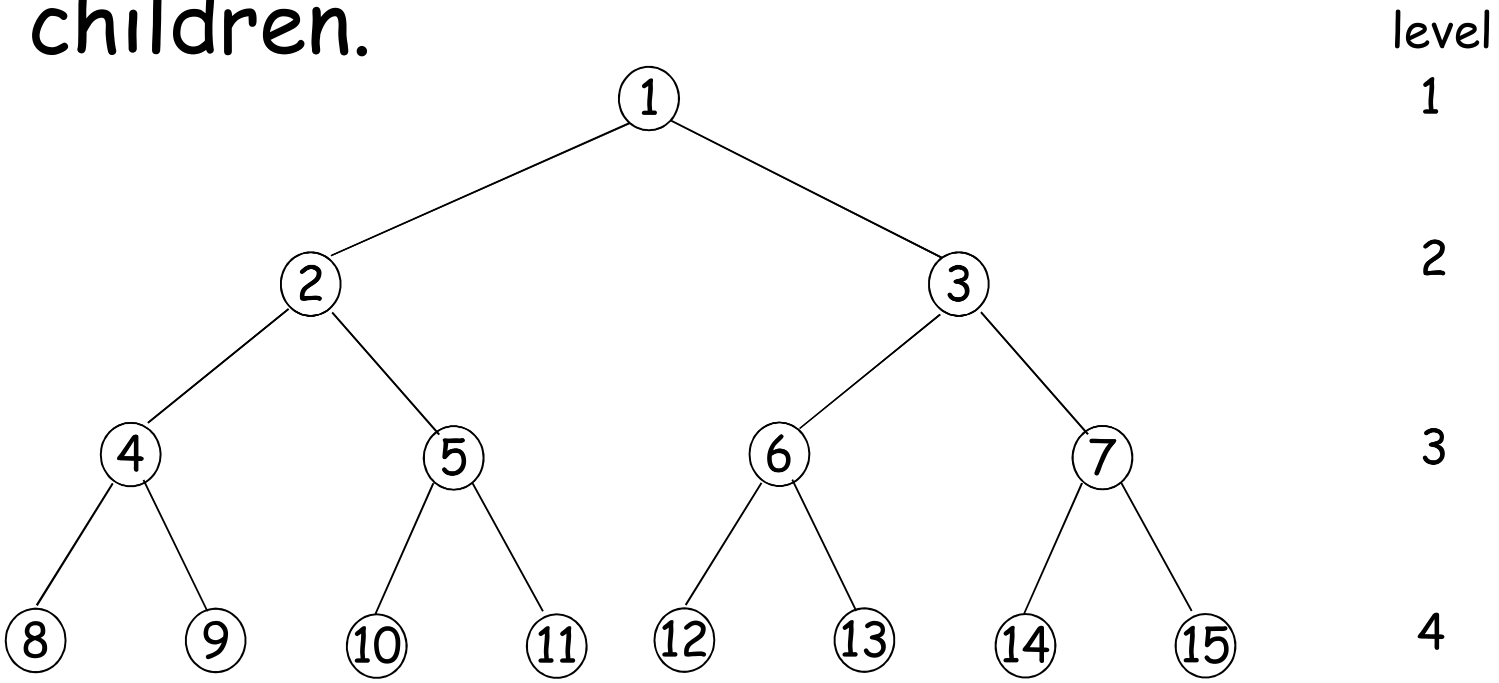
Full binary Tree

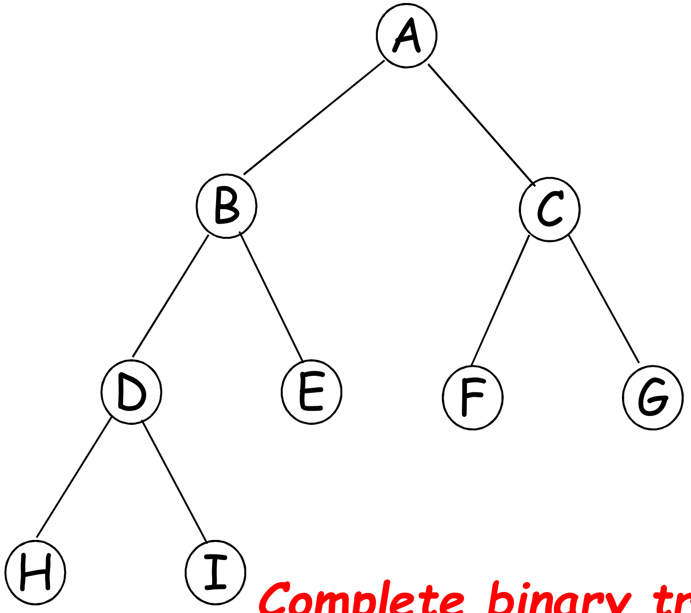
Definition: A **full binary tree** of depth k is a binary tree of depth k having $2^k - 1$ nodes, k ≥ 0 (i.e having the maximum number of nodes). In this every node other than the leaves has two children.



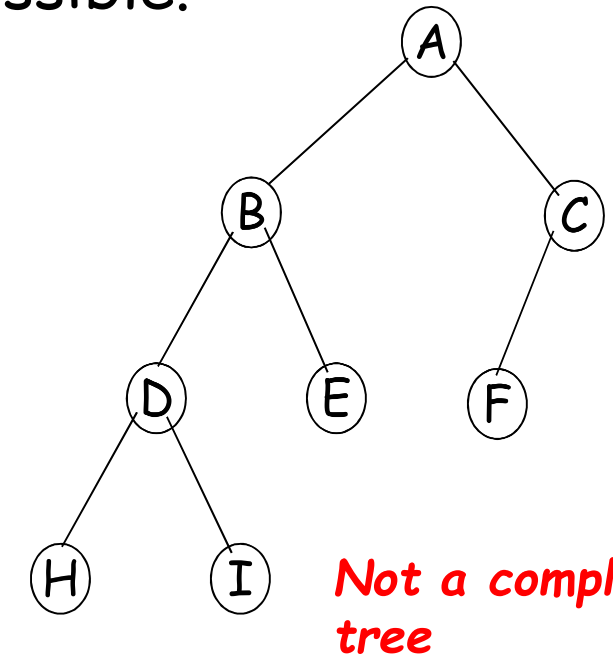
Full Binary Tree of depth 4 with sequential node numbers

Complete binary tree

- **Definition**: A binary tree with n nodes and depth k is ***complete*** iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k. In a complete binary tree, every level, except possibly the last, is completely filled, and all nodes are as far left as possible.



Complete binary tree



Not a complete binary tree

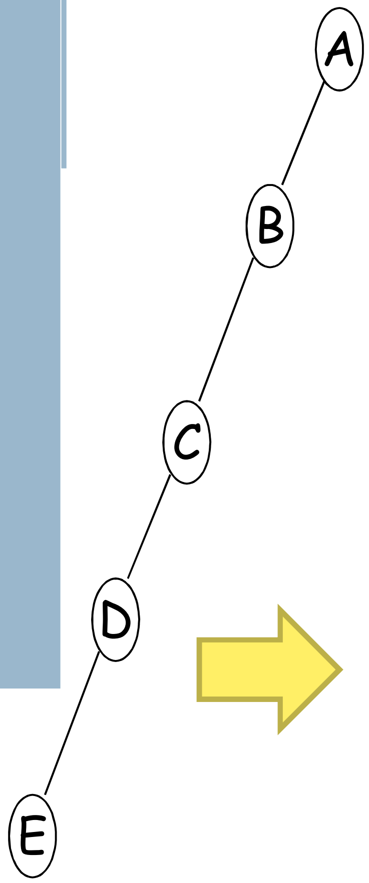
Storage representation of binary trees:

- Trees can be represented using
  - Linear/Sequential (Array) Representation
  - Linked Representation

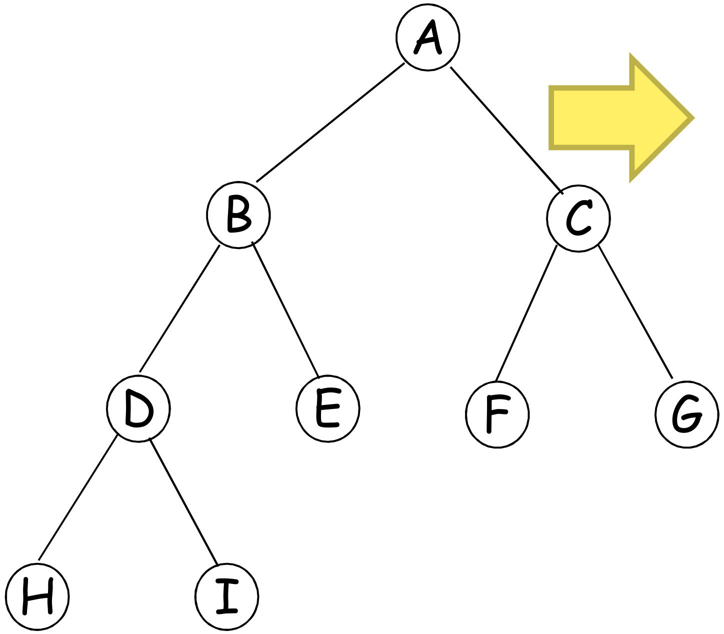
Array Representation of A Binary Tree

- Lemma 5.4: If a complete binary tree with n nodes is represented sequentially, then for any node with index i, $1 \leq i \leq n$, we have:
  - parent(i) is at $\lfloor i/2 \rfloor$ if $i \neq 1$. If $i = 1$, i is at the root and has no parent.
  - left_child(i) is at $2i$ if $2i \leq n$. If $2i > n$, then i has no left child.
  - right_child(i) is at $2i + 1$ if $2i + 1 \leq n$. If $2i + 1 > n$, then i has no right child.
- Position zero of the array is not used.

Array Representation of Binary Trees



[1]	A
[2]	B
[3]	—
[4]	C
[5]	—
[6]	—
[7]	—
[8]	D
[9]	—
⋮	⋮
[16]	E



[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

Advantages and disadvantages of Array representation

Advantages:

1. This representation is very easy to understand.
2. This is the best representation for full and complete binary tree representation.
3. Programming is very easy.
4. It is very easy to move from a child to its parents and vice versa.

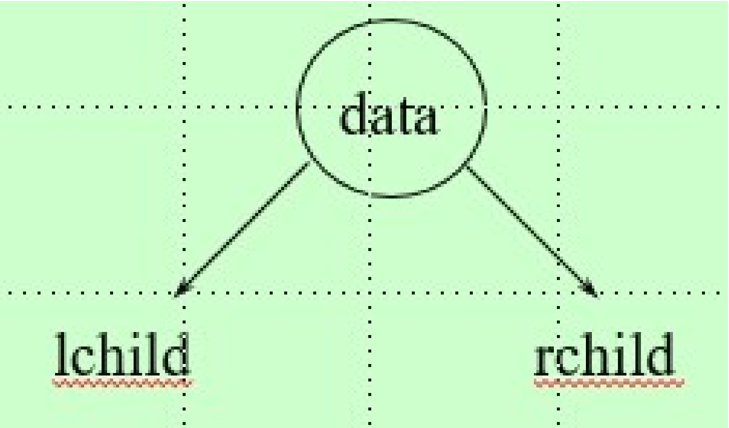
Disadvantages:

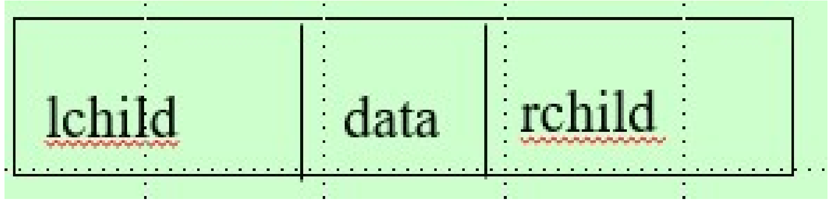
1. Lot of memory area wasted.
2. Insertion and deletion of nodes needs lot of data movement.
3. This is not suited for trees other than full and complete tree.

Linked Representation

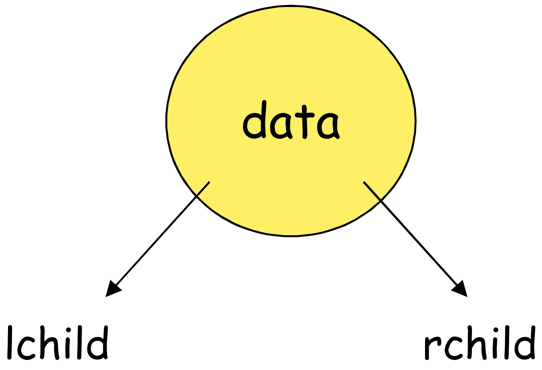
```
typedef struct node *Nodeptr;

struct node{
    int data;
    Nodeptr rchild;
    Nodeptr lchild;
};
```





Node Representation



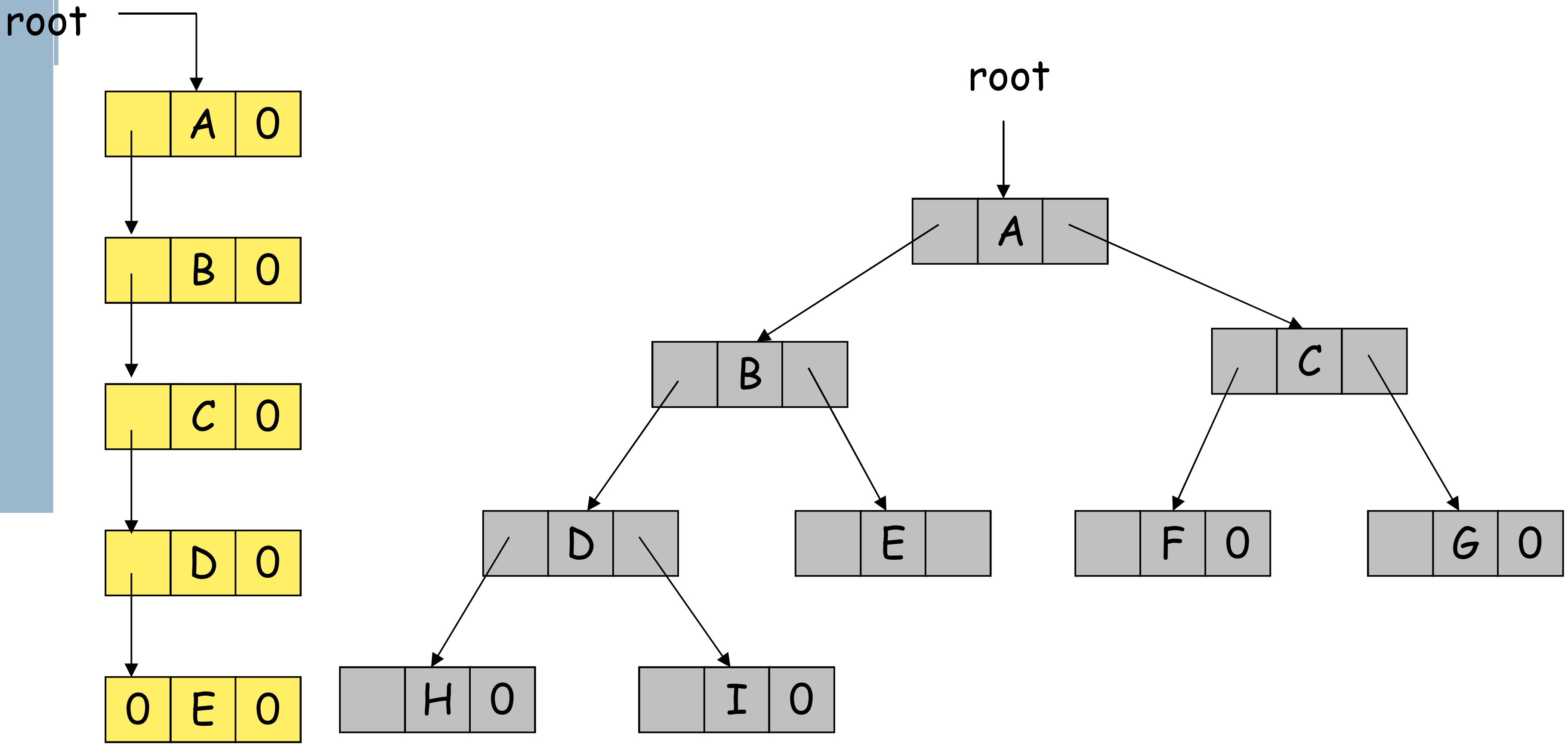
data

lchild

rchild

lchild	data	rchild

Linked List Representation For The Binary Trees



Advantages and disadvantages of linked representation

Advantages

1. A particular node can be placed at any location in the memory.
2. Insertions and deletions can be made directly without data movements.
3. It is best for any type of trees.
4. It is flexible because the system take care of allocating and freeing of nodes.

Disadvantage

1. It is difficult to understand.
2. Additional memory is needed for storing pointers
3. Accessing a particular node is not easy.

Recursive Function to create a binary tree

```
Nodeptr CreateBinaryTree(int item){
    int x;

    if (item!=-1)   { //until input is not equal to -1
            Nodeptr temp = getnode();
            temp->data = item;

            printf("Enter the lchild of %d :",item);
            scanf("%d",&x);
            temp->lchild = CreateBinaryTree(x);

            printf("Enter the rchild of %d :",item);
            scanf("%d",&x);
            temp->rchild = CreateBinaryTree(x);

            return temp;
    }
    return NULL;
}
```

```
int main()
{
   Nodeptr root = NULL;
    int item;

    printf("Creating the tree : \n");
    printf(“Enter the root : “);
    scanf("%d",&item);

    root=CreateBinaryTree(item);
          ...
```

Tree Traversal

- Let L, V, and R stand for moving left, visiting the node, and moving right.
- There are six possible combinations of traversal for a binary tree
  - LVR, LRV, VLR, VRL, RVL, RLV
- Adopt convention that we traverse left before right, only 3 traversals remain
  - LVR, LRV, VLR
  - →inorder, postorder, preorder
- When implementing the traversal, a recursion is perfect for the task.

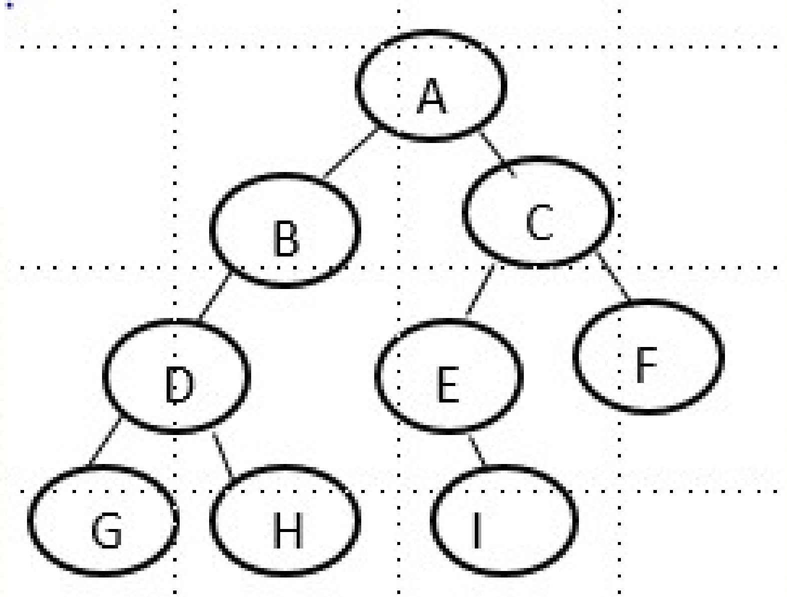
Tree Traversal

Inorder traversal

- It can be recursively defined as follows.
  1. Traverse the left subtree in inorder.
  2. Process the root node.
  3. Traverse the right subtree in inorder.

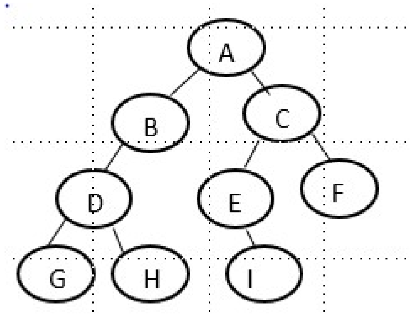
Inorder Traversal

- Move towards the left of the tree(till the leaf node), display that node and then move towards right and repeat the process.
- Since same process is repeated at every stage, recursion will serve the purpose.
- Example:



Inorder traversal of above tree gives GDHBAEICF

Inorder Traversal - Example



- Move towards left, we end up in G. G does not have a left child. Now display the root node(in this case it is G). Hence G is displayed first.
- Move to the right of G, which is also NULL. Hence go back to root of G and print it. So D is printed next.
- Go to the right of D, which is H. Now another root H is visited.
- Move to the left of H, which is NULL. So go back to root H and print it and go to right of H, which is NULL.
- Go back to the root B and print it and go right of B, which is NULL. So go back to root of B, which is A and print it.
- Traversing of left subtree is finished and so move towards right of it & reach C.
- Move to the left of C and reach E. Again move to left, which is NULL. Print root E and go to right of E to reach I.
- Move to left of I, which is NULL. Hence go back to root I, print it and move to its right, which is NULL.
- Go back to root C, print it and go to its right and reach F.
- Move to left of F, which is NULL. Hence go back to F, print it and go to its right, which is also NULL.