Tree
Data Structure

Problem: Family Information

You have to implement a menu based program in which you have the option to Add a person in the family, Search a person, View parent, View children and View all members of the family.

```
1. Add a person
2. Search a person
3. View the Parent of a Person
4. View the Children of a Person
5. View all People in the Family
6. Exit
Your Option---->
```

Family Information: Add a Person

On pressing option 1, user is asked for the information of the person.

```
Your Option----> 1
Enter Name: Dada
Enter Age: 70
Enter Gender: m
Press any Key to Continue...
```

If it is
the first Person

```
Your Option----> 1
Enter Name: Abbu
Enter Age: 45
Enter Gender: m
Enter the name of the Parent: Dada
Press any Key to Continue...
```

If it is other than
the first Person

Family Information: Search a Person

On pressing option 2, the information of the asked person is displayed.

```
Your Option----> 2
Enter the name of the Person: Abbu
Name: Abbu
Age: 45
Gender: Male
Press any Key to Continue...
```

If the person is found

```
Your Option----> 2
Enter the name of the Person: Abba
Person not Found
Press any Key to Continue...
```

If the person is not found

Family Information: View the Parent

On pressing option 3, the information of the parent of a specific person is displayed.

```
Your Option----> 3
Enter the name of the Child: Abbu
Parent Found
Name: Dada
Age: 70
Gender: Male
Press any Key to Continue...
```

If the parent is found

```
Your Option----> 3
Enter the name of the Child: Dada
Press any Key to Continue...
```

If the parent is not found

Family Information: View the Children

On pressing option 4, the information of the children of a specific person is displayed.

```
Your Option----> 4
Enter the name of the Parent: Abbu
First Child Found
Name: Bhai
Age: 20
Gender: Male
Second Child Found
Name: Behan
Age: 15
Gender: Female
Press any Key to Continue...
```

If the children are found

```
Your Option----> 4
Enter the name of the Parent: Bhai
No Child Found
Press any Key to Continue...
```

If the child is not found

Family Information: View the Children

On pressing option 5, the information of the whole family is displayed.

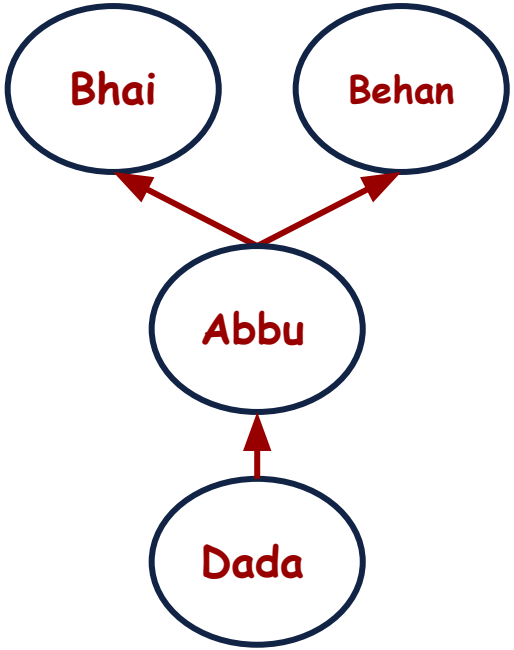
```
Your Option----> 5
Dada
Abbu
Bhai Behan
Press any Key to Continue...
```

Family Information: How to Store?

How can we store this information effectively so that the CRUD operations can be easily performed?

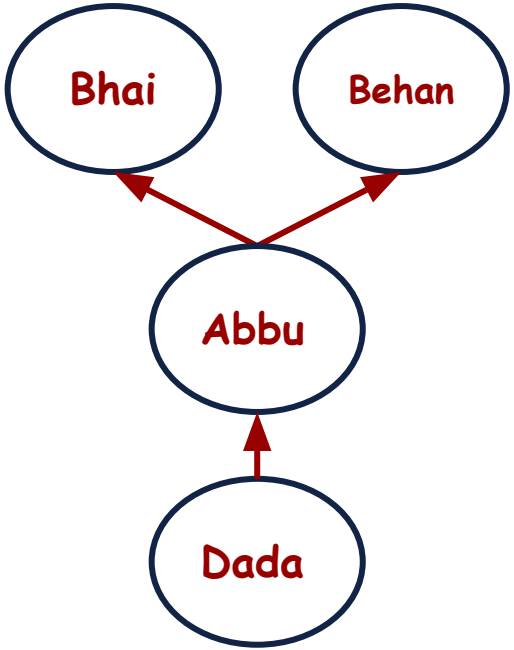
Family Information: How to Store?

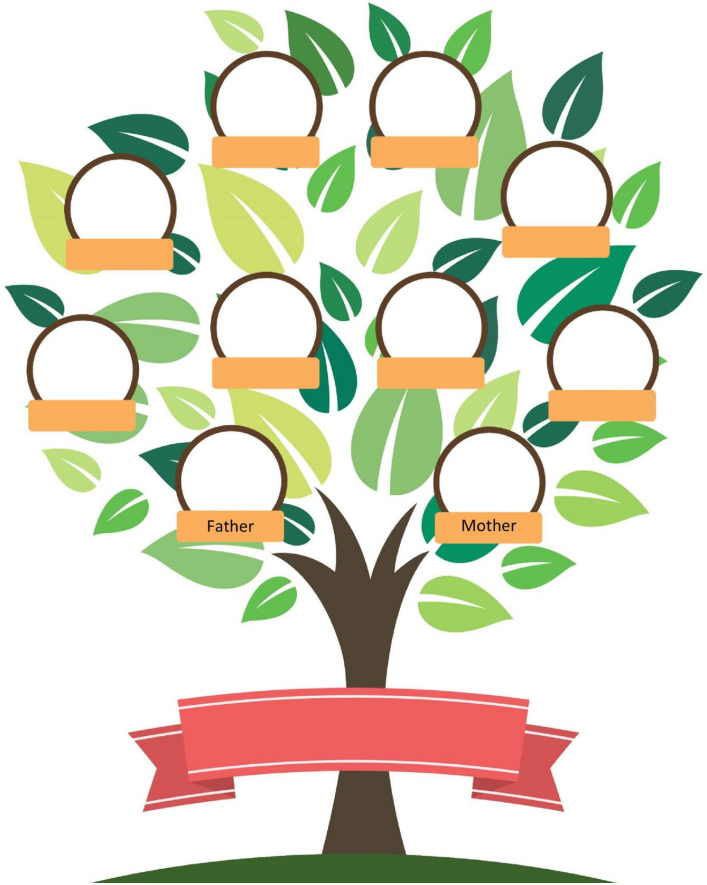
Let's first Draw the information of the Family hierarchically.



Family Information

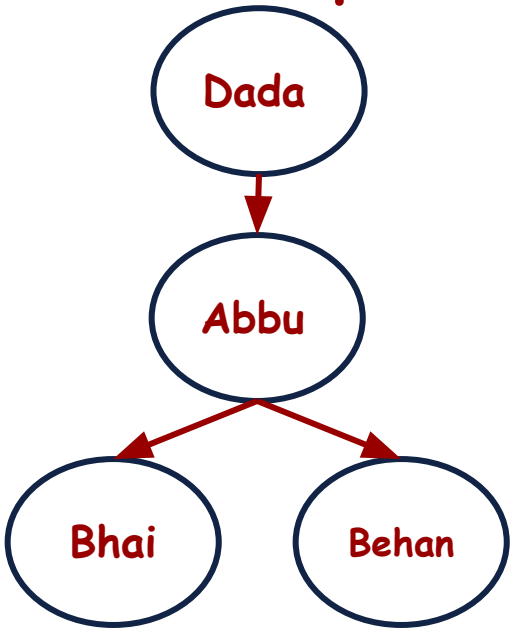
This looks like a Family Tree.

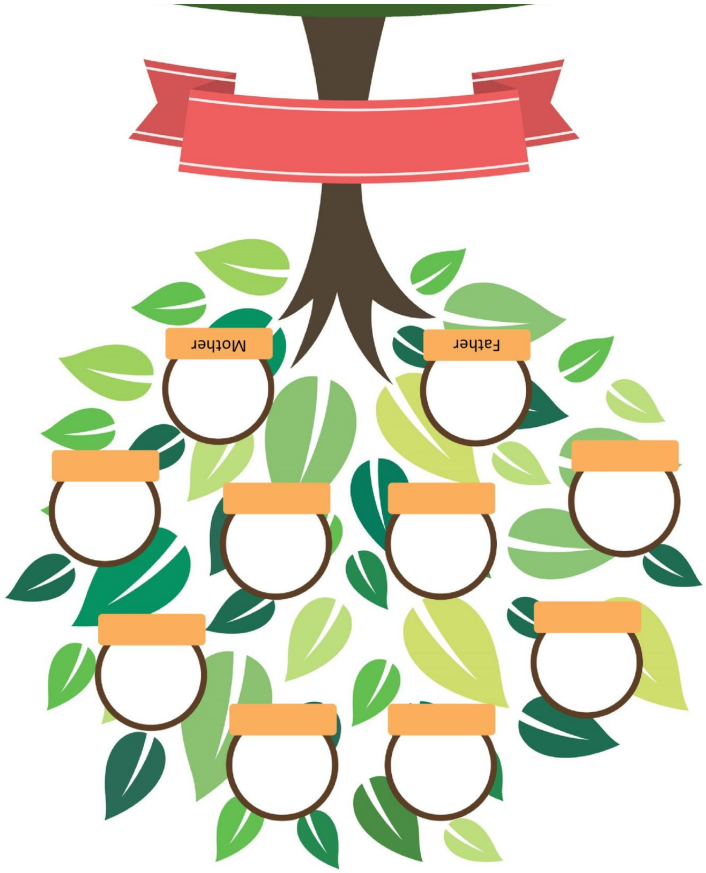




Family Information

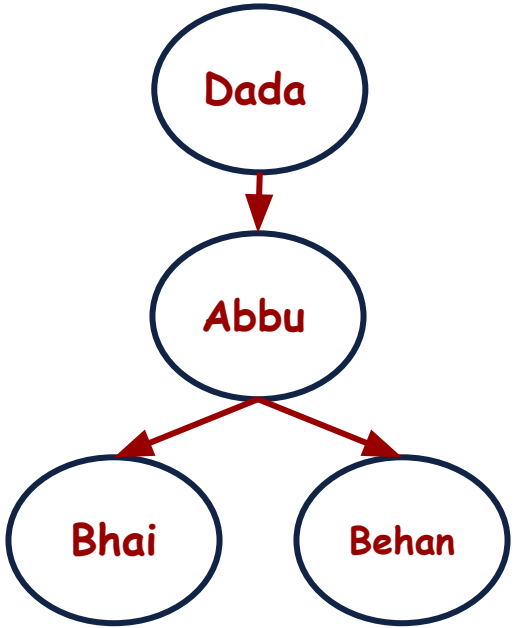
Computer Scientists like to look at the Tree upside down.





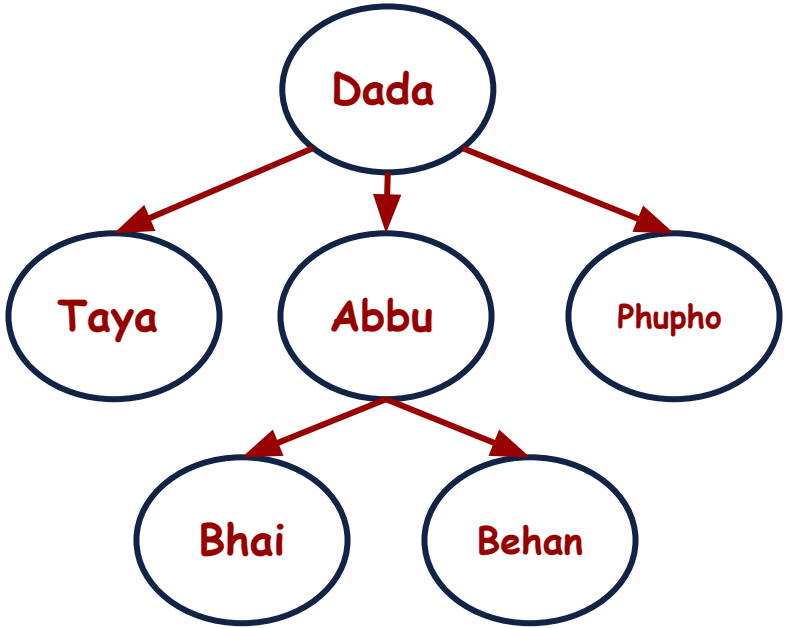
Tree: Data Structure

This is a new Data Structure (non-linear (Hierarchical) Data Structure).



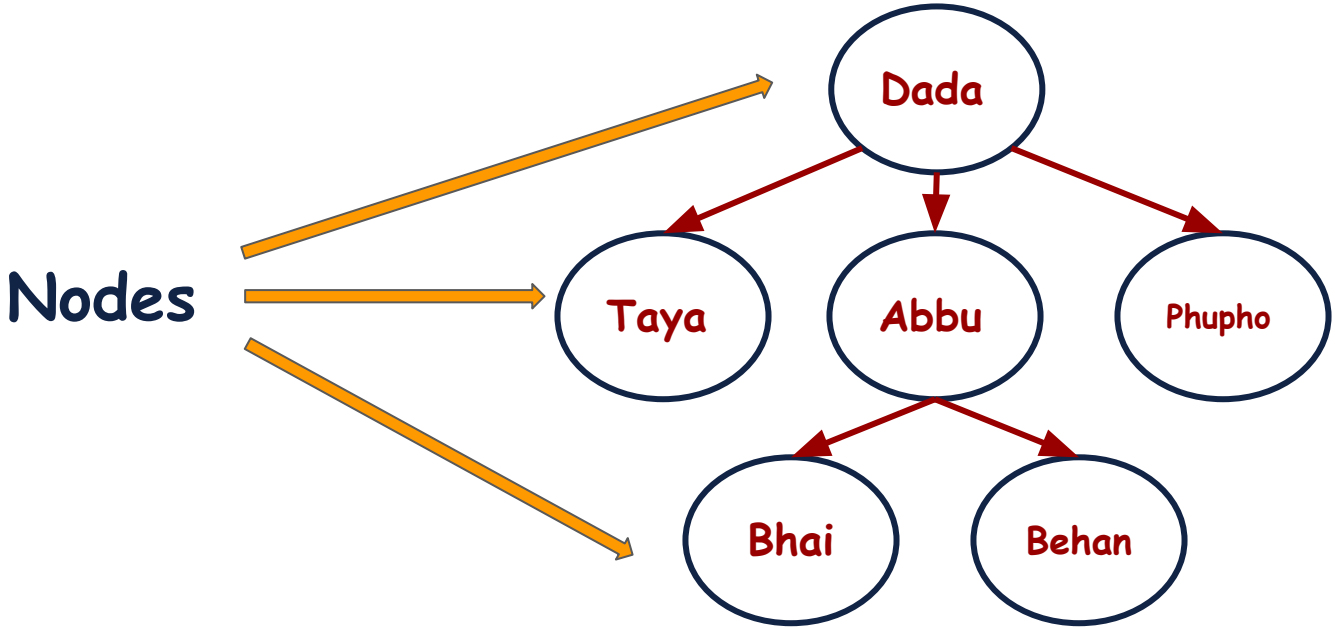
Tree: Terminologies

Every new Data Structure comes with new **Terminologies**.



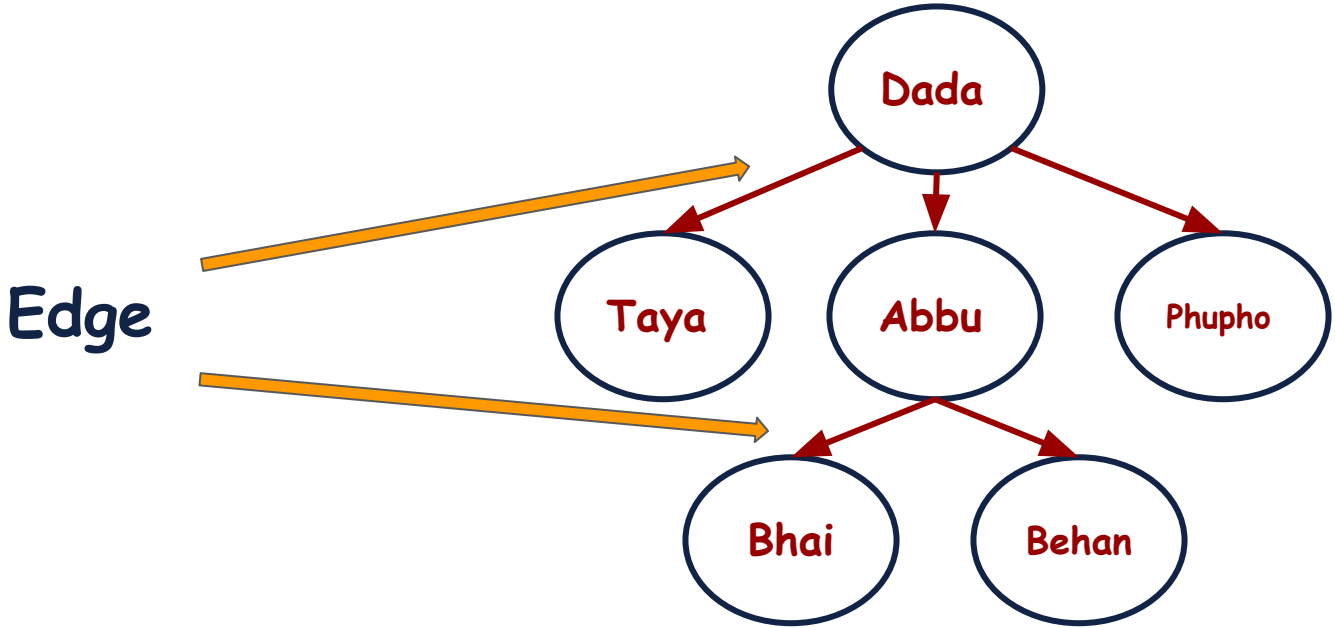
Terminologies: Node

Tree Data Structure is also known as the collection of nodes.



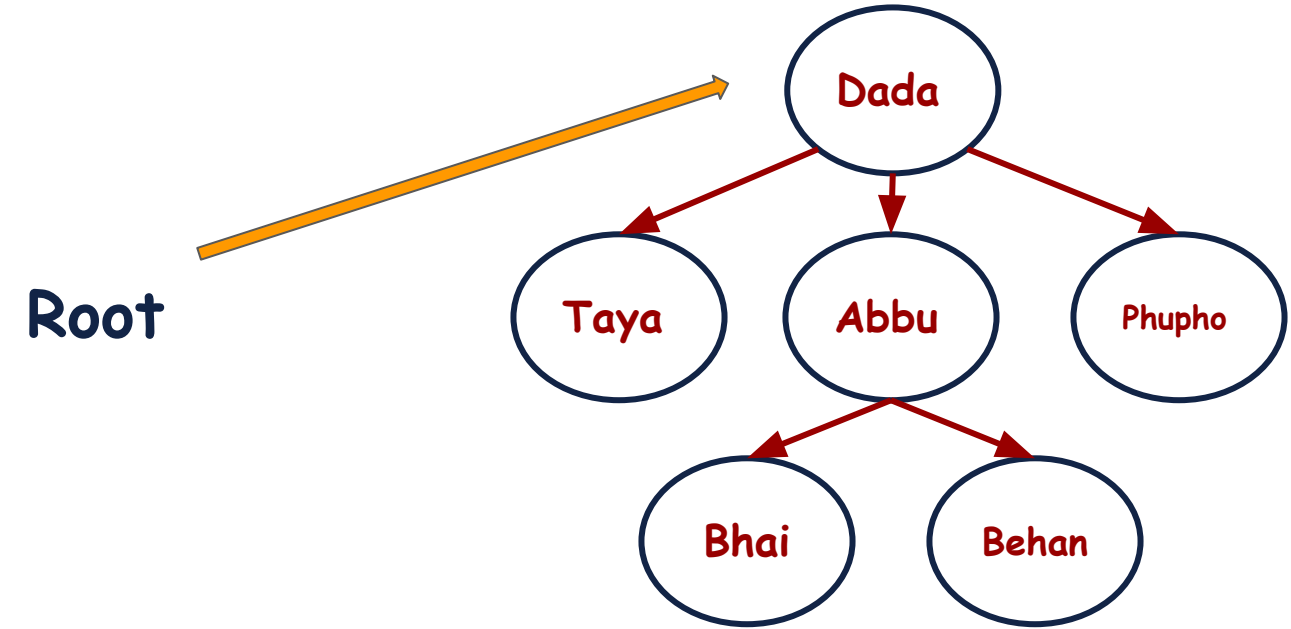
Terminologies: Edges or Vertices

Edge is a connection between one node to another. It is a line **between two nodes**.



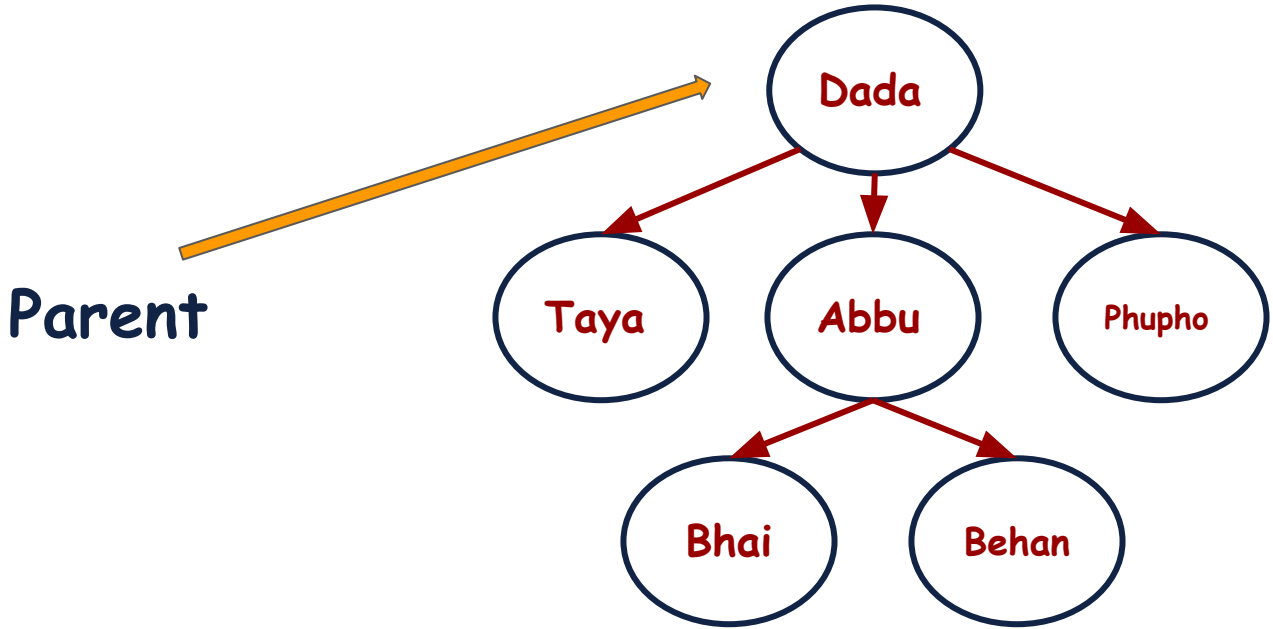
Terminologies: Root Node

Root is a special node in a tree. The **entire tree** originates from it.



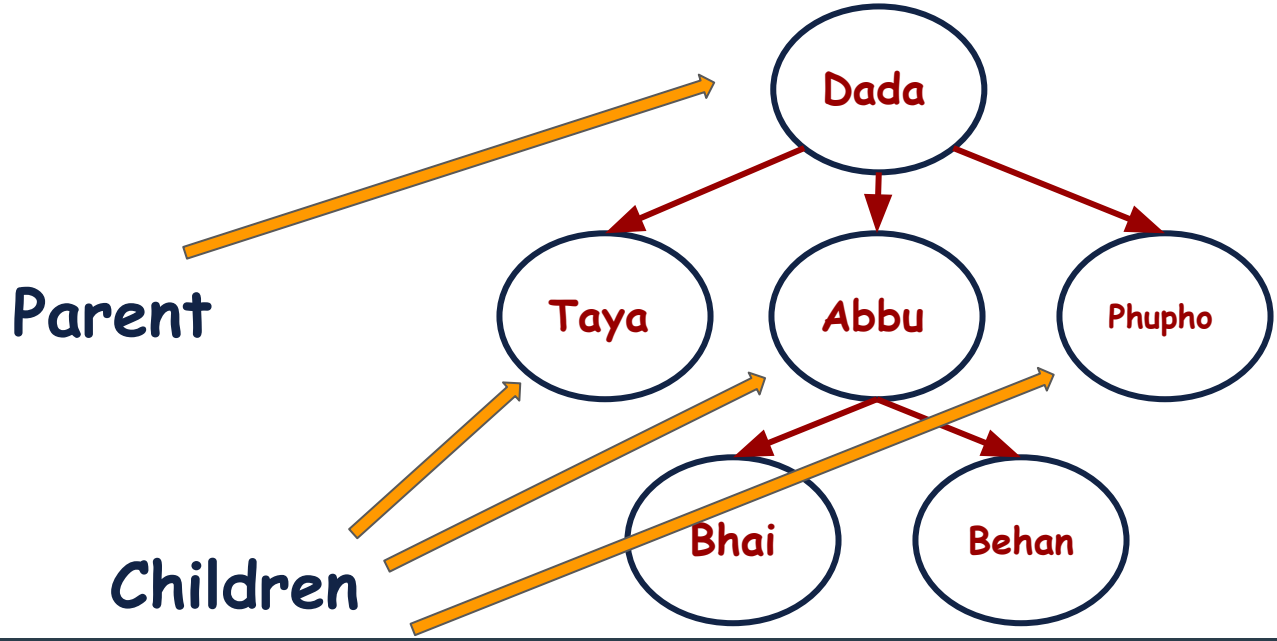
Terminologies: Parent

Parent node is an immediate predecessor of a node.



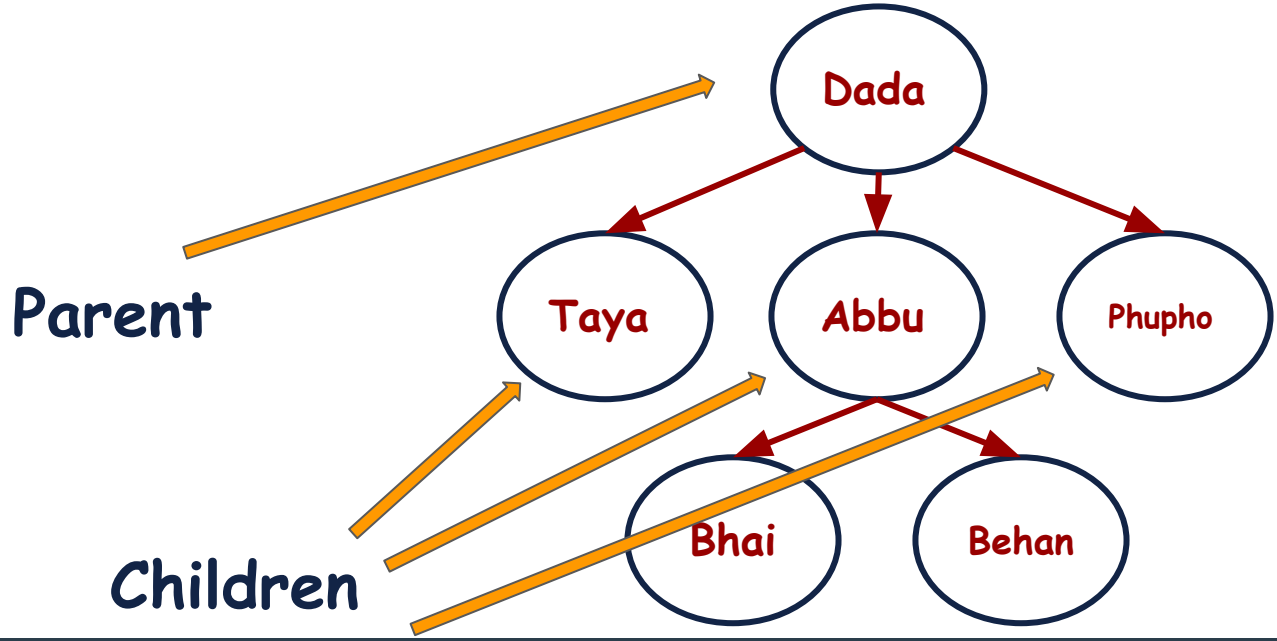
Terminologies: Child

All immediate successors of a node are its children.



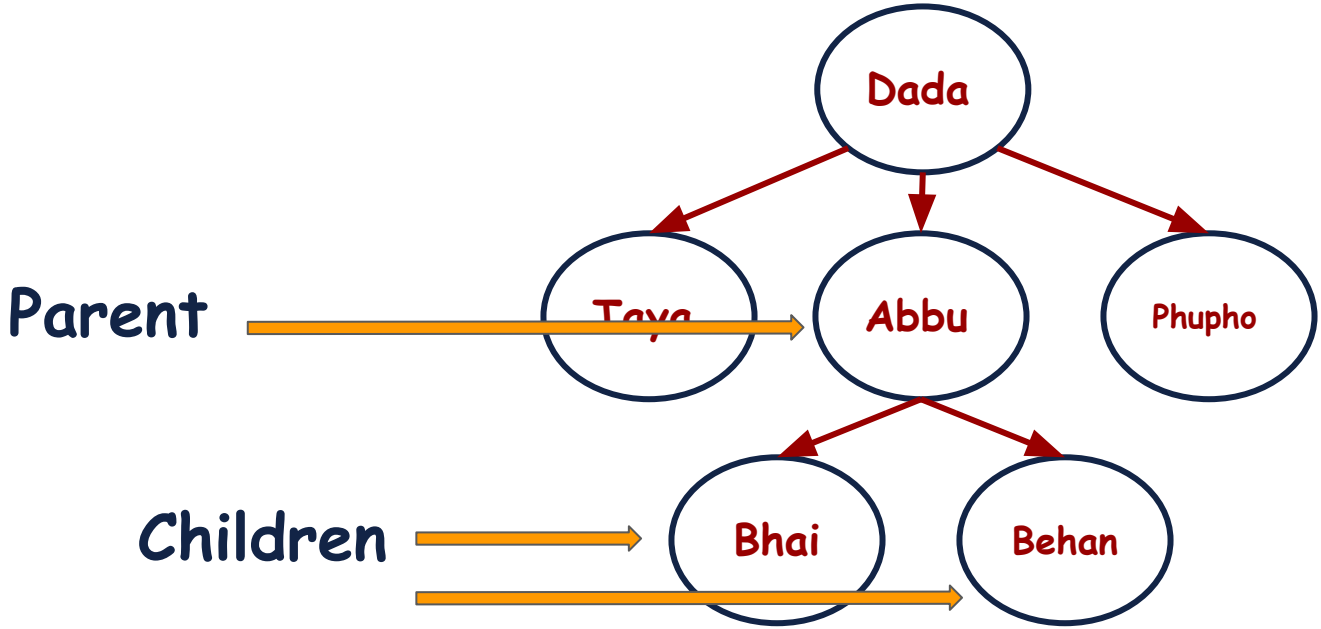
Terminologies: Parent and Children

Each node has one parent only but can have multiple children.



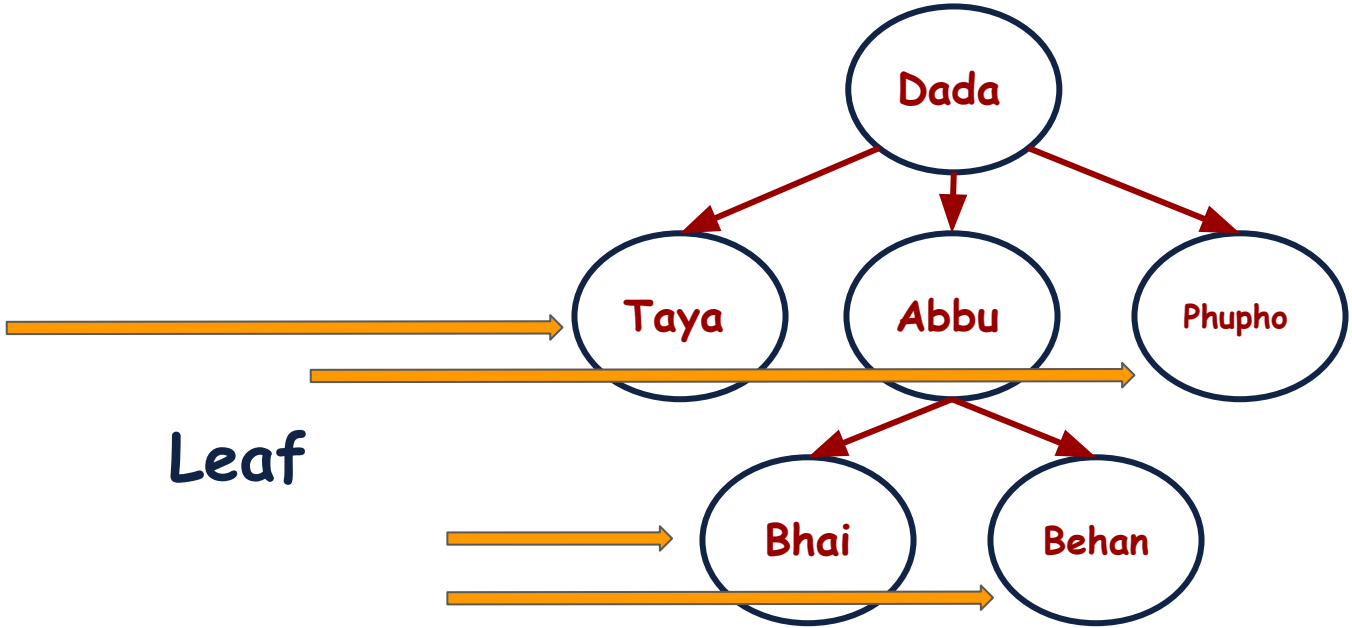
Terminologies: Parent and Children

Each node has one parent only but can have multiple children.



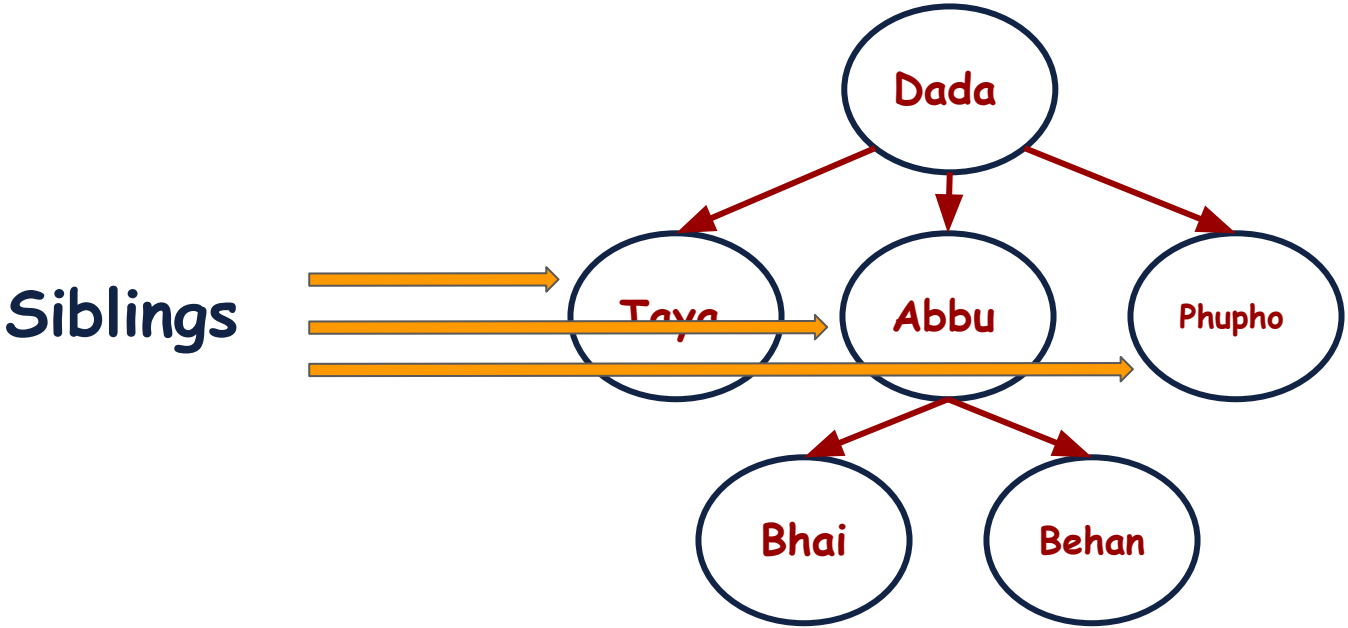
Terminologies: Leaf

Node which does not have any child is called as **leaf.**



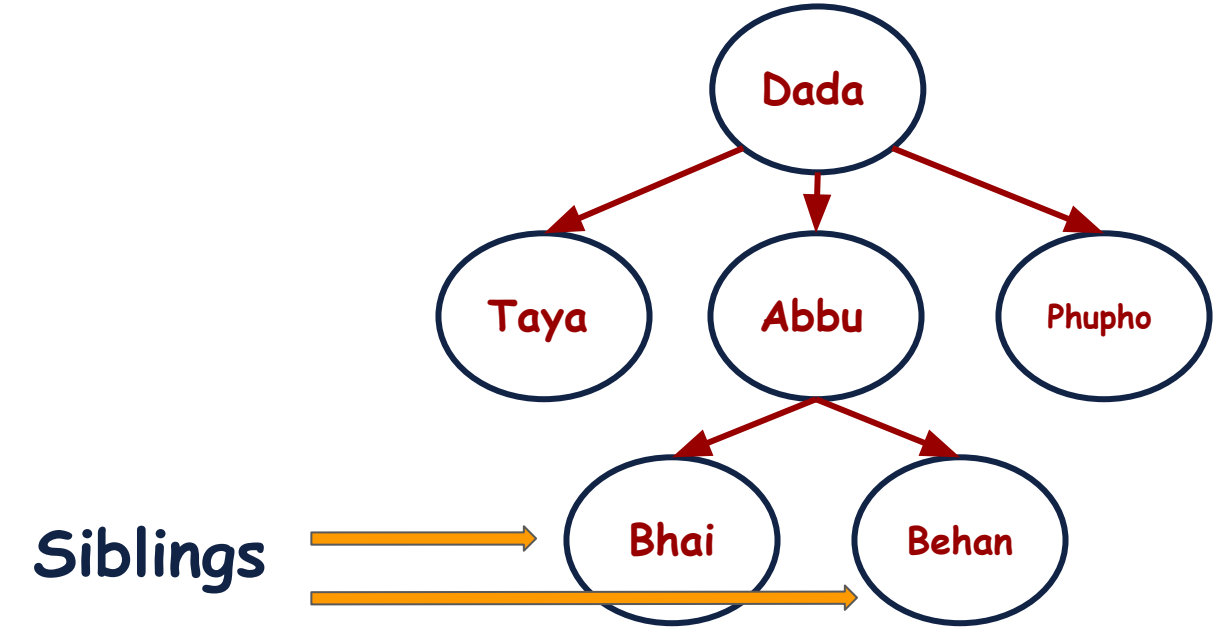
Terminologies: Siblings

Nodes with the same parent are called Siblings.



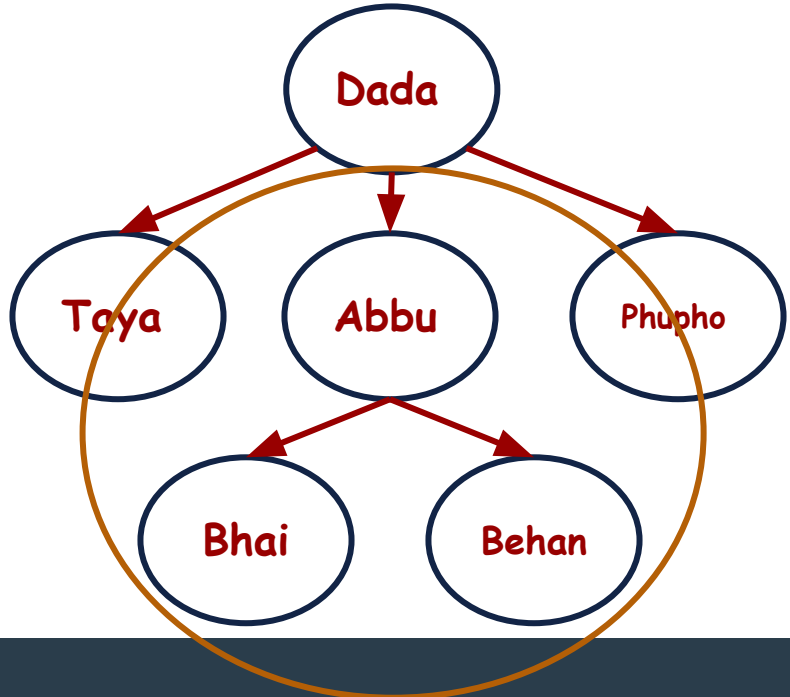
Terminologies: Siblings

Nodes with the same parent are called Siblings.



Terminologies: SubTree

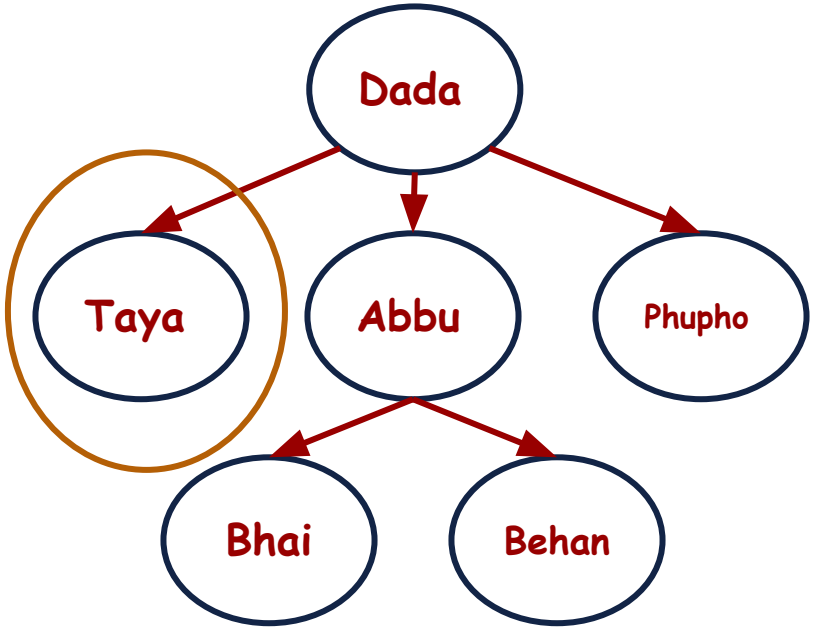
Descendants of a node represent subtree.



Sub Tree

Terminologies: SubTree

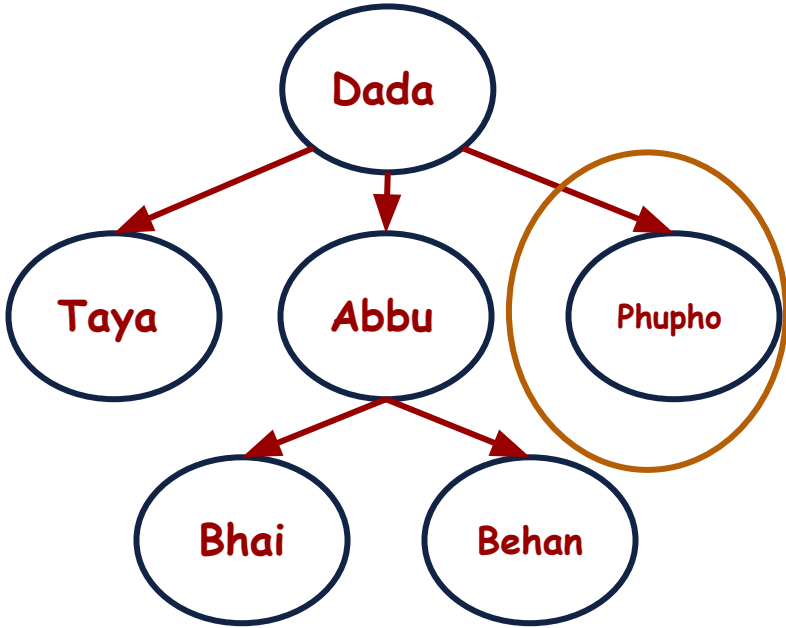
Descendants of a node represent subtree.



Sub Tree

Terminologies: SubTree

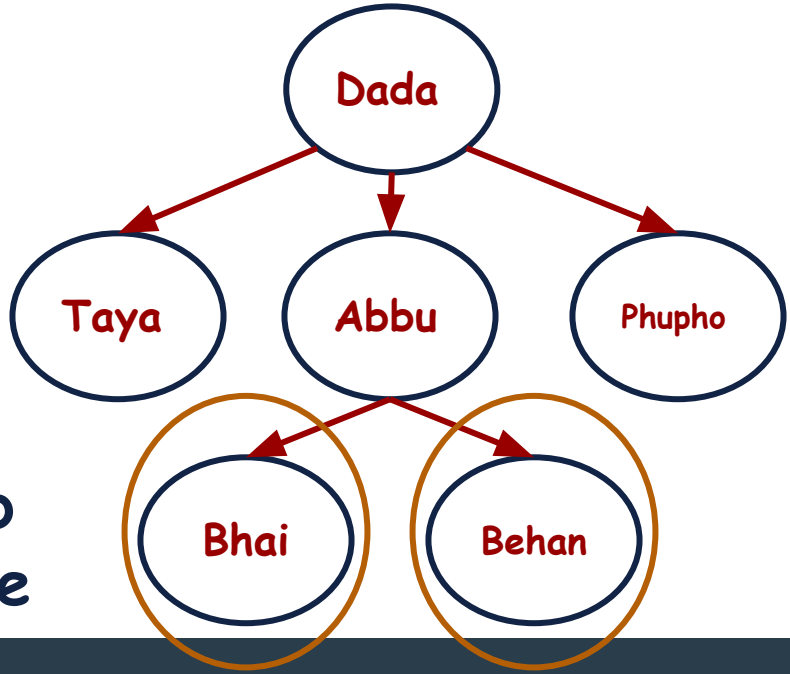
Descendants of a node represent subtree.



Sub
Tree

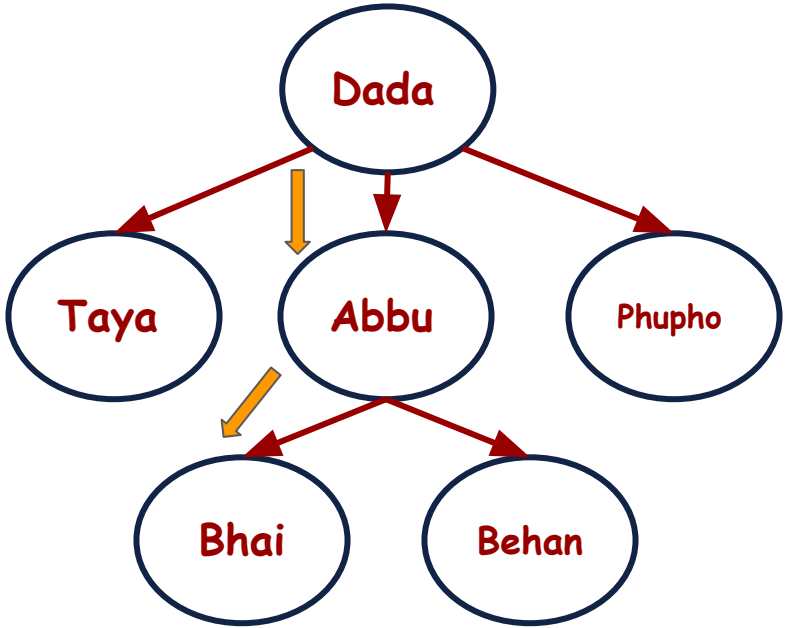
Terminologies: SubTree

Descendants of a node represent subtree.



Terminologies: Path

Path is a number of successive edges from source node to destination node.

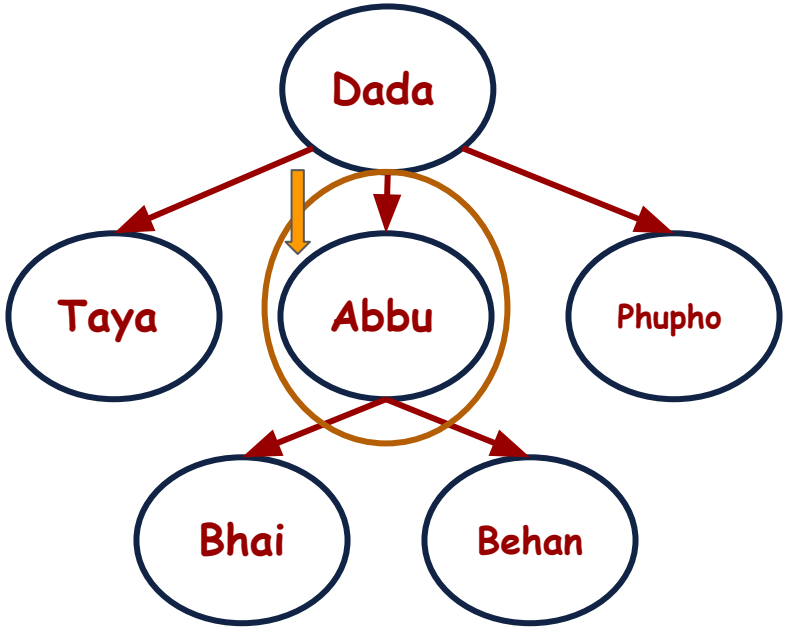


Path

Terminologies: Depth of Node

Depth of a node represents the number of edges in path from **root to the node.**

Depth = 1



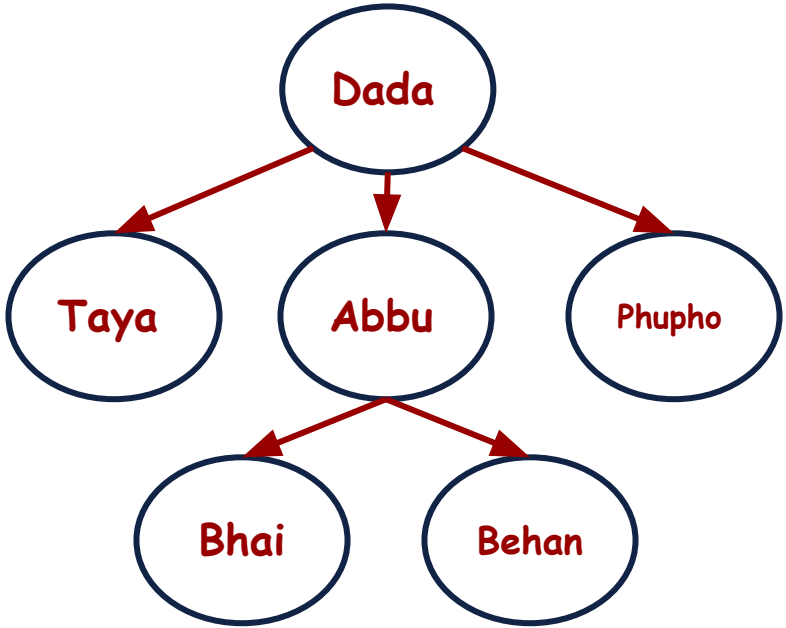
Terminologies: Depth of Node

Depth of a node represents the number of edges in path from **root to the node**.

Depth = 0

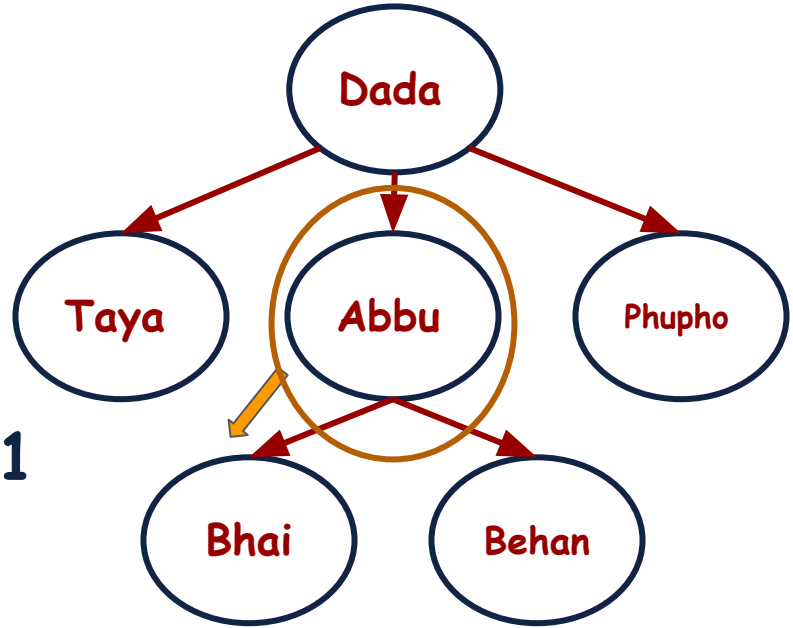
Depth = 1

Depth = 2



Terminologies: Height of Node

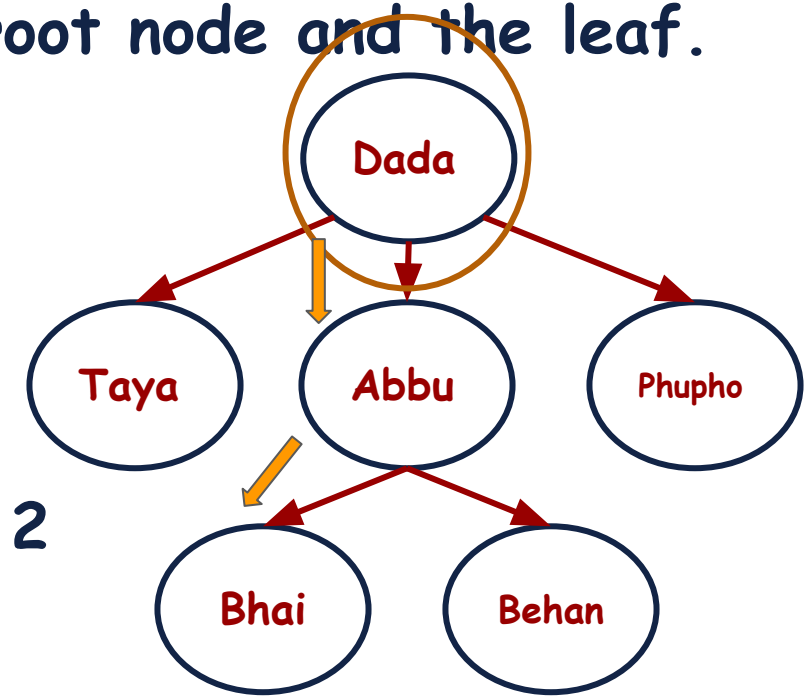
Height of a node represents the number of edges on the longest path between that node and a leaf.



Height = 1

Terminologies: Height of the Tree

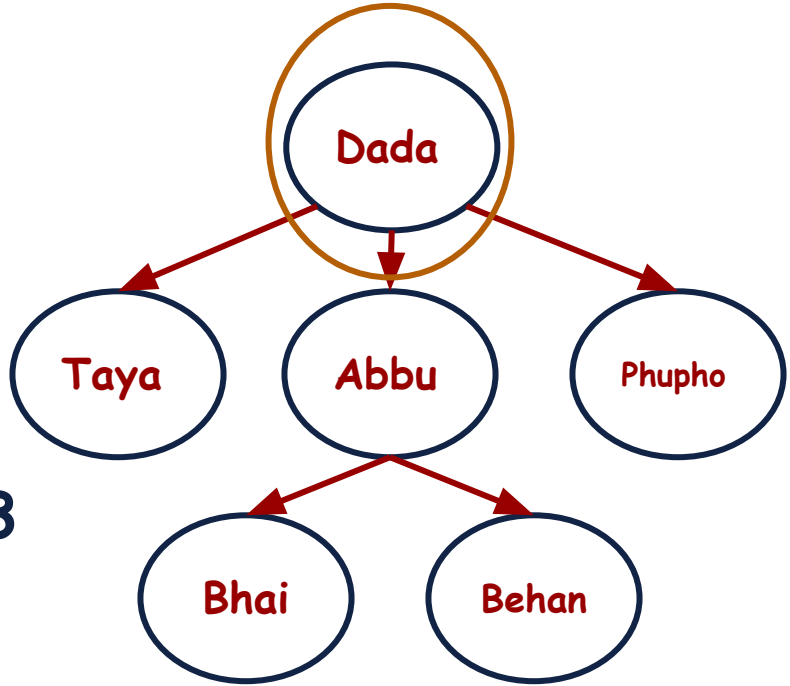
Height of the tree represents the number of edges on the longest path between that root node and the leaf.



Height = 2

Terminologies: Degree of a node

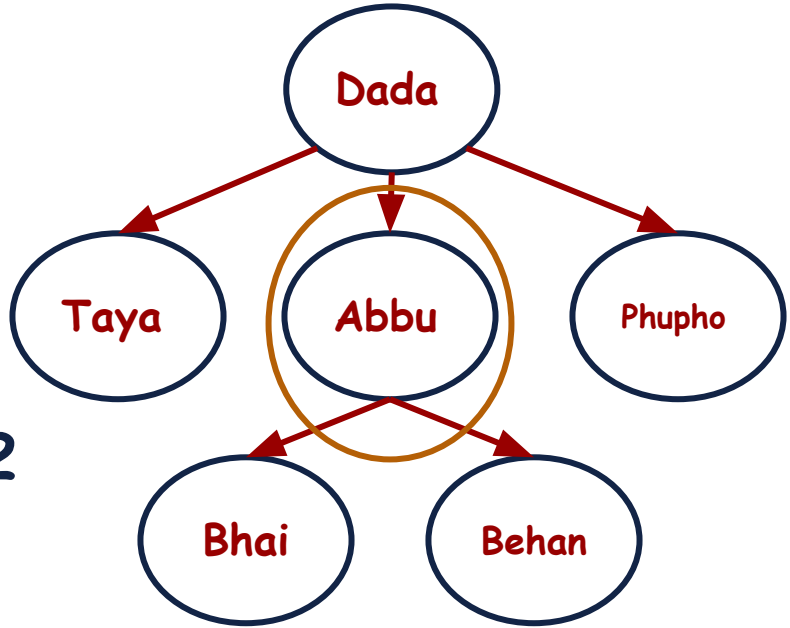
Degree of a node represents the number of children of a node.



Degree = 3

Terminologies: Degree of a node

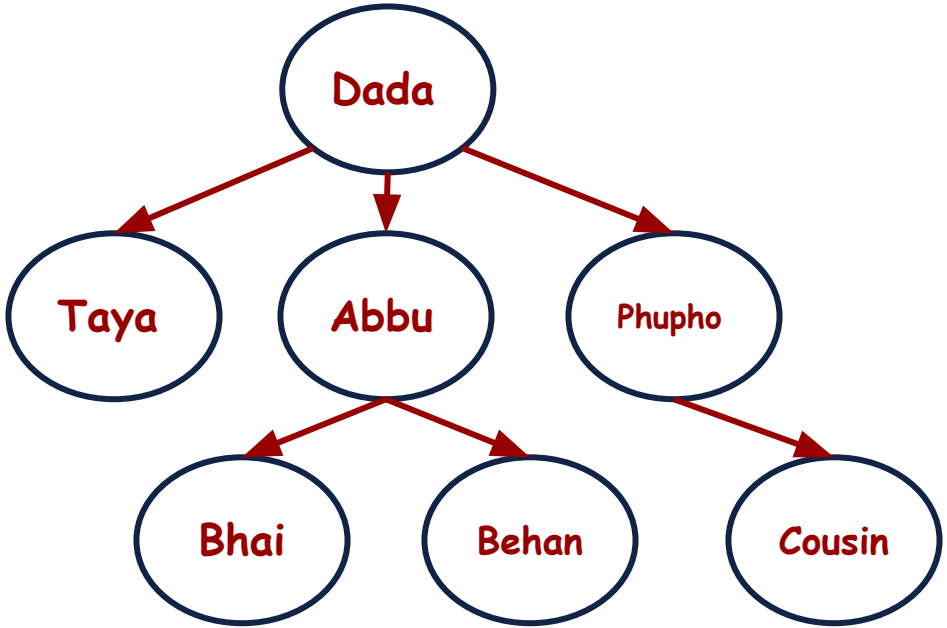
Degree of a node represents the number of children of a node.



Degree = 2

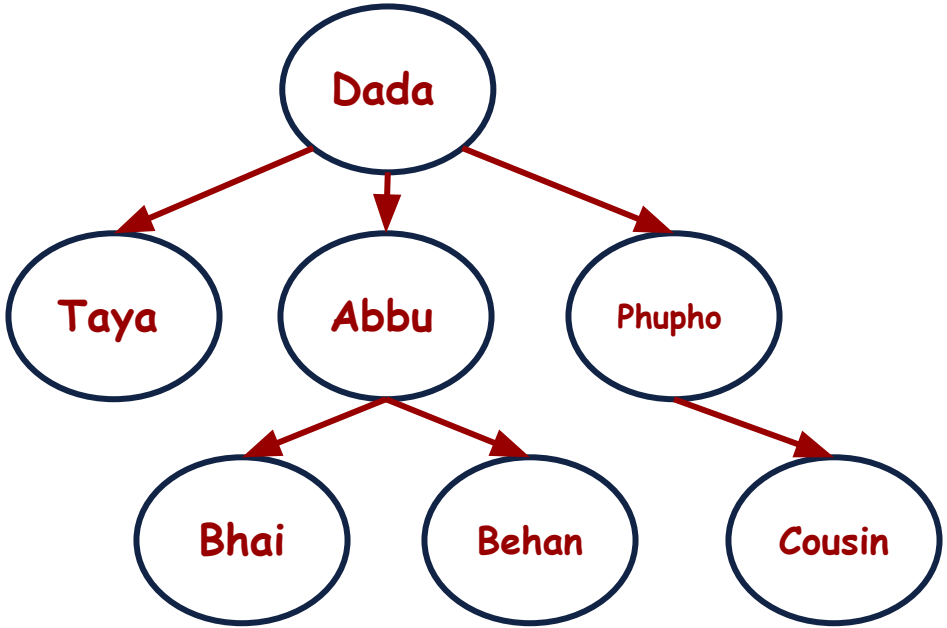
Types of Trees

Types of trees depend on the number of children (Degree) a node has.



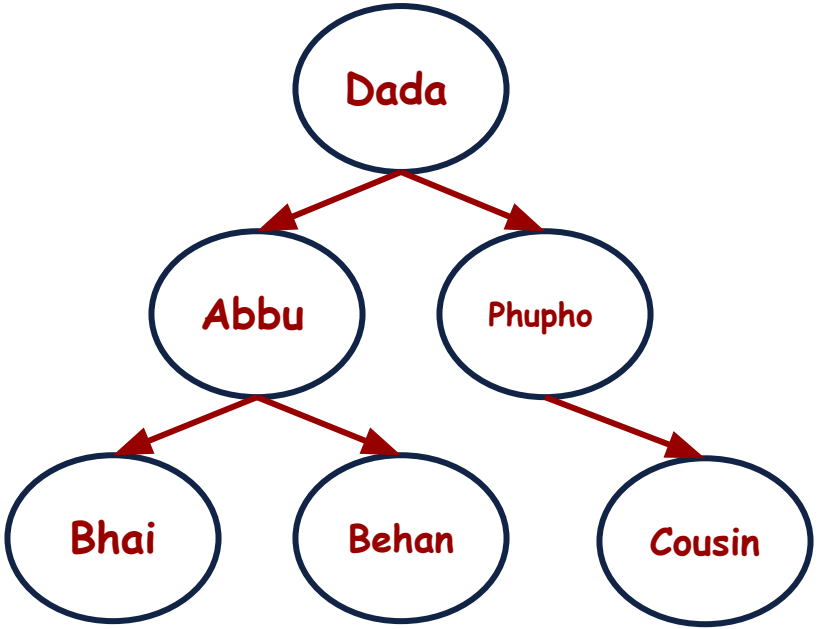
Types of Trees: General Trees

A tree in which there is no restriction on the number of children a node has, is called a General tree.



Types of Trees: Binary Trees

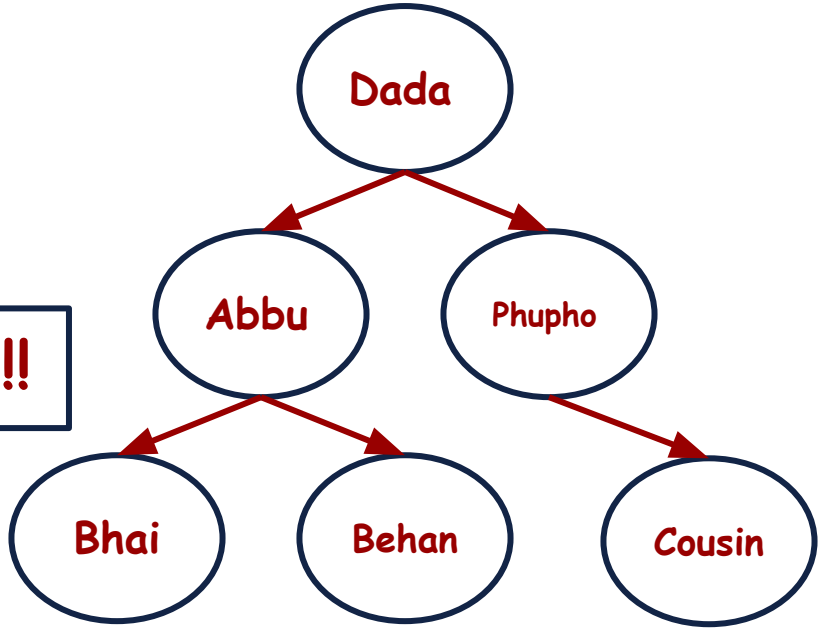
Binary tree is a special case of general trees where every node can have at most 2 children, left and right.



Types of Trees: Binary Trees

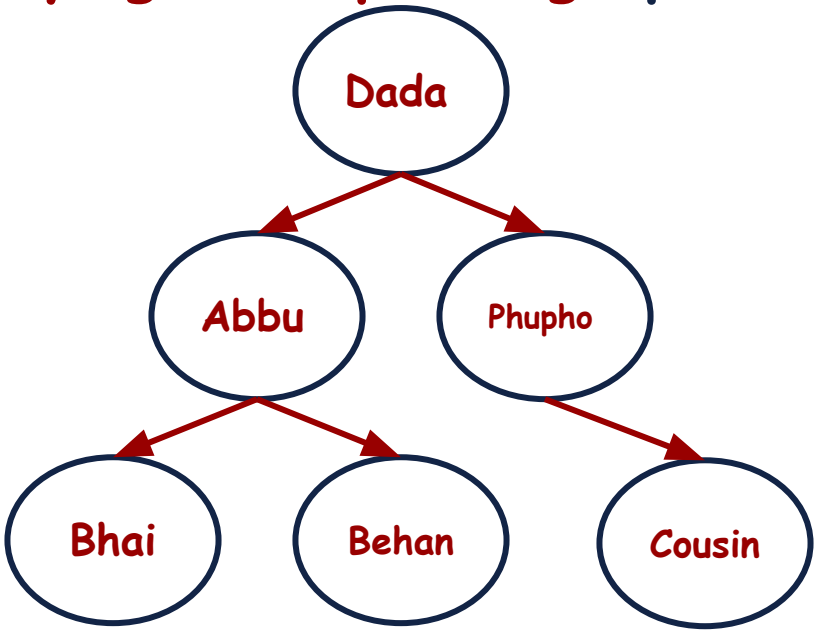
Binary tree is a special case of general trees where every node can have **at most 2 children**, left and right.

Bache 2 hi ache..!!!



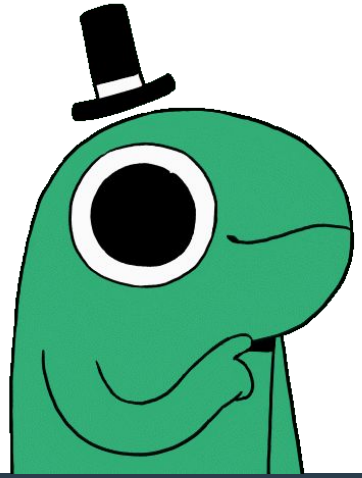
Binary Trees: Most Popular

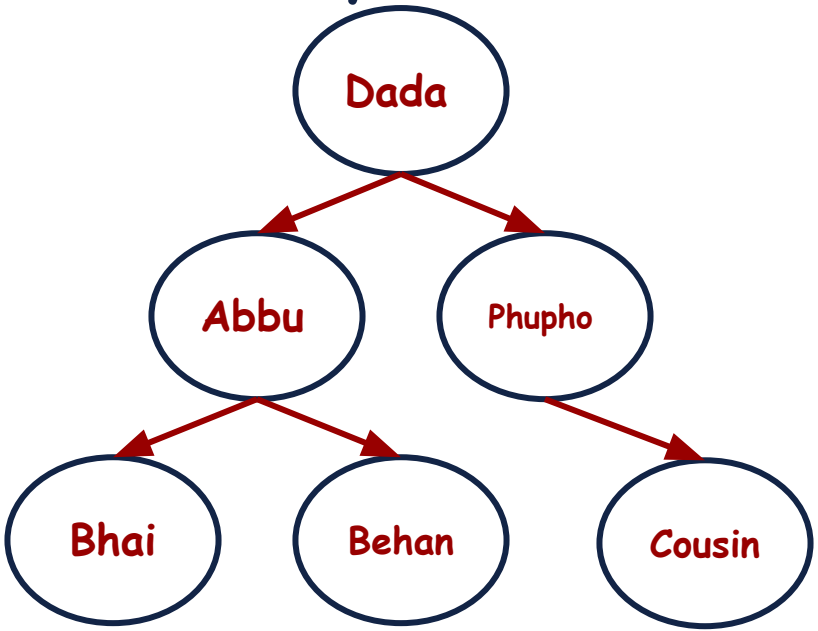
Binary trees are most popular than general trees because of simplifying and speeding up searching and sorting.



Binary Trees: Implementation

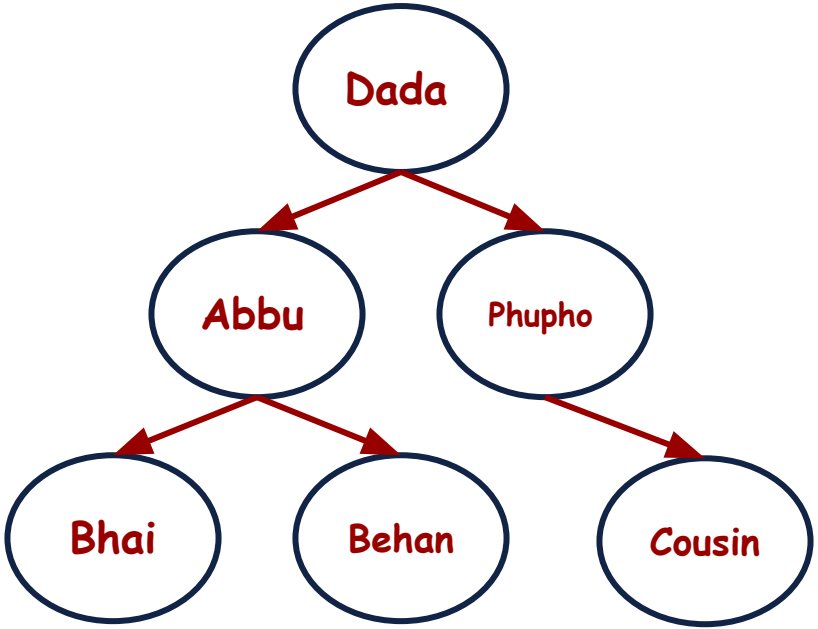
Let's come to the most awaited Question.
How to implement the Binary Trees?

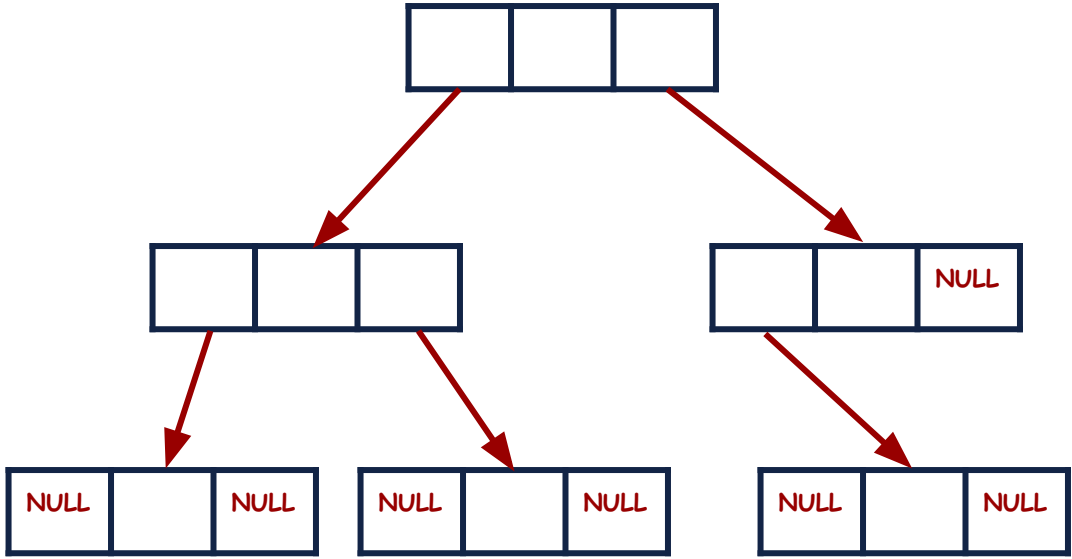




Binary Trees: Implementation

Let's create the nodes dynamically.

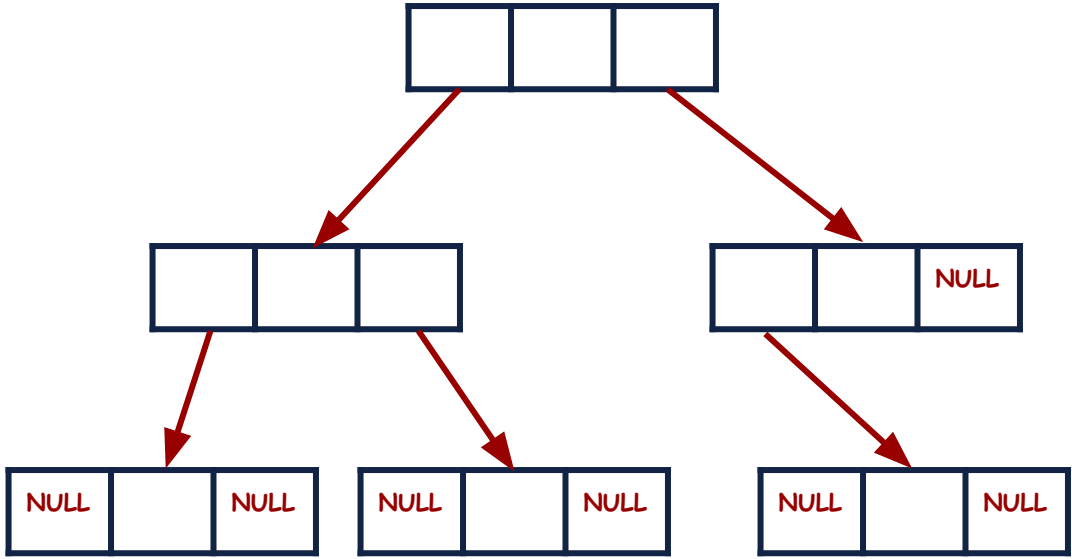




Binary Trees: Implementation

Let's create the nodes dynamically.

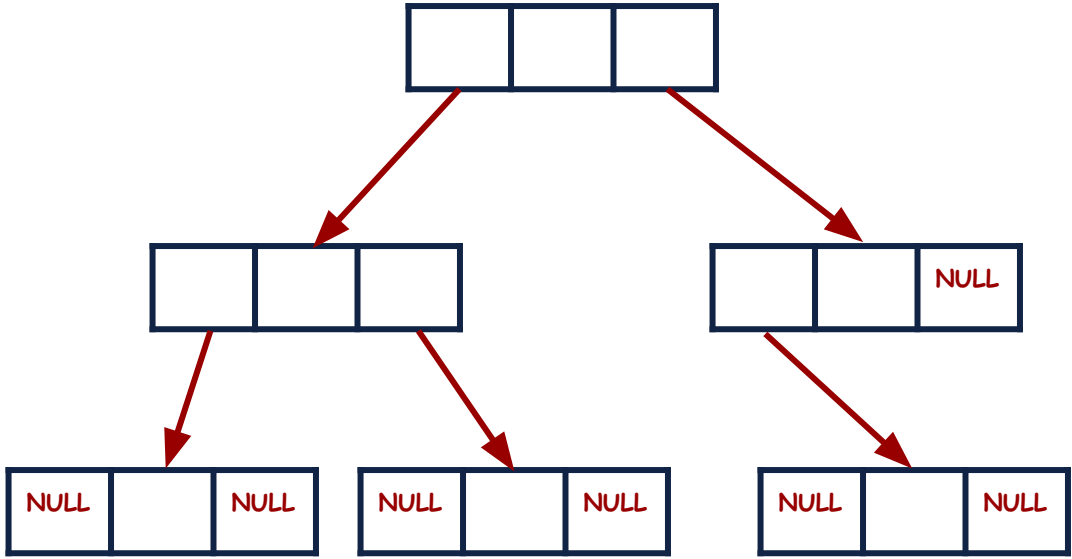
```
struct node
{
     string name;
     int age;
     char gender;
     node *left;
     node *right;
};
```



Binary Trees: Implementation

Let's create the nodes dynamically.

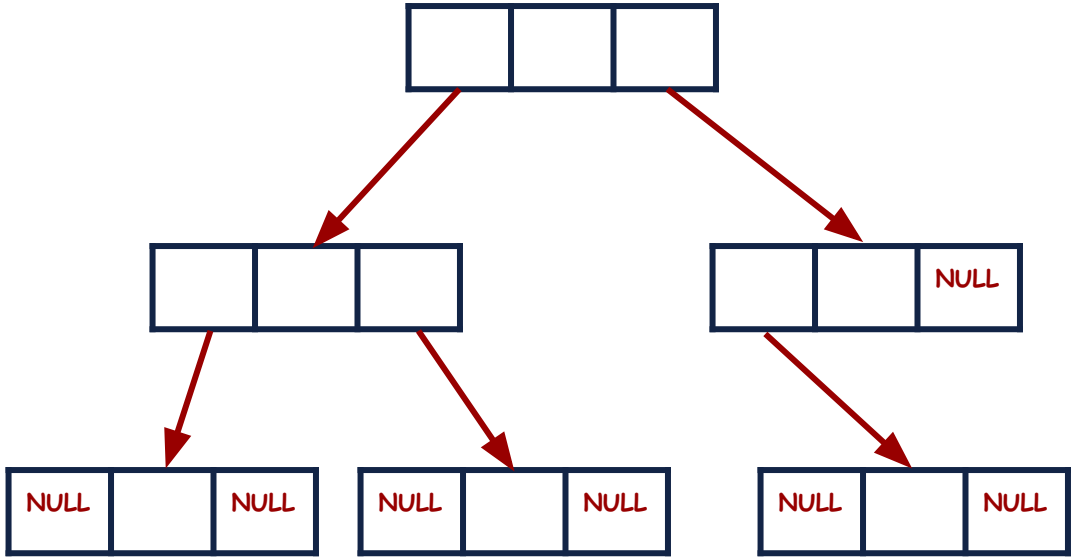
```
struct node
{
     string name;
     int age;
     char gender;
     node *left;
     node *right;
};
```



Binary Trees: Implementation

In order to add the new node, we should know where to add that (search the tree).

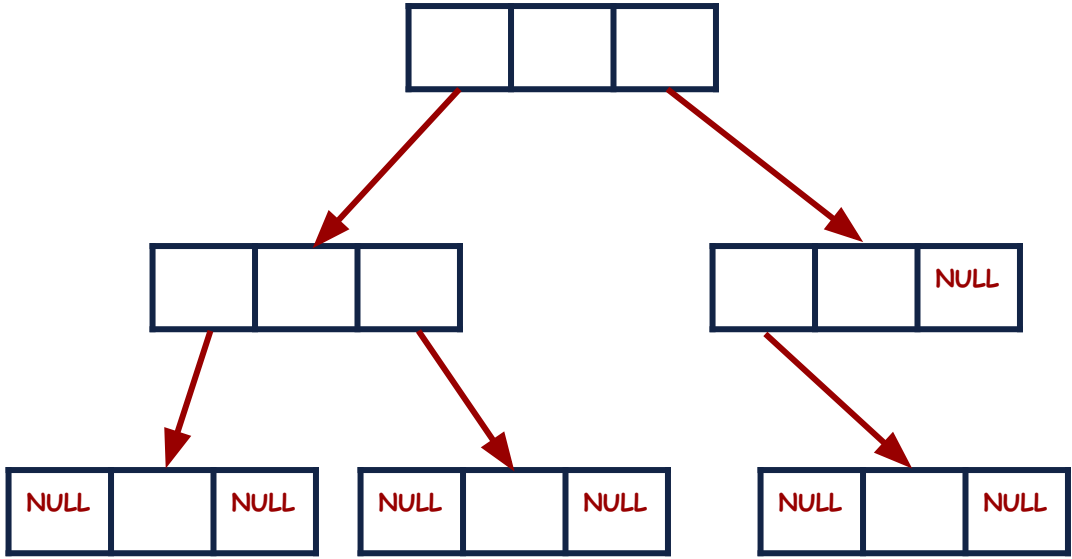
```
struct node
{
     string name;
     int age;
     char gender;
     node *left;
     node *right;
};
```



Binary Trees: Implementation

In order to search the tree we must know how to **traverse** the tree.

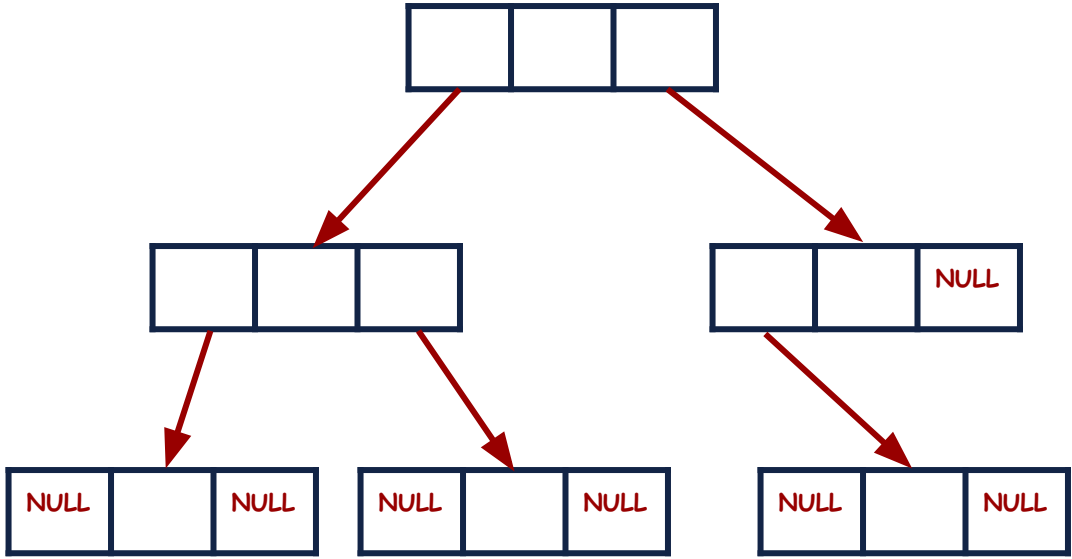
```
struct node
{
     string name;
     int age;
     char gender;
     node *left;
     node *right;
};
```



Binary Trees: View / Traversal

Now, how to view or traverse all the nodes of the Tree?

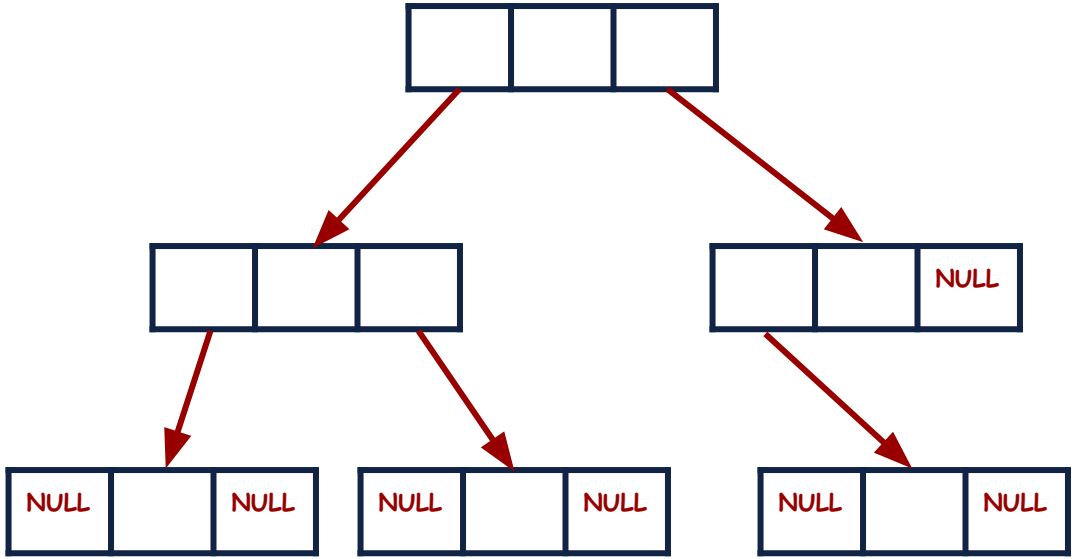
```
struct node
{
    string name;
    int age;
    char gender;
    node *left;
    node *right;
};
```



Binary Trees: View / Traversal

Should we go towards the left child or right child first?

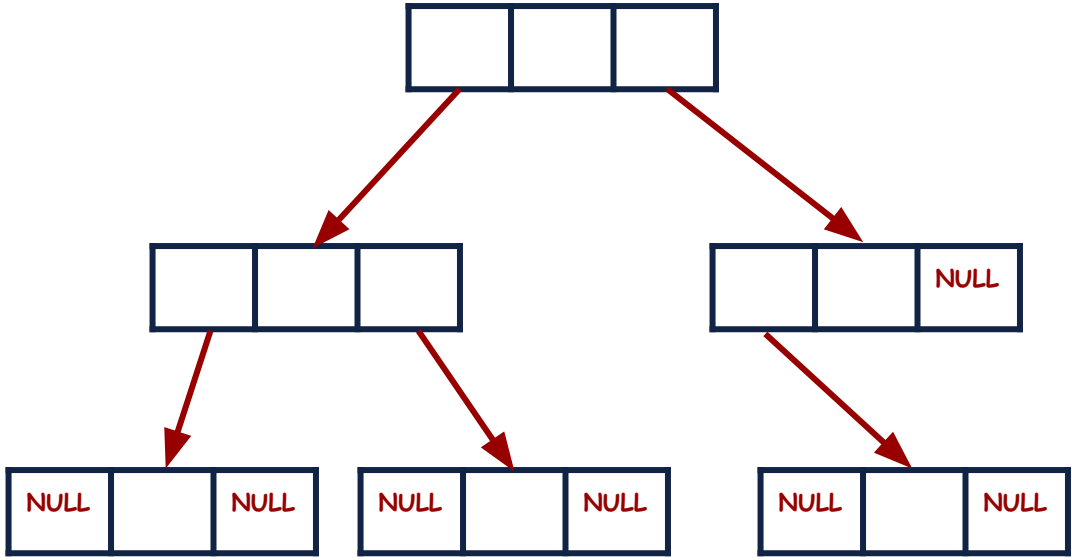
```
struct node
{
    string name;
    int age;
    char gender;
    node *left;
    node *right;
};
```



Binary Trees: View / Traversal

Should we first go to all the left children and then their right children?

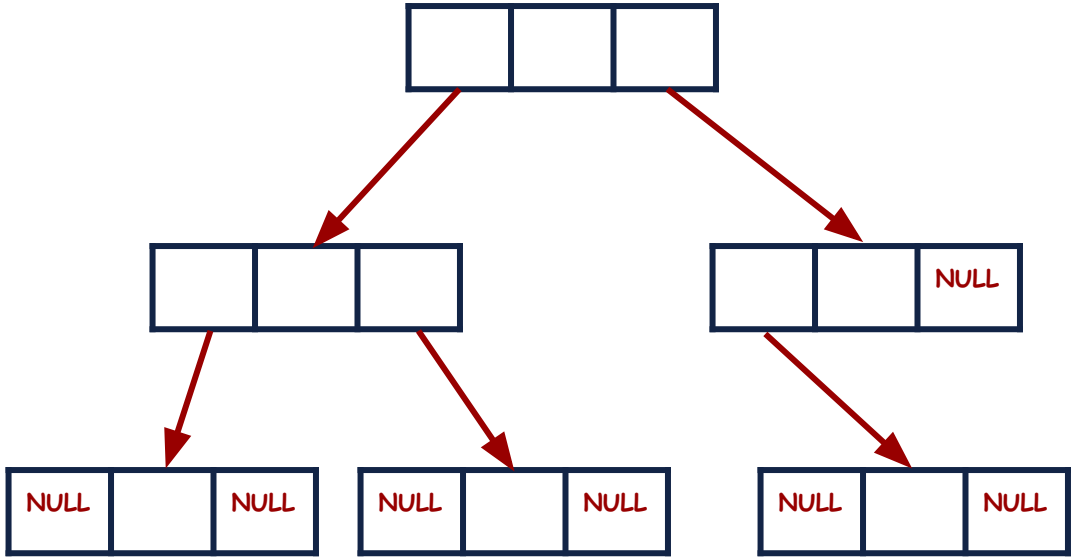
```
struct node
{
    string name;
    int age;
    char gender;
    node *left;
    node *right;
};
```



Binary Trees: View / Traversal

There are **multiple options** in which we can traverse the binary tree as it is a non-linear data structure.

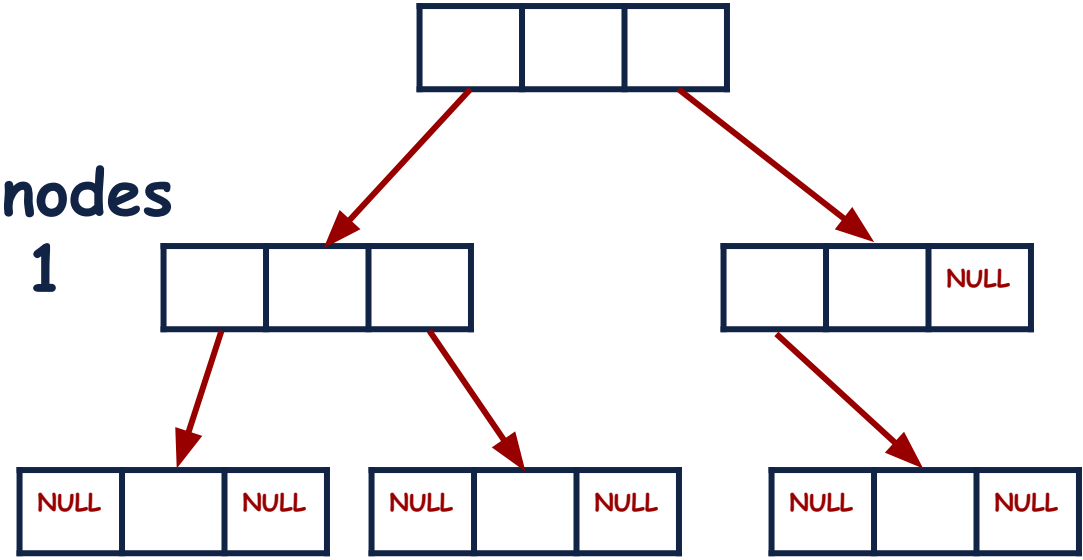
```
struct node
{
    string name;
    int age;
    char gender;
    node *left;
    node *right;
};
```



Binary Trees: View / Traversal

There are multiple options in which we can traverse the binary tree as it is a non-linear data structure.

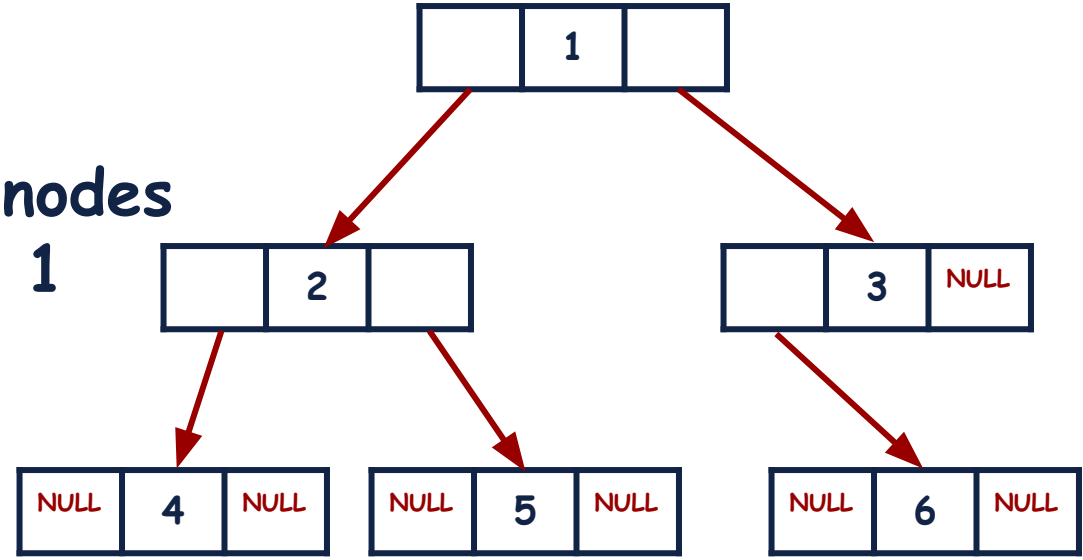
Lets first print all the nodes of depth 0, then depth 1 from left to right, and so on.



Binary Trees: View / Traversal

For simplicity we are using numbers instead of names.

Lets first print all the nodes of depth 0, then depth 1 from left to right, and so on.



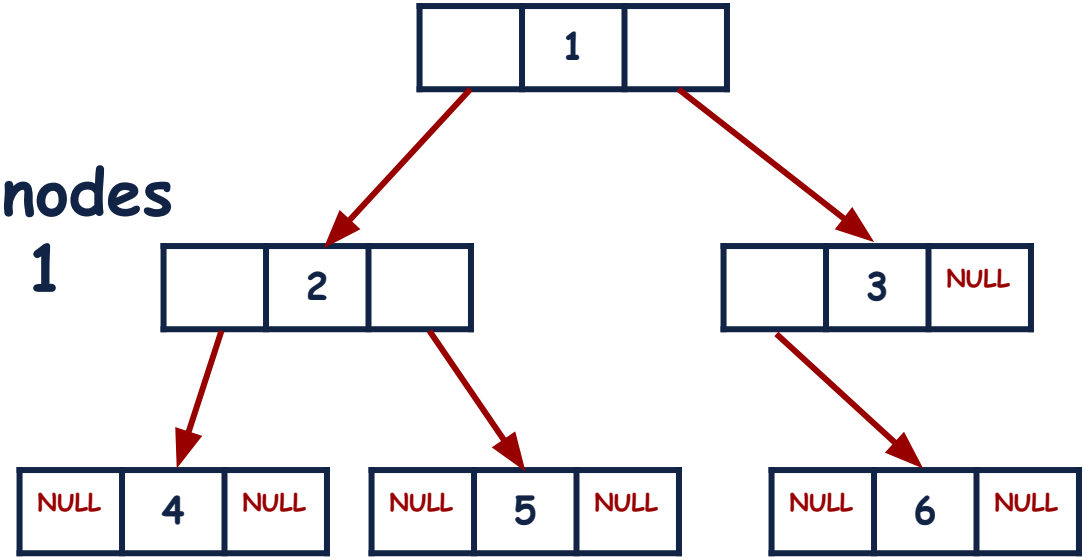
Binary Trees: View / Traversal

For simplicity we are using numbers instead of names.

Lets first print all the nodes of depth 0, then depth 1 from left to right, and so on.

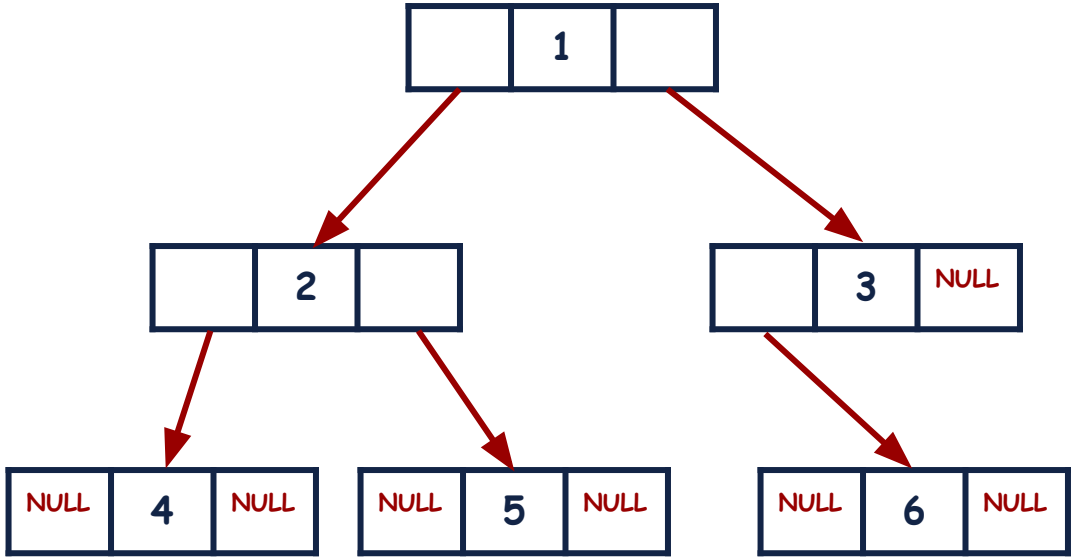
Output:

1, 2, 3, 4, 5 ,6



Binary Trees: View / Traversal

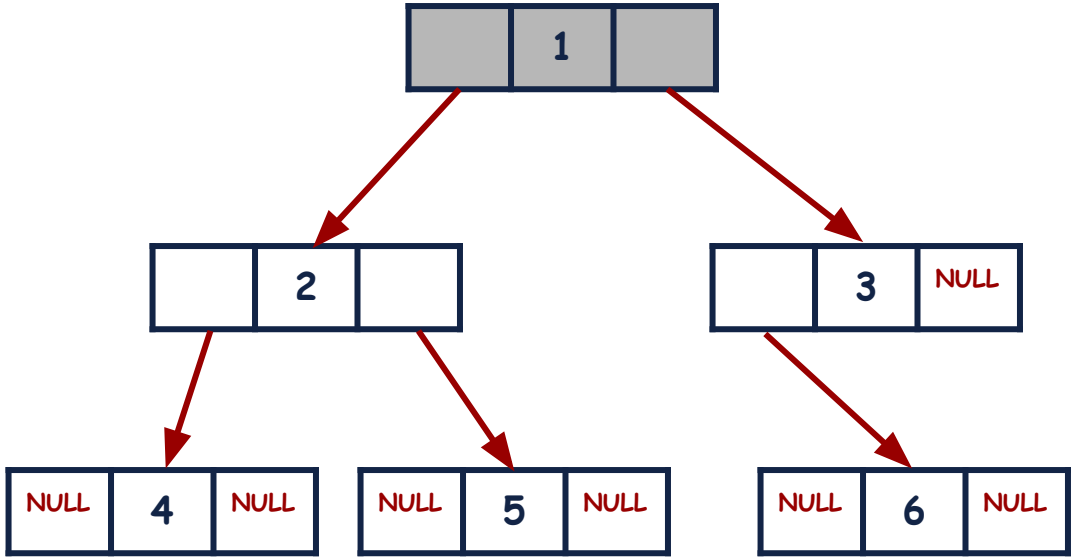
Let's push the root node in the Queue.



Output:

Binary Trees: View / Traversal

Pop the node from the queue and print its data.

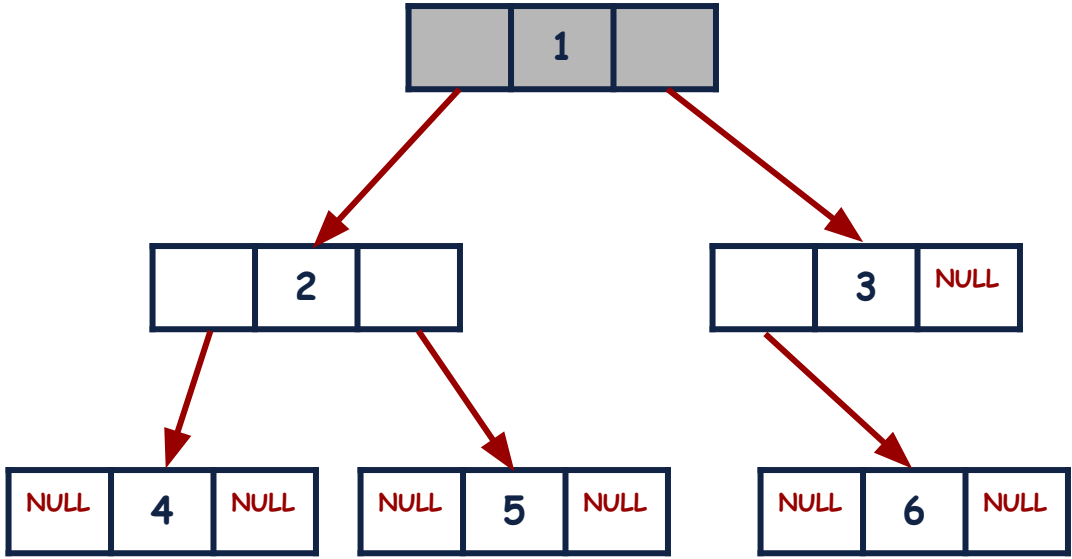


Output:

1,

Binary Trees: View / Traversal

Push the left and right node of the current node in the Queue if they are not NULL.

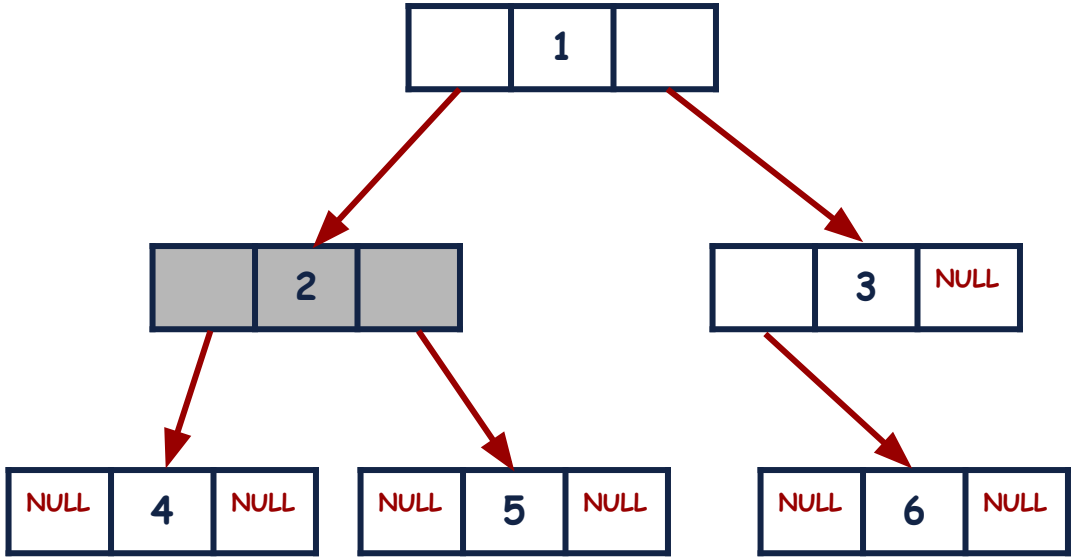


Output:

1,

Binary Trees: View / Traversal

Pop the element from the queue and print its data.

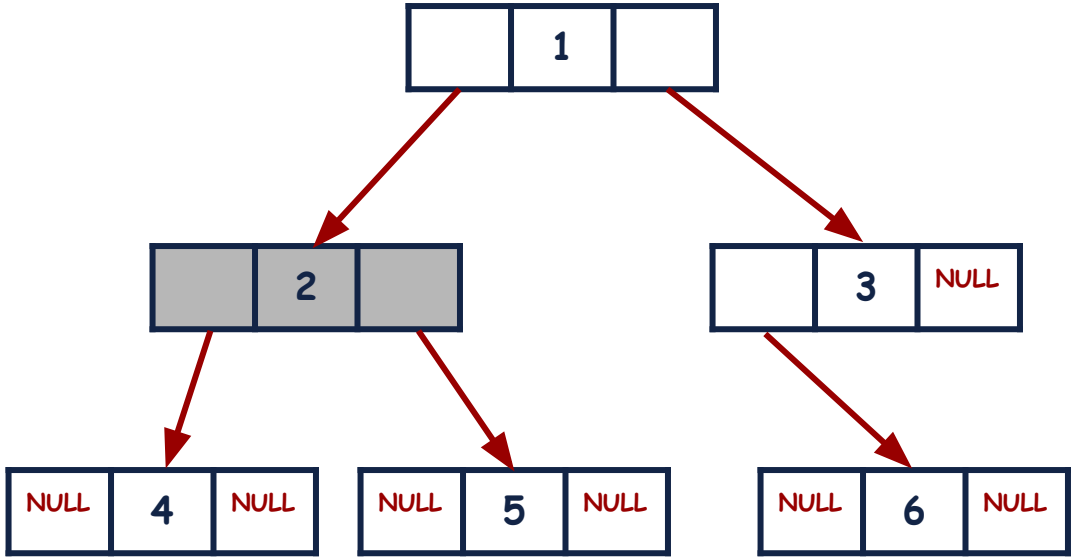


Output:

1, 2,

Binary Trees: View / Traversal

Push the left and right node of the current node in the Queue if they are not NULL.

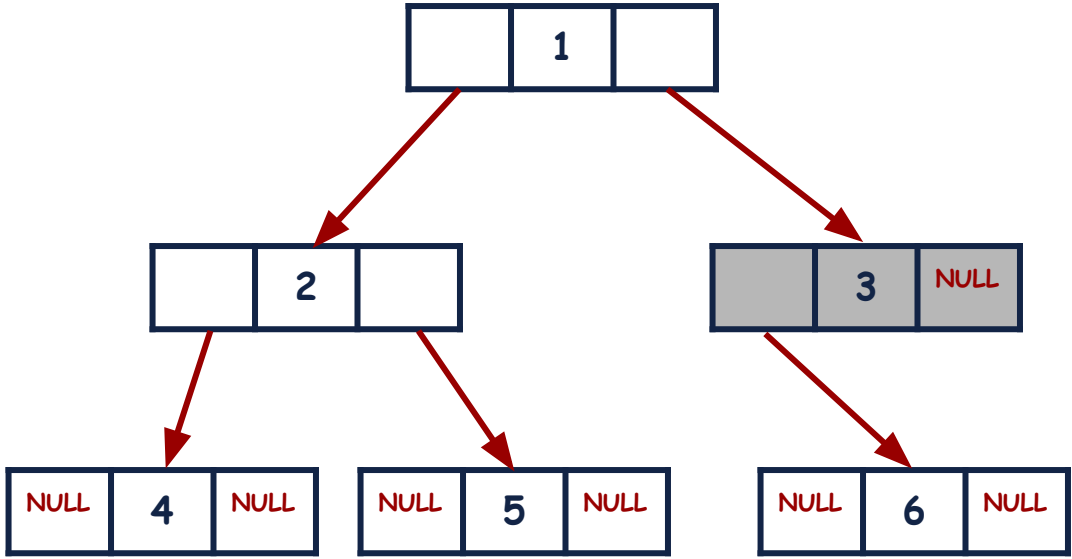


Output:

1, 2,

Binary Trees: View / Traversal

Pop the element from the queue and print its data.

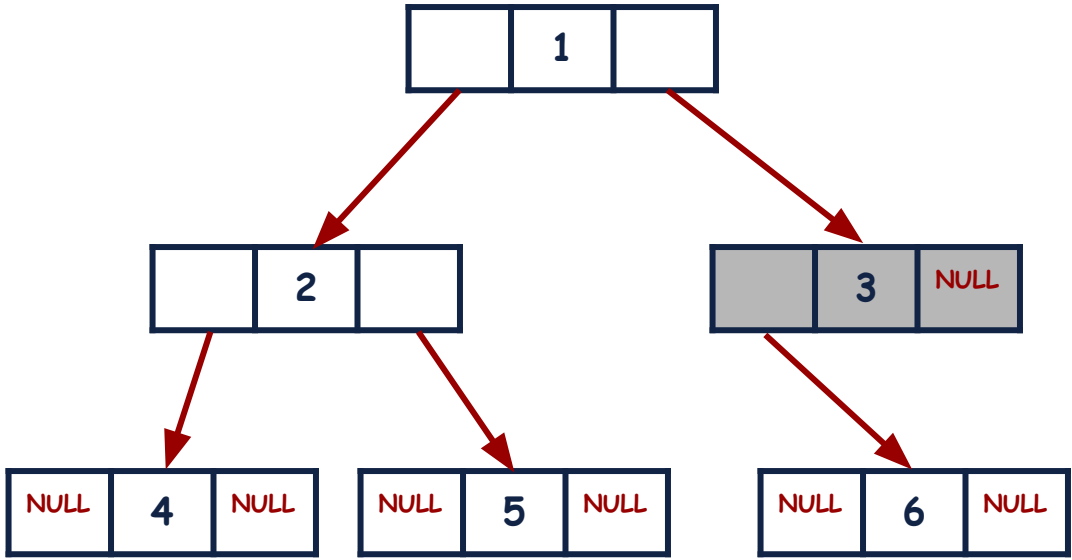


Output:

1, 2, 3,

Binary Trees: View / Traversal

Push the left and right node of the current node in the Queue if they are not NULL.

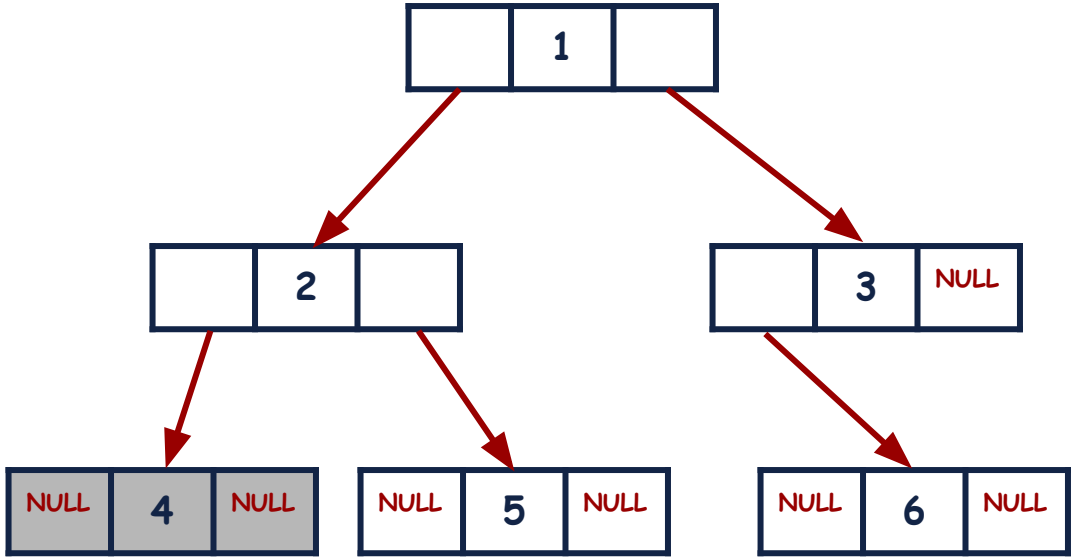


Output:

1, 2, 3,

Binary Trees: View / Traversal

Pop the element from the queue and print its data.

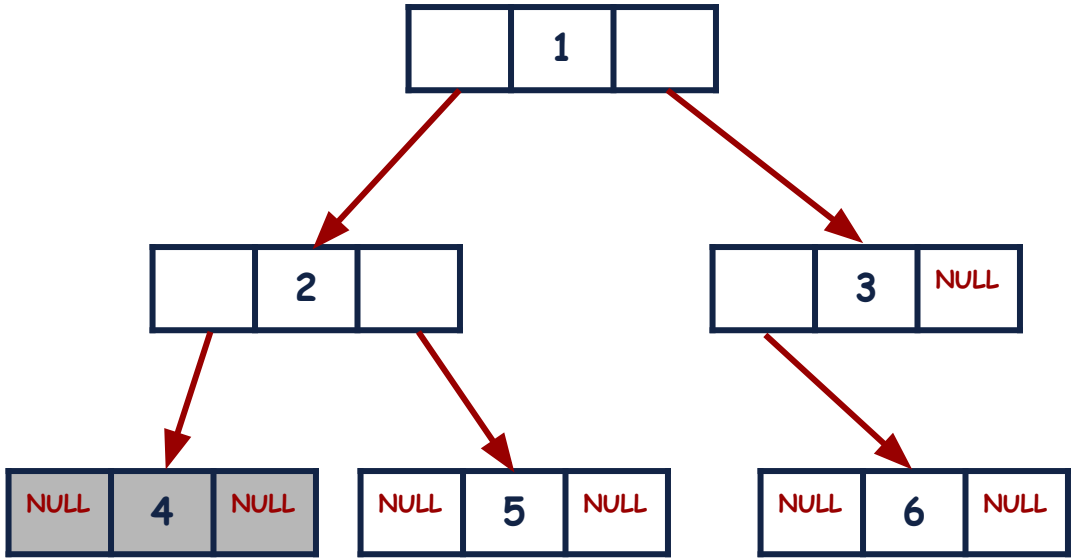


Output:

1, 2, 3, 4,

Binary Trees: View / Traversal

Push the left and right node of the current node in the Queue if they are not NULL.

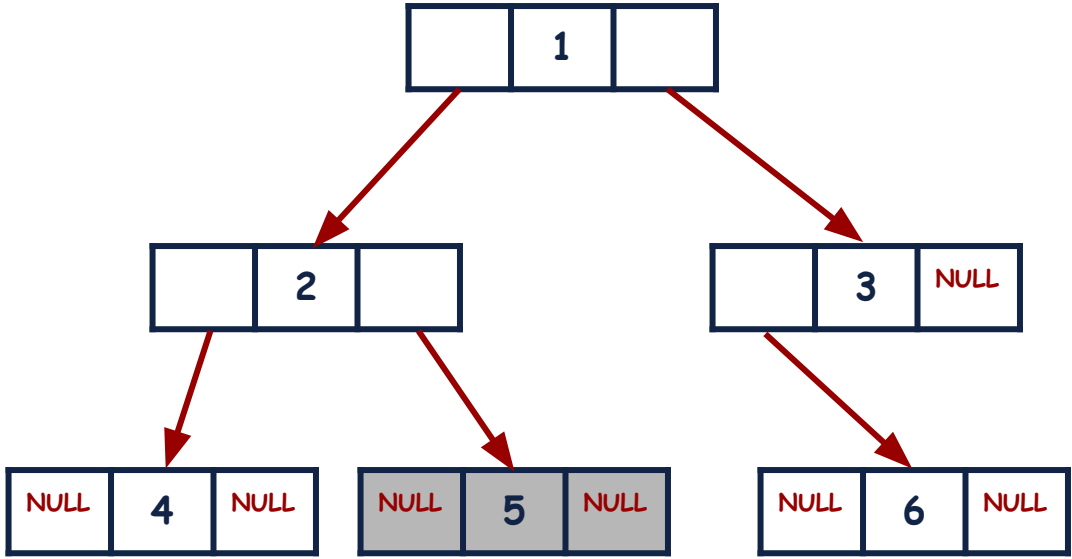


Output:

1, 2, 3, 4,

Binary Trees: View / Traversal

Pop the element from the queue and print its data.

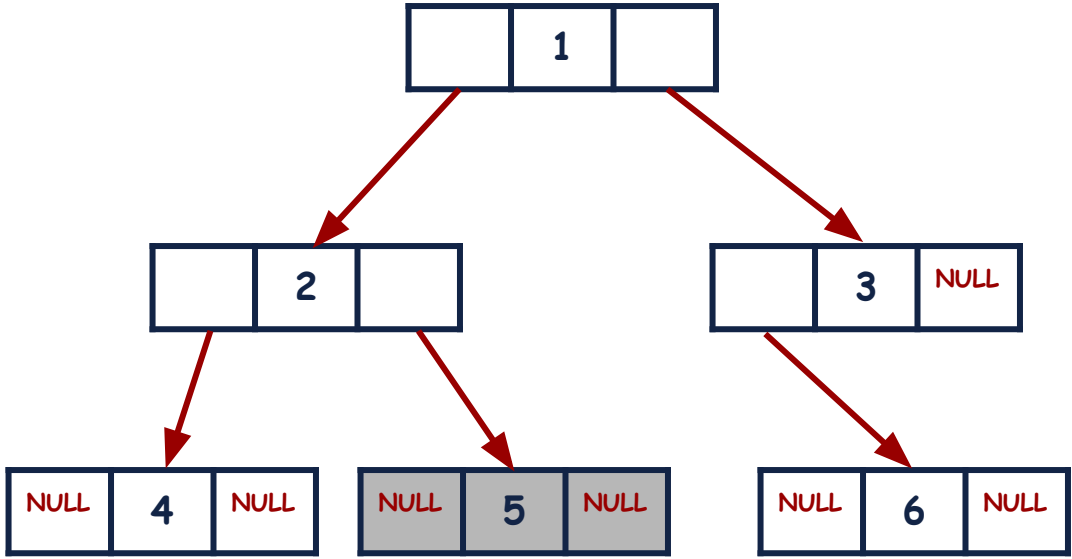


Output:

1, 2, 3, 4, 5,

Binary Trees: View / Traversal

Push the left and right node of the current node in the Queue if they are not NULL.

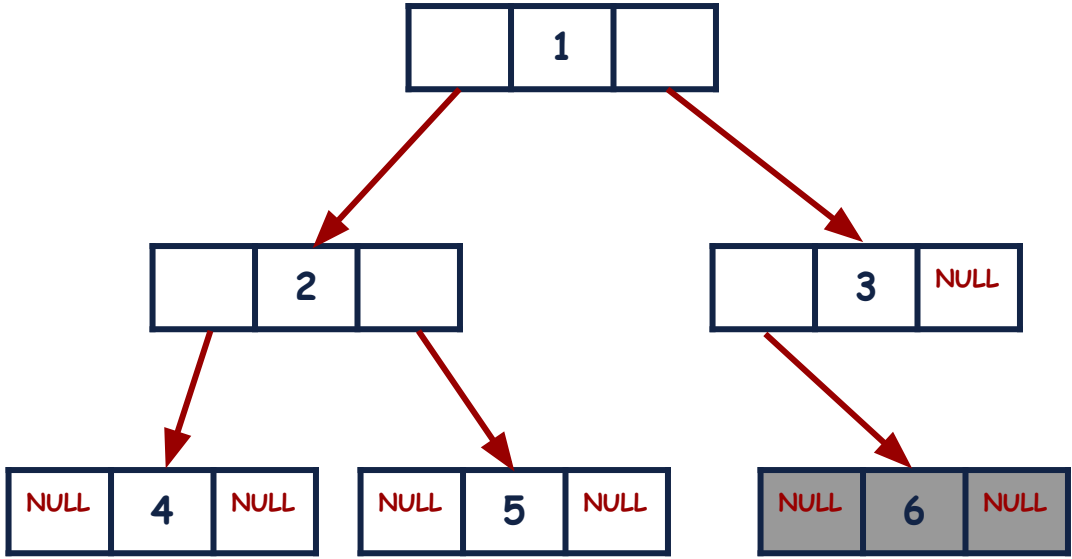


Output:

1, 2, 3, 4, 5,

Binary Trees: View / Traversal

Pop the element from the queue and print its data.

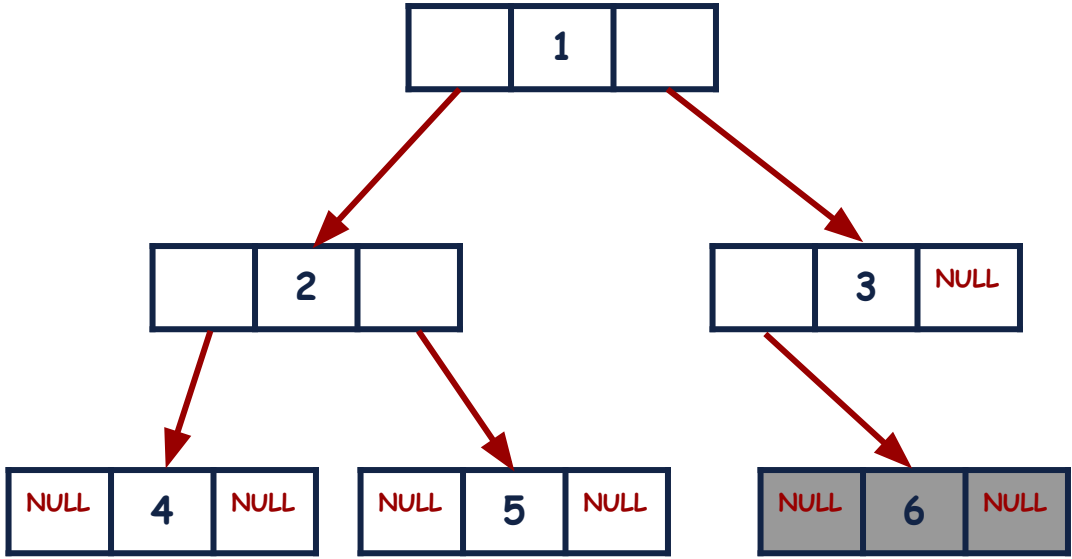


Output:

1, 2, 3, 4, 5, 6

Binary Trees: View / Traversal

Push the left and right node of the current node in the Queue if they are not NULL.

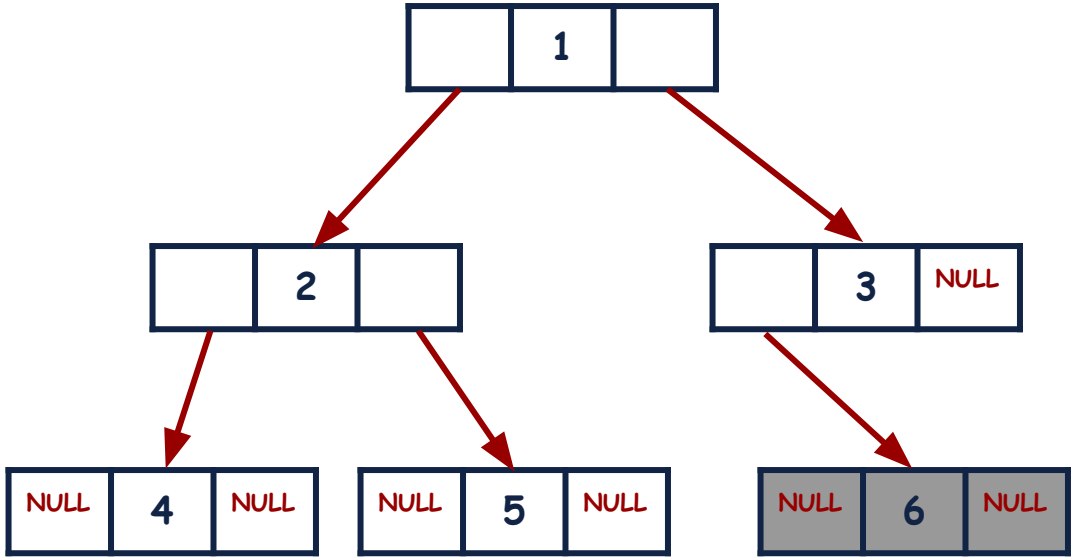


Output:

1, 2, 3, 4, 5, 6

Binary Trees: View / Traversal

Stop if the Queue is empty.



Output:

1, 2, 3, 4, 5, 6

Binary Trees: Traversal (Level Order)

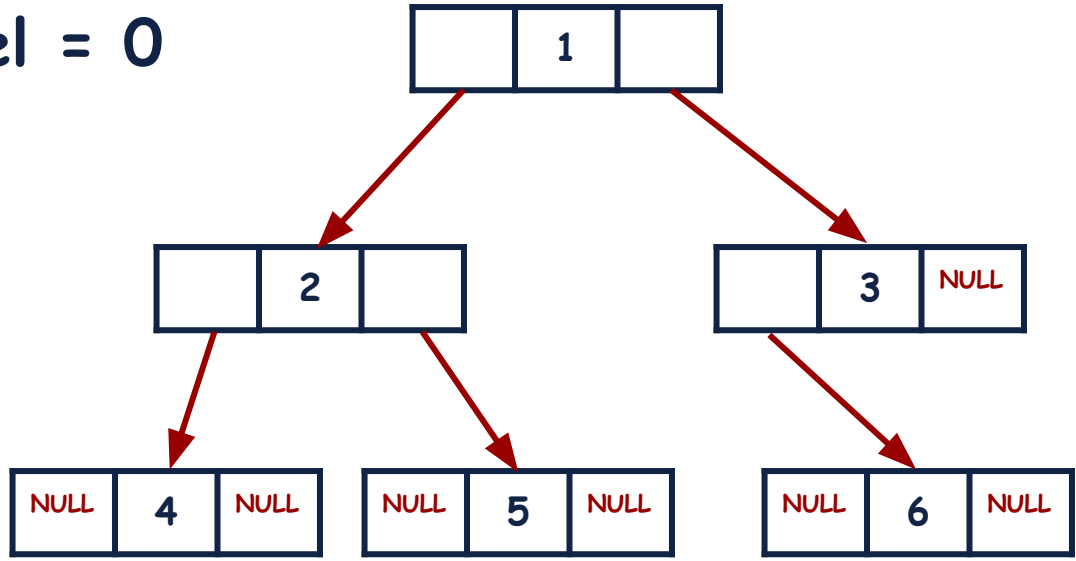
This way of traversing is called **Level Order Traversing.**

Or => Breadth First Traversal

Level = 0

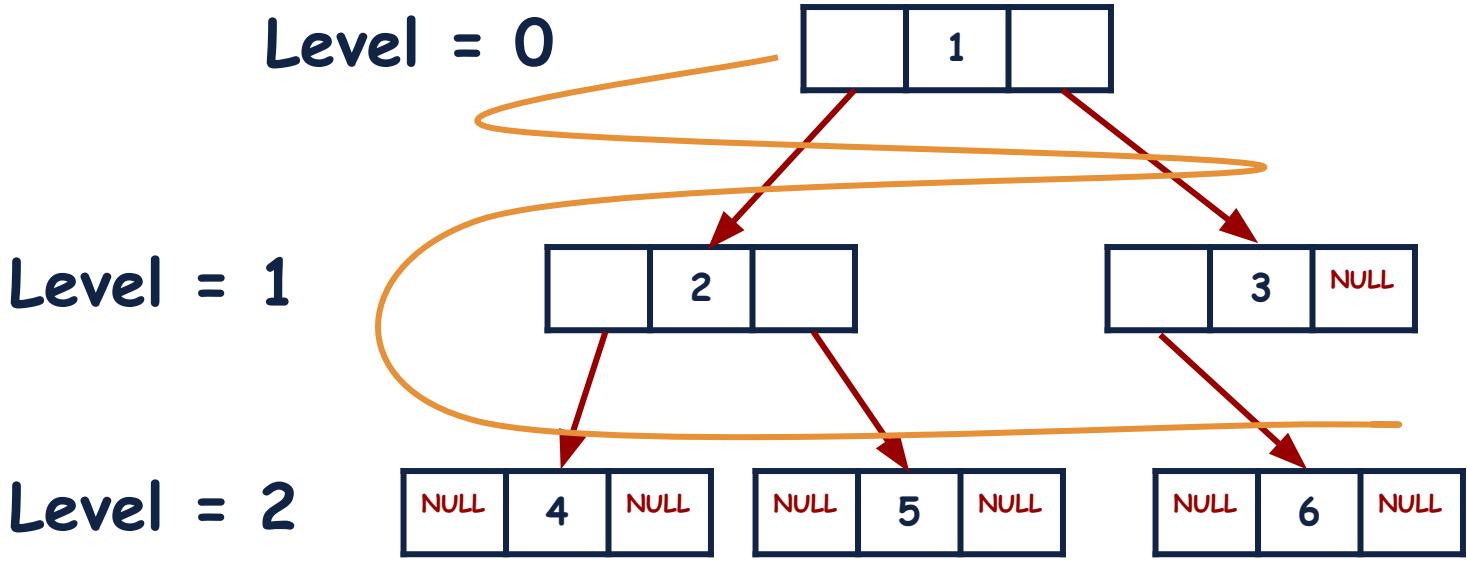
Level = 1

Level = 2



Binary Trees: Traversal (Breadth First)

This way of traversing is also known as **Breadth First Search**



Binary Trees: Traversal (Breadth First)

Pseudocode:

1. Declare the Queue
2. Enqueue the root node
3. while(Queue is not empty)
    a. Dequeue the node
    b. Print the value
    c. if(left node is not NULL)
        i. Enqueue the left node
    d. if(right node is not NULL)
        i. Enqueue the right node

Binary Trees: Traversal (Breadth First)

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

```
class binaryTree
{
    node *root;
public:
    node *addNode(int item)
    {
        node *record = new node();
        record->data = item;
        record->left = NULL;
        record->right = NULL;
        return record;
    }
```

```
void generateData()
{
     root = addNode(1);
     root->left = addNode(2);
     root->right = addNode(3);
     root->left->left = addNode(4);
     root->left->right = addNode(5);
     root->right->left = addNode(6);
}
```

Binary Trees: Traversal (Breadth First)

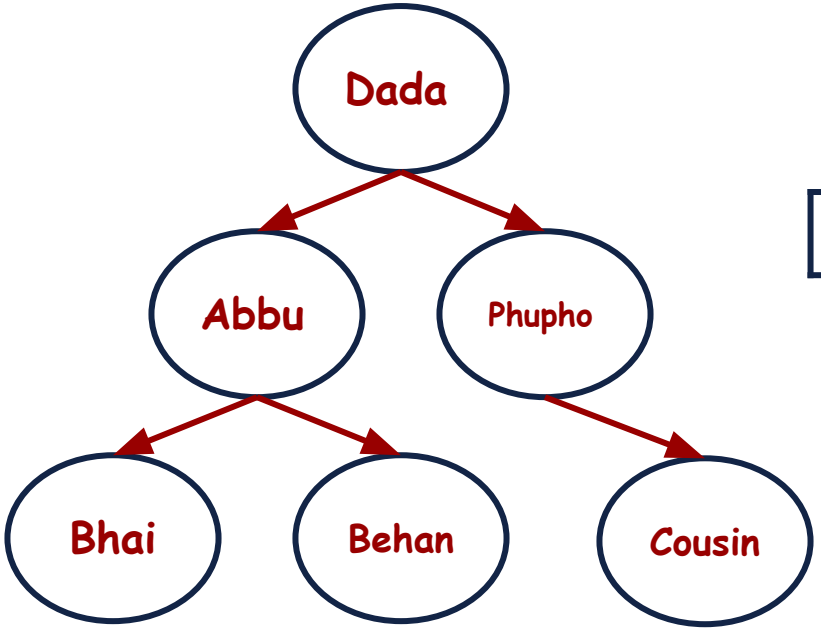
```
void bfs()
    {
        queue<node *> q;
        q.push(root);
        while (!q.empty())
        {
            node *temp = q.front();
            q.pop();
            cout << temp->data << ", ";
            if (temp->left != NULL)
                q.push(temp->left);
            if (temp->right != NULL)
                q.push(temp->right);
        }
    }

};
```

```
int main()
{
    binaryTree b;
    b.generateData();
    b.bfs();
}
```

Binary Trees: Implementation

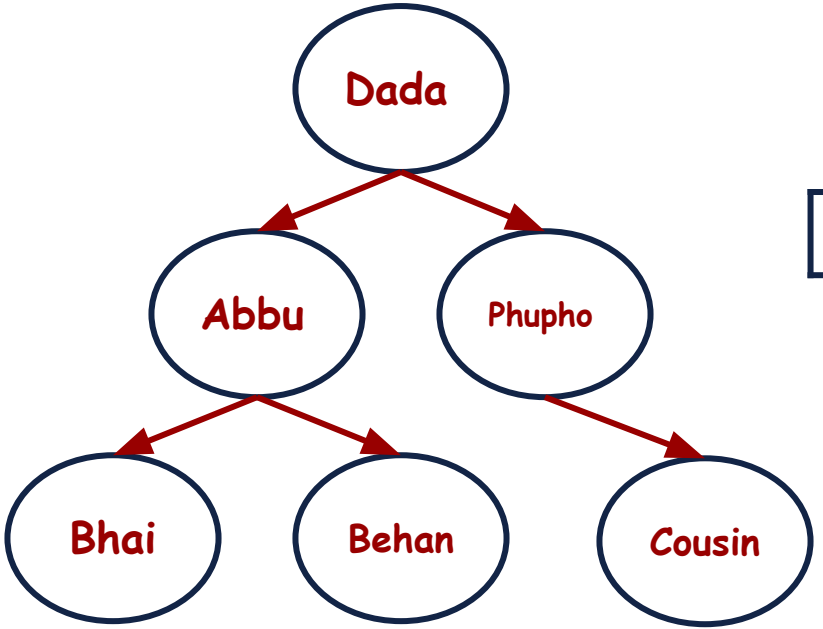
We can also use Arrays for storing the information.



0	1	2	3	4	5
Dada	Abbu	Phupho	Bhai	Behan	Cousin

Binary Trees: Implementation

We can also use Arrays for storing the information.

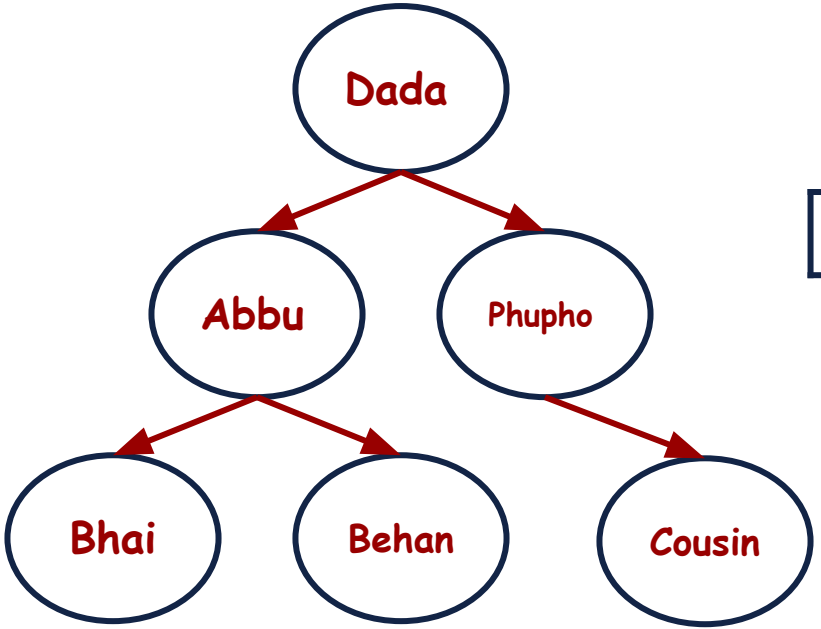


0	1	2	3	4	5
Dada	Abbu	Phupho	Bhai	Behan	Cousin

For node at index i

Binary Trees: Implementation

We can also use Arrays for storing the information.



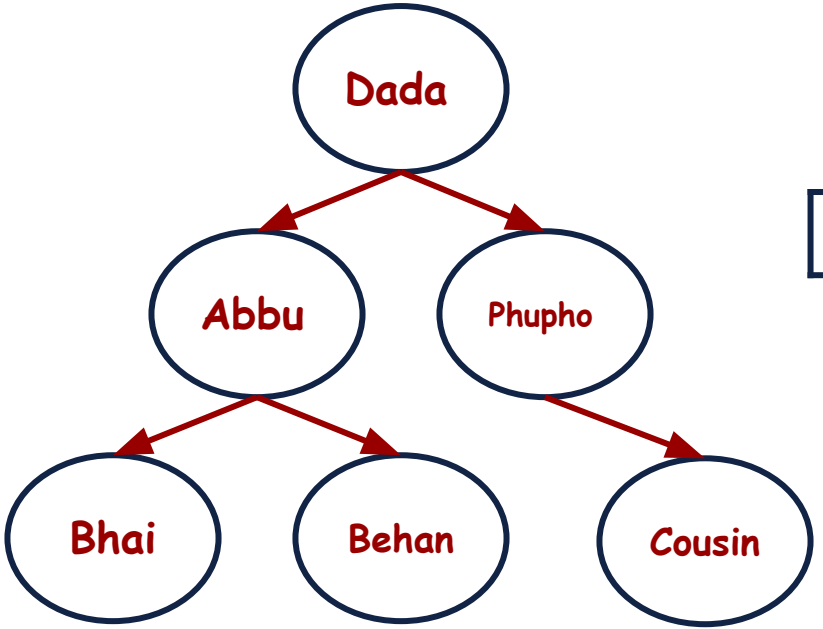
0	1	2	3	4	5
Dada	Abbu	Phupho	Bhai	Behan	Cousin

For node at index **i**

Left Child index: **2i + 1**

Binary Trees: Implementation

We can also use Arrays for storing the information.



0	1	2	3	4	5
Dada	Abbu	Phupho	Bhai	Behan	Cousin

For node at index i

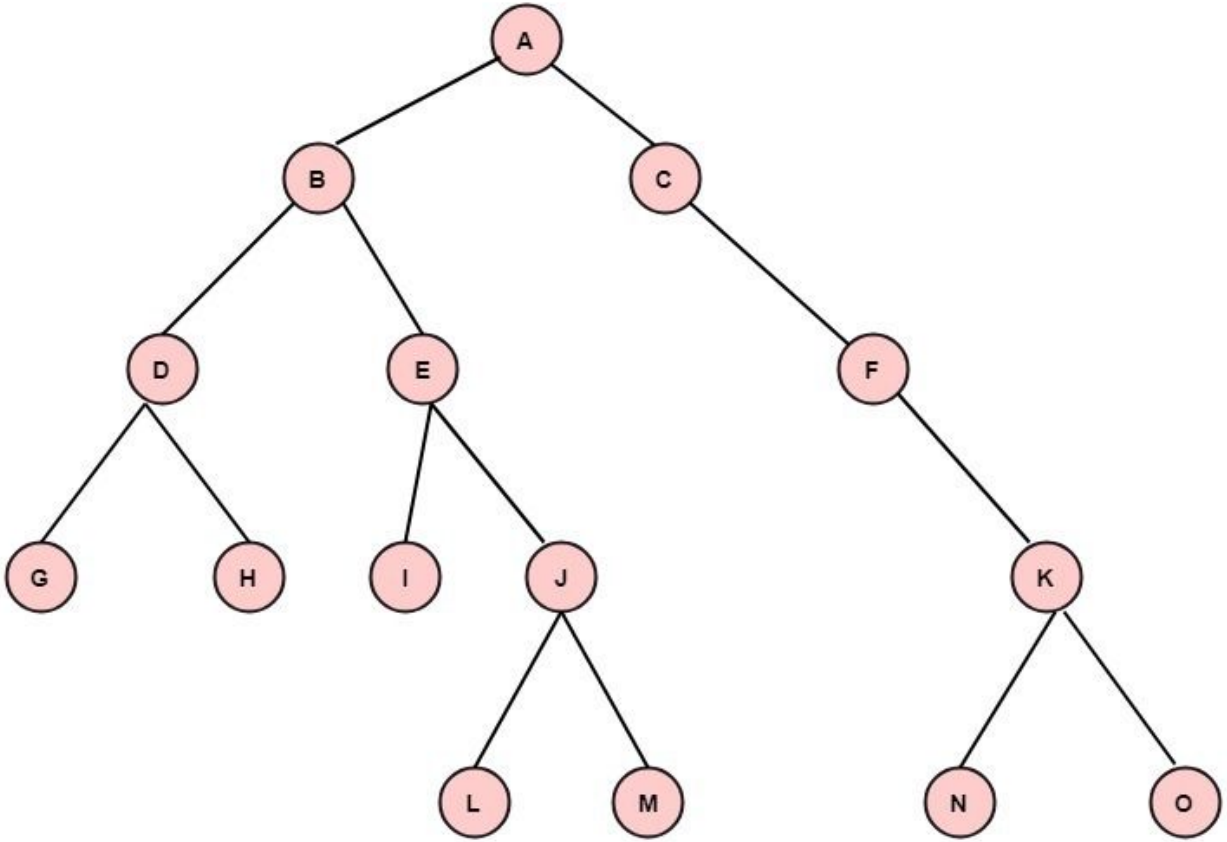
Left Child index: 2i + 1

Right Child index: 2i + 2

Learning Objective

Students should be able to traverse the binary Trees in Breadth First or Level Order to solve the problems efficiently.

Self Assessment



Self Assessment

From the previous figure, answer the following questions.

- Which node is the root?
- Which nodes are leaves?
- Name the parent node of each node.
- What is the Height of node K?
- What is the Depth of node D?
- Write the ancestors of node J.
- Write the descendants of node E.
- Pair up all the Siblings from the previous Tree.
- What is the Height of the Tree?
- Write the path to reach node L.
- Write the total number of sub trees.
- What is the Degree of node F?

Reading Assessment

1. https://www.geeksforgeeks.org/difference-between-general-tree-and-binary-tree/
2. https://www.upgrad.com/blog/5-types-of-binary-tree/
3. https://www.programiz.com/dsa/trees
4. https://www.softwaretestinghelp.com/binary-tree-in-cpp/#Binary_Tree_Traversal