Lecture 5 - Tree (Part 1)

CPE112 - Programming with Data Structures
1 March 2024

Dr. Piyanit Wepulanon & Dr. Taweechai Nuntawisuttiwong

Department of Computer Engineering
KMUTT

Tentative Schedule

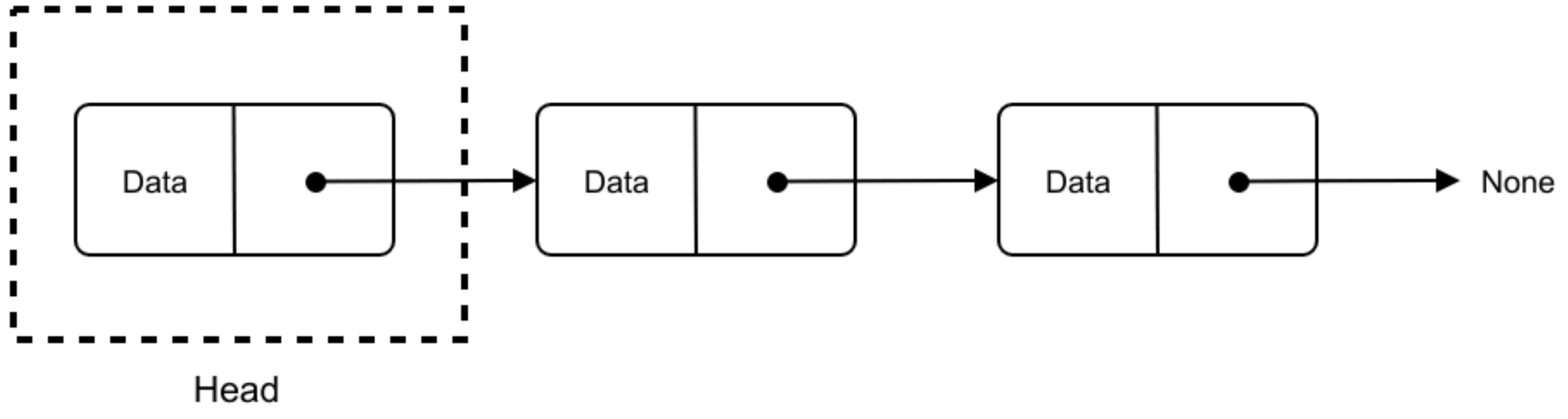
Date	Activity
1 Mar	Tree 1
8 Mar	Tree 2
15 Mar	Graph
22 Mar	Graph (Lab)
1 Apr	Exam
12 Apr	Holiday

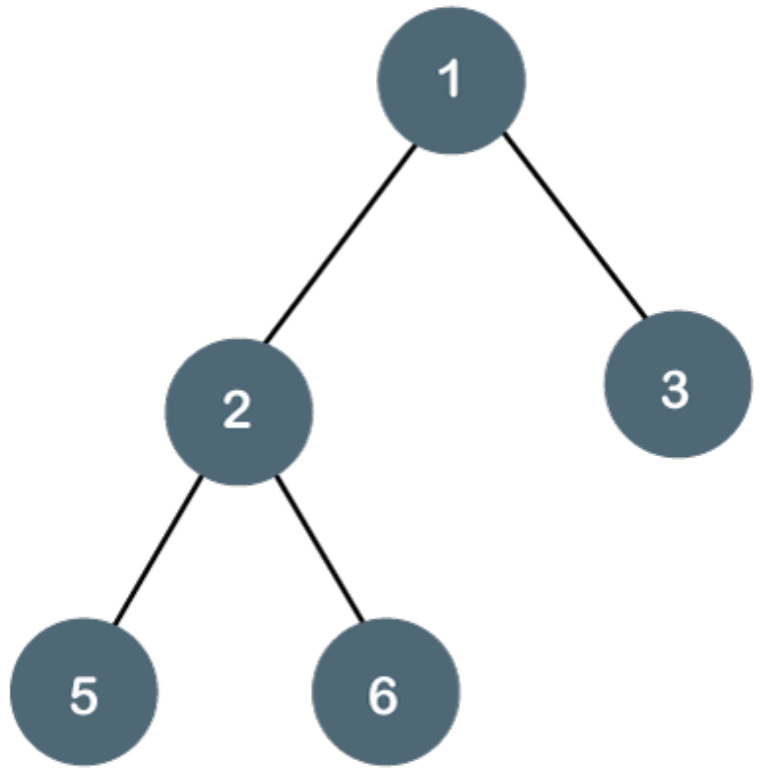
Date	Activity
19 Apr	Active Learning (Applications)
26 Apr	Term Project Topic
3 May	VDO Presentation
10 May	Coding Exam
17 May	Project Presentation

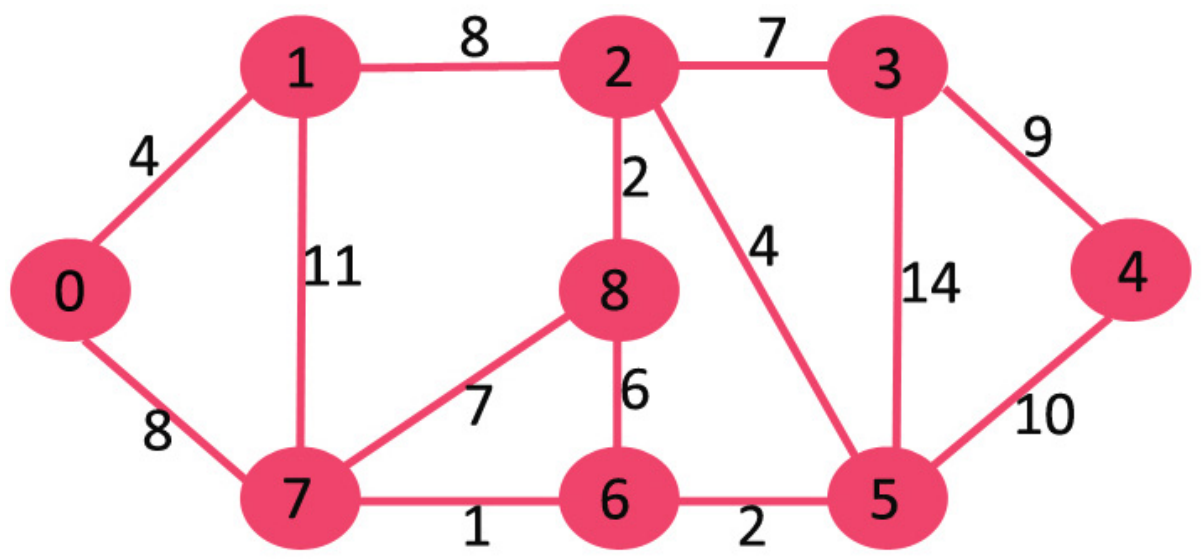
- **Linear data structure report submission: 10 March 2024**

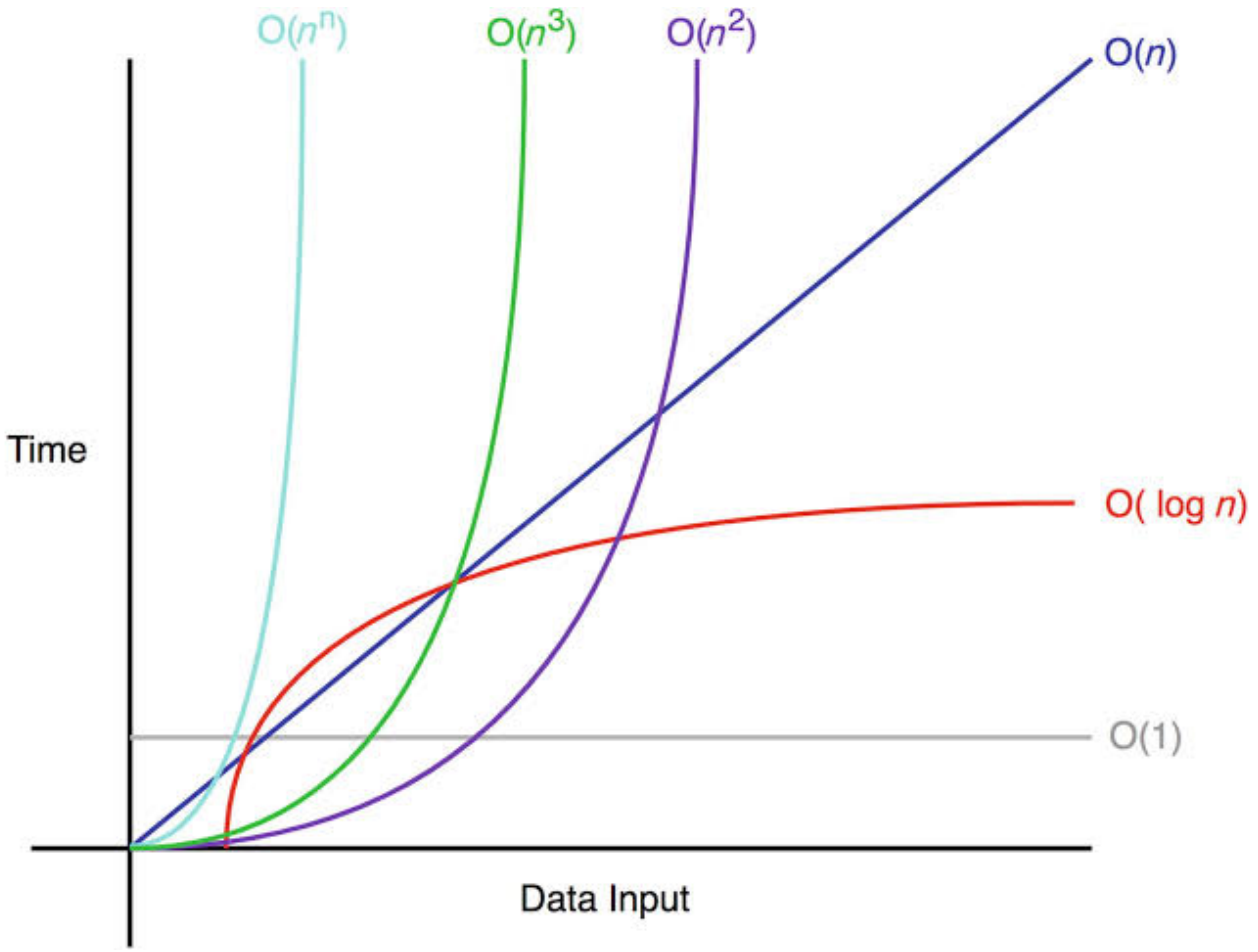
Review

- **Introduction to Programming with Data Structures**
  - Data structures, Algorithms, Efficiency



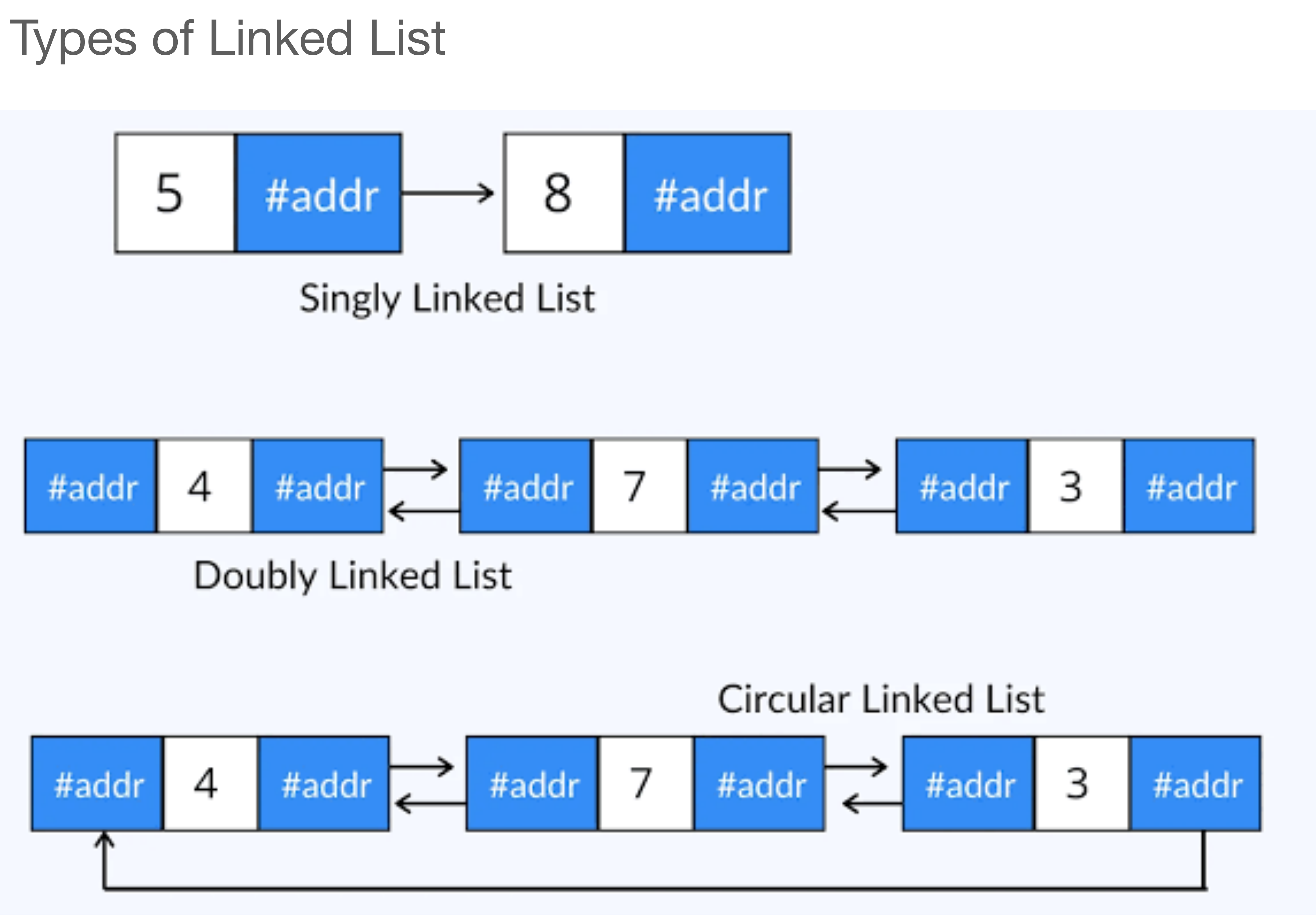


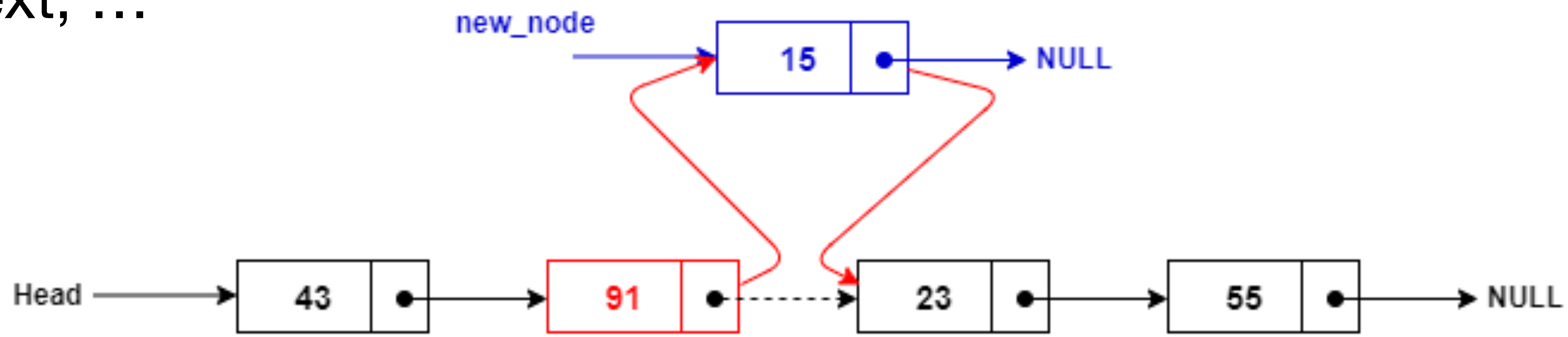




Review

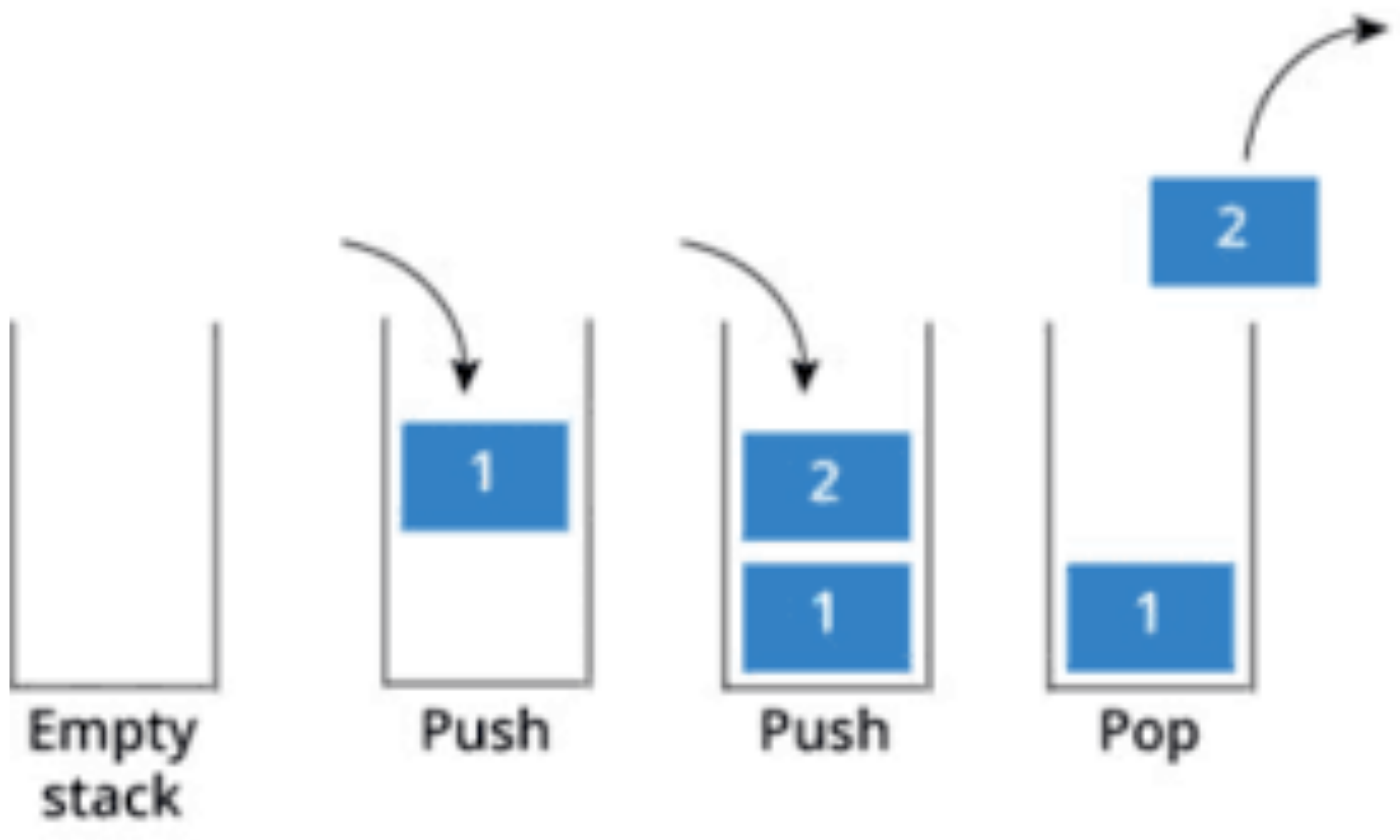
- **List (Array & Linked List)**
  - Pros & cons, capacity, accessibility, insert/remove elements
  - Structure
  - Main functions: Create, Destroy, Size, InsertAt, Append, RemoveAt, Get, Reset, GetNext, …

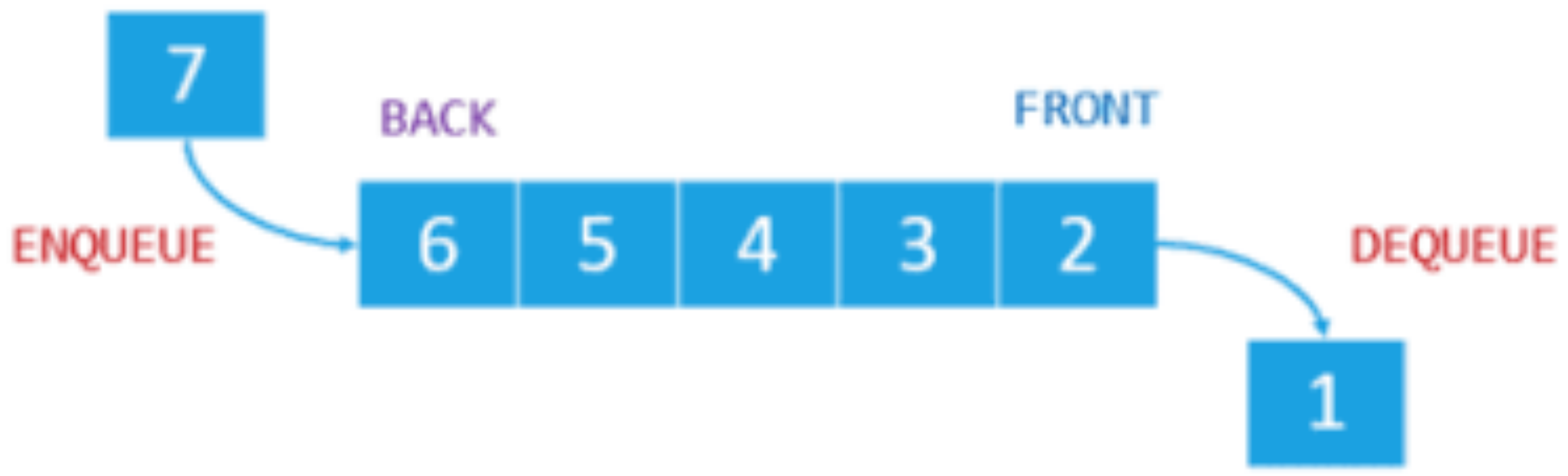




Review

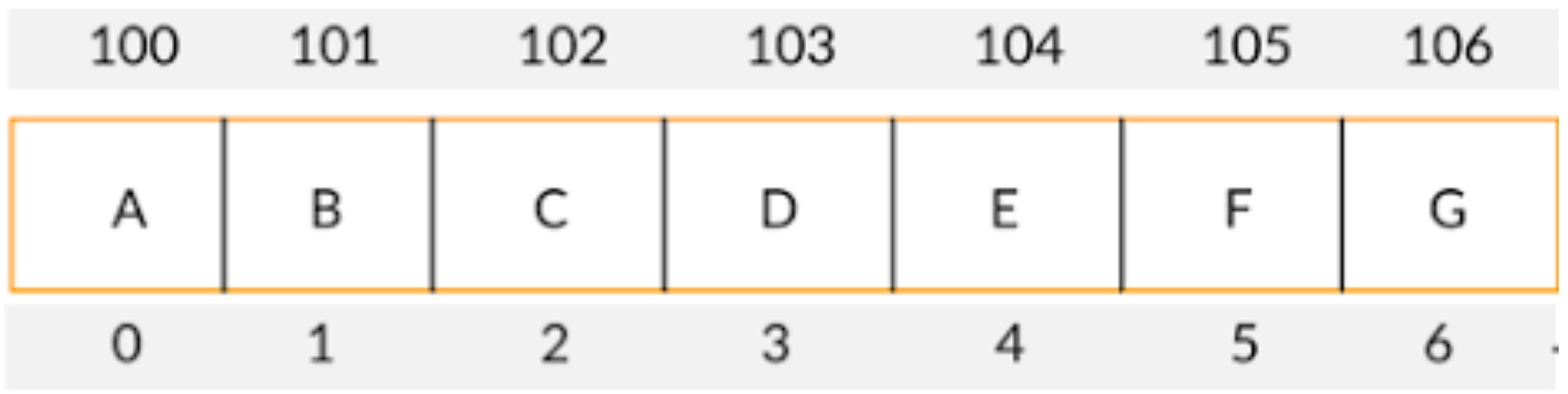
- **Stack & Queue**
  - Structure
  - Main functions (Create, Destroy, Size, Push, Pop, Enqueue, Dequeue)



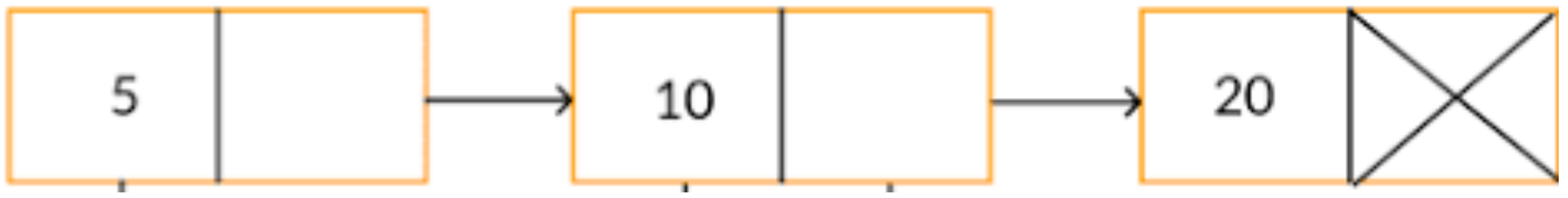


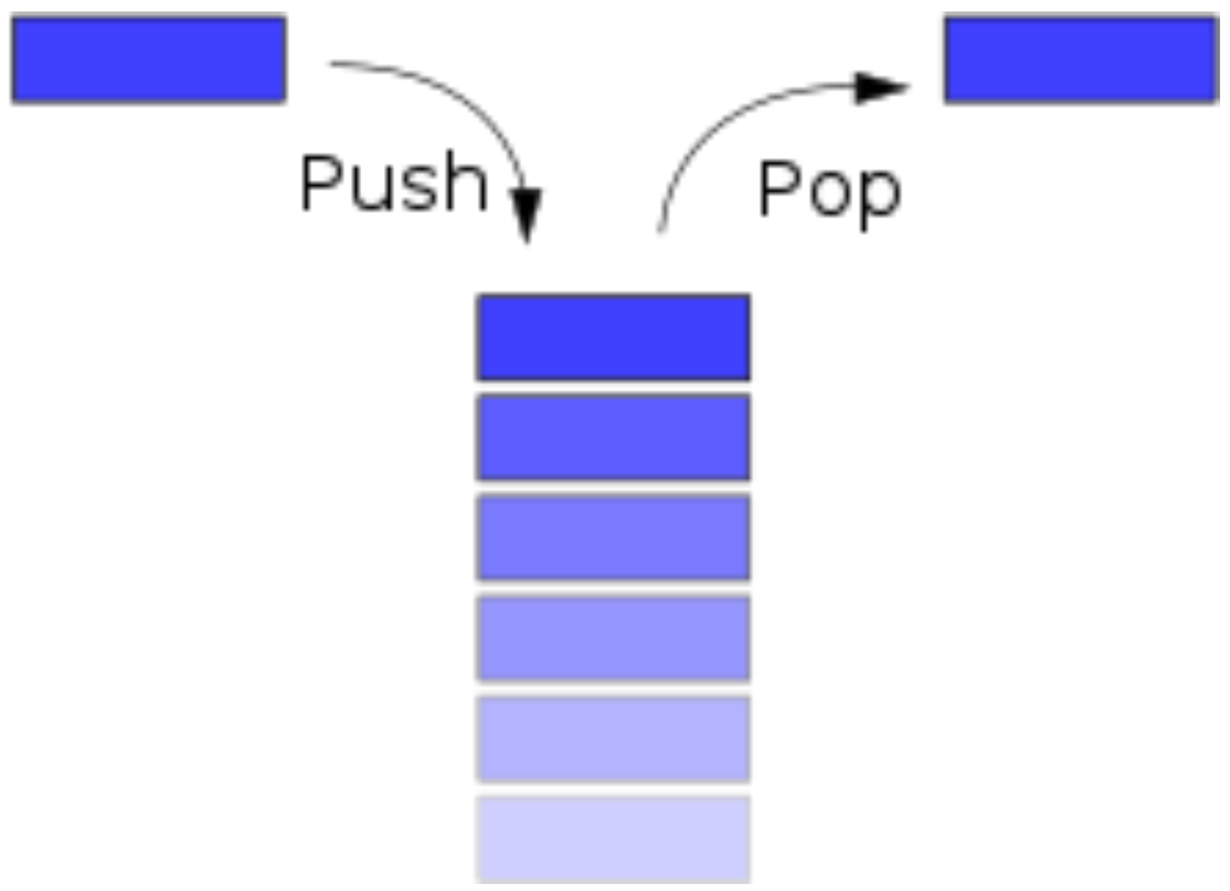
Linear Data Structure

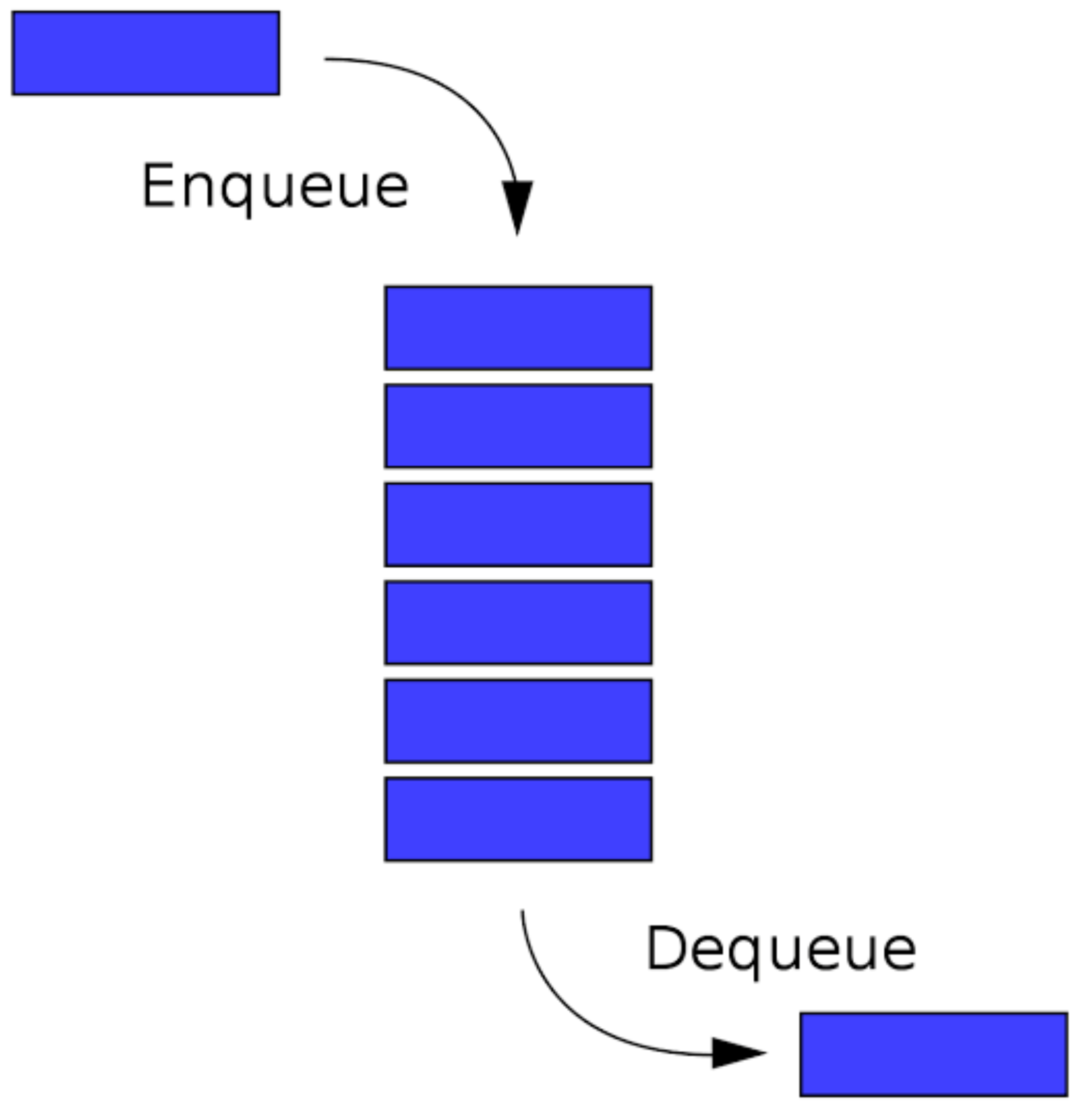
Data structure



- Sequentially arrange
- Start/End of structure
- Previous/Next element



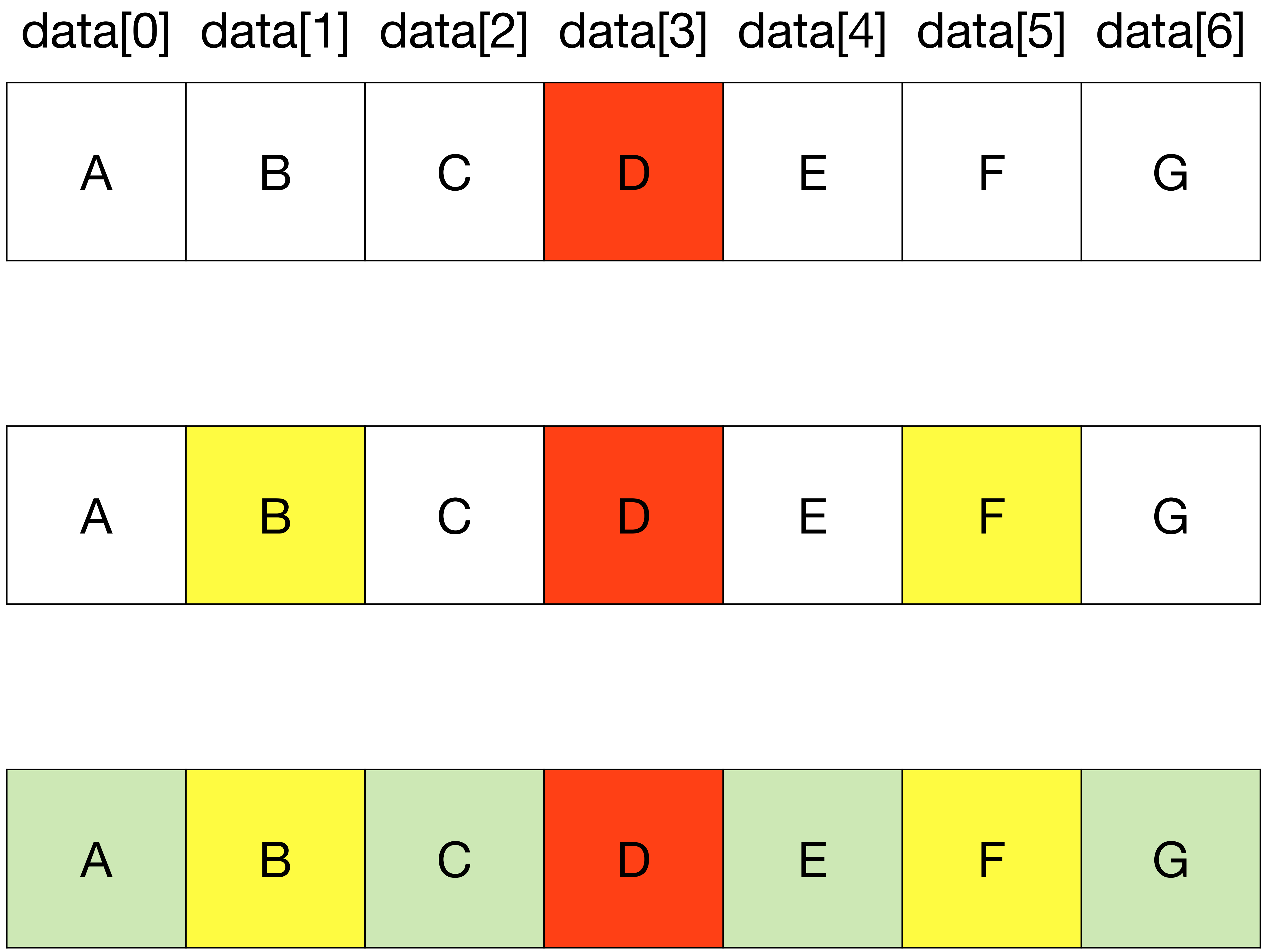




Linear Data Structure

Data structure

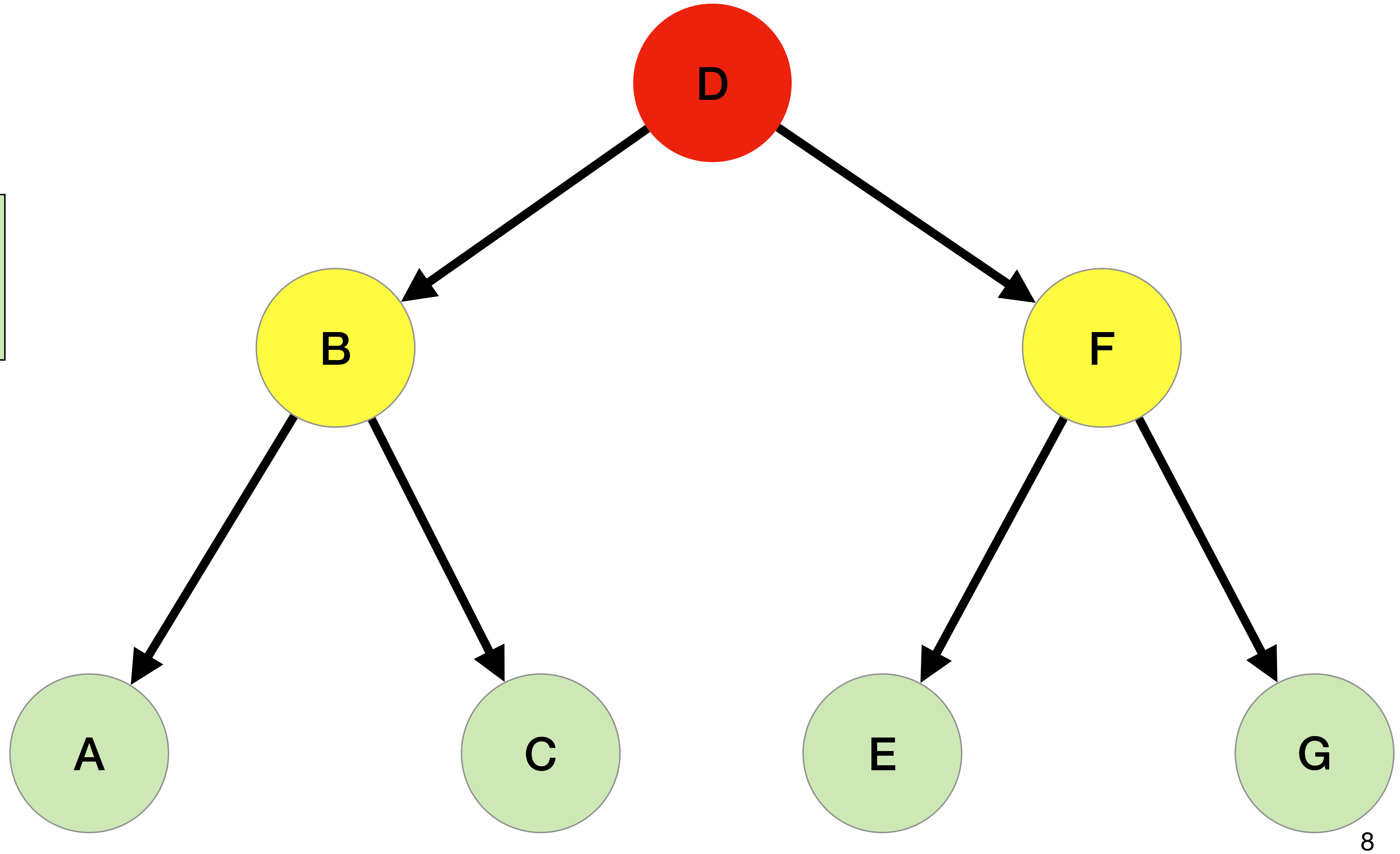
- **Example binary search**



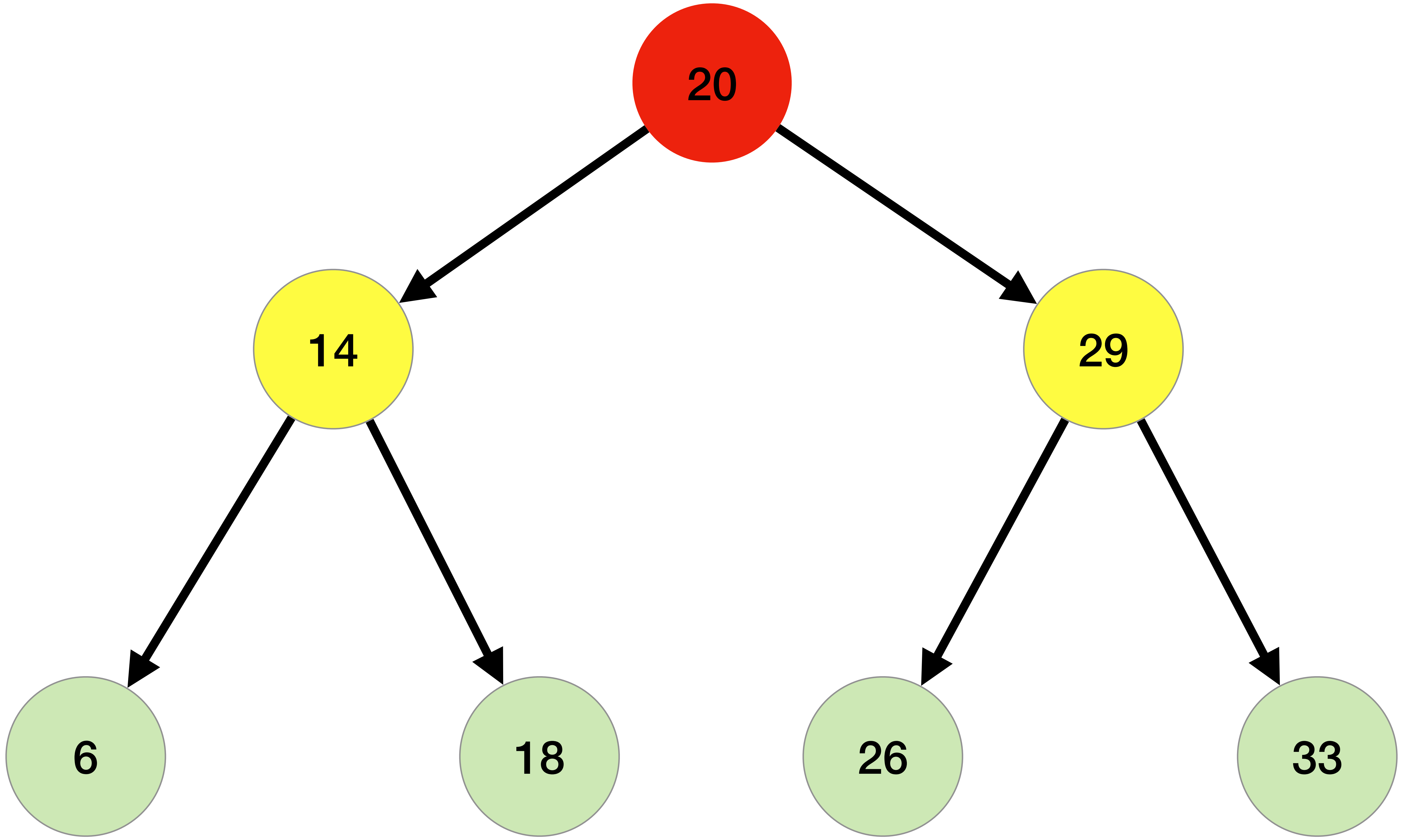
Hierarchical Structure

- **Example binary search**

data[0]	data[1]	data[2]	data[3]	data[4]	data[5]	data[6]
A	B	C	D	E	F	G

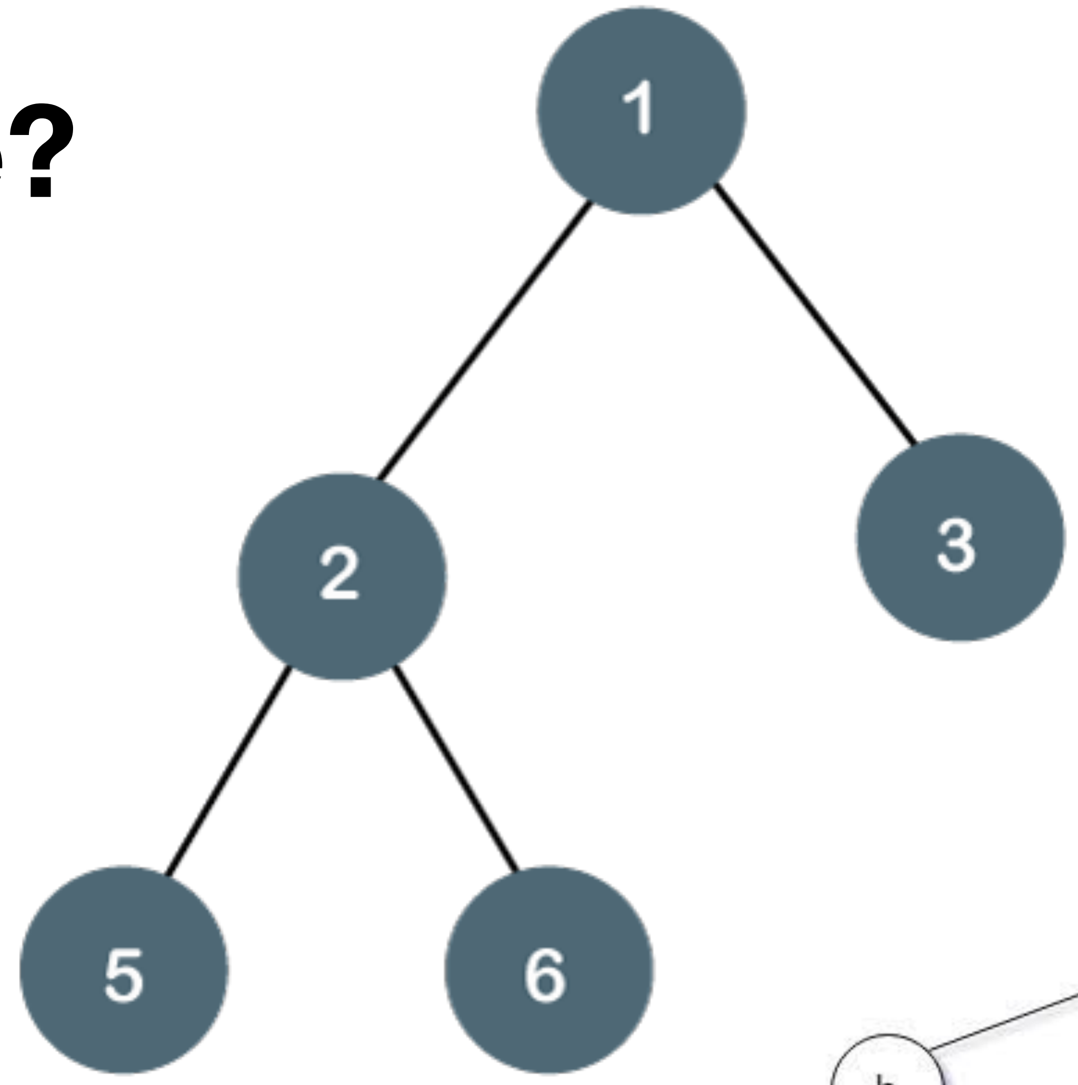


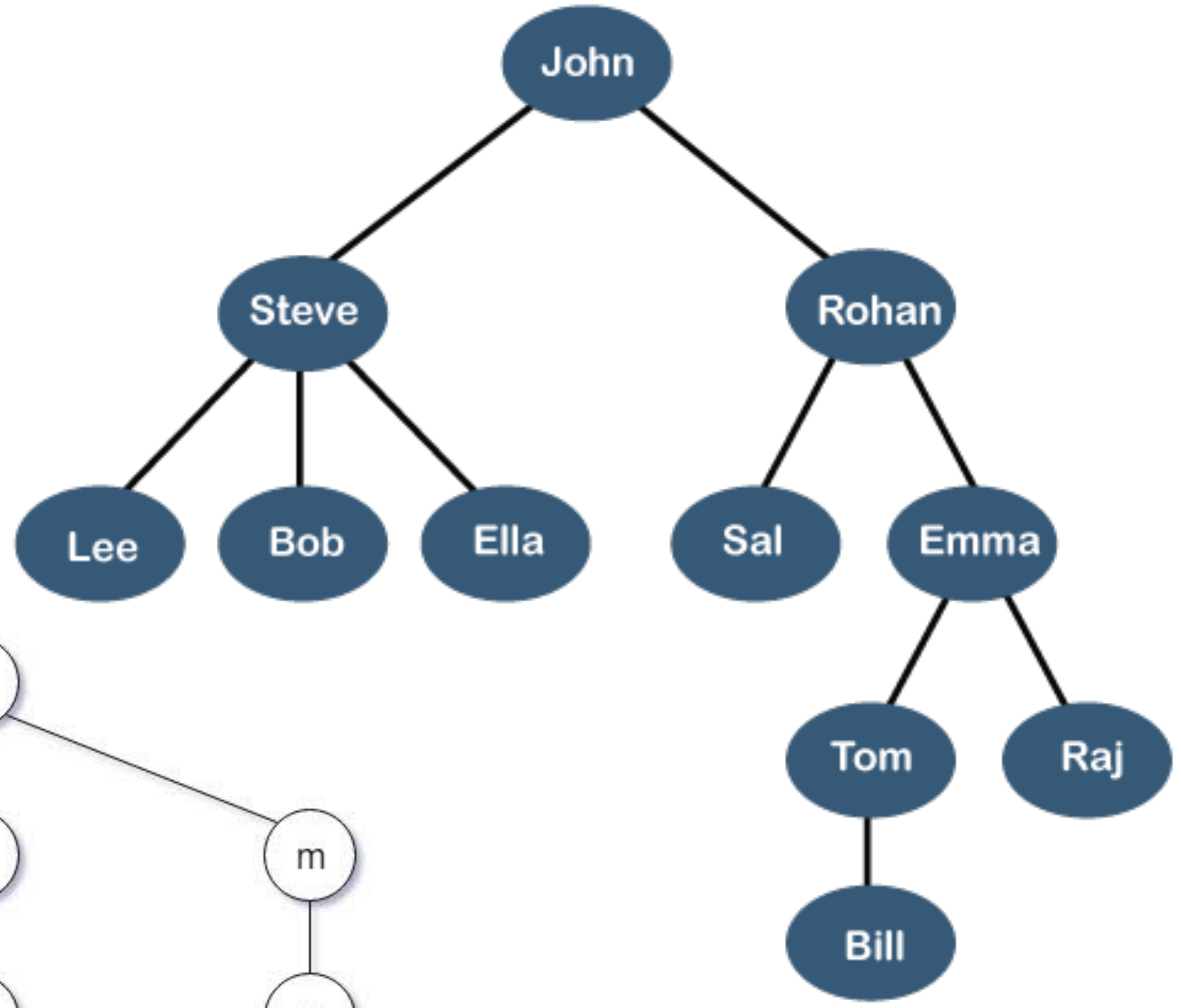
Hierarchical Structure

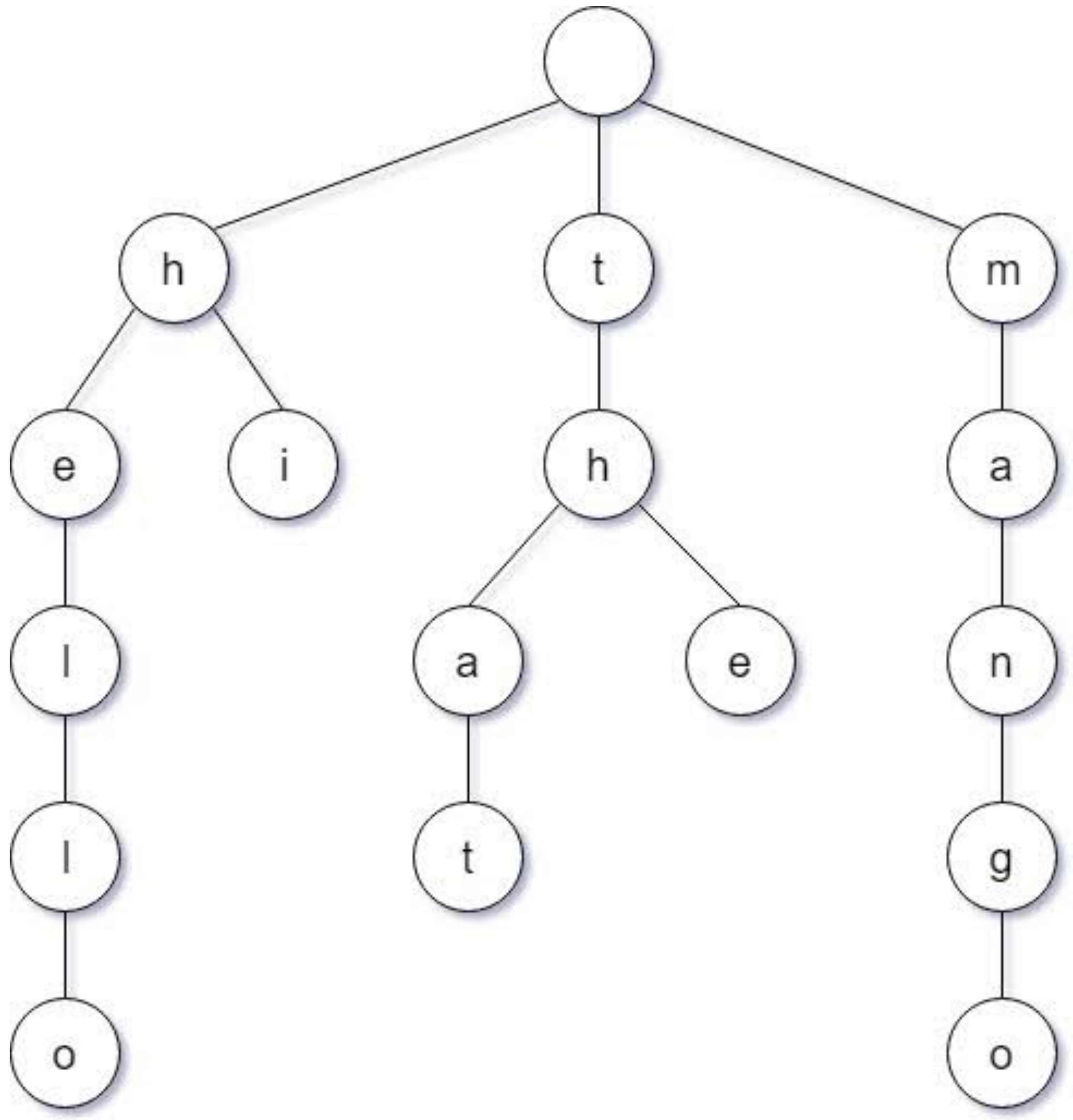


Today's Goal

- What is "tree" data structure?
- Terminologies
- Binary tree
- Tree traversal
- Lab practice





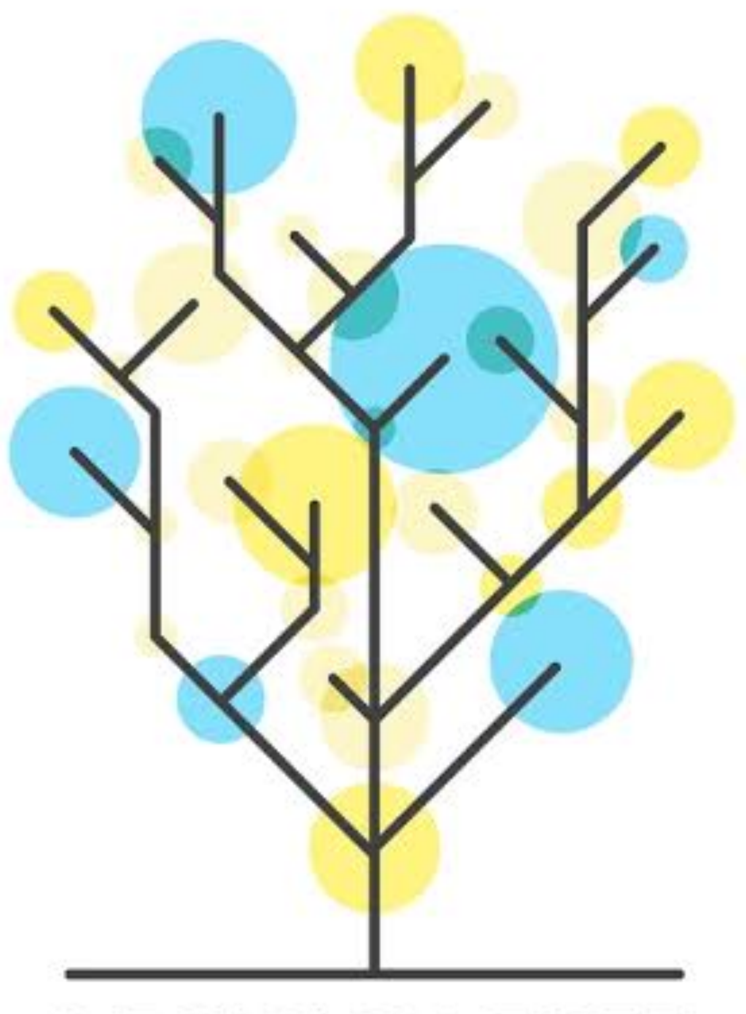


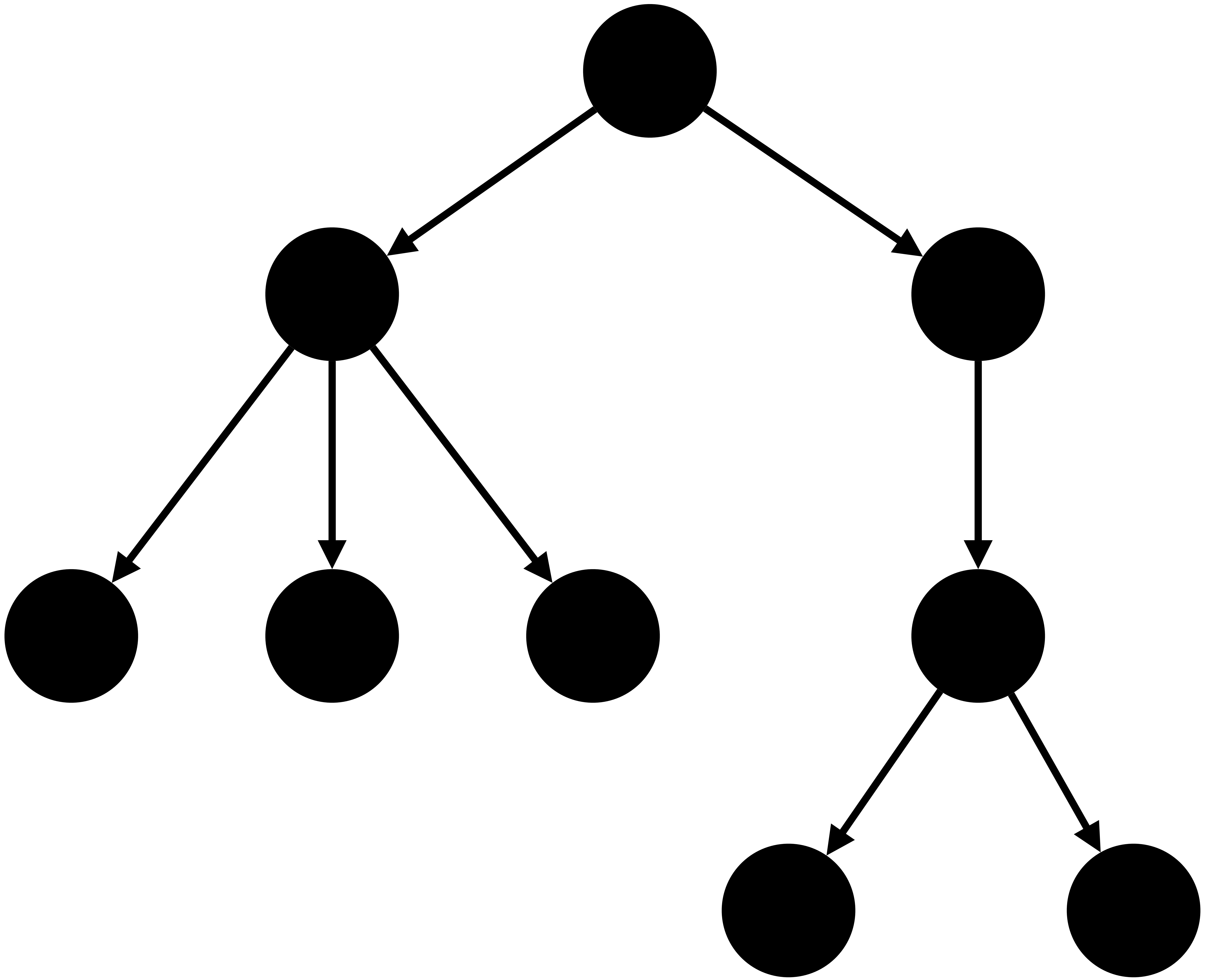
Hierarchical Structure

Tree data structure

N nodes -> N-1 edges
(1 incoming edge for each node)

- **Define with a logical model**
  - Node/Edge/Path
  - Parent/Children/Sibling
  - Ancestor/Descendant
  - Root/Leaf



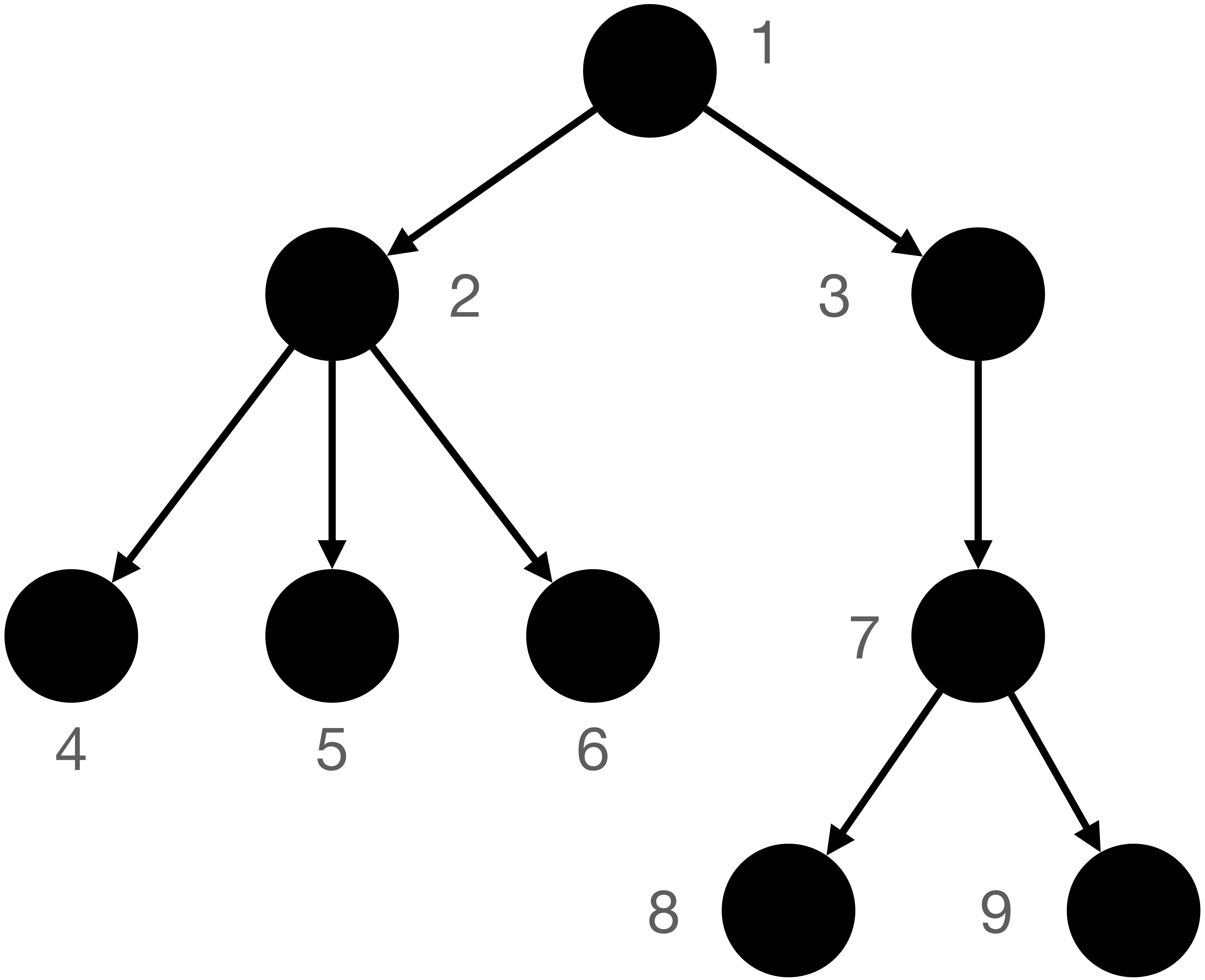


Hierarchical Structure

Tree data structure

- **Parent-child relationship**

Node	Parent	Children
2	1	4,5,6
7		
3		
1		
5		

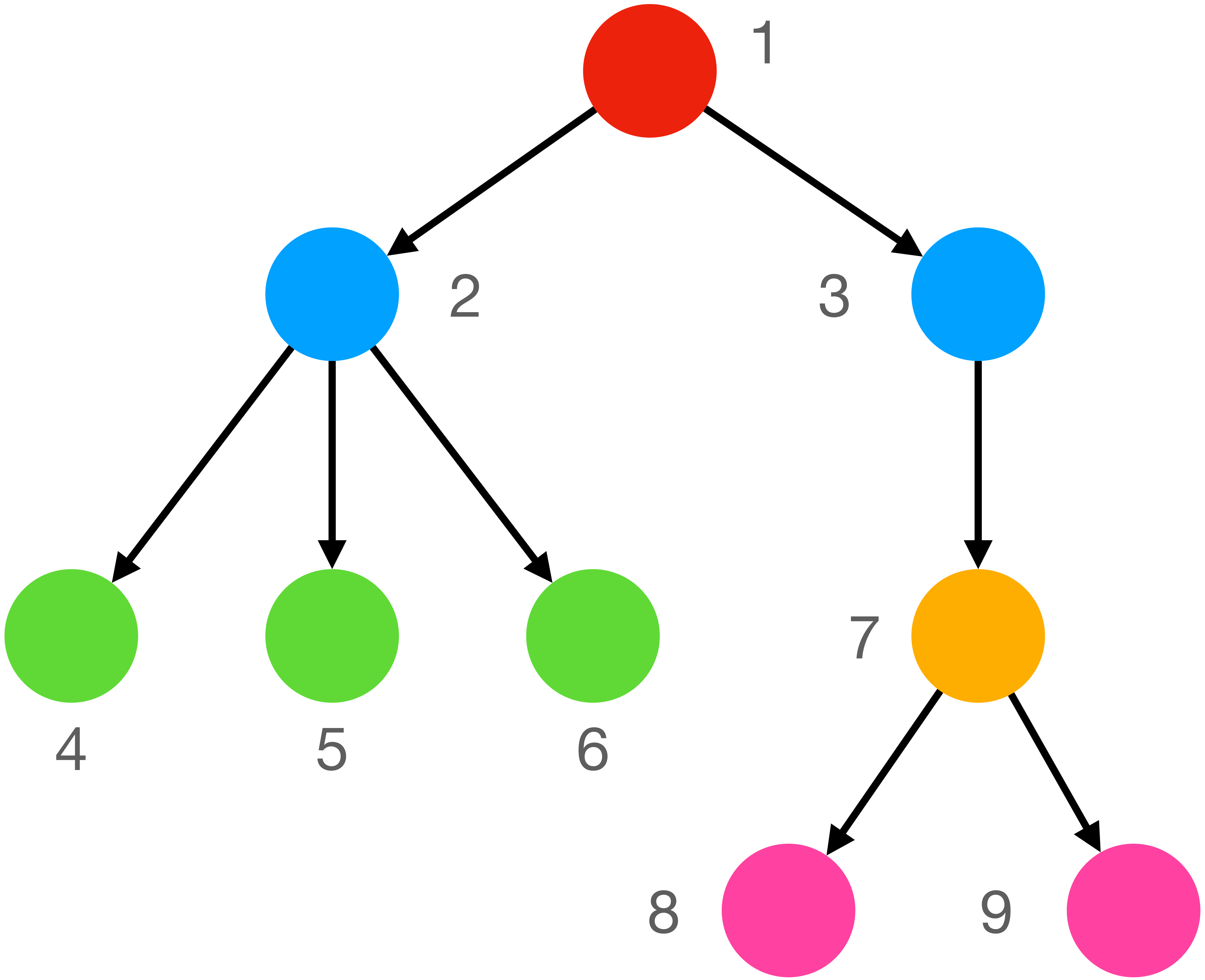


Hierarchical Structure

Tree data structure

- **Parent-child relationship**

Node	Parent	Children
2	1	4,5,6
7	3	8,9
3	1	7
1	-	2,3
5	2	-

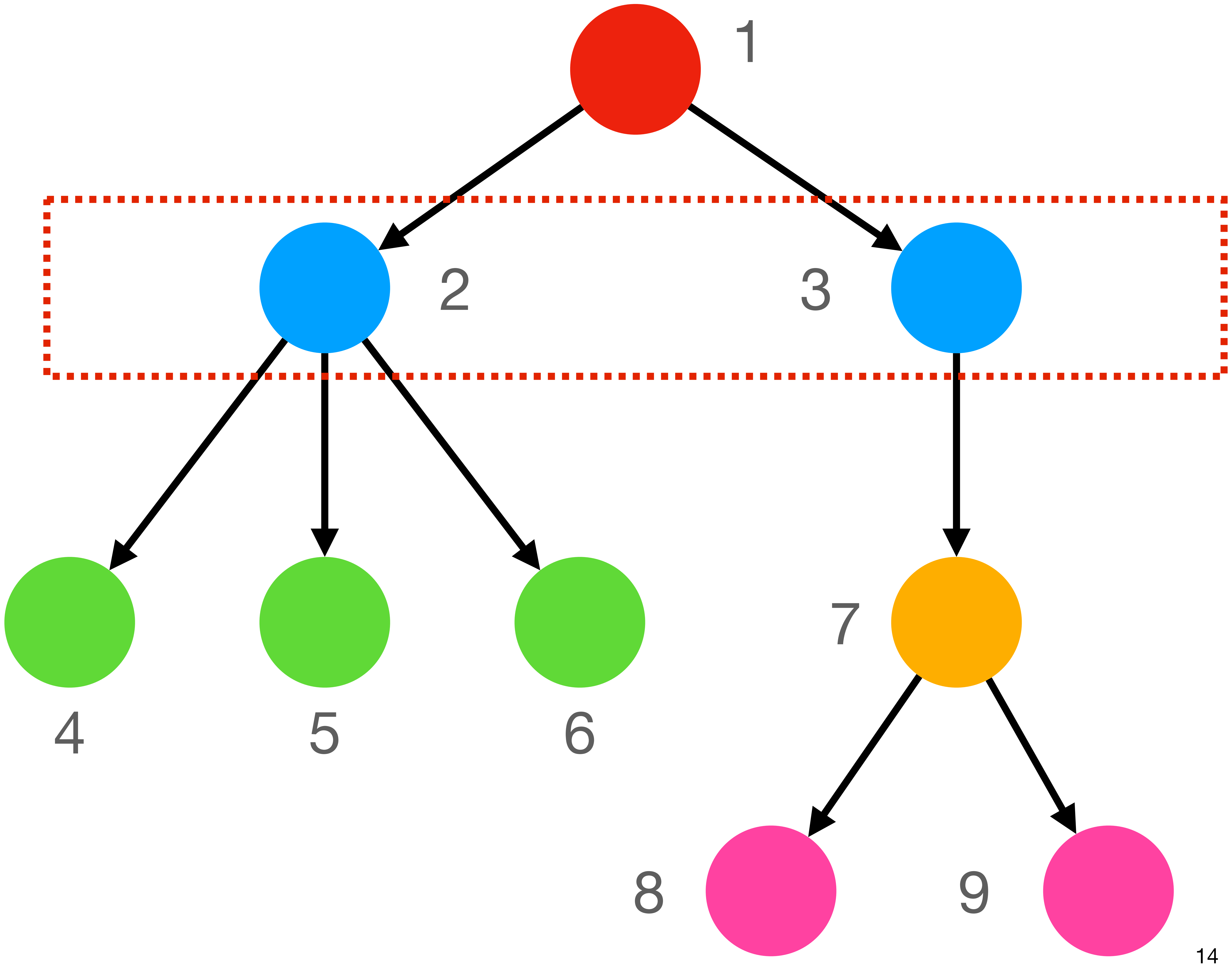


Hierarchical Structure

Tree data structure

- **Sibling**

Node	Parent	Children
2	1	4,5,6
7	3	8,9
3	1	7
1	-	2,3
5	2	-

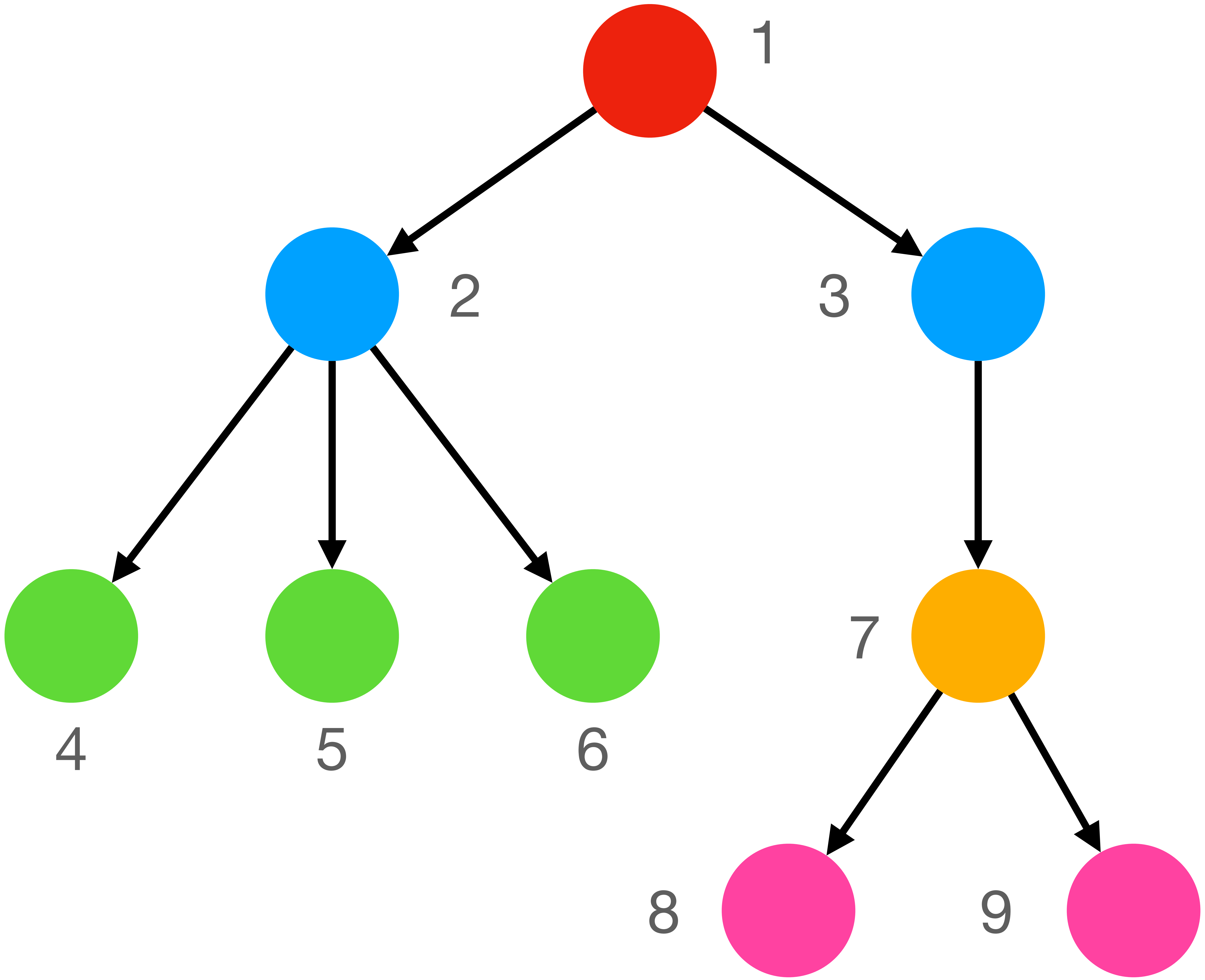


Hierarchical Structure

Tree data structure

- Root/Leaf

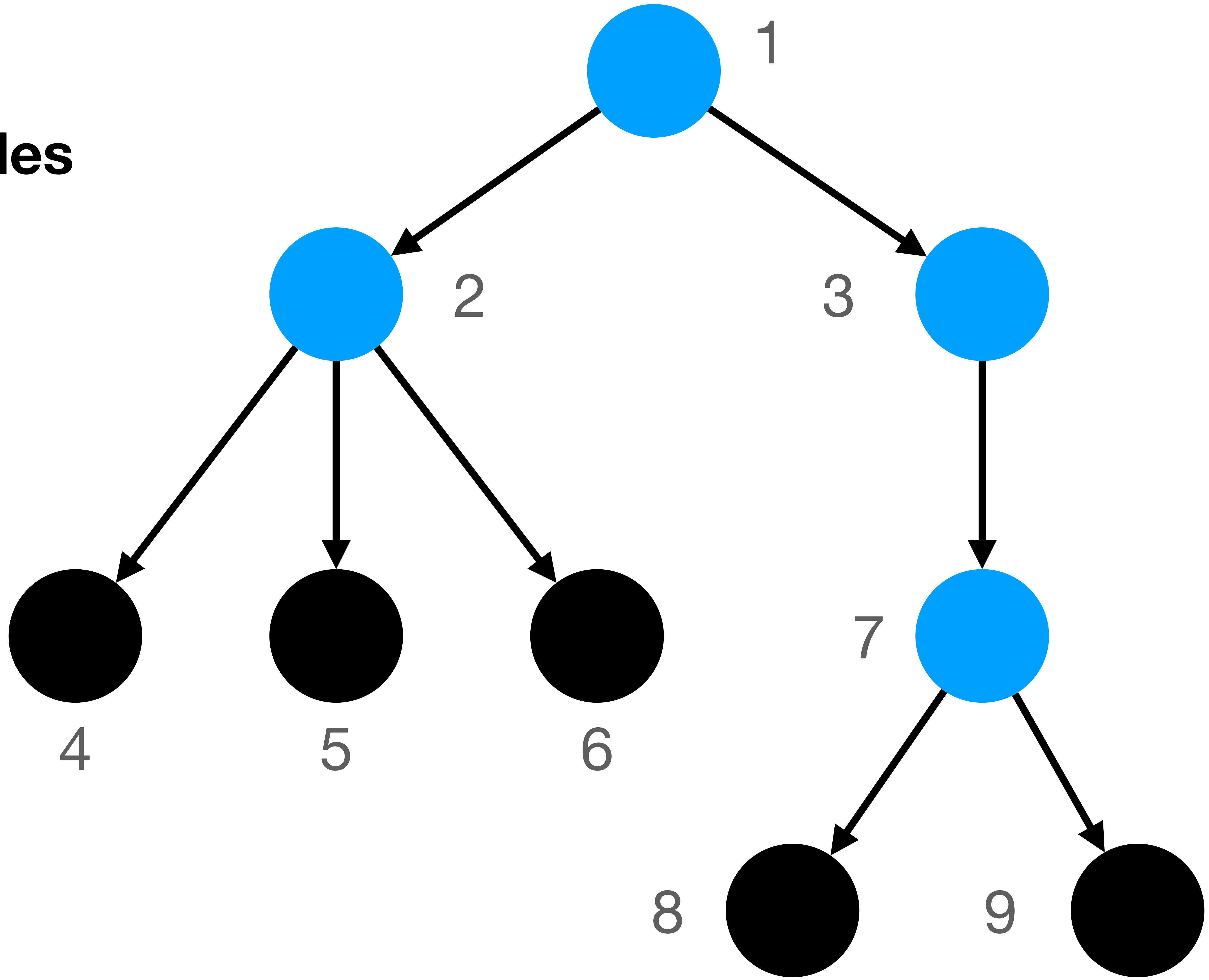
Node	Parent	Children
2	1	4,5,6
7	3	8,9
3	1	7
1	-	2,3
5	2	-



Hierarchical Structure

Tree data structure

- Internal or non-terminal nodes
- External or terminal nodes

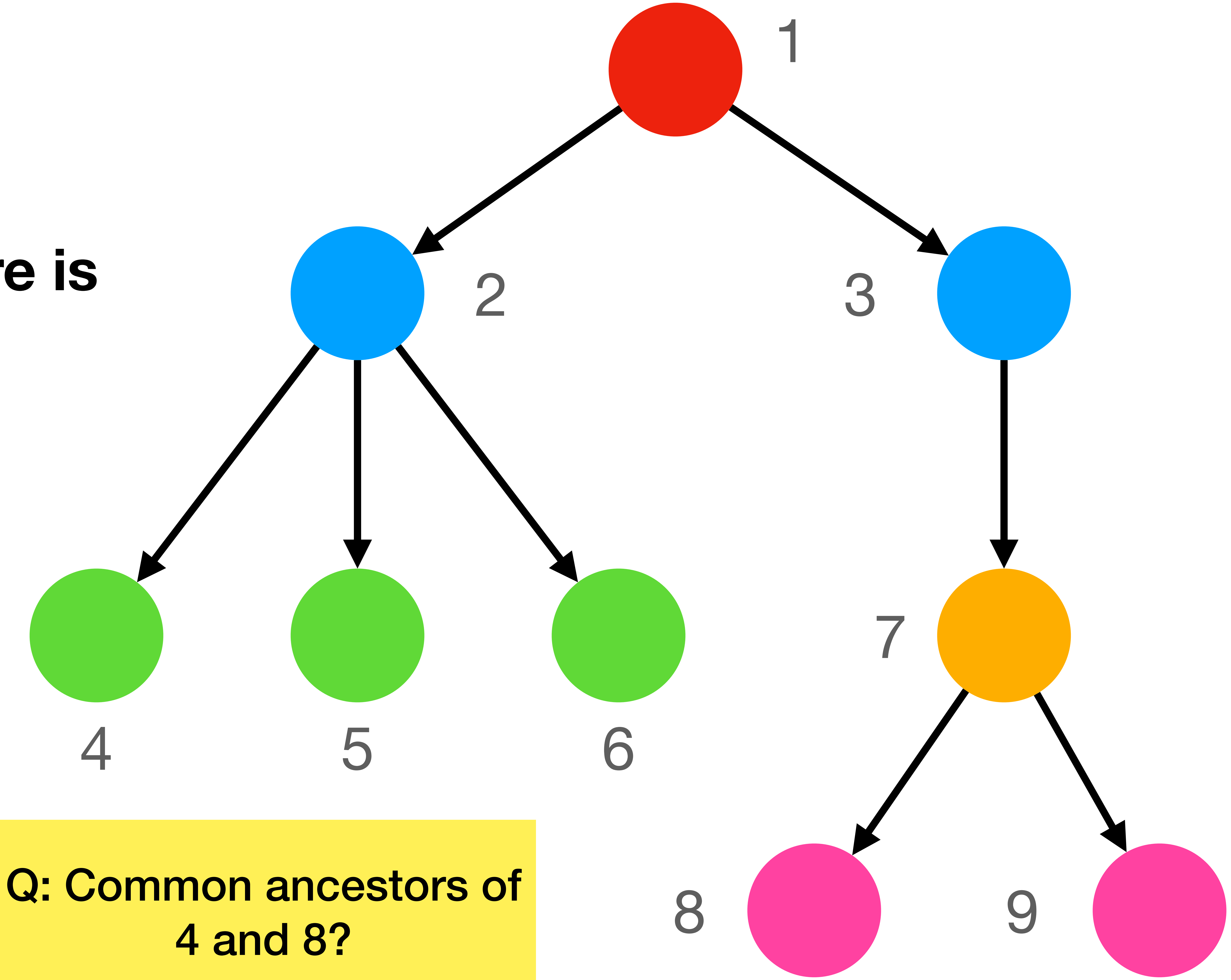


Hierarchical Structure

Tree data structure

- **Ancestor/Descendant**
- **A is an ancestor of B if there is a path from A to B**

Node	Ancestors
4	1,2
7	
8	

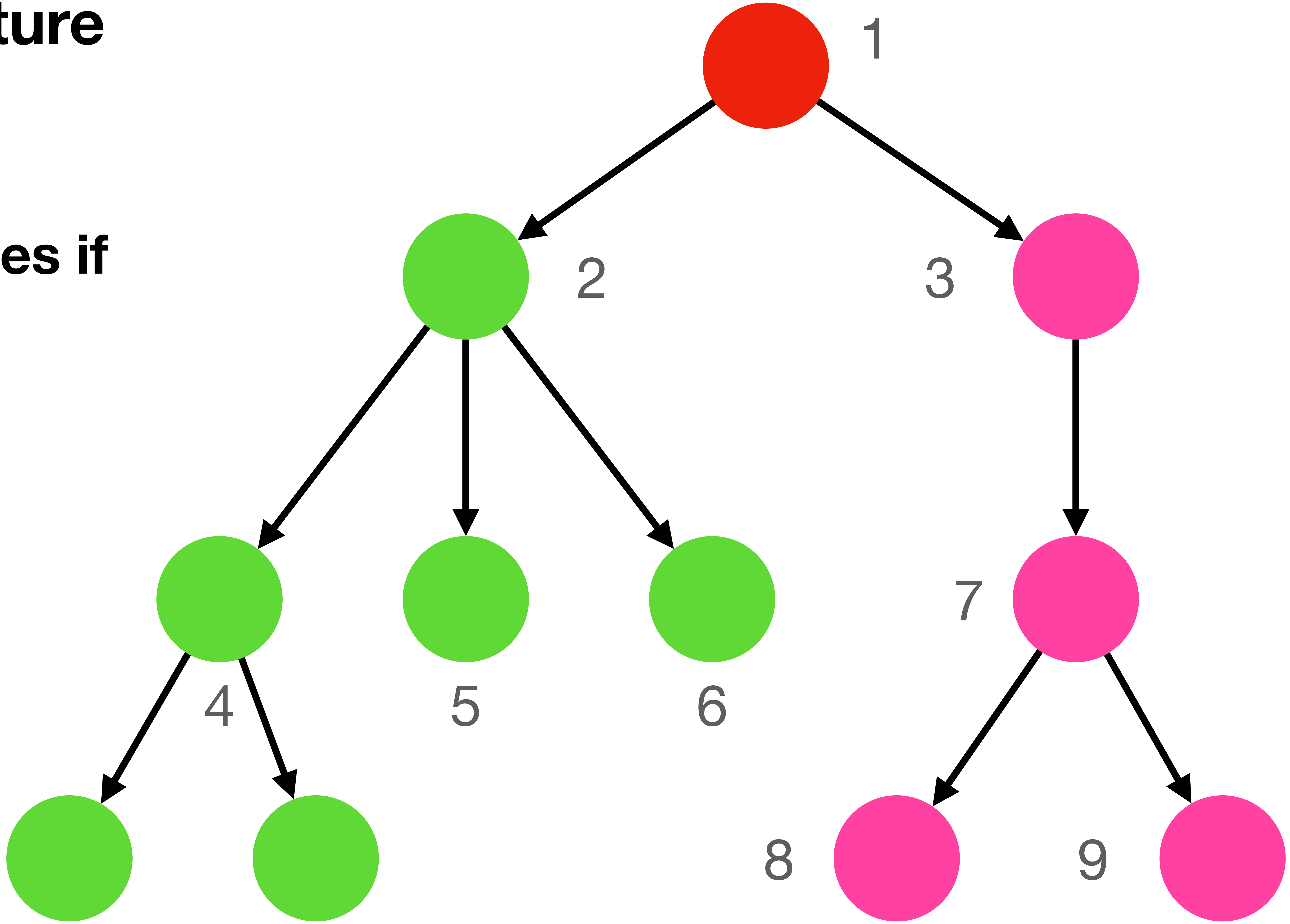


Q: Common ancestors of 4 and 8?

Hierarchical Structure

Recursive data structure

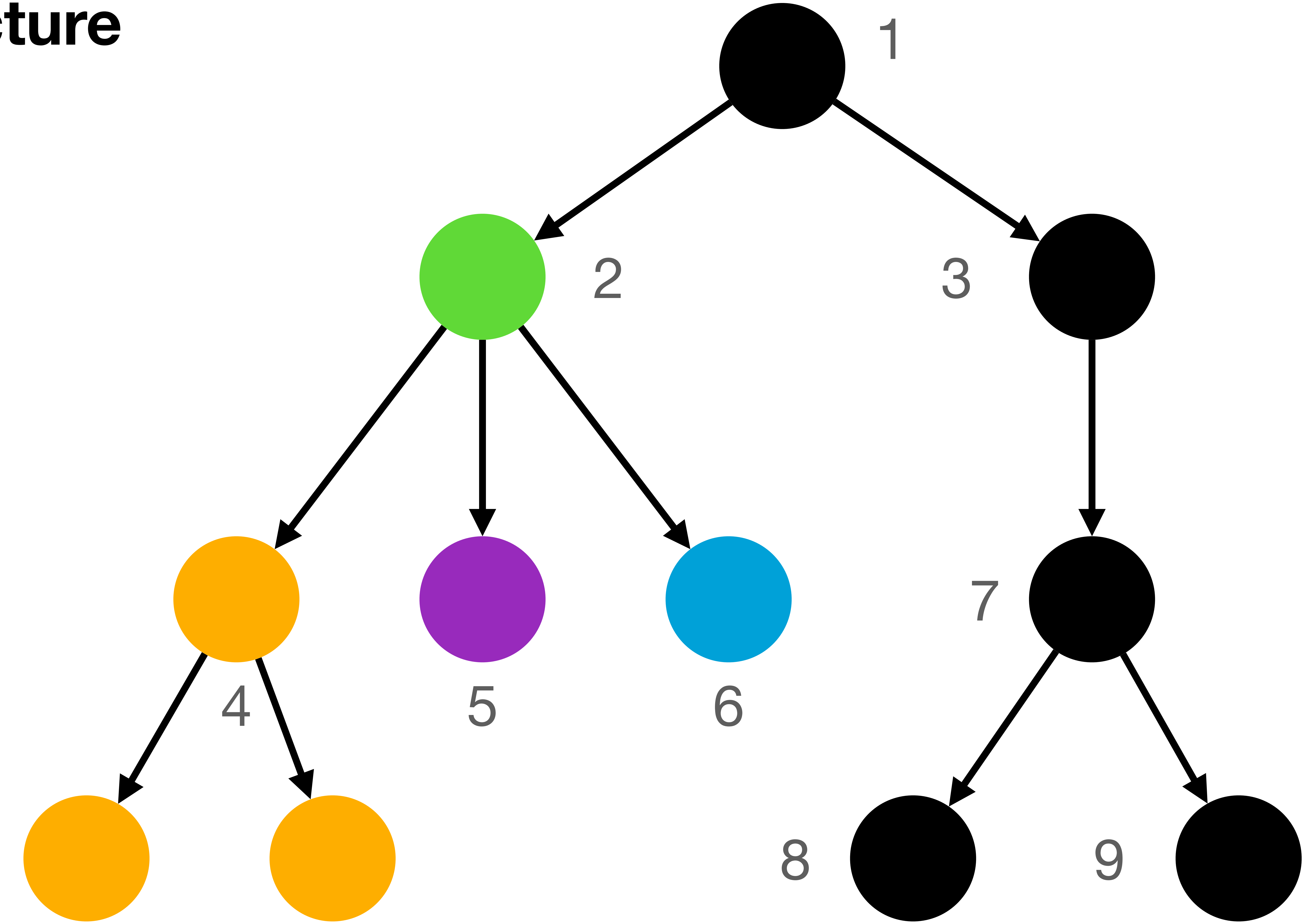
- **Sub-tree**
- **Forest: There are 2 trees if node 1 is removed.**



Hierarchical Structure

Recursive data structure

- **Sub-tree**

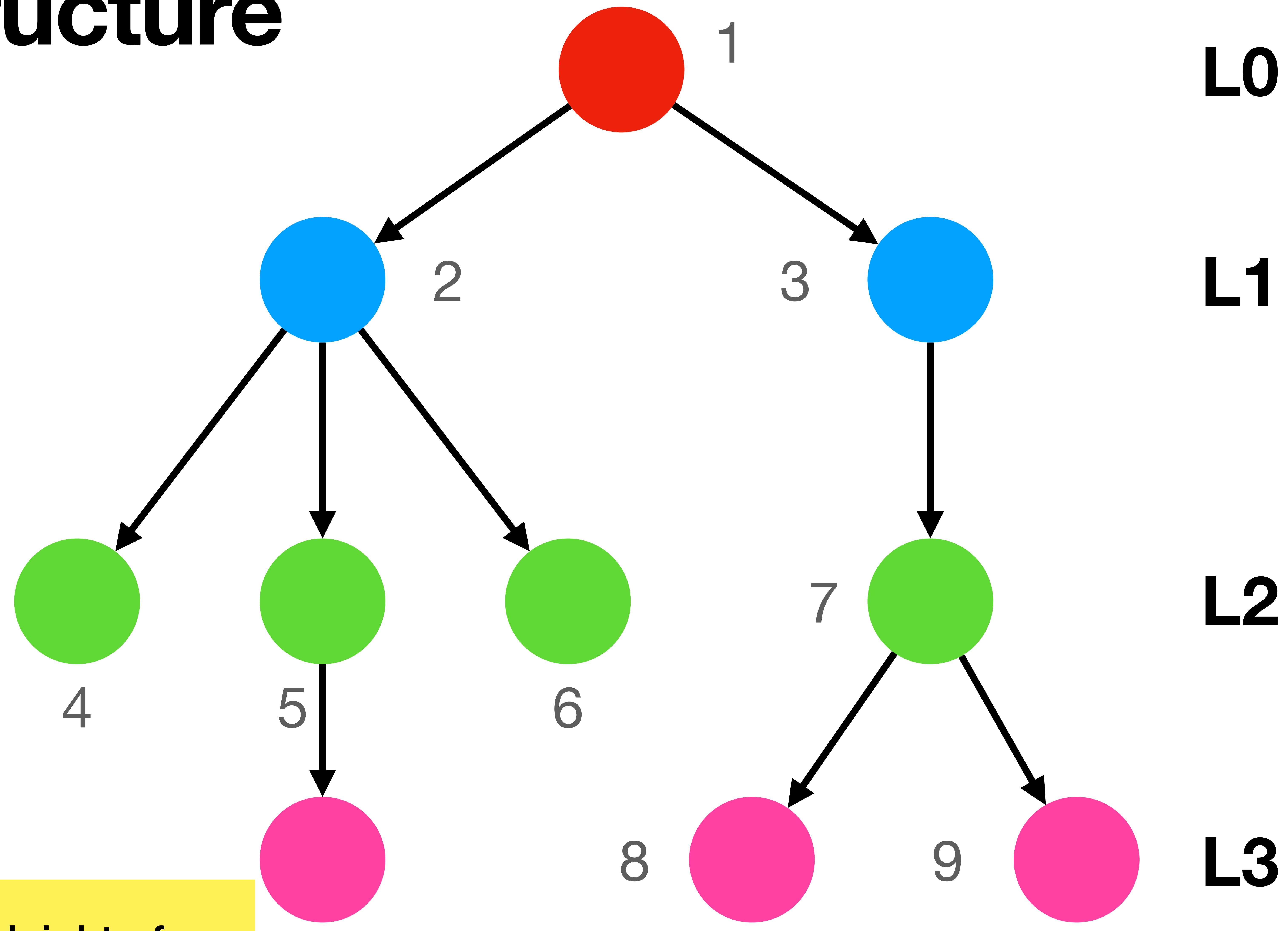


Hierarchical Structure

Tree data structure

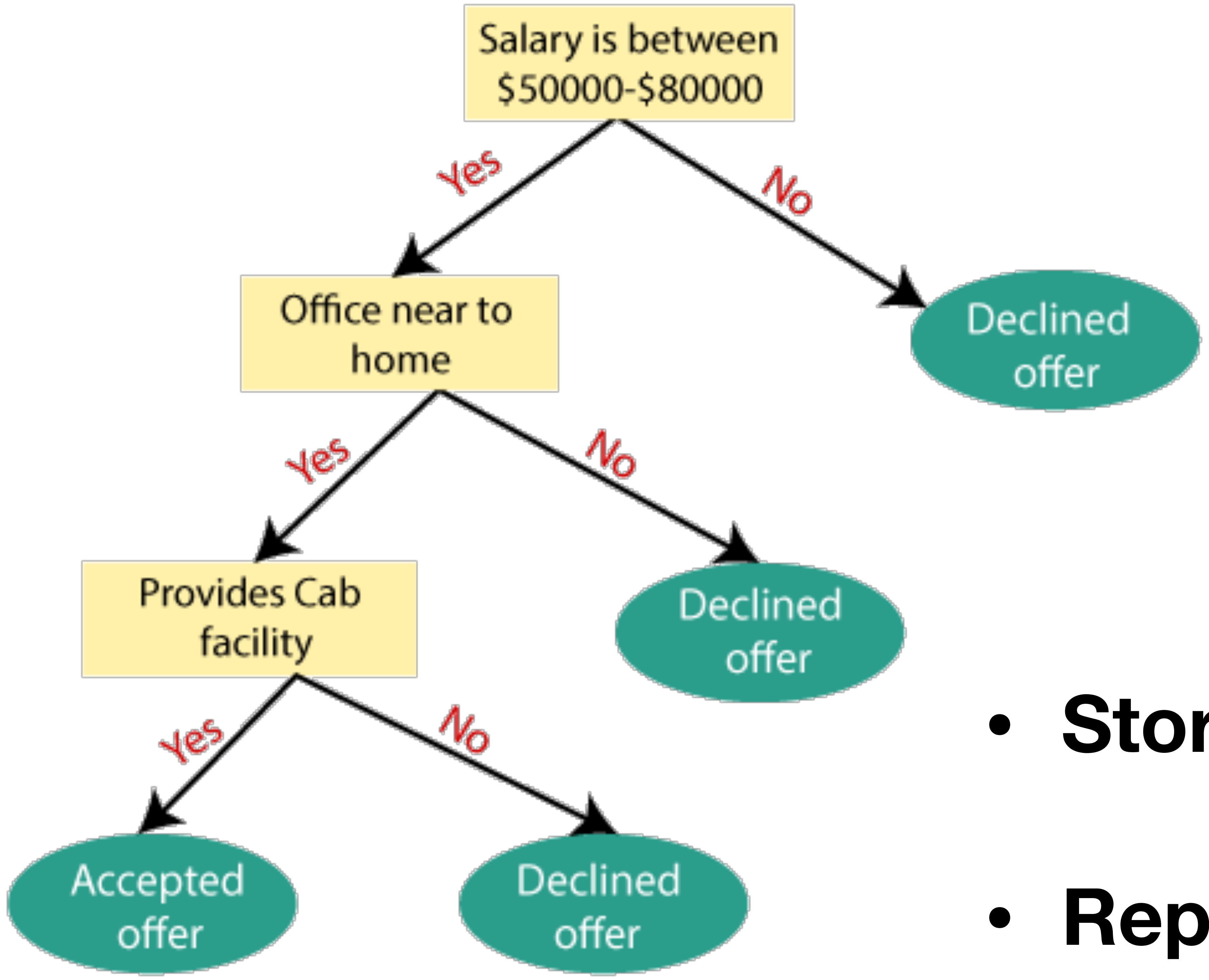
- **Level:** 4 levels
- **Path length:** #edges between 2 nodes
- **Depth:** path length from the root (= level, L)
- **Height:** the longest path length to the leaf

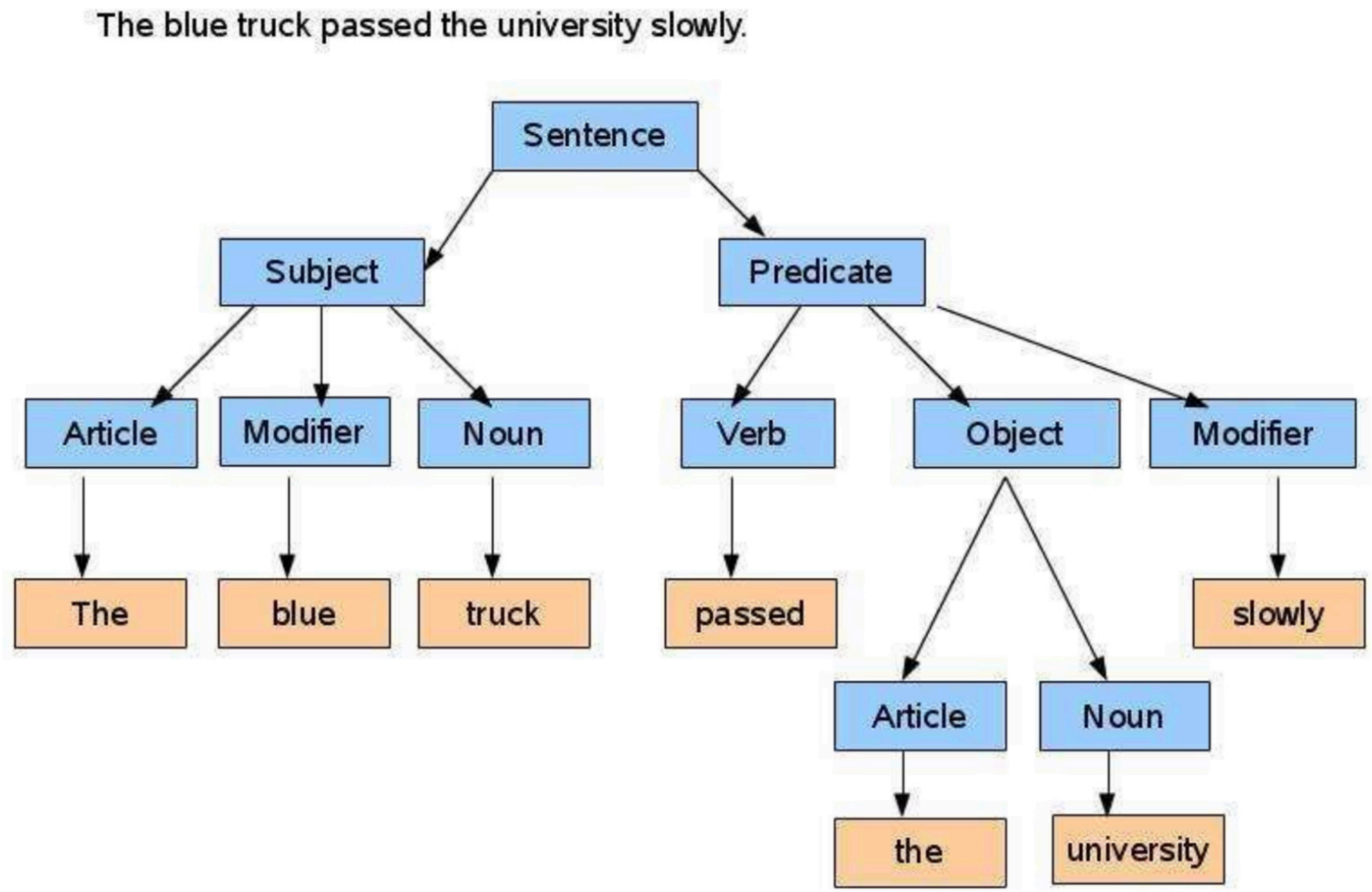
Q: Height of node 2 = ?



Applications

Tree data structure

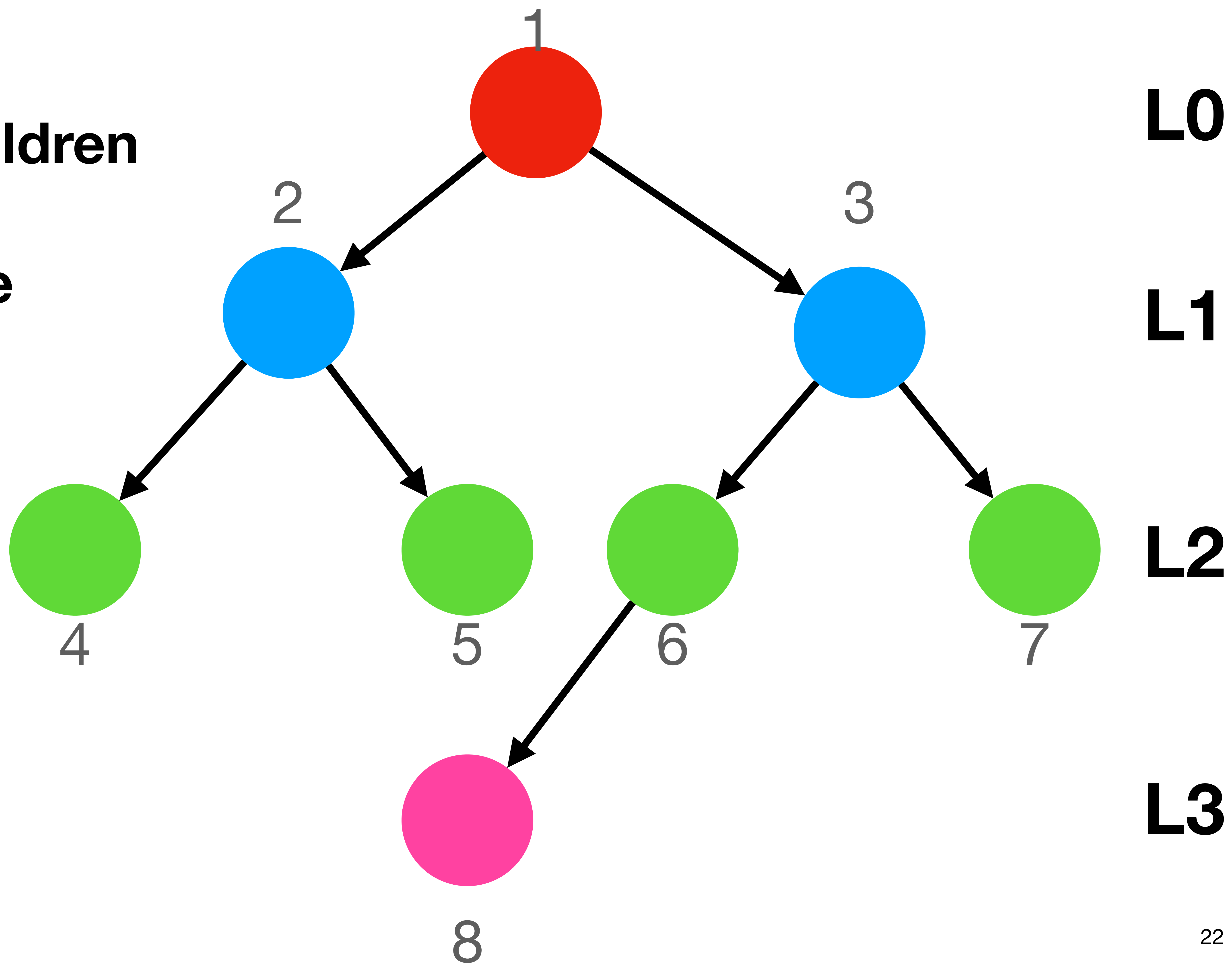




- **Storing hierarchical data:** file system, family tree, decision tree
- **Represent structure of a sentence in natural language**
- **Routing information**

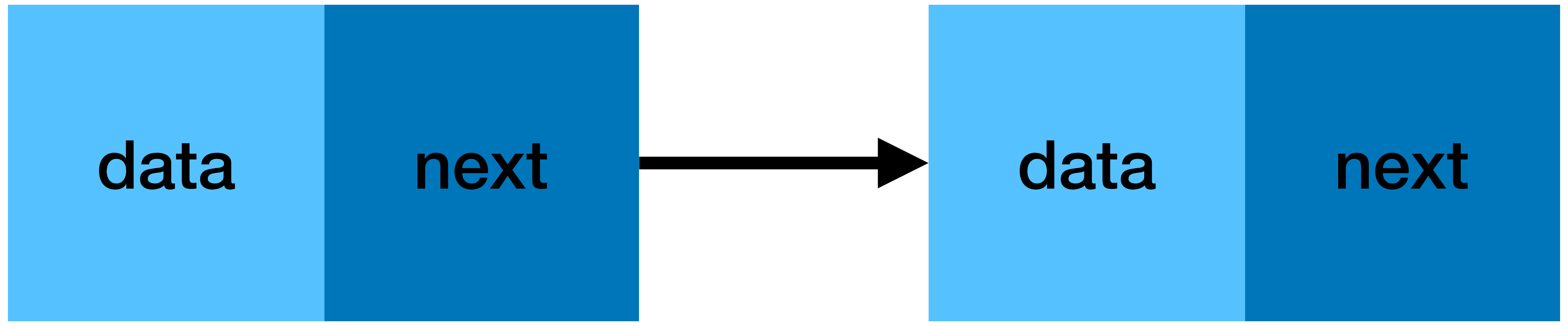
Binary tree

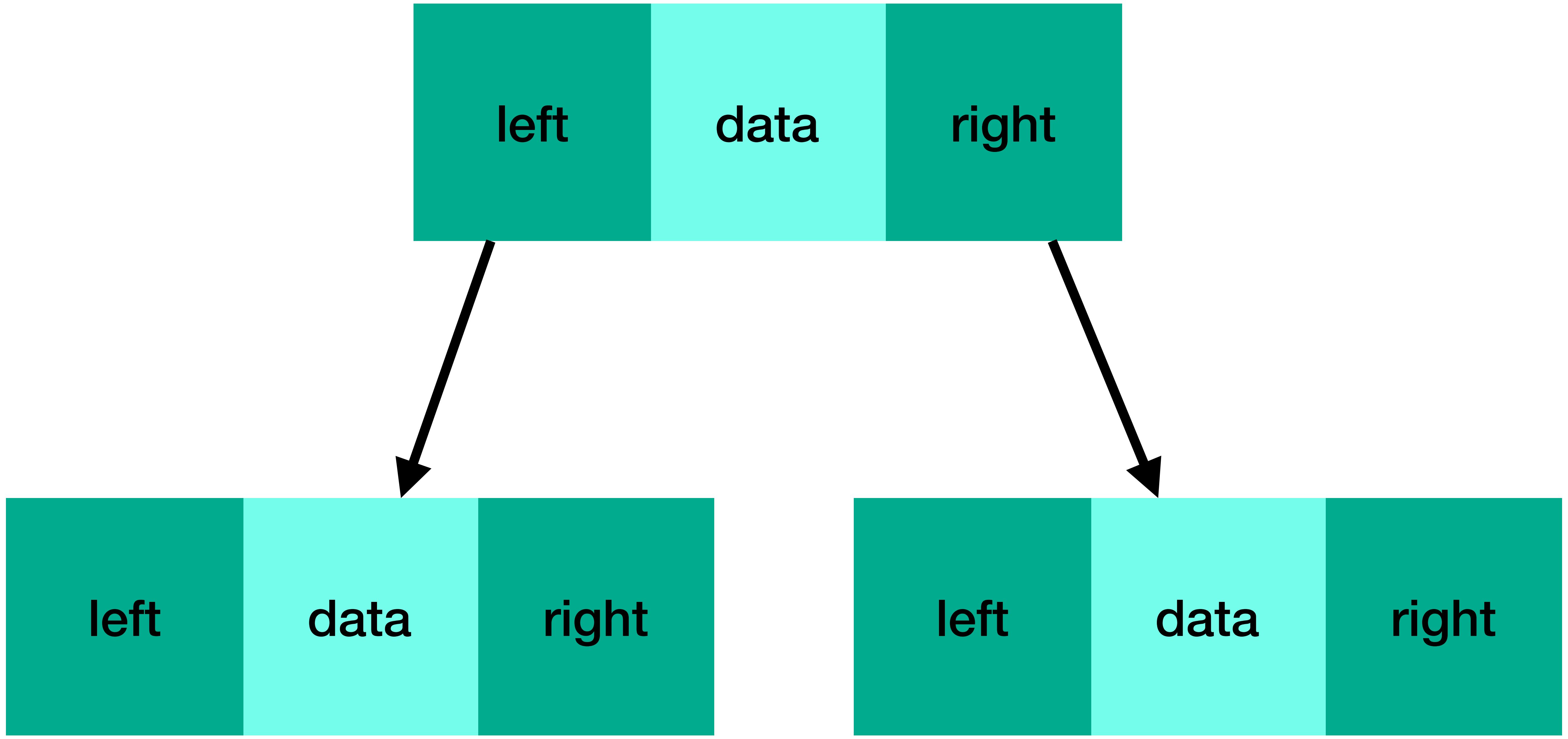
- Any node have at most 2 children
- Left Subtree & Right Subtree
- Left Child & Right Child



Node Structure

Linked List vs Tree



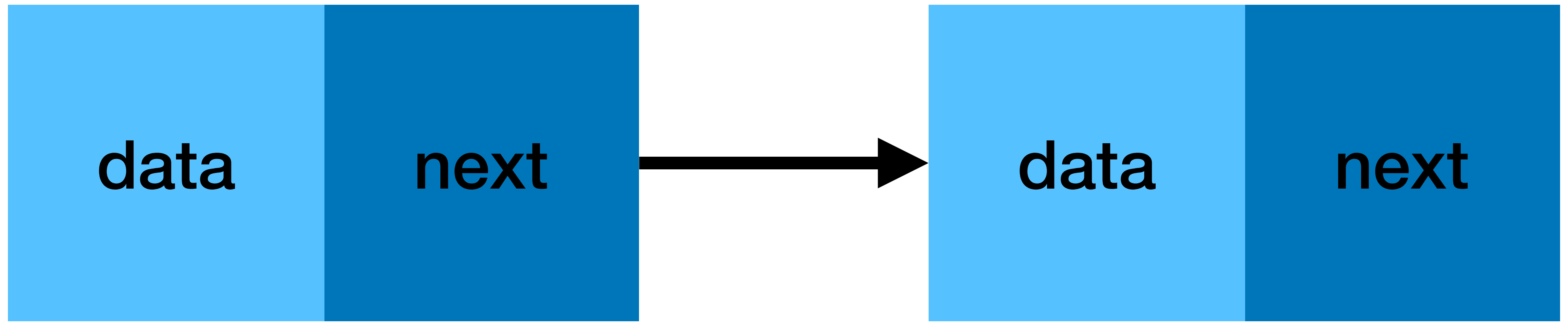


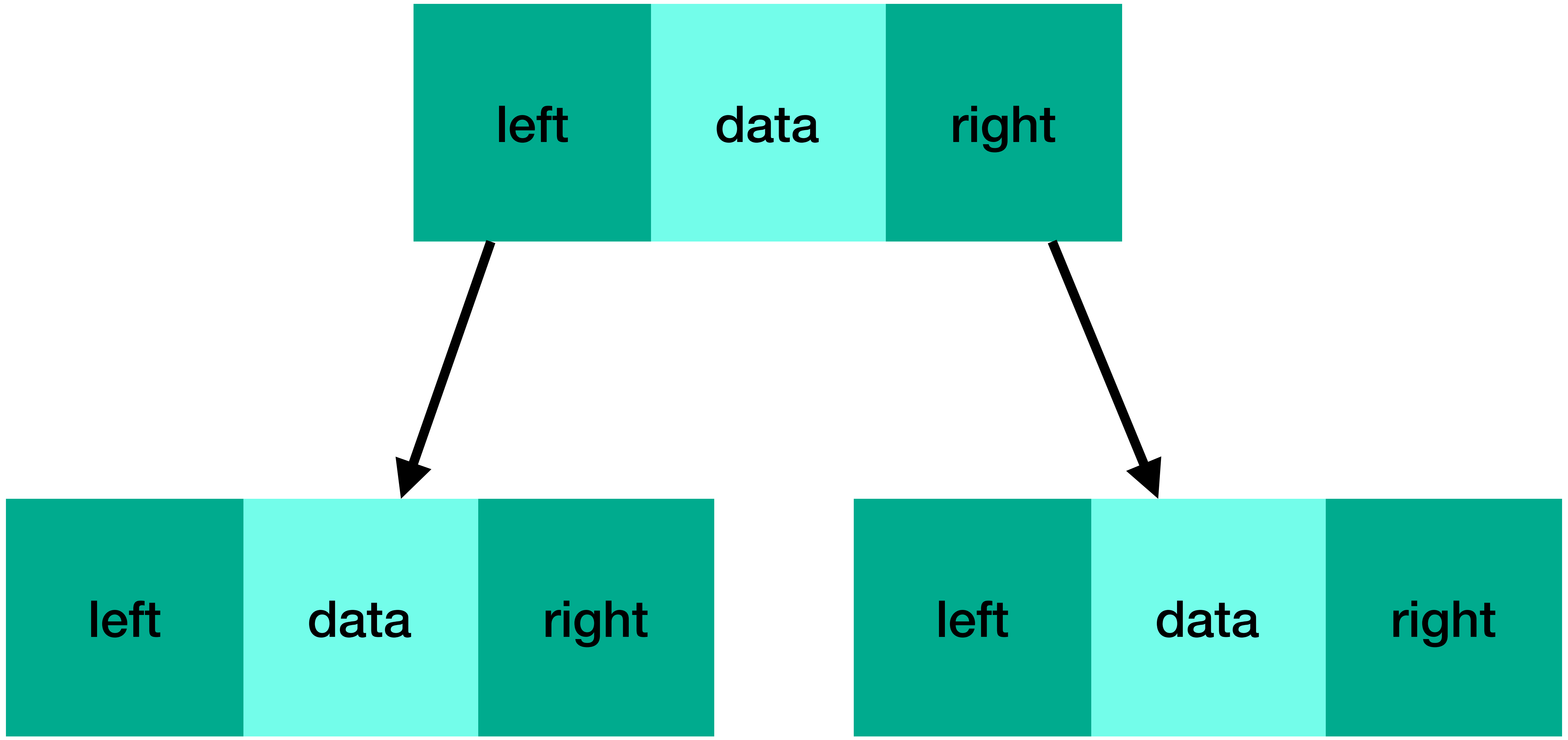

```
typedef struct _listnode
{
   int data;
   struct _listnode* next;
} LISTNODE_T;
```

```
typedef struct _treenode
{



} TREENODE_T;
```

Node Structure

Linked List vs Tree






```
typedef struct _listnode
{
   int data;
   struct _listnode* next;
} LISTNODE_T;
```

```
typedef struct _treenode
{
   int data;
   struct _treenode* left;
   struct _treenode* right;
} TREENODE_T;
```

Tree Structure

Representation of binary trees

```
typedef struct _treenode
{
   int data;
   struct _treenode* left;
   struct _treenode* right;
} TREENODE_T;
```

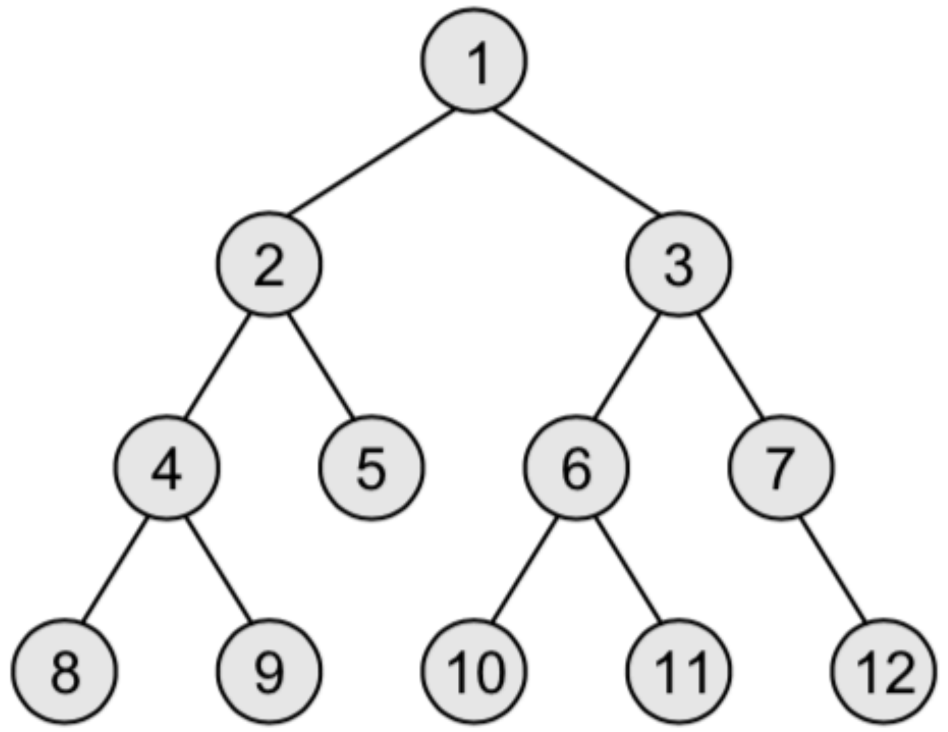


Figure 9.10 Binary tree T

ROOT
3

15
AVAIL

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

Figure 9.11 Linked representation of binary tree T

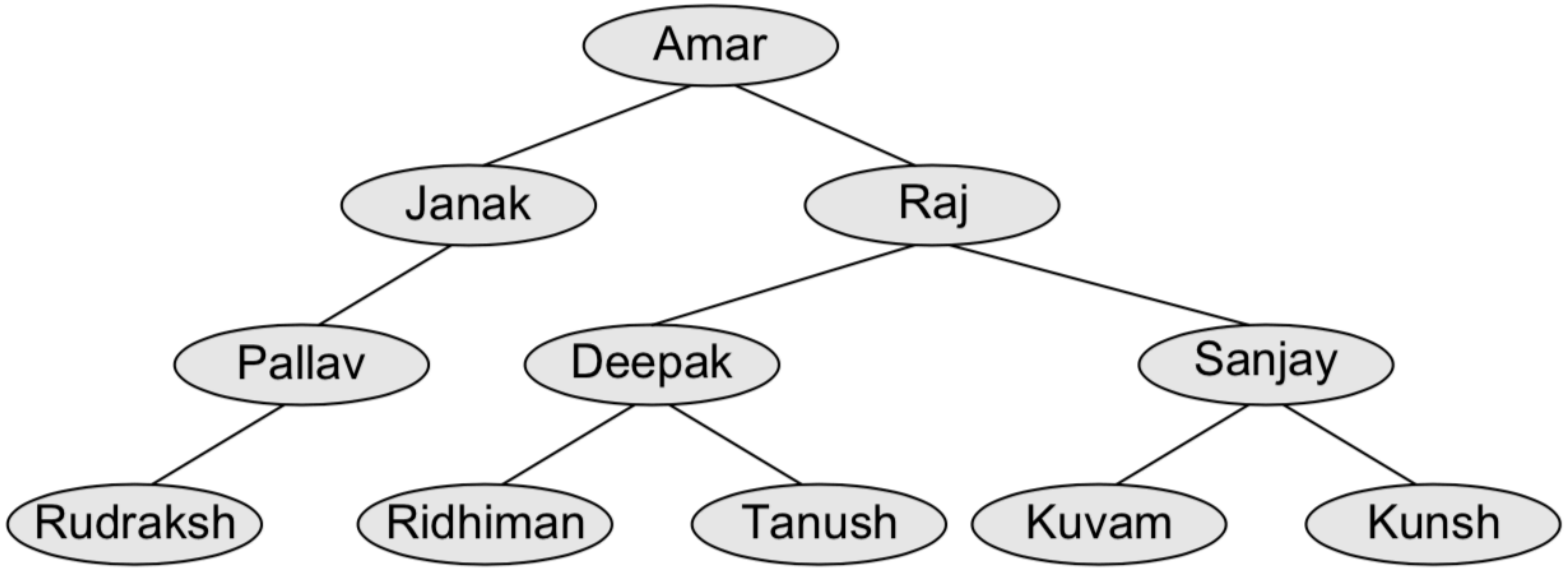
Tree Structure

Representation of binary trees

ROOT: 3

AVAIL: 7

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



Tree Structure

Custom - Structure & New node

tree.h

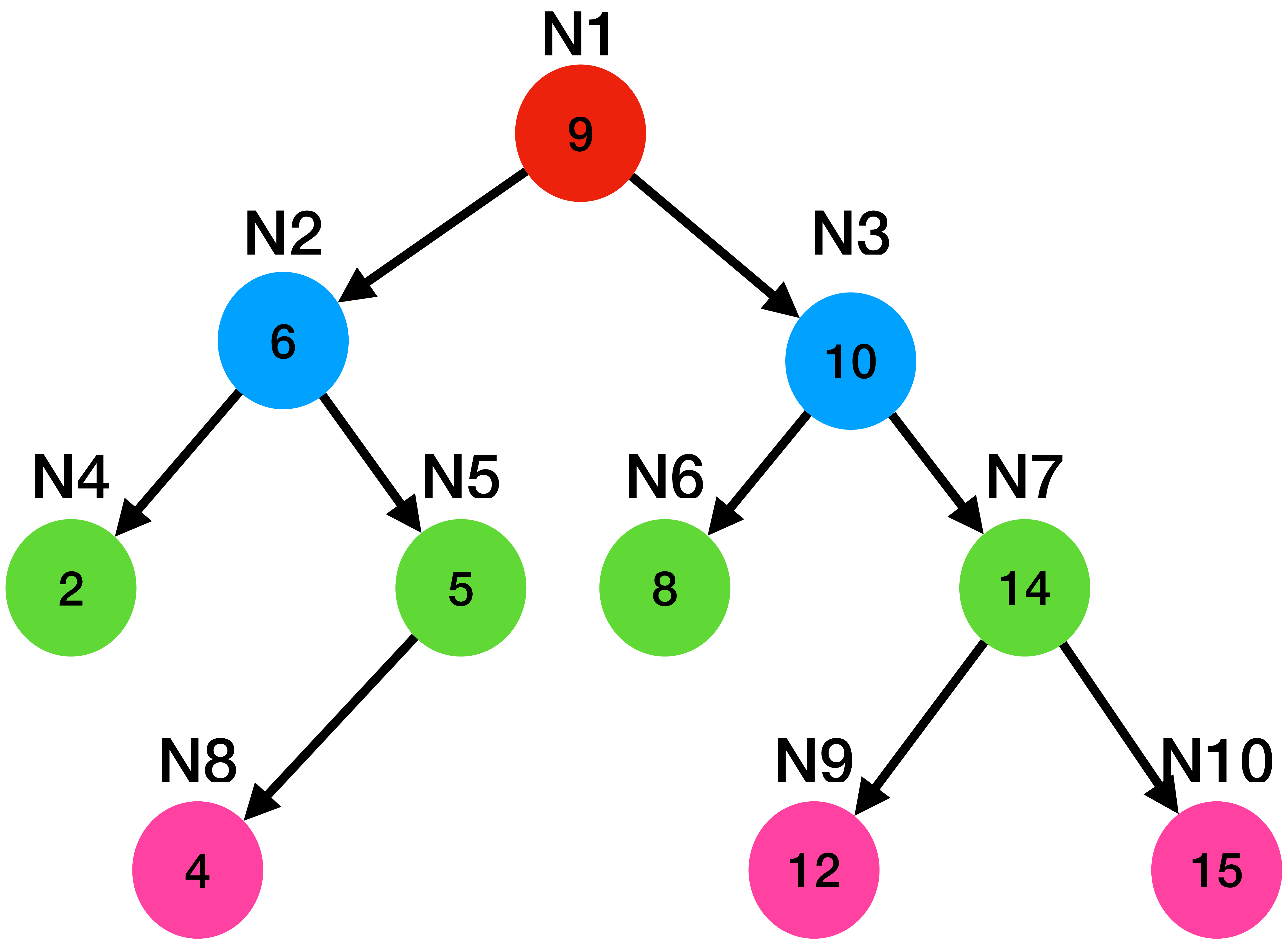
```
typedef struct _treenode
{
   int data;
   struct _treenode* left;
   struct _treenode* right;
} TREENODE_T;
```

tree.c

```
TREENODE_T *newNodeCreate(int item)
{
   TREENODE_T* new_node;
   new_node = (TREENODE_T*)calloc(1,
   sizeof(TREENODE_T));
   new_node->data = item;
   new_node->left = NULL;
   new_node->right = NULL;
   return new_node;
}
```

Tree Structure

Custom - Create new node



main.c

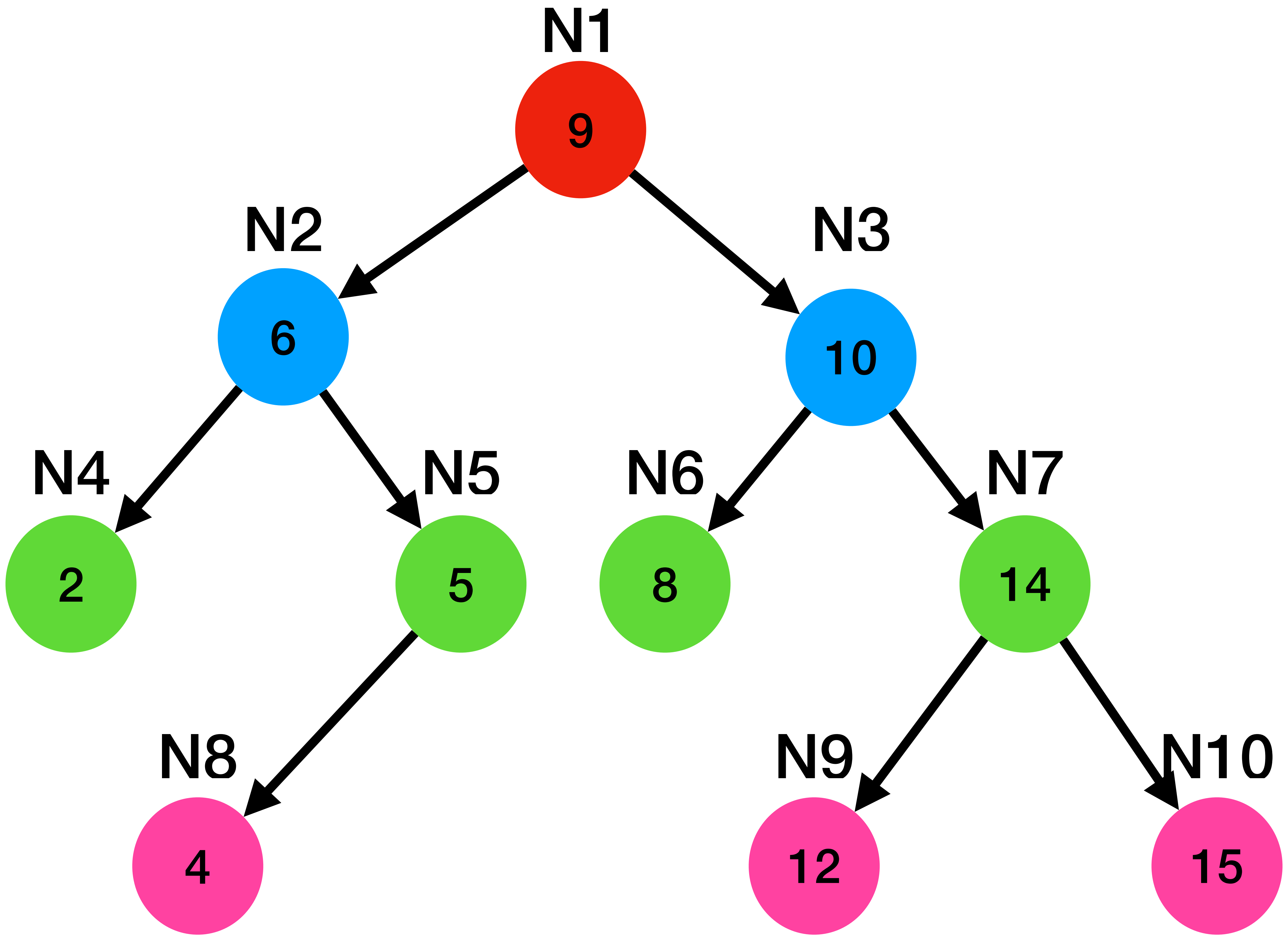
```
TREENODE_T* N1 = newNodeCreate(9);
TREENODE_T* N2 = newNodeCreate(6);
TREENODE_T* N3 = newNodeCreate(10);
TREENODE_T* N4 = newNodeCreate(2);
...
```

Tree Structure

Custom - Child node

tree.c

```
int setChildNode (TREENODE_T* parent,
TREENODE_T* child, char direction)
{
   int success = 1;
   //check null node or other conditions?
   if (direction == ‘L’)
      parent->left = child;
   else if (direction == ‘R’)
      parent->right = child;
   else
      success = 0;
   return success;
}
```



main.c

```
int status;
status = setChildNode (N1, N2, ‘L’);
status = setChildNode (N1, N3, ‘R’);
status = setChildNode (N2, N4, ‘L’);
...
```

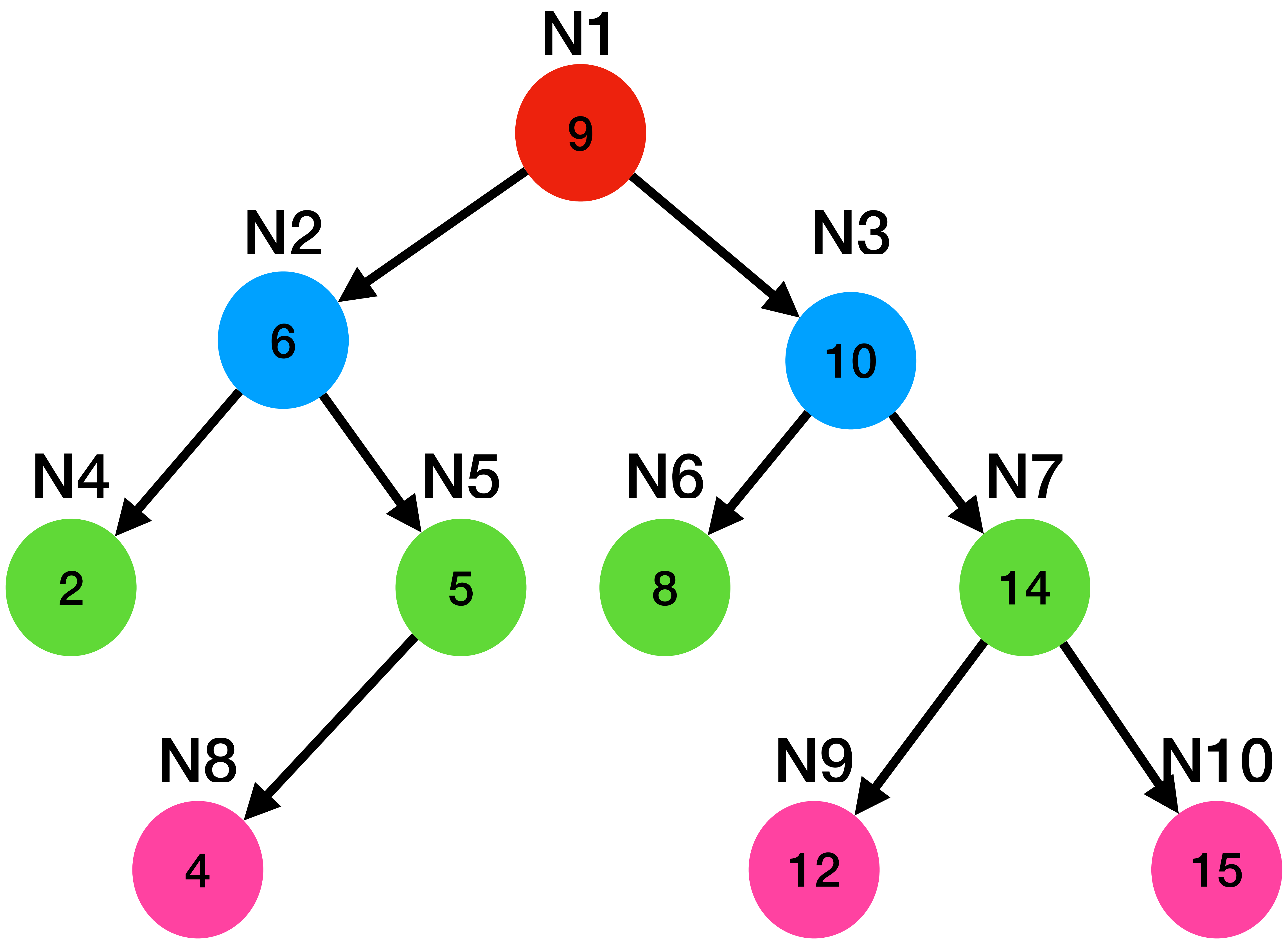
Tree Structure

Custom - Free node

main.c

```
free (N10);
N7->right = NULL;
free (N9);
free (N7);
...
```

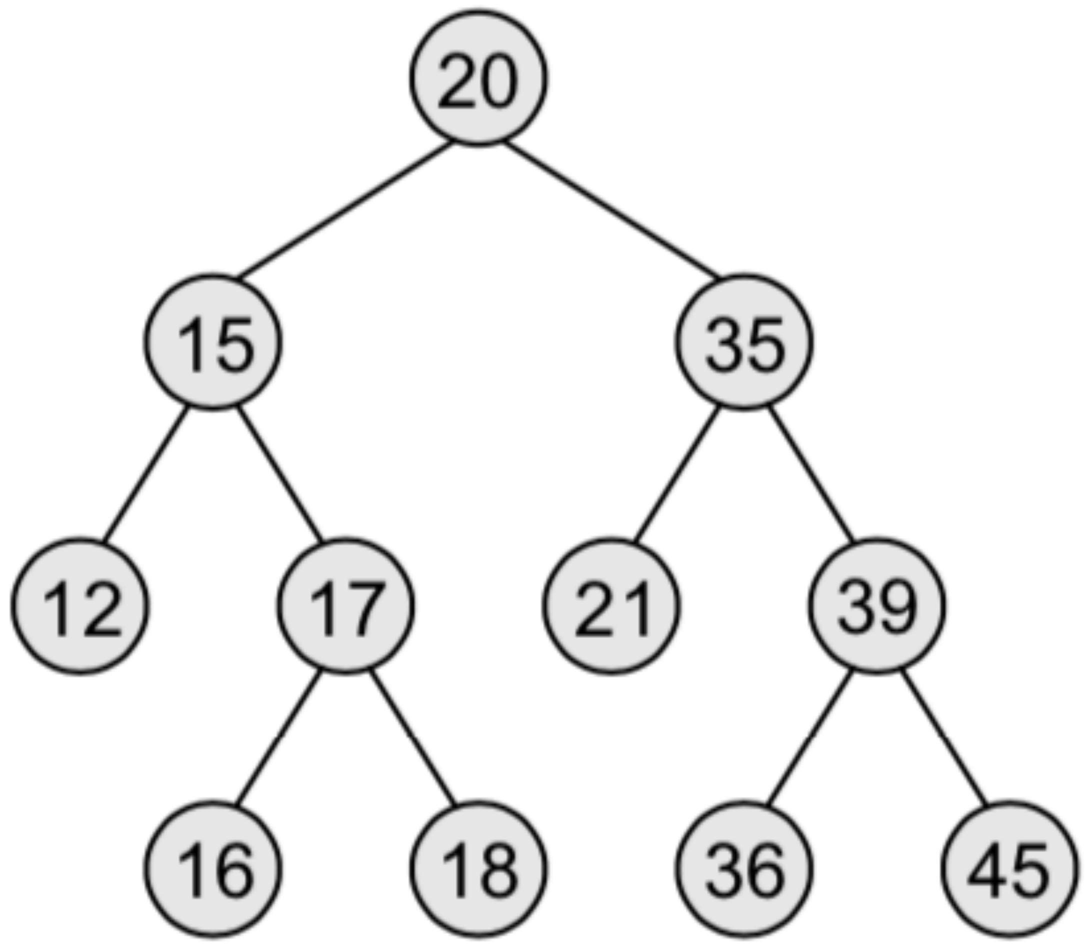
Print to check data before free
EX: N1->left->data



Tree Structure

Sequential representation of binary trees

- 1D Array
- The **root** will be stored in the first location.
- The **children** of a node in location K will be stored in locations (2 x K) and (2 x K + 1).

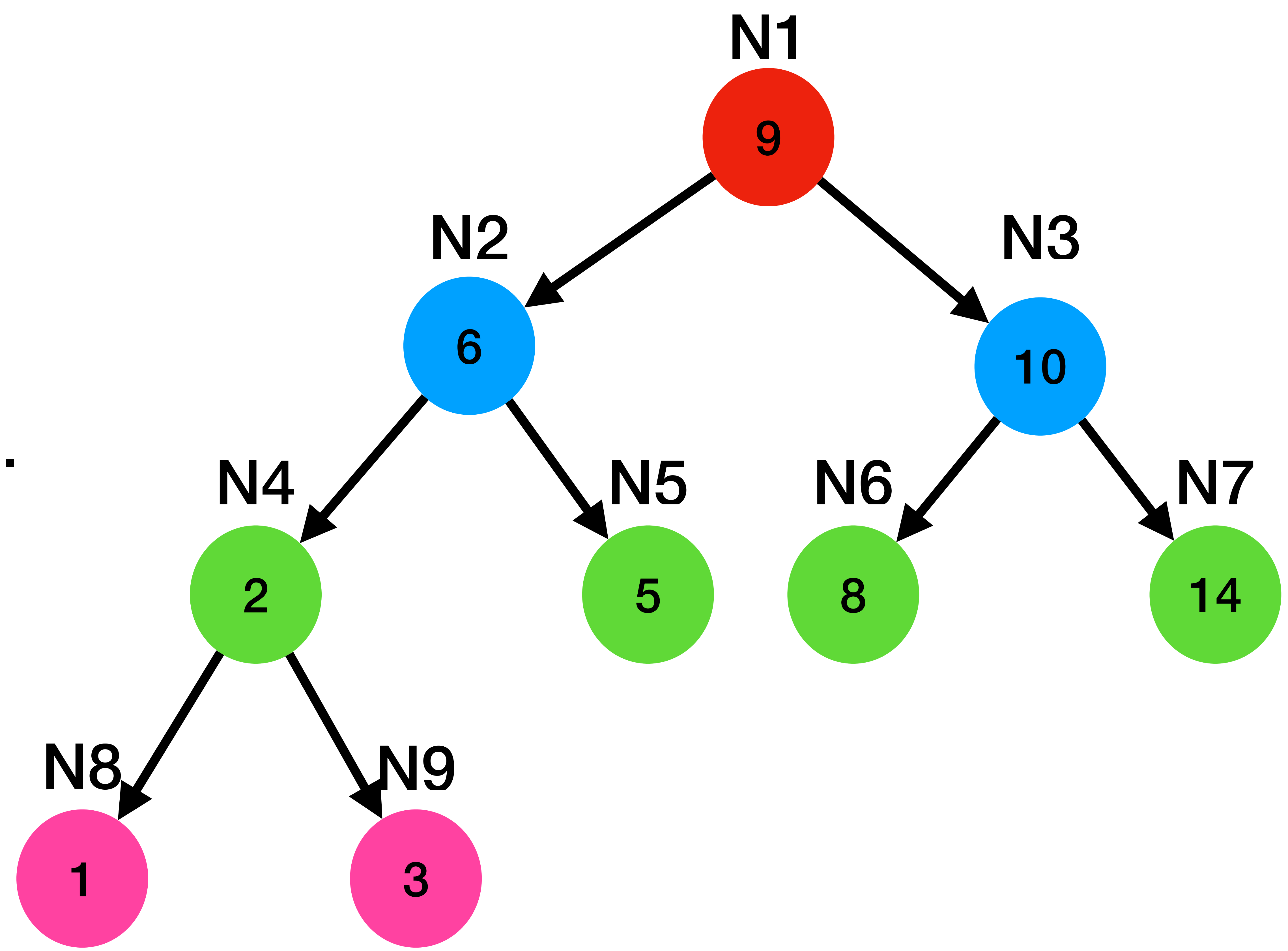


1	20
2	15
3	35
4	12
5	17
6	21
7	39
8	
9	
10	16
11	18
12	
13	
14	36
15	45

Complete Binary Trees

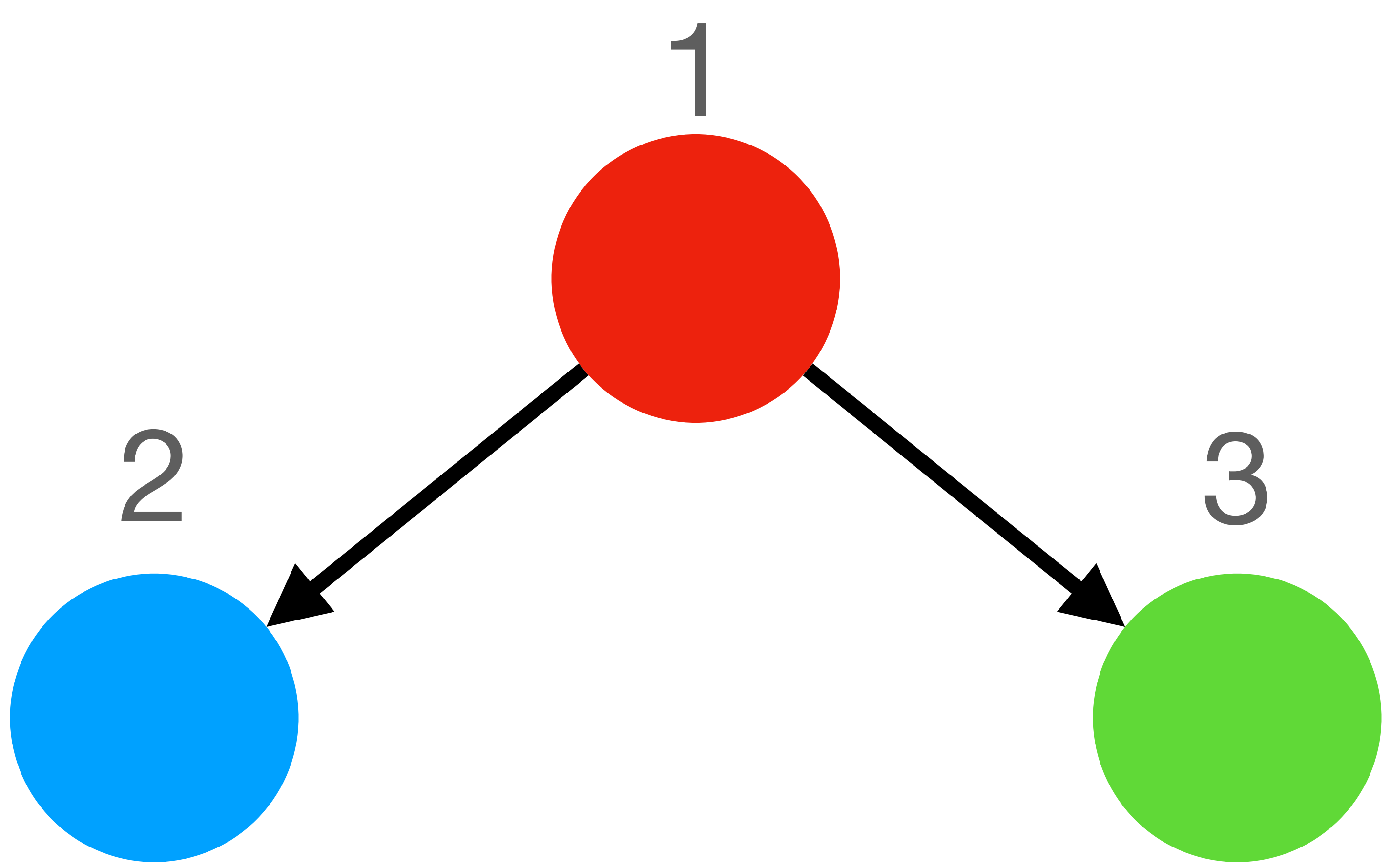
Properties

- Every level except possibly the last is completely filled.
- All nodes appear as far **left** as possible.
- A level r can have **at most 2^r nodes**



Tree Traversal

Depth-first

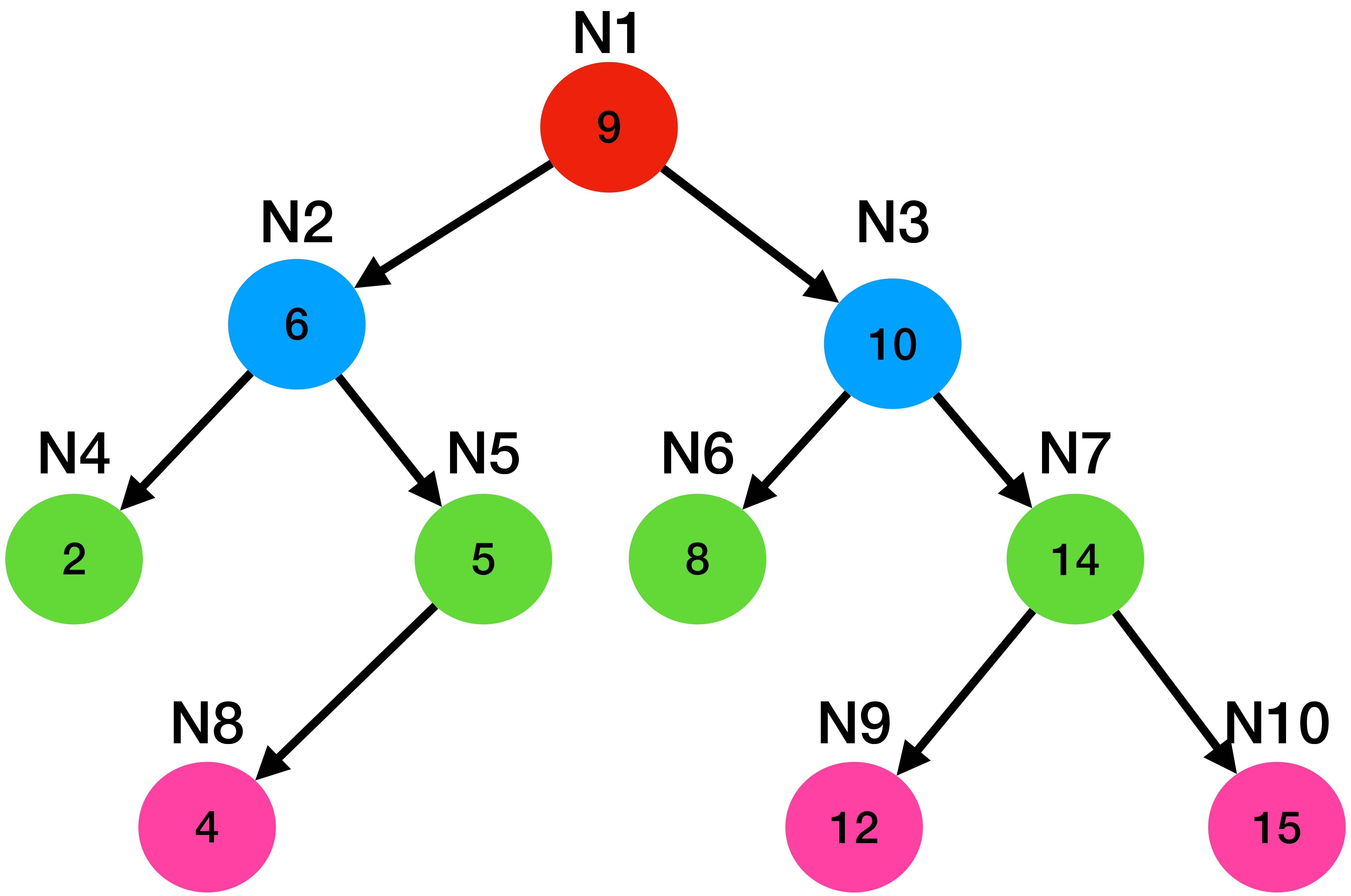


R: Root Node,
LT: Left sub-tree
RT: Right sub-tree

Order	(1)	(2)	(3)
Pre-order	R	LT Recursion	RT Recursion
In-order	LT Recursion	R	RT Recursion
Post-order	LT Recursion	RT Recursion	R

Tree Traversal

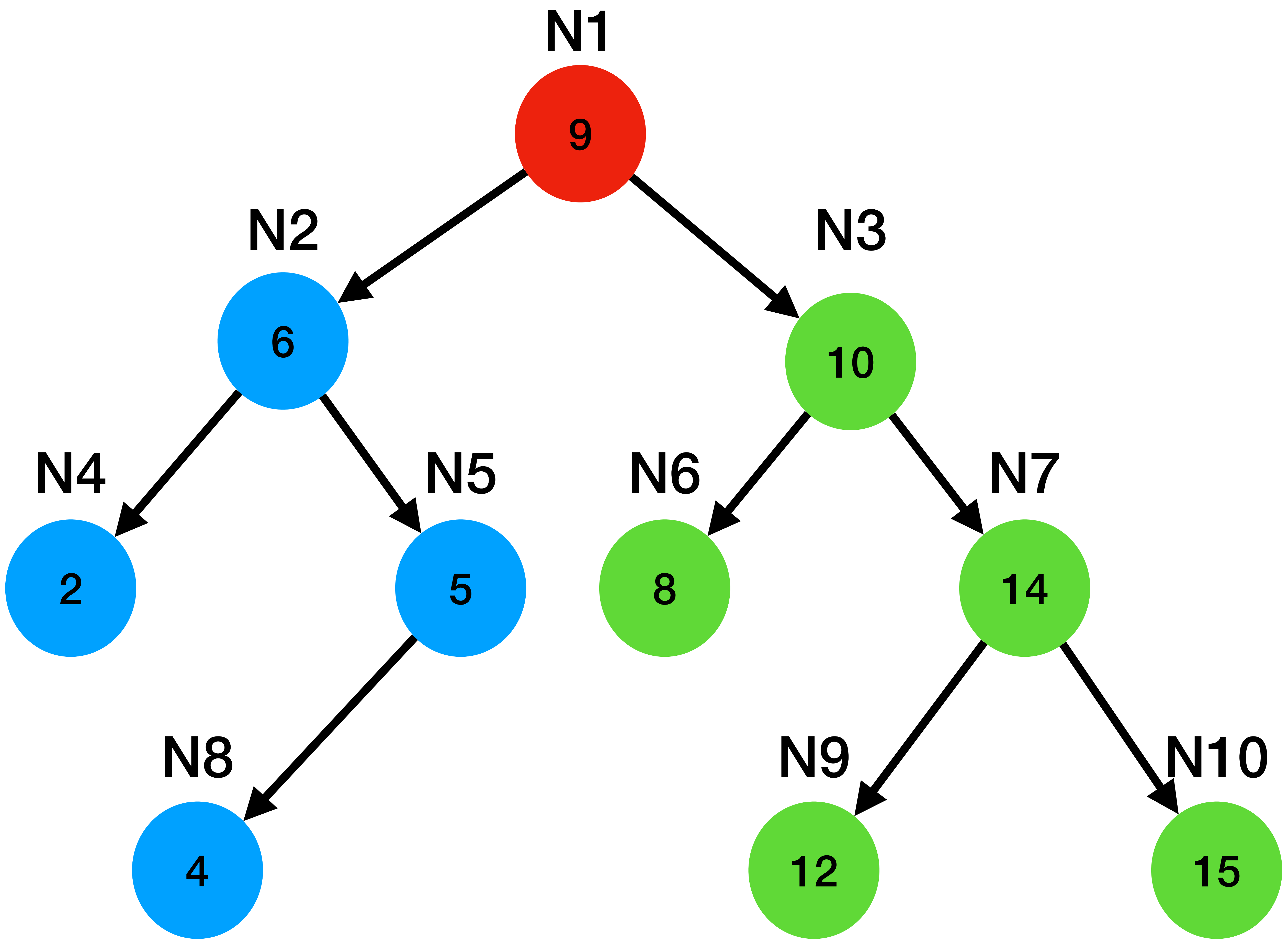
Depth-first



R - LT - RT	LT - R - RT	LT - RT - R
Pre-order	**In-order**	**Post-order**
N1		
N2		
N4		
N5		
N8		
N3		
N6		
N7		
N9		
N10		

Pre-order traversal

Root -> Left sub-tree -> Right sub-tree



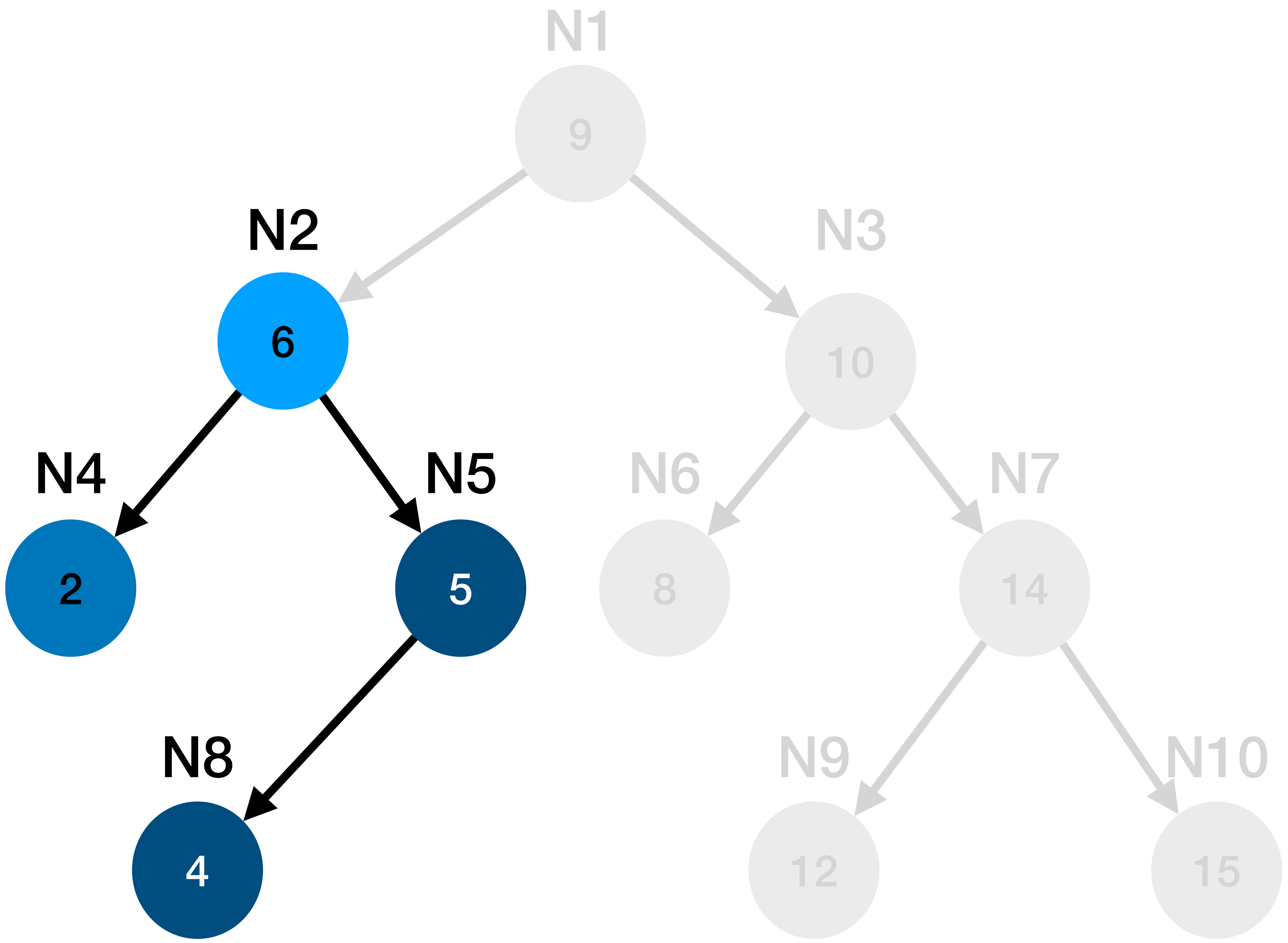
Consider the full tree

Step	Result
Find root	N1
Find Left Sub-tree	Blue Group
Find Right Sub-tree	Green Group

Order:
N1

Pre-order traversal

Root -> Left sub-tree -> Right sub-tree



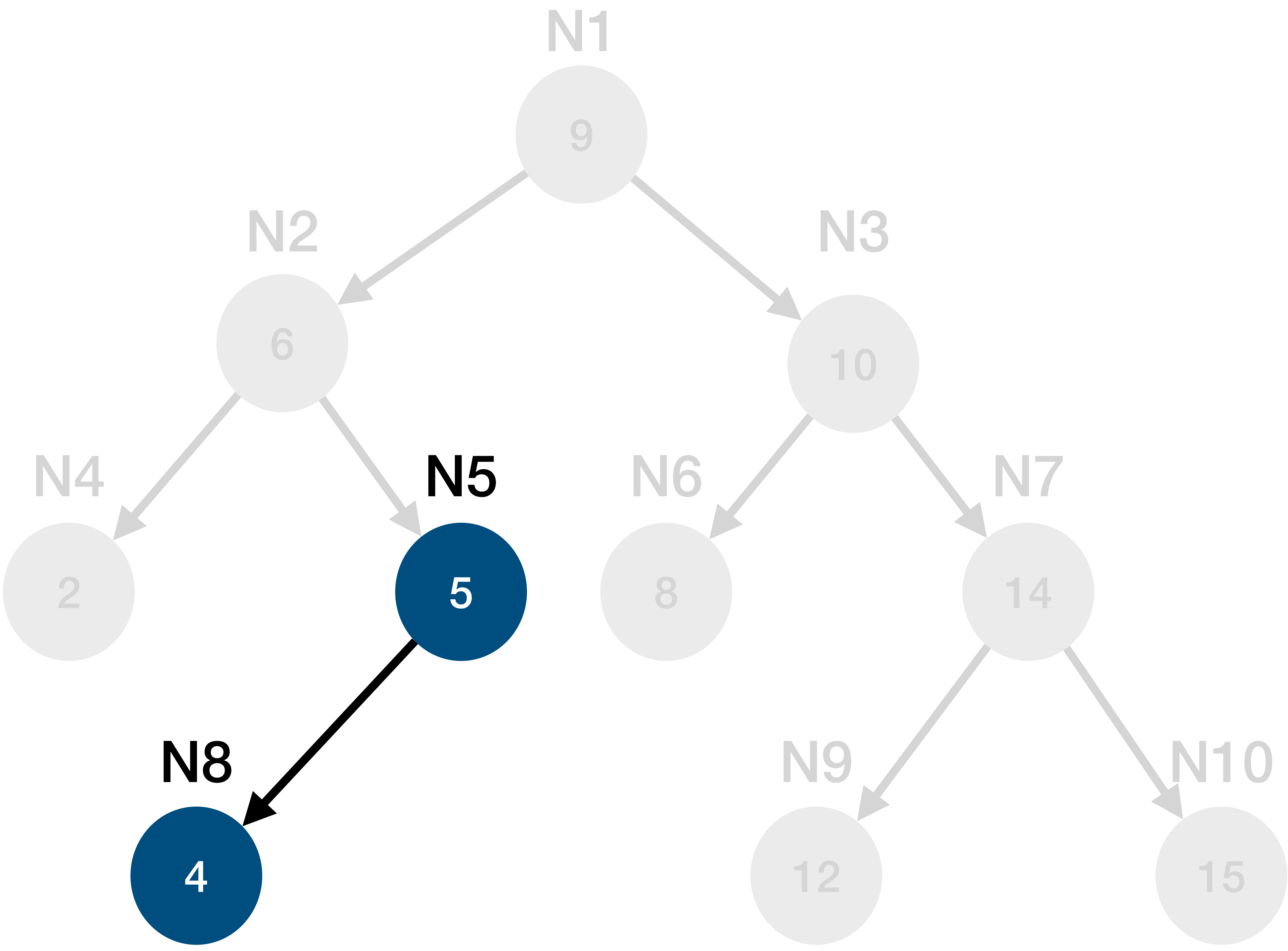
Consider Left Sub-tree (Blue Group)

Step	Result
Find root	N2
Find Left Sub-tree	N4
Find Right Sub-tree	Dark blue group

Order:
N1 -> N2 -> N4

Pre-order traversal

Root -> Left sub-tree -> Right sub-tree



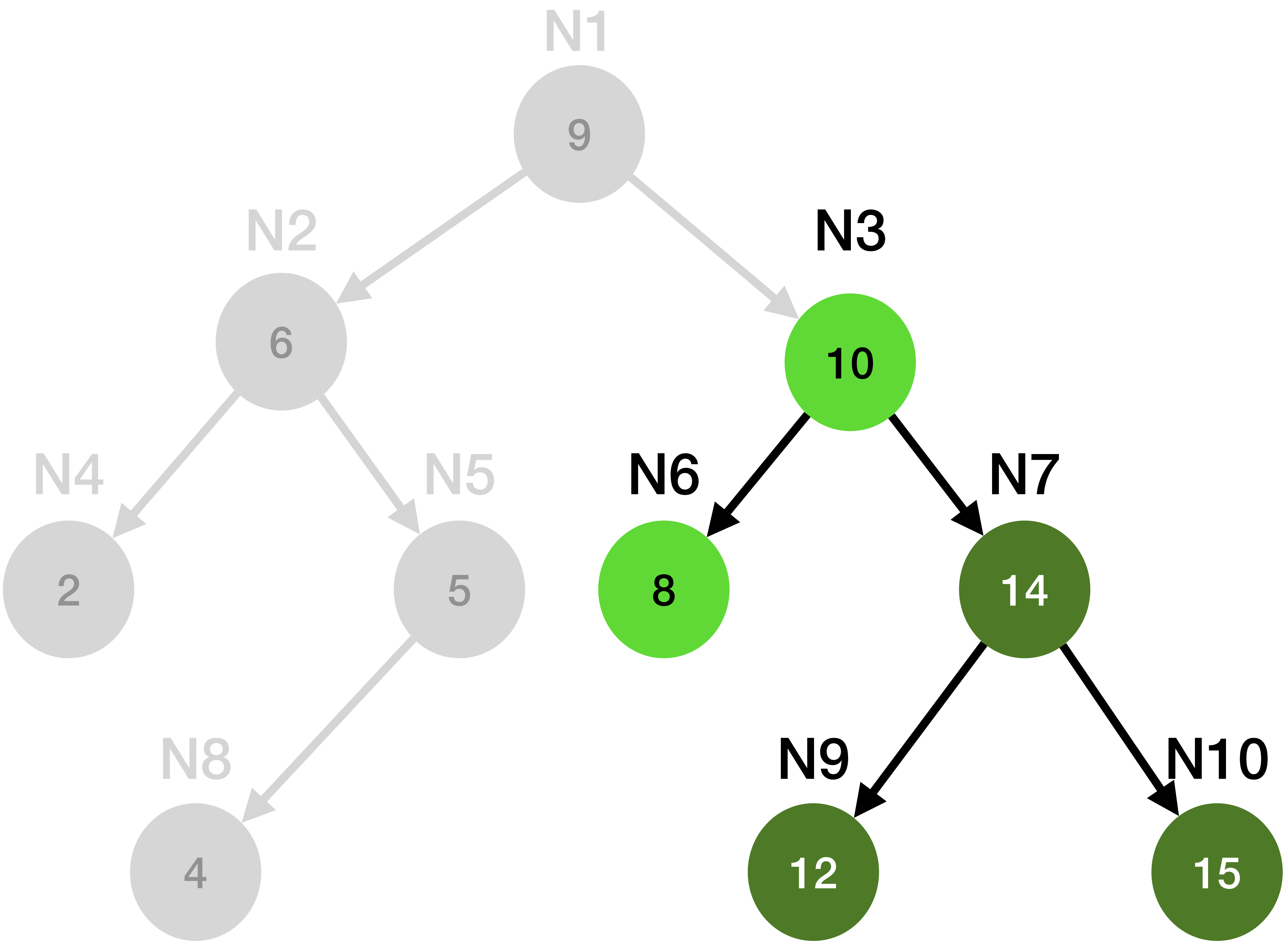
Consider Right Sub-tree (Dark blue group)

Step	Result
Find root	N5
Find Left Sub-tree	N8
Find Right Sub-tree	-

Order:
N1 -> N2 -> N4 -> N5 -> N8

Pre-order traversal

Root -> Left sub-tree -> Right sub-tree



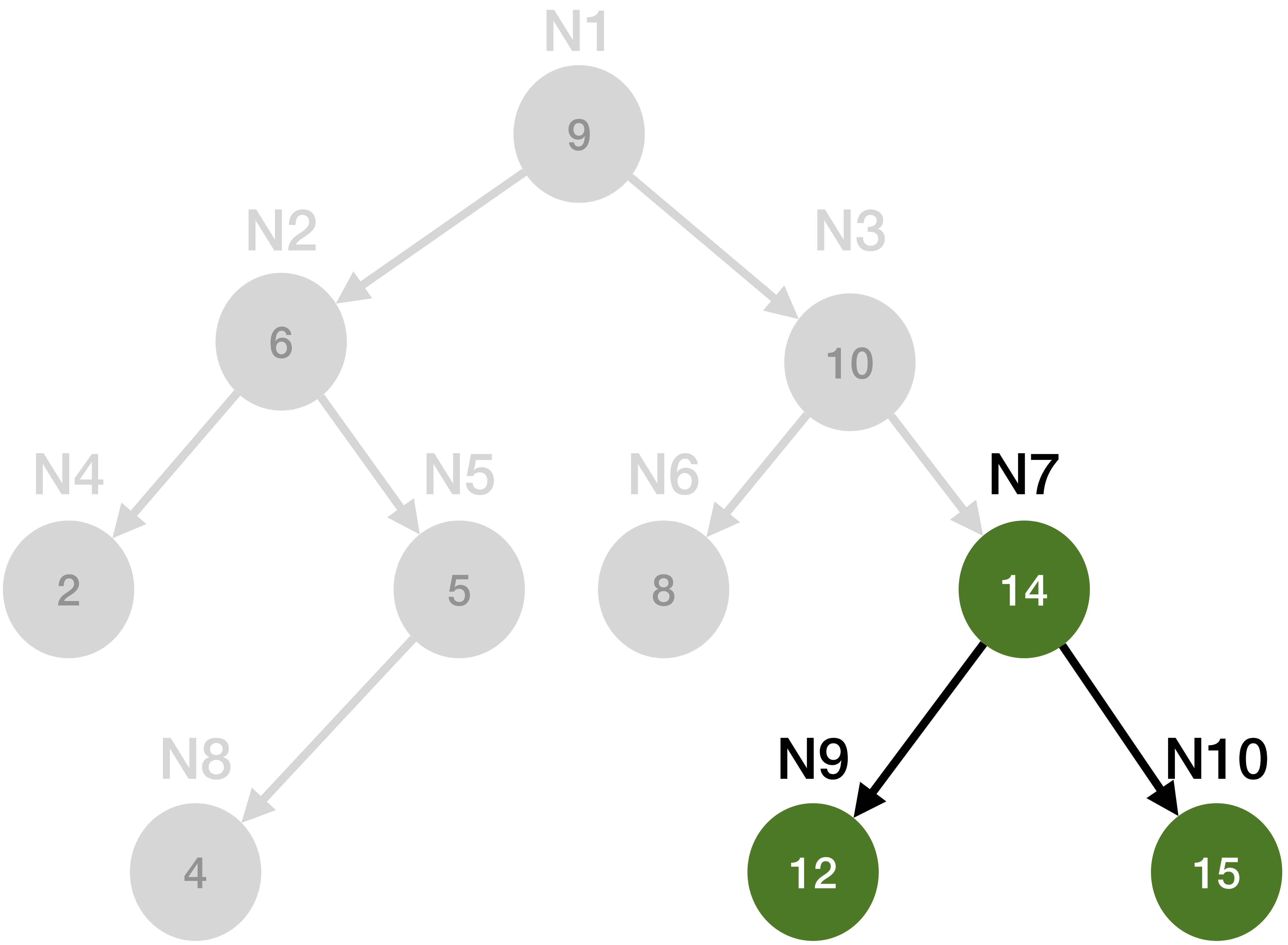
Consider Right Sub-tree (Green Group)

Step	Result
Find root	N3
Find Left Sub-tree	N6
Find Right Sub-tree	Dark Green Group

Order:
N1 -> N2 -> N4 -> N5 -> N8 -> N3-> N6

Pre-order traversal

Root -> Left sub-tree -> Right sub-tree



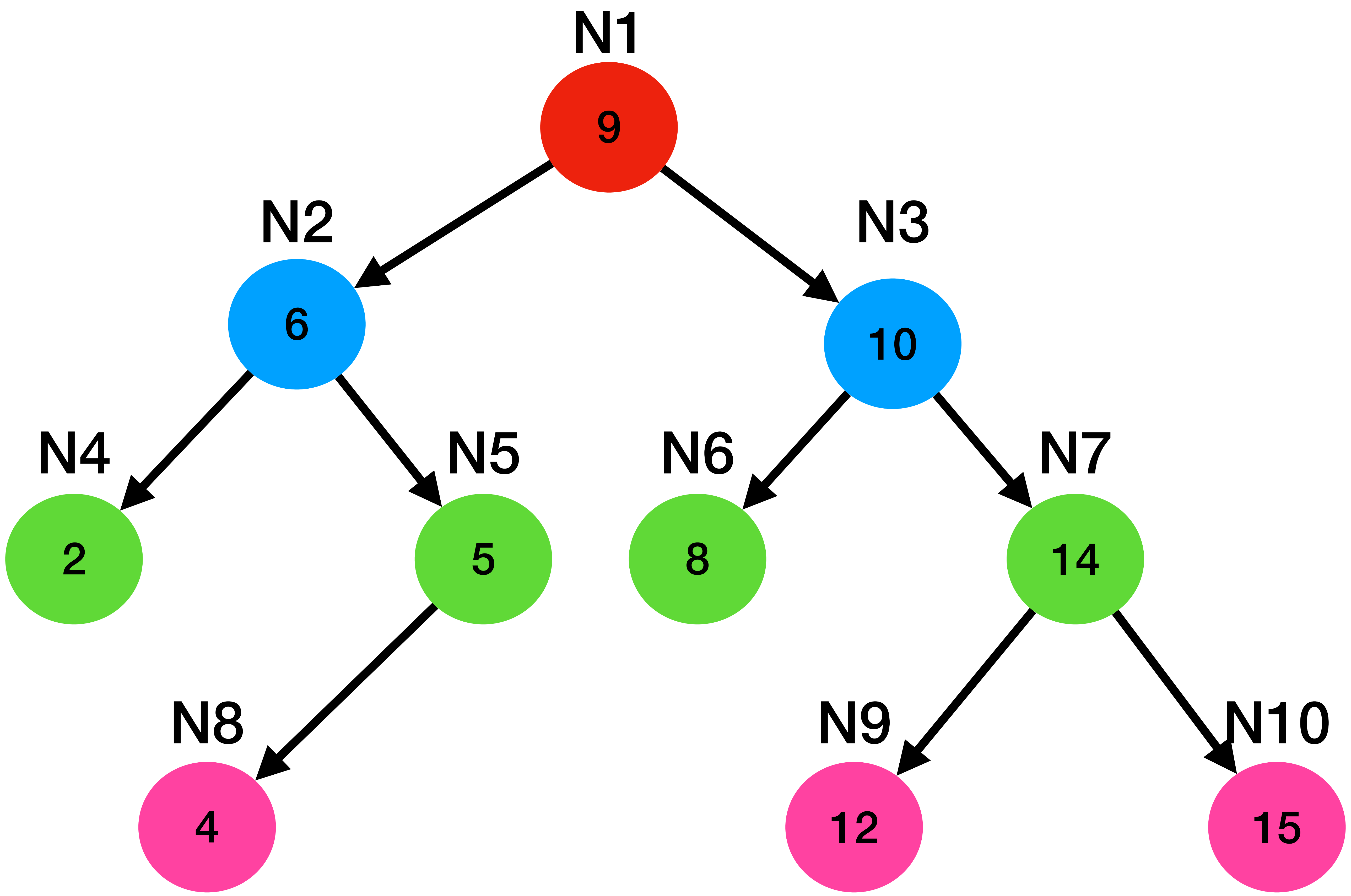
Consider Right Sub-tree (Dark Green Group)

Step	Result
Find root	N7
Find Left Sub-tree	N9
Find Right Sub-tree	N10

Order:
N1 -> N2 -> N4 -> N5 -> N8 -> N3-> N6 -> N7 -> N9 -> N10

Tree Traversal

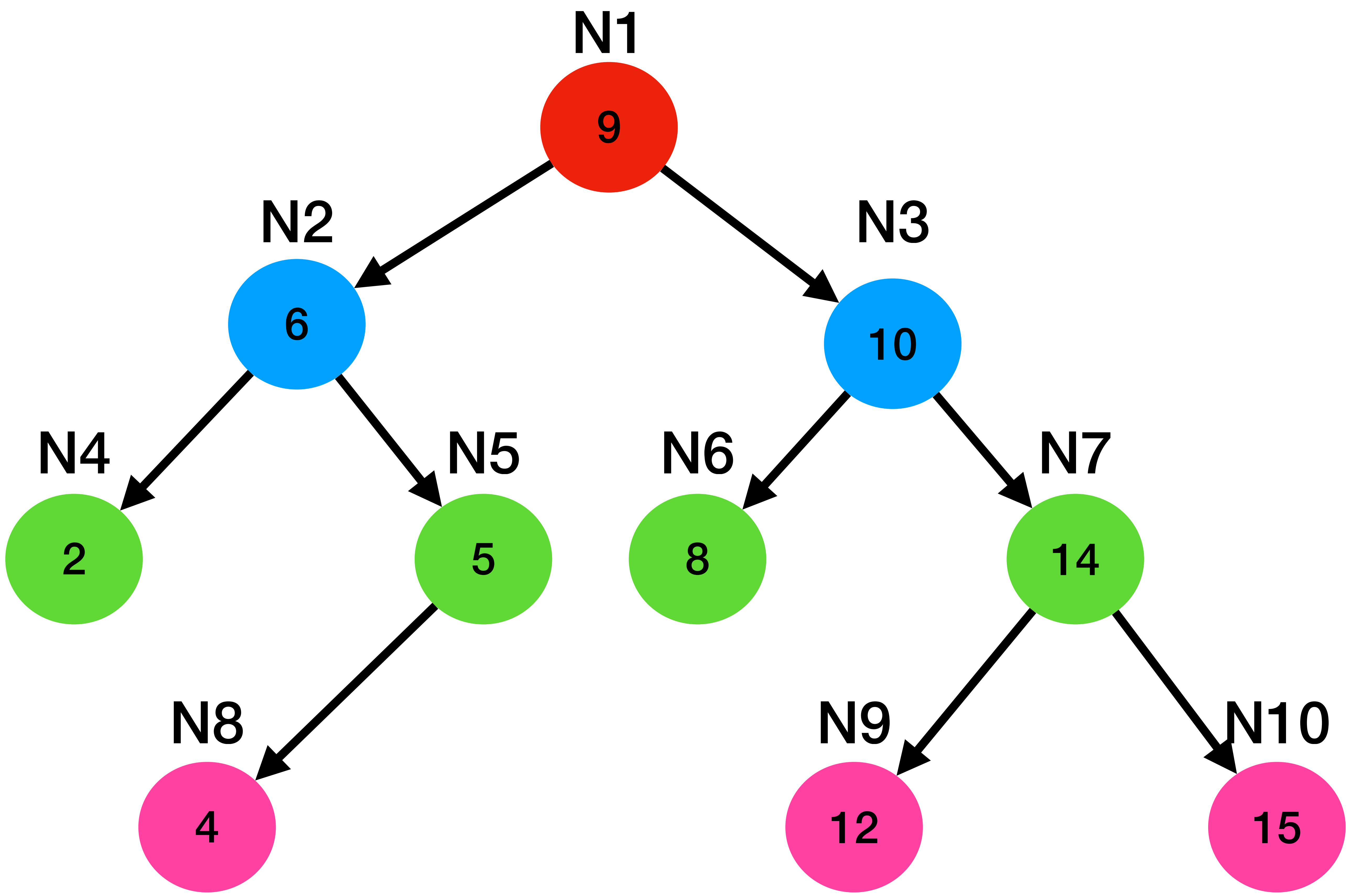
Depth-first



R - LT - RT	LT - R - RT	LT - RT - R
Pre-order	**In-order**	**Post-order**
N1	N4	
N2	N2	
N4	N8	
N5	N5	
N8	N1	
N3	N6	
N6	N3	
N7	N9	
N9	N7	
N10	N10	

Tree Traversal

Depth-first



R - LT - RT	LT - R - RT	LT - RT - R
Pre-order	**In-order**	**Post-order**
N1	N4	N4
N2	N2	N8
N4	N8	N5
N5	N5	N2
N8	N1	N6
N3	N6	N9
N6	N3	N10
N7	N9	N7
N9	N7	N3
N10	N10	N1

Tree Traversal

Depth-first

tree.c

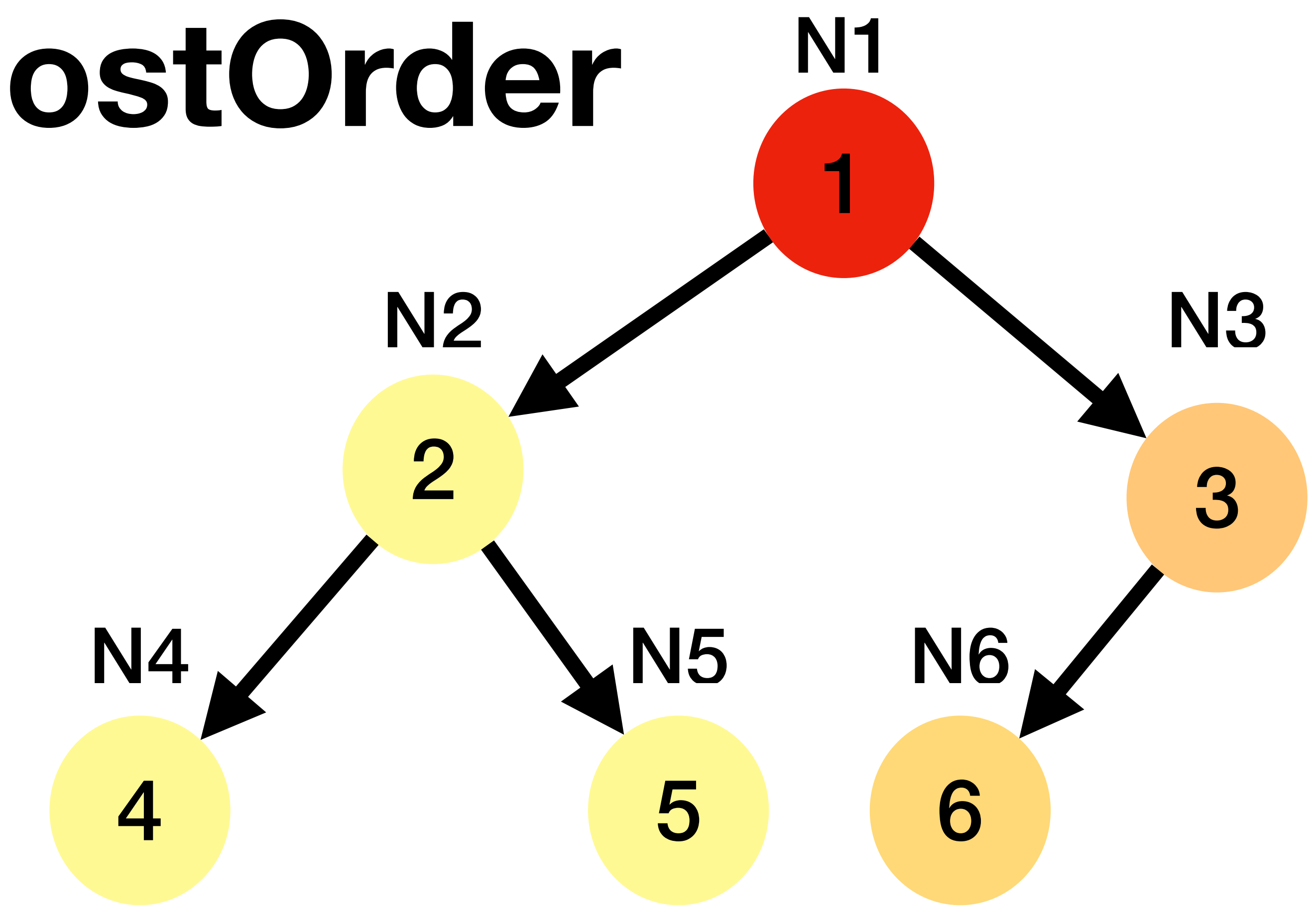
```
void postOrder (TREENODE_T* node)
{
   if (node->left != NULL)
      postOrder(node->left);
   if (node->right != NULL)
      postOrder(node->right);
   printf("access %d\n", node->data);
}
```

main.c

```
TREENODE_T* root = N1;
postOrder(root);
```

Recursive Function

postOrder



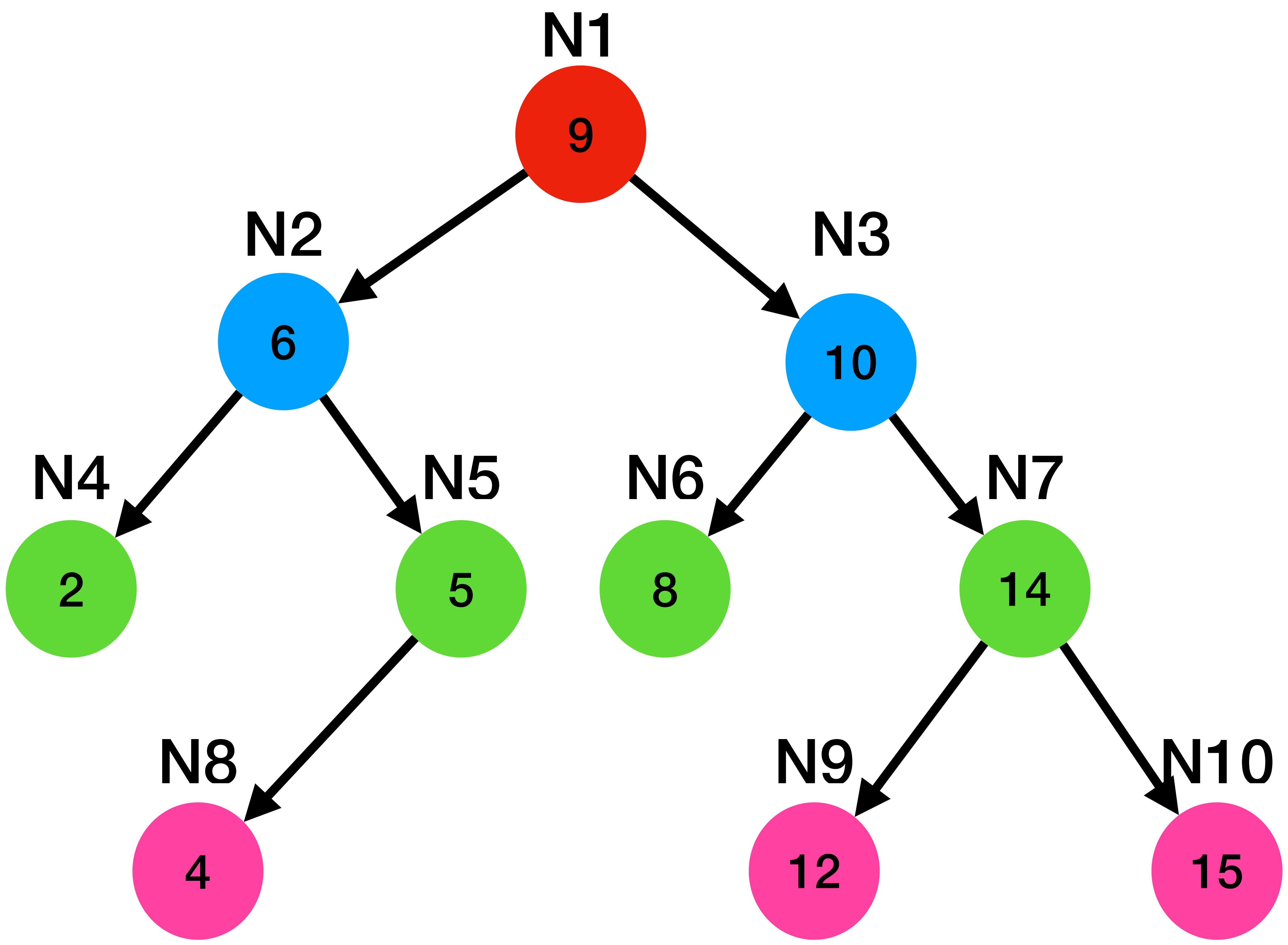

```
1 void postOrder (TREENODE_T* node)
2 {
3 if (node->left != NULL)
4   postOrder(node->left);
5 if (node->right != NULL)
6   postOrder(node->right);
7 printf("access %d\n", node->data);
8 }
```

```
TREENODE_T* root = N1;
postOrder(N1);
```

Step	Current Node	Action	Pass	End function
1	N1	Recursion#1 at Line 4	N1->left (N2)	No
2	N2	Recursion#2 at Line 4	N2->left (N4)	No
3	N4	Print access 4	-	Back to Step2 Line5
4	N2	Recursion#3 at Line 6	N2->right (N5)	No
5	N5	Print access 5	-	Back to Step2 Line7
6	N2	Print access 2	-	Back to Step1 Line5
7	N1	Recursion#4 at Line 6	N1->right (N3)	No
8	N3	Recursion#5 at Line 4	N3->left (N6)	No
9	N6	Print access 6	-	Back to Step8 Line5
10	N3	Print access 3	-	Back to Step7 Line7
11	N1	Print access 1	-	Back to Main

Tree Traversal

Breadth-first



- **Root -> Leaf L -> Leaf R**

Dequeue	PTR	Enqueue	Queue
Start	N1	N2, N3	N2, N3
N2	N2	N4, N5	N3, N4, N5
N3	N3	N6, N7	N4, N5, N6, N7
N4	N4	-	N5, N6, N7
N5	N5	N8	N6, N7, N8
N6	N6	-	N7, N8

Tree Traversal

Construct a binary tree from traversal results

- Require at least 2 traversal results
  - **In-order** > left & right child node
  - **Pre-order or post-order** > root node

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

Tree Traversal

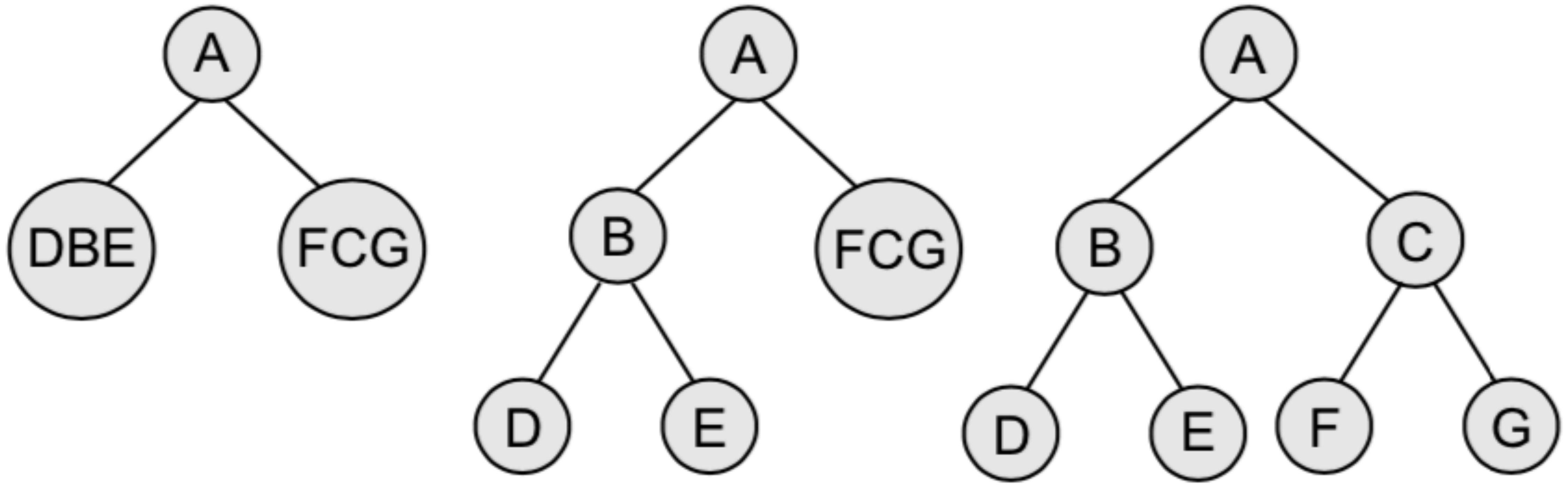
Construct a binary tree from traversal results

STEP1 - Use pre-order/post-order to determine the root (first/last).

STEP2 - Use in-order to determine the left & right subtree of the root.

STEP3 - Recursively select each element from STEP1 to be the root of the subtree.

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



Wrap up

- Hierarchical data structure
- Terminologies
- Binary tree
- Tree traversal
    - Depth-first search: pre-order, in-order, post-order
    - Breadth-first search

Coming in Part 2…

- Tree structure
- Insertion based on conditions
- Binary search
- Node deletion