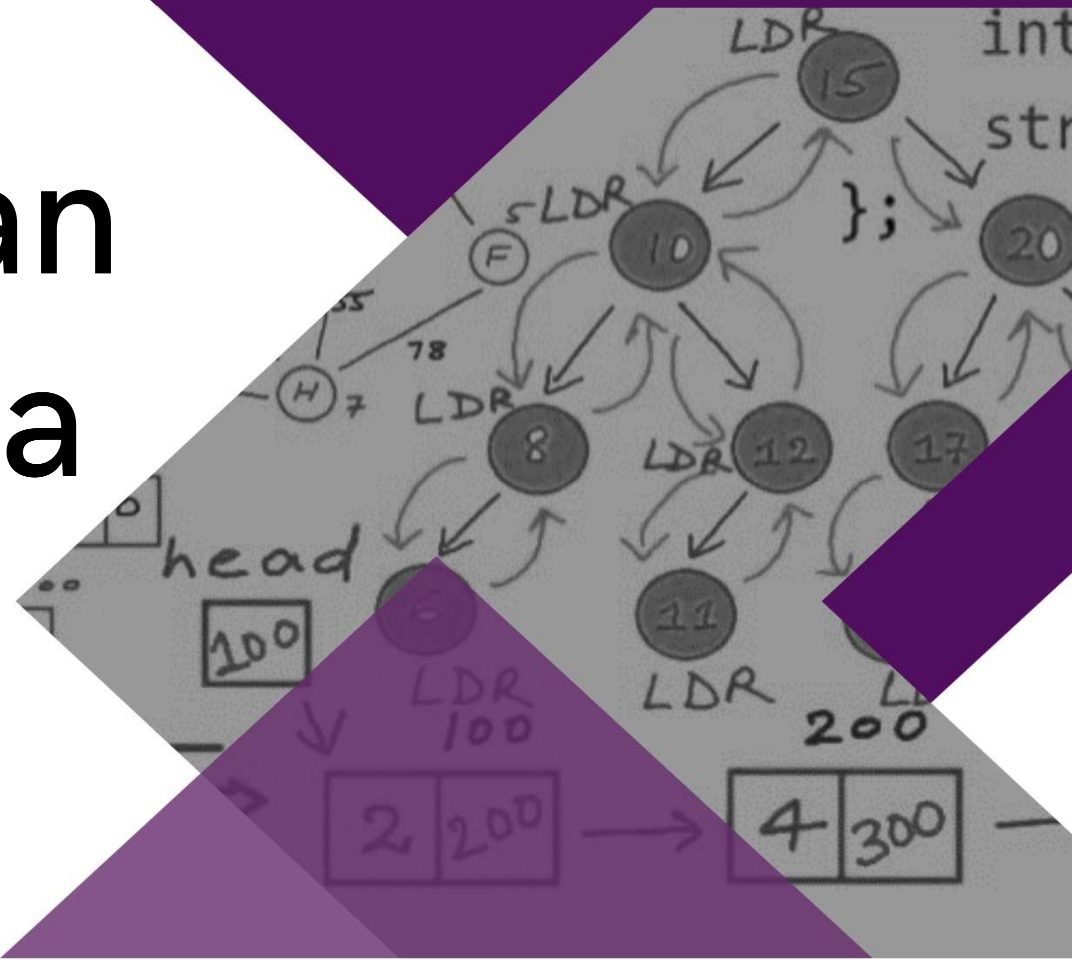
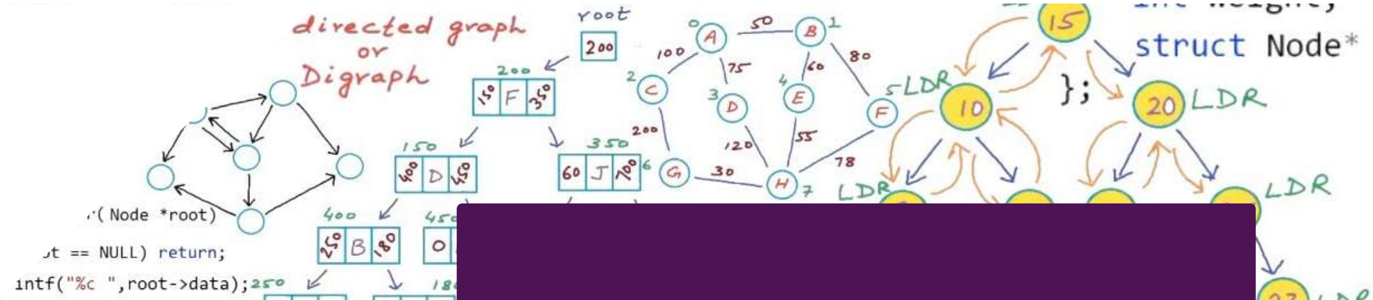
Algoritma dan Struktur Data



Pekan 11

- Array dan Struct
- Pointer
- Alokasi Memori
- List
- Varian *Linked List*
- Stack dan Queue
- Studi Kasus
- UTS
- Tree
- Non-Binary Tree
- **Binary Tree**
- Extended Tree
- Advanced Tree
- Heap
- Hash Table
- UAS

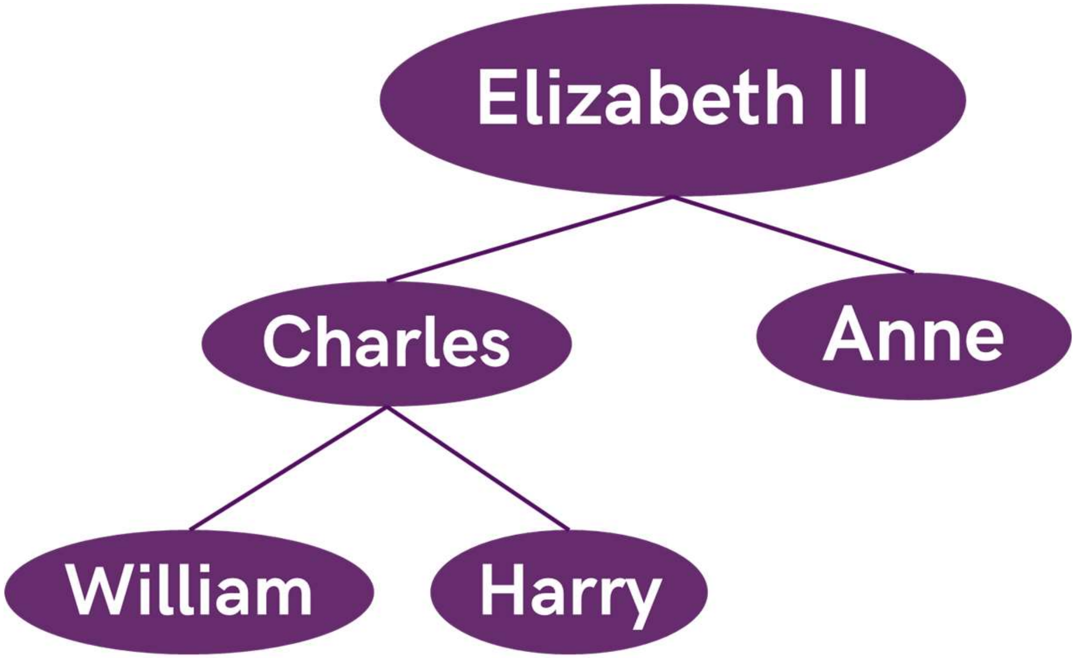
Tujuan



1	Mahasiswa memahami perbedaan non-binary dan binary tree
2	Mahasiswa mampu mengimplementasikan ADT Tree menggunakan array
3	Mahasiswa mampu mengimplementasikan ADT Tree menggunakan linked list

Binary Tree

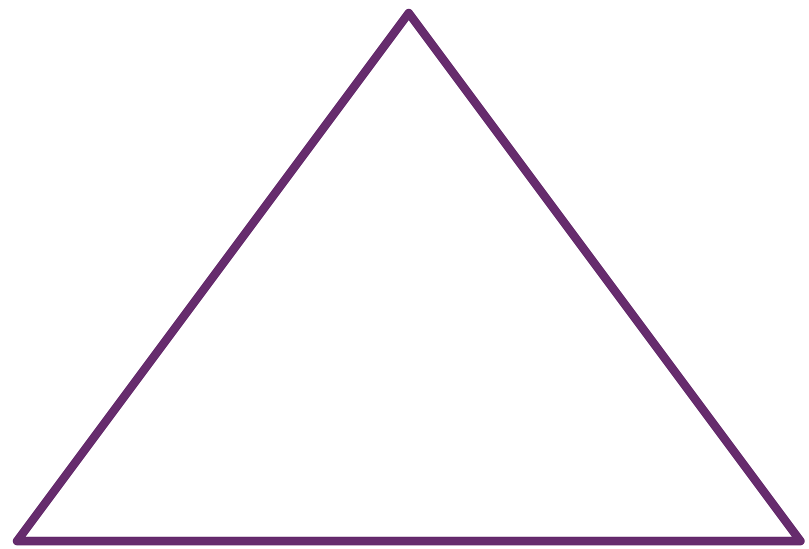
Binary Tree



Any parent can have **at most** two children

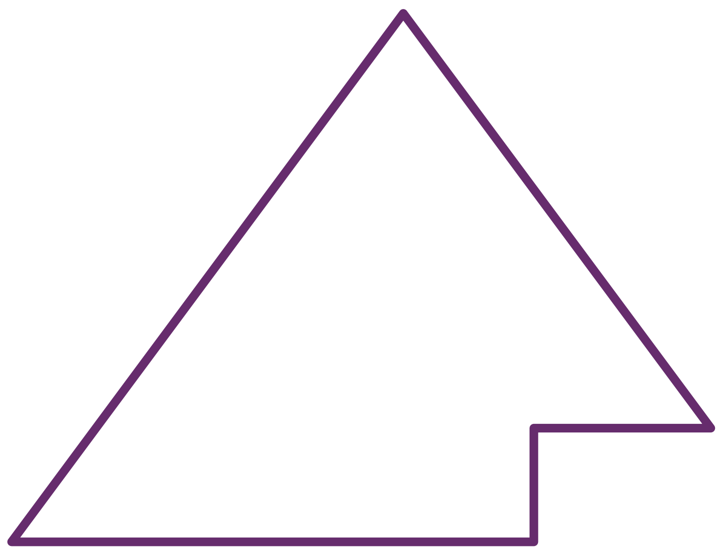
Full Binary Tree

A binary tree in which all the leaves are located on the same level and every non-leaf node has two children



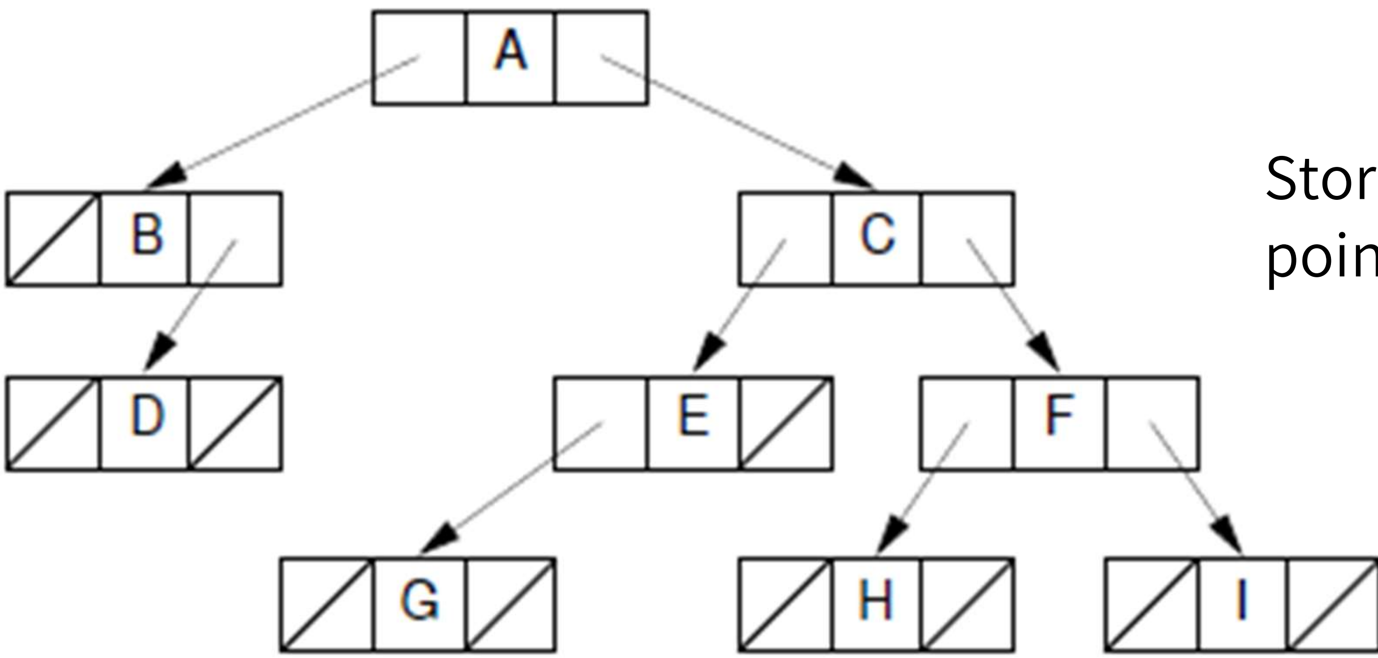
Complete Binary Tree

A binary tree that is either full or full through the next-to-last level, with the leaves on the last level located as far to the left as possible



Implementasi Menggunakan Linked List

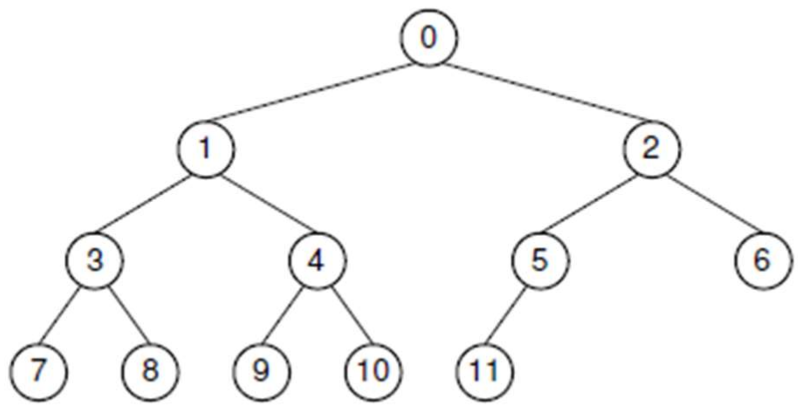
Pointer-Based Node



Store a linked list of child pointers with each node.

Implementasi Menggunakan Array

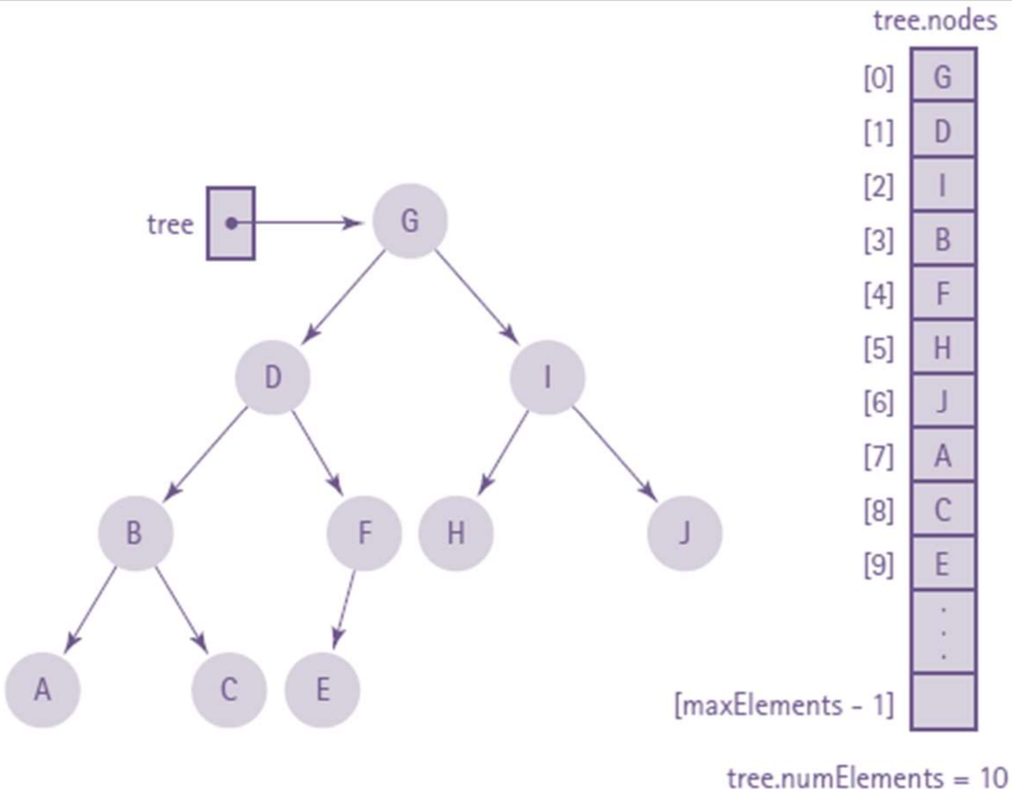
Array Implementation



(a)

Position	0	1	2	3	4	5	6	7	8	9	10	11
Parent	–	0	0	1	1	2	2	3	3	4	4	5
Left Child	1	3	5	7	9	11	–	–	–	–	–	–
Right Child	2	4	6	8	10	–	–	–	–	–	–	–
Left Sibling	–	–	1	–	3	–	5	–	7	–	9	–
Right Sibling	–	2	–	4	–	6	–	8	–	10	–	–

Array Implementation



- Parent(r) = $\left\lfloor \frac{(r-1)}{2} \right\rfloor$ if $r \neq 0$.
- Left child(r) = $2r + 1$ if $2r + 1 < n$.
- Right child(r) = $2r + 2$ if $2r + 2 < n$.
- Left sibling(r) = $r - 1$ if r is even.
- Right sibling(r) = $r + 1$ if r is odd and $r + 1 < n$.

Binary Search Tree

Binary Search Tree (BST)

A binary tree in which the key value in any node is greater than the key value in its left child and any of its left children (the node in the left subtree) and less than the key value in its right child and any of its children (the nodes in the right subtree)

BST Operations

isFull
IsEmpty
LengthIs

RetrieveItem
InsertItem
DeleteItem

CopyTree
ResetTree
GetNextItem

BST Operations - isFull

```
bool isFull()
{
    tree_t* location;
    try
    {
        location = new tree_t;
        delete location;
        return false;
    }
    catch (bad_alloc)
    {
        return true;
    }
}
```

BST Operations - isEmpty

```
bool isEmpty(tree_t* tree)
{
    return tree == nullptr;
}
```

BST Operations - LengthIs

```
int lengthIs(tree_t* tree)
{
    if (tree == nullptr)
        return 0;
    else
        return 1 + lengthIs(tree->left) + lengthIs(tree->right);
}
```

BST Operations – RetrieveItem

```
void retrieveItem(char& item, bool& found, tree_t* tree)
{
    if (tree == nullptr)
        found = false;
    else if (item < tree->info)
        retrieveItem(item, found, tree->left);
    else if (item > tree->info)
        retrieveItem(item, found, tree->right);
    else
    {
        found = true;
        item = tree->info;
    }
}
```

BST Operations – InsertItem

```
void insertItem(char item, tree_t*& tree)
{
    if (tree == nullptr)
    {
        tree = new TreeNode;
        tree->info = item;
        tree->left = nullptr;
        tree->right = nullptr;
    }
    else if (item < tree->info)
        insertItem(item, tree->left);
    else
        insertItem(item, tree->right);
}
```

BST Operations – DeleteItem

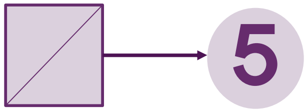
```
void deleteItem(char item, tree_t*& tree)
{
    if (tree == nullptr)
        return;

    if (item < tree->info)
        deleteItem(item, tree->left);
    else if (item > tree->info)
        deleteItem(item, tree->right);
    else
    {
        deleteNode(tree);
    }
}
```

Insert

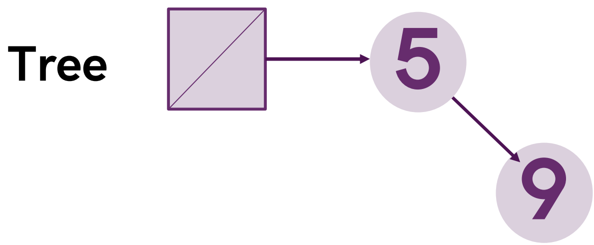
Insert 5

Tree



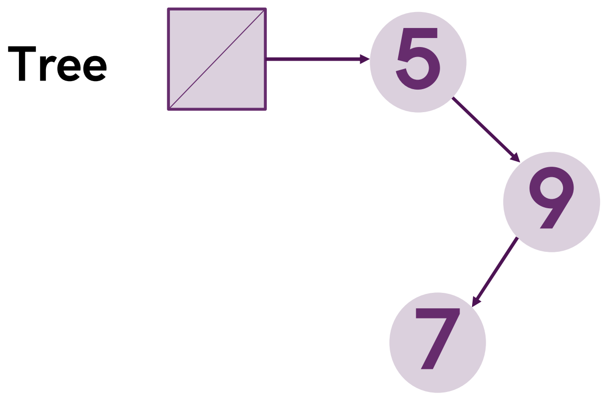
Insert

Insert 9



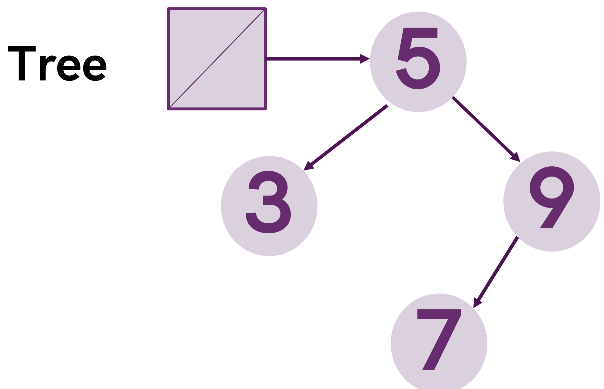
Insert

Insert 7



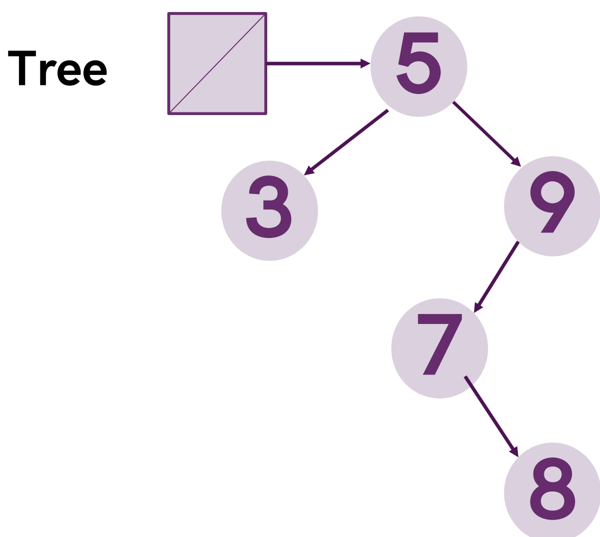
Insert

Insert 3



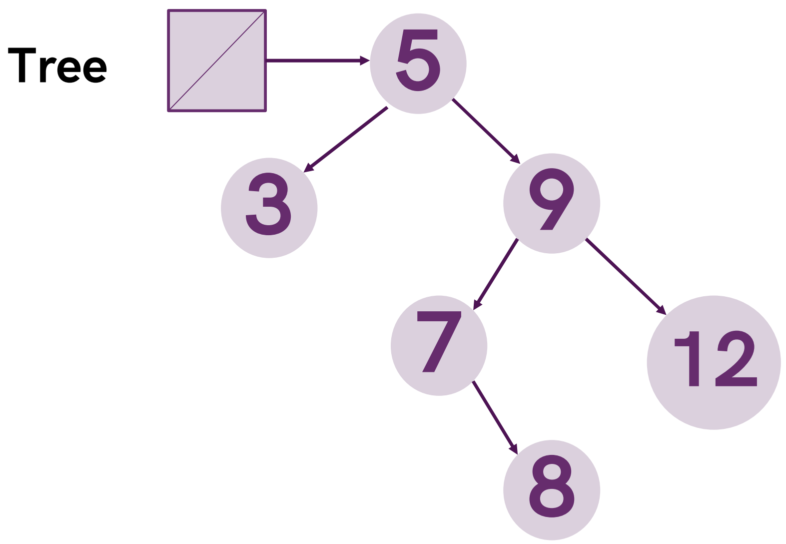
Insert

Insert 8



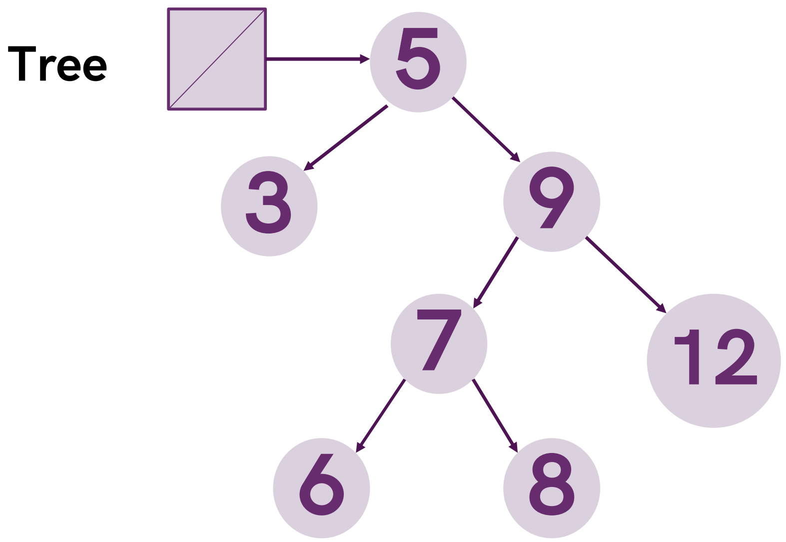
Insert

Insert 12



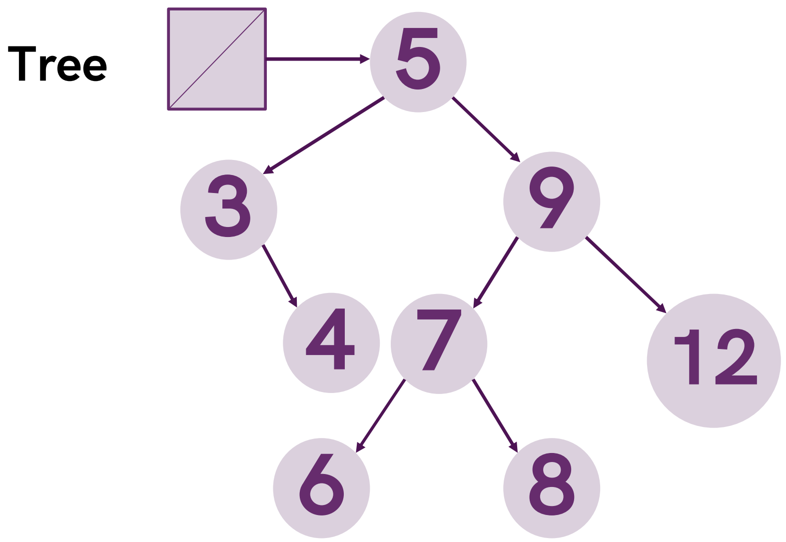
Insert

Insert 6

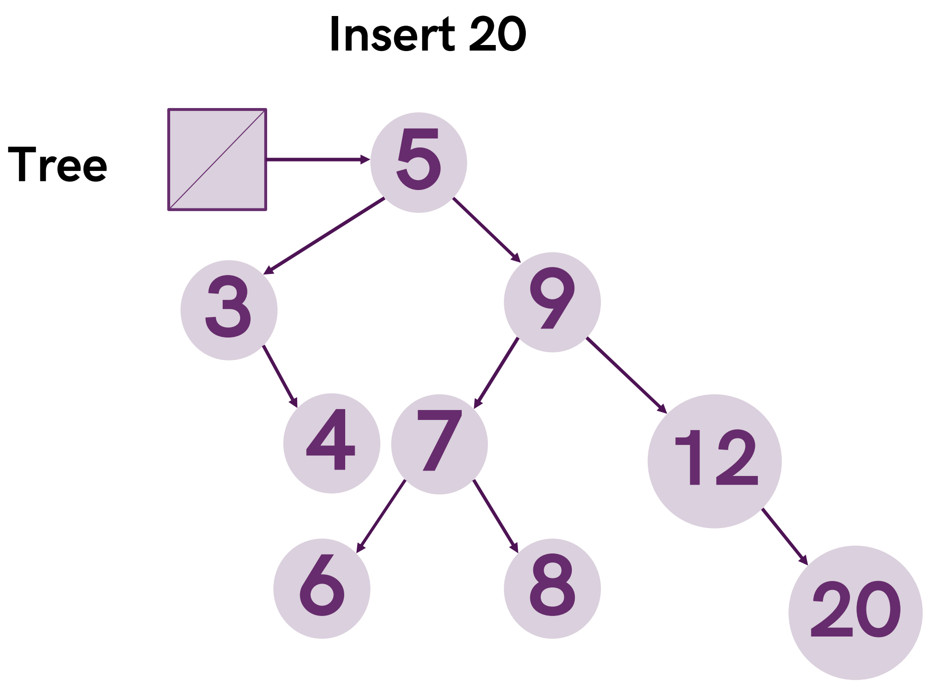


Insert

Insert 4



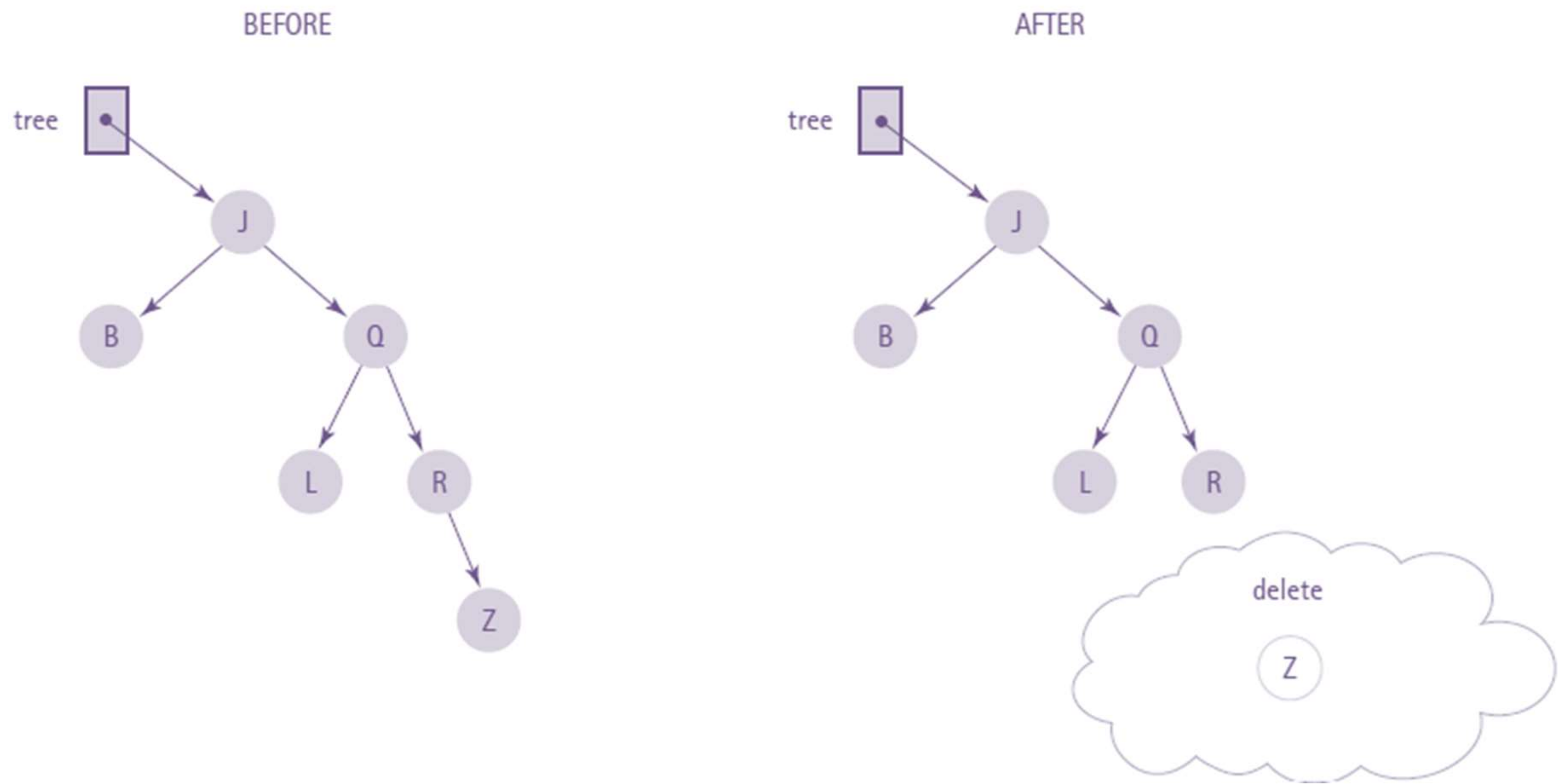
Insert



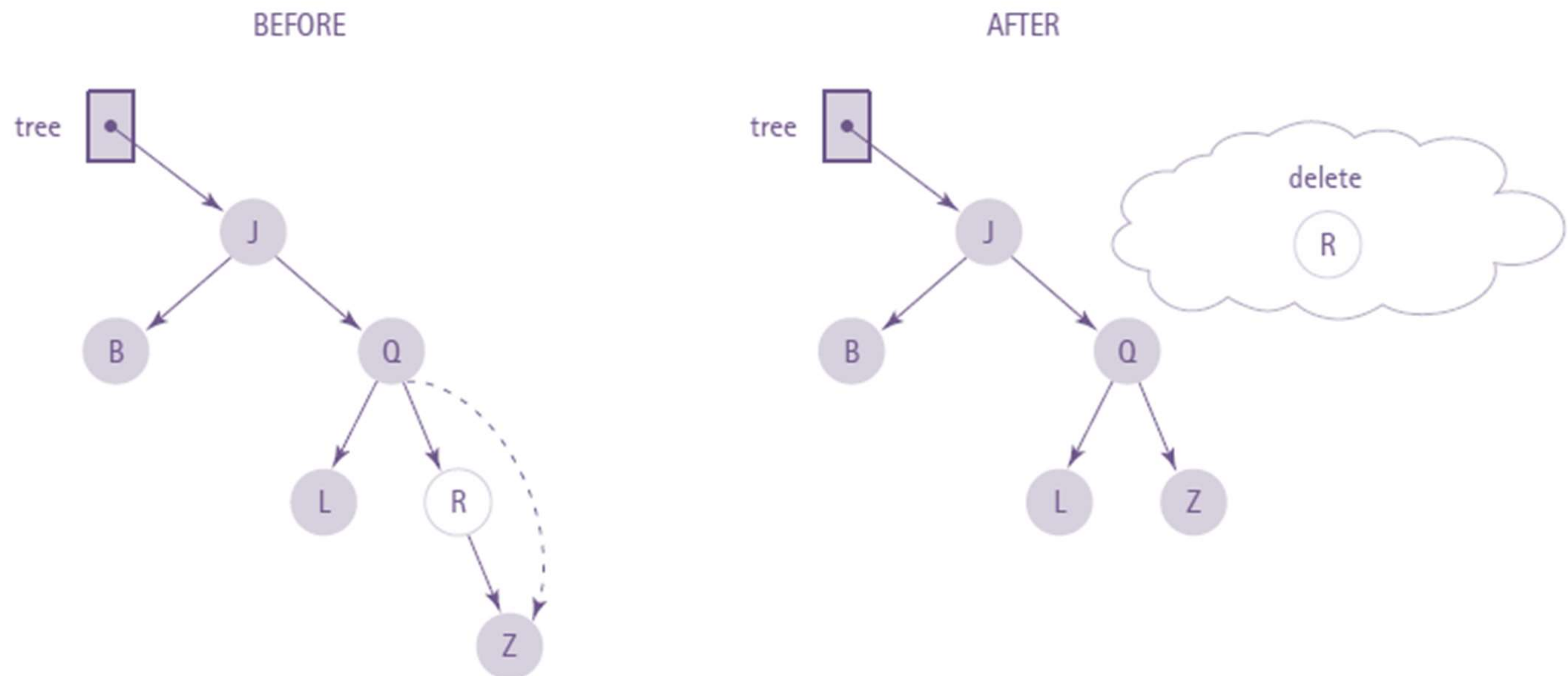
Delete

- Deleting a leaf (no children)
- Deleting a node with only one child
- Deleting a node with two children

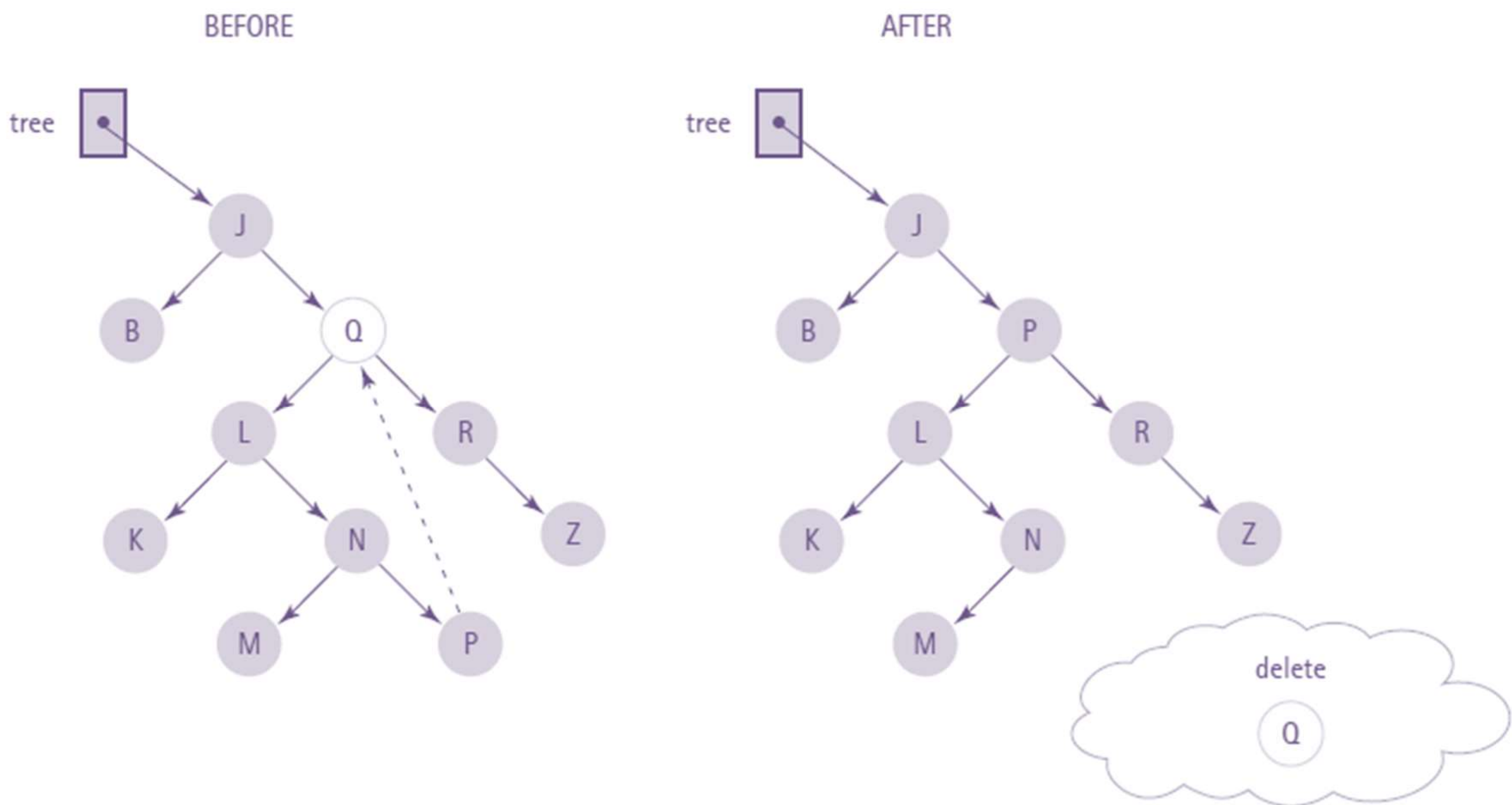
Delete a leaf



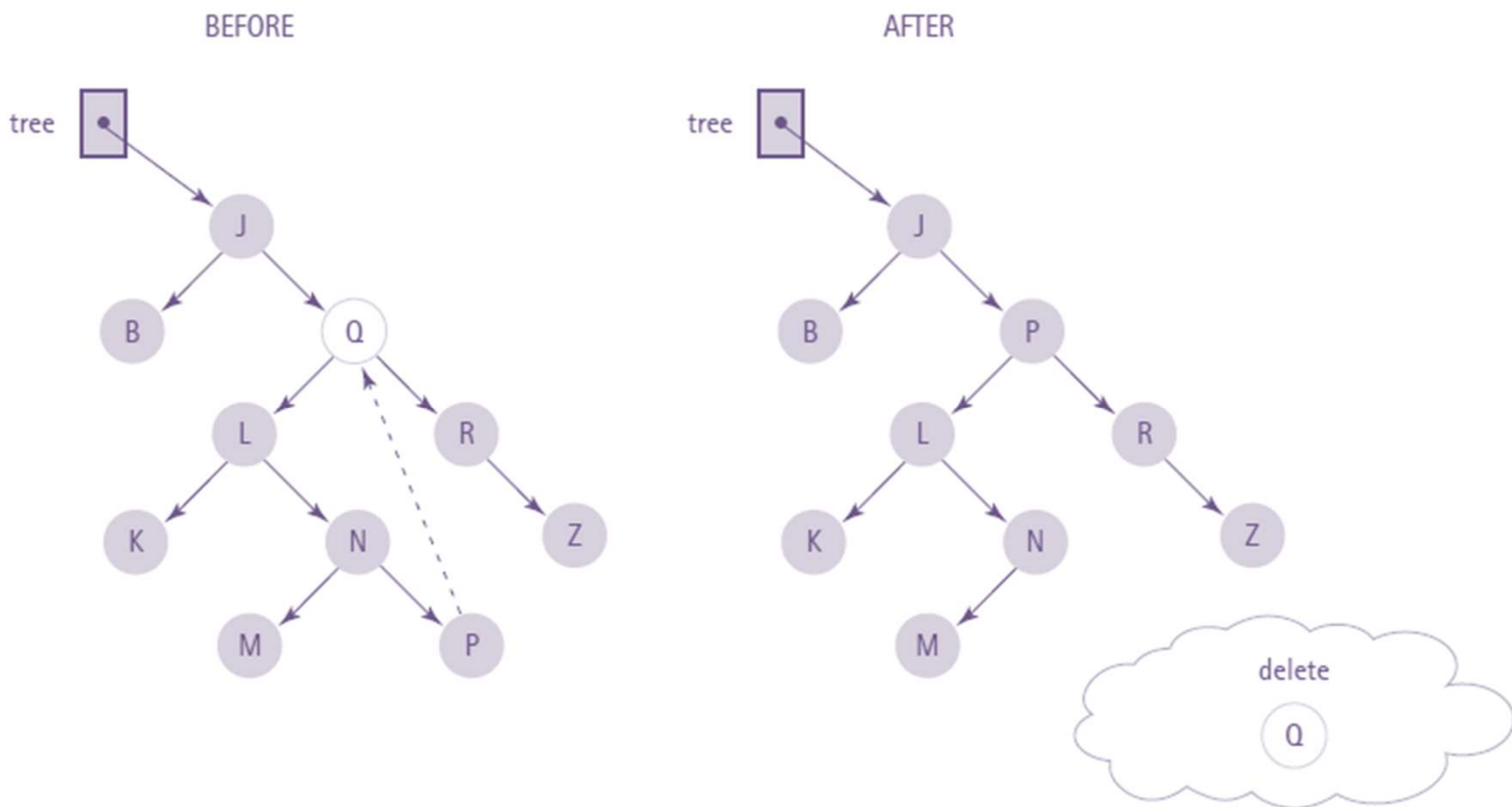
Delete a node with only one child



Delete a node with two child



Print Tree



Insert Algorithm

InsertItem

Create a node to contain the new item.
Find the insertion place.
Attach the new node.

Delete Algorithm

```
DeleteNode
if (Left(tree) is NULL) AND (Right(tree) is NULL)
    Set tree to NULL
else if Left(tree) is NULL
    Set tree to Right(tree)
else if Right(tree) is NULL
    Set tree to Left(tree)
else
    Find predecessor
    Set Info(tree) to Info(predecessor)
    Delete predecessor
```

Kompleksitas Algoritma

	Binary Search Tree	Array-Based Linear List	Linked List
Class constructor	O(1)	O(1)	O(1)
Destructor	O(N)	O(1)*	O(N)
`MakeEmpty`	O(N)	O(1)*	O(N)
`LengthIs`	O(N)	O(1)	O(1)
`IsFull`	O(1)	O(1)	O(1)
`IsEmpty`	O(1)	O(1)	O(1)
`RetrieveItem`			
Find	O($\log_2 N$)	O($\log_2 N$)	O(N)
Process	O(1)	O(1)	O(1)
Total	O($\log_2 N$)	O($\log_2 N$)	O(N)
`InsertItem`			
Find	O($\log_2 N$)	O($\log_2 N$)	O(N)
Process	O(1)	O(N)	O(1)
Total	O($\log_2 N$)	O(N)	O(N)
`DeleteItem`			
Find	O($\log_2 N$)	O($\log_2 N$)	O(N)
Process	O(1)	O(N)	O(1)
Total	O($\log_2 N$)	O(N)	O(N)

*If the items in the array-based list could possibly contain pointers, the items must be deallocated, making this an O(N) operation.