CHAPTER 10, 11

Efficient Binary/Multiway Search Trees

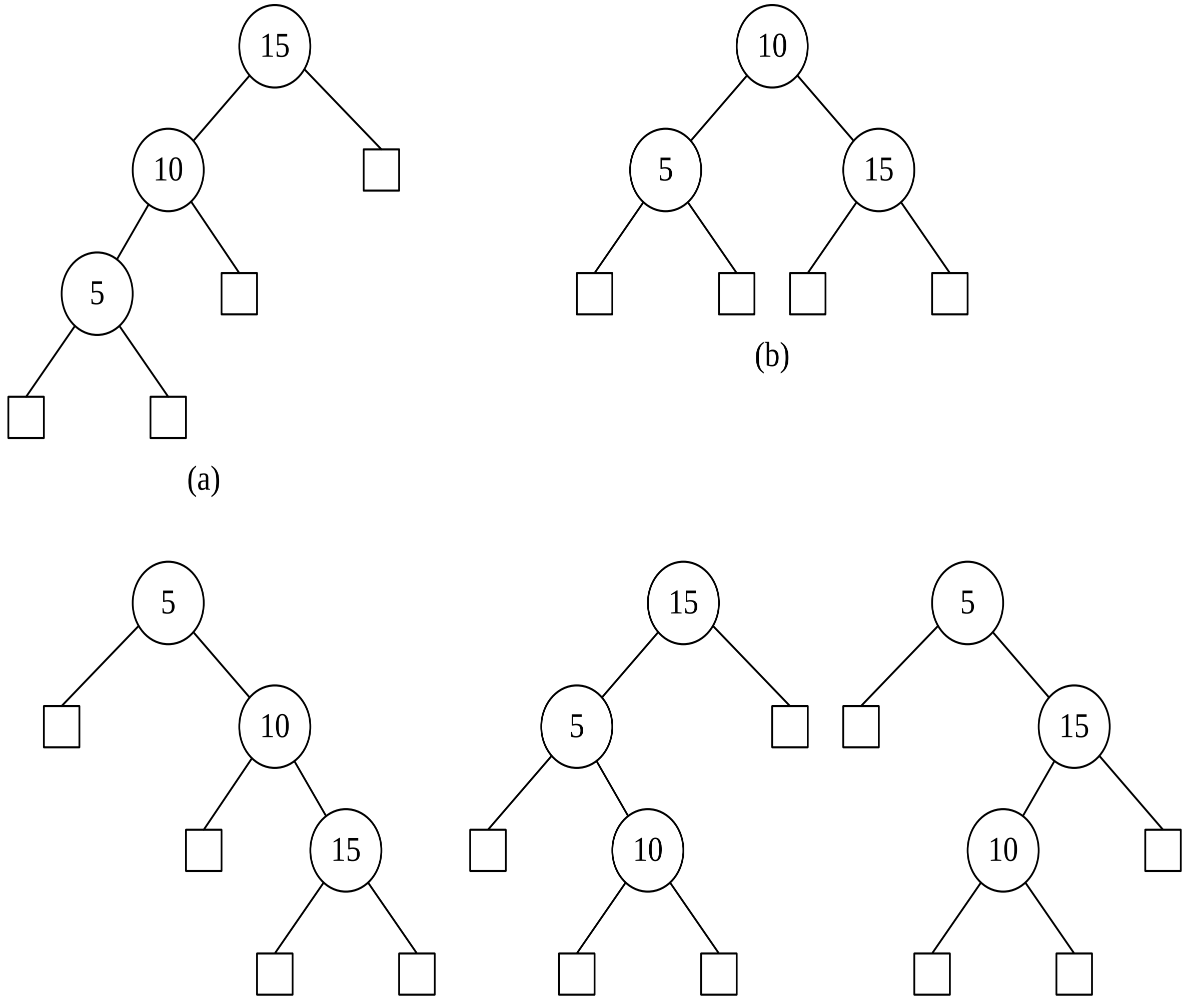
All the programs in this file are selected from

Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed
"Fundamentals of Data Structures in C /2nd Edition",
Silicon Press, 2008.

Outline

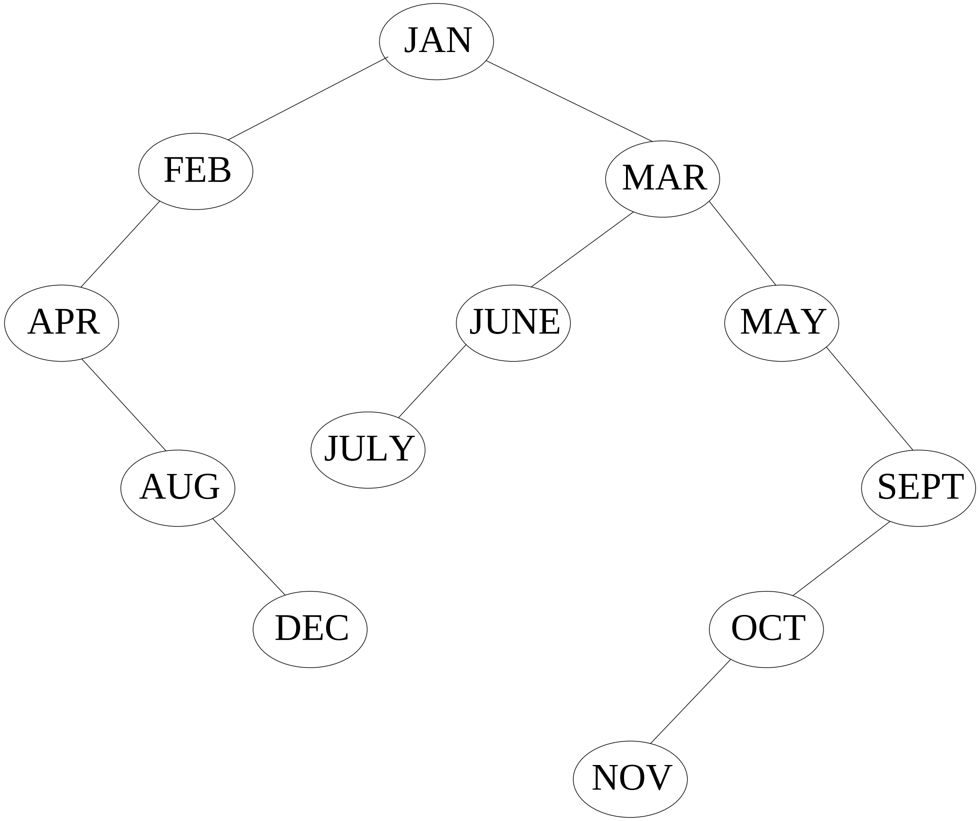
- Optimal Binary Search Trees
- AVL Trees
- 2-3 Trees

Optimal Binary Search Trees



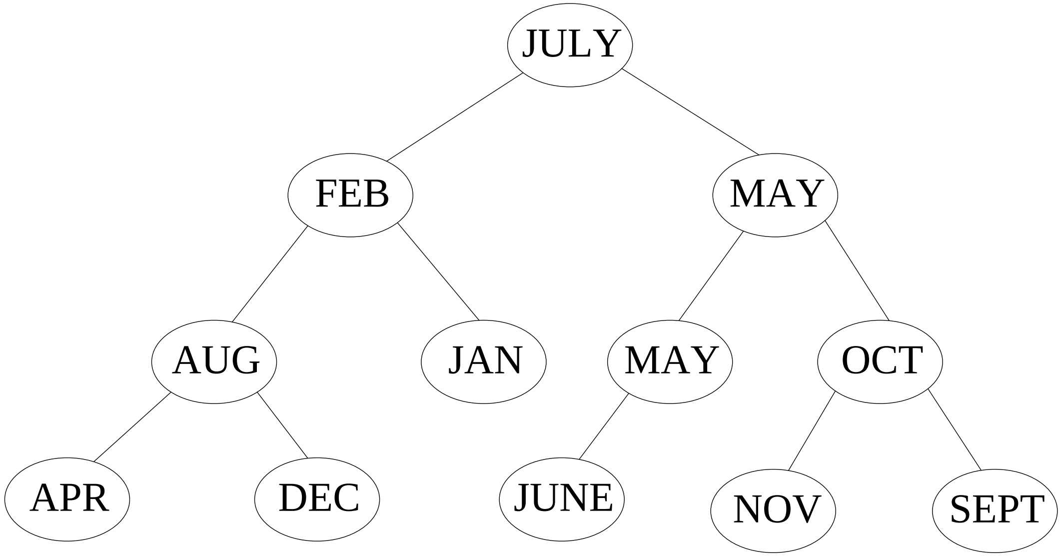
(a) (b) (c) (d) (e)

Adelson-Velskii & Landis Tree (AVL Tree)



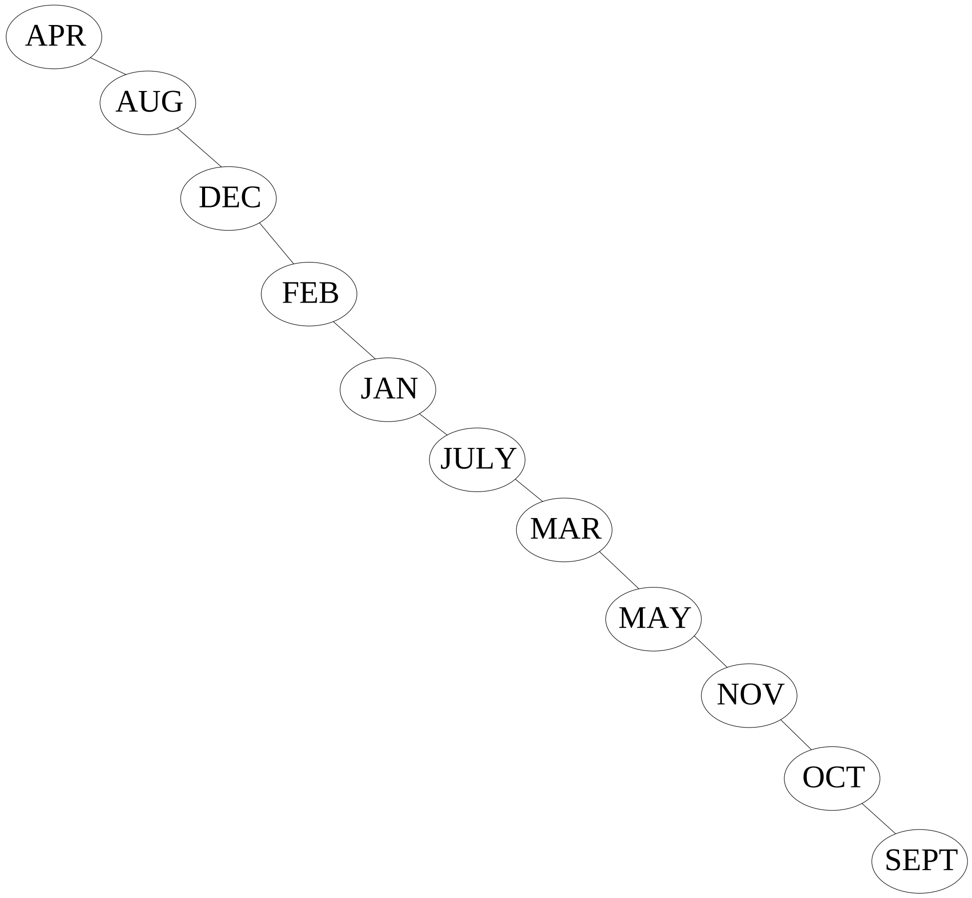
1. Apr
2. Aug
3. Dec
4. Feb
5. Jan
6. July
7. June
8. Mar
9. May
10. Nov
11. Oct
12. Sept

Adelson-Velskii & Landis Tree (AVL Tree)



1. Apr
2. Aug
3. Dec
4. Feb
5. Jan
6. July
7. June
8. Mar
9. May
10. Nov
11. Oct
12. Sept

Adelson-Velskii & Landis Tree (AVL Tree)



1. Apr
2. Aug
3. Dec
4. Feb
5. Jan
6. July
7. June
8. Mar
9. May
10. Nov
11. Oct
12. Sept

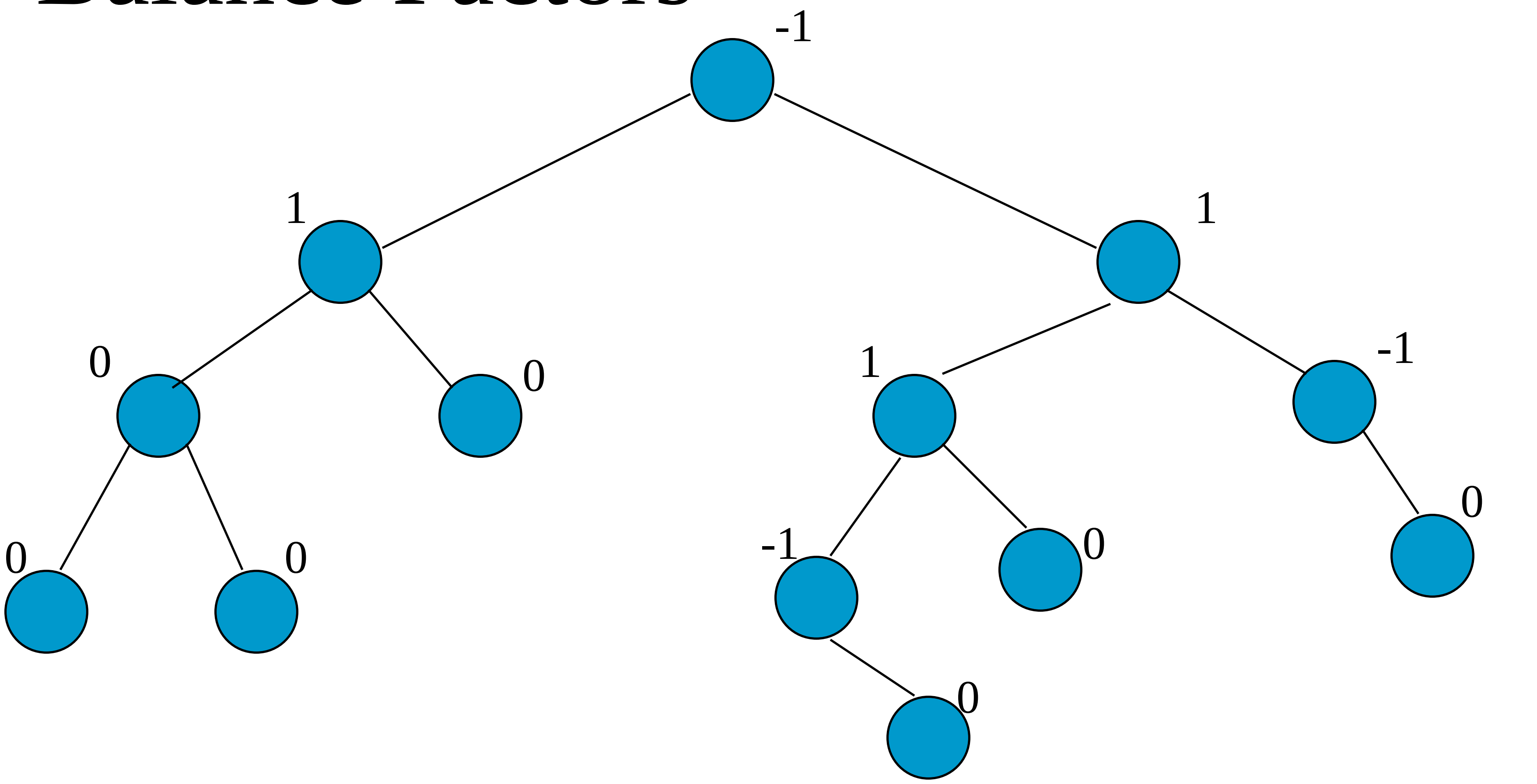
AVL Tree

- binary tree
- for every node x, define its balance factor

  balance factor of x = height of left subtree of x

  – height of right subtree of x
- balance factor of every node x is – 1, 0, or 1

Balance Factors

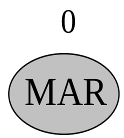


This is an AVL tree.

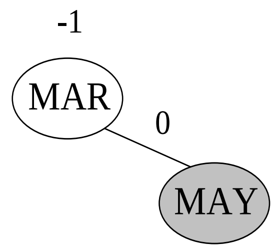
Height Of An AVL Tree

- The height of an AVL tree that has n nodes is at most $1.44 \log_2 (n+2)$.
- The height of every n node binary tree is at least $\log_2 (n+1)$.
- $\log_2 (n+1) <= height <= 1.44 \log_2 (n+2)$

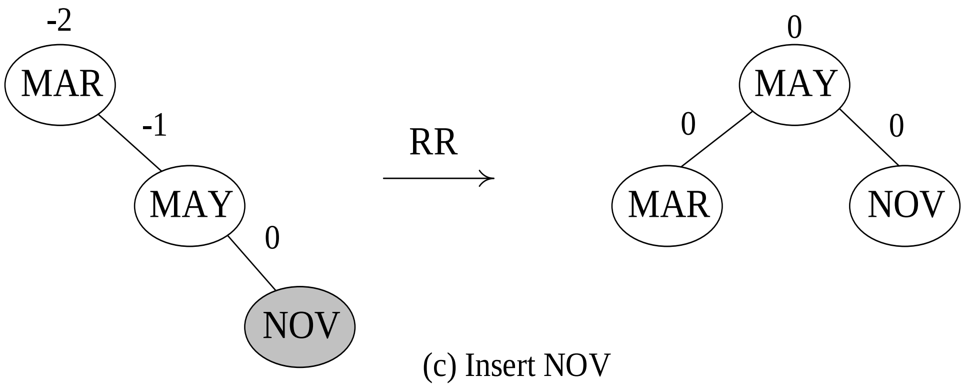
AVL Trees (1/9)



(a) Insert MARCH



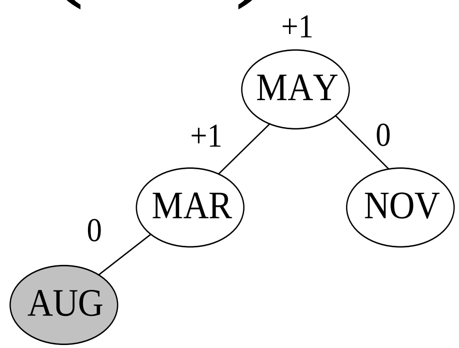
(b) Insert MAY



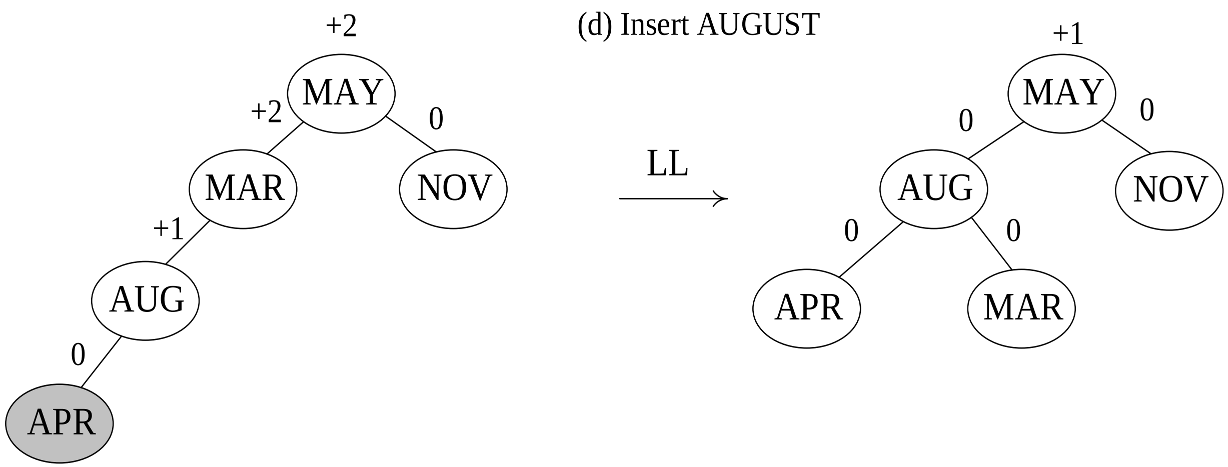
(c) Insert NOV

1. Apr
2. Aug
3. Dec
4. Feb
5. Jan
6. July
7. June
8. Mar
9. May
10. Nov
11. Oct
12. Sept

AVL Trees (2/9)



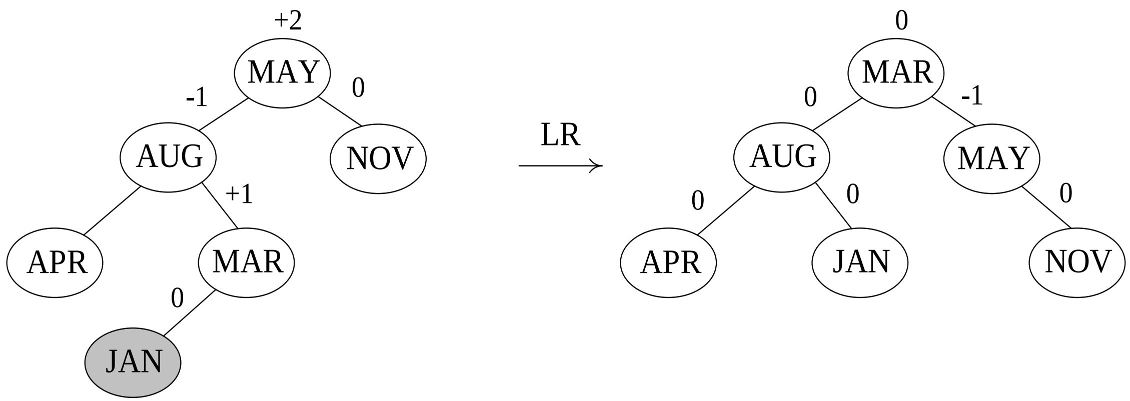
(d) Insert AUGUST



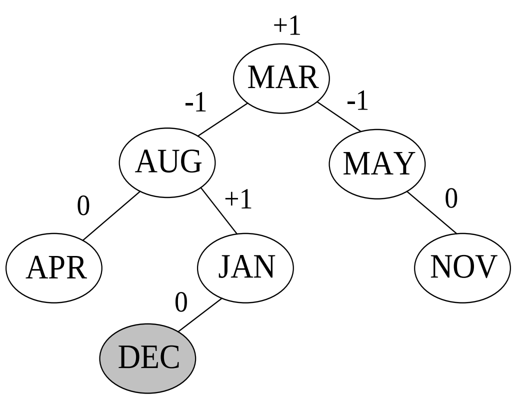
(e) Insert APRIL

1. Apr
2. Aug
3. Dec
4. Feb
5. Jan
6. July
7. June
8. Mar
9. May
10. Nov
11. Oct
12. Sept

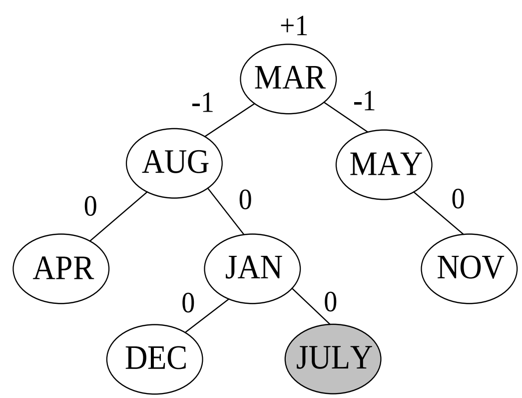
AVL Trees (3/9)



(f) Insert JANUARY



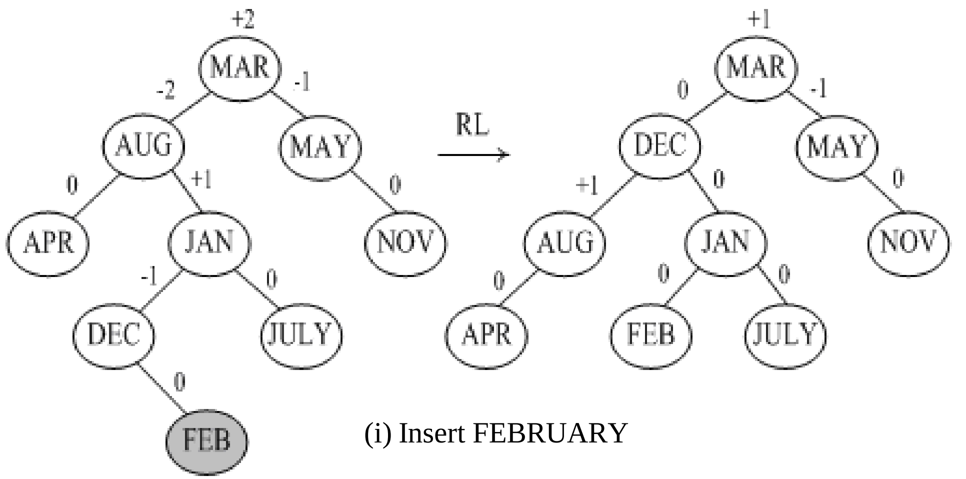
(g) Insert DECMBER



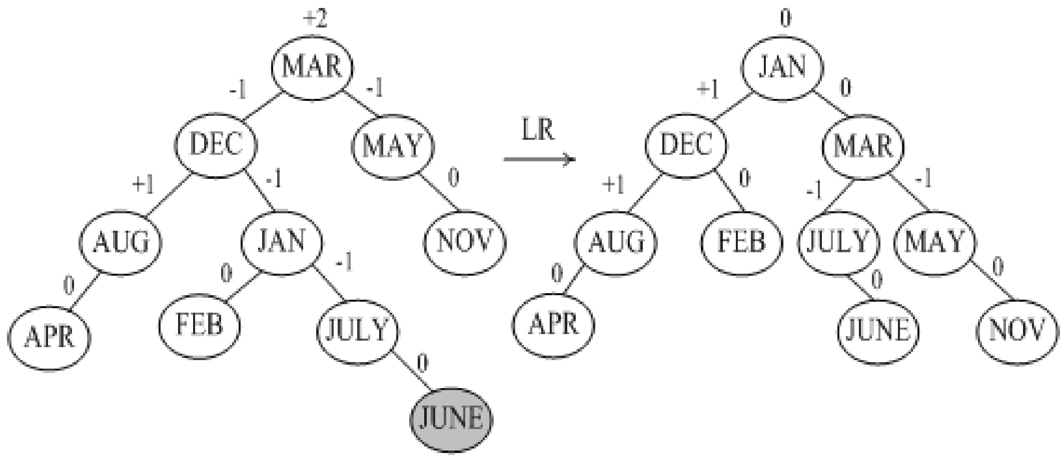
(h) Insert JULY

1.Apr
2.Aug
3.Dec
4.Feb
5.Jan
6.July
7.June
8.Mar
9.May
10.Nov
11.Oct
12.Sept

AVL Trees (4/9)



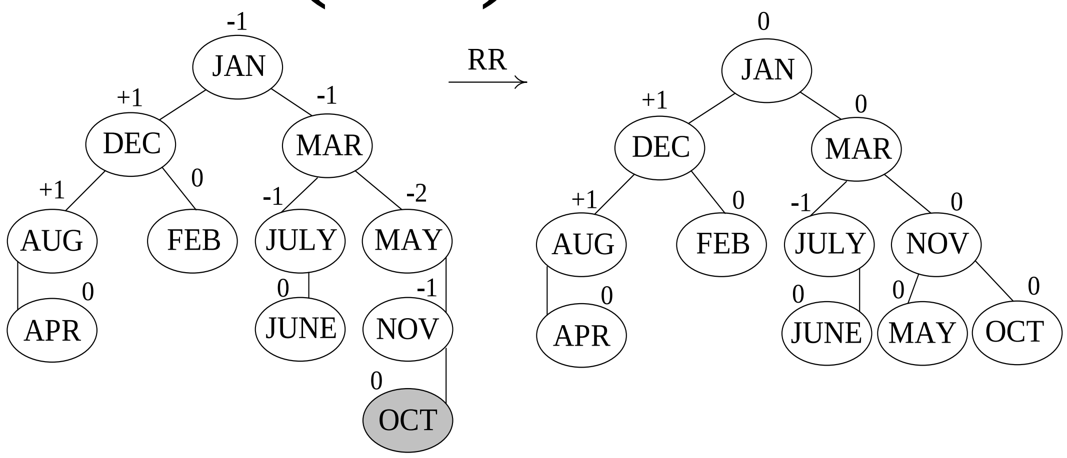
(i) Insert FEBRUARY



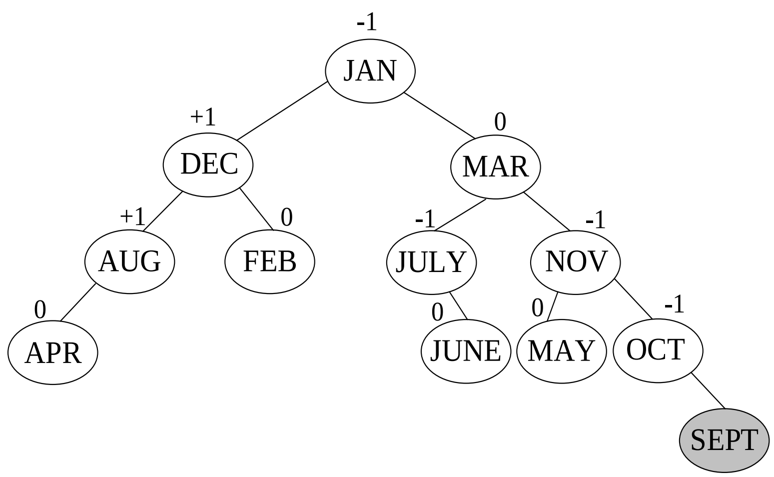
(j) Insert JUNE

1.Apr
2.Aug
3.Dec
4.Feb
5.Jan
6.July
7.June
8.Mar
9.May
10.Nov
11.Oct
12.Sept

AVL Trees (5/9)



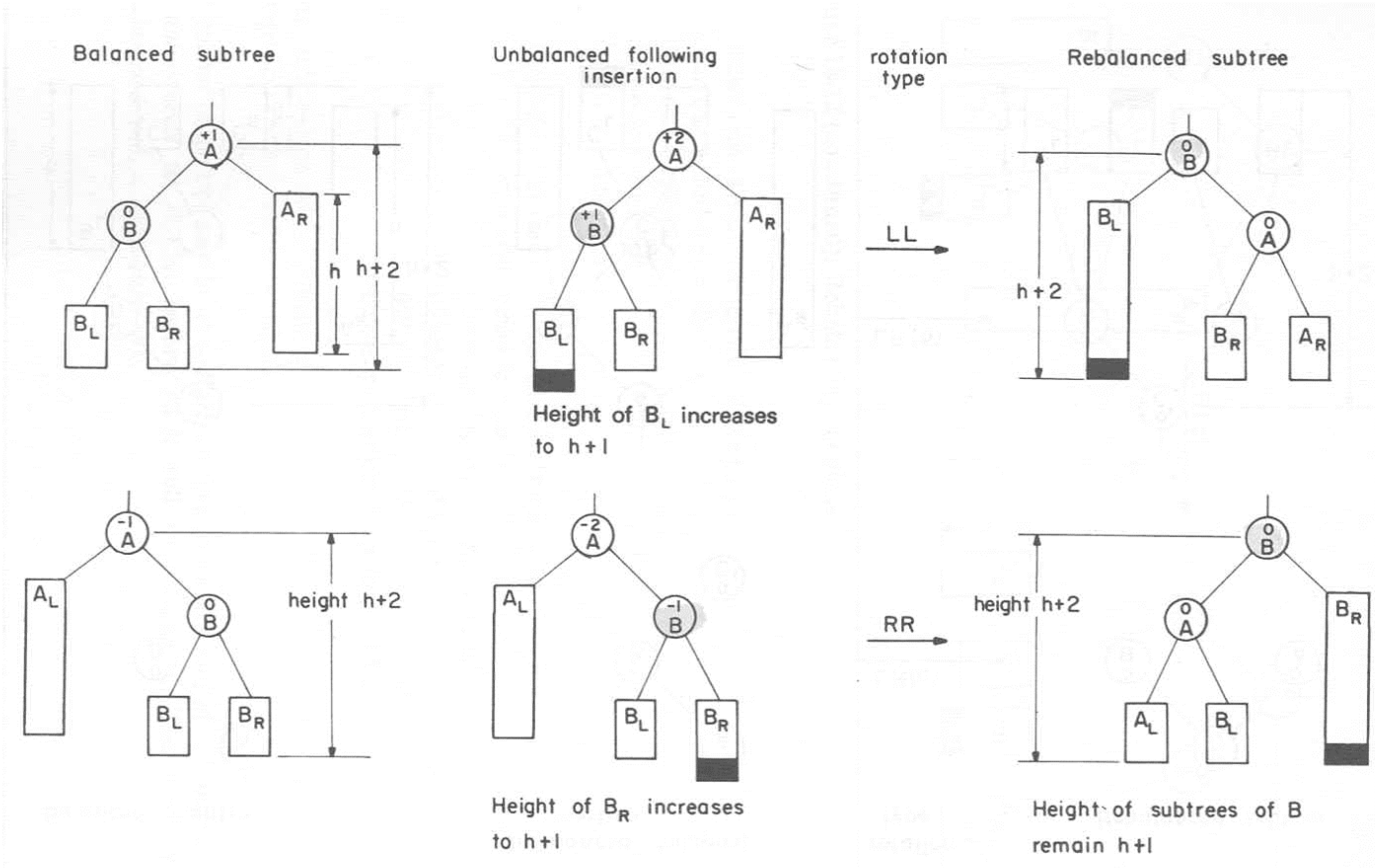
(k) Insert OCTOBER



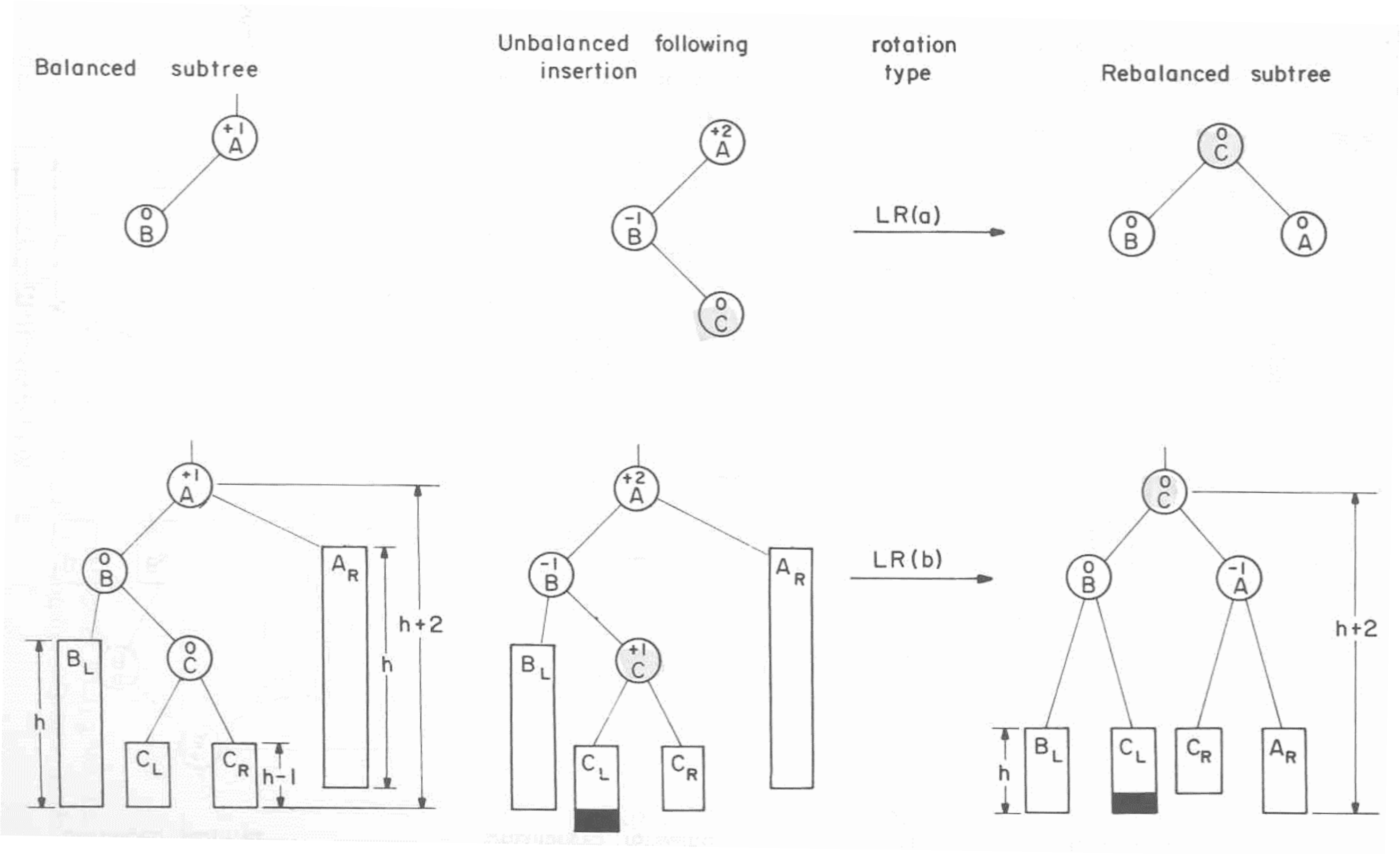
(l) Insert SEPTEMBER

1. Apr
2. Aug
3. Dec
4. Feb
5. Jan
6. July
7. June
8. Mar
9. May
10. Nov
11. Oct
12. Sept

AVL Trees (6/9)

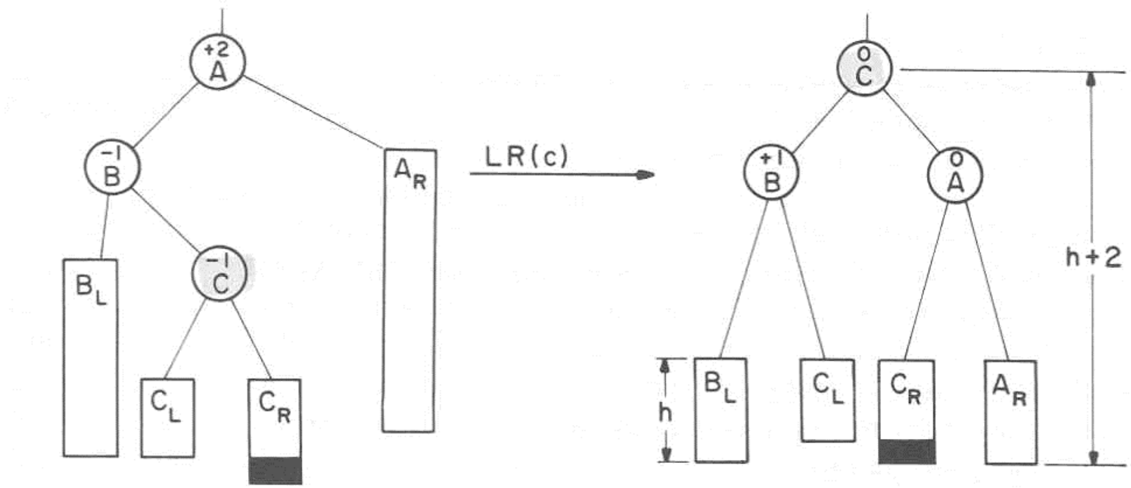

Balanced subtree
Unbalanced following insertion
rotation type
Rebalanced subtree
+1 A
0 B
B_L
B_R
A_R
h h+2
+2 A
+1 B
Height of B_L increases to h+1
LL
0 B
0 A
h+2
-1 A
A_L
height h+2
-2 A
-1 B
Height of B_R increases to h+1
RR
height h+2
Height of subtrees of B remain h+1

AVL Trees (7/9)



Balanced subtree
Unbalanced following insertion
rotation type
Rebalanced subtree
LR(a)
LR(b)
h+2
h
h-1

AVL Trees (8/9)



AVL Trees (9/9)

Operation	Sequential list	Linked list	AVL tree
Search for x	$O(\log n)$	$O(n)$	$O(\log n)$
Search for kth item	$O(1)$	$O(k)$	$O(\log n)$
Delete x	$O(n)$	$O(1)$[1]	$O(\log n)$
Delete kth item	$O(n - k)$	$O(k)$	$O(\log n)$
Insert x	$O(n)$	$O(1)$[2]	$O(\log n)$
Output in order	$O(n)$	$O(n)$	$O(n)$

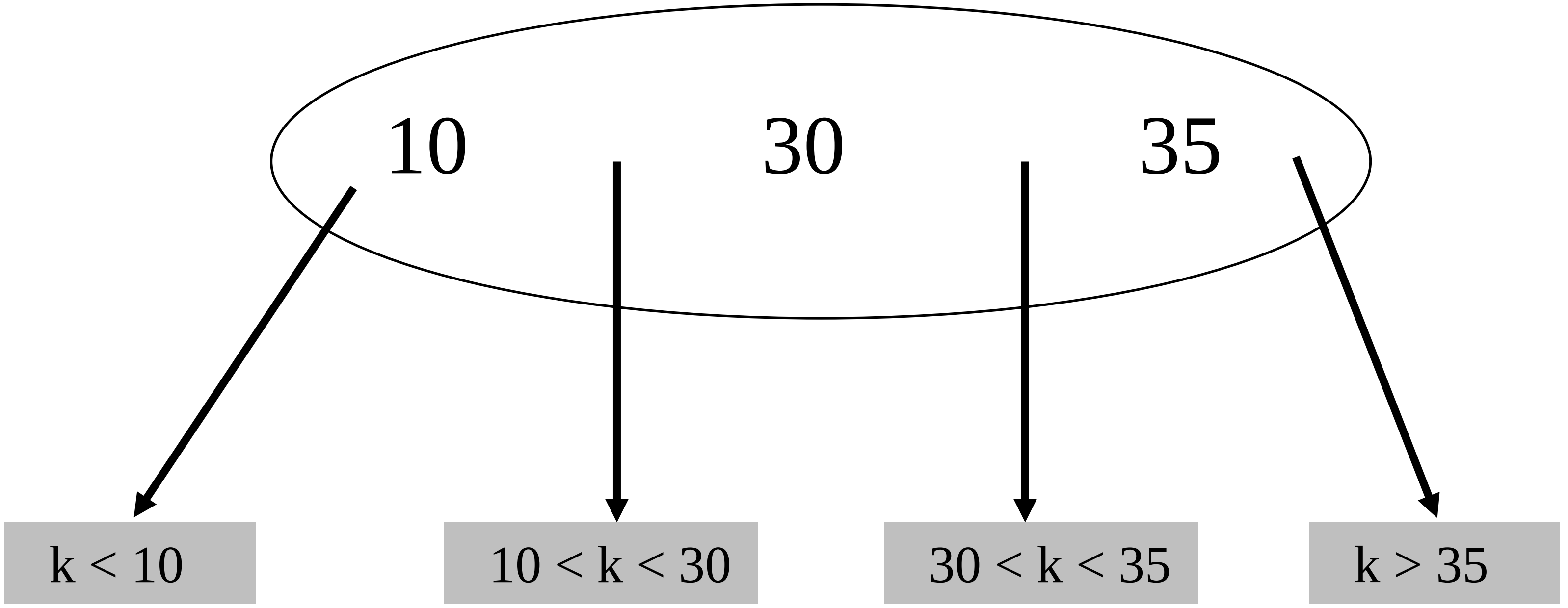
1. Doubly linked list and position of x known.
2. If position for insertion is known.

Figure 10.14: Comparison of various structures

m-way Search Trees

- Each node has up to m – 1 pairs and m children.
- m = 2 => binary search tree.

4-Way Search Tree



Definition Of B-Tree

- Definition assumes external nodes (extended m-way search tree).
- B-tree of order m.
  - m-way search tree.
  - Not empty => root has at least 2 children.
  - Remaining internal nodes (if any) have at least ceil(m/2) children.
  - External (or failure) nodes on same level.

2-3 Tree and 2-3-4 Tree

- 2-3 tree is B-tree of order 3.
- 2-3-4 tree is B-tree of order 4.

2-3 Trees(1/7)

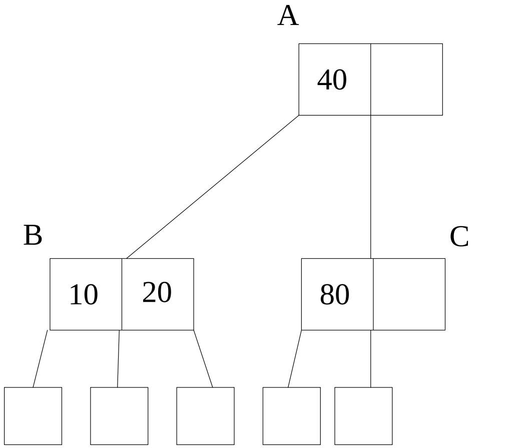


Figure 11.2 Example of a 2-3 tree

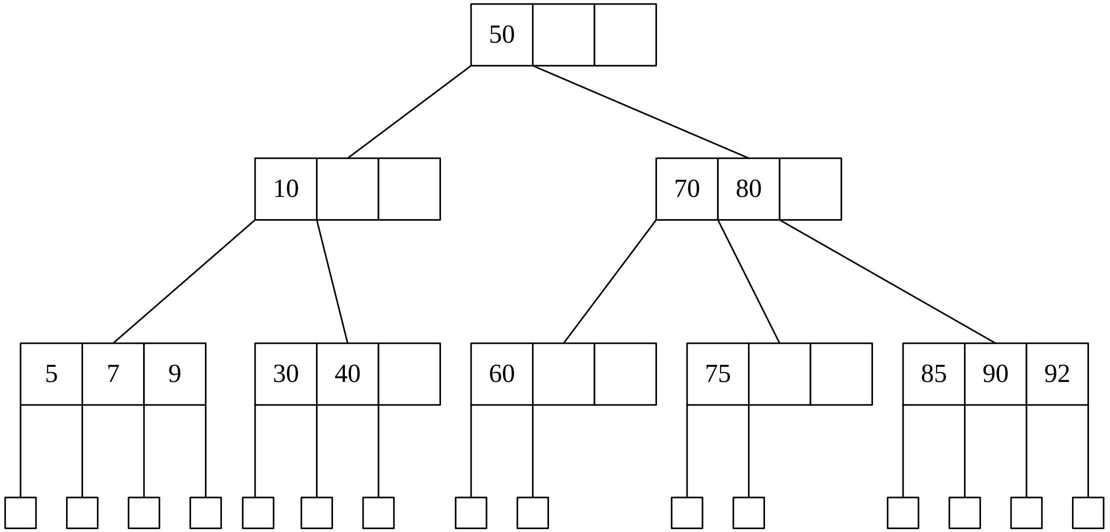
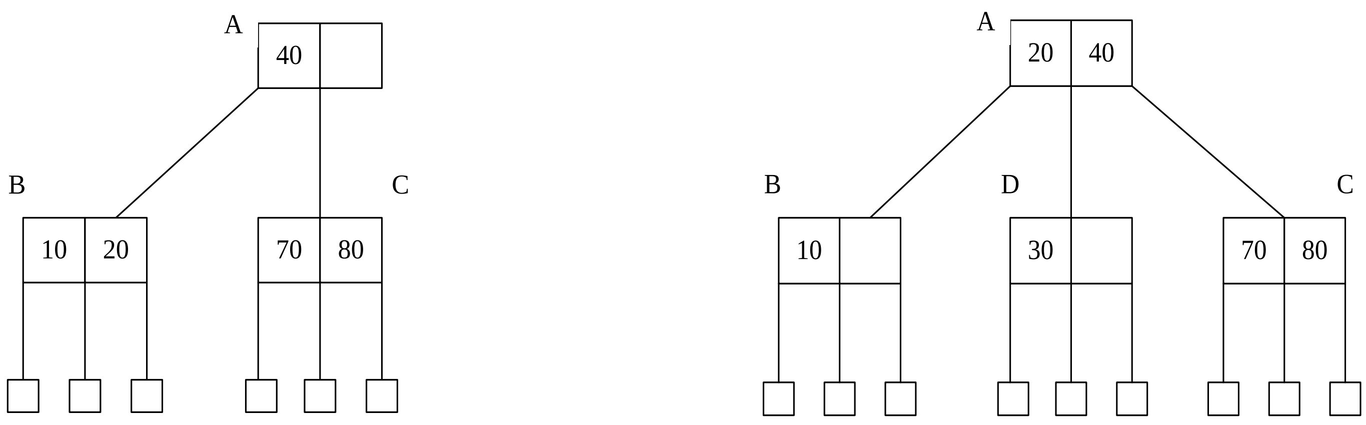


Figure 11.3 Example of a 2-3-4 tree

2-3 Trees(2/7)-Split



(a) 70 inserted (b) 30 inserted

Figure 11.4 Insertion into the 2-3 tree of Figure 11.2

2-3 Trees(3/7)-Split

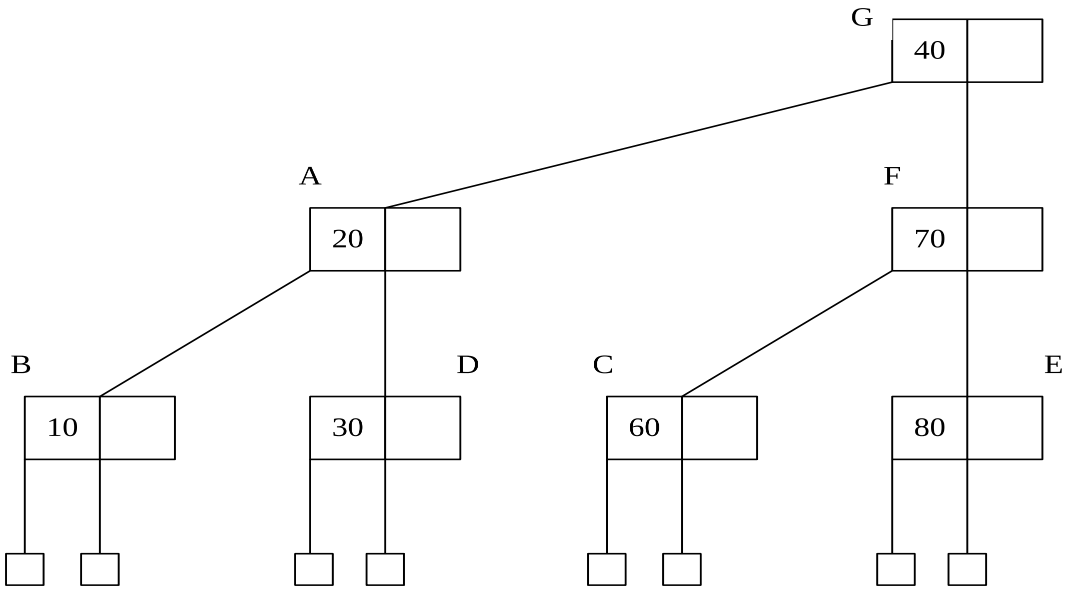
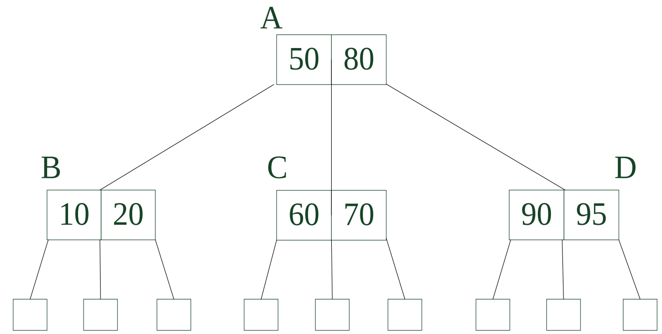
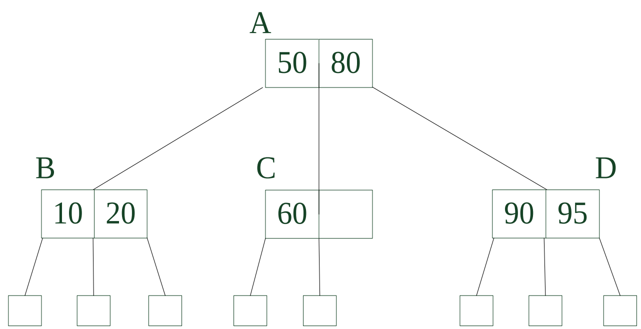


Figure 11.5 Insertion of 60 into the 2-3 tree of Figure 11.4(b)

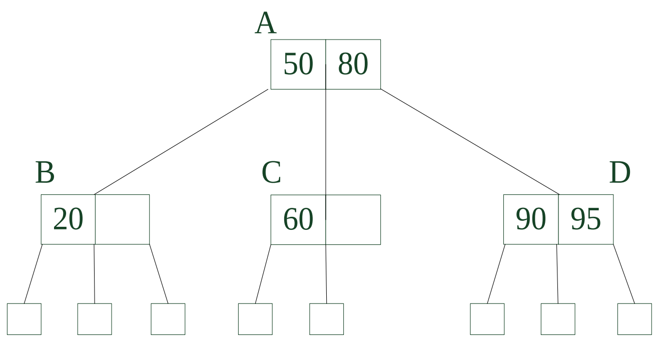
2-3 Trees (4/7)



(a) Initial 2-3 tree



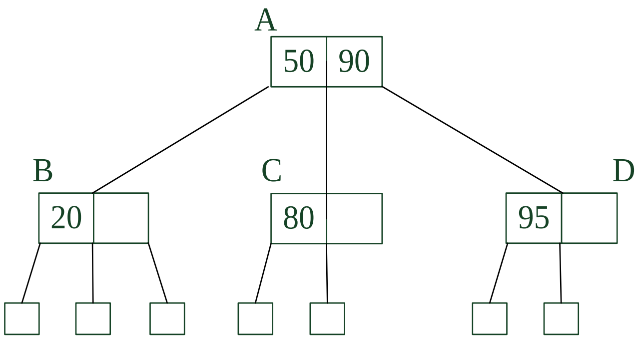
(b) 70 deleted



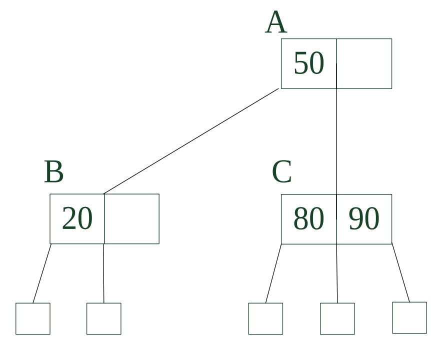
(c) 10 deleted

Figure 11.9 Deletion from a 2-3 tree

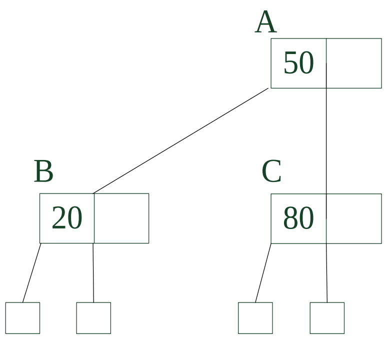
2-3 Trees (5/7)-Rotation&Combine



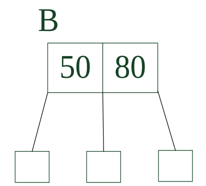
(a) 60 deleted



(b) 95 deleted



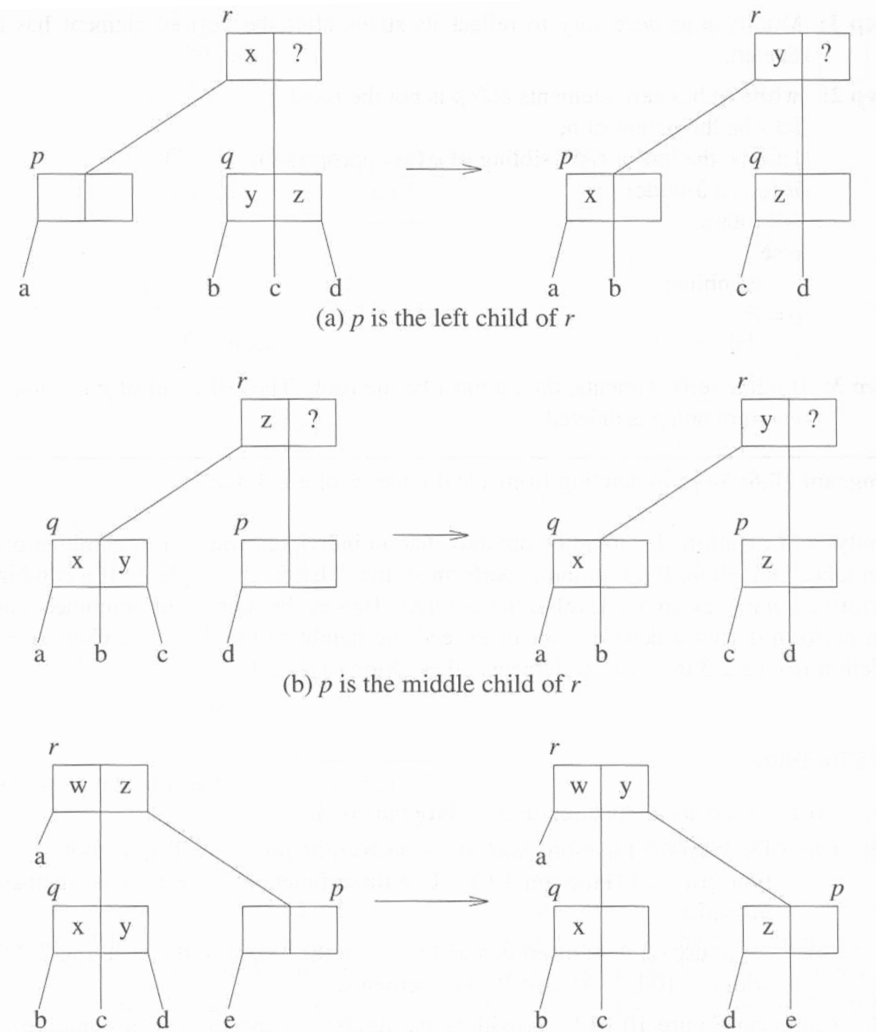
(c) 90 deleted



(d) 20 deleted

Figure 11.9 Deletion from a 2-3 tree

2-3 Trees (6/7) -Rotation



(a) *p* is the left child of *r*

(b) *p* is the middle child of *r*

(c) *p* is the right child of *r*

Figure 11.7 The three cases for rotation in a 2-3 tree

2-3 Trees (7/7)-Combine

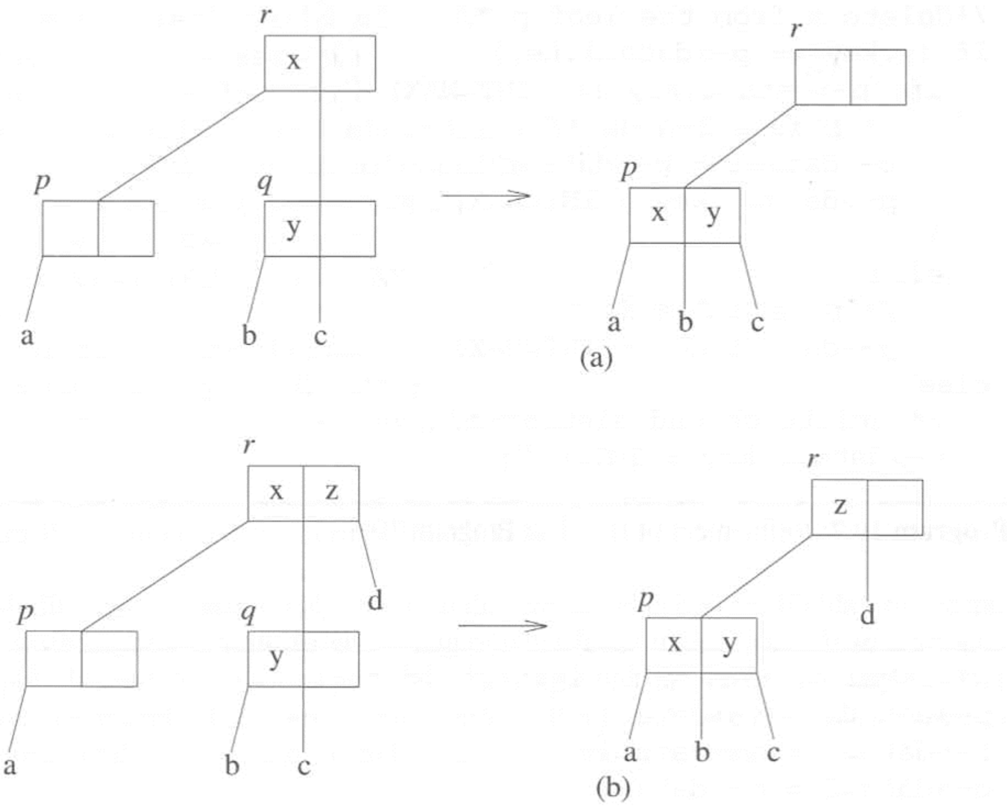


Figure 11.8 Combining in a 2-3 tree when *p* is the left child of *r*