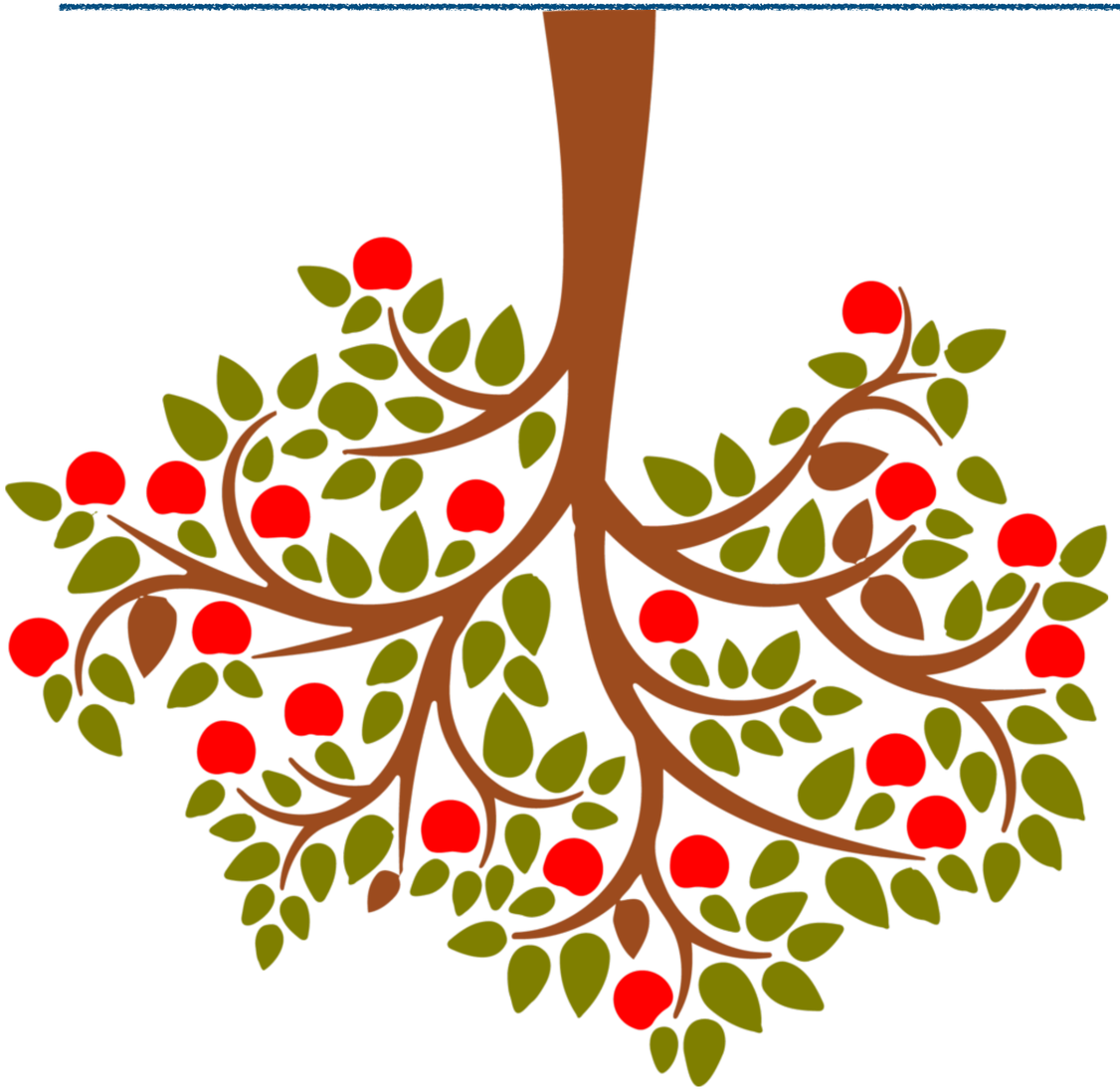
CPE112
Programming with Data Structures

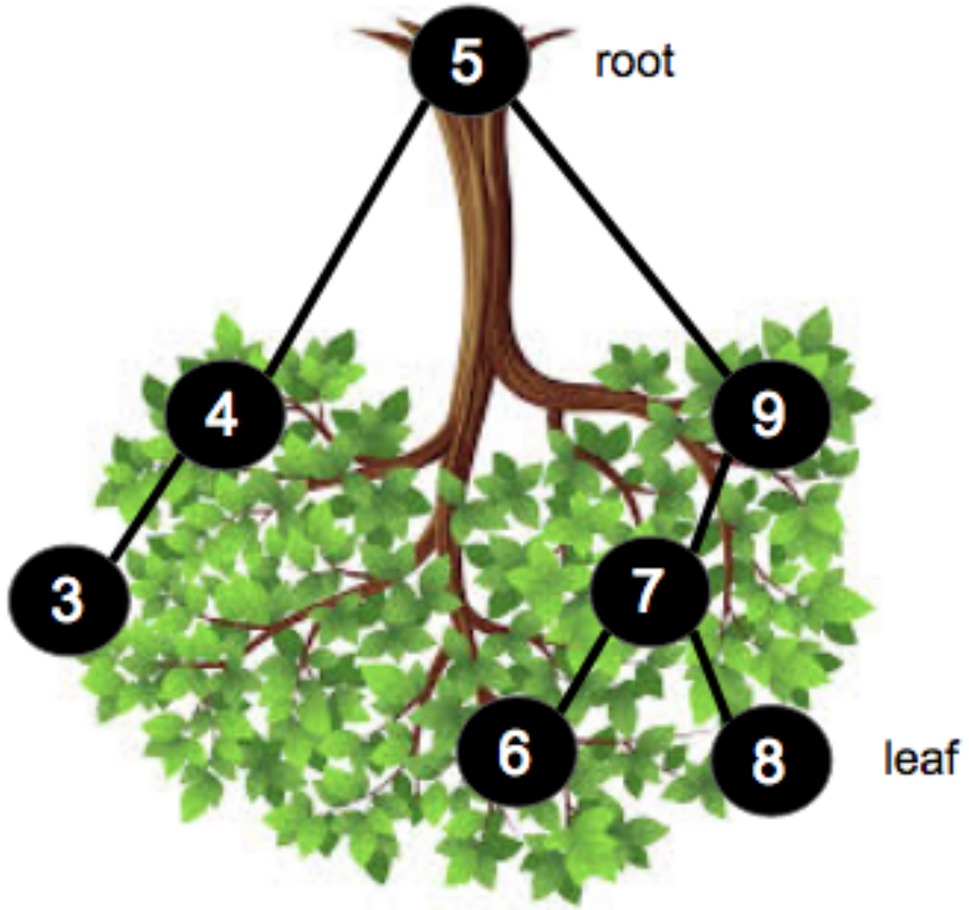
MODULE 2-1
Tree

Assc. Prof. Dr. Natasha Dejdumrong

KING MONGKUT'S UNIVERSITY
OF TECHNOLOGY THONBURI

Tree





Outlines

- Introduction to Tree
- Types of Trees
- General Trees
- Binary Trees
- Representation of Binary Trees
- Binary Search Trees
- Expression Trees
- Tree Traversal
- Huffman's Tree
- Applications of Trees

Basic Terminology

Root node The root node R is the topmost node in the tree. If R = NULL, then it means the tree is empty.

Sub-trees If the root node R is not NULL, then the trees T_1, T_2, and T_3 are called the sub-trees of R.

Leaf node A node that has no children is called the leaf node or the terminal node.

Path A sequence of consecutive edges is called a *path*. For example, in Fig. 9.1, the path from the root node A to node I is given as: A, D, and I.

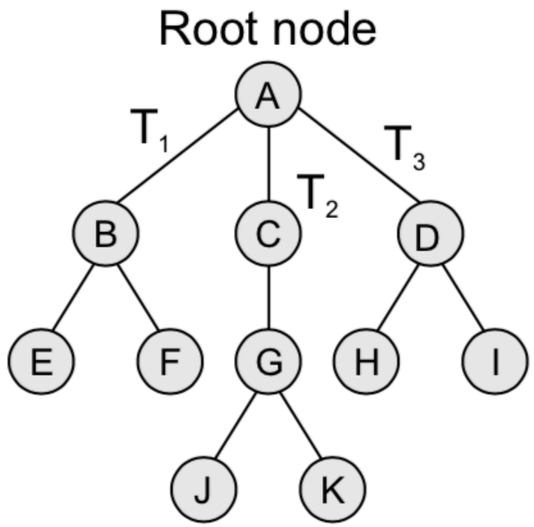


Figure 9.1 Tree

Basic Terminology

Ancestor node An ancestor of a node is any predecessor node on the path from root to that node. The root node does not have any ancestors. In the tree given in Fig. 9.1, nodes A, C, and G are the ancestors of node K.

Descendant node A descendant node is any successor node on any path from the node to a leaf node. Leaf nodes do not have any descendants. In the tree given in Fig. 9.1, nodes C, G, J, and K are the descendants of node A.

Level number Every node in the tree is assigned a *level number* in such a way that the root node is at level 0, children of the root node are at level number 1. Thus, every node is at one level higher than its parent. So, all child nodes have a level number given by parent's `level number + 1`.

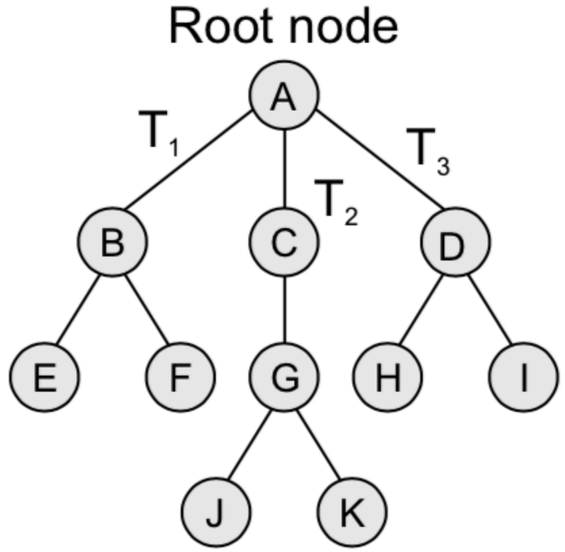


Figure 9.1 Tree

Basic Terminology

Degree Degree of a node is equal to the number of children that a node has. The degree of a leaf node is zero.

In-degree In-degree of a node is the number of edges arriving at that node.

Out-degree Out-degree of a node is the number of edges leaving that node.

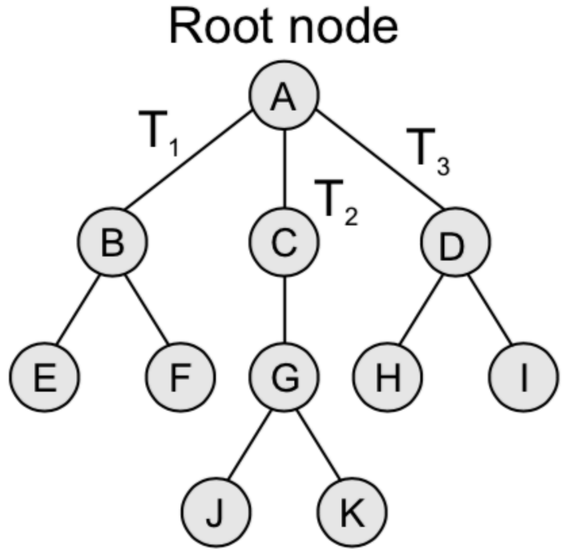


Figure 9.1 Tree

Types of Trees

- Trees are of following 6 types:
  1. General trees
  2. Forests
  3. Binary trees
  4. Binary search trees
  5. Expression trees
  6. Tournament trees.

General Trees

- General trees are data structures that store elements hierarchically.
- The top node of a tree is the root node and each node, except the root, has a parent.
- A node in a general tree (except the leaf nodes) may have zero or more sub-trees.
- General trees which have 3 sub-trees per node are called ternary trees.
- However, the number of sub-trees for any node may be variable.
- For example, a node can have 1 sub-tree, whereas some other node can have 3 sub-trees.

Forests

★ A forest is a disjoint union of trees.

★ A set of disjoint trees (or forests) is obtained by deleting the root and the edges connecting the root node to nodes at level 1.

★ We have already seen that every node of a tree is the root of some sub-tree. Therefore, all the sub-trees immediately below a node form a forest.

★ A forest can also be defined as an ordered set of zero or more general trees. While a general tree must have a root, a forest on the other hand may be empty because by definition it is a set, and sets can be empty.

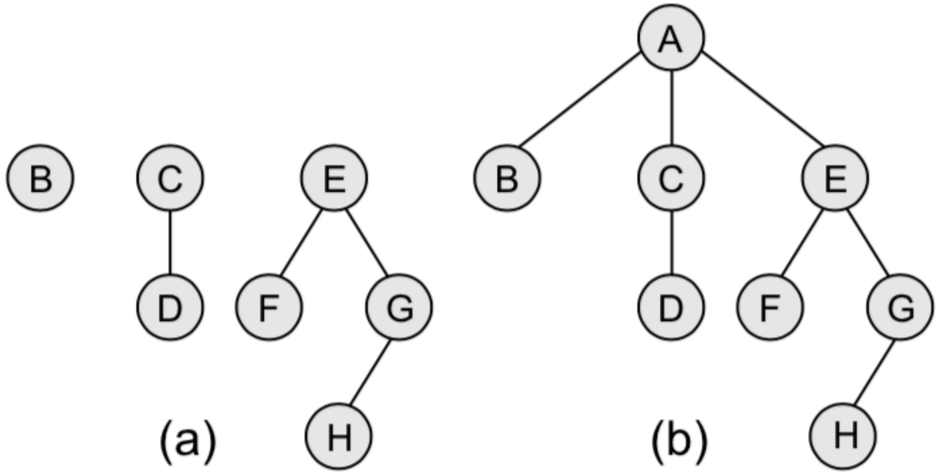
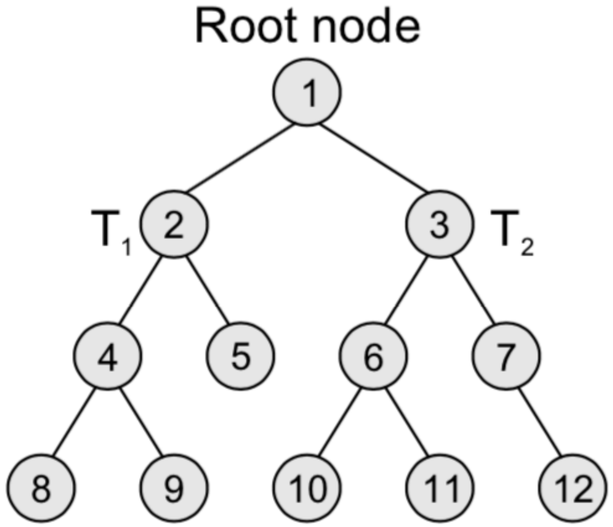


Figure 9.2 Forest and its corresponding tree

Binary Trees

- A binary tree is a data structure that is defined as a collection of elements called nodes.
- In a binary tree, the topmost element is called the root node, and each node has 0, 1, or at the most 2 children.
- A node that has zero children is called a leaf node or a terminal node.
- Every node contains a data element, a left pointer which points to the left child, and a right pointer which points to the right child.

Binary Trees



The root element is pointed by a `'root'` pointer. If `root = NULL`, then it means the tree is empty.

Figure 9.3 shows a binary tree. In the figure, `R` is the root node and the two trees T_1 and T_2 are called the left and right sub-trees of `R`. T_1 is said to be the left successor of `R`. Likewise, T_2 is called the right successor of `R`.

Note that the left sub-tree of the root node consists of the nodes: 2, 4, 5, 8, and 9. Similarly, the right sub-tree of the root node consists of nodes: 3, 6, 7, 10, 11, and 12.

Terminology for Binary Trees

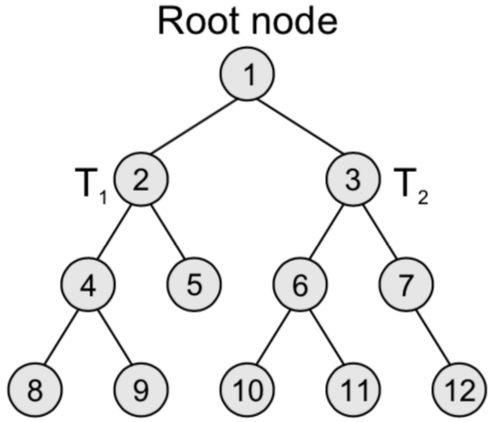


Figure 9.3 Binary tree

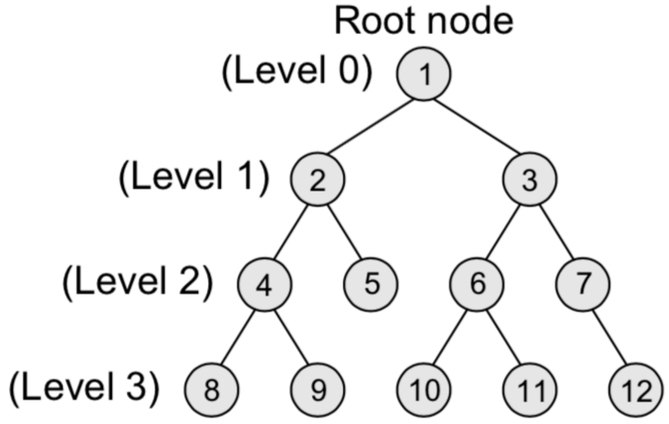


Figure 9.4 Levels in binary tree

Parent If N is any node in T that has *left successor* S_1 and *right successor* S_2, then N is called the *parent* of S_1 and S_2. Correspondingly, S_1 and S_2 are called the left child and the right child of N. Every node other than the root node has a parent.

Level number Every node in the binary tree is assigned a *level number* (refer Fig. 9.4). The root node is defined to be at level 0. The left and the right child of the root node have a level number 1. Similarly, every node is at one level higher than its parents. So all child nodes are defined to have level number as `parent's level number + 1`.

Degree of a node It is equal to the number of children that a node has. The degree of a leaf node is zero. For example, in the tree, degree of node 4 is 2, degree of node 5 is zero and degree of node 7 is 1.

Sibling All nodes that are at the same level and share the same parent are called *siblings* (brothers). For example, nodes 2 and 3; nodes 4 and 5; nodes 6 and 7; nodes 8 and 9; and nodes 10 and 11 are siblings.

Terminology for Binary Trees

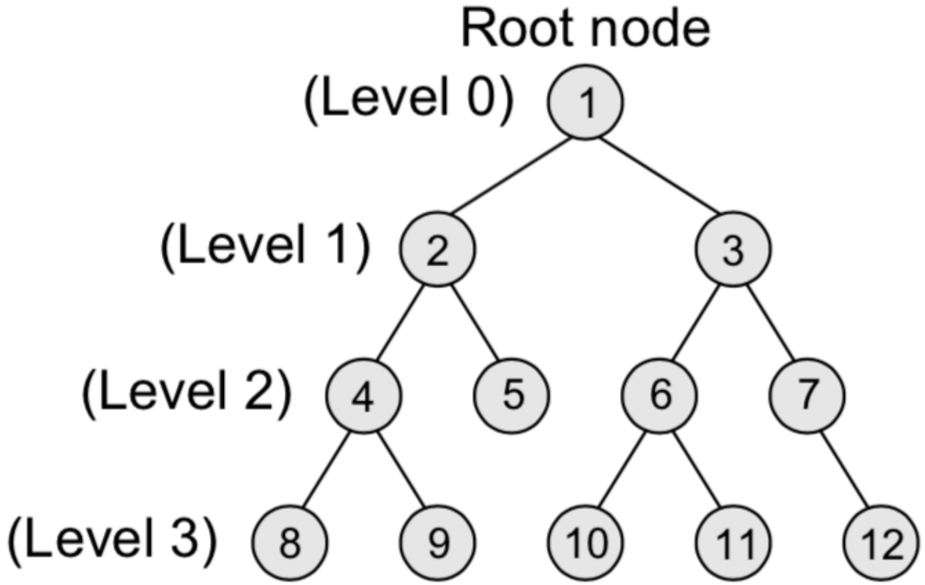


Figure 9.4 Levels in binary tree

Leaf node A node that has no children is called a leaf node or a terminal node. The leaf nodes in the tree are: 8, 9, 5, 10, 11, and 12.

Terminology for Binary Trees

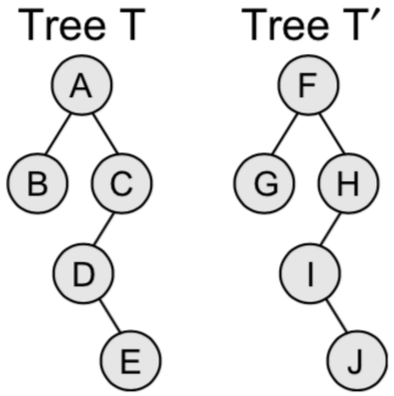


Figure 9.5 Similar binary trees

Similar binary trees Two binary trees T and T′ are said to be similar if both these trees have the same structure. Figure 9.5 shows two *similar binary trees*.

Copies Two binary trees T and T′ are said to be *copies* if they have similar structure and if they have same content at the corresponding nodes. Figure 9.6 shows that T′ is a copy of T.

Edge It is the line connecting a node N to any of its successors. A binary tree of n nodes has exactly n - 1 edges because every node except the root node is connected to its parent via an edge.

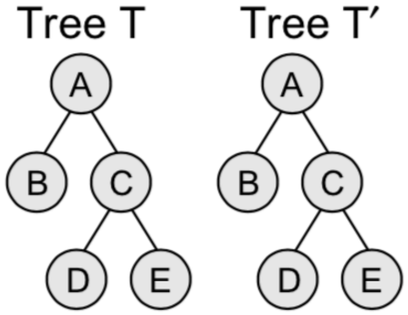


Figure 9.6 T′ is a copy of T

Path A sequence of consecutive edges. For example, in Fig. 9.4, the path from the root node to the node 8 is given as: 1, 2, 4, and 8.

Depth The *depth* of a node N is given as the length of the path from the root R to the node N. The depth of the root node is zero.

Height of a tree It is the total number of nodes on the path from the root node to the deepest node in the tree. A tree with only a root node has a height of 1.

Complete Binary Trees

- A complete binary tree is a binary tree that satisfies two properties.
  - First, in a complete binary tree, every level, except possibly the last, is completely filled.
  - Second, all nodes appear as far left as possible.

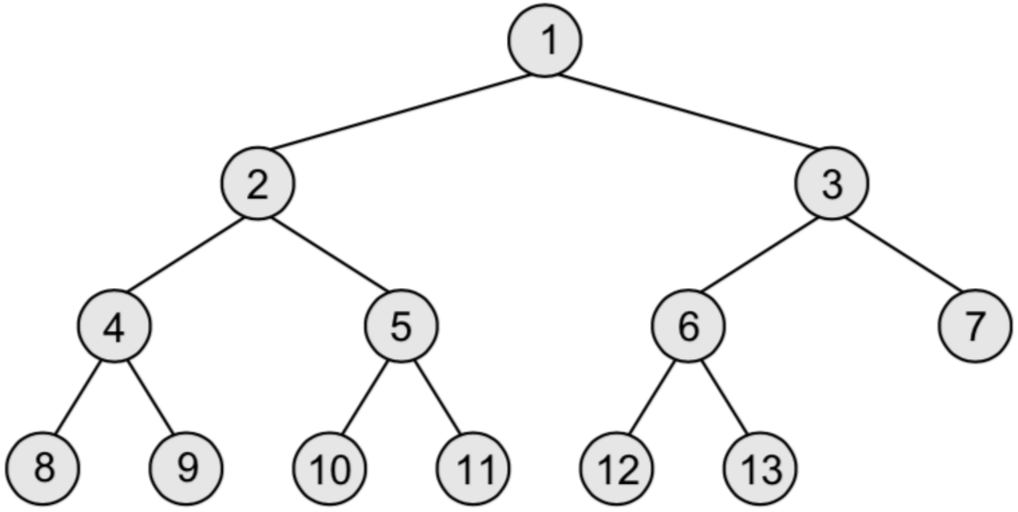


Figure 9.7 Complete binary tree

Complete Binary Trees

In a complete binary tree T_n, there are exactly n nodes and level r of T can have at most 2^r nodes. Figure 9.7 shows a complete binary tree.

Note that in Fig. 9.7, level 0 has $2^0 = 1$ node, level 1 has $2^1 = 2$ nodes, level 2 has $2^2 = 4$ nodes, level 3 has 6 nodes which is less than the maximum of $2^3 = 8$ nodes.

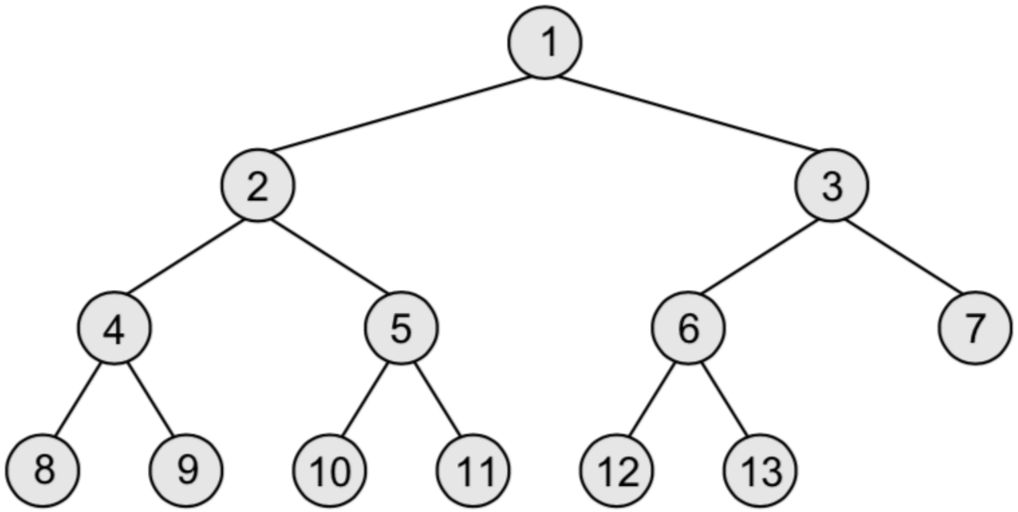


Figure 9.7 Complete binary tree

Complete Binary Trees

In Fig. 9.7, tree T_{13} has exactly 13 nodes. They have been purposely labelled from 1 to 13, so that it is easy for the reader to find the parent node, the right child node, and the left child node of the given node. The formula can be given as—if K is a parent node, then its left child can be calculated as 2 × K and its right child can be calculated as 2 × K + 1. For example, the children of the node 4 are 8 (2 × 4) and 9 (2 × 4 + 1). Similarly, the parent of the node K can be calculated as | K/2 |. Given the node 4, its parent can be calculated as | 4/2 | = 2. The height of a tree T_n having exactly n nodes is given as:

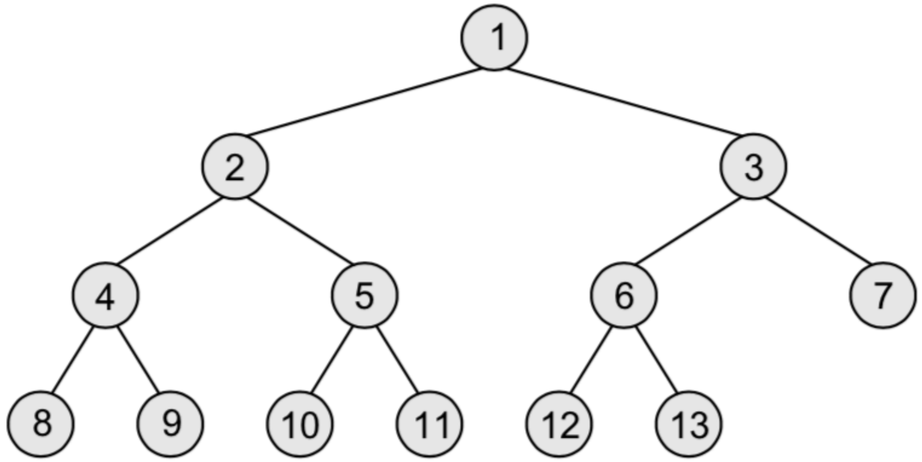


Figure 9.7 Complete binary tree

$$H_n = \lfloor \log_2 (n + 1) \rfloor$$

This means, if a tree T has 10,00,000 nodes, then its height is 21.

Representation of Binary Trees

In the computer's memory, a binary tree can be maintained either by using a linked representation or by using a sequential representation.

Linked representation of binary trees In the linked representation of a binary tree, every node will have three parts: the data element, a pointer to the left node, and a pointer to the right node. So in C, the binary tree is built with a node type given below.

```
struct node {
        struct node *left;
        int data;
        struct node *right;
};
```

Representation of Binary Trees

```
struct node {
        struct node *left;
        int data;
        struct node *right;
};
```

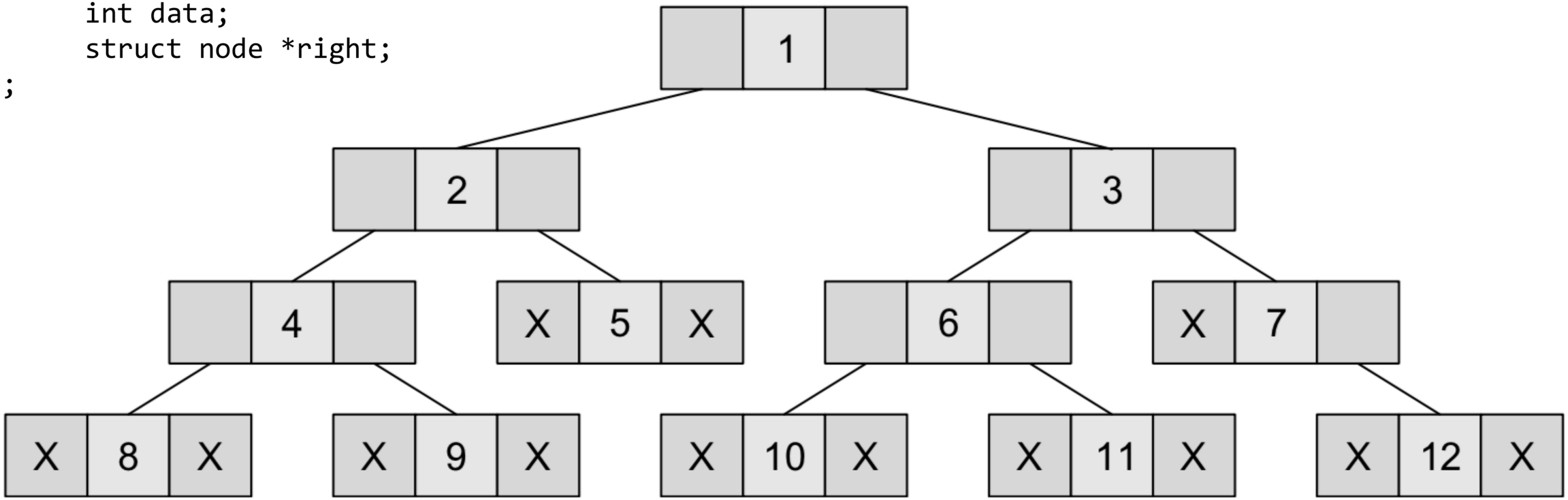


Figure 9.9 Linked representation of a binary tree

Representation of Binary Trees

```
struct node {
        struct node *left;
        int data;
        struct node *right;
};
```

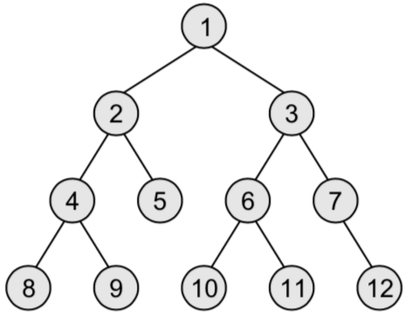


Figure 9.10 Binary tree T

ROOT: 3

AVAIL: 15

	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

Figure 9.11 Linked representation of binary tree T

Example

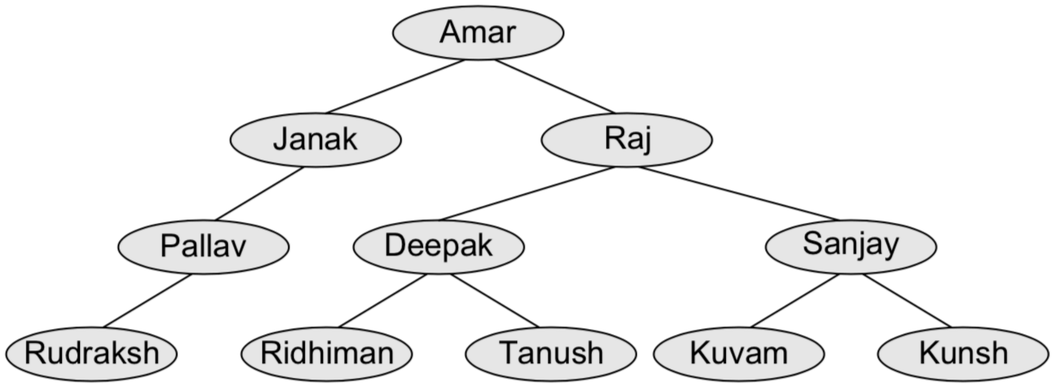
Example 9.1 Given the memory representation of a tree that stores the names of family members, construct the corresponding tree from the given data.

Solution

ROOT 3

7 AVAIL

	LEFT	NAMES	RIGHT
1	12	Pallav	–1
2			
3	9	Amar	13
4			
5			
6	19	Deepak	17
7			
8			
9	1	Janak	–1
10			
11	–1	Kuvam	–1
12	–1	Rudraksh	–1
13	6	Raj	20
14			
15	–1	Kunsh	–1
16			
17	–1	Tanush	–1
18			
19	–1	Ridhiman	–1
20	11	Sanjay	15



Sequential Representation of Binary Trees

Sequential representation of binary trees Sequential representation of trees is done using single or one-dimensional arrays. Though it is the simplest technique for memory representation, it is inefficient as it requires a lot of memory space. A sequential binary tree follows the following rules:

- A one-dimensional array, called `TREE`, is used to store the elements of tree.
- The root of the tree will be stored in the first location. That is, `TREE[1]` will store the data of the root element.
- The children of a node stored in location `K` will be stored in locations `(2 × K)` and `(2 × K+1)`.
- The maximum size of the array `TREE` is given as (2^h-1), where `h` is the `height` of the tree.
- An empty tree or sub-tree is specified using `NULL`. If `TREE[1] = NULL`, then the tree is empty.

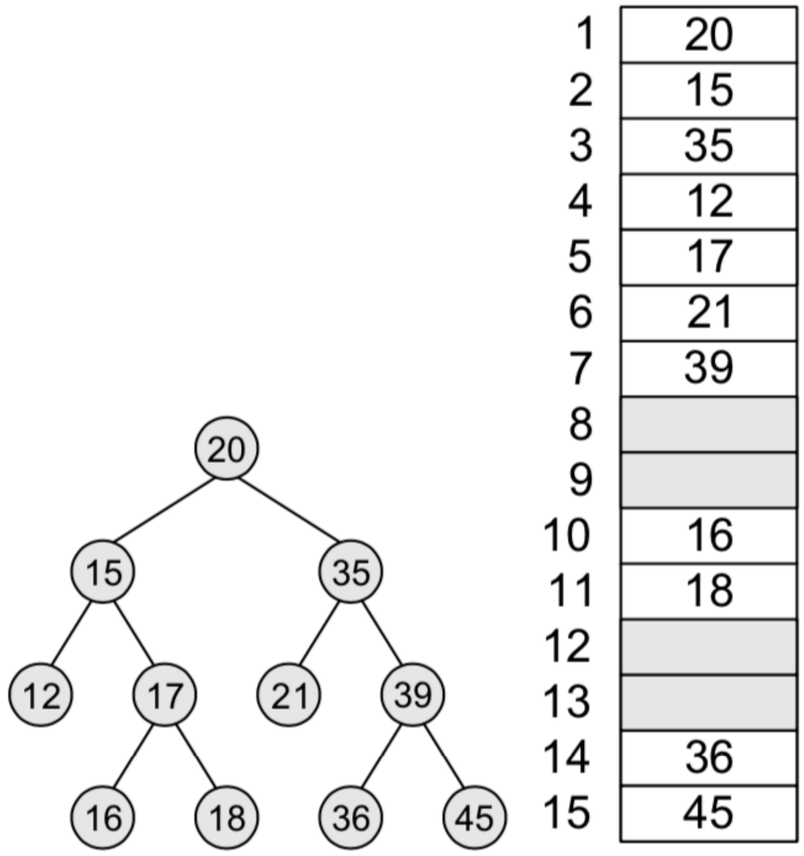


Figure 9.12 Binary tree and its sequential representation

Binary Search Trees

★ A binary search tree, also known as an ordered binary tree, is a variant of binary tree in which the nodes are arranged in an order.

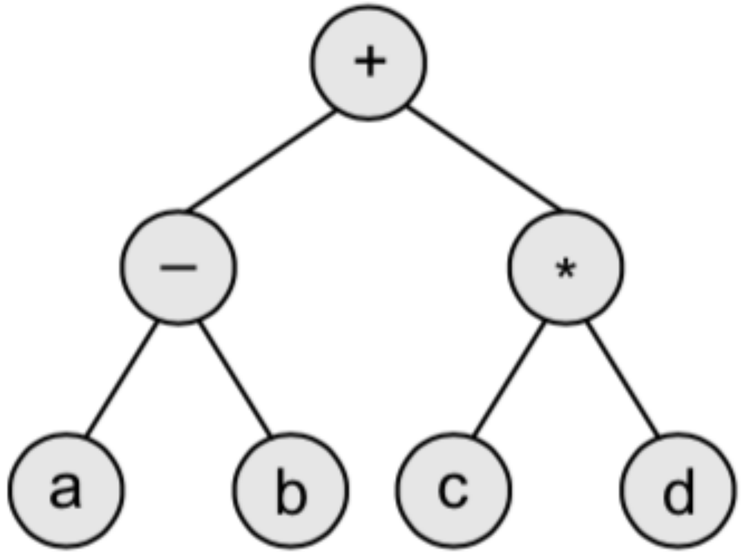
★ We will discuss the concept of binary search trees and different operations performed on them in the next chapter.

Expression Trees

Binary trees are widely used to store algebraic expressions. For example, consider the algebraic expression given as:

```
Exp = (a - b) + (c * d)
```

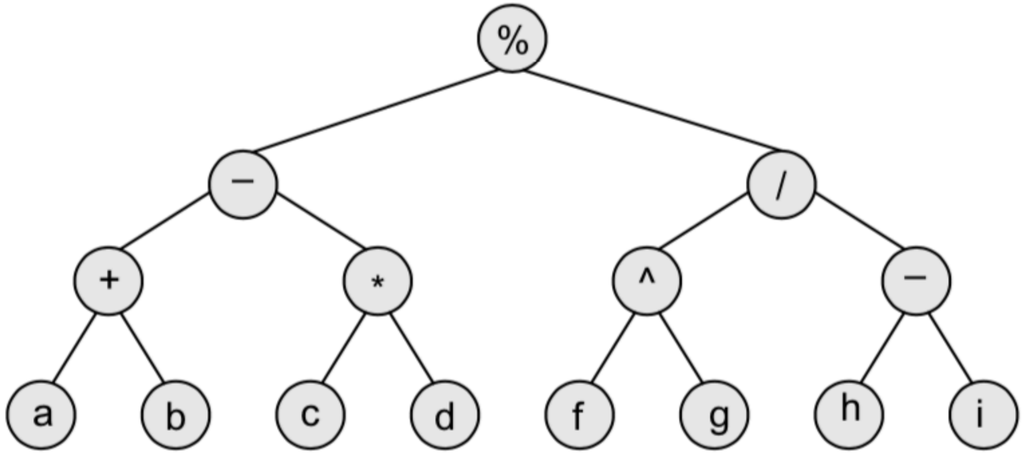
This expression can be represented using a binary tree as shown in Fig. 9.13.



Expression Trees

Example 9.2 Given an expression, `Exp = ((a + b) - (c * d)) % ((e ^f) / (g - h))`, construct the corresponding binary tree.

Solution

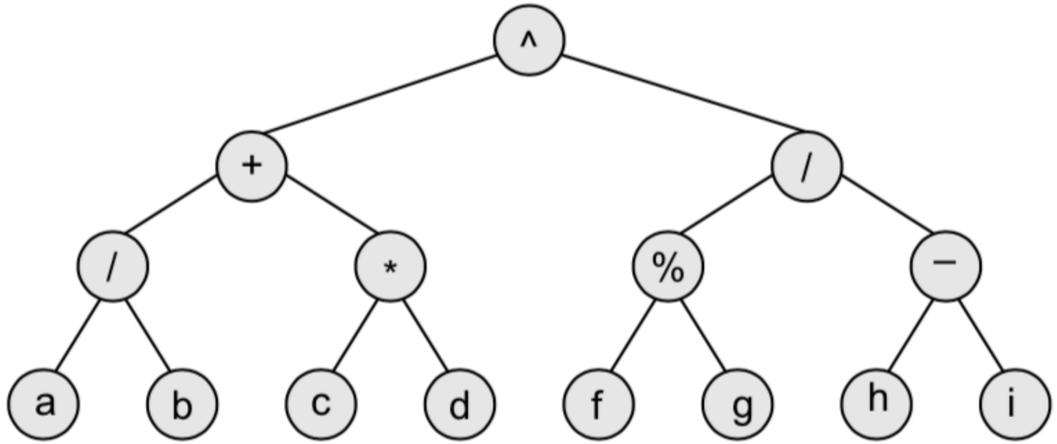


Expression tree

Expression Trees

Example 9.3 Given the binary tree, write down the expression that it represents.

Solution



Expression for the above binary tree is

```
[{(a/b) + (c*d)} ^ {(f % g)/(h – i)}]
```

Tree Traversal

- Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way.
- Unlike linear data structures in which the elements are traversed sequentially, tree is a non- linear data structure in which the elements can be traversed in many different ways.
- There are different algorithms for tree traversals. These algorithms differ in the order in which the nodes are visited.

Pre-Order Traversal

To traverse a non-empty binary tree in pre-order, the following operations are performed recursively at each node. The algorithm works by:

1. Visiting the root node,
2. Traversing the left sub-tree, and finally
3. Traversing the right sub-tree.

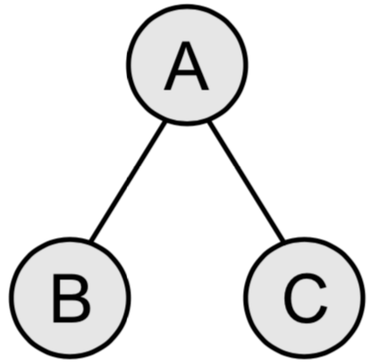


Figure 9.15 Binary tree

Pre-Order Traversal

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:             Write TREE -> DATA
Step 3:             PREORDER(TREE -> LEFT)
Step 4:             PREORDER(TREE -> RIGHT)
        [END OF LOOP]
Step 5: END
```

Figure 9.16 Algorithm for pre-order traversal

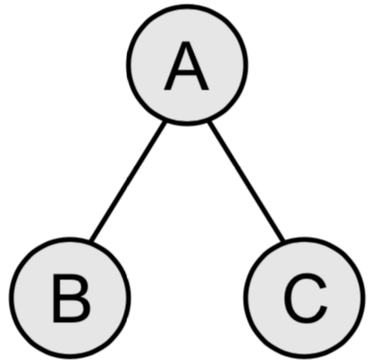


Figure 9.15 Binary tree

Pre-order traversal algorithms are used to extract a prefix notation from an expression tree. For example, consider the expressions

```
+ - a b * c d (from Fig. 9.13)
% - + a b * c d / ^ e f - g h (from Fig of Example 9.2)
^ + / a b * c d / % f g - h i (from Fig of Example 9.3)
```

Pre-Order Traversal

Pre-order traversal algorithms are used to extract a prefix notation from an expression tree. For example, consider the expressions

```
+ - a b * c d (from Fig. 9.13)
% - + a b * c d / ^ e f - g h (from Fig of Example 9.2)
^ + / a b * c d / % f g - h i (from Fig of Example 9.3)
```

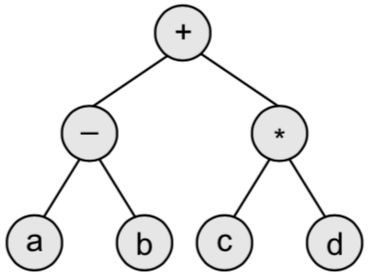
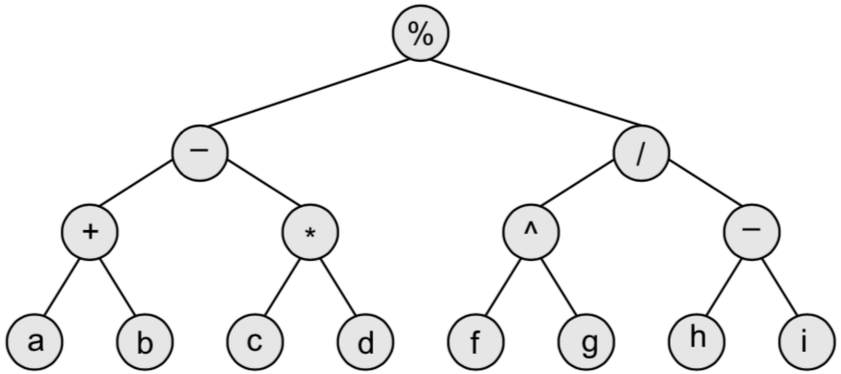
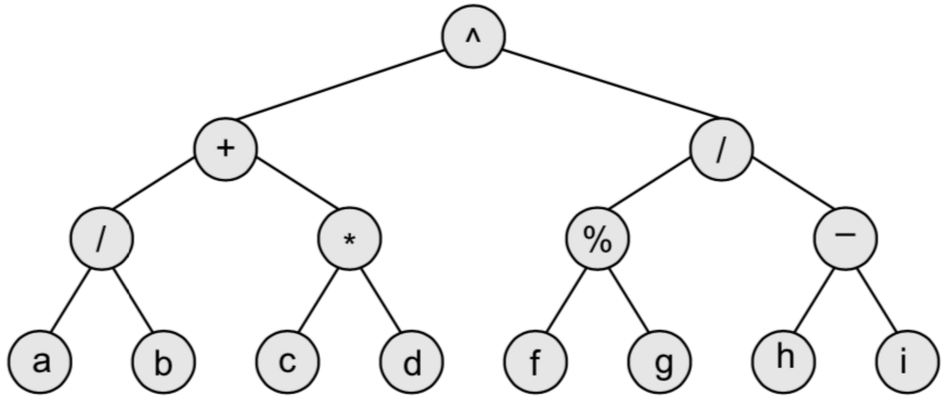


Figure 9.13 Expression tree



Expression tree

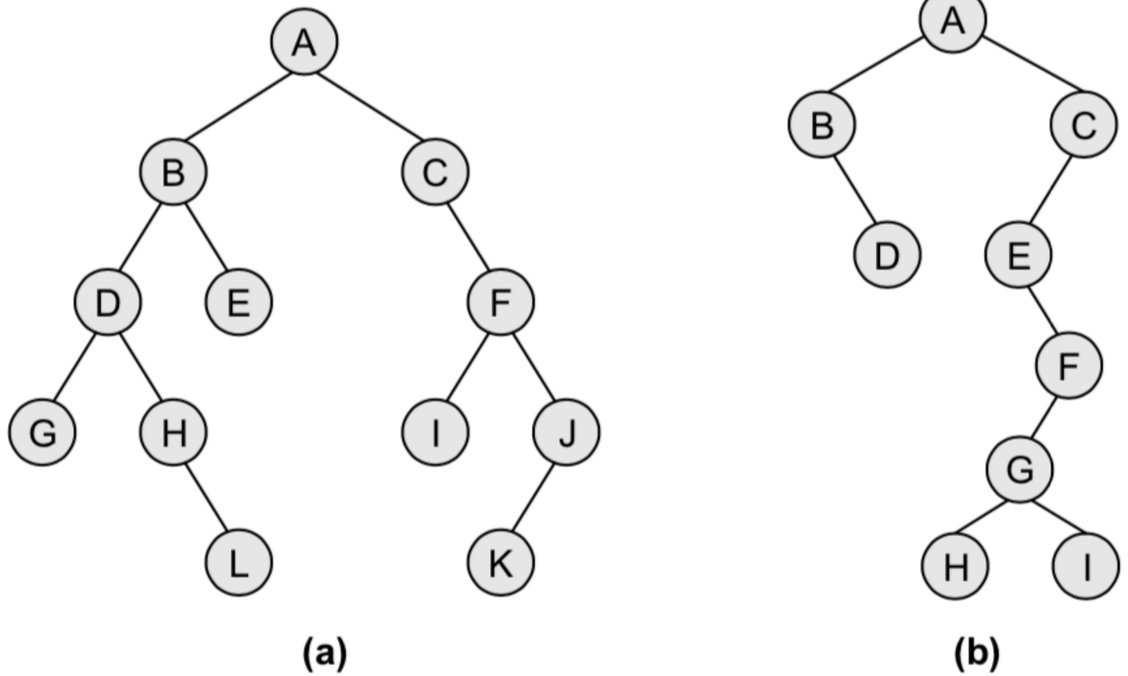


Pre-Order Traversal

Example 9.6 In Figs (a) and (b), find the sequence of nodes that will be visited using pre-order traversal algorithm.

Solution

```
TRAVERSAL ORDER: A, B, D, G, H, L, E, C, F, I, J,
                 and K
TRAVERSAL ORDER: A, B, D, C, D, E, F, G, H, and I
```



In-Order Traversal

To traverse a non-empty binary tree in in-order, the following operations are performed recursively at each node. The algorithm works by:

1. Traversing the left sub-tree,
2. Visiting the root node, and finally
3. Traversing the right sub-tree.

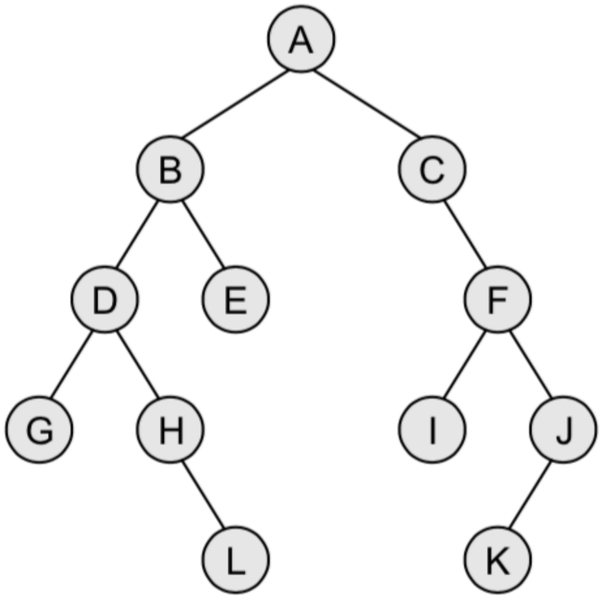
```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:            INORDER(TREE -> LEFT)
Step 3:            Write TREE -> DATA
Step 4:            INORDER(TREE -> RIGHT)
       [END OF LOOP]
Step 5: END
```

Figure 9.17 Algorithm for in-order traversal

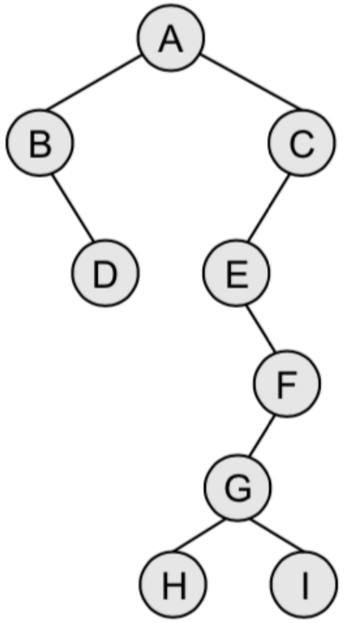
In-Order Traversal

Example 9.7 For the trees given in Example 9.6, find the sequence of nodes that will be visited using in-order traversal algorithm.

```
TRAVERSAL ORDER: G, D, H, L, B, E, A, C, I, F, K, and J
TRAVERSAL ORDER: B, D, A, E, H, G, I, F, and C
```



(a)



(b)

Post-Order Traversal

To traverse a non-empty binary tree in post-order, the following operations are performed recursively at each node. The algorithm works by:

1. Traversing the left sub-tree,
2. Traversing the right sub-tree, and finally
3. Visiting the root node.

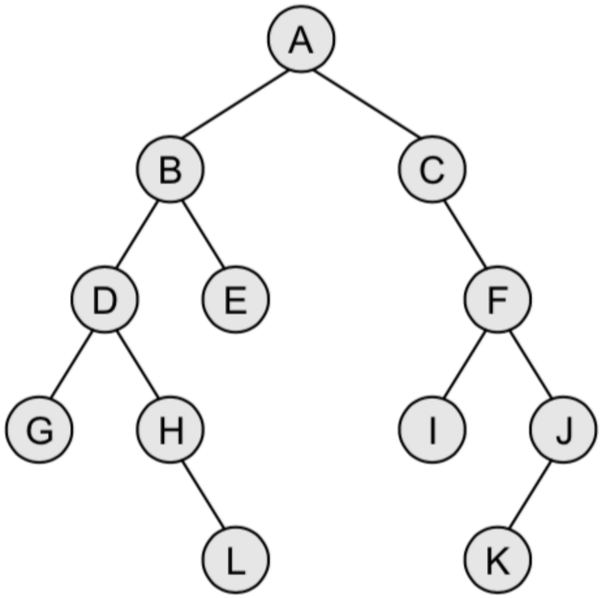
```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:            POSTORDER(TREE -> LEFT)
Step 3:            POSTORDER(TREE -> RIGHT)
Step 4:            Write TREE -> DATA
       [END OF LOOP]
Step 5: END
```

Figure 9.18 Algorithm for post-order traversal

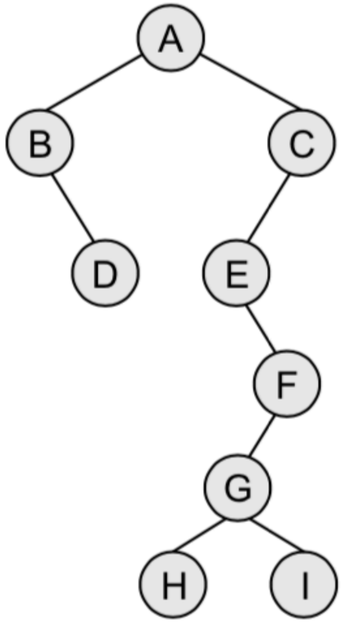
Post-Order Traversal

Example 9.8 For the trees given in Example 9.6, give the sequence of nodes that will be visited using post-order traversal algorithm.

```
TRAVERSAL ORDER: G, L, H, D, E, B, I, K, J, F, C, and A
TRAVERSAL ORDER: D, B, H, I, G, F, E, C, and A
```



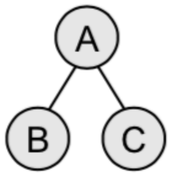
(a)



(b)

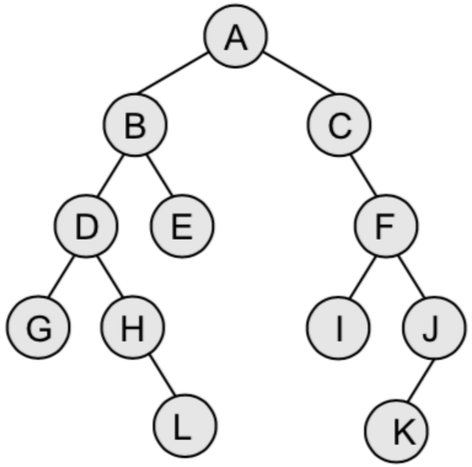
Level-Order Traversal

In level-order traversal, all the nodes at a level are accessed before going to the next level. This algorithm is also called as the *breadth-first traversal algorithm*. Consider the trees given in Fig. 9.19 and note the level order of these trees.



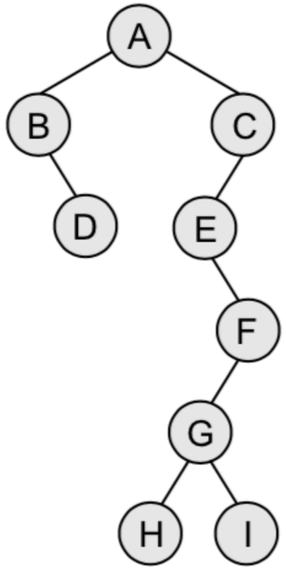
(a)

TRAVERSAL ORDER:
A, B, and C



(b)

TRAVERSAL ORDER:
A, B, C, D, E, F, G, H, I, J, L, and K



(c)

TRAVERSAL ORDER:
A, B, C, D, E, F, G, H, and I

Constructing a Binary Tree from Traversal Results

Step 1 Use the pre-order sequence to determine the root node of the tree. The first element would be the root node.

Step 2 Elements on the left side of the root node in the in-order traversal sequence form the left sub-tree of the root node. Similarly, elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node.

Step 3 Recursively select each element from pre-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence. Look at Fig. 9.20 which constructs the tree from its traversal results. Now consider the in-order traversal and post-order traversal sequences of a given binary tree. Before constructing the binary tree, remember that in post-order traversal the root node is the last node. Rest of the steps will be the same as mentioned above Fig. 9.21.

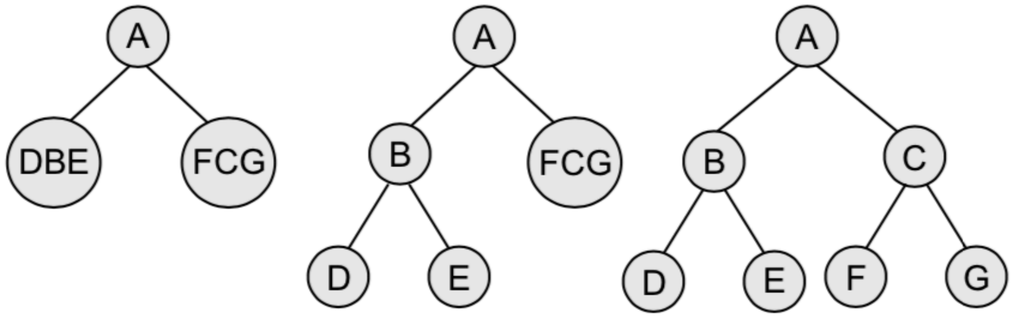


Figure 9.20

In–order Traversal: D B H E I A F J C G Post order Traversal: D H I E B J F G C A

Constructing a Binary Tree from Traversal Results

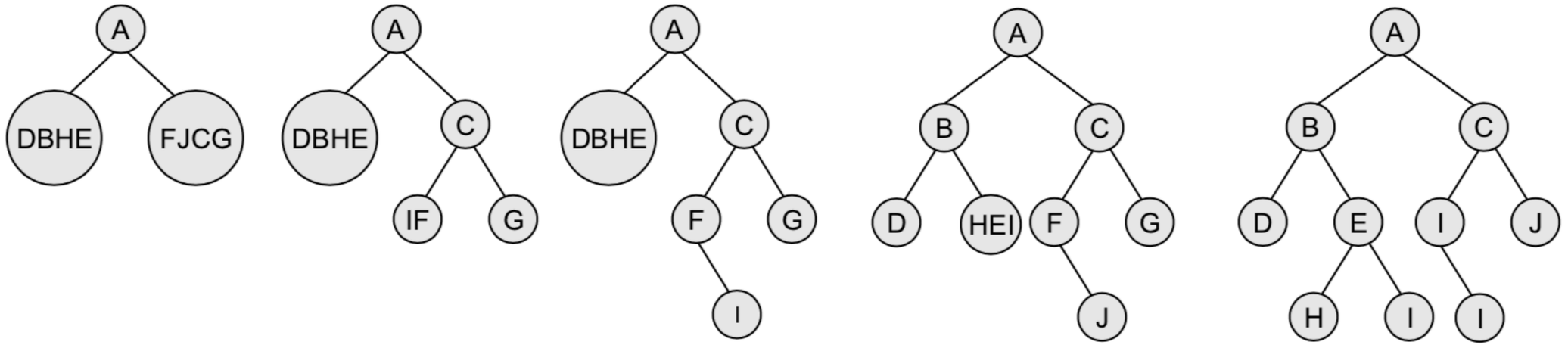


Figure 9.21 Steps to show binary tree

Huffman's Tree

★ Huffman coding is an entropy encoding algorithm developed by David A. Huffman that is widely used as a lossless data compression technique.

★ The Huffman coding algorithm uses a variable- length code table to encode a source character where the variable-length code table is derived on the basis of the estimated probability of occurrence of the source character.

★ The key idea behind Huffman algorithm is that it encodes the most common characters using shorter strings of bits than those used for less common source characters.

Huffman's Tree

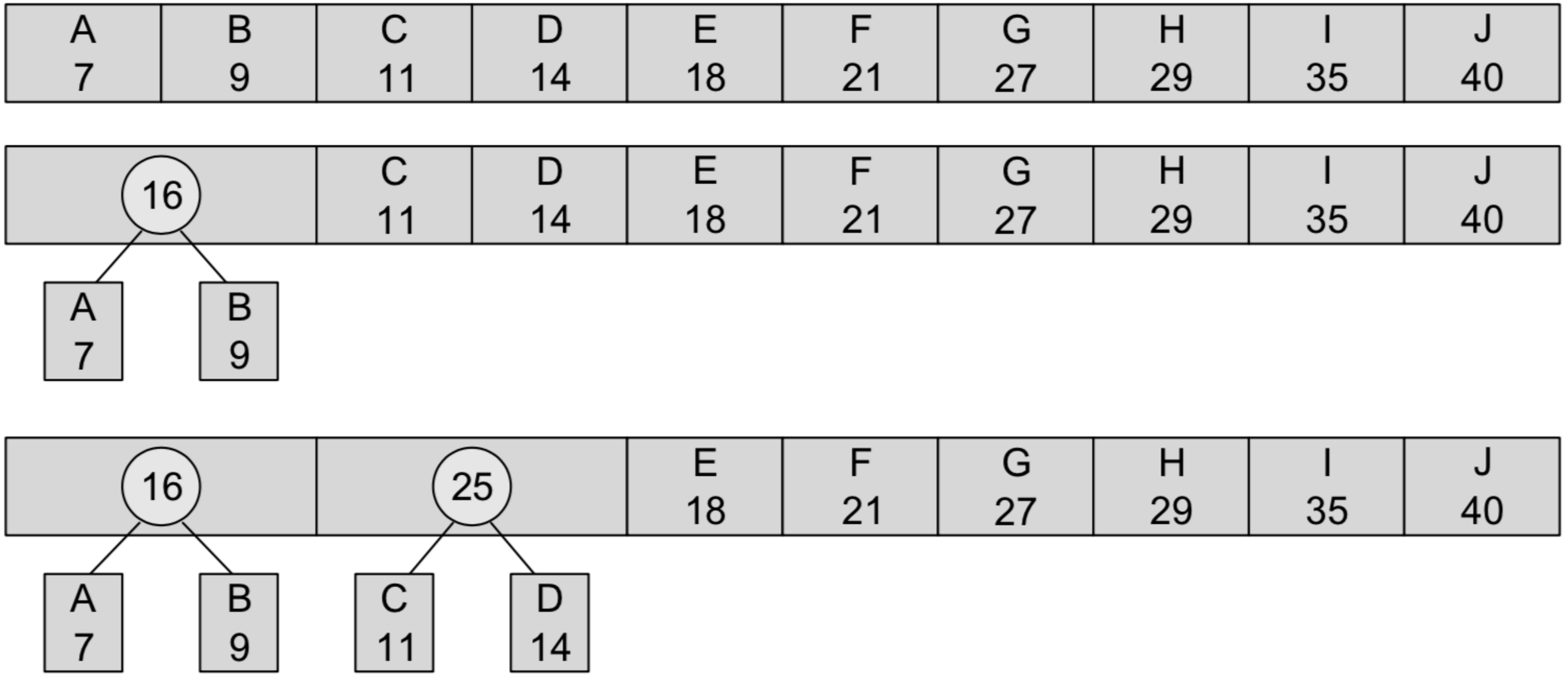
The Huffman algorithm can be implemented using a priority queue in which all the nodes are placed in such a way that the node with the lowest weight is given the highest priority. The algorithm is shown in Fig. 9.23.

```
Step 1: Create a leaf node for each character. Add the character and its weight or frequency
        of occurrence to the priority queue.
Step 2: Repeat Steps 3 to 5 while the total number of nodes in the queue is greater than 1.
Step 3: Remove two nodes that have the lowest weight (or highest priority).
Step 4: Create a new internal node by merging these two nodes as children and with weight
        equal to the sum of the two nodes' weights.
Step 5: Add the newly created node to the queue.
```

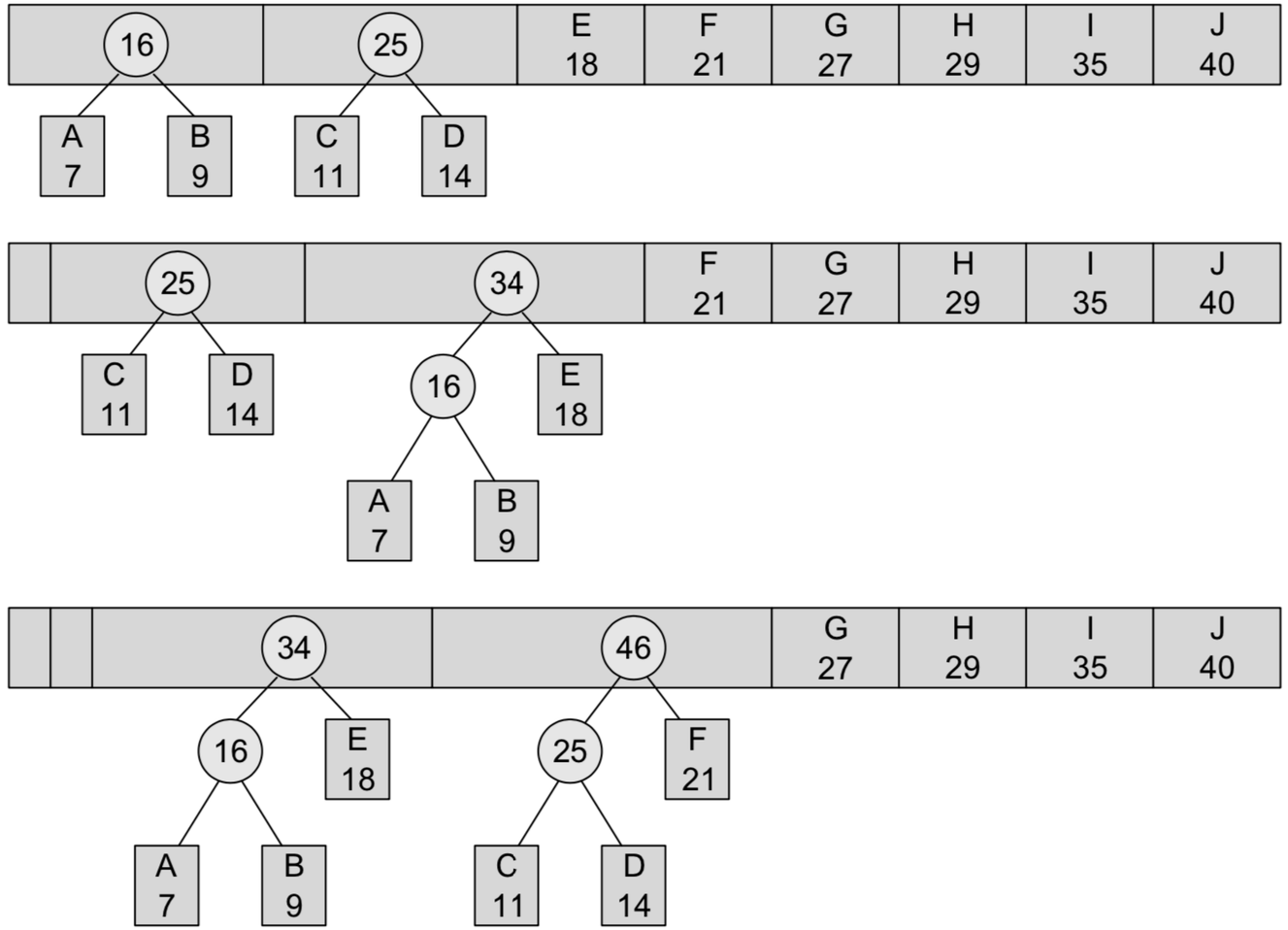
Figure 9.23 Huffman algorithm

Example: Huffman's Tree

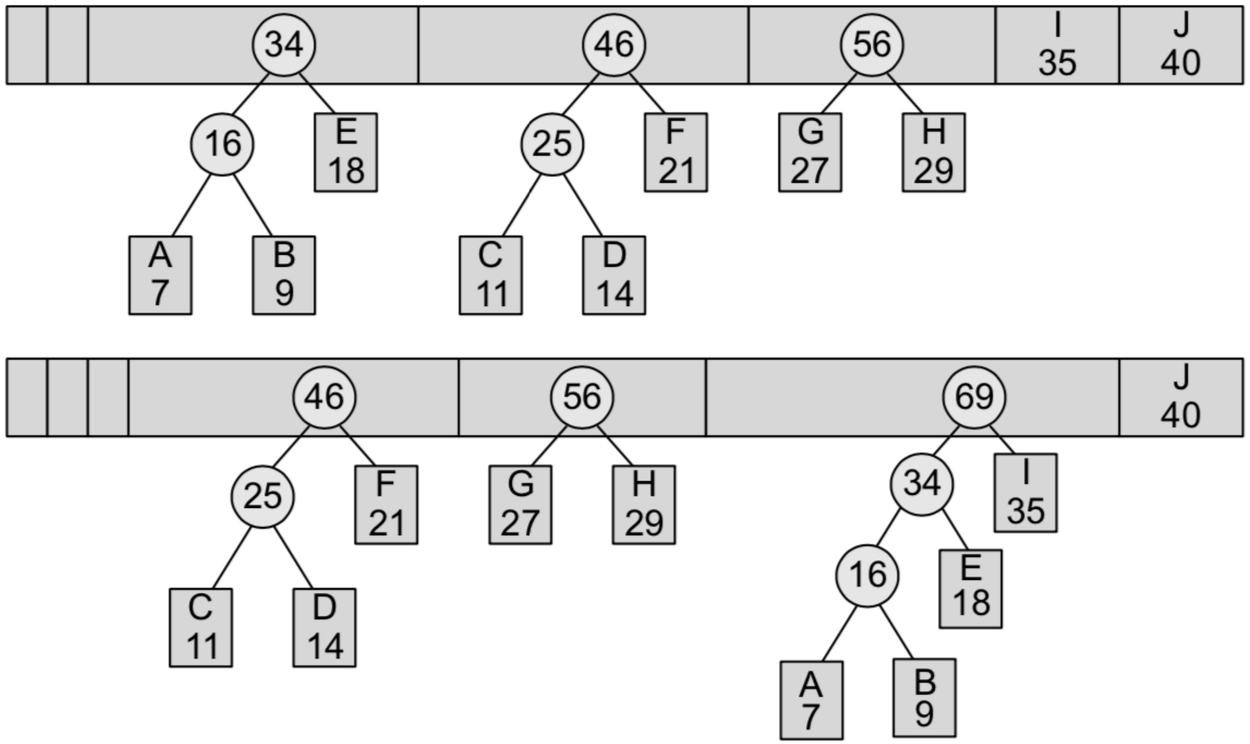
Example 9.10 Create a Huffman tree with the following nodes arranged in a priority queue.



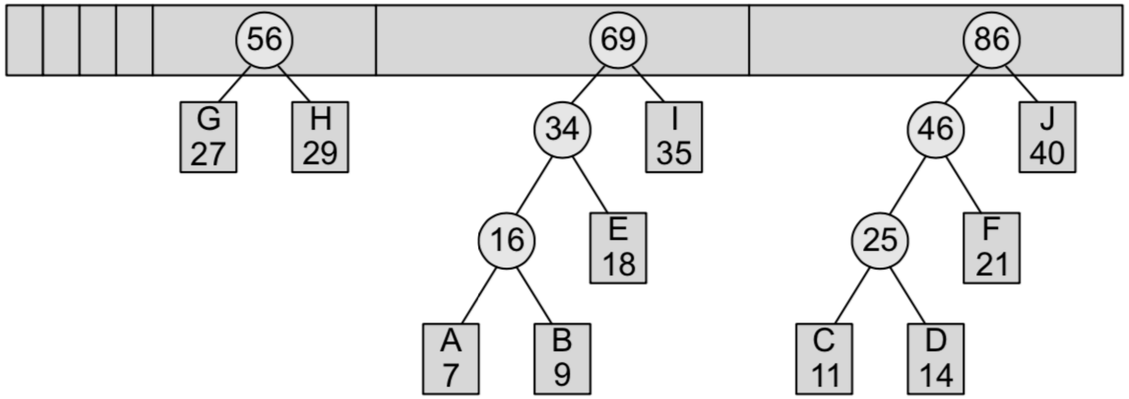
Example: Huffman's Tree

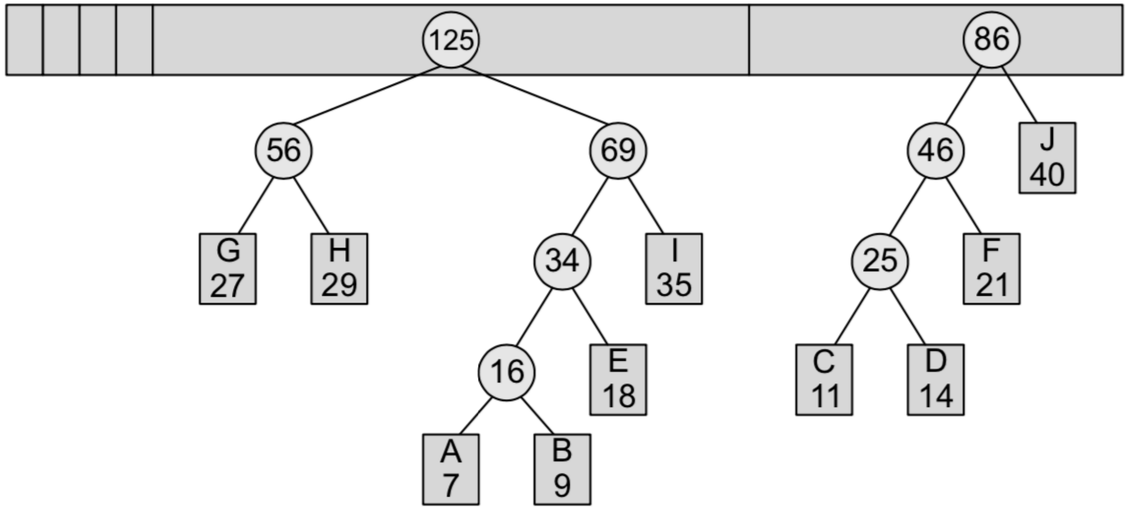


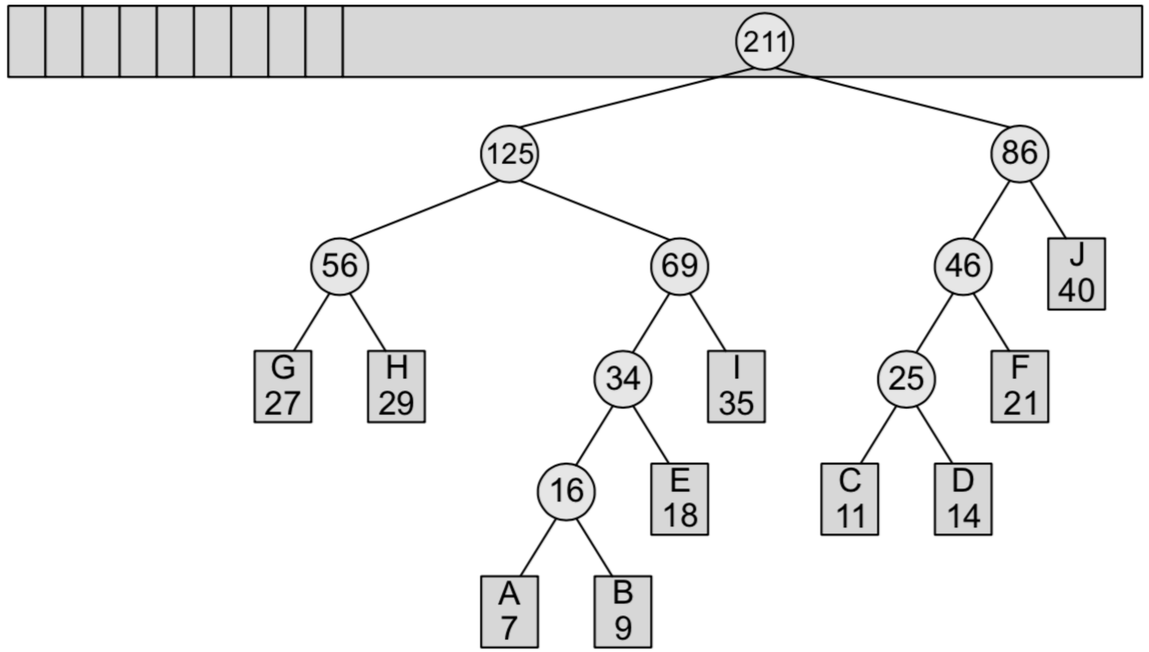
Example: Huffman's Tree



Example: Huffman's Tree







Example: Huffman's Tree

Table 9.3 Characters with their codes

Character	Code
A	00
E	01
R	11
W	1010
X	1000
Y	1001
Z	1011

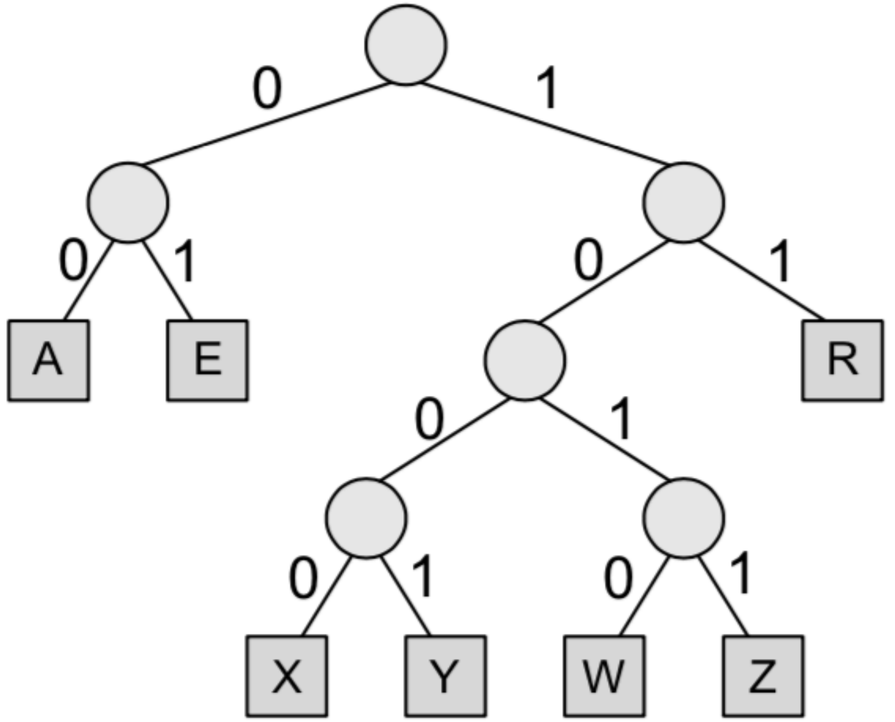


Figure 9.24 Huffman tree

Applications of Trees

- Trees are used to store simple as well as complex data. Here simple means an integer value, character value and complex data means a structure or a record.
- Trees are often used for implementing other types of data structures like hash tables, sets, and maps.
- A self-balancing tree, Red-black tree is used in kernel scheduling, to preempt massively multi-processor computer operating system use. (We will study red-black trees in next chapter.)
- Another variation of tree, B-trees are prominently used to store tree structures on disc. They are used to index a large number of records. (We will study B-Trees in Chapter 11.)
- B-trees are also used for secondary indexes in databases, where the index facilitates a select operation to answer some range criteria.
- Trees are an important data structure used for compiler construction.
- Trees are also used in database design.
- Trees are used in file system directories.
- Trees are also widely used for information storage and retrieval in symbol tables.

Question and Answer