CMR
INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test III – May 2023

Sub:	Data Structures						Sub Code:		22MCA13
Date :	24/05/2023	Duration:	90 min's	Max Marks:	50	Sem :	I	Branch :	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Part.

	PART I	MARKS	OBE CO	OBE RBT
1	Define tree? With example explain the following i) Degree of a node, ii)Level of a binary tree iii)Siblings iv)Ancestors v) Root	[10]	CO4	L2
	OR			
2	Define Hashing. Explain the different hashing functions with example.	[10]	CO4	L3
	PART II			
3	Define a binary tree. With example show array and linked representation of binary tree.	[5+5]	CO4	L2
	OR			
4	Mention different types of binary trees and explain them briefly. State the properties of a binary tree in detail.	[5+5]	CO4	L3
	PART III			
5	Write the C-routines to traverse the given tree using i)Inorder ii) Pre-order iii) Post-order 	[10]	CO4	L3
	OR			
6	What is a graph? Write the terminologies used in graph. Explain matrix and adjacency list representation of graphs with example.	[5+5]	CO5	L2
	PART IV			
7	i)Construct a binary search tree for inputs i) 22, 14, 18, 50, 9, 15, 7, 6, 12, 32, 25 ii)Construct a binary tree where preorder and Inorder of a traversal yields the following sequence of nodes. Inorder: 8,4,10,9,11,2,5,1,6,3,7 Preorder:1,2,4,8,9,10,11,5,3,6,7	[4+6]	CO4	L4
	OR			
8	What is threaded binary tree? Write the rules to construct the threads and explain with example.	[10]	CO5	L3
	PART V			
9	Sort the numbers given below using radix sort and insertion sort 345, 654, 924, 123, 567, 472, 555, 808, 911 with appropriate figure.	[5+5]	CO5	L3
	OR			
10	What is collision? Explain the different collision resolution techniques, if K=3,2,9,6,11,13,7,12 and h(k)=2k+3 where M=10. Use division method and Linear probing to store these values in a hash table.	[10]	CO5	L4

Solution

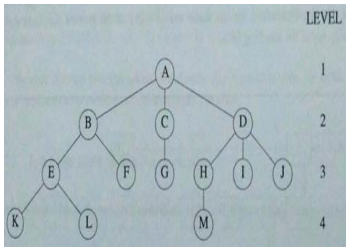
1. Define tree? With example explain the following i) Degree of a node ii)Level of a binary tree iii)Siblings iv)Ancestors v) Root

   The tree is a nonlinear data structure. This structure is mainly used to represent data containing a hierarchical relationship between elements, e.g., records, family trees and tables of contents. A tree structure means that the data are organized in a hierarchical manner.

   Definition: A tree is a finite set of one or more nodes such that

   i) There is a specially designated node called the **root**.

   ii) The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, T_2, \ldots, T_n$, where each ofthese sets is a tree. $T_1, T_2, \ldots, T_n$, are called the **subtrees** of the root.



There are many terms that are often used when referring to trees.

i) A **node** stands for the item of information plus the branches to other nodes.
Consider the tree in Fig.4.2. This tree has 13 nodes, each item of data being a single letter. The **root** is A, and we will normally draw trees with the root at the top.

ii) The number of subtrees of a node is called its **degree**.
The degree of A is 3, of C is I, and of F is zero.

iii) Nodes that have degree zero are called **leaf** or **terminal nodes**.
{K,L,F,G,M,I,J} is the set of leaf nodes.

iv) Consequently, the other nodes are referred to as **nonterminals**.

v) The roots of the subtrees of a node X are the **children** of X. X is the **parent** of its children.
Thus, the children of D are H, I, and J; the parent of D is A.

Every node N in a tree, except the root, has a unique parent, called the **predecessor** of N.

vi) Children of the same parent are said to be **siblings**.
H, I, and J are siblings.

vii) We can extend this terminology if we need to so that we can ask for the **grandparent** of M, which is D, and so on.

viii) A node L is called a **descendant** of a node N if there is a succession of children from N to L. N is called an **ancestor** of L.

ix) The **degree of a tree** is the maximum of the degree of the nodes in the tree.
The tree of Fig.4.2 has degree 3. The ancestors of a node are all the nodes along the path from the root to that node. The ancestors of M are A, D, and H.

x) The **level of a node** is defined by letting the root be at level one. If a node is at level l, then its children are at level l+1.
Fig. 4.2 shows the level of all nodes in that tree.

xi) The **height** or **depth** of a tree is defined to be the maximum level of any node in the tree. Thus, the depth of the tree in Fig. 4.2 is 4.

2. Define Hashing. Explain the different hashing functions with example

Hashing

Linear search has a running time proportional to O(n), while binary search takes time proportional to $O(\log_2 n)$, where n is the number of elements in the array. Binary search and binary search trees are efficient algorithms to search for an element. Using hashing, we can perform the search operation in time proportional to O(1) i.e. in constant time, irrespective of its size.

Let us take an example of a small company of 100 employees, each employee is assigned an Emp_ID in the range 0–99. To store the records in an array, each employee's Emp_ID acts as an index into the array where the employee's record will be stored as shown in Fig. 5.1. In this case, we can directly access the record of any employee, once we know his Emp_ID, because the array index is the same as the Emp_ID number.

Different Hash Functions

Many hash functions are being used in practice, namely:

i) Division
ii) Mid-Square
iii) Folding
iv) Folding with Reversing
v) Correspondence between letter and address
vi) Additive transformation

1) Division:

The key k is divided by some number D and the remainder is used as the home bucket for k. Formally,

$$h(k) = k \% D$$

- This hash function assumes that the keys are non-negative integers.
- This function gives bucket addresses in the range 0 through D-1, so the hash table must have at least b = D buckets.
- For practical dictionaries, a very uniform distribution of keys to buckets is obtained when D has no prime factor smaller than 20.
- Choose a number D larger than the number n of keys in K. The number D is usually chosen to be a prime number or a number without s mall divisors, since this frequently minimizes the number of collisions.
- If we want the hash address to range from 1 to m, use the formula: H(k) = k % m + 1.

Example: A compa ny has 68 employees. Each e mployee is assigned a unique 4 -digit employee-id. Suppose L consists of 100 two-digit addresses: 00, 01, 02, …, 99. Apply the division hash function to the employee-ids: 3205, 7148, 2345 and obtain the hash address.
Solution: Choose a prime number m close to 99 i.e. m = 97. Then:

H(3205) = 3205 % 97 = 4
H(7148) = 7148 % 97 = 67
H(2345) = 2345 % 97 = 17

2) Mid Square:

The key k is squared. Then the hash function H is defined by

$$H(k) = l$$

where l is obtained by deleting digits from both ends of k^2. The same positions of k^2 must be used for all of the keys.

Example: A compa ny has 68 employees. Each e mployee is assigned a unique 4-digit e mployee-id. Suppose L consists of 100 two-digit addresses: 00, 01, 02, …, 99. Apply the midsquare hash function to the employee-ids: 3205, 7148, 2345 and obtain the hash address.
Solution: The following calculations are performed:

k	3205	7148	2345
k^2	10272025	51093904	5499025
H(k)	72	93	99

The fourth and fifth digits, counting from the right are chosen for the hash address.

3) **Folding:** The key k is partitioned into a number of parts, k_1, k_2, … k_r where each part, except possibly the last, has the same number of digits as the required address. Then the se parts are added together, ignoring the last carry. That is,

$$H(k) = k_1 + k_2 + \ldots + k_r$$

where the leading-digit carries, if any, ignored.

Sometimes, for extra "milling", the even numbered parts, k_2, k_4, …, are reversed before the addition.

Example: A compa ny has 68 employees. Each e mployee is assigned a unique 4-digit e mployee-id. Suppose L consists of 100 two-digit addresses: 00, 01, 02, …, 99. Apply the folding hash function to the employee-ids: 3205, 7148, 2345 and obtain the hash address.

Solution: Chopping the key k into two parts and adding yields the following hash addresses:H(3205): 32 + 05 = 37,
H(7148): 71 + 48 = 19,
H(2345): 23 + 45 = 68

Observe that the leading digit 1 in H(7148) is ignored.

4) **Folding with Reversing:** The method is the sa me as the folding method, except that the even parts (i.e. the 2nd part, 4th part, 6th part, etc. are reversed before adding, thus producing the following hash addresses:

H(3205) = 32 + 50 = 82,
H(7148) = 71 + 84 = 55,
H(2345) = 23 + 54 = 77

3. Define a binary tree. With example show array and linked representation of binary tree

BINARY TREES

A binary tree is a special kind of tree which can be easily maintained in the computer. Although such a tree may seem to be very restrictive, more general trees may be viewed as binary trees.

A **binary tree** T is defined as a finite set of elements, called **nodes**, such that:

i) T is empty (called the **null tree** or **empty tree**), or
ii) T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 and T_2.

a) If T does contain a root R, then the two trees T_1 and T_2 are called, respectively, the **left and right subtrees** of R.
b) If T_1 is nonempty, then its root is called the **left successor** of R; similarly, if T_2 is nonempty, then its root is called the **right successor** of R.

Binary Tree Representations: Array and linked Representation of Binary Trees

Array Representation

The array or sequential representation of a tree uses a single one-dimensional array, TREE as follows:

1) The root R of T is stored in TREE[1].
2) If a node N occupies TREE[K], then its left child is stored in TREE[2*K] and its right child in TREE[2* K+1].

Location 0 is not used.

NULL is used to indicate an empty subtree. TREE[1] = NULL indicates the tree is empty.

In other words, the nodes are numbered from 1 to n as per a full binary tree. Location 0 is not used. The nodes are then stored in a one-dimensional array whose position 0 is left e mpty, and the node numbered x is mapped to position with index x in the array. We can easily determine the locations of the parent, left child and right child of any node, i, in the binary tree using the following rule:

If a complete binary tree with n nodes is represented sequentially,
then for any node with index i, $1 \le i \le n$,

1) parent(i) is at $\lfloor i/2 \rfloor$ if $i \neq 1$. If $i = 1$, i is at the root and has no parent.
2) leftChild(i) is at 2i if $2i \le n$. If $2i > n$, then I has no left child.
3) rightChild(i) is at 2i+1 if $2i+1 \le n$. If $2i+1 > n$, then I has no right child.

The sequential representation of the binary tree T in Fig. 4.8(a) appears in Fig. 4.8(b).

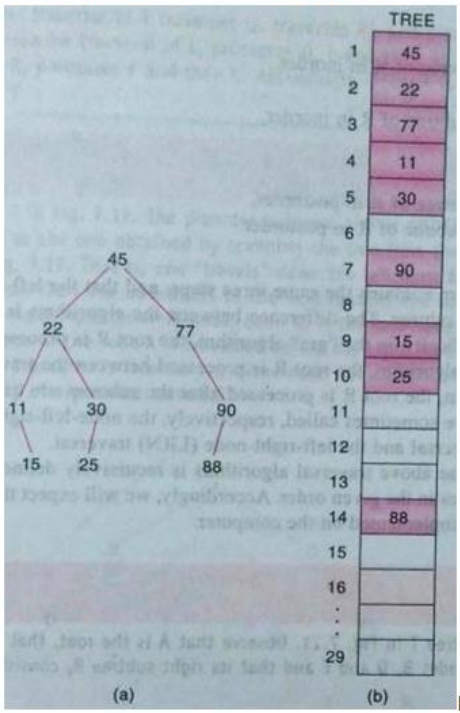


Fig. 4.8 Array Representation of Binary Tree

From the Fig.4.8 we see that even though T has only 9 nodes, 14 locations are required to represent the tree in the array.

Linked Representation

The array or sequential representation of a binary tree is an efficient way of maintaining the tree in memory; especially complete or nearly complete binary trees. It wastes space for many other binary trees. In addition, the representation suffers from the general inadequacies of sequential representations. Insertion and deletion of nodes from the middle of a tree require the move me nt of potentially many nodes to reflect the change in level number of these nodes. These problems can be overcome easily through the use of linked representation.

In the linked representation of a tree, each node has three fields: leftChild, data and rightChild. In C this can be defined as:

```
struct treenode
{
        int data;
        struct treenode *leftChild, *rightChild;
};
typedef struct treenode *treePtr;
```

Fig. 4.11 Tree Node Representation

The pointer root will contain the location of the root R of T. If any subtree is empty, then the corresponding pointer will contain null value; if the tree T itself is empty, root will contain null value.

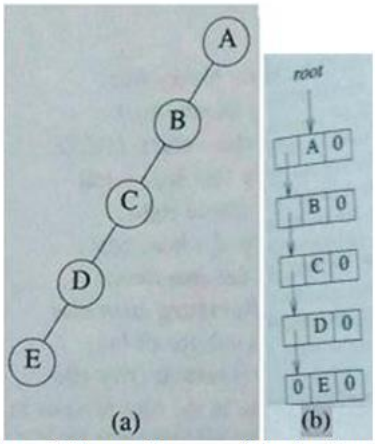
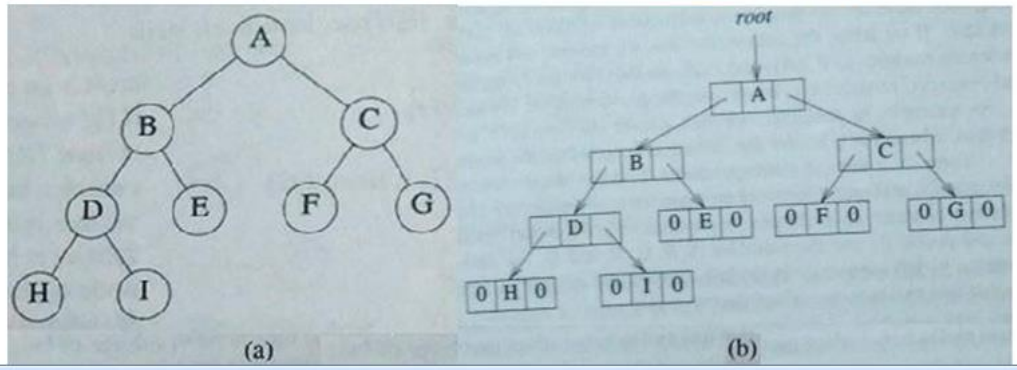


Fig. 4.12 Link Representation of Binary Tree (a) The Tree (b) Its link representation



4. Mention different types of binary trees and explain them briefly. State the properties of a binary tree in detail.

Types of Binary Trees

There are various **types of binary trees**, and each of these **binary tree types** has unique characteristics. Here are each of the **binary tree types** in detail:

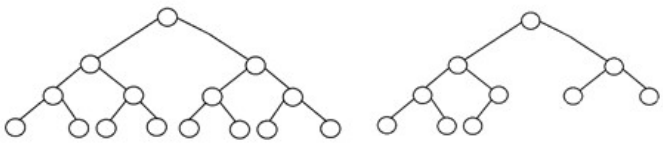
1. Full Binary Tree

It is a special kind of a binary tree that has either zero children or two children. It means that all the nodes in that binary tree should either have two child nodes of its parent node or the parent node is itself the leaf node or the external node.

In other words, a full binary tree is a unique binary tree where every node except the external node has two children. When it holds a single child, such a binary tree will not be a full binary tree. *Here, the quantity of leaf nodes is equal to the number of internal nodes plus one. The equation is like* L=I+1, where L is the number of leaf nodes, and I is the number of internal nodes.

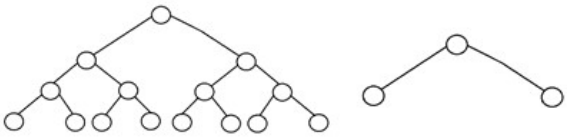
2. Complete Binary Tree

A complete binary tree is another specific type of binary tree where all the tree levels are filled entirely with nodes, except the lowest level of the tree. Also, in the last or the lowest level of this binary tree, every node should possibly reside on the left side. Here is the structure of a complete binary tree:



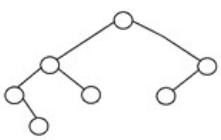
3. Perfect Binary Tree

A binary tree is said to be 'perfect' if all the internal nodes have strictly two children, and every external or leaf node is at the same level or same depth within a tree. A perfect binary tree having height 'h' has 2h – 1 node. Here is the structure of a perfect binary tree:



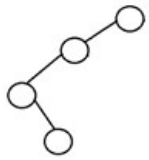
4. Balanced Binary Tree

A binary tree is said to be 'balanced' if the tree height is O(logN), where 'N' is the number of nodes. In a balanced binary tree, the height of the left and the right subtrees of each node should vary by at most one. An AVL Tree and a Red-Black Tree are some common examples of data structure that can generate a balanced binary search tree. Here is an example of a balanced binary tree:



5. Degenerate Binary Tree

A binary tree is said to be a degenerate binary tree or pathological binary tree if every internal node has only a single child. Such trees are similar to a linked list performance-wise. Here is an example of a degenerate binary tree:



Properties of Binary trees

1) The maximum number of nodes on level i of a binary tree is 2^{i-1}, $i \geq 1$.

Proof: The proof is by induction on i.

Induction Base: The root is the only node on level $i=1$.
Hence, the maximum number of nodes on level $i = 1$ is $2^{i-1} = 2^0 = 1$.

Induction Hypothesis: Let i be a positive integer greater than 1. Assume that the maximum number of nodes on level i-1 is 2^{i-2}.

Induction Step: The maximum number of nodes on level i-1 is 2^{i-2} by the induction hypothesis.
Since each node in a binary tree has a maximum degree of 2, the maximum number of nodes on level i = Two times the maximum number of nodes on level i
$= 2.\ 2^{i-2} = 2^{i-1}$.

2) The maximum number of nodes in a binary tree of depth k is $2^k - 1$, $k \geq 1$.

The maximum number of nodes in a binary tree of depth k

$$= \sum_{i=1}^{k} (\text{maximum number of nodes on level } i)$$

$$= \sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

3) **Relation between the number of leaf nodes and degree-2 nodes:** For any nonempty binary tree, T, if n_0 is the number of leaf nodes and n_2 is the number of nodes of degree 2, then
$n_0 = n_2 + 1$.

Proof: Let n_1 be the number of nodes of degree one and n the total number of nodes.
Since all nodes in T are at most of degree two,
$n = n_0 + n_1 + n_2$ ... (4.1)
If we count the number of branches in a binary tree, every node except the root has a branch leading to it.
If B is the number of branches, then $n = B + 1$.
All branches stem from a node of degree one or two. Thus, $B = n_1 + 2n_2$.
Therefore, $n = B+1 = n_1 + 2n_2 + 1$ ... (4.2)

Subtracting (4.2) from (4.1) and rearranging the terms, we get

$n_0 = n_2 + 1$.

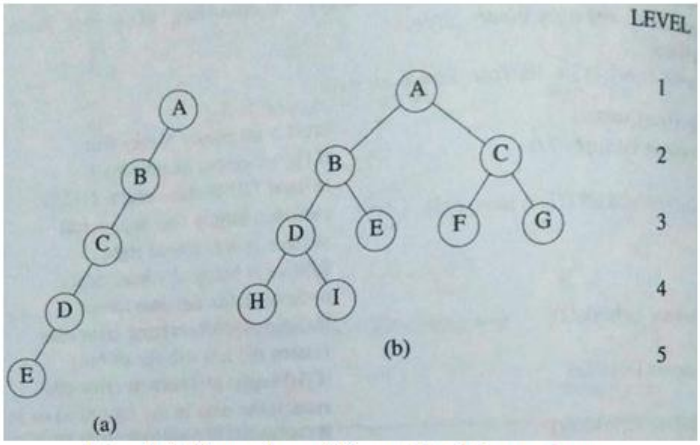


Fig. 4.6 Skewed and Complete binary trees

Example: In Fig. 4.6(a), $n_0 = 1$ and $n_2 = 0$.
In Fig. 4.6(b), $n_0 = 5$ and $n_2 = 4$.

Full binary tree: A full binary tree of depth k is a binary tree of depth k having $2^k - 1$ nodes, $k \geq 0$.

The maximum number of nodes in a binary tree of depth k = $2^k - 1$.

Fig. 4.7 shows a full binary tree of depth 4.

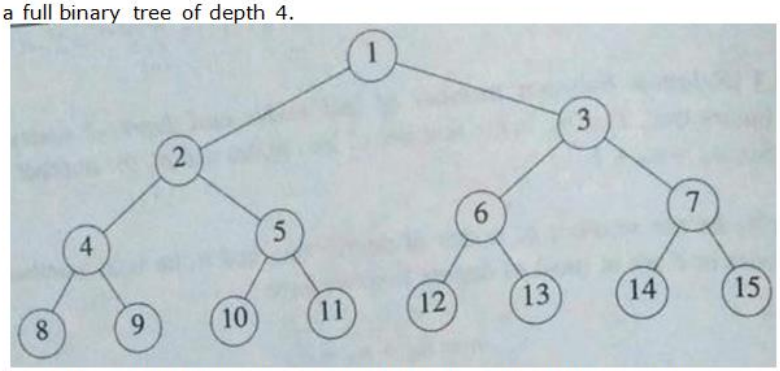


Fig. 4.7 Full binary tree of depth 4 with sequential node numbers

We number the nodes in a full binary tree starting with the root on level 1, continuing with the nodes on level 2, and so on. Nodes on any level are numbered from left to right.

Complete binary tree: A binary tree with n nodes and depth k is complete if and only if its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k.

The height of a complete binary tree with n nodes is $\lceil \log_2(n+1) \rceil$.

$n = 2^k - 1$ i.e. $2^k = n+1$ i.e. $k = \lceil \log_2(n+1) \rceil$

($\lceil x \rceil$ is the smallest integer $\geq x$.)

5. Write the C-routines to traverse the given tree using i)Inorder ii) Pre-order iii) Post-order

C function for inorder traversal of a binary tree:

```
void inorder(treePtr root)
{       /* inorder tree traversal */
        if (root)
        {
                inorder(root->leftChild);
                printf("%d", root->data);
                inorder(root->rightChild);
        }
}
```

C function for preorder traversal of a binary tree:

```
void preorder(treePtr root)
{       /* preorder tree traversal */
        if (root)
        {
                printf("%d", root->data);
                preorder(root->leftChild);
                preorder(root->rightChild);
        }
}
```

C function for postorder traversal of a binary tree:

```
void postorder(treePtr root)
{       /* postorder tree traversal */
        if (root)
        {
                postorder(root->leftChild);
                printf("%d", root->data);
                postorder(root->rightChild);
        }
}
```

Preorder: ABDGECF
Inorder: GDBEAFC
Postorder: GDEPFCA

6. What is a graph? Write the terminologies used in graph.

GRAPHS

Graph is a non-linear data structure.

Terminology

A **graph** G consists of two things:

1) A set V of elements called **nodes** or **points** or **vertices**.
2) A set E of **edges** such that each edge e in E is identified with a unique unordered pair [u,v] of nodes in V, denoted by e = [u,v].

The parts of the graph are indicated by writing G = (V,E).

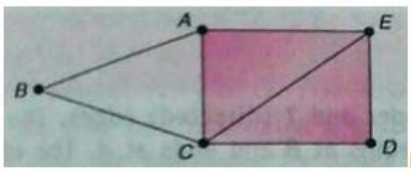


Fig. 5.1 A graph

Suppose e = [u,v].

- Nodes u and v are called the **endpoints** of e.
- u and v are said to be **adjacent nodes** or **neighbours**.
- The **degree of a node** u, written deg(u), is the number of edges containing u.
- If deg(u) = 0, i.e. if u does not belong to any edge, then u is called an **isolated node**.
- A **path** P of length n from a node u to a node v is defined as a sequence of n+1 nodes.
  $P = (v_0, v_1, v_2, \ldots, v_n)$
  such that $u=v_0$; v_{i-1} is adjacent to v_i for i = 1, 2, ..., n; and $v_n = v$.
- The path P is said to be **closed** if $v_0 = v_n$.
- The path P is said to be **simple** if all the nodes are distinct, with the exception that v_0 may equal v_n; i.e. P is simple if the nodes $v_0, v_1, \ldots, v_{n-1}$ are distinct and the nodes $v_1, v_2, \ldots, v_n$ are distinct.
- A **cycle** is a closed simple path with length 3 or more.
- A cycle of length k is called a **k-cycle**.

Example: Fig. 5.1 is a graph with 5 nodes – A, B, C, D and E, and 7 edges: [A,B], [B,C], [C,D], [D,E], [A,E], [C,E], [A,C].

There are two simple paths of length 2 from B to E: (B,A,E) and (B,C,E).

There is only one simple path from B to D of length 2: (B,C,D).

(B,A,D) is not a path since [A,D] is not an edge. There are two 4-cycles in the graph: [A,B,C,E,A] and [A,C,D,E,A].

Deg(A) = 3, since A belongs to 3 edges.

Deg(C) = 4 and deg(D) = 2.

- A **graph** G is said to be **connected** if there is a path between any two of its nodes.
- If there is a path P from a node u to a node v, then, by eliminating unnecessary edges, one can obtain a simple path Q from u to v.

Explain matrix and adjacency list representation of graphs with example.

REPRESENTATION OF GRAPHS.

There are two standard ways of maintaining a graph G in the memory of a computer.

1. Sequential Representation of G – by means of its adjacency matrix
2. Linked Representation of G – by means of linked lists of neighbour.

Regardless of the way one maintains a graph G in the memory of a computer, the graph G is normally input into the computer by its collection of nodes and its collection of edges.

SEQUENTIAL REPRESENTATION OF GRAPHS

Adjacency Matrix

Suppose G is a simple directed graph with m nodes, and suppose the nodes of G have been ordered and are called $v_1, v_2, \ldots, v_m$. Then the **adjacency matrix** $A = (a_{ij})$ of the graph is the m x m matrix defined as follows:

$$a_{ij} = \begin{cases} 1 \text{ if } v_i \text{ is adjacent to } v_j, \text{ that is, if there is an edge } (v_i, v_j) \\ 0 \text{ otherwise} \end{cases}$$

Such a matrix A, which contains entries of only 0 and 1, is called a bit matrix or a Boolean matrix.

The number of 1's in A is equal to the number of edges in G.

Example: Consider the graph in Fig. 5.6. Suppose the nodes are stored in memory in a linear array DATA as follows:

DATA: X, Y, Z, W

Assume that the ordering of the nodes in G is as follows: $v_1 = X$, $v_2 = Y$, $v_3 = Z$ and $v_4 = W$. Find the adjacency matrix of G.

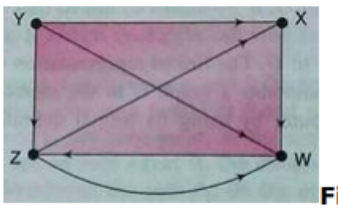


Fig. 5.6

The adjacency matrix of G is

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The adjacency matrix A of the graph depends on the ordering of the nodes of G; i.e. a different ordering of the nodes may result in a different adjacency matrix. However, the matrices resulting from two different orderings are closely related in that one can be obtained from the other by simply interchanging rows and columns.

Suppose G is an undirected graph, then the adjacency matrix A of G will be a **symmetric matrix** i.e. one in which $a_{ij} = a_{ji}$ for every i and j. This is because each undirected edge [u,v] corresponds to the two directed edges (u,v) and (v,u).

This matrix representation can be extended to multigraphs. If G is a multigraph, then the adjacency matrix of G is the m x m matrix $A = (a_{ij})$ defined by setting a_{ij} equal to the number of edges from v_i to v_j.

Consider the powers A, A^2, A^3, ... of the adjacency matrix A of a graph G. Let

$a_K(i,j)$ = the ij entry in the matrix A^K.

$a_1(i,j) = a_{ij}$ gives the number of paths of length 1 from node v_i to node v_j.

$a_2(i,j)$ gives the number of paths of length 2 from node v_i to node v_j.

LINKED REPRESENTATION OF A GRAPH

The major drawbacks of the sequential representation of a directed graph G with m nodes in memory are:

1) Sparse graphs with few edges for a given number of vertices result in many zero entries in adjacency matrix. This wastes space and makes many algorithms less efficient (eg. To find nodes adjacent to a given node, we have to iterate through the whole row even if there are a few 1s there.
2) It may be difficult to insert and delete nodes in G. This is because the size of A may need to be changed and the nodes may need to be reordered, so there may be many changes in the matrix A.
3) The adjacency matrix A of the graph depends on the ordering of the nodes of G; i.e. a different ordering of the nodes may result in a different adjacency matrix.

Accordingly, G is usually represented in memory by a **linked representation**, also called an **adjacency structure**.

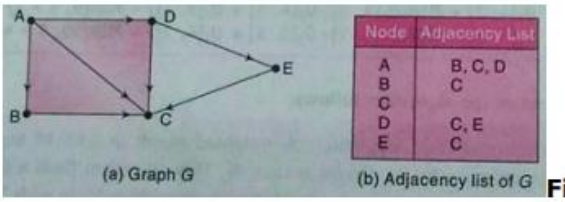


Fig. 5.8

Consider the graph G in Fig.5.8(a). The table in Fig.5.8(b) shows each node in G followed by its **adjacency list**, which is its list of adjacent nodes, also called its **successors** or **neighbors**.

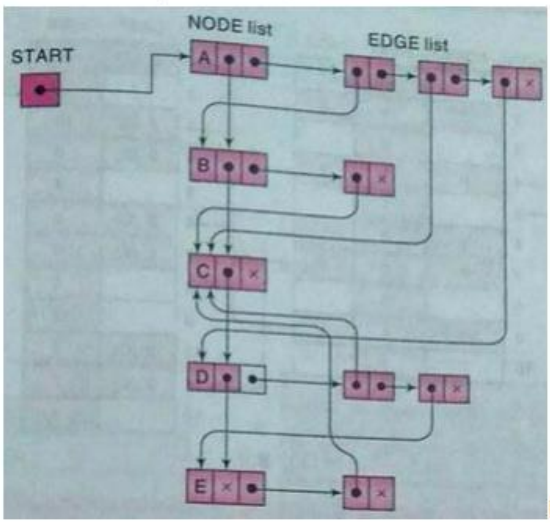
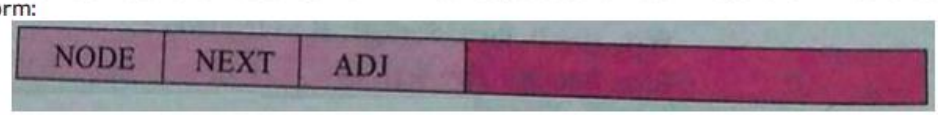


Fig. 5.9

Fig. 5.9 shows a schematic diagram of a linked representation of G in memory. Specifically, the linked representation will contain two lists (or files), a node list NODE and an edge list EDGE, follows :

1) **Node list**: Each element in the list NODE will correspond to a node in G, and it will be a record of the form:



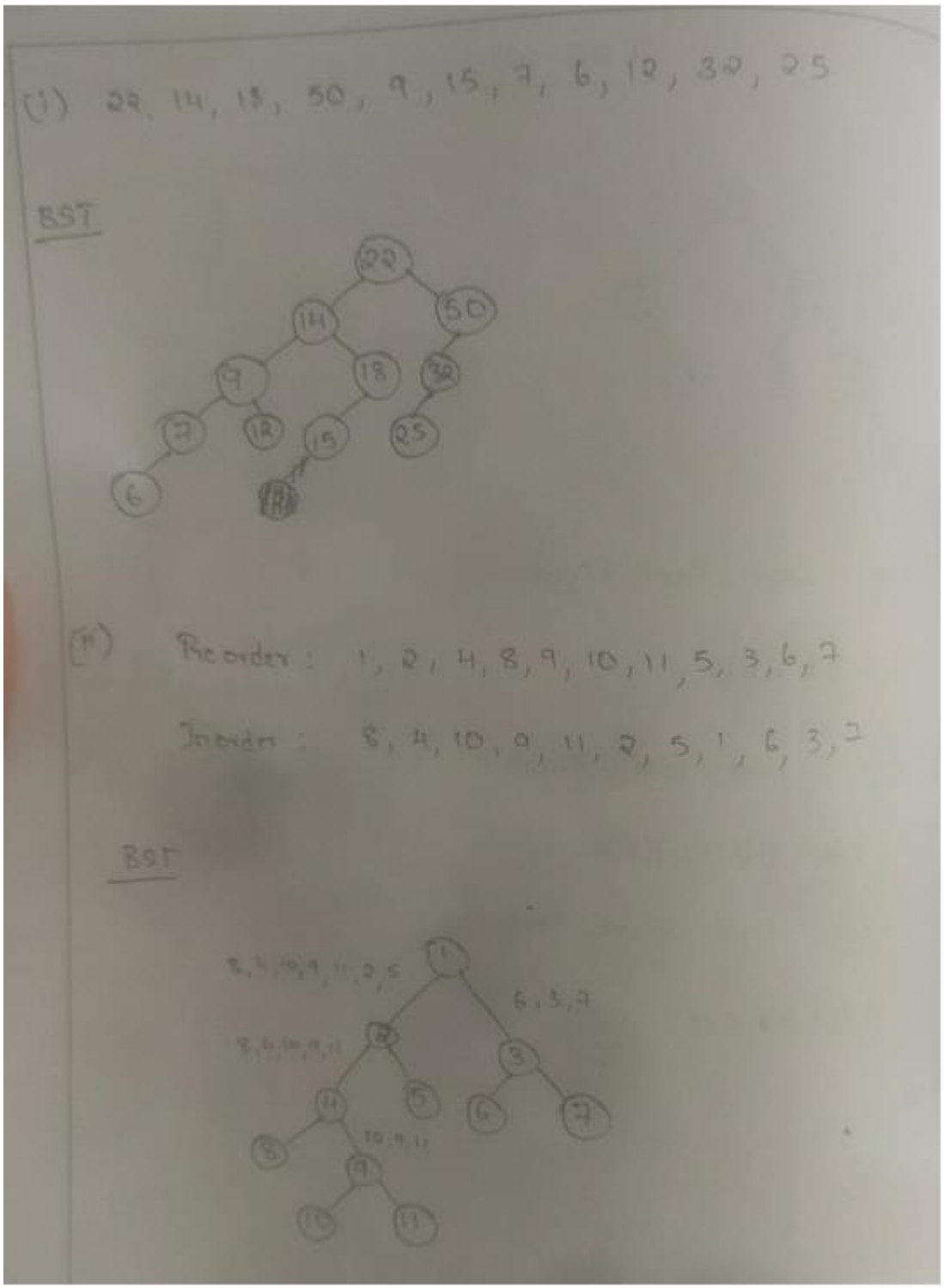
Here NODE will be the name or key value of the node, NEXT will be a pointer to the next node in the list NODE and ADJ will be a pointer to the first element in the adjacency list of the node, which is maintained in the list EDGE. The shaded area indicates that there may be other information in the record, such as:

i) the indegree INDEG of the node,
ii) the outdegree OUTDEG of the node,
iii) the STATUS of the node during the execution of an algorithm, and so on.

(Alternatively, one may assume that NODE is an array of records containing fields such as NAME, INDEG, OUTDEG, STATUS, ...)

The nodes themselves, as pictured in Fig.5.8, will be organized as a linked list and hence will have a pointer variable START for the beginning of the list and a pointer variable AVAILN for the list of available space.

7. i)Construct a binary search tree for inputs i) 22, 14, 18, 50, 9, 15, 7, 6, 12, 32, 25 ii)Construct a binary tree where preorder and Inorder of a traversal yields the following sequence of nodes. Inorder: 8,4,10,9,11,2,5,1,6,3,7 Preorder:1,2,4,8,9,10,11,5,3,6,7



8. What is threaded binary tree? Write the rules to construct the threads and explain with example.

Threaded binary trees

In a linked representation of any binary tree, there are more links than actual pointers i.e. there are n+1 null links out of 2n total links.

A.J.Perlis and C. Thornton devised threaded binary tree in which they replaced the null links by pointers called **threads** to other nodes in the tree.

To construct the threads, use the following rules:

i) If ptr->leftChild is null, replace ptr->leftChild with a pointer to the node that would be visited before ptr in an inorder traversal i.e. the null is replaced with a po inter to the *inorder predecessor* of ptr.

ii) If ptr->rightChild is null, replace ptr->rightChild with a pointer to the node that would be visited after ptr in an inorder traversal i.e. the null is replaced with a pointer to the *inorder successor* of ptr.

A **threaded binary tree** is a binary tree which contains threads i.e. addresses of some nodes which facilitate move ment in the tree.

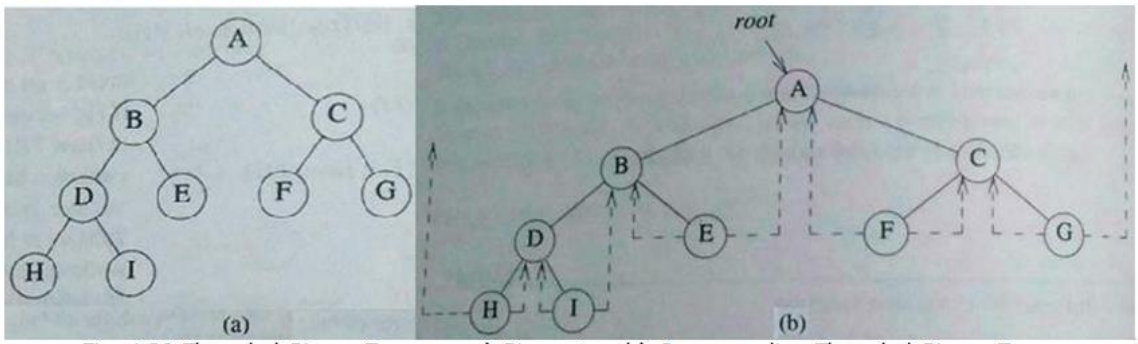


Fig. 4.20 Threaded Binary Tree a) Binary tree b) Corresponding Threaded Binary Tree

Fig. 4.20 shows a binary tree and the corresponding binary tree. The threads are shown as broken lines. This tree has 9 nodes and 10 threads. If we traverse the tree in inorder, the nodes will be visited in the order H, D, I, B, E, A, F, C, G. Example: node E has a predecessor thread that points to B and a successor thread that points to A.

When the tree is represented is memory, two additional fields, leftThread and rightThread are used to distinguish between threads and normal pointers. Assume that ptr is an arbitrary node in a threaded tree.

If ptr- >leftThread = TRUE, then ptr- >leftChild contains a thread; otherwise it contains a pointer to the left child.

Similarly, if ptr- >rightThread = TRUE, then ptr- >rightChild contains a thread; otherwise it contains a pointer to the right child.

This node structure is given by the following C declarations:

```
struct threadedTree
{
      short int leftThread;
      struct threadedTree *leftChild;
      int data;
      struct threadedTree *rightChild;
      short int rightThread;
 };
typedef struct threadedTree *threadedPTR;
```

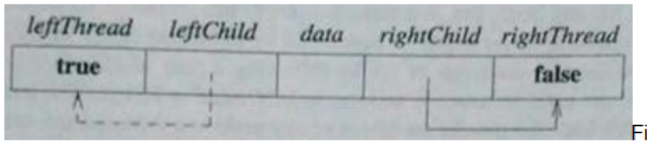


Fig. 4.21 An empty threaded binary tree

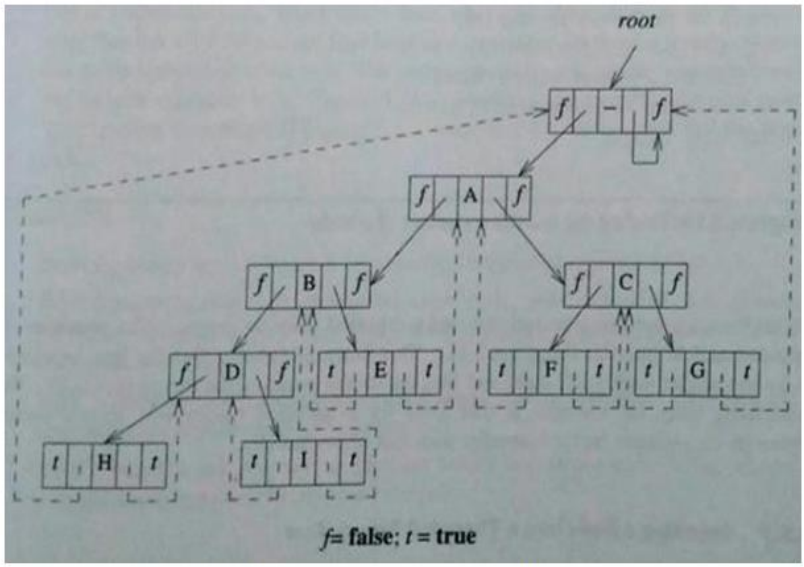


Fig. 4.22 Memory representation of threaded tree

Fig. 4. 22 gives the memory representation of a threaded tree. The variable root points to the header node of the tree, while root->leftChild points to the start of the first node of the actual tree. This is true for all threaded trees. The loose threads point to the head node, root.

9. Sort the numbers given below using radix sort and insertion sort 345, 654, 924, 123, 567, 472, 555, 808, 911 with appropriate figure.
   Solution: 123, 345, 472, 555, 567, 654, 808, 911, 924

10. What is collision? Explain the different collision resolution techniques, if K=3,2,9,6,11,13,7,12 and h(k)=2k+3 where M=10. Use division method and Linear probing to store these values in a hash table.
    Hashing in data structure falls into a collision if two keys are assigned the same index number in the hash table. The collision creates a problem because each index in a hash table is supposed to store only one value. Hashing in data structure uses several collision resolution techniques to manage the performance of a hash table.
    It is a process of finding an alternate location. The collision resolution techniques can be named as-
    Open Hashing (Separate Chaining)
    Closed Hashing (Open Addressing)
    Linear Probing
    Quadratic Probing
    Double Hashing