CSCI 3230 Data Structures

Trees

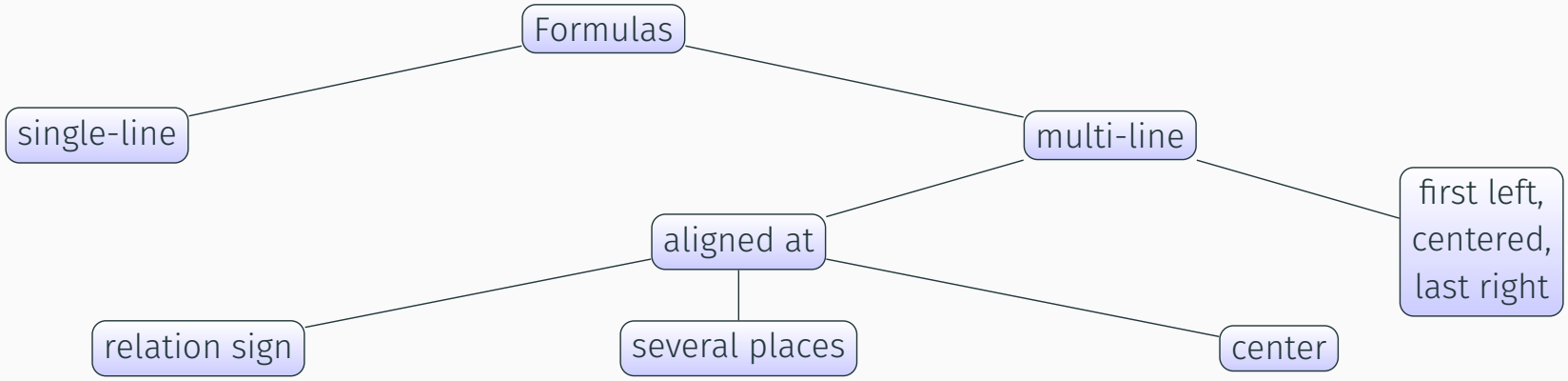
Weitian Tong, Ph.D.

Department of Computer Science
Georgia Southern University
Website: www.weitiantong.com
Email:wtong@georgiasouthern.edu

Table of contents

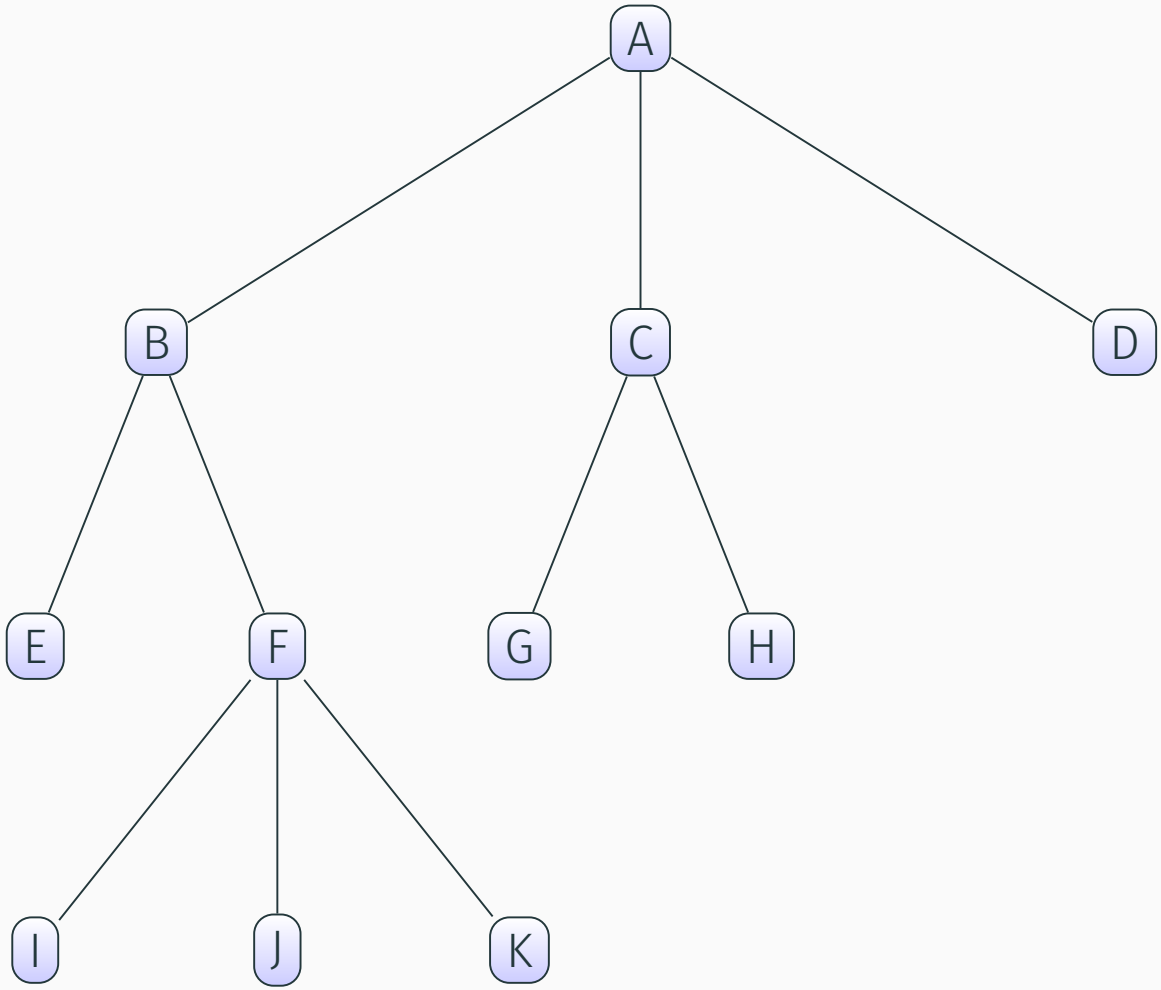
What is a tree?

What is a tree?



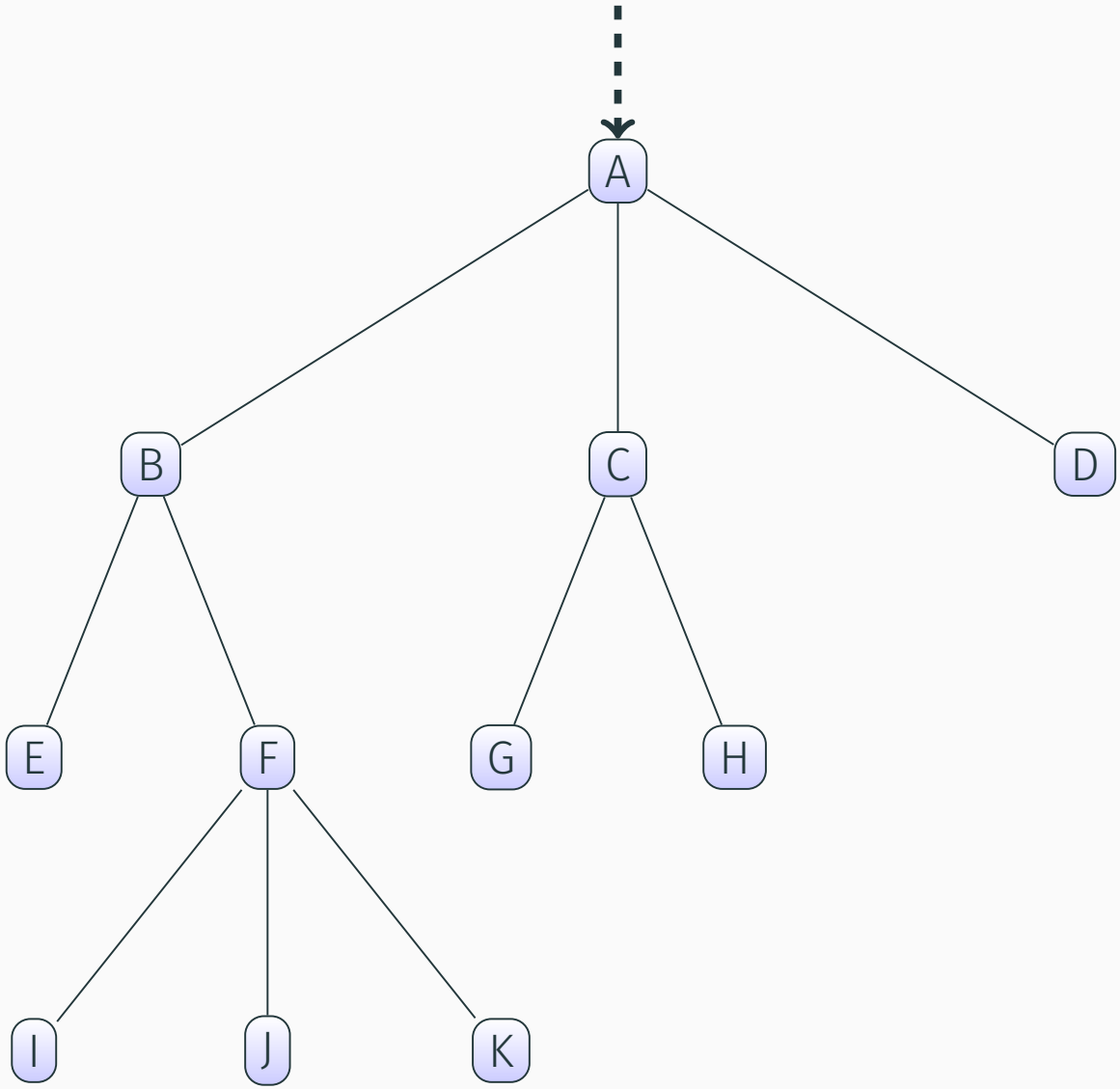
- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments

Tree Terminology

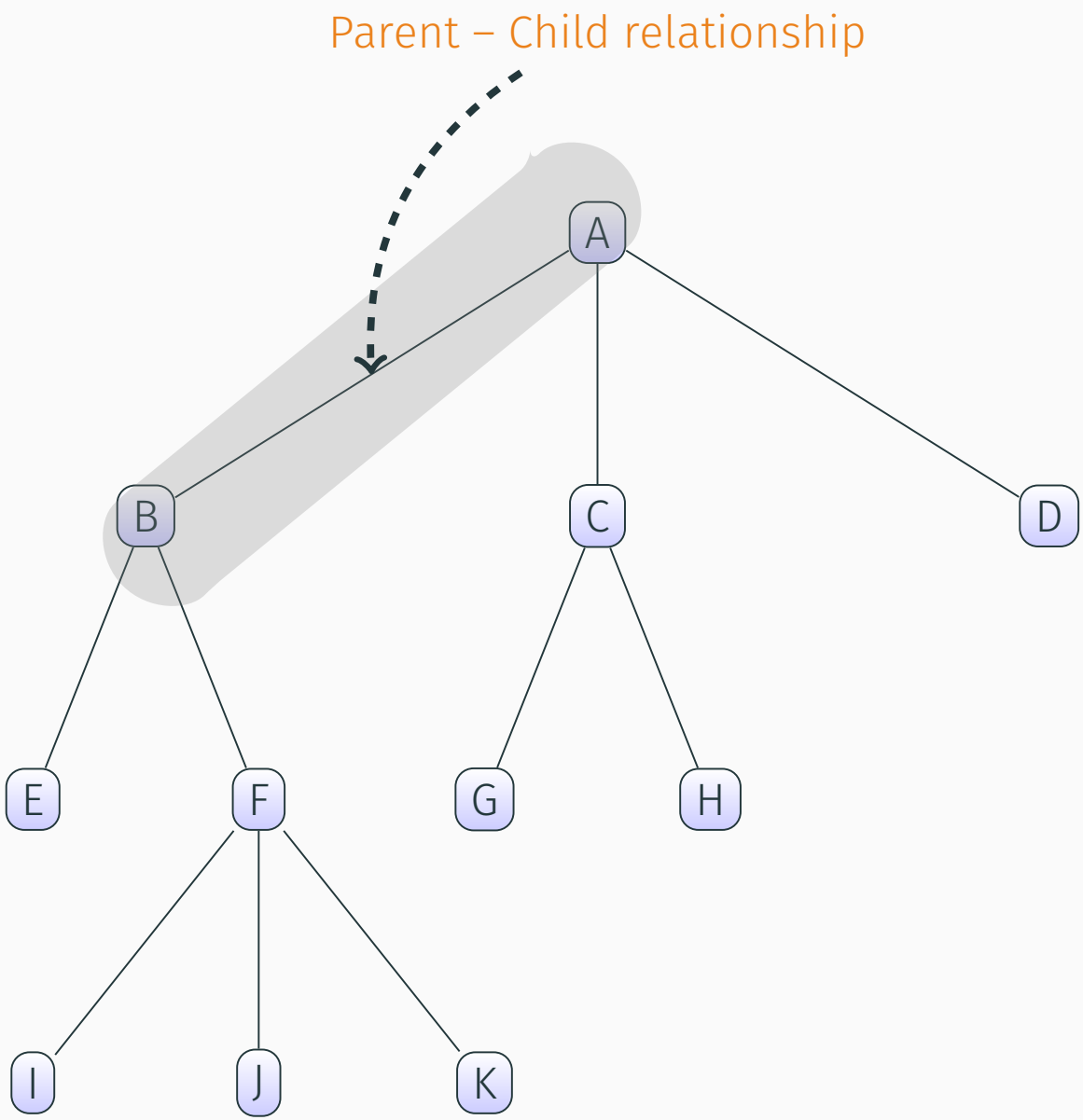


Tree Terminology

Root: node without parent

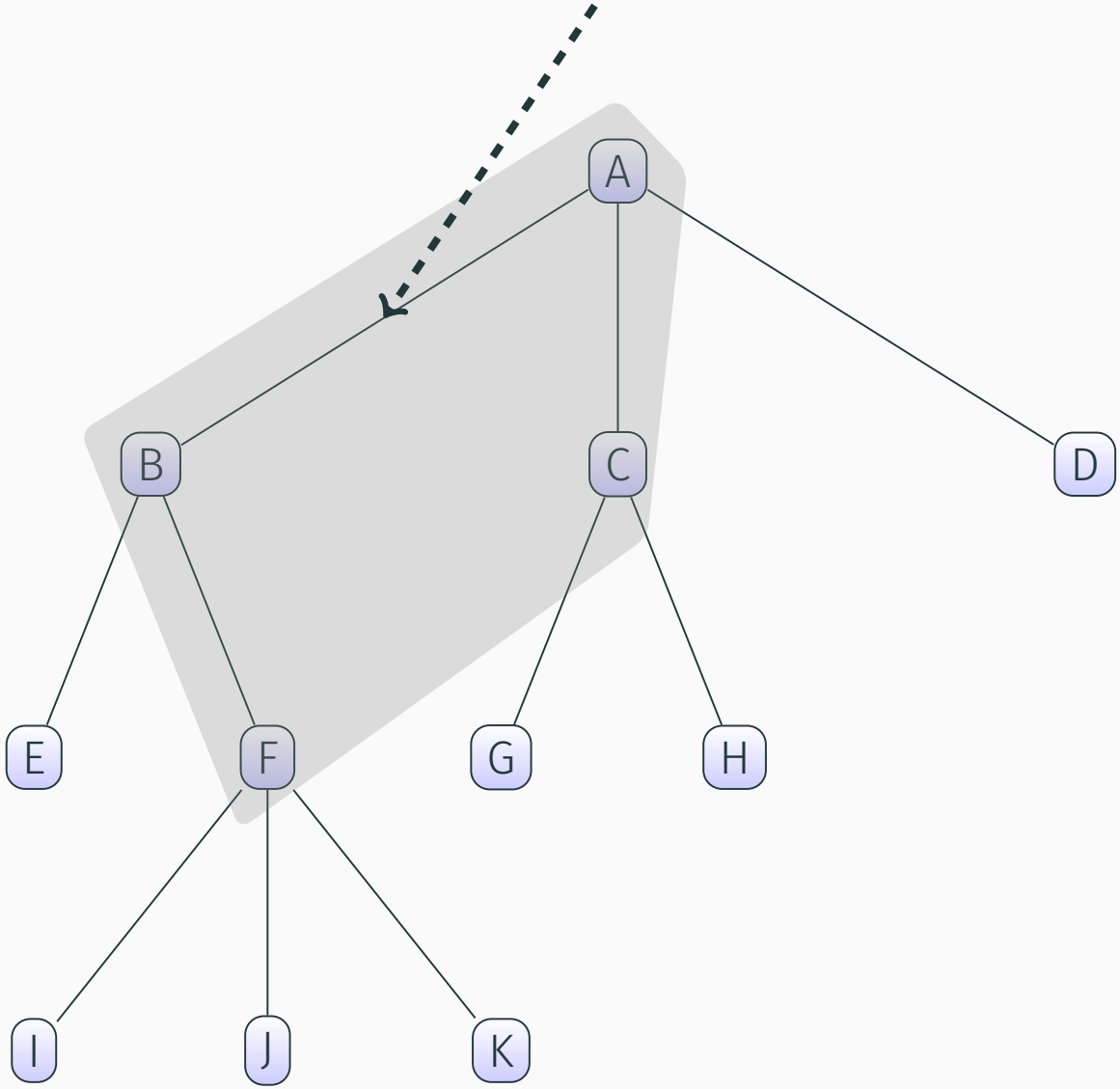


Tree Terminology



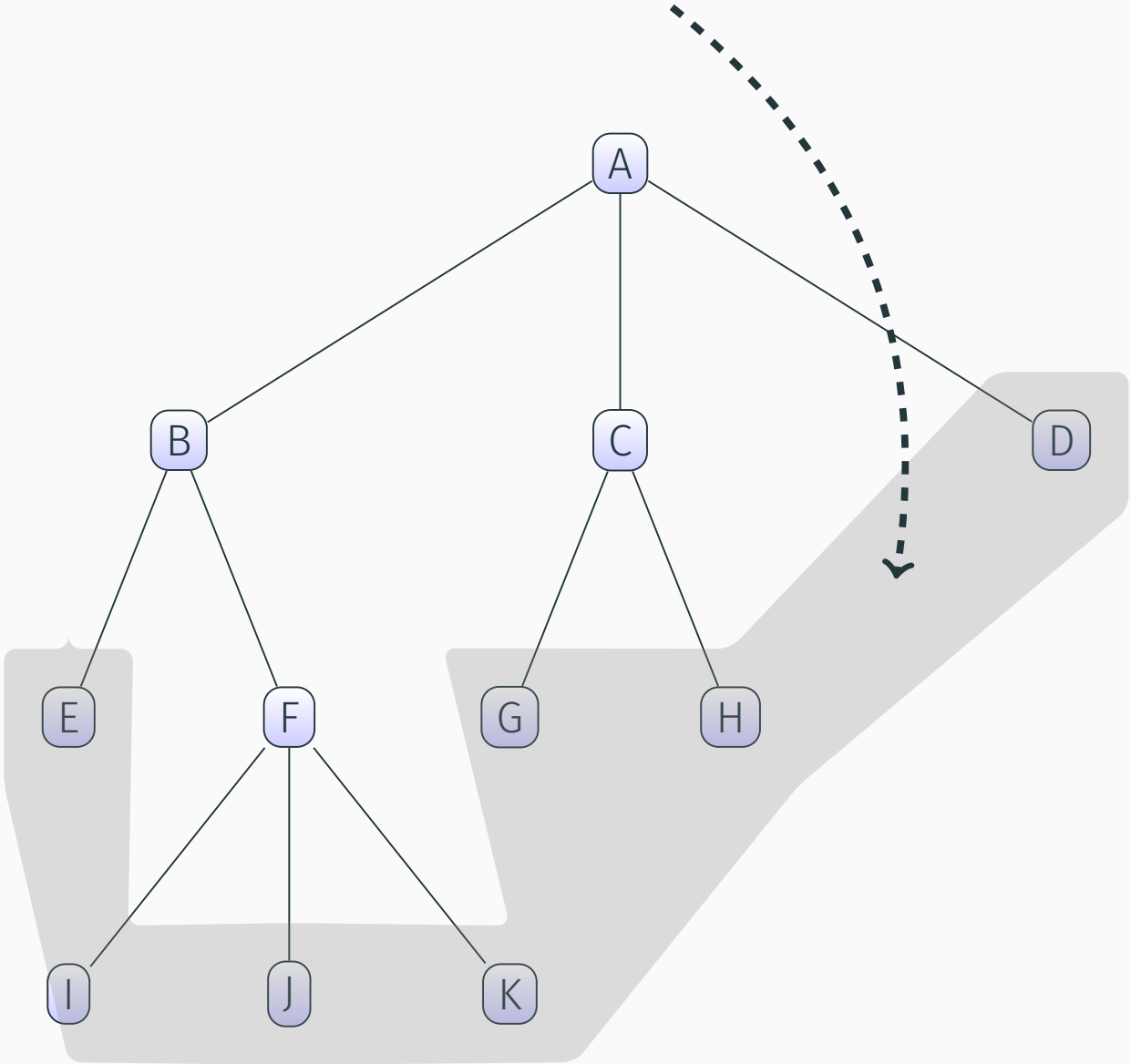
Tree Terminology

Internal node: node with at least one child



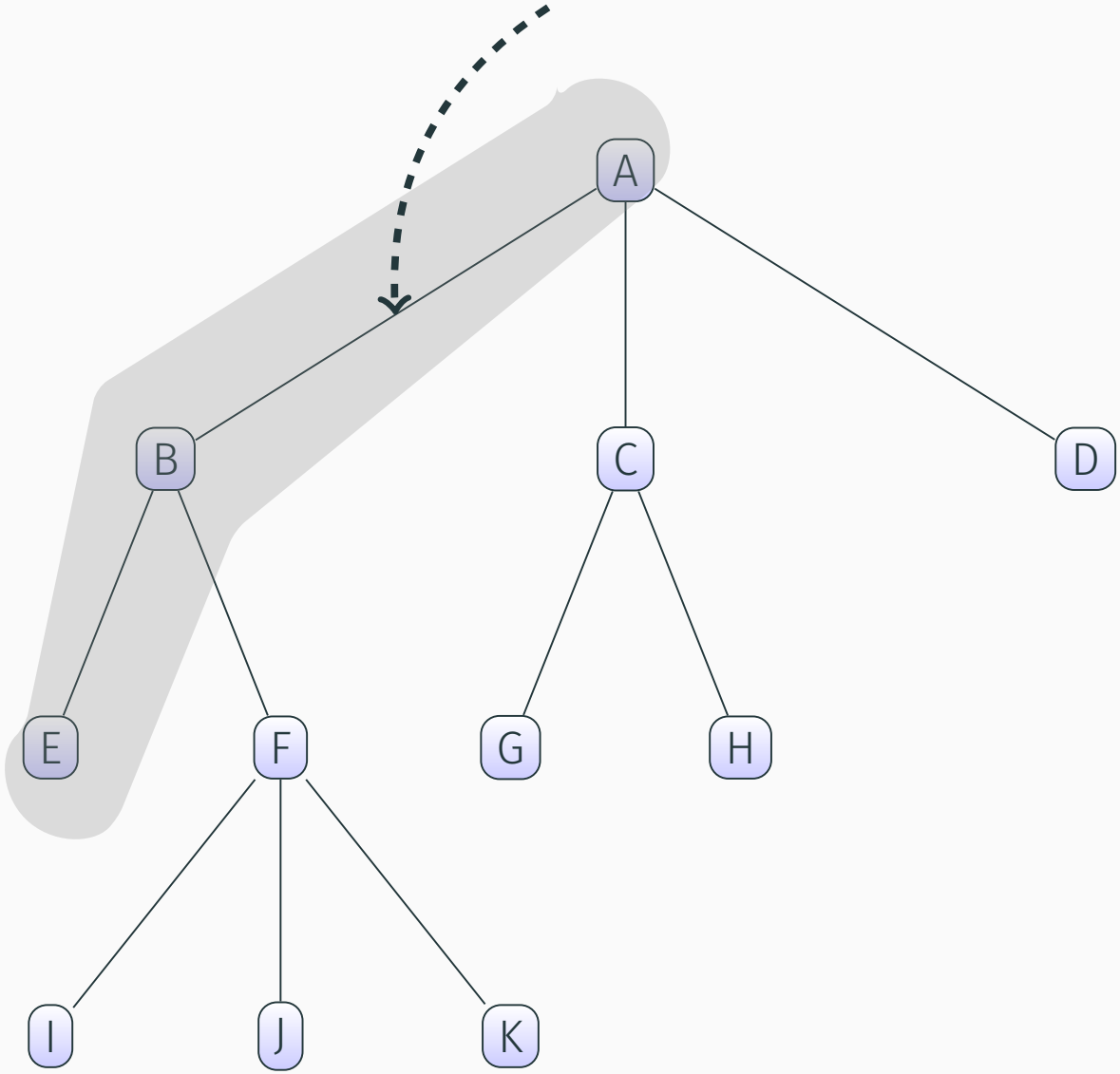
Tree Terminology

External node (a.k.a leaf): node without children



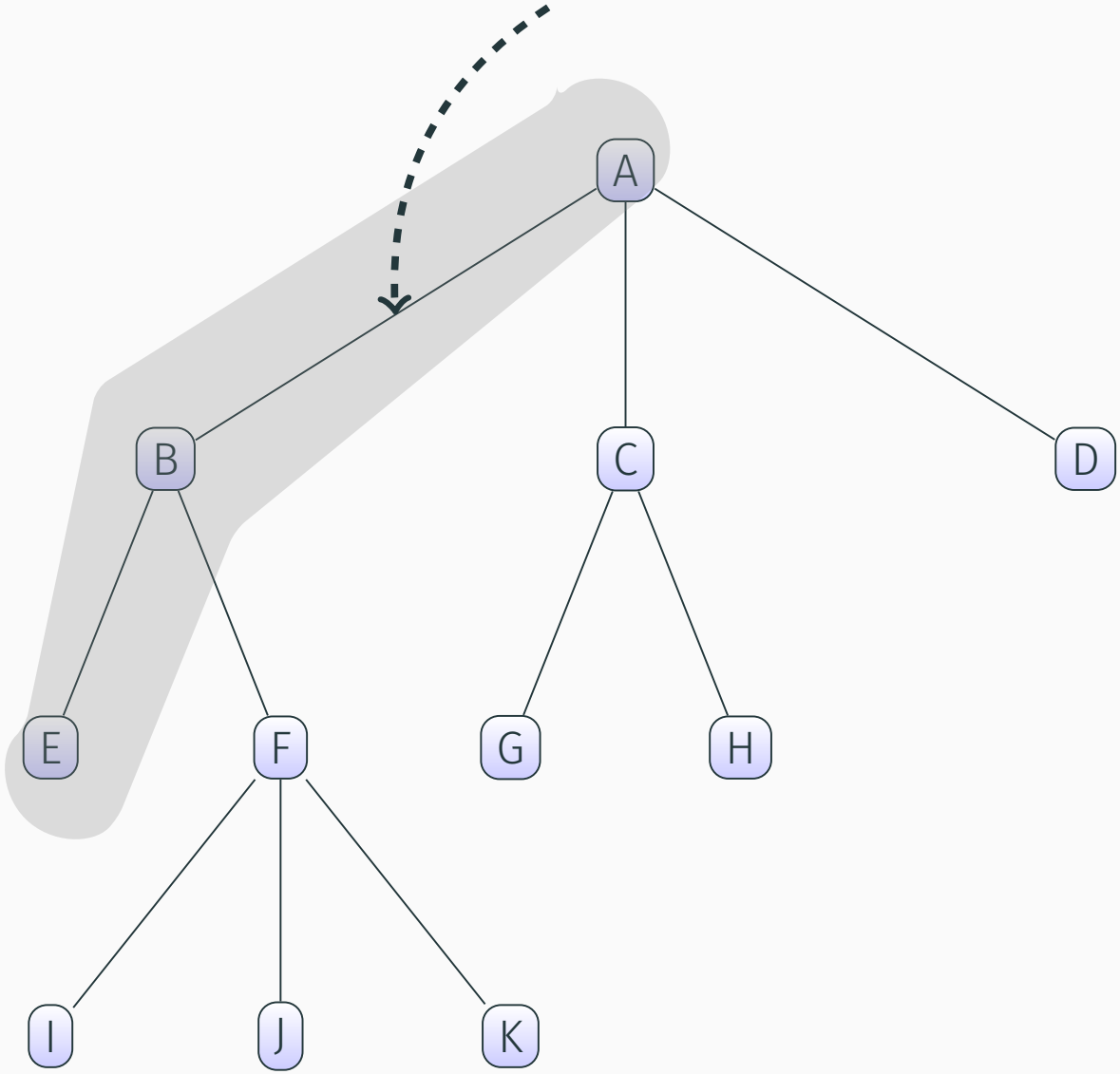
Tree Terminology

Ancestors of a node: itsself, parent, grandparent, grand-grandparent, etc.



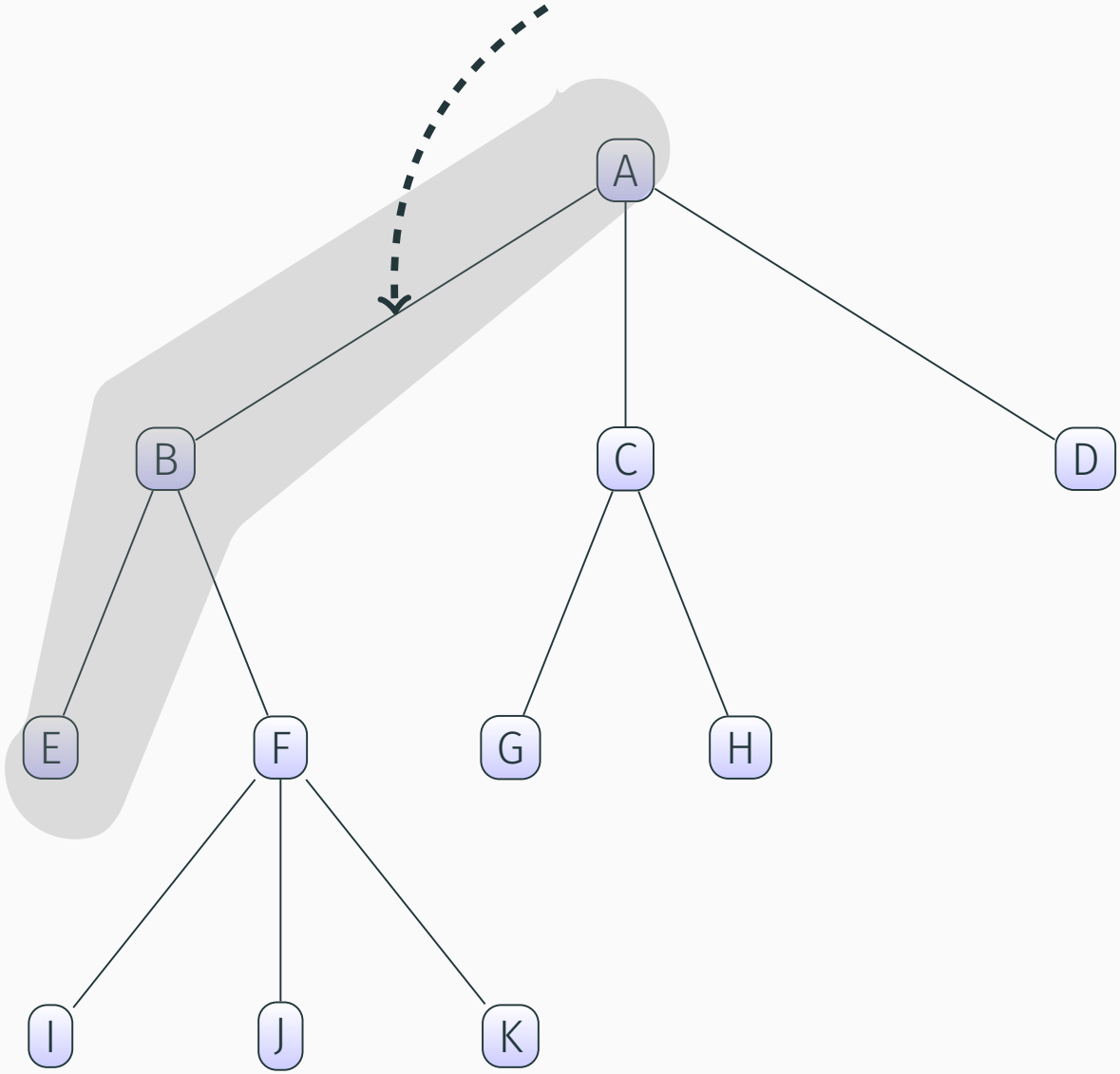
Tree Terminology

Descendant of a node: itself, child, grandchild, grand-grandchild, etc.



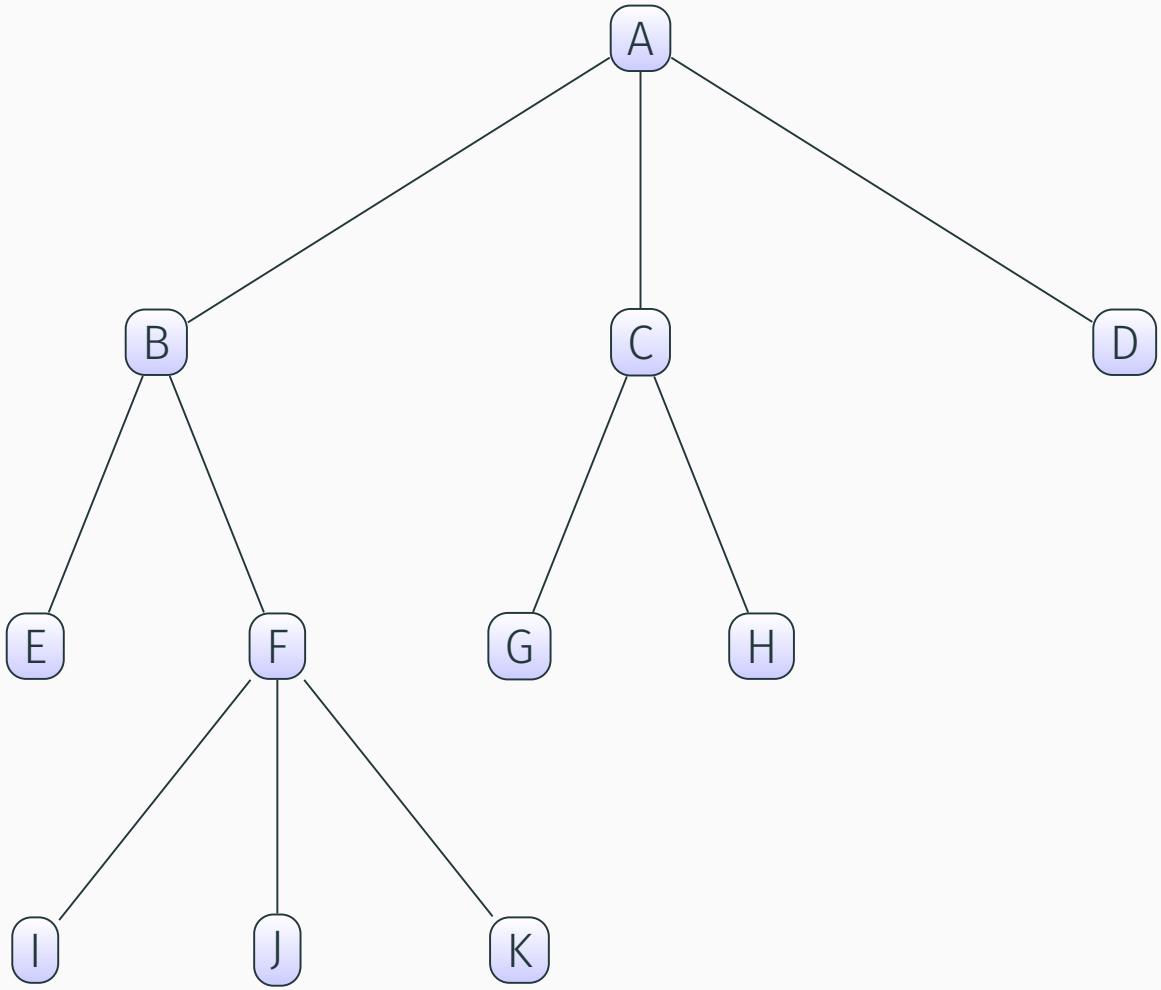
Tree Terminology

Depth of a node: the number of edges from the root to the node, $d(E) = 2$



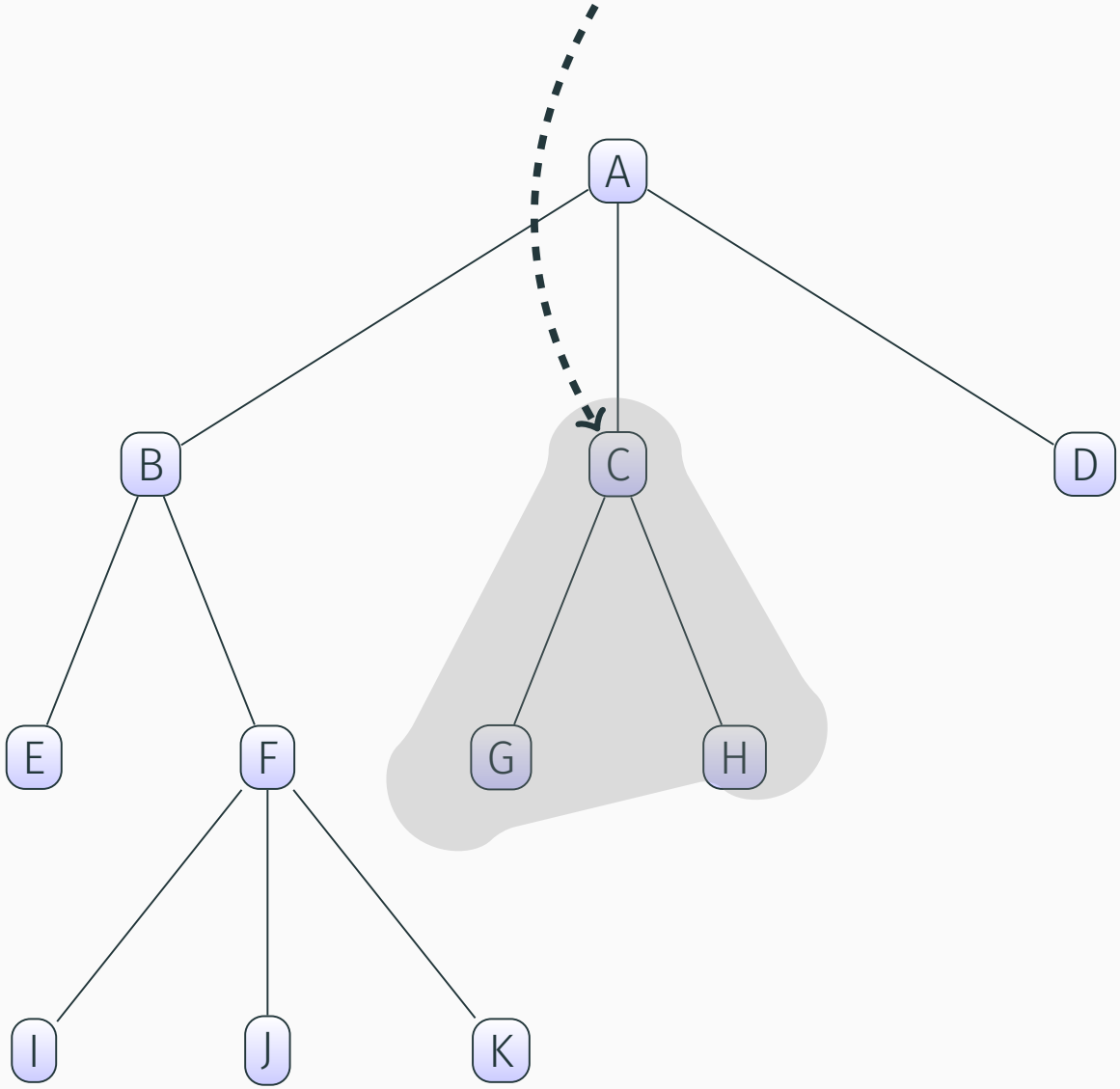
Tree Terminology

Height of a tree: maximum depth of any node, $H = 3$



Tree Terminology

Subtree: tree consisting of a node and its descendants



Tree Interface

```
class treeNode {
  element
  parent
  children
  ------------------
  getParent()
  setPartent()
  getChildren()
  setChildren()
  numChildren()
  ...
}
```

```
class Tree {
  root
  size
  height
  -------------
  root()
  size()
  isEmpty()
  isInternal(node)
  isExternal(node)
  isRoot(node)
  traversal()
  ...
}
```

Traverse a tree

Preorder Traversal

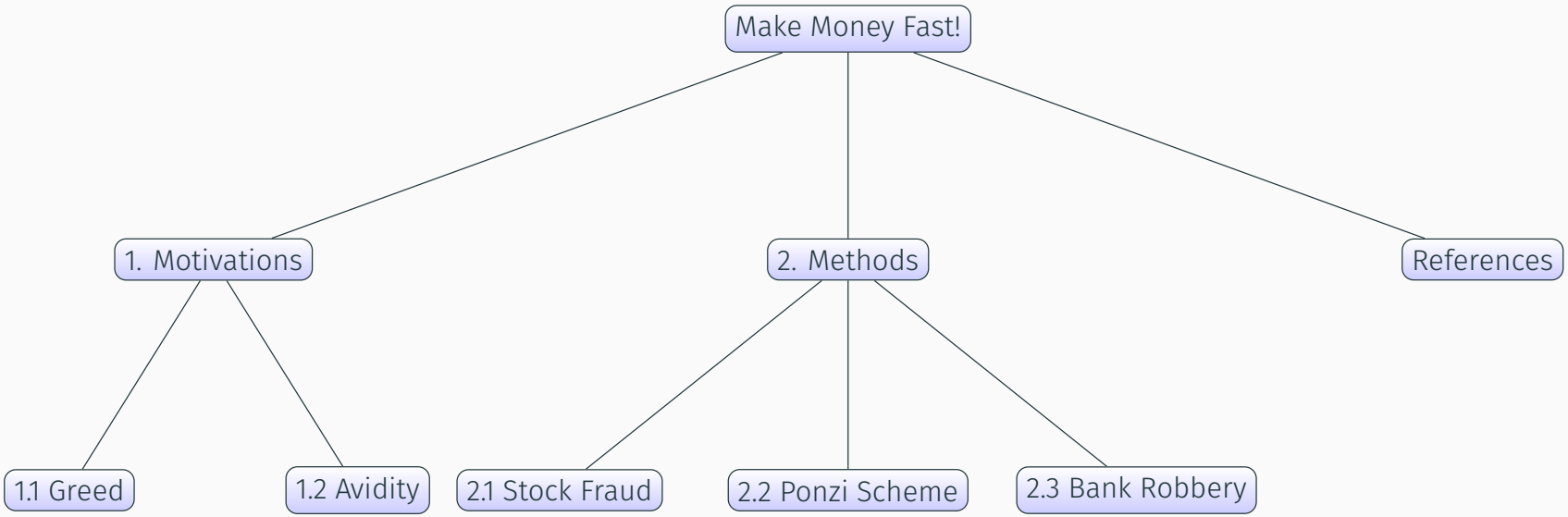
```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

In a preorder traversal, a node is visited before its descendants

Preorder Traversal

```
1 Algorithm preOrder(v)
2
3 visit(v)
4 for each child w of v
5   preorder (w)
```

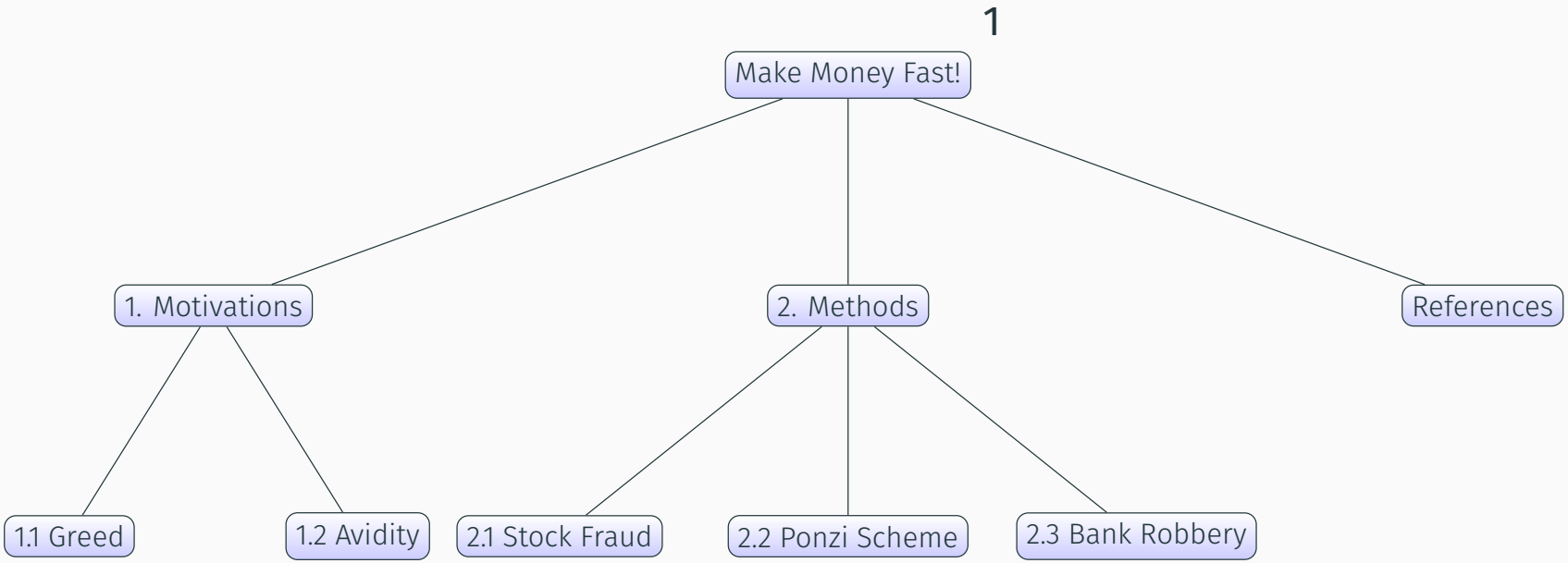
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

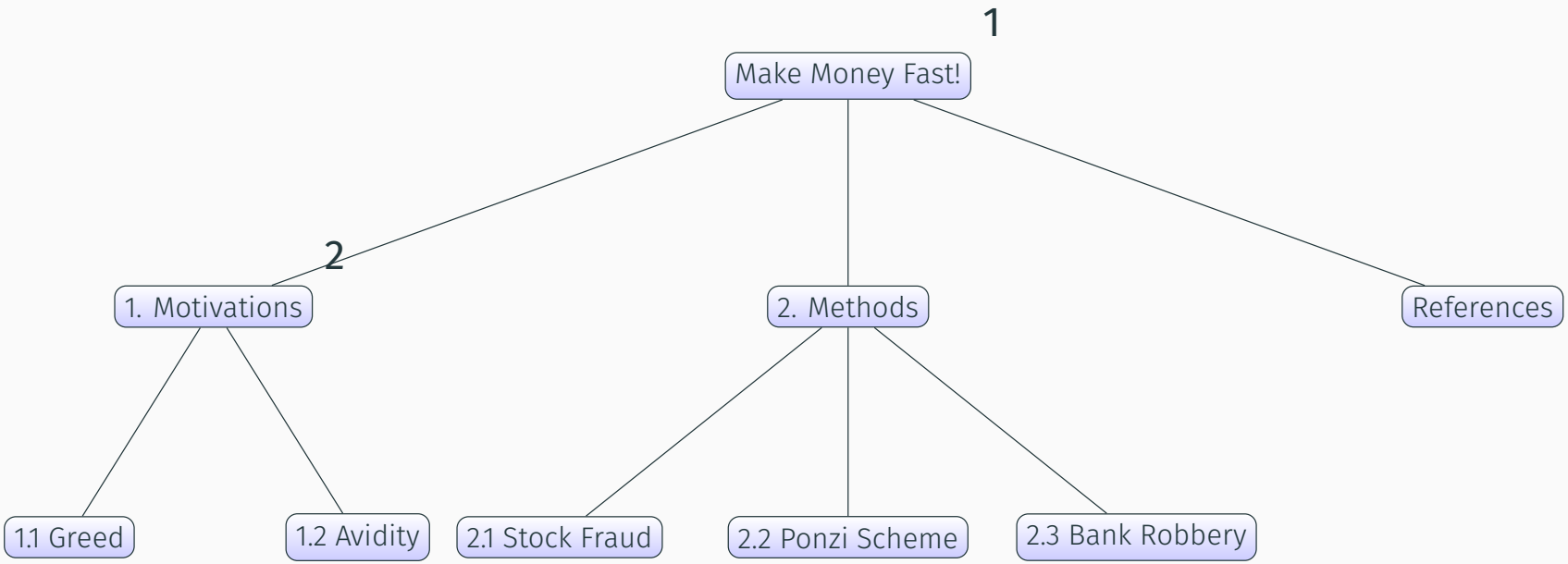
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

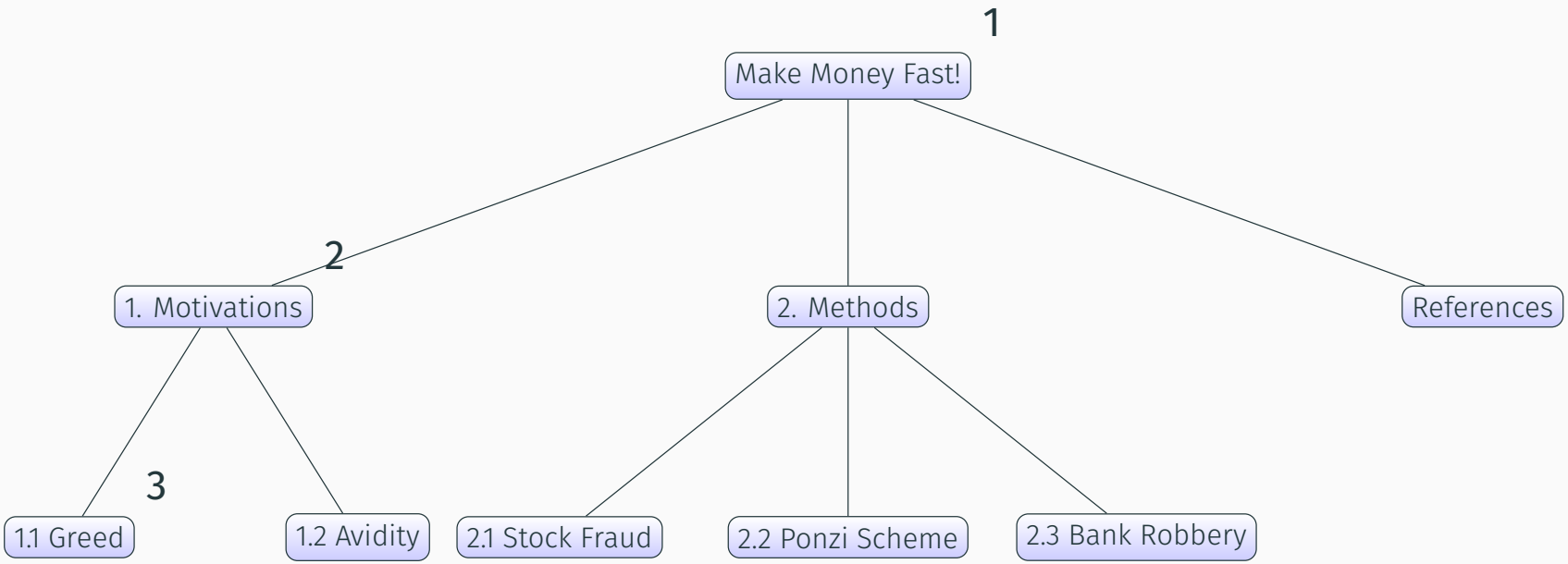
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

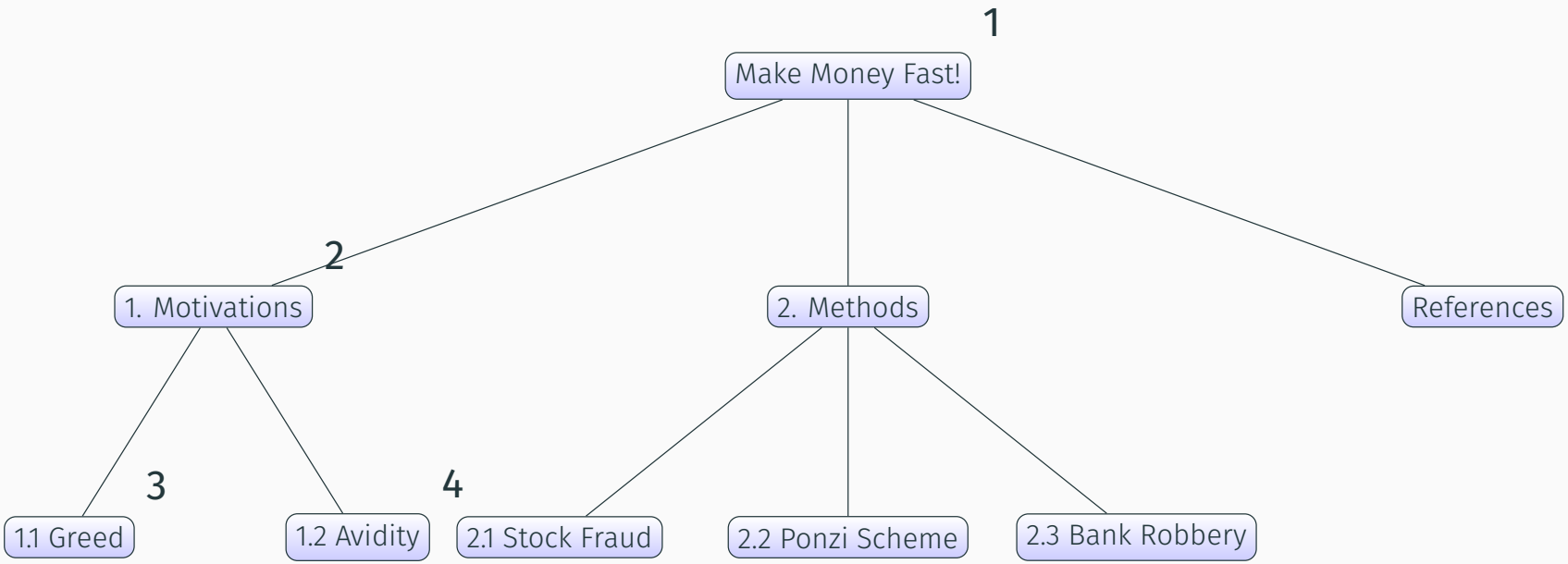
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

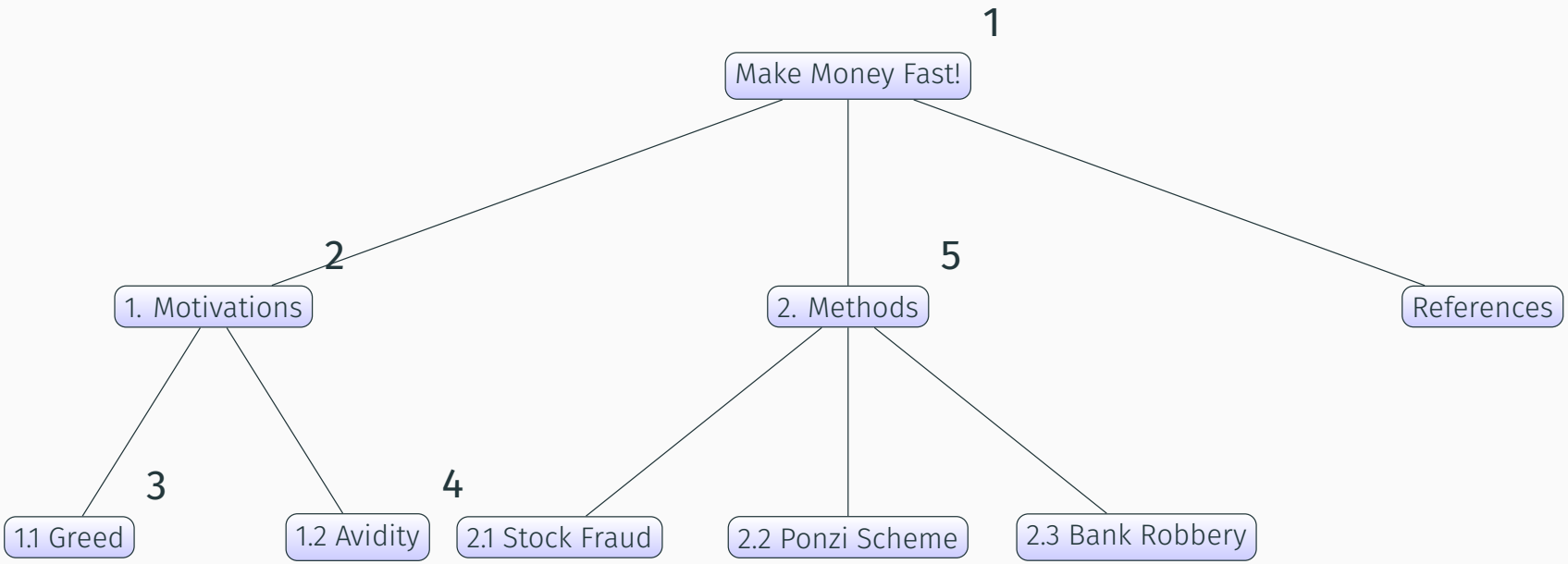
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

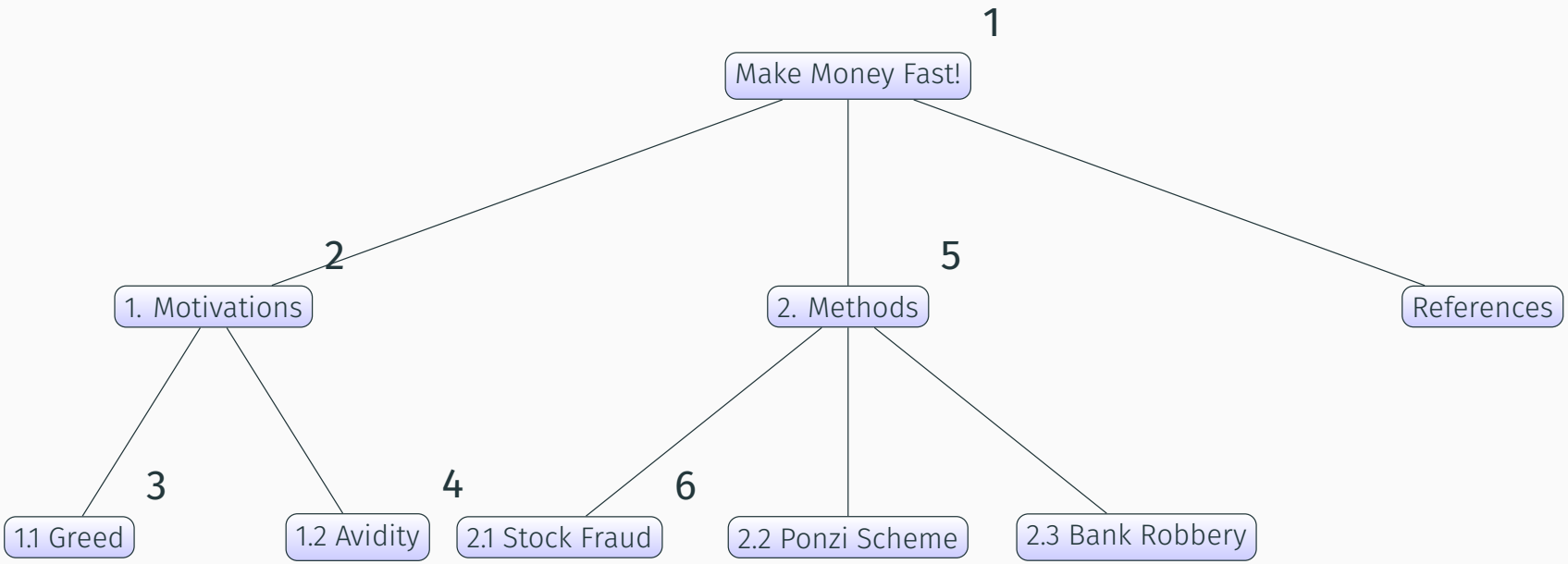
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

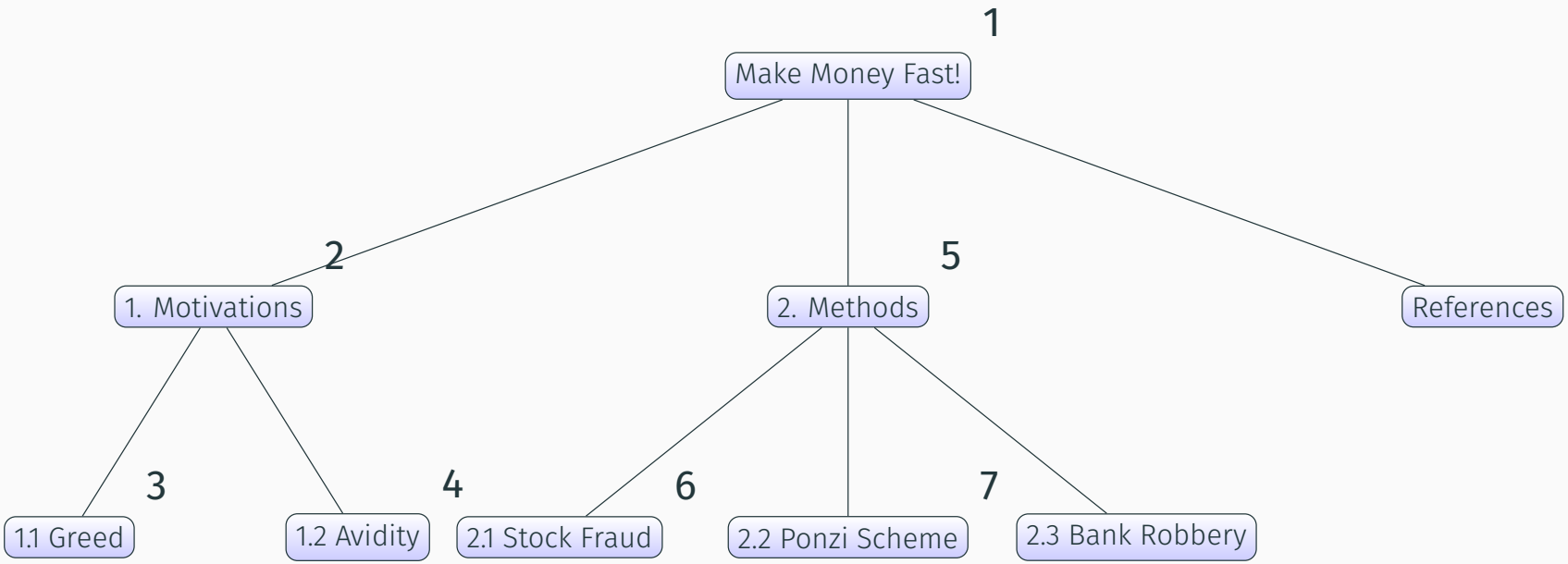
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

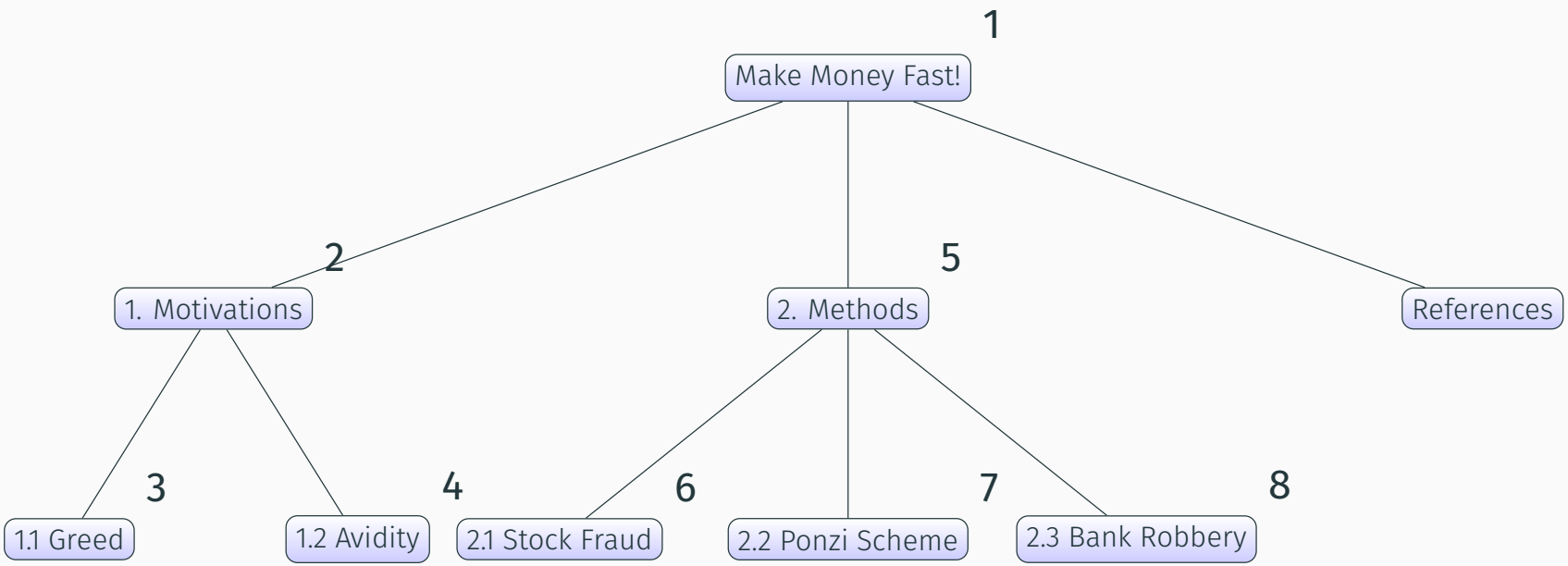
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

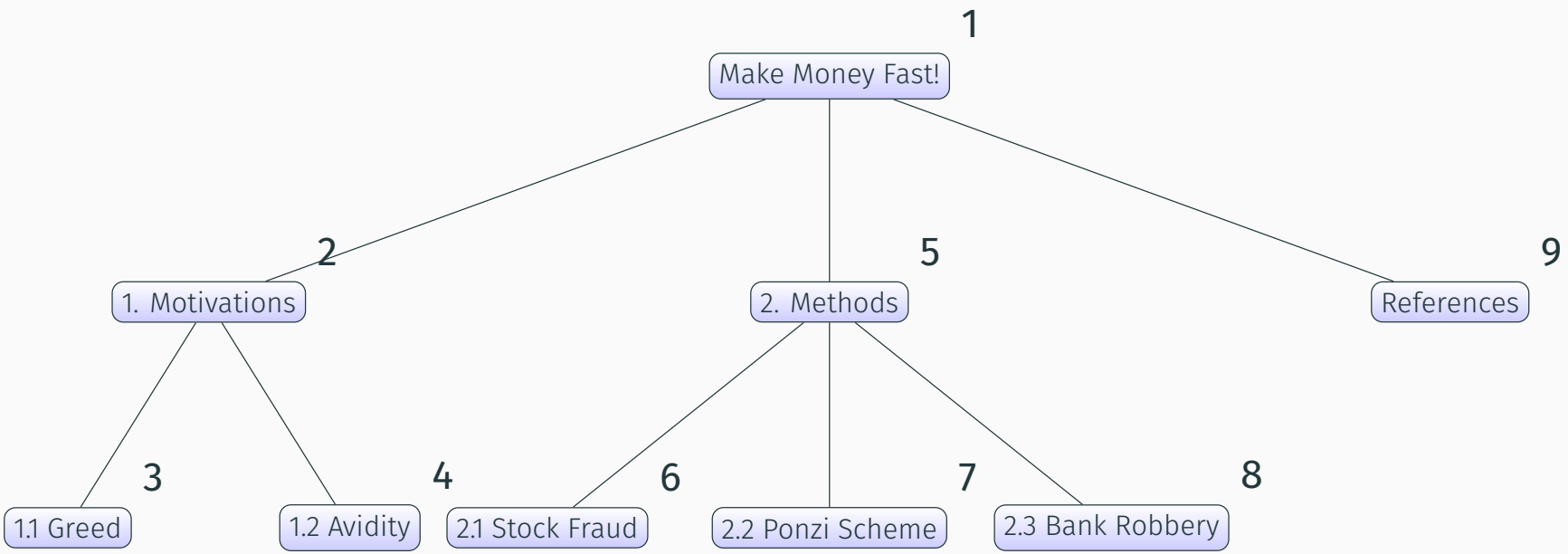
In a preorder traversal, a node is visited before its descendants



Preorder Traversal

```
1  Algorithm preOrder(v)
2
3  visit(v)
4  for each child w of v
5    preorder (w)
```

In a preorder traversal, a node is visited before its descendants



Postorder Traversal

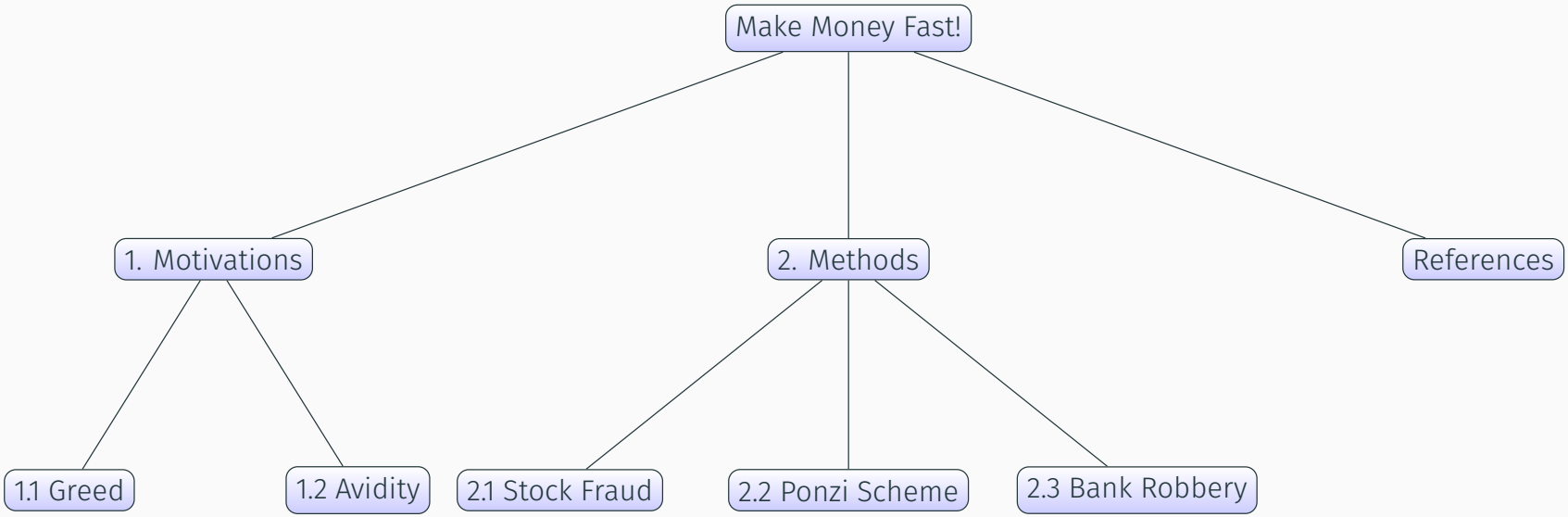
```
1 Algorithm postOrder(v)
2
3 for each child w of v
4   postorder (w)
5 visit(v)
```

In a postorder traversal, a node is visited after its descendants

Postorder Traversal

```
1 Algorithm postOrder(v)
2
3 for each child w of v
4   postorder (w)
5 visit(v)
```

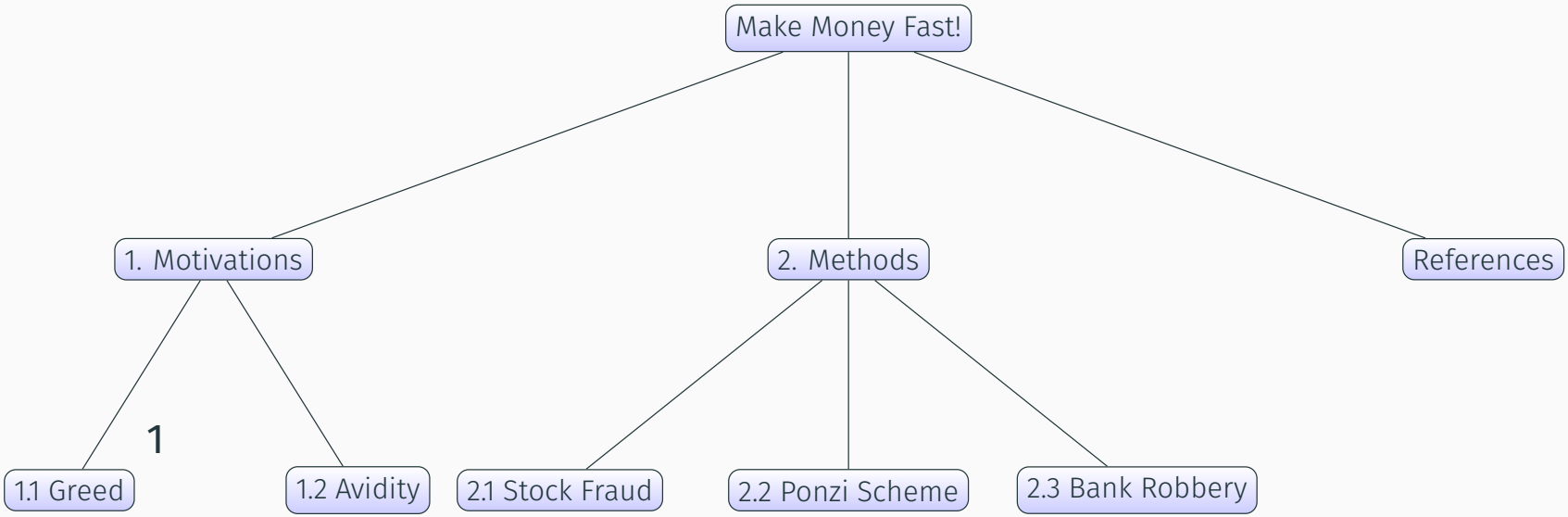
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

```
1 Algorithm postOrder(v)
2
3 for each child w of v
4   postorder (w)
5 visit(v)
```

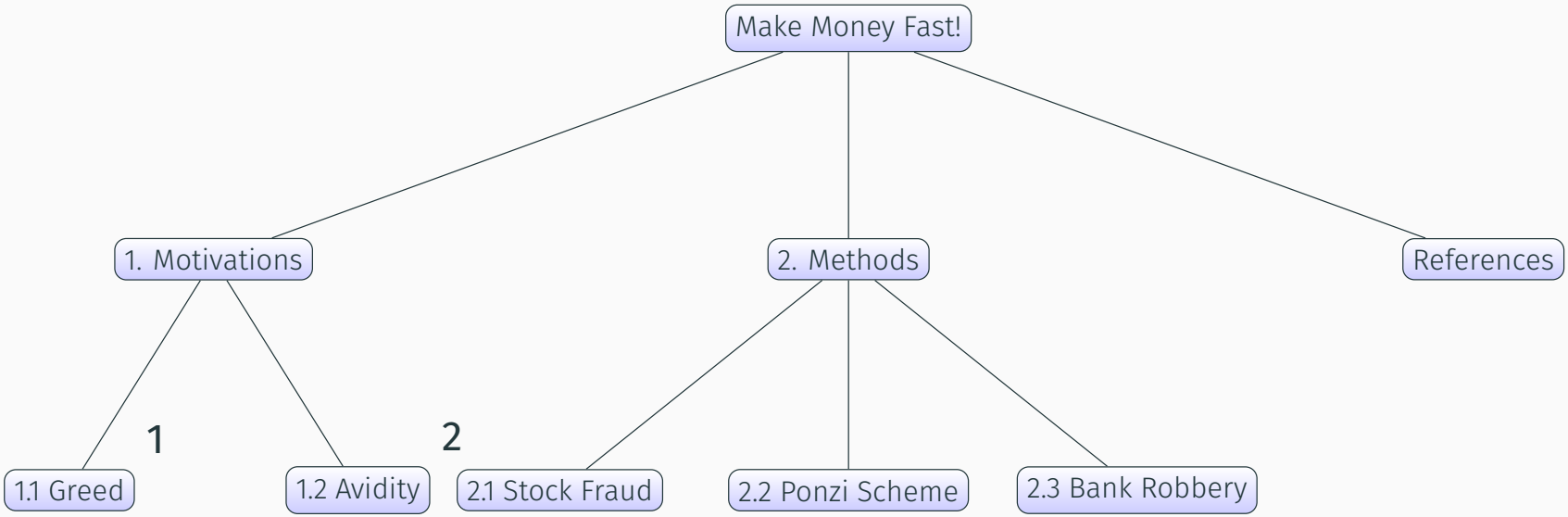
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

```
1  Algorithm postOrder(v)
2
3  for each child w of v
4    postorder (w)
5  visit(v)
```

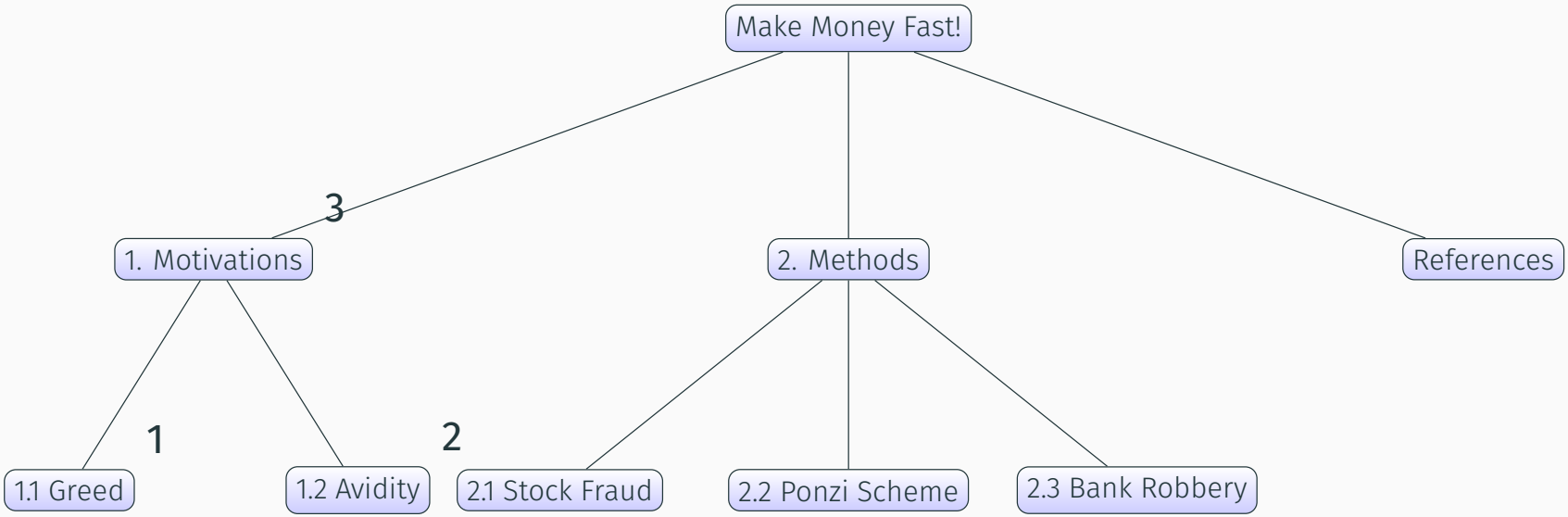
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

```
1  Algorithm postOrder(v)
2
3  for each child w of v
4    postorder (w)
5  visit(v)
```

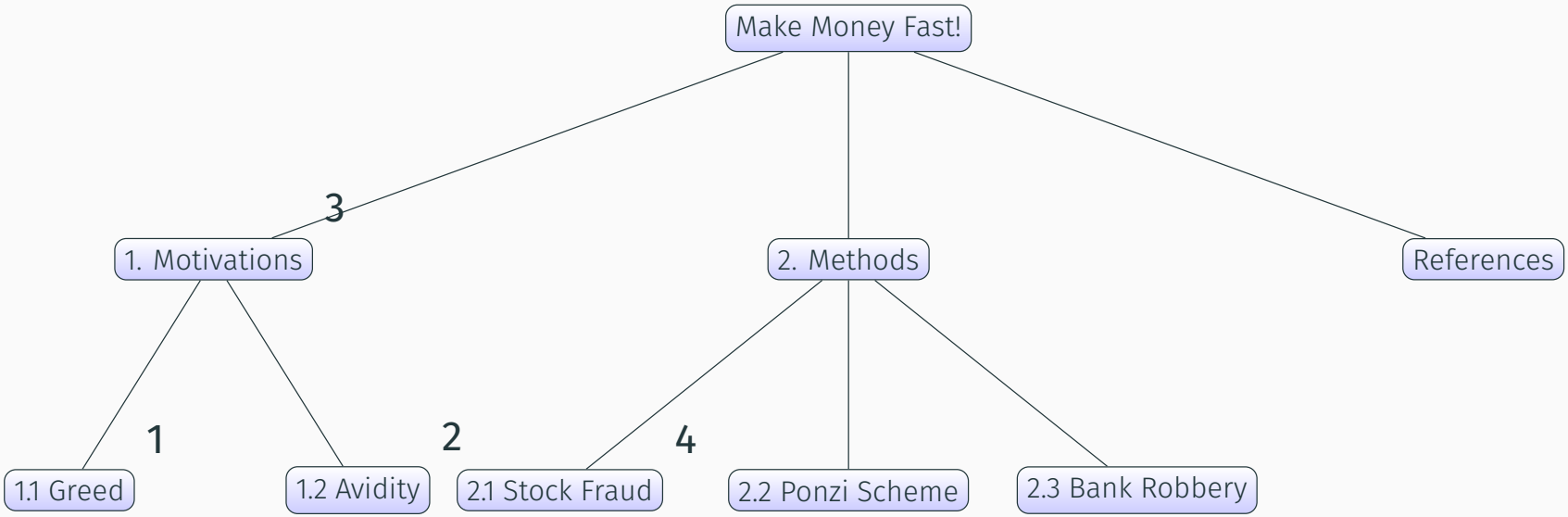
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

```
1  Algorithm postOrder(v)
2
3  for each child w of v
4    postorder (w)
5  visit(v)
```

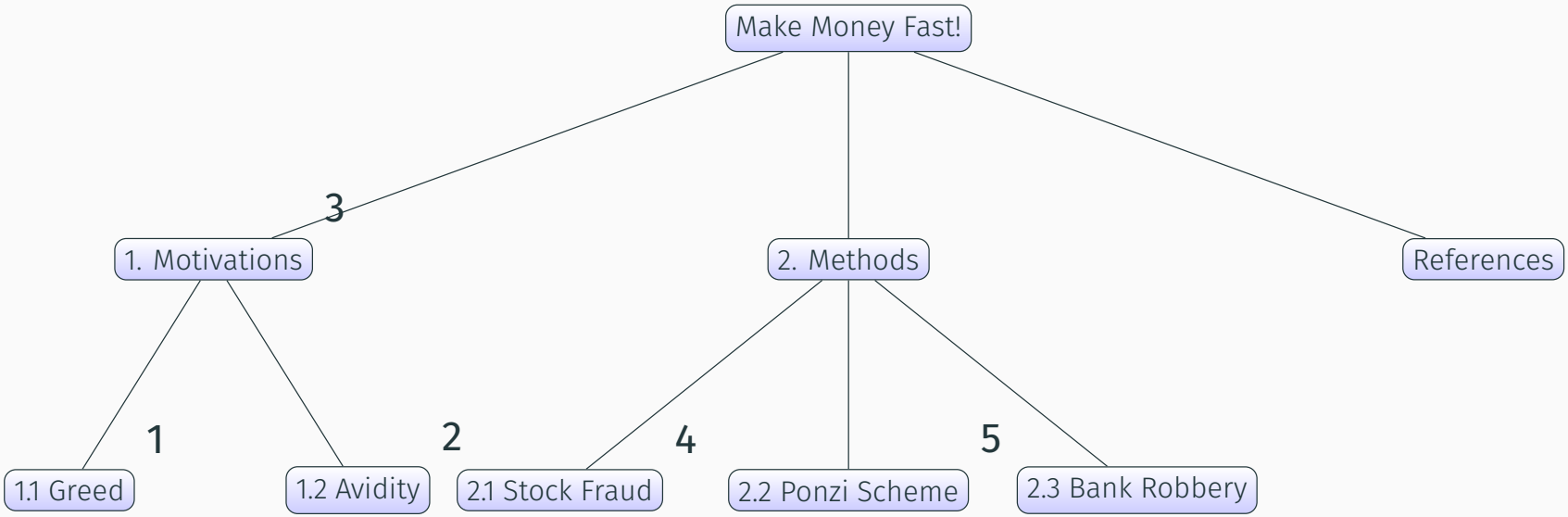
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

```
1 Algorithm postOrder(v)
2
3 for each child w of v
4   postorder (w)
5 visit(v)
```

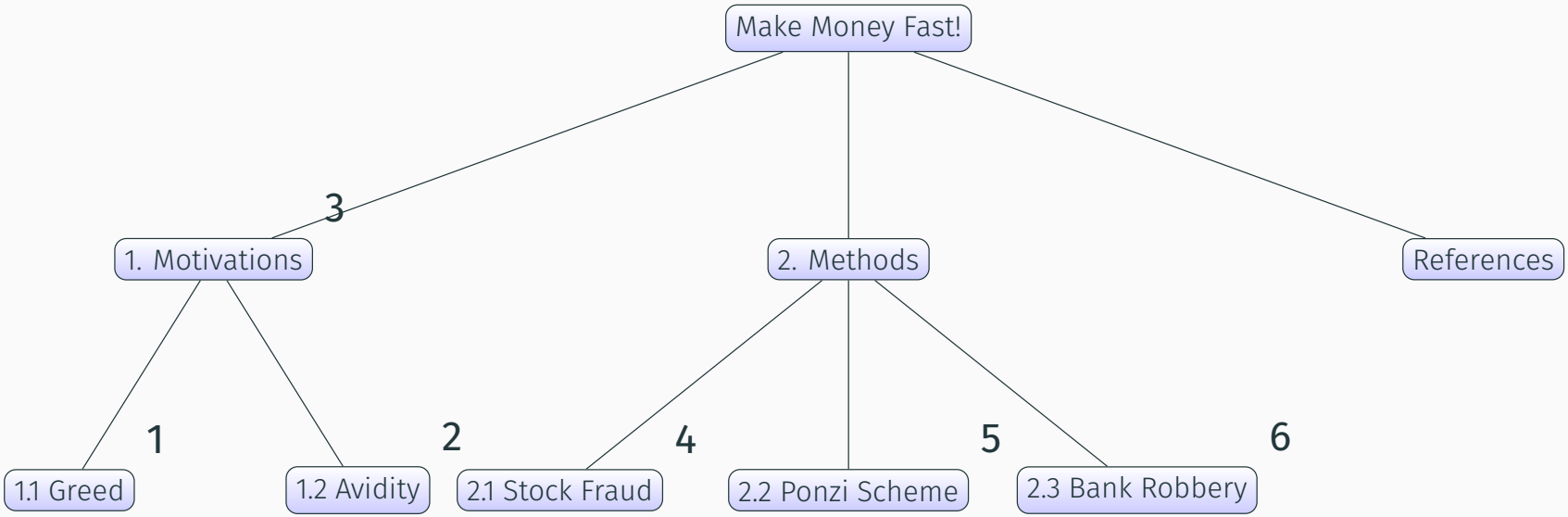
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

```
1 Algorithm postOrder(v)
2
3 for each child w of v
4   postorder (w)
5 visit(v)
```

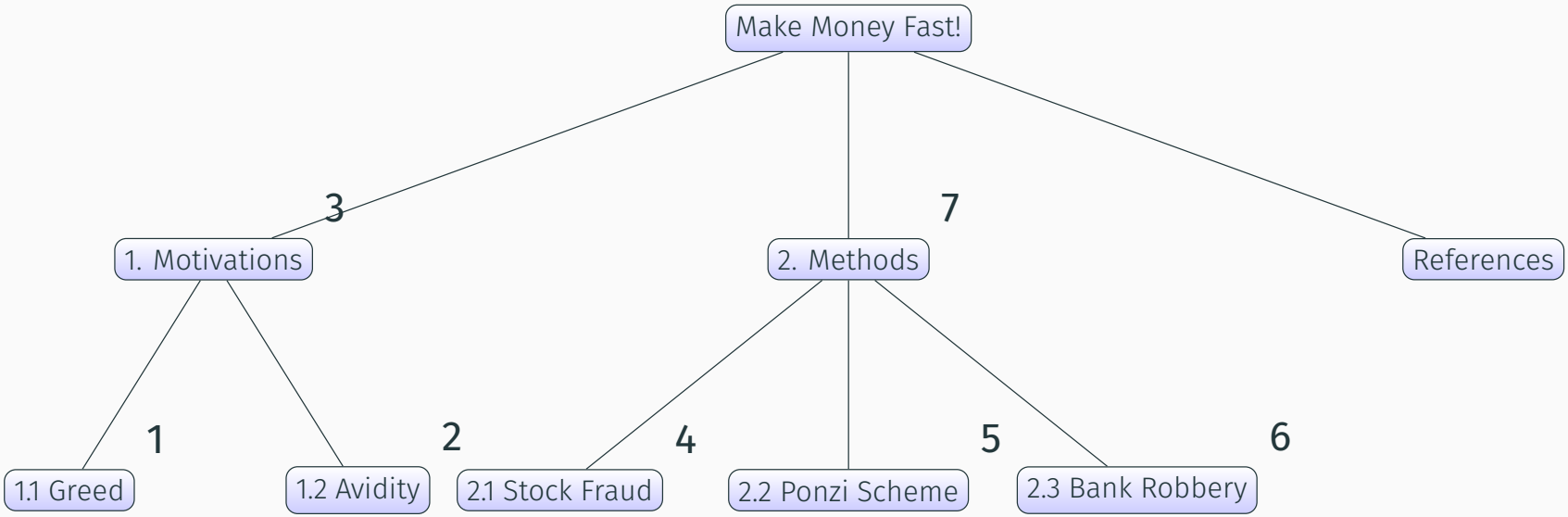
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

```
1  Algorithm postOrder(v)
2
3  for each child w of v
4    postorder (w)
5  visit(v)
```

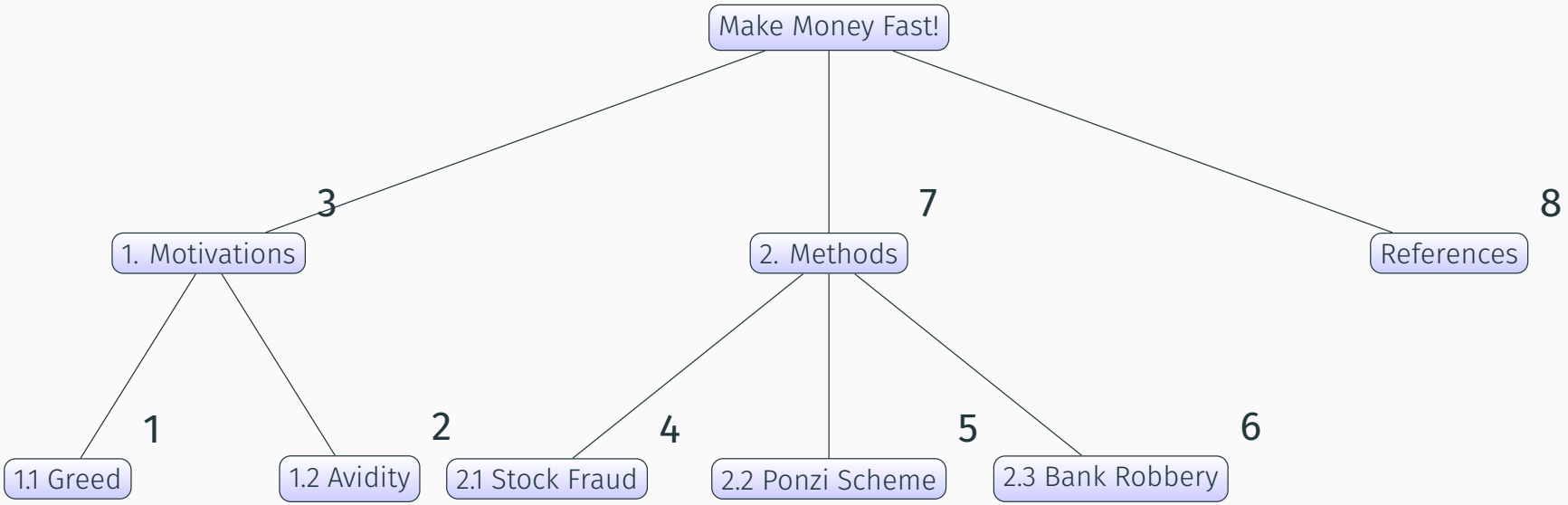
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

```
1  Algorithm postOrder(v)
2
3  for each child w of v
4    postorder (w)
5  visit(v)
```

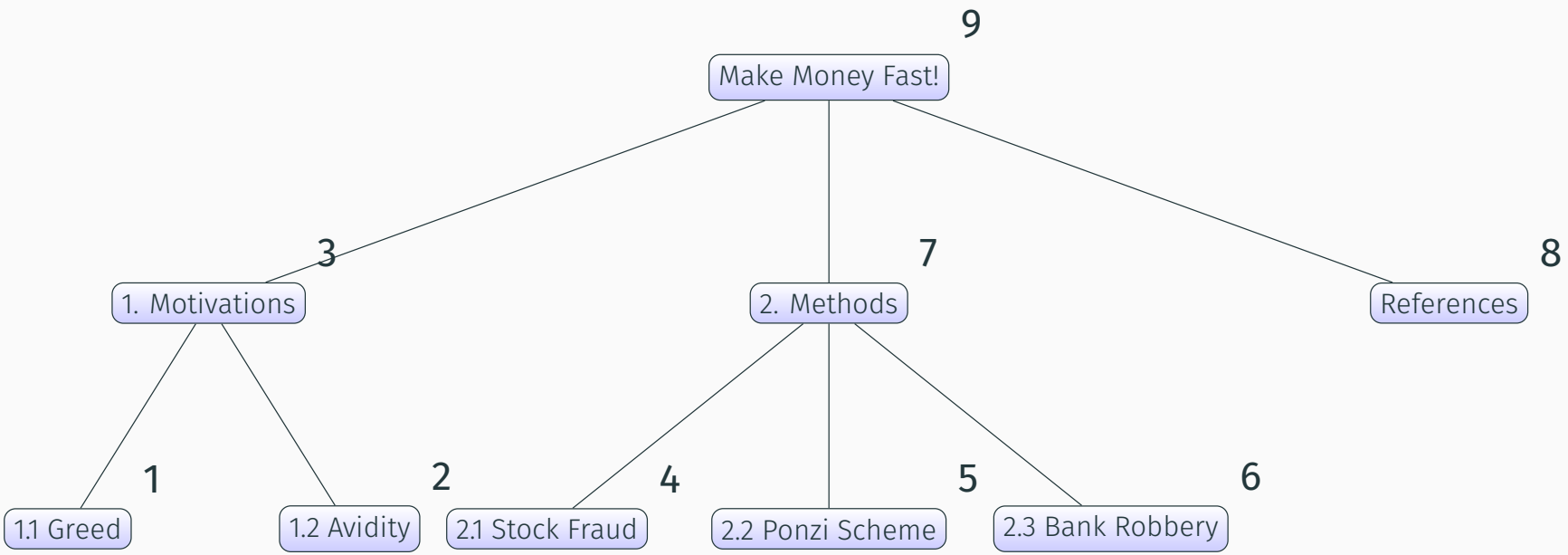
In a postorder traversal, a node is visited after its descendants



Postorder Traversal

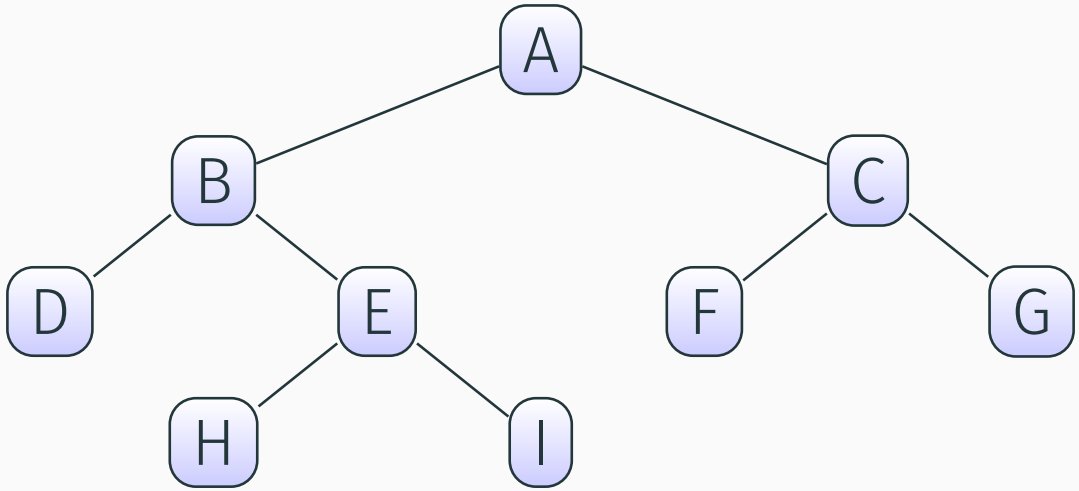
```
1  Algorithm postOrder(v)
2
3  for each child w of v
4    postorder (w)
5  visit(v)
```

In a postorder traversal, a node is visited after its descendants



Binary tree

Binary tree



Binary tree: each internal node has at most two children (exactly two for **proper binary trees**)

Applications:

- arithmetic expressions
- decision processes
- searching

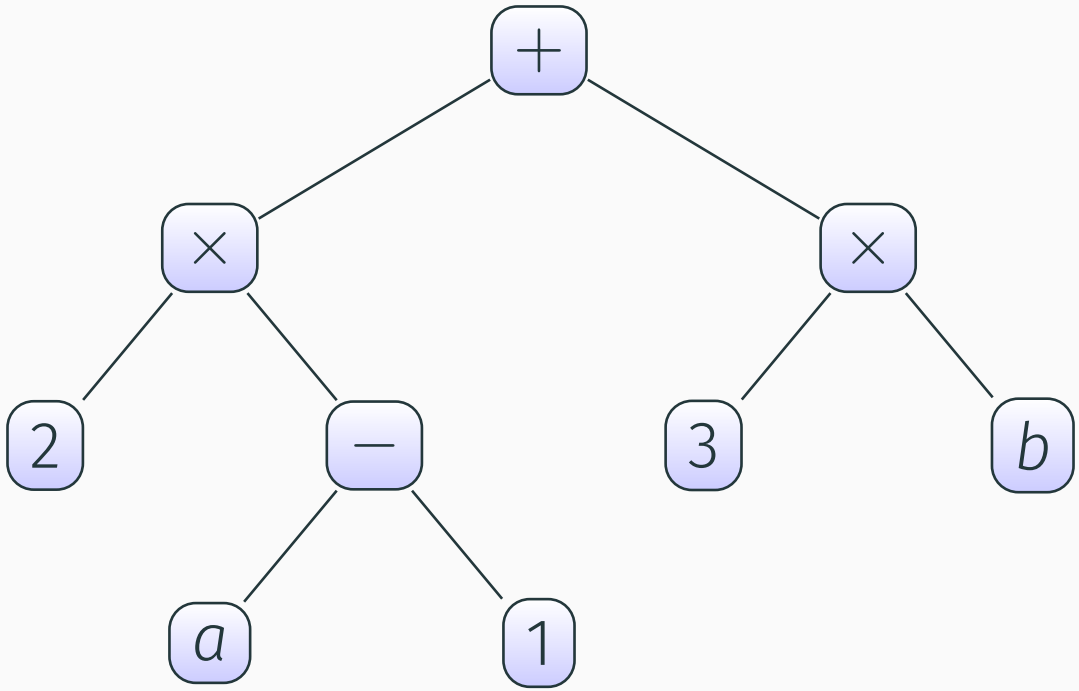
Arithmetic Expression Tree

Binary tree associated with an arithmetic expression

- internal nodes: operators
- external nodes: operands

Example: arithmetic expression tree for the expression

$$(2 \times (a - 1) + (3 \times b))$$

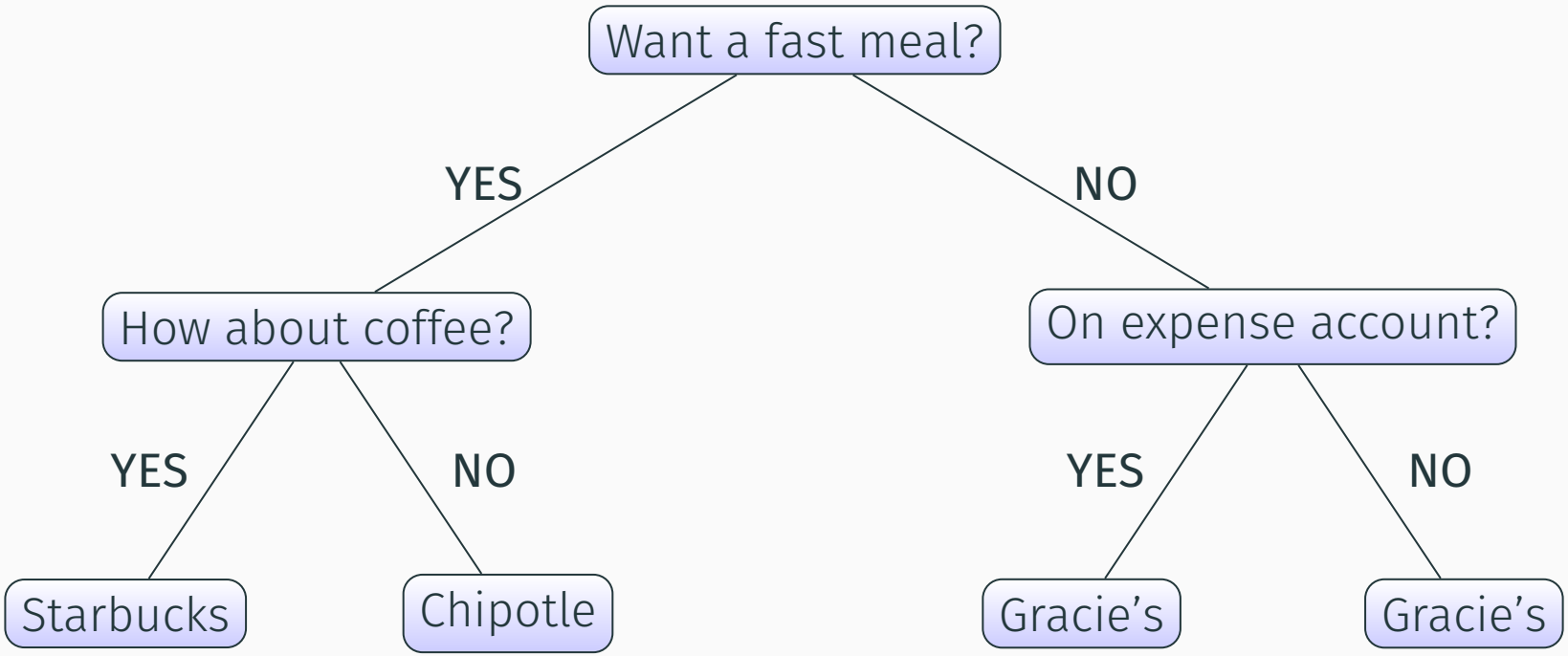


Decision Tree

Binary tree associated with a decision process

- internal nodes: questions with yes/no answer
- external nodes: decisions

Example: dining decision

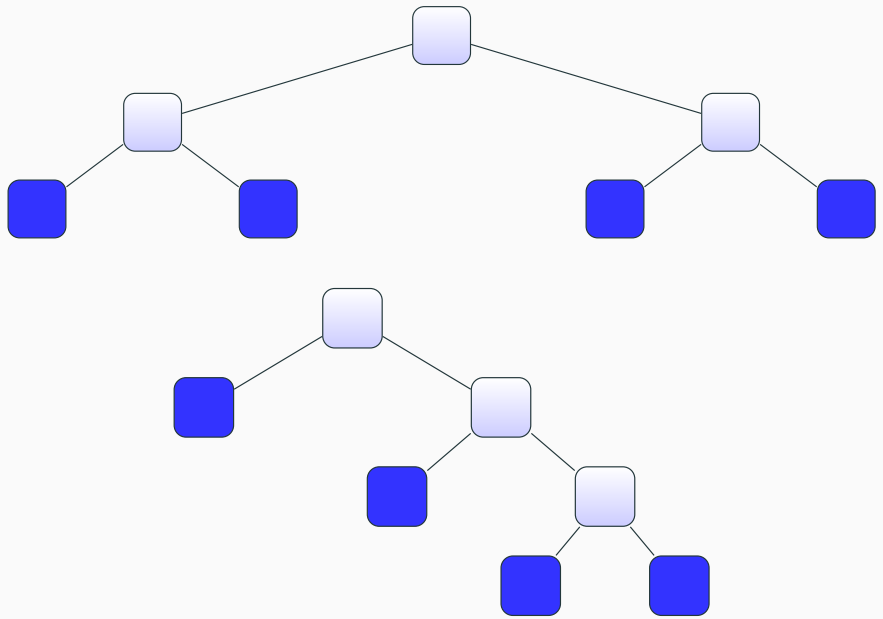


Properties of Proper Binary Trees

Proper Binary Tree: every internal node has exactly two children

Let

- n: number of nodes
- e: number of external nodes
- i: number of internal nodes
- h: height



Properties:

1. $e = i + 1$
2. $n = 2e - 1$
3. $h \leq i$
4. $h \leq (n - 1)/2$
5. $e \leq 2^h$
6. $h \geq \log_2 e$
7. $h \geq \log_2(n + 1) - 1$

Binary Tree Interface

```
class binaryTreeNode {
    ...
    // children
    leftChild
    rightChild
    ------------------
    ...
    // getChildren()
    // setChildren()
    getLeftChild()
    setLeftChild()
    getRightChild()
    setRightChild()
    ...
}
```

```
class Tree {
    root
    size
    height
    --------------
    root()
    size()
    isEmpty()
    isInternal(node)
    isExternal(node)
    isRoot(node)
    traversal()
    ...
}
```

Binary tree

Inorder Traversal

Inorder Traversal

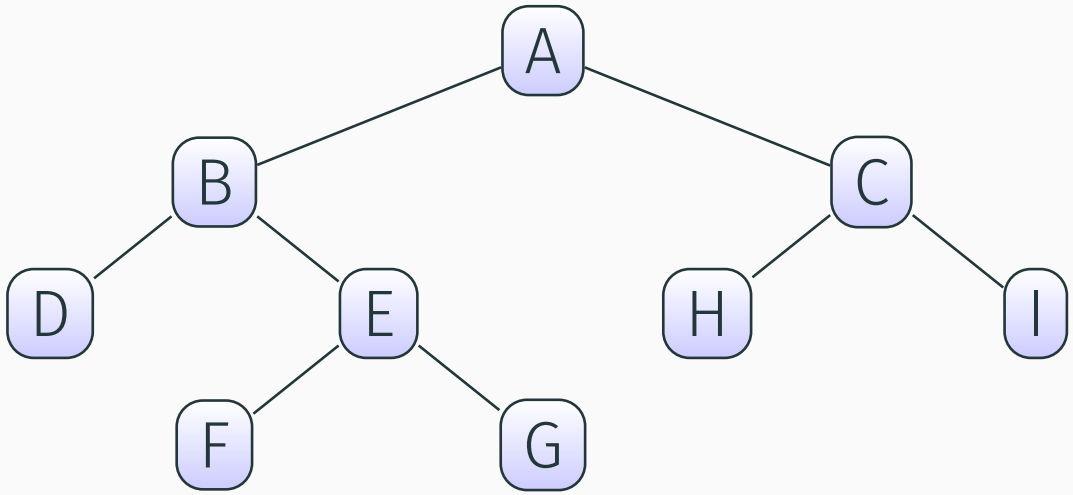
```
1 Algorithm inOrder(v)
2
3 if left (v) ≠ null
4   inOrder (left (v))
5 visit(v)
6 if right(v) ≠ null
7   inOrder (right (v))
```

In an inorder traversal, a node is visited after its left subtree and before its right subtree

Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

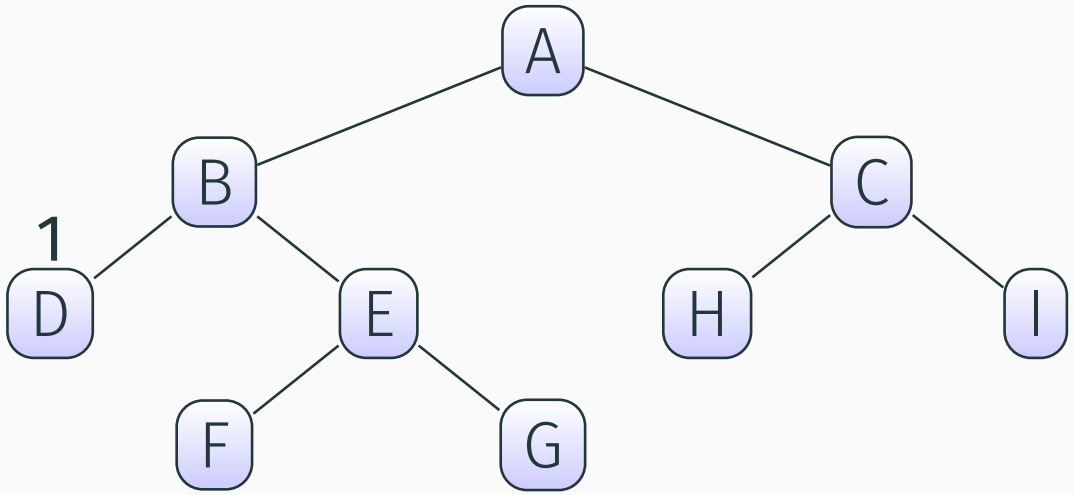
In an inorder traversal, a node is visited after its left subtree and before its right subtree



Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

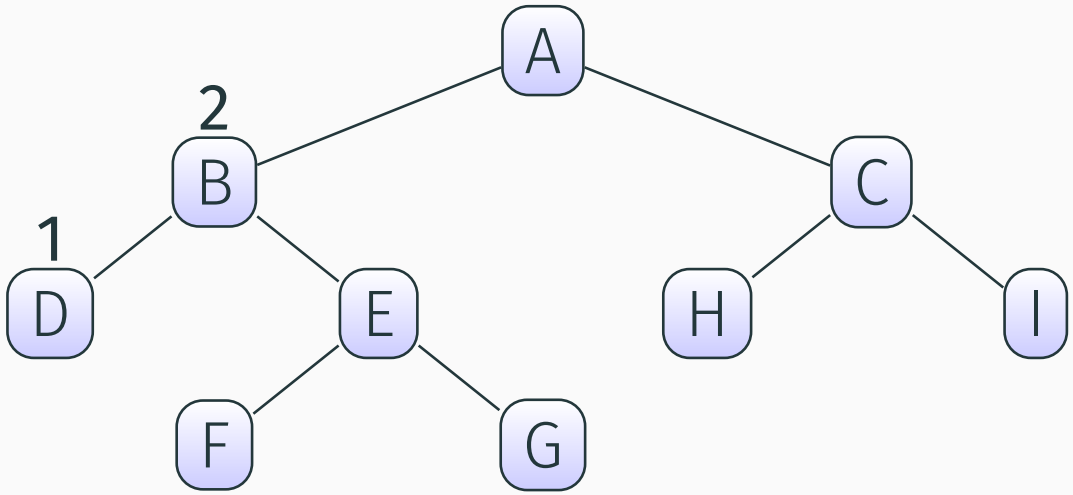
In an inorder traversal, a node is visited after its left subtree and before its right subtree



Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

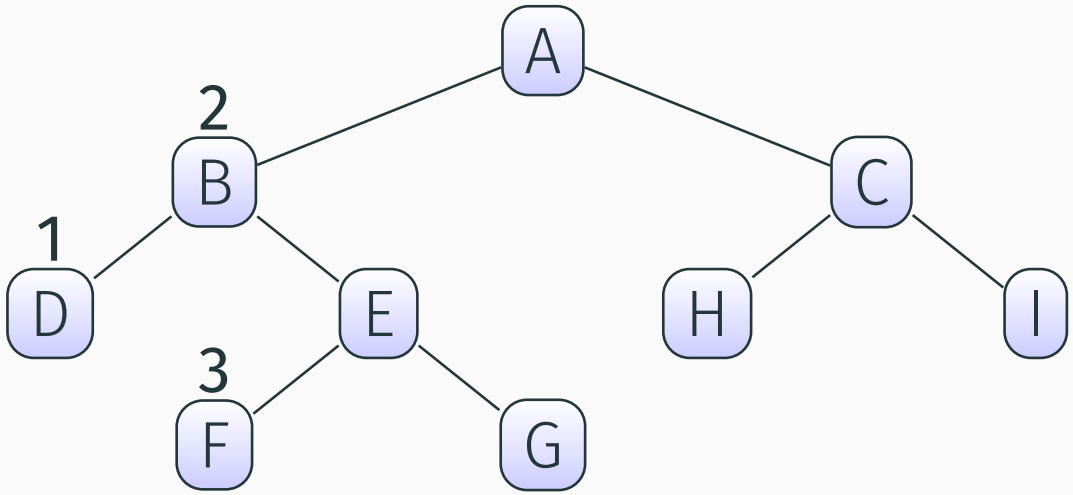
In an inorder traversal, a node is visited after its left subtree and before its right subtree



Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

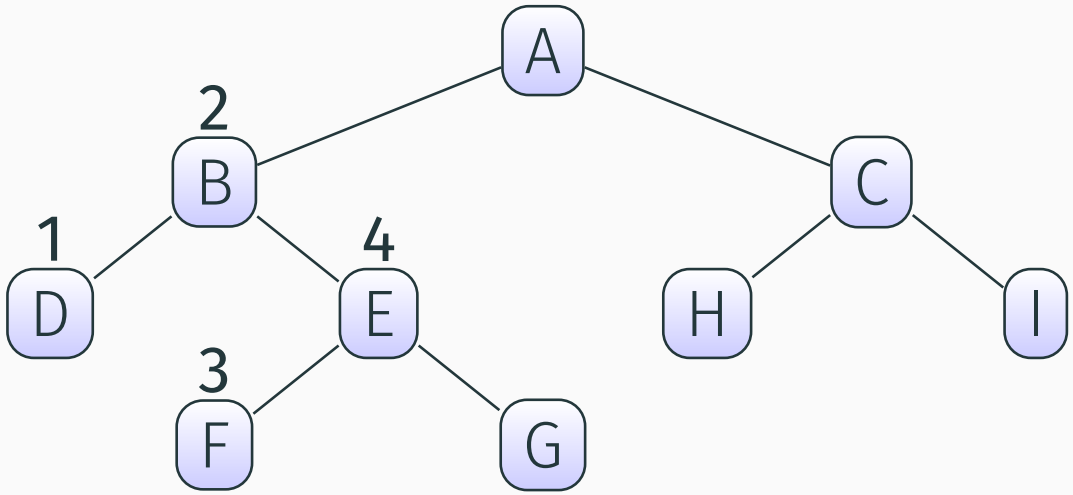
In an inorder traversal, a node is visited after its left subtree and before its right subtree



Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

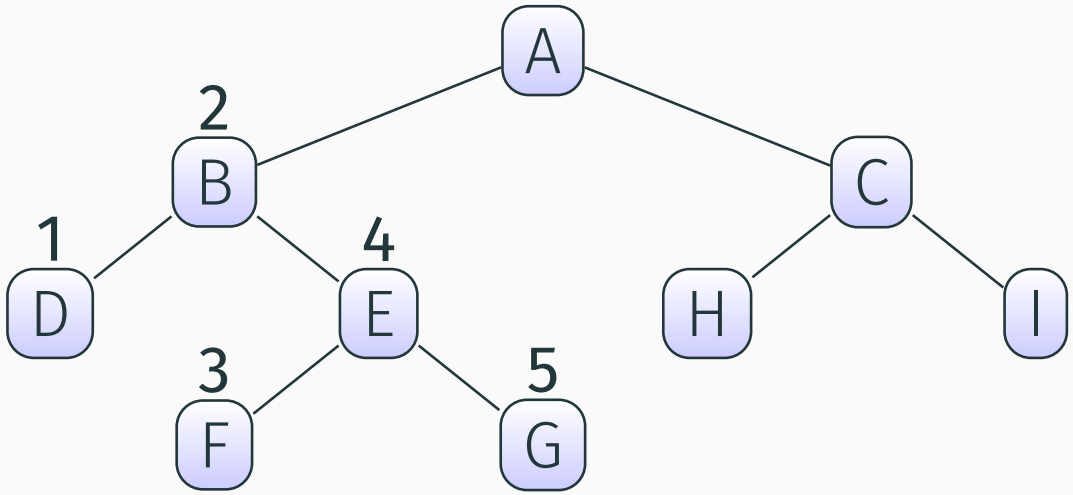
In an inorder traversal, a node is visited after its left subtree and before its right subtree



Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

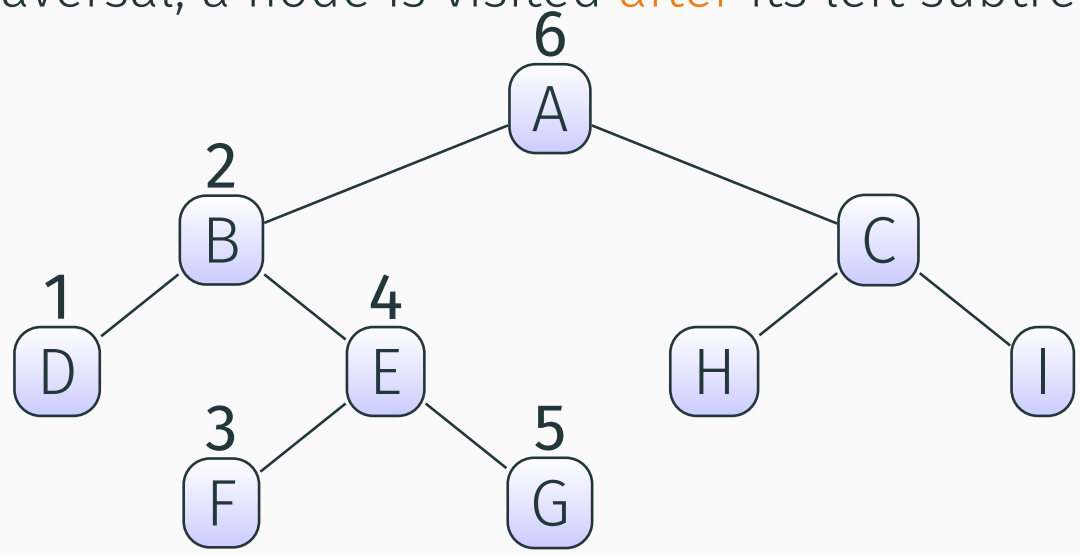
In an inorder traversal, a node is visited after its left subtree and before its right subtree



Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

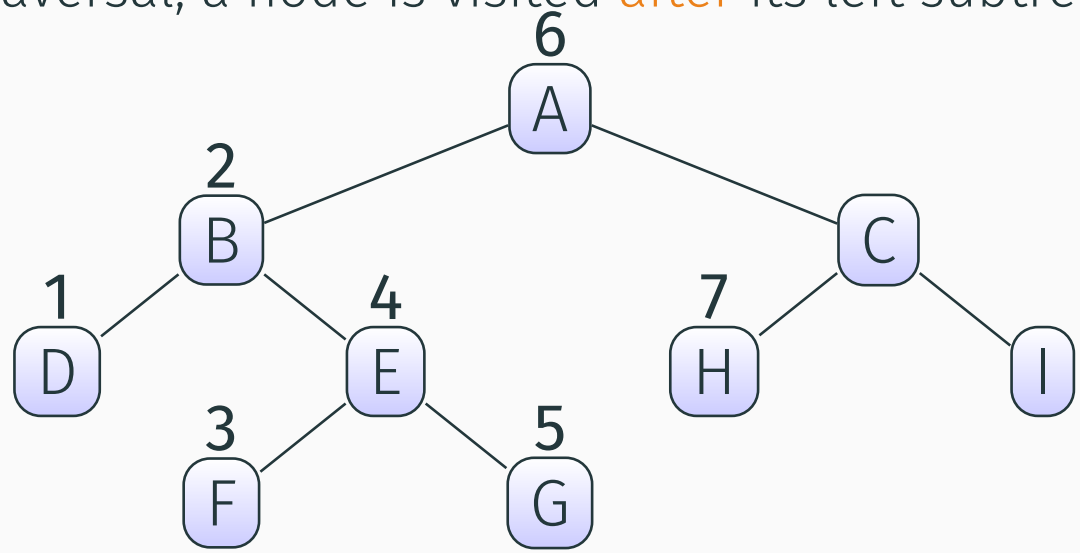
In an inorder traversal, a node is visited after its left subtree and before its right subtree



Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

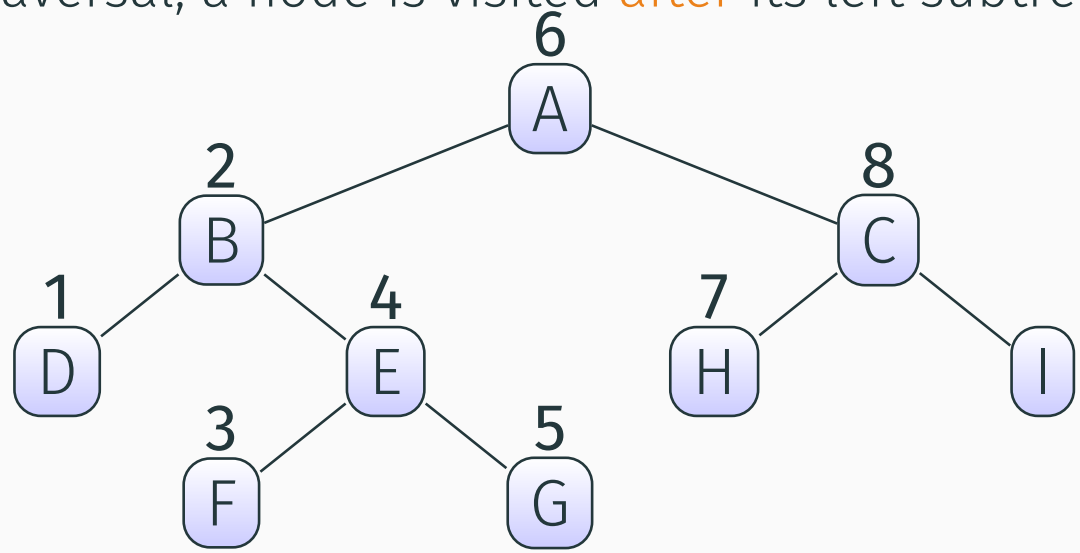
In an inorder traversal, a node is visited after its left subtree and before its right subtree



Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

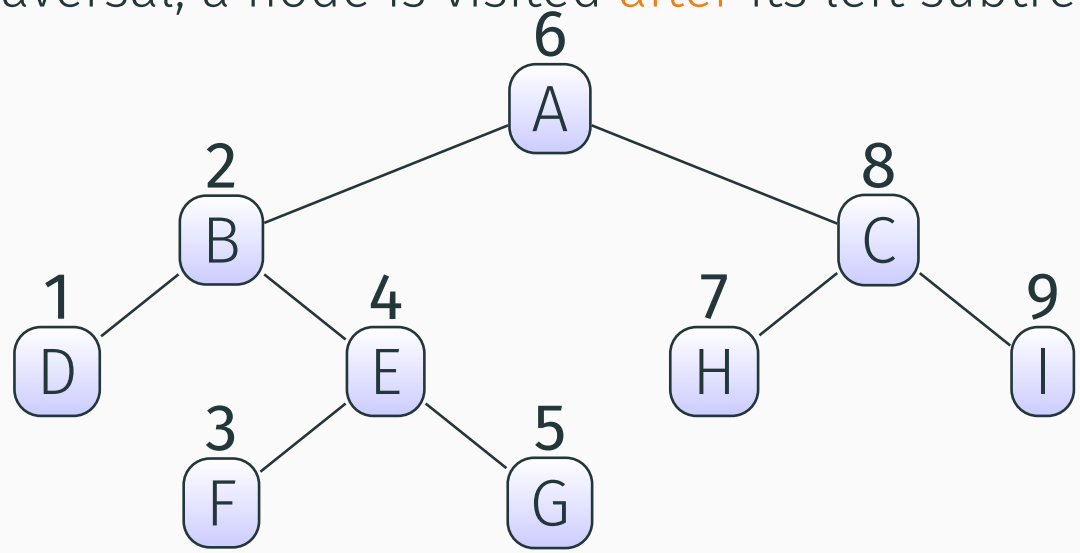
In an inorder traversal, a node is visited after its left subtree and before its right subtree



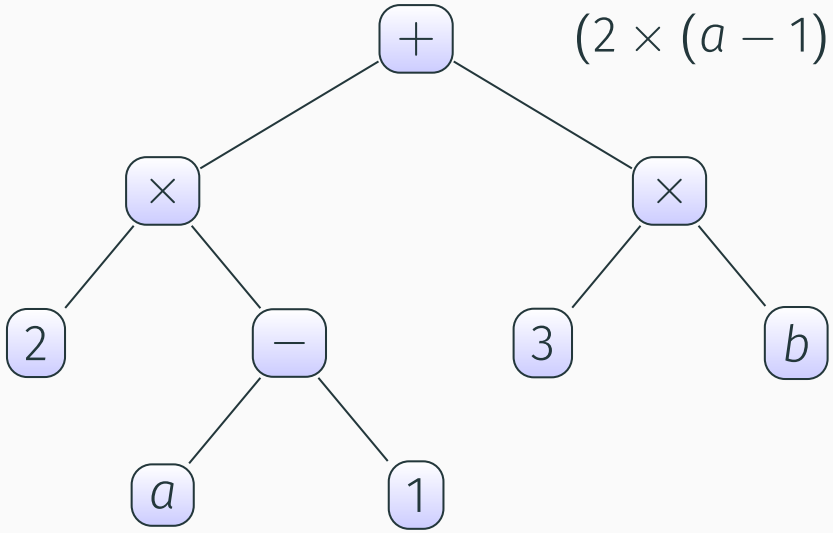
Inorder Traversal

```
1  Algorithm inOrder(v)
2
3  if left (v) ≠ null
4    inOrder (left (v))
5  visit(v)
6  if right(v) ≠ null
7    inOrder (right (v))
```

In an inorder traversal, a node is visited after its left subtree and before its right subtree



Print and Evaluate Arithmetic Expressions



$(2 \times (a - 1) + (3 \times b))$

```
1  printExpression(v):
2  if left (v) ≠ null
3    print("(")
4    inOrder(left(v))
5  print(v.element())
6  if right(v) ≠ null
7    inOrder(right(v))
8    print (")")
```

```
1  evalExpr(v):
2  if isExternal(v)
3    return v.element()
4  else
5    x = evalExpr(left(v))
6    y = evalExpr(right(v))
7    * = operator stored at v
8  return x * y
```

Binary tree

Implementation of Binary Tree

Array-Based Representation of Binary Trees

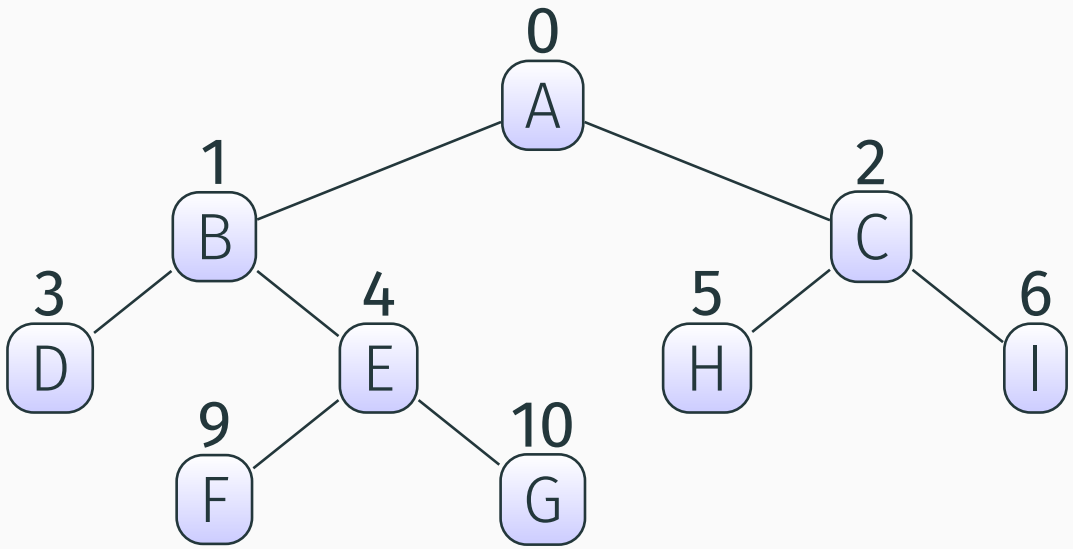
Node v is stored at A[rank(v)]

- $rank(root) = 0$
- if node is the left child of parent(node),

$$rank(node) = 2 \cdot rank(parent(node)) + 1$$

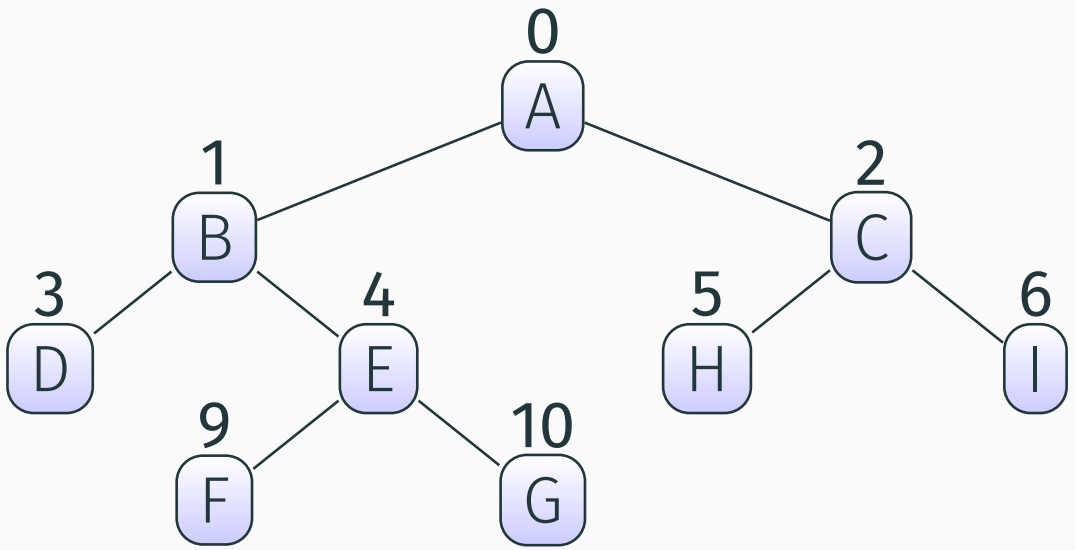
- if node is the right child of parent(node),

$$rank(node) = 2 \cdot rank(parent(node)) + 2$$



Array-Based Representation of Binary Trees

0	1	2	3	4	5	6	7	8	9	10	11
A	B	C	D	E	H	I			F	G	



Comparison

- Linked Structure:
  - Requires explicit representation of 3 links per position:

    parent, left child, right child
  - Data structure grows as needed – no wasted space.
- Array:
  - Parent and children are implicitly represented:

    Lower memory requirements per position
  - Memory requirements determined by height of tree. If tree is sparse, this is highly inefficient.

Thank you!

Questions?