød-Sorte Træer

Rotationer



LEFT-ROTATE(T, x)

- 1 $y \leftarrow \text{right}[x]$ \triangleright Set y .
- 2 $\text{right}[x] \leftarrow \text{left}[y]$ \triangleright Turn y 's left subtree into x 's right subtree.
- 3 **if** $\text{left}[y] \neq \text{nil}[T]$
- 4 **then** $p[\text{left}[y]] \leftarrow x$
- 5 $p[y] \leftarrow p[x]$ \triangleright Link x 's parent to y .
- 6 **if** $p[x] = \text{nil}[T]$
- 7 **then** $\text{root}[T] \leftarrow y$
- 8 **else if** $x = \text{left}[p[x]]$
- 9 **then** $\text{left}[p[x]] \leftarrow y$
- 10 **else** $\text{right}[p[x]] \leftarrow y$
- 11 $\text{left}[y] \leftarrow x$ \triangleright Put x on y 's left.
- 12 $p[x] \leftarrow y$

Insert

RB-INSERT(T, z)

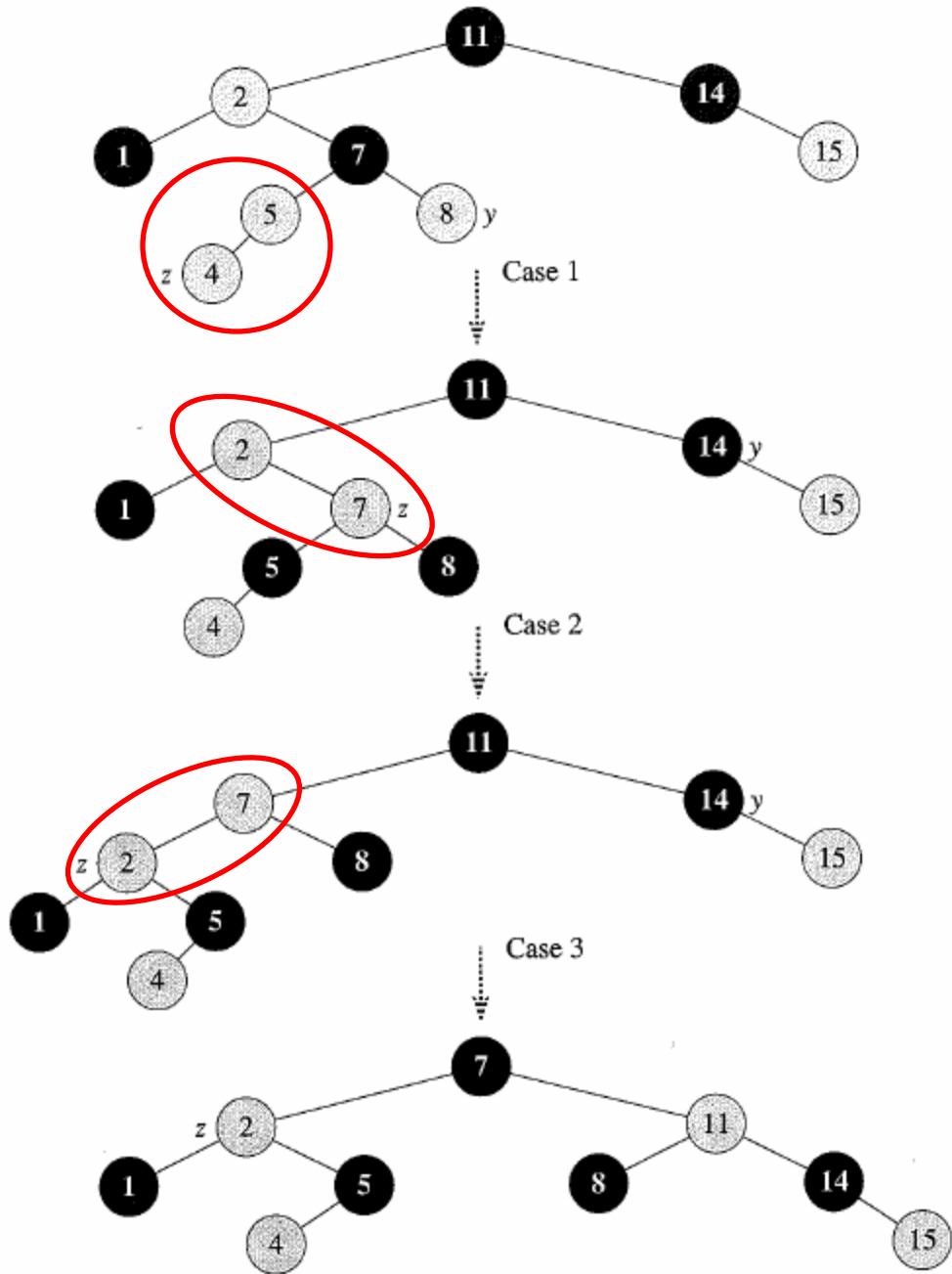
Ubalanceret
indsættelse

```
1   $y \leftarrow nil[T]$ 
2   $x \leftarrow root[T]$ 
3  while  $x \neq nil[T]$ 
4      do  $y \leftarrow x$ 
5          if  $key[z] < key[x]$ 
6              then  $x \leftarrow left[x]$ 
7              else  $x \leftarrow right[x]$ 
8   $p[z] \leftarrow y$ 
9  if  $y = nil[T]$ 
10     then  $root[T] \leftarrow z$ 
11     else if  $key[z] < key[y]$ 
12         then  $left[y] \leftarrow z$ 
13         else  $right[y] \leftarrow z$ 
14   $left[z] \leftarrow nil[T]$ 
15   $right[z] \leftarrow nil[T]$ 
16   $color[z] \leftarrow RED$ 
17  RB-INSERT-FIXUP( $T, z$ )
```

RB-INSERT-FIXUP(T, z)

```
1  while color[p[z]] = RED
2      do if p[z] = left[p[p[z]]]
3          then y ← right[p[p[z]]]
4              if color[y] = RED
5                  then color[p[z]] ← BLACK           ▷ Case 1
6                     color[y] ← BLACK                ▷ Case 1
7                     color[p[p[z]]] ← RED            ▷ Case 1
8                     z ← p[p[z]]                     ▷ Case 1
9              else if z = right[p[z]]
10                 then z ← p[z]                       ▷ Case 2
11                    LEFT-ROTATE( $T, z$ )                ▷ Case 2
12                 color[p[z]] ← BLACK                 ▷ Case 3
13                 color[p[p[z]]] ← RED                ▷ Case 3
14                 RIGHT-ROTATE( $T, p[p[z]]$ )           ▷ Case 3
15             else (same as then clause
16                   with “right” and “left” exchanged)
```

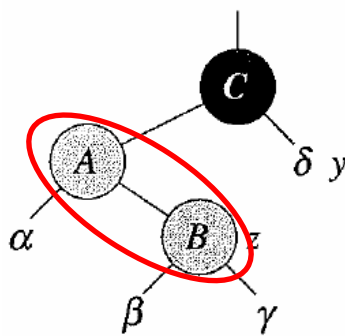
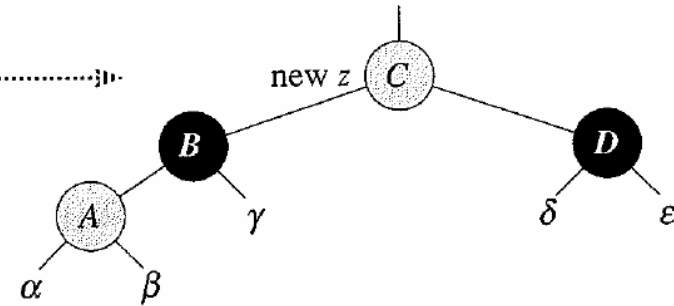
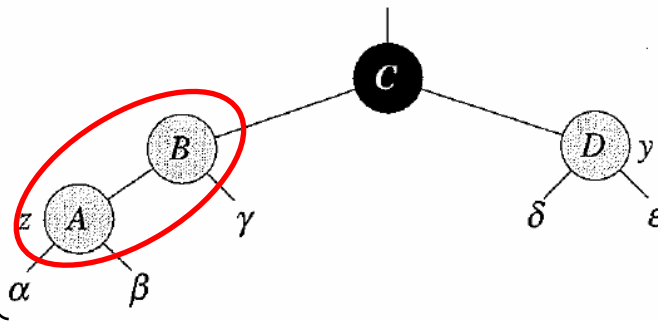
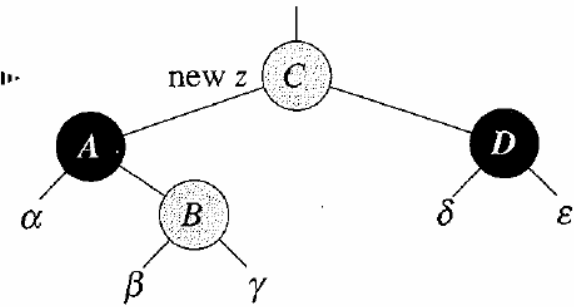
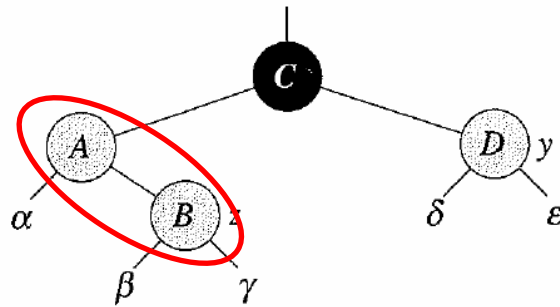
16 color[root[T]] ← BLACK



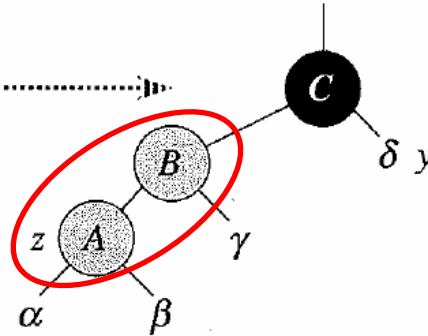
Indsættelse i rød-sort trær: rebalancering

Omfarv c og dens to røde børn

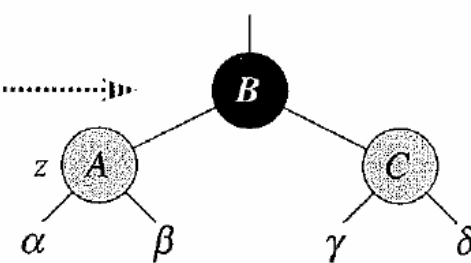
Case 1



Case 2



Case 3



Delete

RB-DELETE(T, z)

Ubalanceret
slettelse

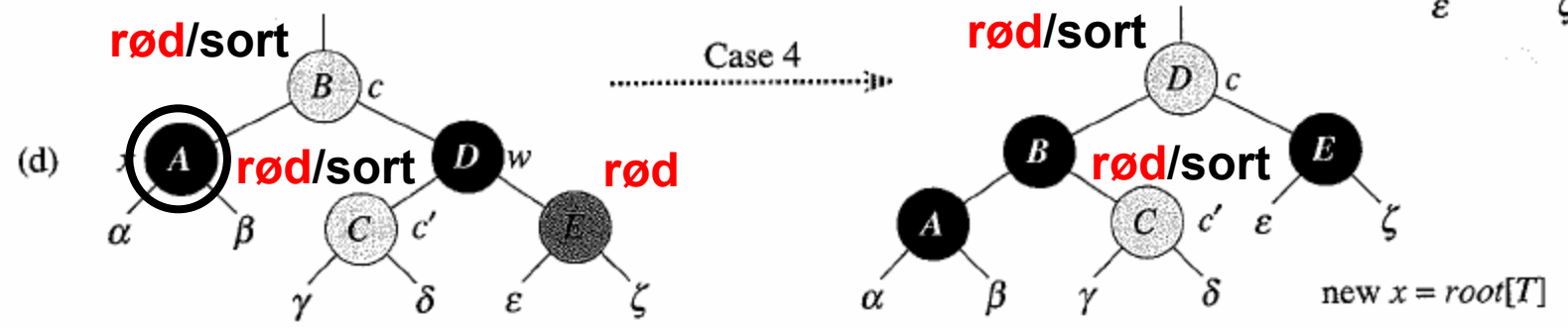
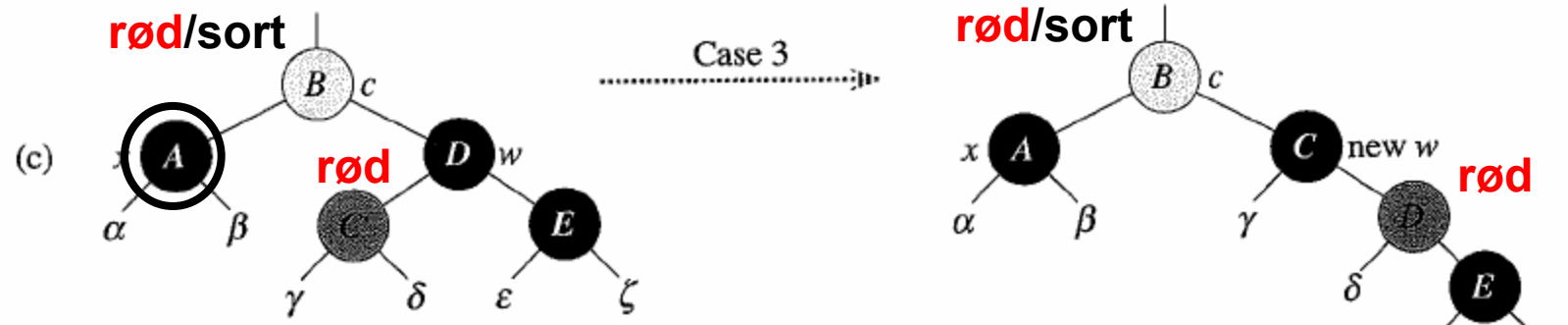
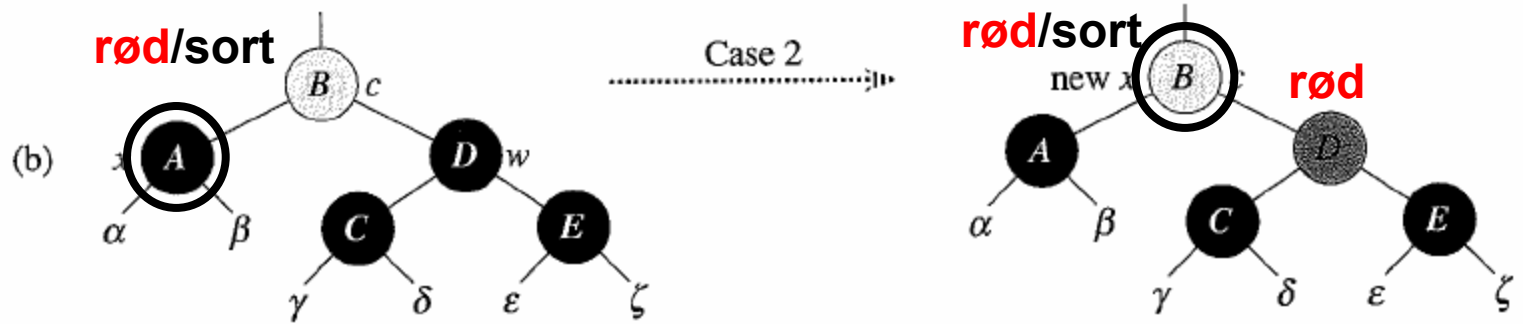
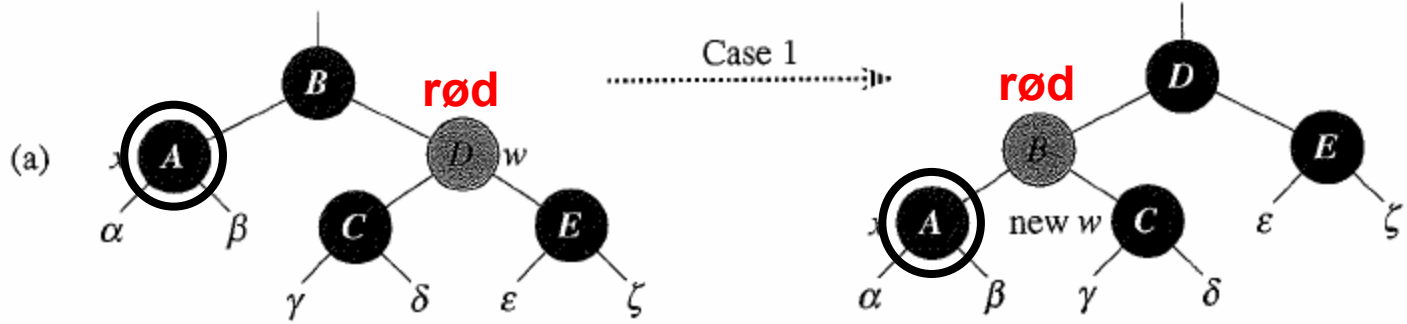
```
1  if  $left[z] = nil[T]$  or  $right[z] = nil[T]$ 
2    then  $y \leftarrow z$ 
3    else  $y \leftarrow TREE-SUCCESSOR(z)$ 
4  if  $left[y] \neq nil[T]$ 
5    then  $x \leftarrow left[y]$ 
6    else  $x \leftarrow right[y]$ 
7   $p[x] \leftarrow p[y]$ 
8  if  $p[y] = nil[T]$ 
9    then  $root[T] \leftarrow x$ 
10 else if  $y = left[p[y]]$ 
11       then  $left[p[y]] \leftarrow x$ 
12       else  $right[p[y]] \leftarrow x$ 
13 if  $y \neq z$ 
14   then  $key[z] \leftarrow key[y]$ 
15       copy  $y$ 's satellite data into  $z$ 
16 if  $color[y] = BLACK$ 
17   then RB-DELETE-FIXUP( $T, x$ )
18 return  $y$ 
```

RB-DELETE-FIXUP(T, x)

```

1  while  $x \neq \text{root}[T]$  and  $\text{color}[x] = \text{BLACK}$ 
2      do if  $x = \text{left}[p[x]]$ 
3          then  $w \leftarrow \text{right}[p[x]]$ 
4              if  $\text{color}[w] = \text{RED}$ 
5                  then  $\text{color}[w] \leftarrow \text{BLACK}$                                 ▷ Case 1
6                       $\text{color}[p[x]] \leftarrow \text{RED}$                                 ▷ Case 1
7                      LEFT-ROTATE( $T, p[x]$ )                                ▷ Case 1
8                       $w \leftarrow \text{right}[p[x]]$                                 ▷ Case 1
9              if  $\text{color}[\text{left}[w]] = \text{BLACK}$  and  $\text{color}[\text{right}[w]] = \text{BLACK}$ 
10                 then  $\text{color}[w] \leftarrow \text{RED}$                                 ▷ Case 2
11                      $x \leftarrow p[x]$                                 ▷ Case 2
12                 else if  $\text{color}[\text{right}[w]] = \text{BLACK}$ 
13                     then  $\text{color}[\text{left}[w]] \leftarrow \text{BLACK}$                                 ▷ Case 3
14                          $\text{color}[w] \leftarrow \text{RED}$                                 ▷ Case 3
15                         RIGHT-ROTATE( $T, w$ )                                ▷ Case 3
16                          $w \leftarrow \text{right}[p[x]]$                                 ▷ Case 3
17                      $\text{color}[w] \leftarrow \text{color}[p[x]]$                                 ▷ Case 4
18                      $\text{color}[p[x]] \leftarrow \text{BLACK}$                                 ▷ Case 4
19                      $\text{color}[\text{right}[w]] \leftarrow \text{BLACK}$                                 ▷ Case 4
20                     LEFT-ROTATE( $T, p[x]$ )                                ▷ Case 4
21                      $x \leftarrow \text{root}[T]$                                 ▷ Case 4
22                 else (same as then clause with “right” and “left” exchanged)
23  $\text{color}[x] \leftarrow \text{BLACK}$ 

```



Dynamisk Ordbog :

Rød-Sorte Træer

Search(S,x)	$O(\log n)$
Insert(S,x)	
Delete(S,x)	