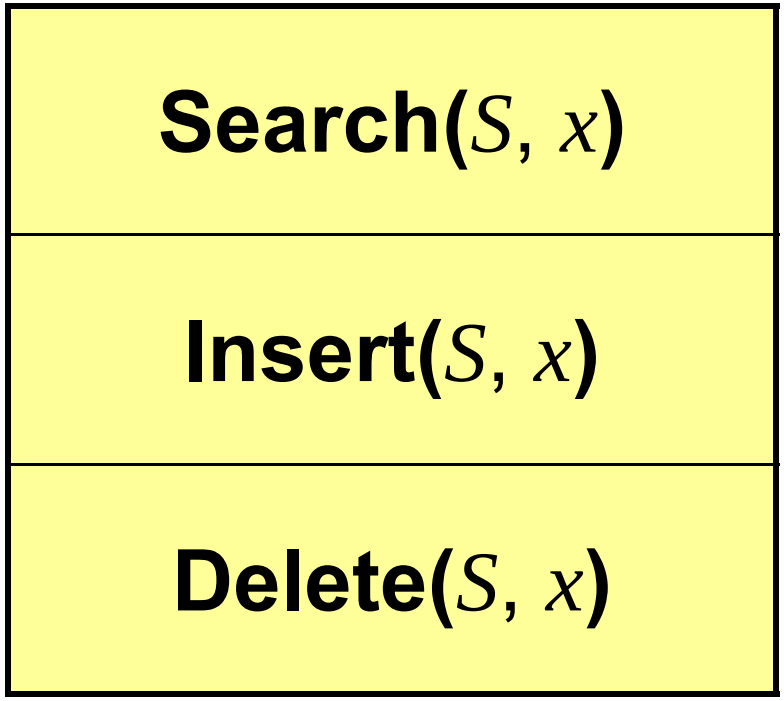
Algoritmer og Datastrukturer 1

Gerth Stølting Brodal

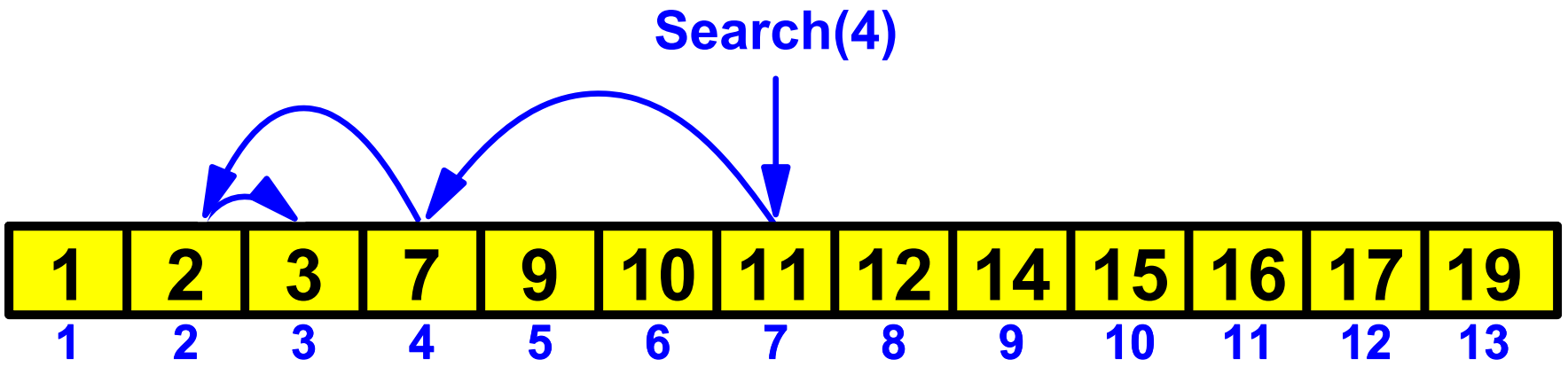
Binære Søgetræer [CLRS, kapitel 12]

AARHUS UNIVERSITET

Abstrakt Datastruktur: Ordbog

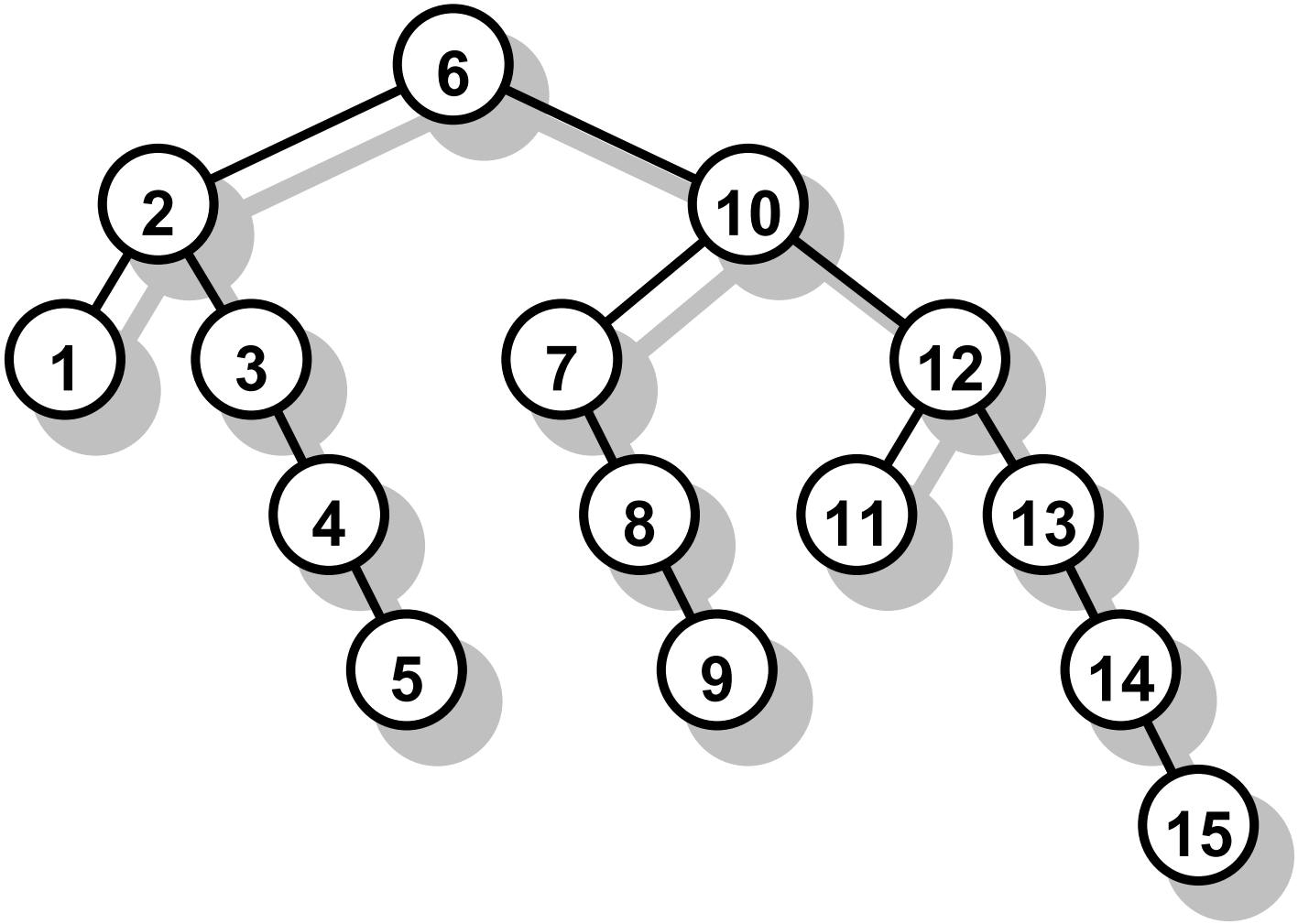


(Statisk) Ordbog : Sorteret Array



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

Søgetræ



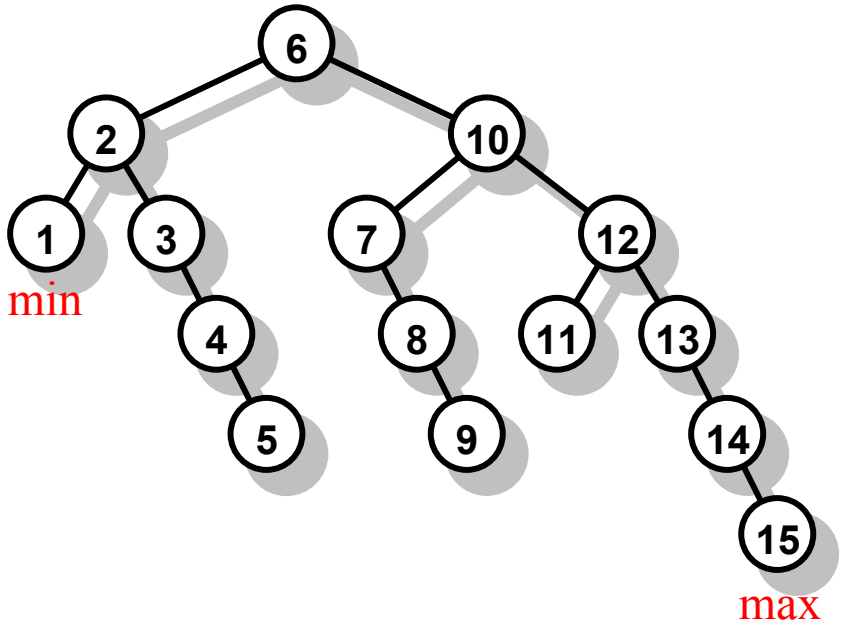
Invariant For alle knuder er elementerne i venstre (højre) undertræ mindre (større) end eller lig med elementet i knuden

Søgetræs søgninger

```
TREE-SEARCH(x, k)
1  if x == NIL or k == x.key
2        return x
3  if k < x.key
4        return TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)
```

```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2        if k < x.key
3              x = x.left
4        else x = x.right
5  return x
```

```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2        return TREE-MINIMUM(x.right)
3  y = x.p
4  while y ≠ NIL and x == y.right
5        x = y
6        y = y.p
7  return y
```




```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2        INORDER-TREE-WALK(x.left)
3        print x.key
4        INORDER-TREE-WALK(x.right)
```

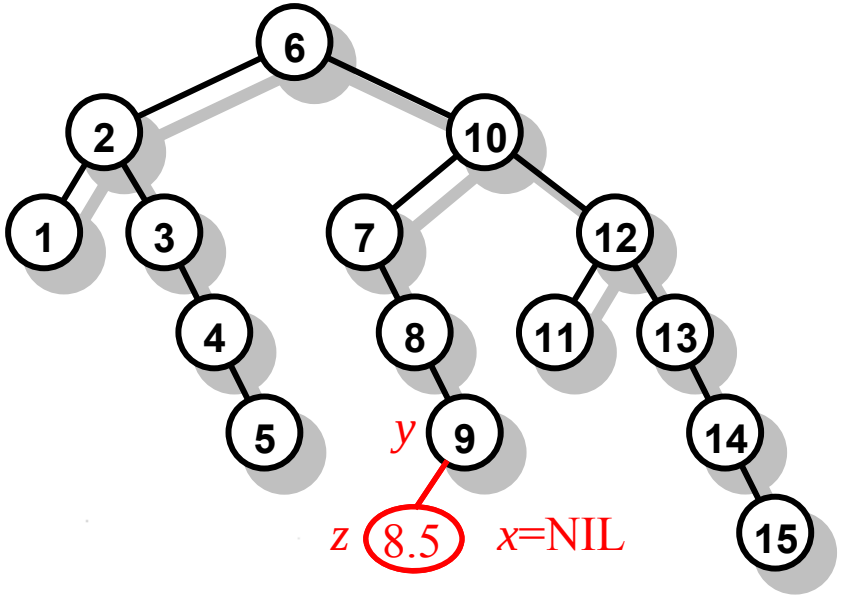
```
TREE-MINIMUM(x)
1  while x.left ≠ NIL
2        x = x.left
3  return x
```

```
TREE-MAXIMUM(x)
1  while x.right ≠ NIL
2        x = x.right
3  return x
```

Indsættelse i Søgetræ

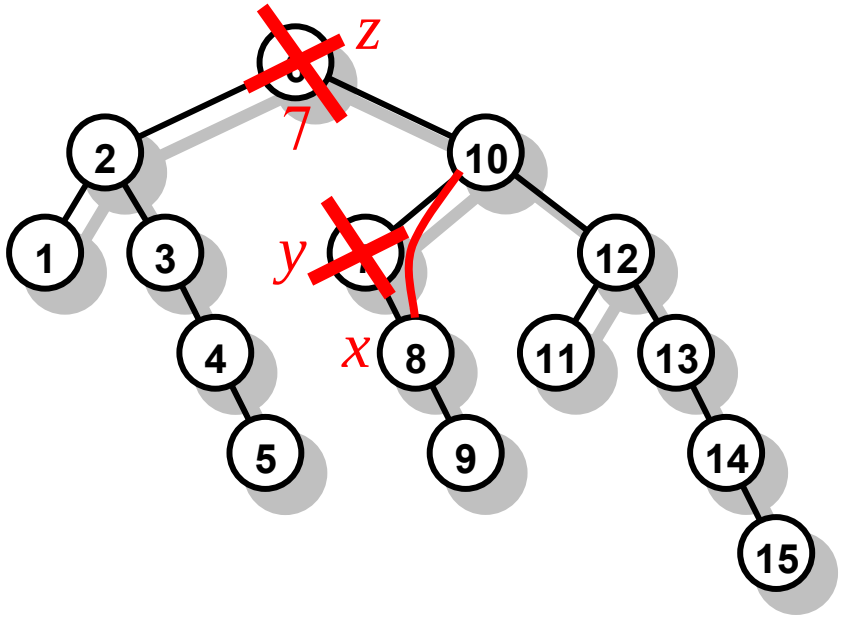
```
TREE-INSERT(T, z)
 1   y = NIL
 2   x = T.root
 3   while x ≠ NIL
 4         y = x
 5         if z.key < x.key
 6               x = x.left
 7         else x = x.right
 8   z.p = y
 9   if y == NIL
10         T.root = z          // tree T was empty
11   elseif z.key < y.key
12         y.left = z
13   else y.right = z
```

Iterativ søgning



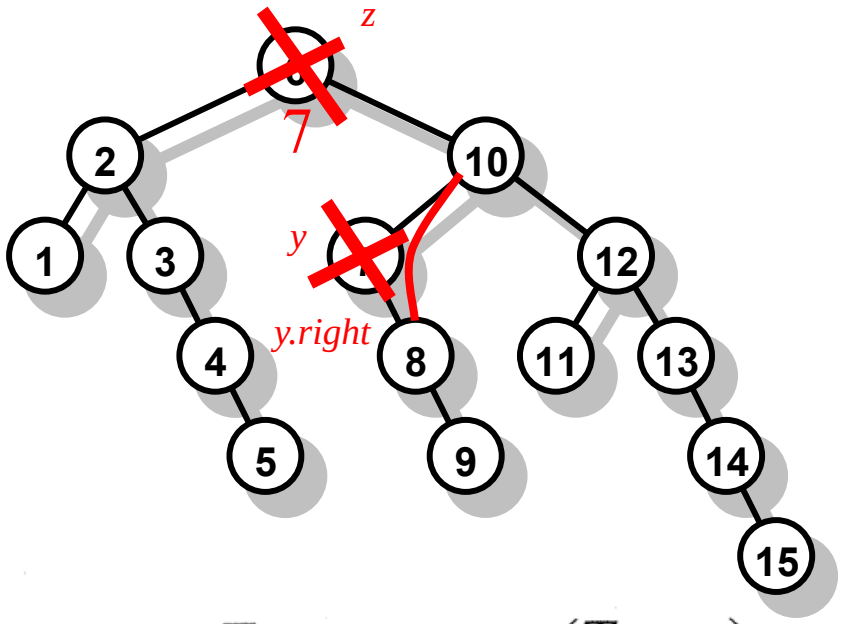
Slettelse fra Søgetræ [CLRS, Ed. 2]

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




```
TREE-SUCCESSOR(x)
1  if right[x] ≠ NIL
2     then return TREE-MINIMUM(right[x])
3  y ← p[x]
4  while y ≠ NIL and x = right[y]
5        do x ← y
6           y ← p[y]
7  return y
```

Slettelse fra Søgetræ [CLRS, Ed. 3]




```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2       TRANSPLANT(T, z, z.right)
 3  elseif z.right == NIL
 4       TRANSPLANT(T, z, z.left)
 5  else y = TREE-MINIMUM(z.right)
 6       if y.p ≠ z
 7            TRANSPLANT(T, y, y.right)
 8            y.right = z.right
 9            y.right.p = y
10       TRANSPLANT(T, z, y)
11       y.left = z.left
12       y.left.p = y
```

```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2       T.root = v
3  elseif u == u.p.left
4       u.p.left = v
5  else u.p.right = v
6  if v ≠ NIL
7       v.p = u.p
```

Søgetræer

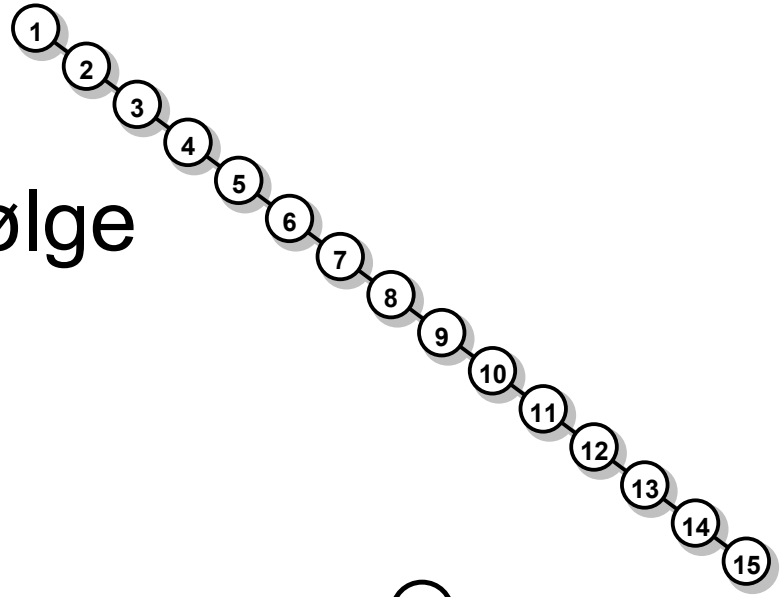
Inorder-Tree-Walk	$O(n)$
Tree-Search **Iterative-Tree-Search**	$O(h)$
Tree-Minimum **Tree-Maximum**	$O(h)$
Tree-Insert	$O(h)$
Tree-Delete	$O(h)$

h = højden af træet, n = antal elementer

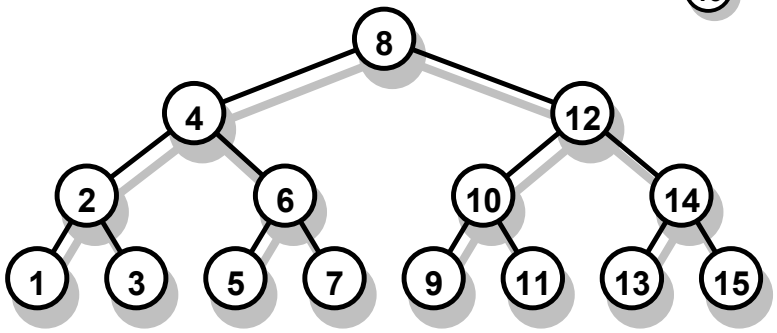
Højden af et Søgetræ ?

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-sorte træer

- Randomized trees
- (2,3)-træer
- (2,4)-træer
- B-træer
- Vægtbalancerede B-træer
- ...

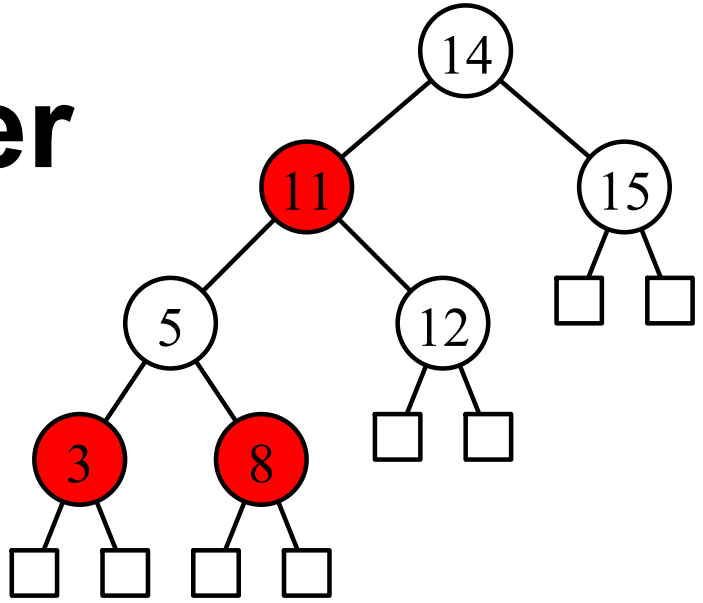
Algoritmer og Datastrukturer 1

Gerth Stølting Brodal

Rød-Sorte Søgetræer [CLRS, kapitel 13]

AARHUS UNIVERSITET

Rød-Sorte Søgetræer



Mål

Søgetræer med dybde $O(\log n)$

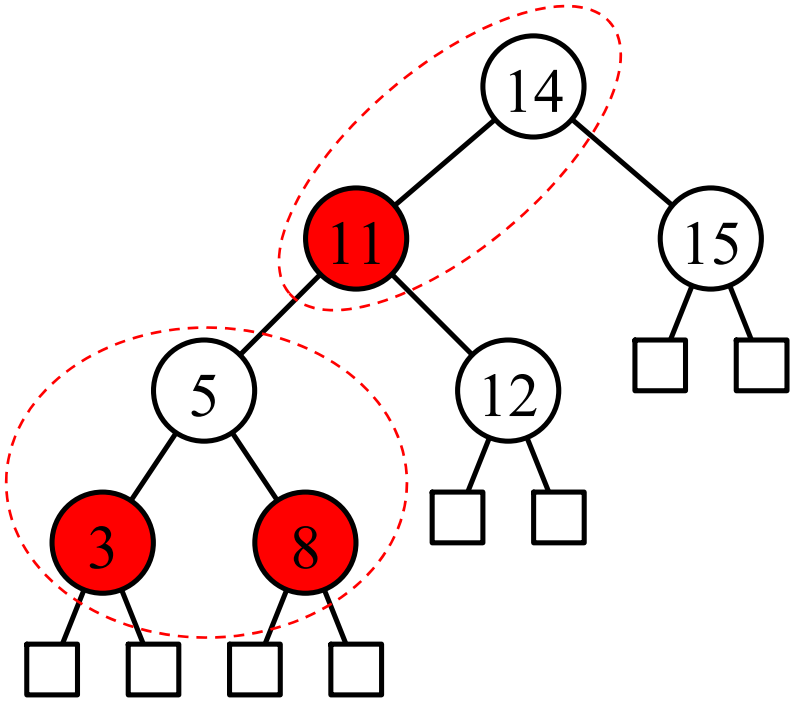
Invarianter

- 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**

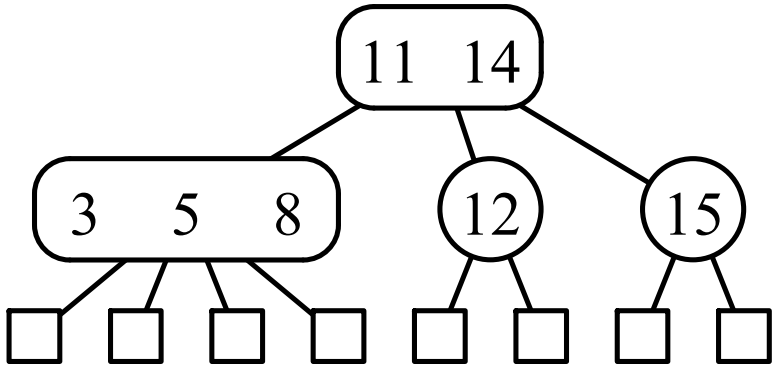
Sætning

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

Rød-Sorte vs (2-4)-træer



Rød-sort træ

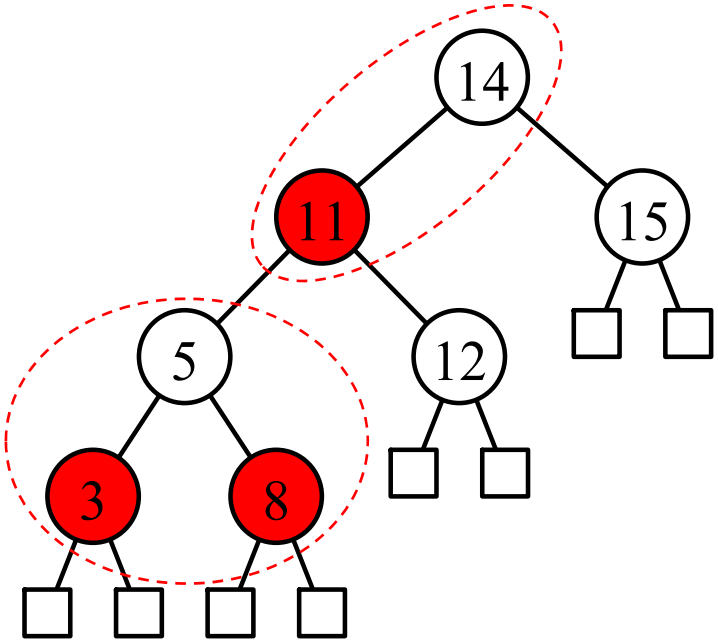


(2,4)-træ

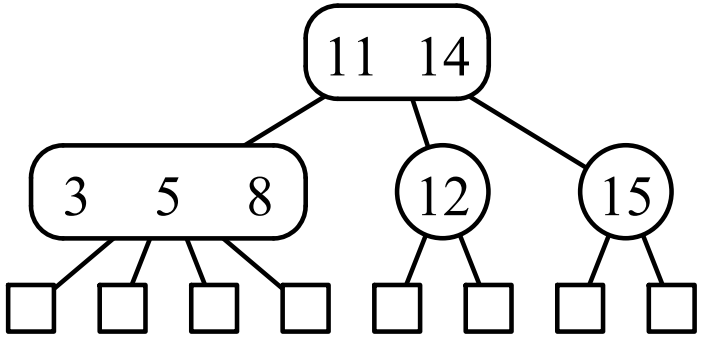
(2,4)-træ ≡ **rød-sort træ** hvor alle røde knuder er slået sammen med faderen

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

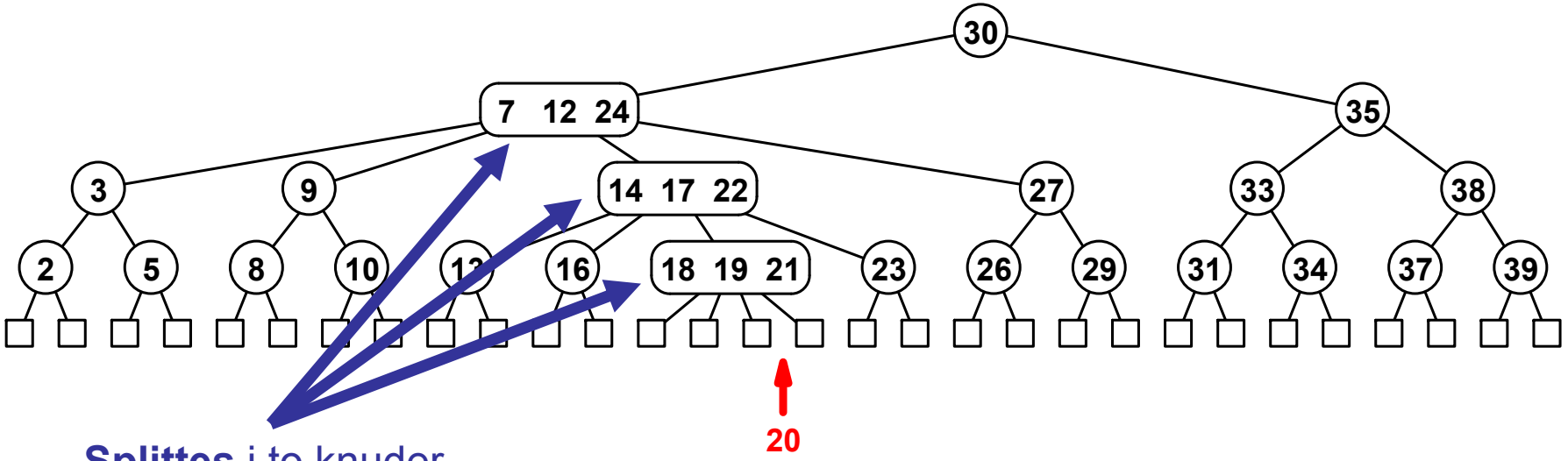


Rød-sort træ

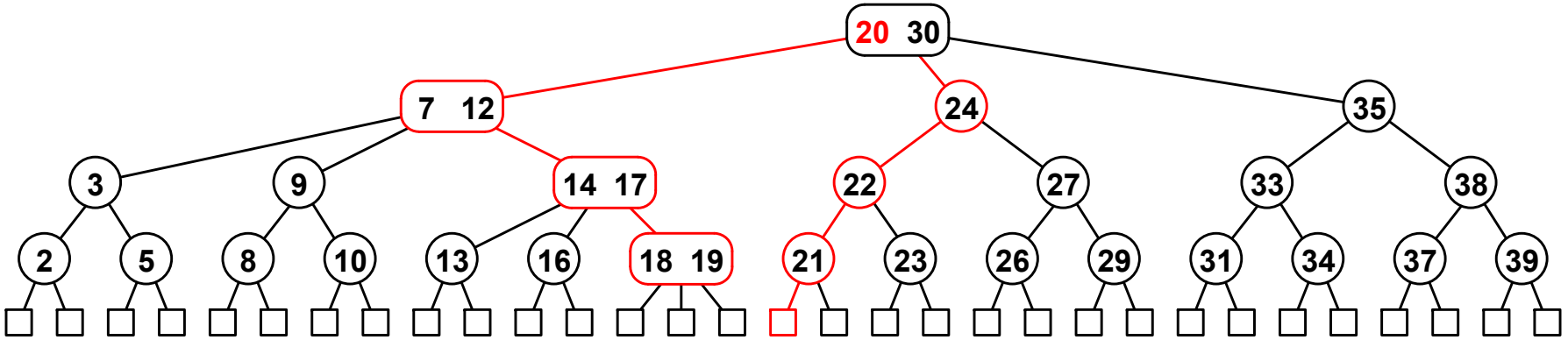


(2,4)-træ

Indsættelse i et (2,4)-træ

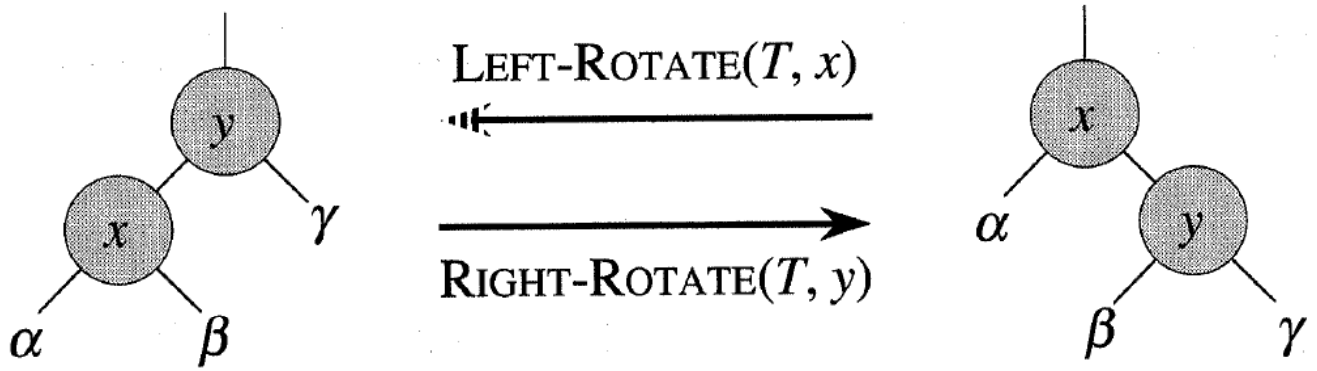


Splittes i to knuder
og midterste nøgle
flyttes et niveu op



Opdateringer af Rød-Sorte Træer

Rotationer



LEFT-ROTATE(T, x)

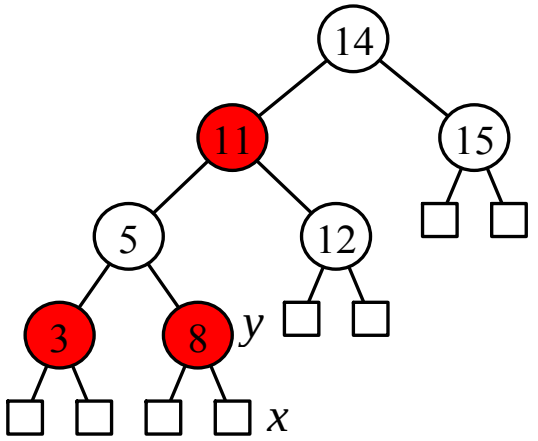
```
 1  y = x.right                    // set y
 2  x.right = y.left               // turn y's left subtree into x's right subtree
 3  if y.left ≠ T.nil
 4       y.left.p = x
 5  y.p = x.p                      // link x's parent to y
 6  if x.p == T.nil
 7       T.root = y
 8  elseif x == x.p.left
 9       x.p.left = y
10  else x.p.right = y
11  y.left = x                     // put x on y's left
12  x.p = y
```

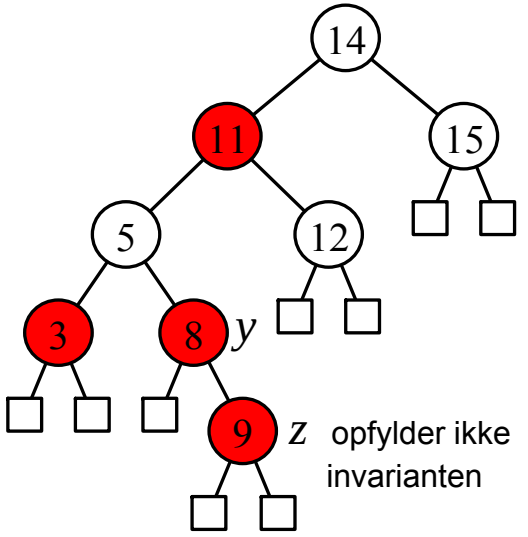
Insert

Ubalanceret indsættelse

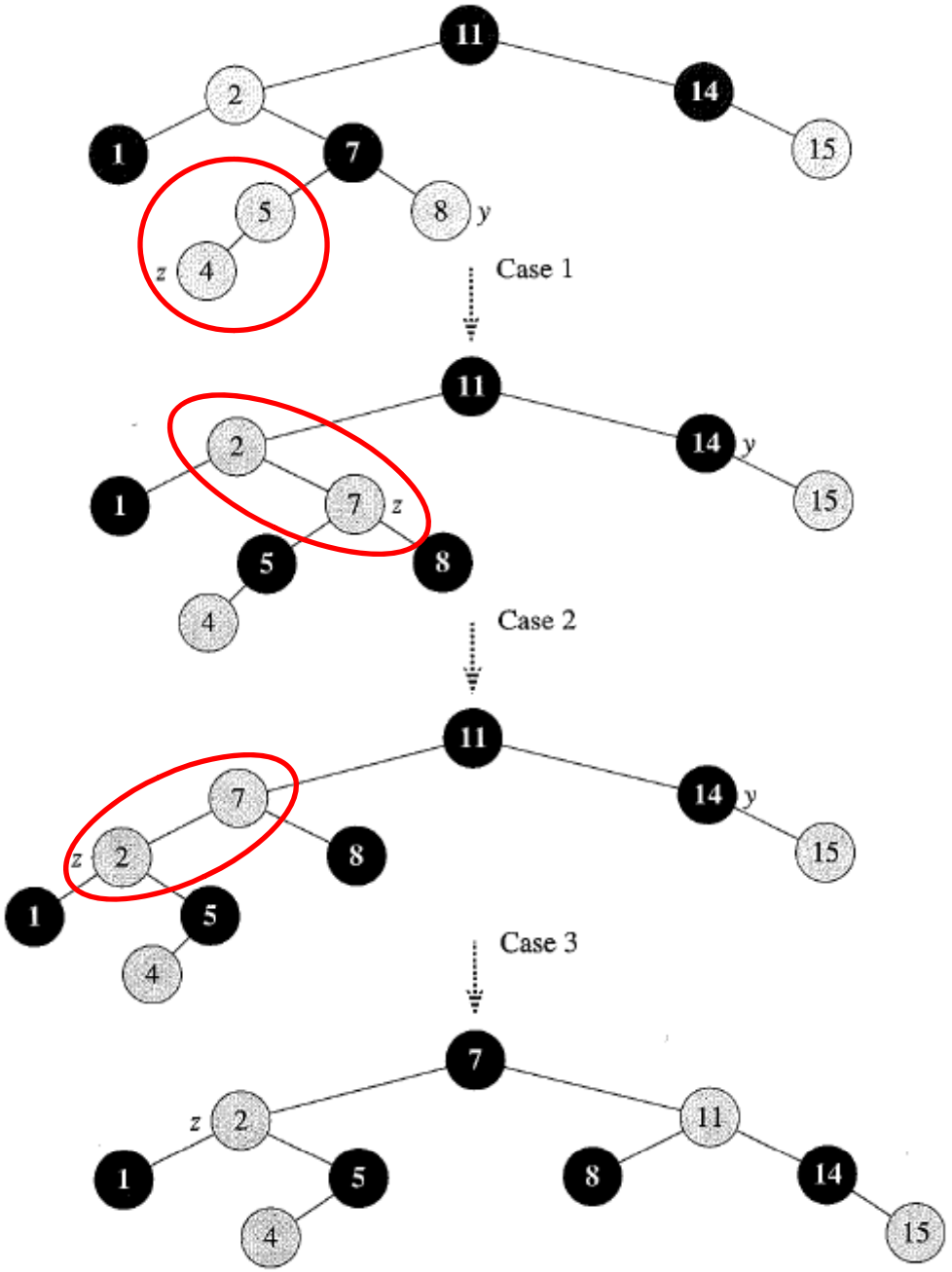
```
RB-INSERT(T, z)
 1   y = T.nil
 2   x = T.root
 3   while x ≠ T.nil
 4         y = x
 5         if z.key < x.key
 6               x = x.left
 7         else x = x.right
 8   z.p = y
 9   if y == T.nil
10         T.root = z
11   elseif z.key < y.key
12         y.left = z
13   else y.right = z
14   z.left = T.nil
15   z.right = T.nil
16   z.color = RED
17   RB-INSERT-FIXUP(T, z)
```

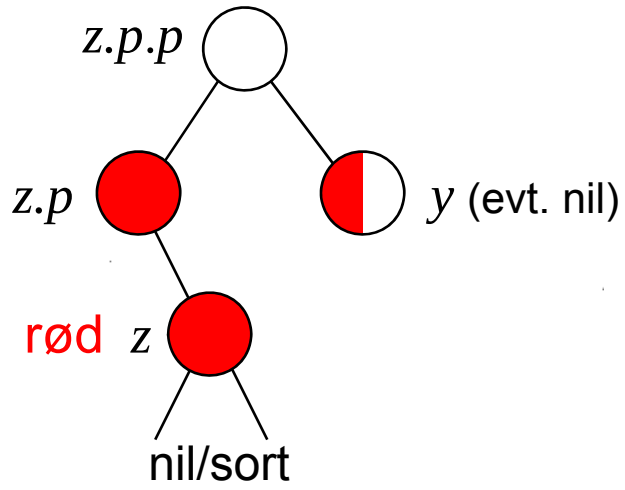
Insert(9)





z opfylder ikke invarianten


11
2
14
1
7
15
5
8
y
z
4
Case 1
11
2
14
y
1
7
z
15
5
8
4
Case 2
11
7
14
y
z
2
8
15
1
5
4
Case 3
7
z
2
11
1
5
8
14
4
15

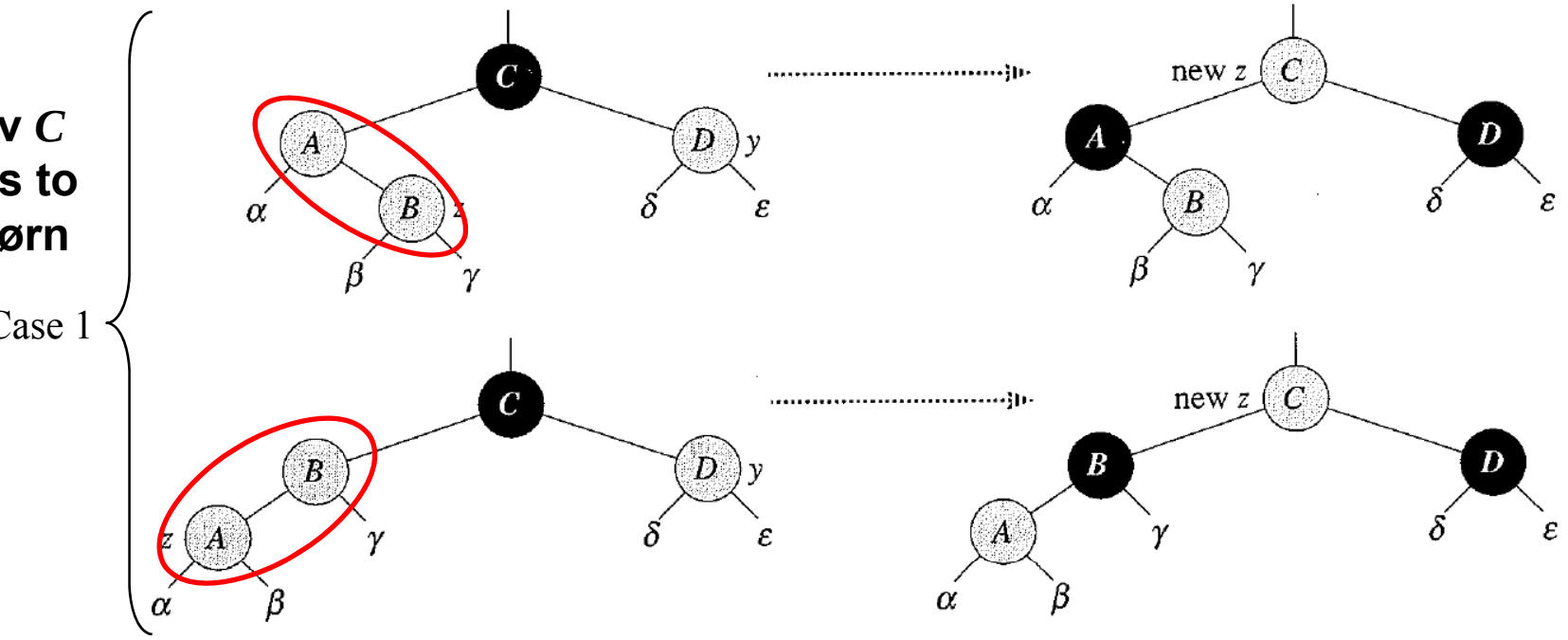


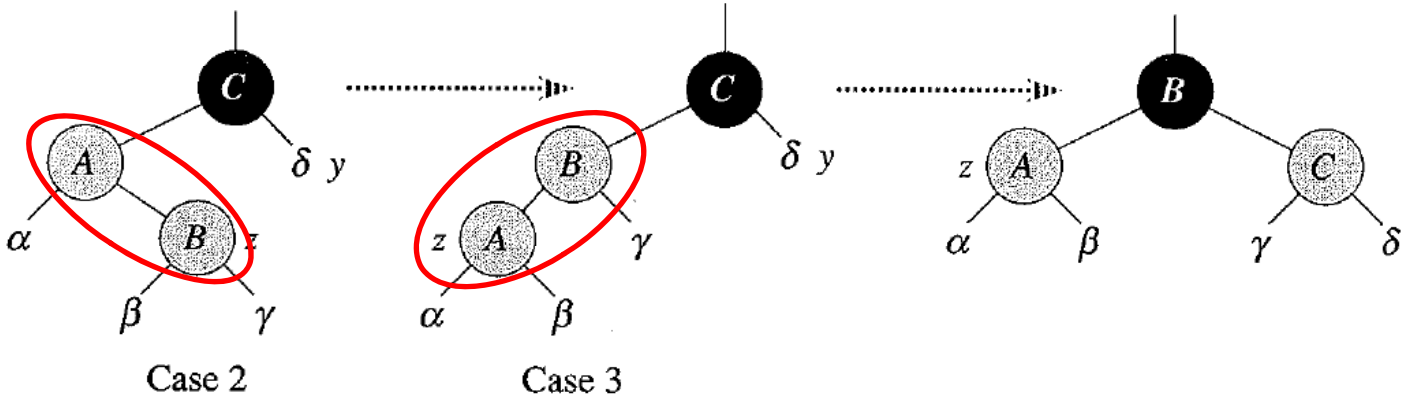

```
RB-INSERT-FIXUP(T, z)
 1  while z.p.color == RED
 2      if z.p == z.p.p.left
 3          y = z.p.p.right
 4          if y.color == RED
 5              z.p.color = BLACK                    // case 1
 6              y.color = BLACK                      // case 1
 7              z.p.p.color = RED                    // case 1
 8              z = z.p.p                            // case 1
 9          else if z == z.p.right
10                  z = z.p                          // case 2
11                  LEFT-ROTATE(T, z)                // case 2
12              z.p.color = BLACK                    // case 3
13              z.p.p.color = RED                    // case 3
14              RIGHT-ROTATE(T, z.p.p)               // case 3
15      else (same as then clause
                with "right" and "left" exchanged)
16  T.root.color = BLACK
```

Indsættelse i rød-sorte træer: rebalancering

Omfarv C og dens to røde børn

Case 1





Case 2

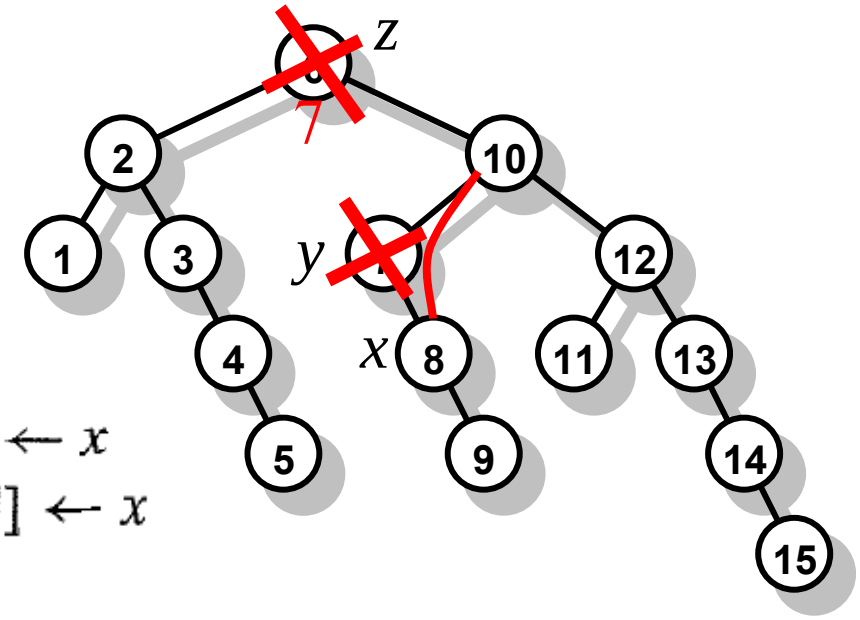
Case 3

Delete

[CLRS, Ed. 2]

```
RB-DELETE(T, z)
 1  if left[z] = nil[T] or right[z] = nil[T]
 2     then y ← z
 3     else y ← TREE-SUCCESSOR(z)
 4  if left[y] ≠ nil[T]
 5     then x ← left[y]
 6     else x ← right[y]
 7  p[x] ← p[y]
 8  if p[y] = nil[T]
 9     then root[T] ← x
10     else if y = left[p[y]]
11             then left[p[y]] ← x
12             else right[p[y]] ← x
13  if y ≠ z
14     then key[z] ← 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
```

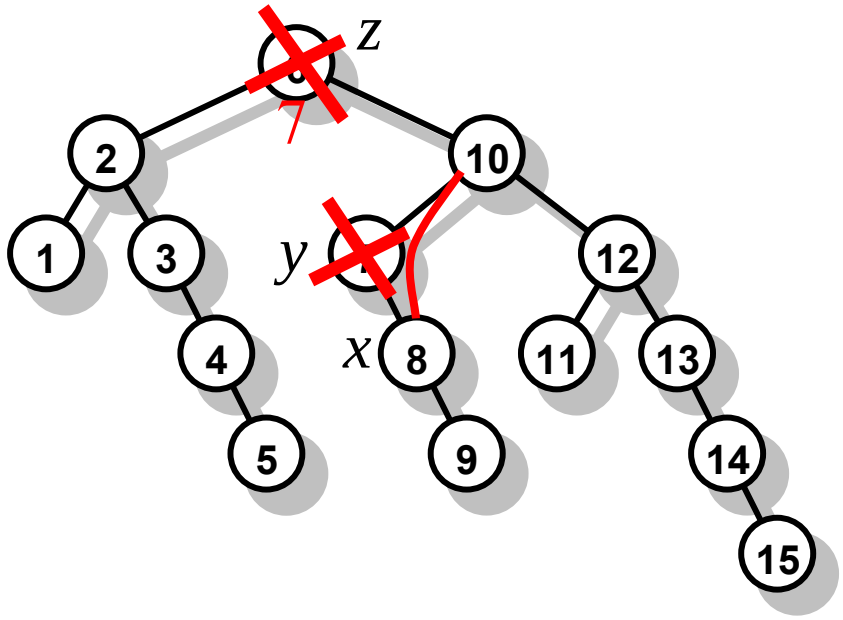
Ubalanceret slettelse



[CLRS, Ed. 3]

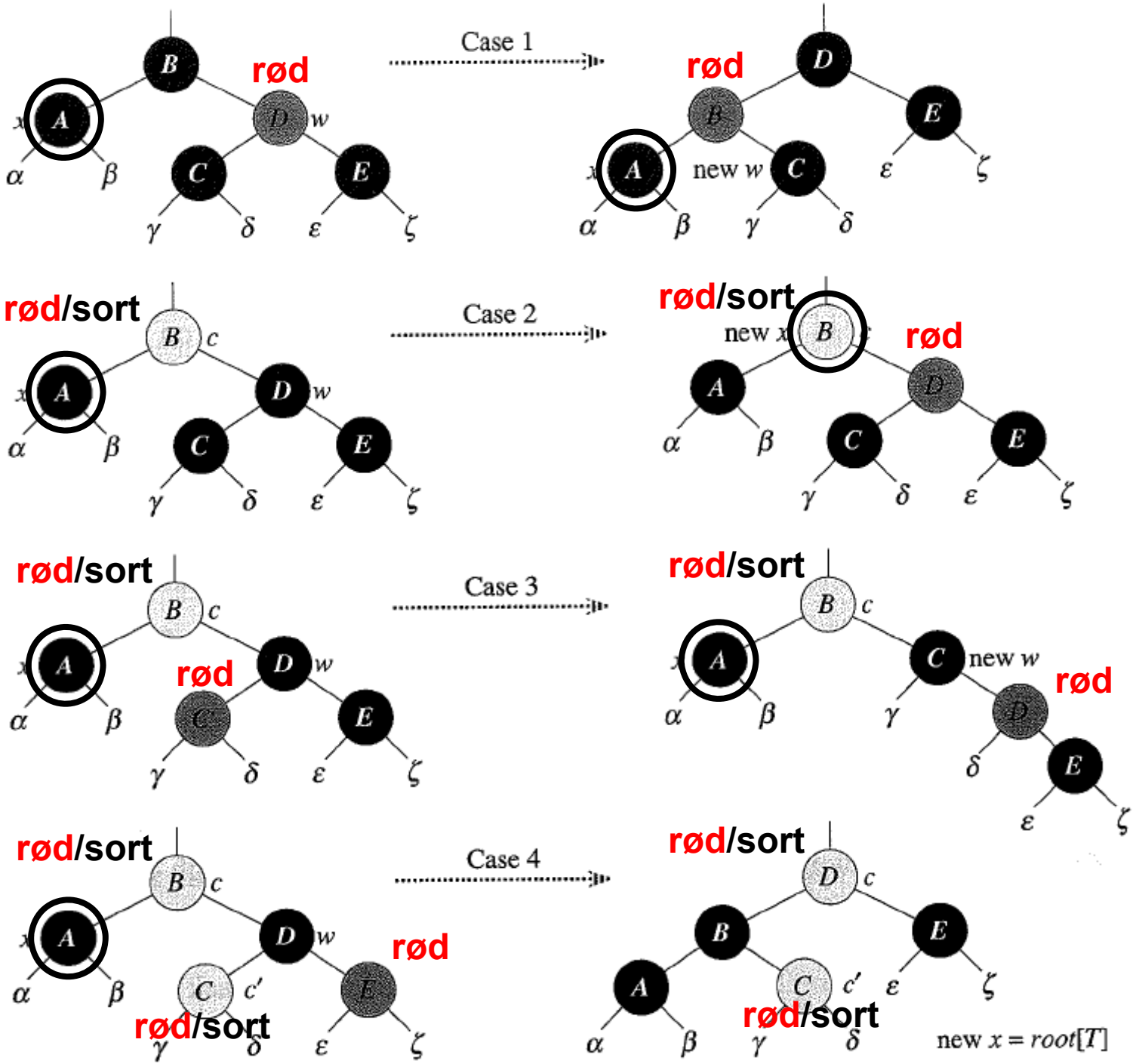
```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

Ubalanceret slettelse




```
RB-TRANSPLANT(T, u, v)
1  if u.p == T.nil
2       T.root = v
3  elseif u == u.p.left
4       u.p.left = v
5  else u.p.right = v
6  v.p = u.p
```

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 4          if w.color == RED
 5              w.color = BLACK                          // case 1
 6              x.p.color = RED                          // case 1
 7              LEFT-ROTATE(T, x.p)                      // case 1
 8              w = x.p.right                            // case 1
 9          if w.left.color == BLACK and w.right.color == BLACK
10              w.color = RED                            // case 2
11              x = x.p                                  // case 2
12          else if w.right.color == BLACK
13                  w.left.color = BLACK                 // case 3
14                  w.color = RED                        // case 3
15                  RIGHT-ROTATE(T, w)                   // case 3
16                  w = x.p.right                        // case 3
17              w.color = x.p.color                      // case 4
18              x.p.color = BLACK                        // case 4
19              w.right.color = BLACK                    // case 4
20              LEFT-ROTATE(T, x.p)                      // case 4
21              x = T.root                               // case 4
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```



Case 1 → 2,3,4

Case 2 → Problemet x flyttet tættere på roden

Case 3 → 4

Case 4 → Færdig

Dynamisk Ordbog : Rød-Sorte Træer

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