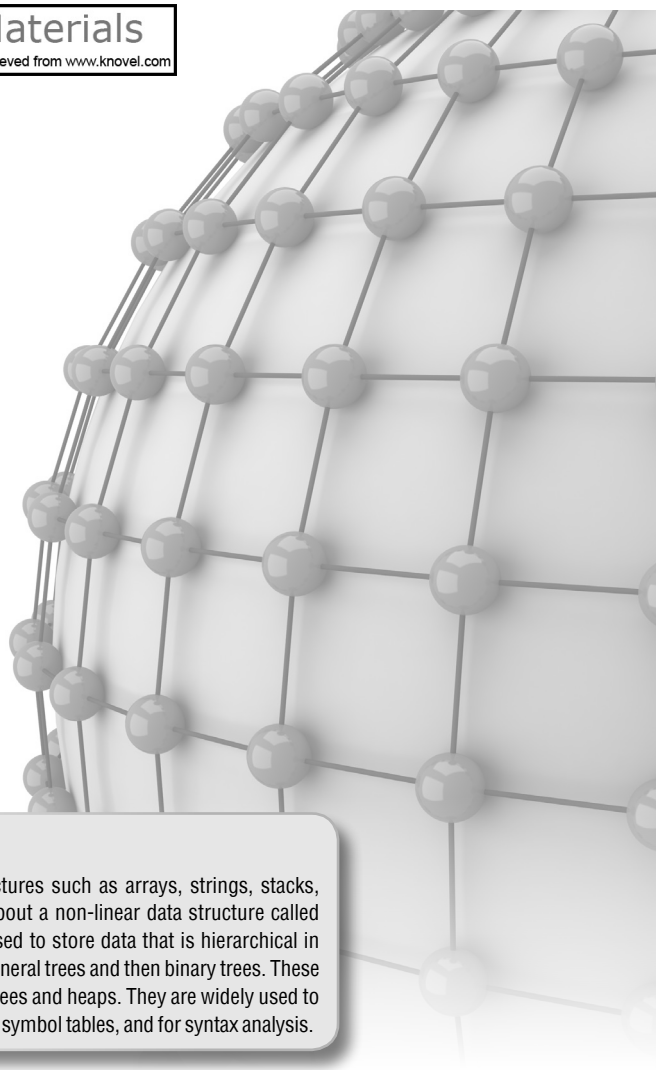
CHAPTER 9

Trees



LEARNING OBJECTIVE

So far, we have discussed linear data structures such as arrays, strings, stacks, and queues. In this chapter, we will learn about a non-linear data structure called tree. A tree is a structure which is mainly used to store data that is hierarchical in nature. In this chapter, we will first discuss general trees and then binary trees. These binary trees are used to form binary search trees and heaps. They are widely used to manipulate arithmetic expressions, construct symbol tables, and for syntax analysis.

9.1 INTRODUCTION

A tree is recursively defined as a set of one or more nodes where one node is designated as the root of the tree and all the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root. Figure 9.1 shows a tree where node A is the root node; nodes B, C, and D are children of the root node and form sub-trees of the tree rooted at node A.

9.1.1 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.

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.

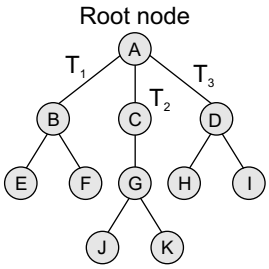


Figure 9.1 Tree

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`.

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.

9.2 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

9.2.1 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.

Although general trees can be represented as ADTs, there is always a problem when another sub-tree is added to a node that already has the maximum number of sub-trees attached to it. Even the algorithms for searching, traversing, adding, and deleting nodes become much more complex as there are not just two possibilities for any node but multiple possibilities.

To overcome the complexities of a general tree, it may be represented as a graph data structure (to be discussed later), thereby losing many of the advantages of the tree processes. Therefore, a better option is to convert general trees into binary trees.

A general tree when converted to a binary tree may not end up being well formed or full, but the advantages of such a conversion enable the programmer to use the algorithms for processes that are used for binary trees with minor modifications.

9.2.2 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.

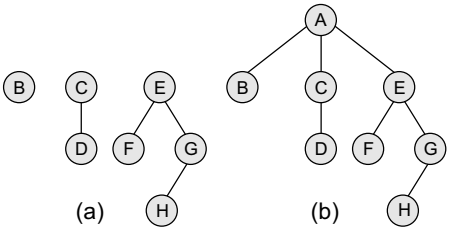


Figure 9.2 Forest and its corresponding tree

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.

We can convert a forest into a tree by adding a single node as the root node of the tree. For example, Fig. 9.2(a) shows a forest and Fig. 9.2(b) shows the corresponding tree.

Similarly, we can convert a general tree into a forest by deleting the root node of the tree.

9.2.3 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. 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.

In the tree, root node 1 has two successors: 2 and 3. Node 2 has two successor nodes: 4 and 5. Node 4 has two successors: 8 and 9. Node 5 has no successor. Node 3 has two successor nodes: 6 and 7. Node 6 has two successors: 10 and 11. Finally, node 7 has only one successor: 12.

A binary tree is recursive by definition as every node in the tree contains a left sub-tree and a right sub-tree. Even the terminal nodes contain an empty left sub-tree and an empty right sub-tree. Look at Fig. 9.3, nodes 5, 8, 9, 10, 11, and 12 have no successors and thus said to have empty sub-trees.

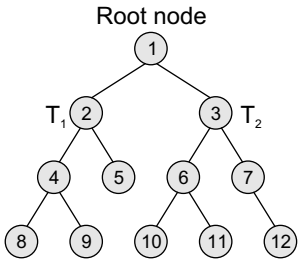


Figure 9.3 Binary tree

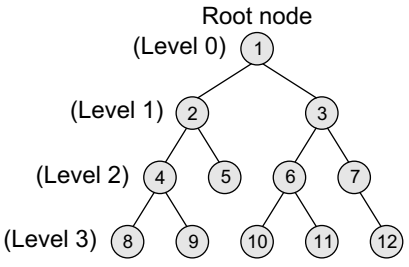


Figure 9.4 Levels in binary tree

Terminology

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.

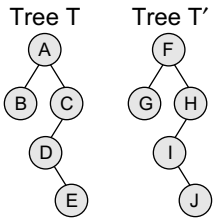


Figure 9.5 Similar binary trees

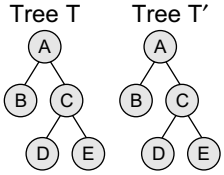


Figure 9.6 T′ is a copy of T

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.

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.

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.

A binary tree of height h has at least h nodes and at most $2^h - 1$ nodes. This is because every level will have at least one node and can have at most 2 nodes. So, if every level has two nodes then a tree with height h will have at the most $2^h - 1$ nodes as at level 0, there is only one element called the root. The height of a binary tree with n nodes is at least $\log_2(n+1)$ and at most n.

In-degree/out-degree of a node It is the number of edges arriving at a node. The root node is the only node that has an in-degree equal to zero. Similarly, *out-degree* of a node is the number of edges leaving that node.

Binary trees are commonly used to implement binary search trees, expression trees, tournament trees, and binary heaps.

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.

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.

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:

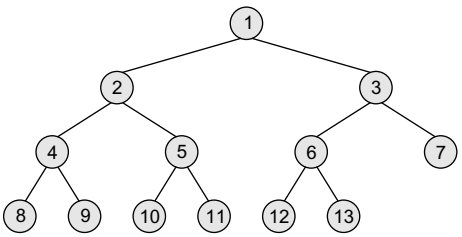


Figure 9.7 Complete binary tree

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

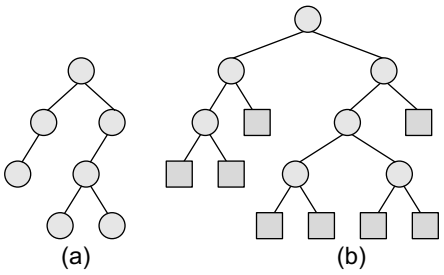


Figure 9.8 (a) Binary tree and (b) extended binary tree

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

Extended Binary Trees

A binary tree T is said to be an extended binary tree (or a 2-tree) if each node in the tree has either no child or exactly two children. Figure 9.8 shows how an ordinary binary tree is converted into an extended binary tree.

In an extended binary tree, nodes having two children are called *internal nodes* and nodes having no children are called *external nodes*. In Fig. 9.8, the internal nodes are represented using circles and the external nodes are represented using squares.

To convert a binary tree into an extended tree, every empty sub-tree is replaced by a new node. The original nodes in the tree are the internal nodes, and the new nodes added are called the external nodes.

Representation of Binary Trees in the Memory

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;
};
```

Every binary tree has a pointer ROOT, which points to the root element (topmost element) of the tree. If ROOT = NULL, then the tree is empty. Consider the binary tree given in Fig. 9.3. The schematic diagram of the linked representation of the binary tree is shown in Fig. 9.9.

In Fig. 9.9, the left position is used to point to the left child of the node or to store the address of the left child of the node. The middle position is used to store the data. Finally, the right position is used to point to the right child of the node or to store the address of the right child of the node. Empty sub-trees are represented using X (meaning NULL).

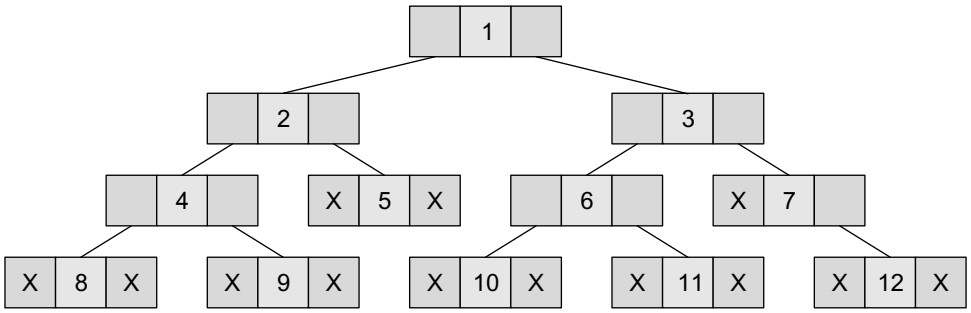


Figure 9.9 Linked representation of a binary tree

Look at the tree given in Fig. 9.10. Note how this tree is represented in the main memory using a linked list (Fig. 9.11).

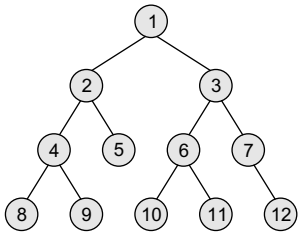


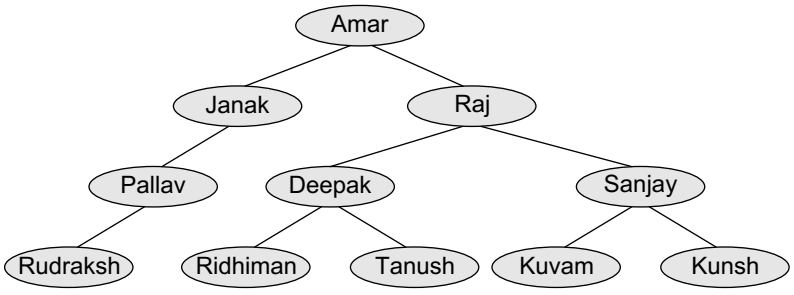
Figure 9.10 Binary tree T

		LEFT	DATA	RIGHT
ROOT 3	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			
15 AVAIL	16	–1	11	–1
	17			
	18	–1	12	–1
	19			
	20	2	6	16

Figure 9.11 Linked representation of binary tree T

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



		LEFT	NAMES	RIGHT
ROOT 3	1	12	Pallav	–1
	2			
	3	9	Amar	13
	4			
	5			
	6	19	Deepak	17
	7			
7 AVAIL	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

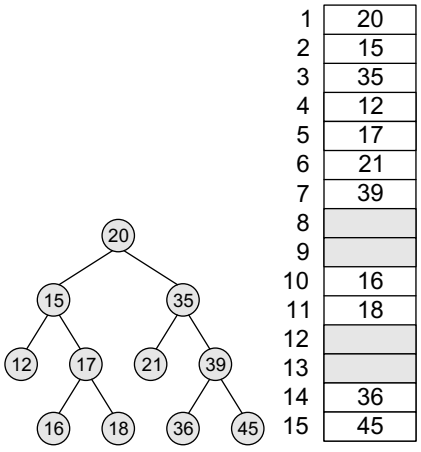


Figure 9.12 Binary tree and its sequential representation

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 shows a binary tree and its corresponding sequential representation. The tree has 11 nodes and its height is 4.

9.2.4 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.

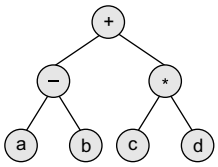


Figure 9.13 Expression tree

9.2.5 Expression Trees

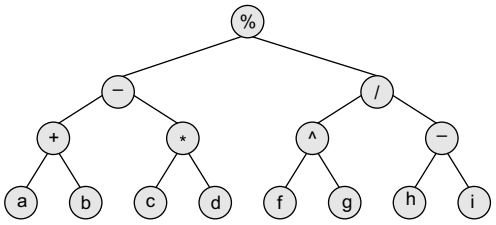
Binary trees are widely used to store algebraic expressions. For example, consider the algebraic expression given as:

```
Exp = (a – b) + (c * d)
```

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

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

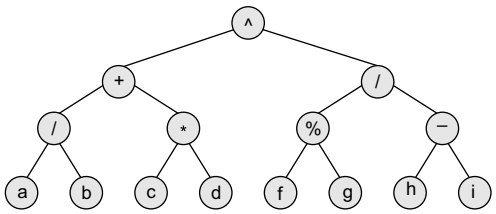
Solution

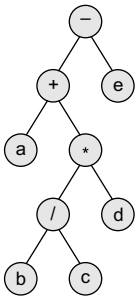


Expression tree

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

Solution





Expression tree

Expression for the above binary tree is
`[{(a/b) + (c*d)} ^ {(f % g)/(h – i)}]`

Example 9.4 Given the expression, `Exp = a + b / c * d – e`, construct the corresponding binary tree.

Solution Use the operator precedence chart to find the sequence in which operations will be performed. The given expression can be written as
`Exp = ((a + ((b/c) * d)) – e)`

9.2.6 Tournament Trees

We all know that in a tournament, say of chess, `n` number of players participate. To declare the winner among all these players, a couple of matches are played and usually three rounds are played in the game.

In every match of round 1, a number of matches are played in which two players play the game against each other. The number of matches that will be played in round 1 will depend on the number of players. For example, if there are 8 players participating in a chess tournament, then 4 matches will be played in round 1. Every match of round 1 will be played between two players.

Then in round 2, the winners of round 1 will play against each other. Similarly, in round 3, the winners of round 2 will play against each other and the person who wins round 3 is declared the winner. Tournament trees are used to represent this concept.

In a tournament tree (also called a *selection tree*), each external node represents a player and each internal node represents the winner of the match played between the players represented by its children nodes. These tournament trees are also called *winner trees* because they are being used to record the winner at each level. We can also have a *loser tree* that records the loser at each level.

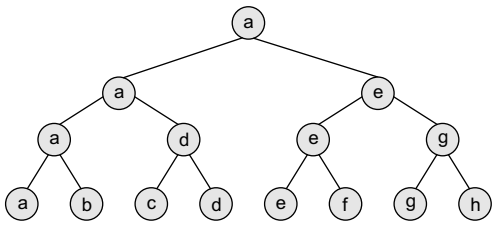
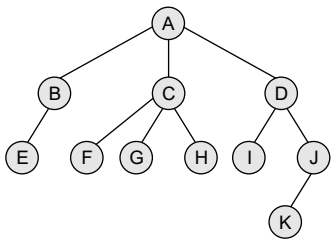


Figure 9.14 Tournament tree

Consider the tournament tree given in Fig. 9.14. There are 8 players in total whose names are represented using `a`, `b`, `c`, `d`, `e`, `f`, `g`, and `h`. In round 1, `a` and `b`; `c` and `d`; `e` and `f`; and finally `g` and `h` play against each other. In round 2, the winners of round 1, that is, `a`, `d`, `e`, and g play against each other. In round 3, the winners of round 2, `a` and `e` play against each other. Whosoever wins is declared the winner. In the tree, the root node `a` specifies the winner.

9.3 CREATING A BINARY TREE FROM A GENERAL TREE

The rules for converting a general tree to a binary tree are given below. Note that a general tree is converted into a binary tree and not a binary search tree.

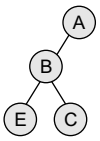


Rule 1: Root of the binary tree = Root of the general tree
Rule 2: Left child of a node = Leftmost child of the node
in the binary tree in the general tree
Rule 3: Right child of a node
in the binary tree = Right sibling of the node in the general tree

Example 9.5 Convert the given general tree into a binary tree.

Step 1

Step 2



Step 3

Now let us build the binary tree.

Step 1: Node A is the root of the general tree, so it will also be the root of the binary tree.

Step 2: Left child of node A is the leftmost child of node A in the general tree and right child of node A is the right sibling of the node A in the general tree. Since node A has no right sibling in the general tree, it has no right child in the binary tree.

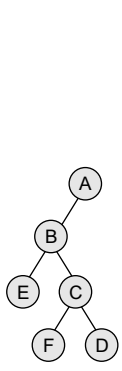
Step 3: Now process node B. Left child of B is E and its right child is C (right sibling in general tree).

Step 4: Now process node C. Left child of C is F (leftmost child) and its right child is D (right sibling in general tree).

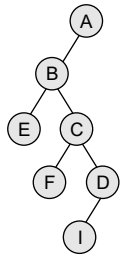
Step 5: Now process node D. Left child of D is I (leftmost child). There will be no right child of D because it has no right sibling in the general tree.

Step 6: Now process node I. There will be no left child of I in the binary tree because I has no left child in the general tree. However, I has a right sibling J, so it will be added as the right child of I.

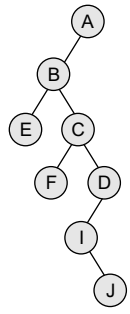
Step 7: Now process node J. Left child of J is K (leftmost child). There will be no right child of J because it has no right sibling in the general tree.



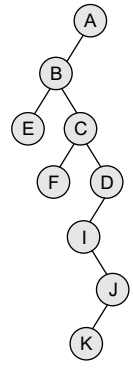
Step 4



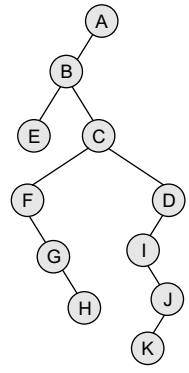
Step 5



Step 6



Step 7



Step 8

Step 8: Now process all the unprocessed nodes (E, F, G, H, K) in the same fashion, so the resultant binary tree can be given as follows.

9.4 TRAVERSING A BINARY TREE

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. In this section, we will discuss these algorithms.

9.4.1 Pre-order Traversal

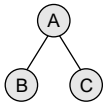


Figure 9.15 Binary tree

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.

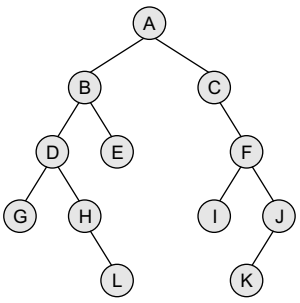
Consider the tree given in Fig. 9.15. The pre-order traversal of the tree is given as A, B, C. Root node first, the left sub-tree next, and then the right sub-tree. Pre-order traversal is also called as *depth-first traversal*. In this algorithm, the left sub-tree is always traversed before the right sub-tree. The word 'pre' in the pre-order specifies that the root node is accessed prior to any other nodes in the left and right sub-trees. Pre-order algorithm is also known as the NLR traversal algorithm (Node-Left-Right). The algorithm for pre-order traversal is shown in Fig. 9.16.

```
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
```

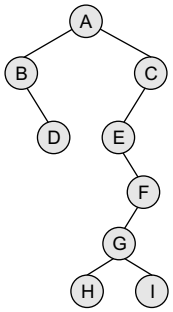
Figure 9.16 Algorithm for pre-order traversal

Pre-order traversal algorithms are used to extract a prefix notation from an expression tree. For example, consider the expressions given below. When we traverse the elements of a tree using the pre-order traversal algorithm, the expression that we get is a prefix expression.

```
+ - 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)
```



(a)



(b)

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
```

9.4.2 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.

Consider the tree given in Fig. 9.15. The in-order traversal of the tree is given as B, A, and C. Left sub-tree first, the root node next, and then the right sub-tree. In-order traversal is also called as *symmetric traversal*. In this algorithm, the left sub-tree is always traversed before the root node and the right sub-tree. The word 'in' in the in-order specifies that the root node is accessed in between the left and the right sub-trees. In-order algorithm is also known as the LNR traversal algorithm (Left-Node-Right). The algorithm for in-order traversal is shown in Fig. 9.17.

```
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 algorithm is usually used to display the elements of a binary search tree. Here, all the elements with a value lower than a given value are accessed before the elements with a higher value. We will discuss binary search trees in detail in the next chapter.

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
```

9.4.3 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

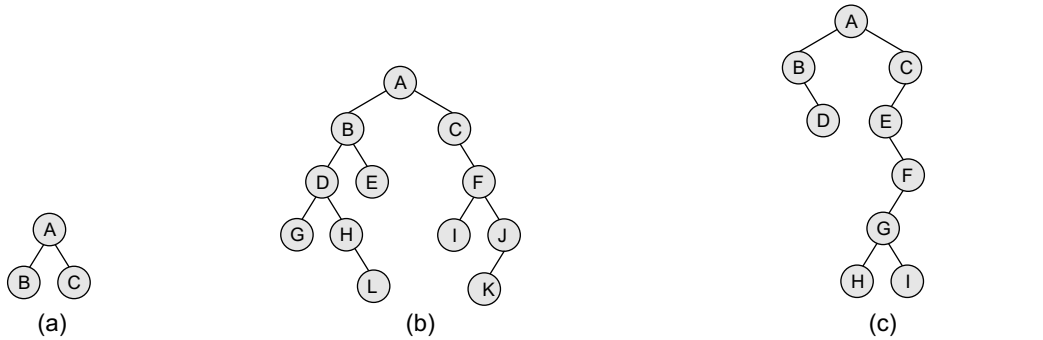
Consider the tree given in Fig. 9.18. The post-order traversal of the tree is given as B, C, and A. Left sub-tree first, the right sub-tree next, and finally the root node. In this algorithm, the left sub-tree is always traversed before the right sub-tree and the root node. The word 'post' in the post-order specifies that the root node is accessed after the left and the right sub-trees. Post-order algorithm is also known as the LRN traversal algorithm (Left-Right-Node). The algorithm for post-order traversal is shown in Fig. 9.18. Post-order traversals are used to extract postfix notation from an expression tree.

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
```

9.4.4 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.




```
TRAVERSAL ORDER:
A, B, and C

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

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

Figure 9.19 Binary trees

9.4.5 Constructing a Binary Tree from Traversal Results

We can construct a binary tree if we are given at least two traversal results. The first traversal must be the in-order traversal and the second can be either pre-order or post-order traversal. The in-order traversal result will be used to determine the left and the right child nodes, and the pre-order/post-order can be used to determine the root node. For example, consider the traversal results given below:

```
In-order Traversal: D B E A F C G          Pre-order Traversal: A B D E C F G
```

Here, we have the in-order traversal sequence and pre-order traversal sequence. Follow the steps given below to construct the tree:

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.

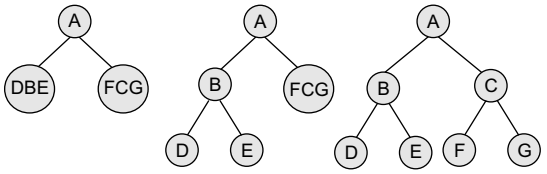


Figure 9.20

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

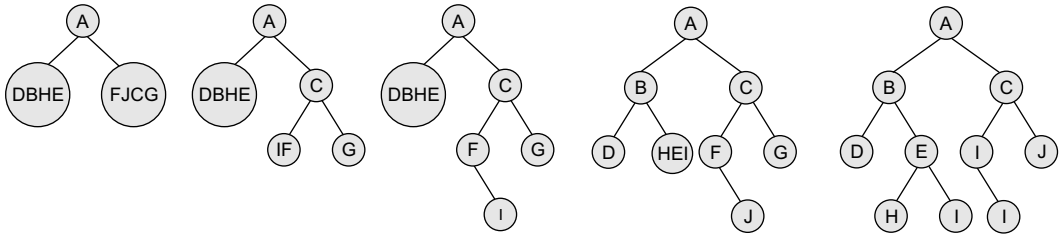


Figure 9.21 Steps to show binary tree

9.5 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.

The algorithm works by creating a binary tree of nodes that are stored in an array. A node can be either a leaf node or an internal node. Initially, all the nodes in the tree are at the leaf level and store the source character and its frequency of occurrence (also known as weight).

While the internal node is used to store the weight and contains links to its child nodes, the external node contains the actual character. Conventionally, a `'0'` represents following the left

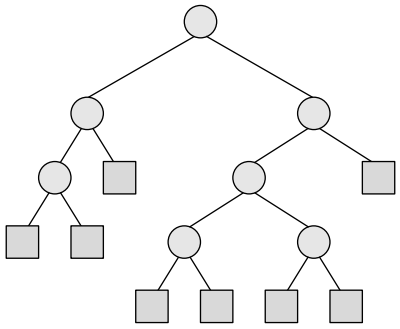


Figure 9.22 Binary tree

child and a '1' represents following the right child. A finished tree that has n leaf nodes will have n - 1 internal nodes.

The running time of the algorithm depends on the length of the paths in the tree. So, before going into further details of Huffman coding, let us first learn how to calculate the length of the paths in the tree. The *external path length* of a binary tree is defined as the sum of all path lengths summed over each path from the root to an external node. The internal path length is also defined in the same manner. The *internal path length* of a binary tree is defined as the sum of all path lengths summed over each path from the root to an internal node. Look at the binary tree given in Fig. 9.22.

The internal path length, $L_I = 0 + 1 + 2 + 1 + 2 + 3 + 3 = 12$

The external path length, $L_E = 2 + 3 + 3 + 2 + 4 + 4 + 4 + 4 = 26$

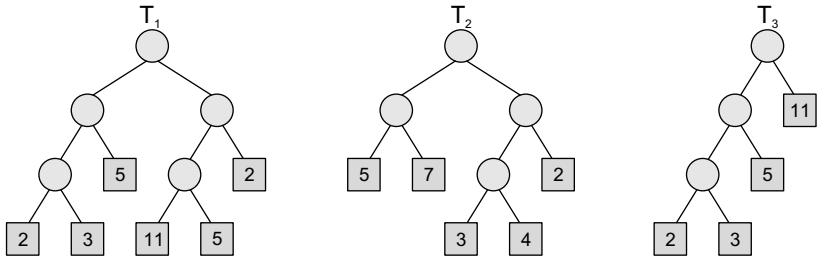
Note that, $L_I + 2 * n = 12 + 2 * 7 = 12 + 14 = 26 = L_E$

Thus, $L_I + 2n = L_E$, where n is the number of internal nodes. Now if the tree has n external nodes and each external node is assigned a weight, then the weighted path length P is defined as the sum of the weighted path lengths.

Therefore, $P = W_1L_1 + W_2L_2 + \ldots + W_nL_n$

where W_i and L_i are the weight and path length of an external node N_i.

Example 9.9 Consider the trees T_1, T_2, and T_3 given below, calculate their weighted external path lengths.



Binary tree

Solution

Weighted external path length of T_1 can be given as,

$P_1 = 2{\cdot}3 + 3{\cdot}3 + 5{\cdot}2 + 11{\cdot}3 + 5{\cdot}3 + 2{\cdot}2 = 6 + 9 + 10 + 33 + 15 + 4 = 77$

Weighted external path length of T_2 can be given as,

$P_2 = 5{\cdot}2 + 7{\cdot}2 + 3{\cdot}3 + 4{\cdot}3 + 2{\cdot}2 = 10 + 14 + 9 + 12 + 4 = 49$

Weighted external path length of T_3 can be given as,

$P_3 = 2{\cdot}3 + 3{\cdot}3 + 5{\cdot}2 + 11{\cdot}1 = 6 + 9 + 10 + 11 = 36$

Technique

Given n nodes and their weights, the Huffman algorithm is used to find a tree with a minimum-weighted path length. The process essentially begins by creating a new node whose children are the two nodes with the smallest weight, such that the new node's weight is equal to the sum of the children's weight. That is, the two nodes are merged into one node. This process is repeated

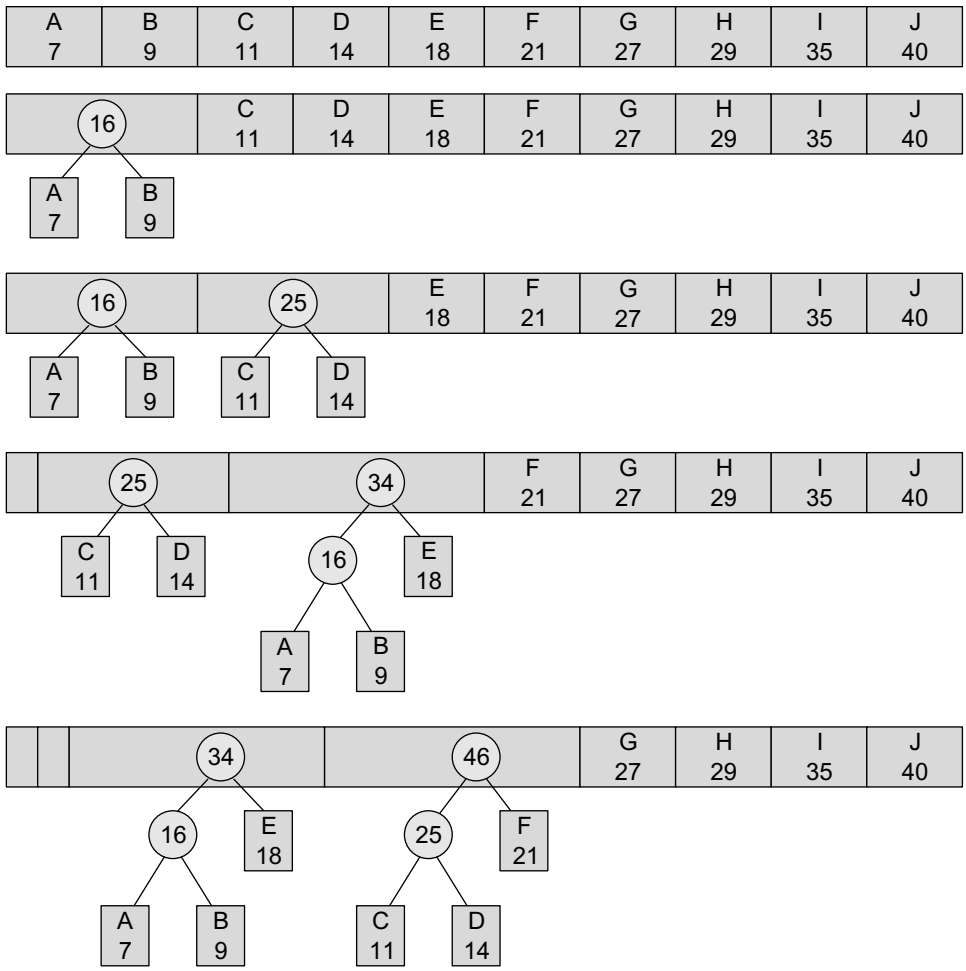
until the tree has only one node. Such a tree with only one node is known as the Huffman tree.

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



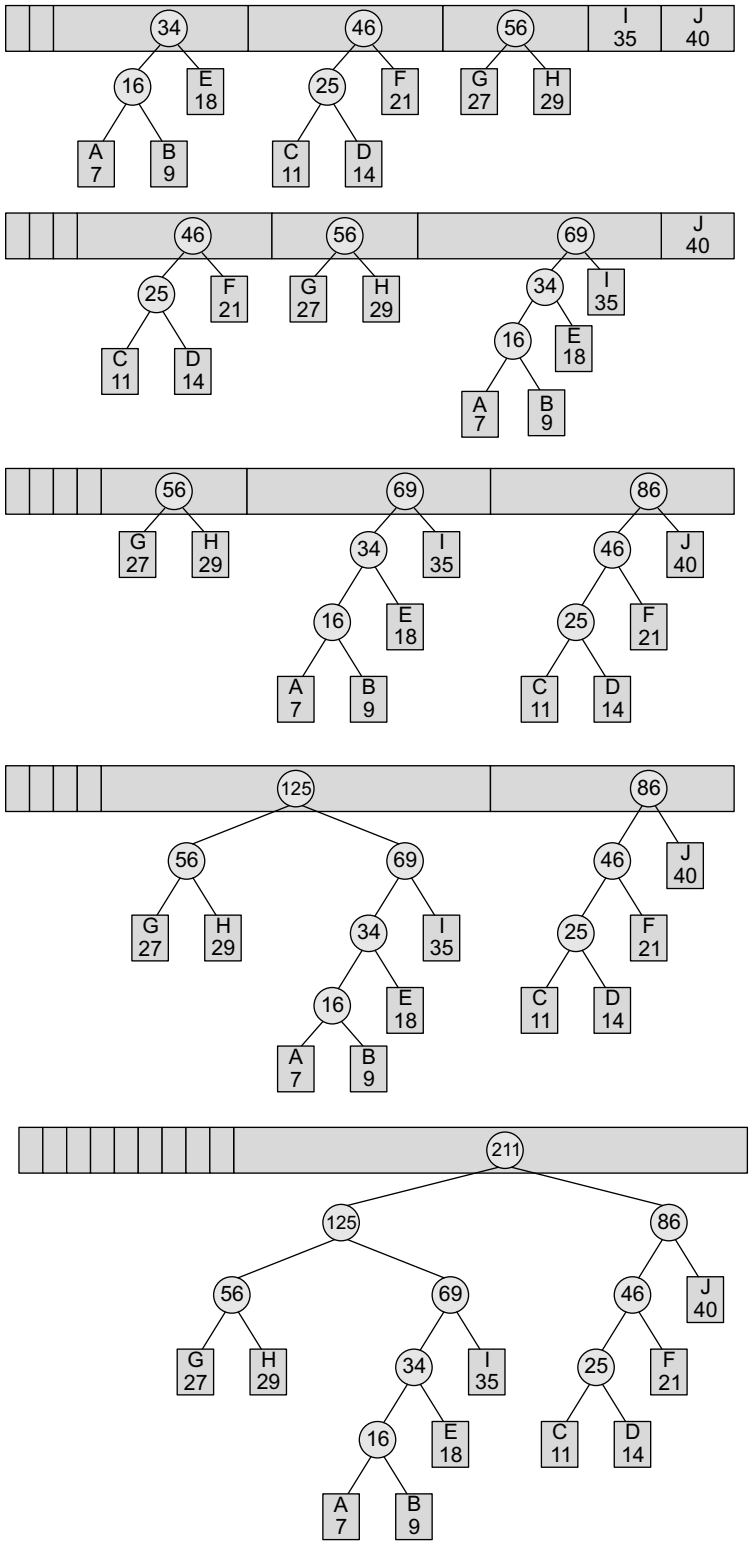

34
46
56
I
35
J
40
16
E
18
25
F
21
G
27
H
29
A
7
B
9
C
11
D
14
46
56
69
J
40
25
F
21
G
27
H
29
34
I
35
16
E
18
C
11
D
14
A
7
B
9
56
69
86
G
27
H
29
34
I
35
46
J
40
16
E
18
25
F
21
A
7
B
9
C
11
D
14
125
86
56
69
46
J
40
G
27
H
29
34
I
35
25
F
21
16
E
18
C
11
D
14
A
7
B
9
211
125
86
56
69
46
J
40
G
27
H
29
34
I
35
25
F
21
16
E
18
C
11
D
14
A
7
B
9

Table 9.1 Range of characters that can be coded using r = 2

Code	Character
00	A
01	B
10	C
11	D

Table 9.2 Range of characters that can be coded using r = 3

Code	Character
000	A
001	B
010	C
011	D
100	E
101	F
110	G
111	H

Data Coding

When we want to code our data (character) using bits, then we use `r` bits to code 2^r characters. For example, if `r=1`, then two characters can be coded. If these two characters are `A` and `B`, then A can be coded as 0 and `B` can be coded as 1 and vice versa. Look at Tables 9.1 and 9.2 which show the range of characters that can be coded by using `r=2` and `r=3`.

Now, if we have to code the data string `ABBBBBBAAAACDEFGGGGH`, then the corresponding code would be:

```
00000100100100100100100000000000000100111001011101101101101111
```

This coding scheme has a fixed-length code because every character is being coded using the same number of bits. Although this technique of coding is simple, coding the data can be made more efficient by using a variable-length code.

You might have observed that when we write a text in English, all the characters are not used frequently. For example, characters like `a`, `e`, `i`, and `r` are used more frequently than `w`, `x`, `y`, `z` and so on. So, the basic idea is to assign a shorter code to the frequently occurring characters and a longer to less frequently occurring characters. Variable-length coding is preferred over fixed-length coding because it requires lesser number of bits to encode the same data.

For variable-length encoding, we first build a Huffman tree. First, arrange all the characters in a priority queue in which the character with the lowest frequency of occurrence has the highest priority. Then, create a Huffman tree as explained in the previous section. Figure 9.24 shows a Huffman tree that is used for encoding the data set.

In the Huffman tree, circles contain the cumulative weights of their child nodes. Every left branch is coded with 0 and every right branch is coded with 1. So, the characters `A`, `E`, `R`, `W`, `X`, `Y`, and `Z` are coded as shown in Table 9.3.

Table 9.3 Characters with their codes

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

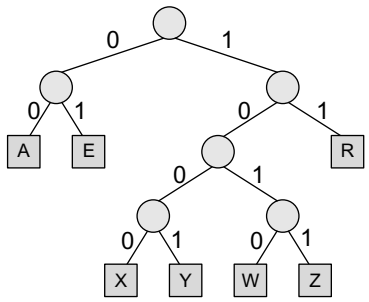


Figure 9.24 Huffman tree

Thus, we see that frequent characters have a shorter code and infrequent characters have a longer code.

9.6 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.

Points to Remember

- A tree is a data structure which is mainly used to store hierarchical data. A tree is recursively defined as collection of one or more nodes where one node is designated as the root of the tree and the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root.
- In a binary tree, every node has zero, one, or at the most two successors. A node that has no successors is called a leaf node or a terminal node. Every node other than the root node has a parent.
- The degree of a node is equal to the number of children that a node has. The degree of a leaf node is zero. All nodes that are at the same level and share the same parent are called siblings.
- Two binary trees having a similar structure are said to be copies if they have the same content at the corresponding nodes.
- A binary tree of `n` nodes has exactly `n – 1` edges. 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.
- A binary tree of height `h` has at least h nodes and at most $2^h - 1$ nodes.
- The height of a binary tree with `n` nodes is at least $\log_2(n+1)$ and at most `n`. In-degree of a node is the number of edges arriving at that node. The root node is the only node that has an in-degree equal to zero. Similarly, out-degree of a node is the number of edges leaving that node.
- In a complete binary tree, every level (except possibly the last) is completely filled and nodes appear as far left as possibly.
- A binary tree `T` is said to be an extended binary tree (or a 2-tree) if each node in the tree has either no children or exactly two children.
- Pre-order traversal is also called as depth-first traversal. It is also known as the NLR traversal algorithm (Node-Left-Right) and is used to extract a prefix notation from an expression tree. In-order algorithm is known as the LNR traversal algorithm (Left-Node-Right). Similarly, post-order algorithm is known as the LRN traversal algorithm (Left-Right-Node).
- 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.

Exercises

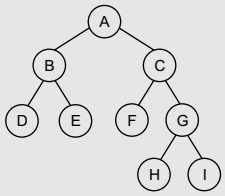
Review Questions

1. Explain the concept of a tree. Discuss its applications.
2. What are the two ways of representing binary trees in the memory? Which one do you prefer and why?
3. List all possible non-similar binary trees having four nodes.
4. Draw the binary expression tree that represents the following postfix expression:

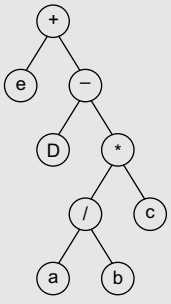
   A B + C * D –

5. Write short notes on:
(a) Complete binary trees
(b) Extended binary trees
(c) Tournament trees
(d) Expression trees
(e) Huffman trees
(f) General trees
(g) Forests

6. Consider the tree given below. Now, do the following:
(a) Name the leaf nodes
(b) Name the non-leaf nodes
(c) Name the ancestors of E
(d) Name the descendants of A
(e) Name the siblings of C
(f) Find the height of the tree
(g) Find the height of sub-tree rooted at E
(h) Find the level of node E
(i) Find the in-order, pre-order, post-order, and level-order traversal

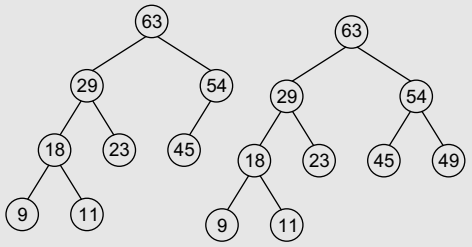


7. For the expression tree given below, do the following:
(a) Extract the infix expression it represents
(b) Find the corresponding prefix and postfix expressions
(c) Evaluate the infix expression, given a = 30, b = 10, c = 2, d = 30, e = 10



8. Convert the prefix expression `-/ab*+bcd` into infix expression and then draw the corresponding expression tree.

9. Consider the trees given below and state whether they are complete binary tree or full binary tree.



10. What is the maximum number of levels that a binary search tree with 100 nodes can have?

11. What is the maximum height of a tree with 32 nodes?

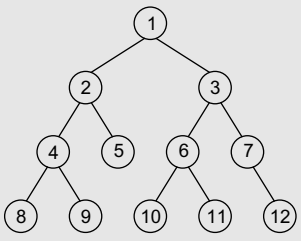
12. What is the maximum number of nodes that can be found in a binary tree at levels 3, 4, and 12?

13. Draw all possible non-similar binary trees having three nodes.

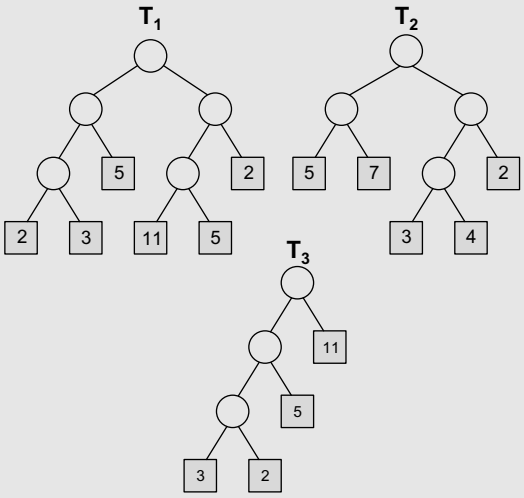
14. Draw the binary tree having the following memory representation:

		LEFT	DATA	RIGHT
ROOT 3 →	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	15			
	16	–1	11	–1
AVAIL 15	17			
	18	–1	12	–1
	19			
	20	2	6	16

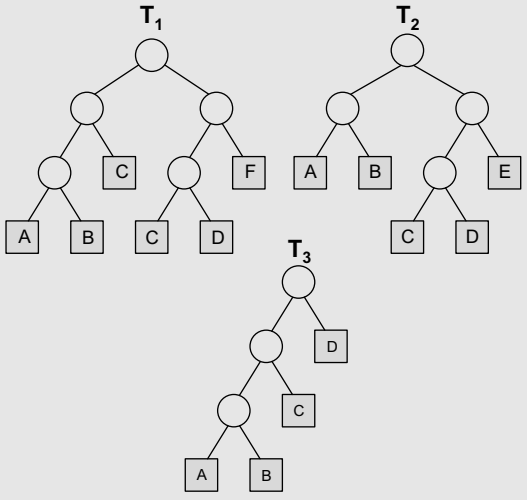
15. Draw the memory representation of the binary tree given below.



16. Consider the trees T_1, T_2, and T_3 given below and calculate their weighted path lengths.



17. Consider the trees T_1, T_2, and T_3 given below and find the Huffman coding for the characters.



Multiple-choice Questions

1. Degree of a leaf node is _____.
(a) 0 (b) 1
(c) 2 (d) 3

2. The depth of root node is _____.
(a) 0 (b) 1
(c) 2 (d) 3

3. A binary tree of height h has at least h nodes and at most _____ nodes.
(a) 2h (b) 2^h
(c) 2^{h+1} (d) $2^h - 1$

4. Pre-order traversal is also called _____.
(a) Depth first (b) Breadth first
(c) Level order (d) In-order

5. The Huffman algorithm can be implemented using a _____.
(a) Dequeue (b) Queue
(c) Priority queue (d) None of these

6. Total number of nodes at the nth level of a binary tree can be given as
(a) 2^n (b) 2^n
(c) 2^{n+1} (d) 2^{n-1}

True or False

1. Nodes that branch into child nodes are called parent nodes.

2. The size of a tree is equal to the total number of nodes.

3. A leaf node does not branch out further.

4. A node that has no successors is called the root node.

5. A binary tree of n nodes has exactly n – 1 edges.

6. Every node has a parent.

7. The Huffman coding algorithm uses a variable-length code table.

8. The internal path length of a binary tree is defined as the sum of all path lengths summed over each path from the root to an external node.

Fill in the Blanks

1. Parent node is also known as the _____ node.

2. Size of a tree is basically the number of _____ in the tree.

3. The maximum number of nodes at the k^{th} level of a binary tree is _____.

4. In a binary tree, every node can have a maximum of _____ successors.

5. Nodes at the same level that share the same parent are called _____.

6. Two binary trees are said to be copies if they have similar _____ and _____.

7. The height of a binary tree with n nodes is at least _____ and at most _____.

8. A binary tree T is said to be an extended binary tree if _____.

9. _____ traversal algorithm is used to extract a prefix notation from an expression tree.

10. In a Huffman tree, the code of a character depends on _____.