

The image features a light beige background with decorative illustrations of tropical leaves in the corners. The top-left and bottom-right corners show large, detailed leaves with prominent veins, rendered in a light beige color with dark green outlines. The top-right and bottom-left corners feature solid green leaf shapes. The central text is positioned in the middle of the page.

# TREES

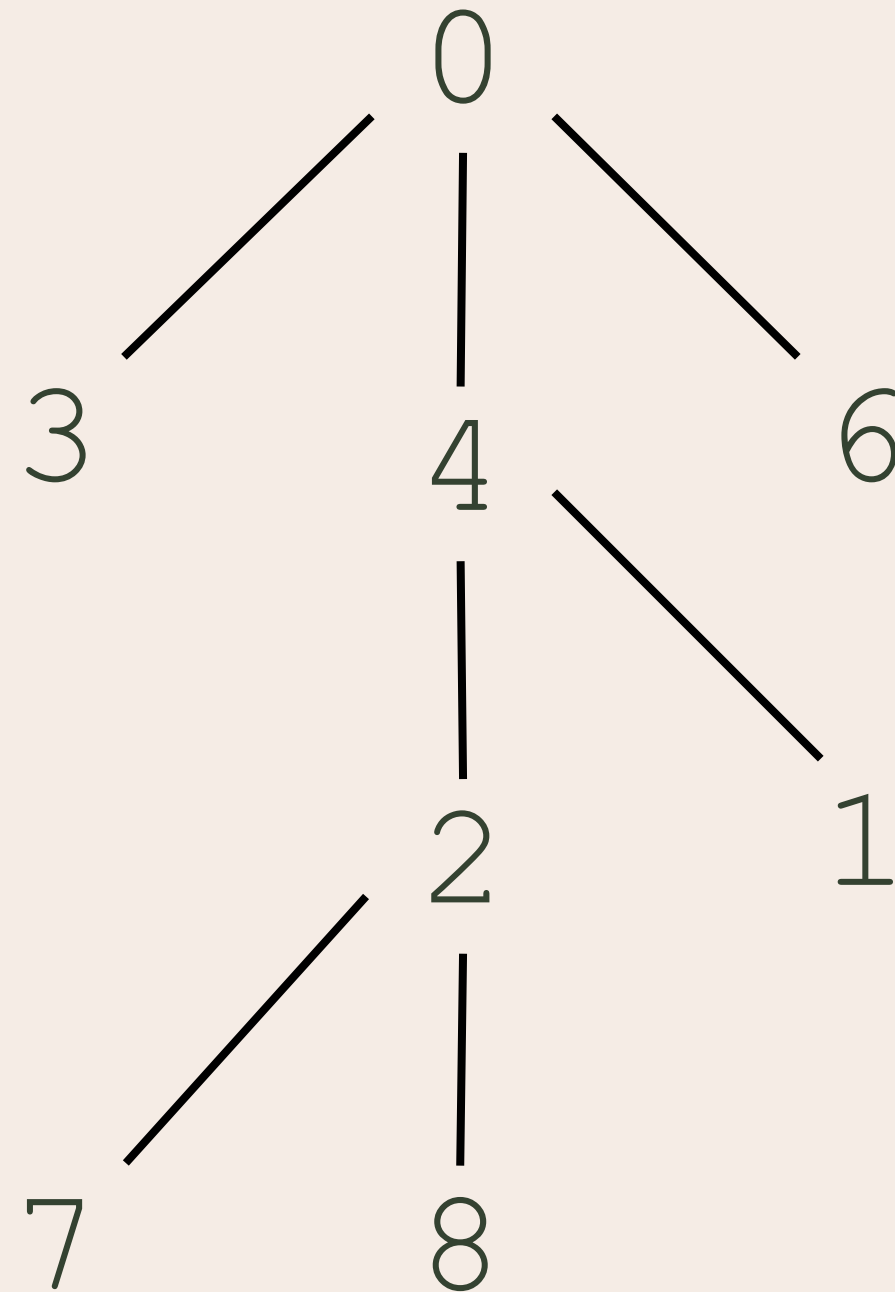
Janie Lane Sabado



# TREES

collection of elements known as nodes;  
connected undirected graph with no loops,  
no multiple edges, and no simple circuits

# PROPERTIES



ROOT

SIBLINGS

PATH

ANCESTOR

DESCENDANT

SUBTREE

HEIGHT

DEPTH

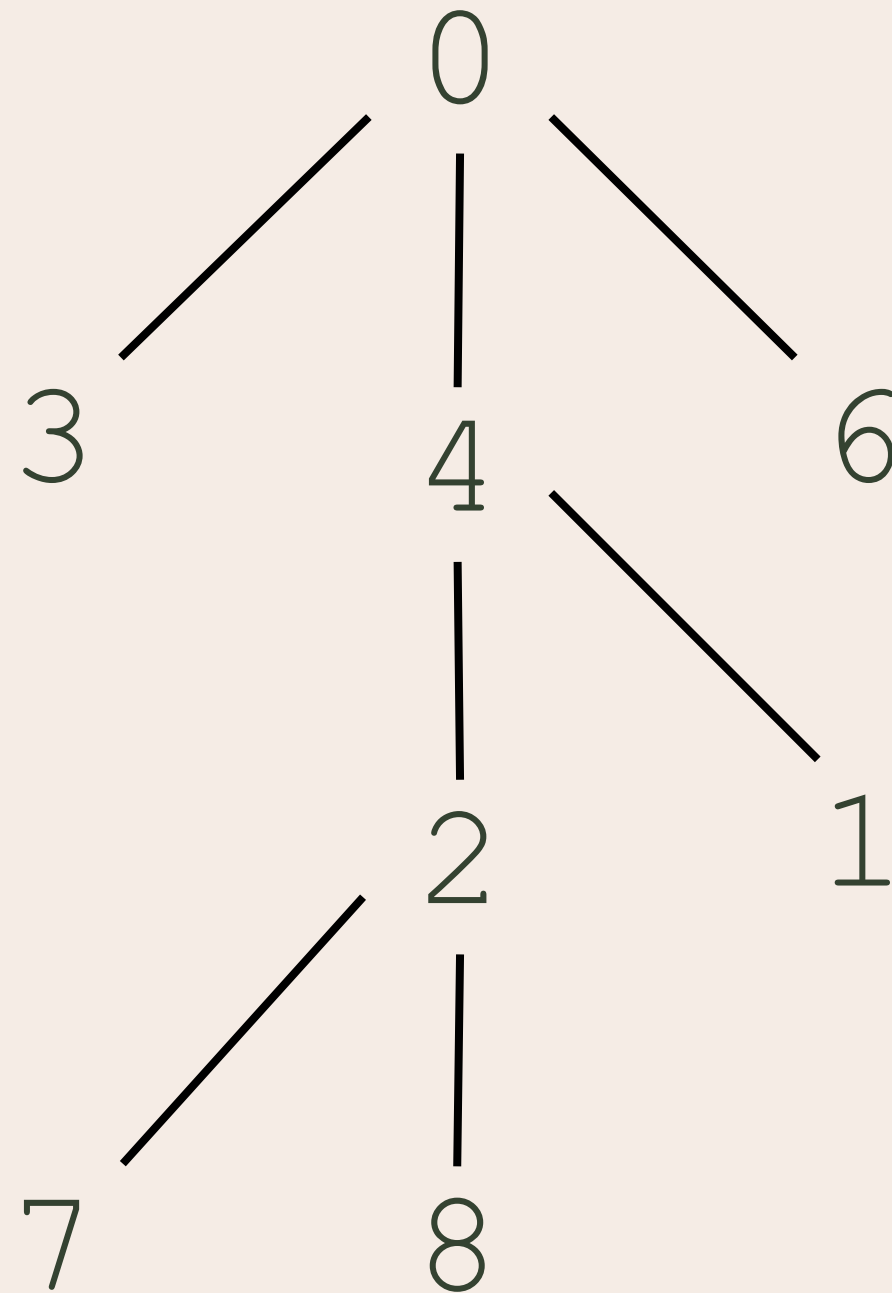
PARENT

CHILDREN

LEAF

COUSINS

# SYSTEMATIC ORDER



PREORDER

- root, left, right

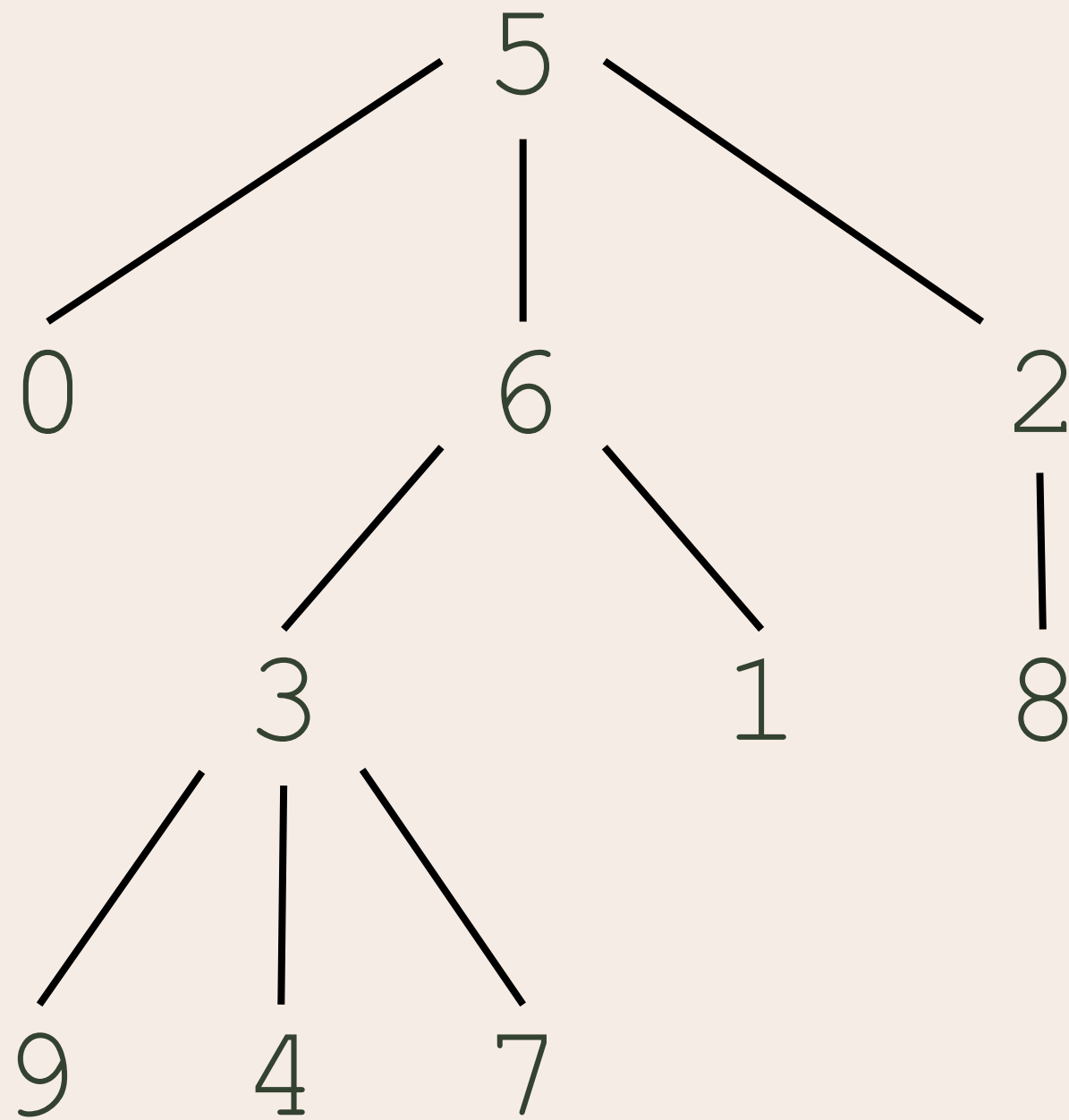
POSTORDER

- left, right, root

INORDER

- left, root, right

# SYSTEMATIC ORDER



PREORDER

- root, left, right

POSTORDER

- left, right, root

INORDER

- left, root, right



# LABEL & EXPRESSION TREE

Label Tree

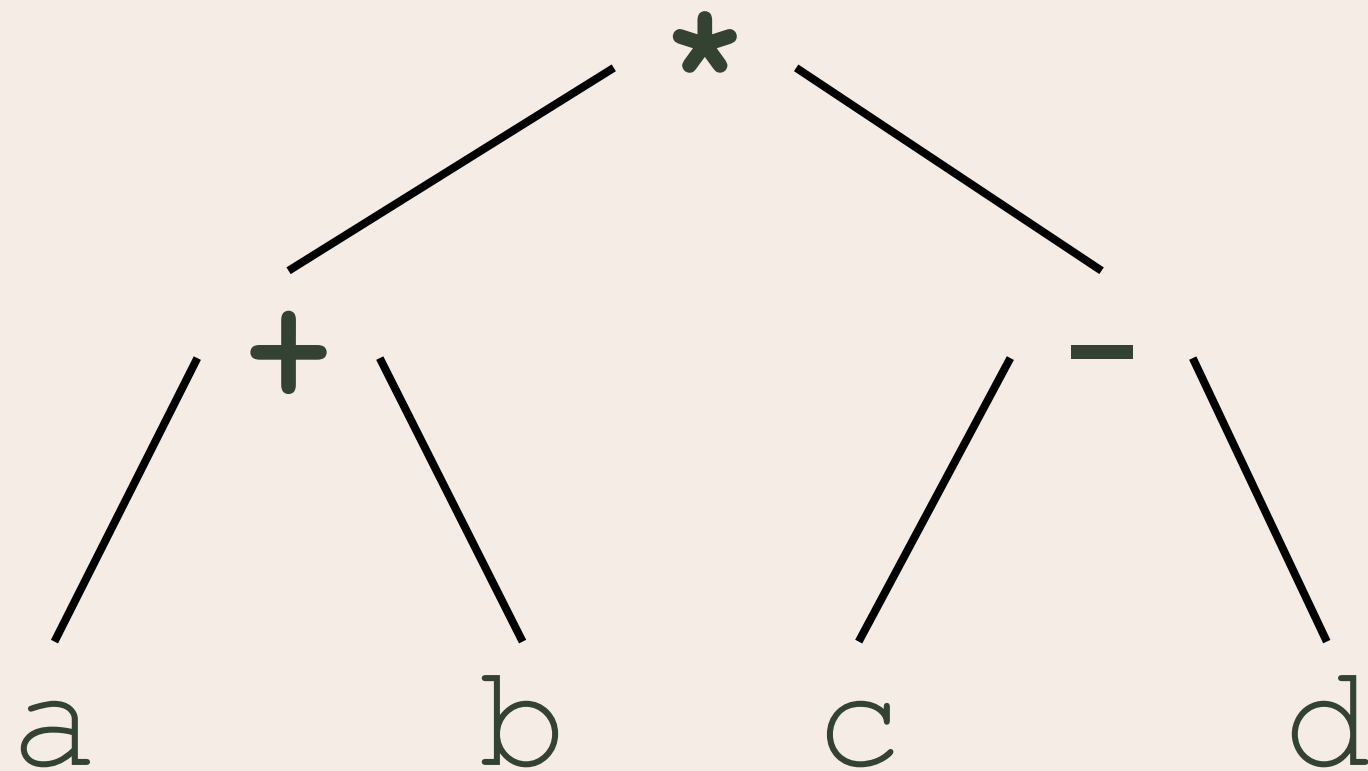
tree whose nodes have labels

Expression Tree

where every node is an expression or operand



# EXPRESSIONS



## PREFIX NOTATION

- listing of labels in preorder

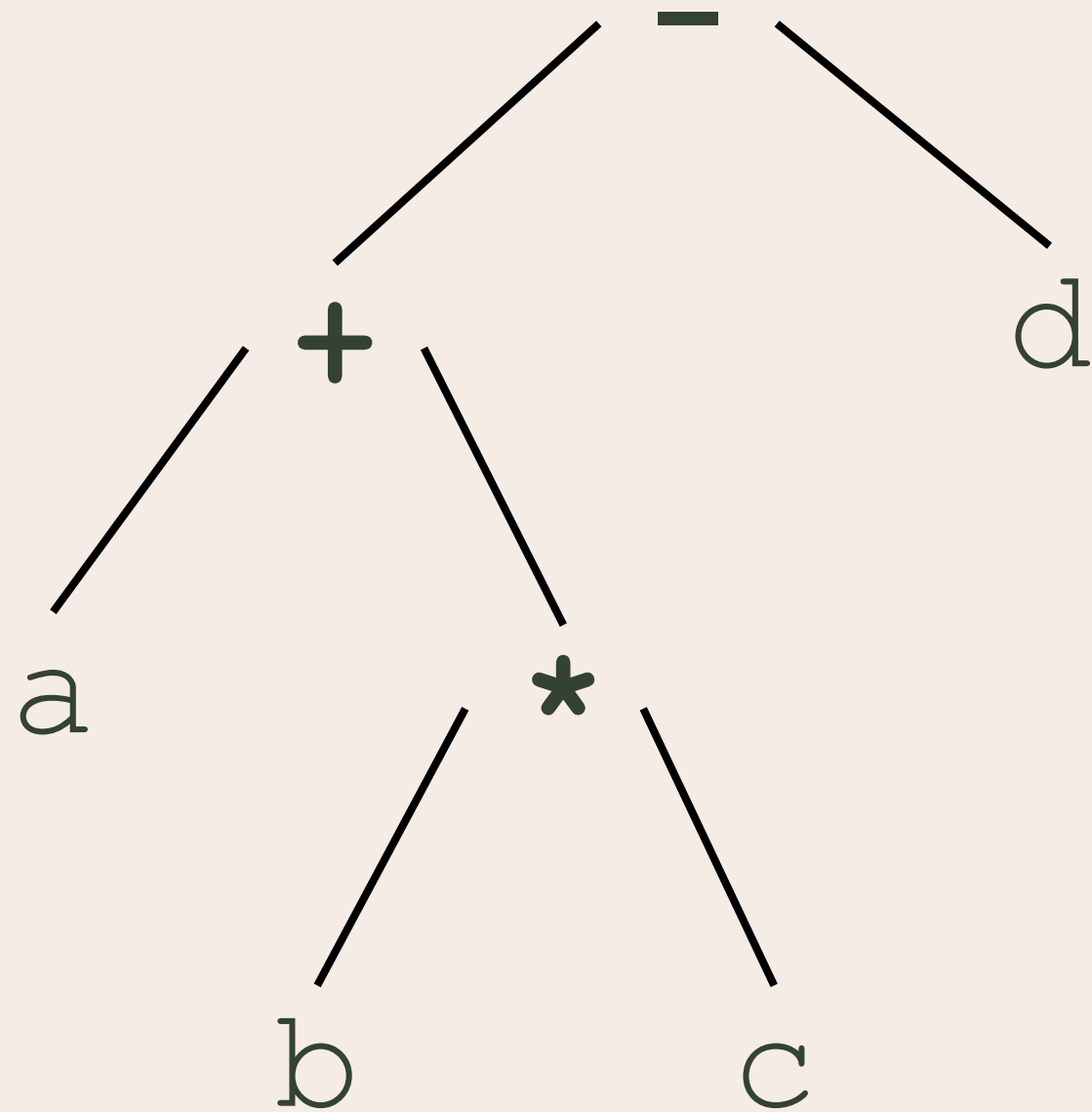
## POSTFIX NOTATION

- listing of labels in postorder

## INFIX NOTATION

- listing of labels in inorder

# EXPRESSIONS



## PREFIX NOTATION

- listing of labels in preorder

## POSTFIX NOTATION

- listing of labels in postorder

## INFIX NOTATION

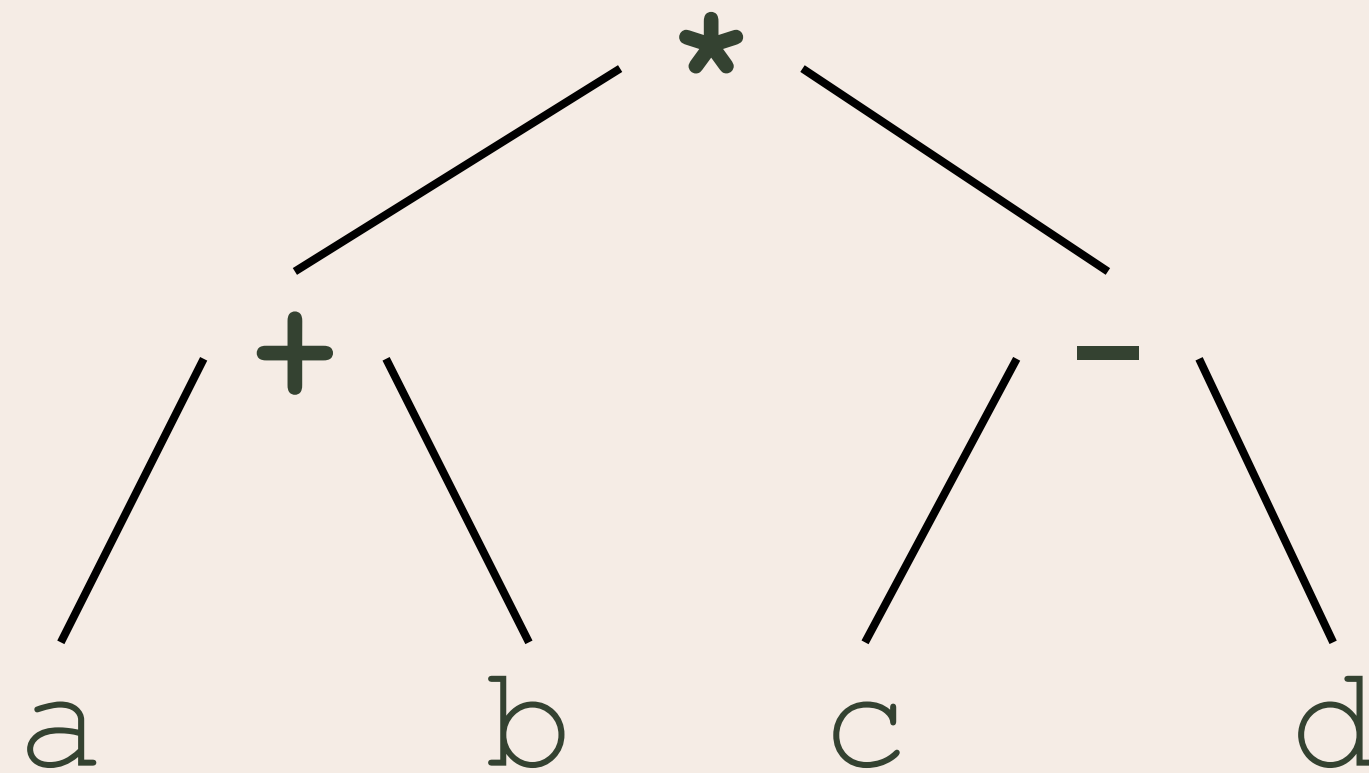
- listing of labels in inorder



# POLISH PREFIX

Given:  $*$ ,  $+$ ,  $a$ ,  $b$ ,  $-$ ,  $c$ ,  $d$

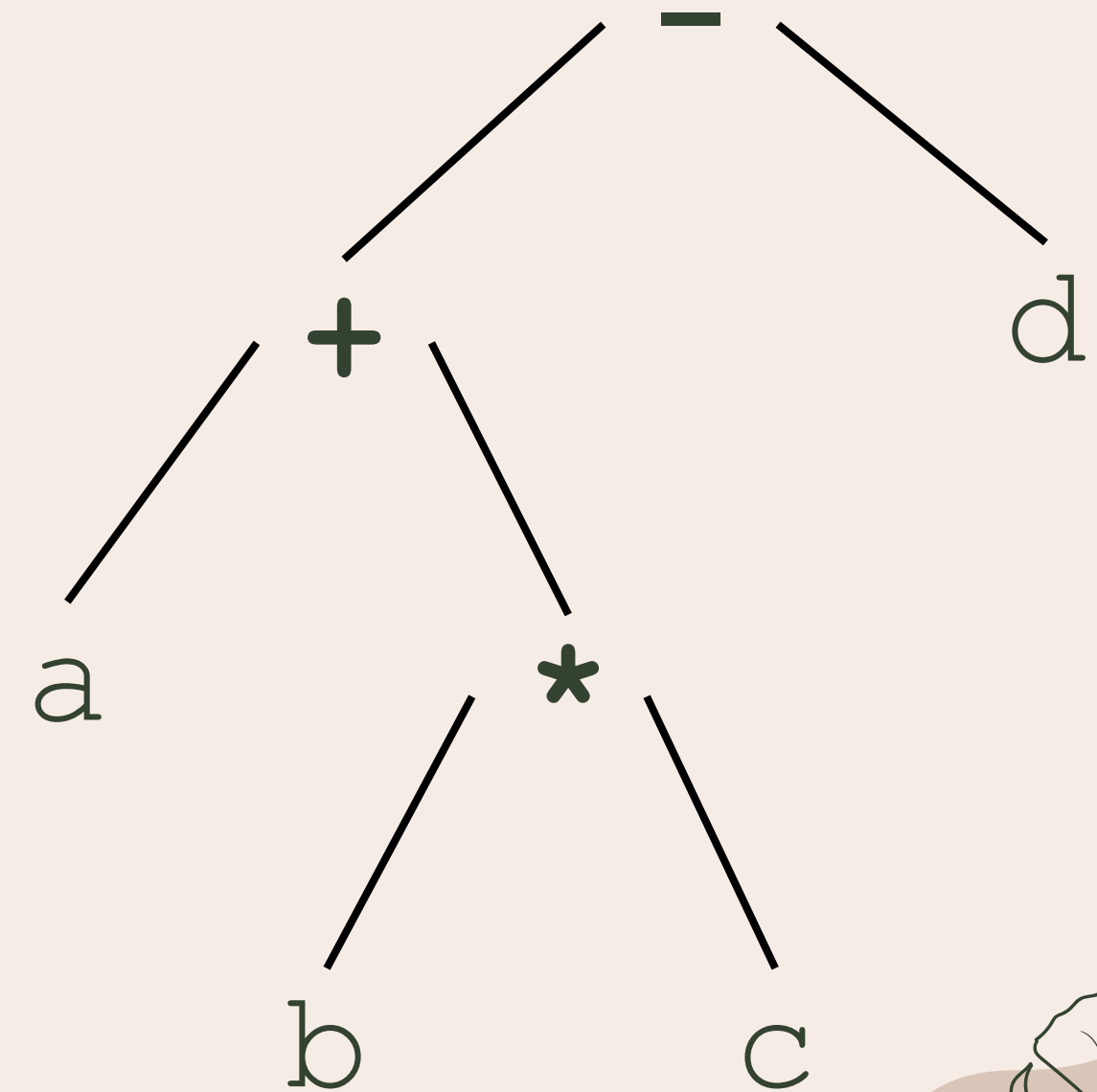
1. start from the left, find occurrences of operator-operand-operand
2. convert to trees with parent-left-right assignment
3. leftmost node is a root



# REVERSE POLISH PREFIX

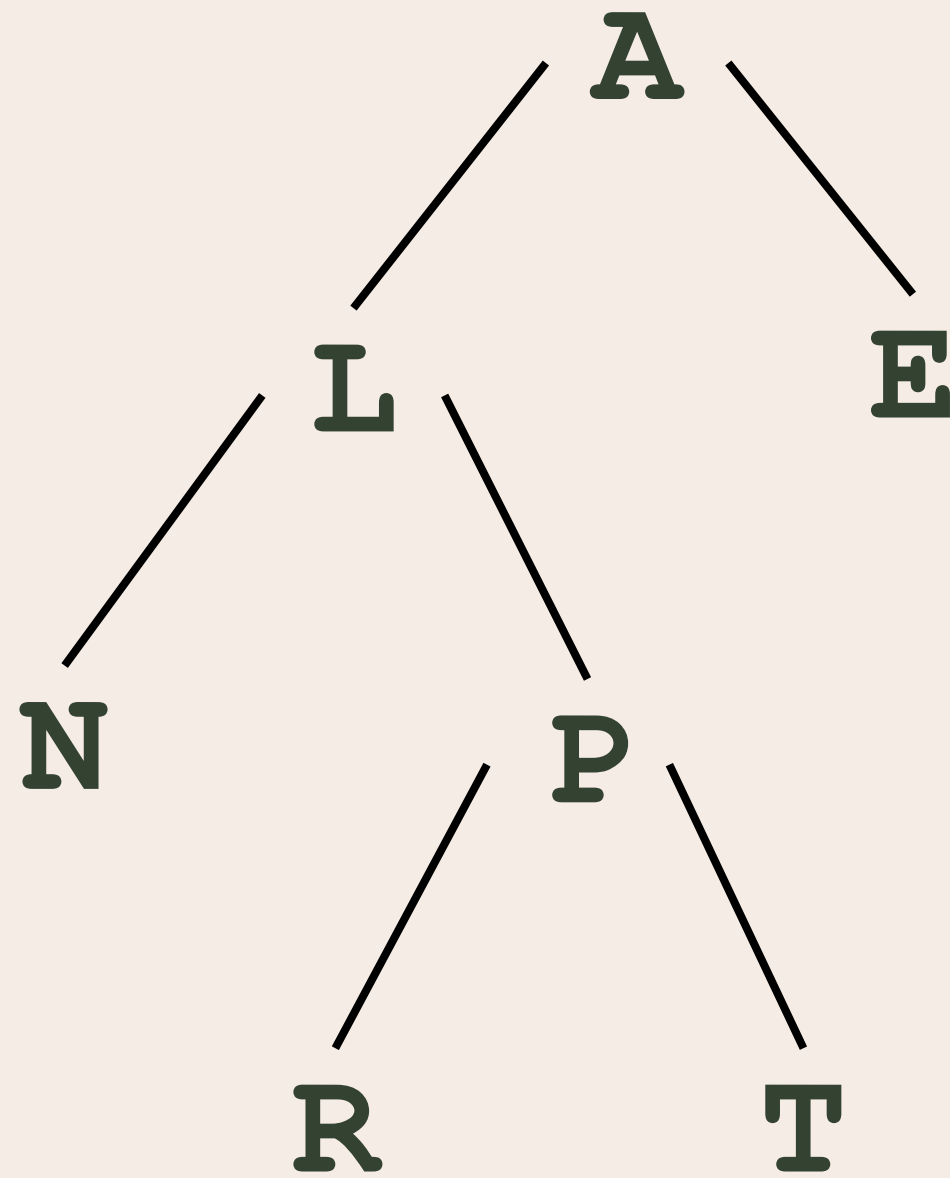
Given:  $a, b, c, *, +, d, -$

1. start from the left, find occurrences of operand-operand-operator
2. convert to trees with left-right-parent assignment
3. rightmost node is a root



# ADT TREES

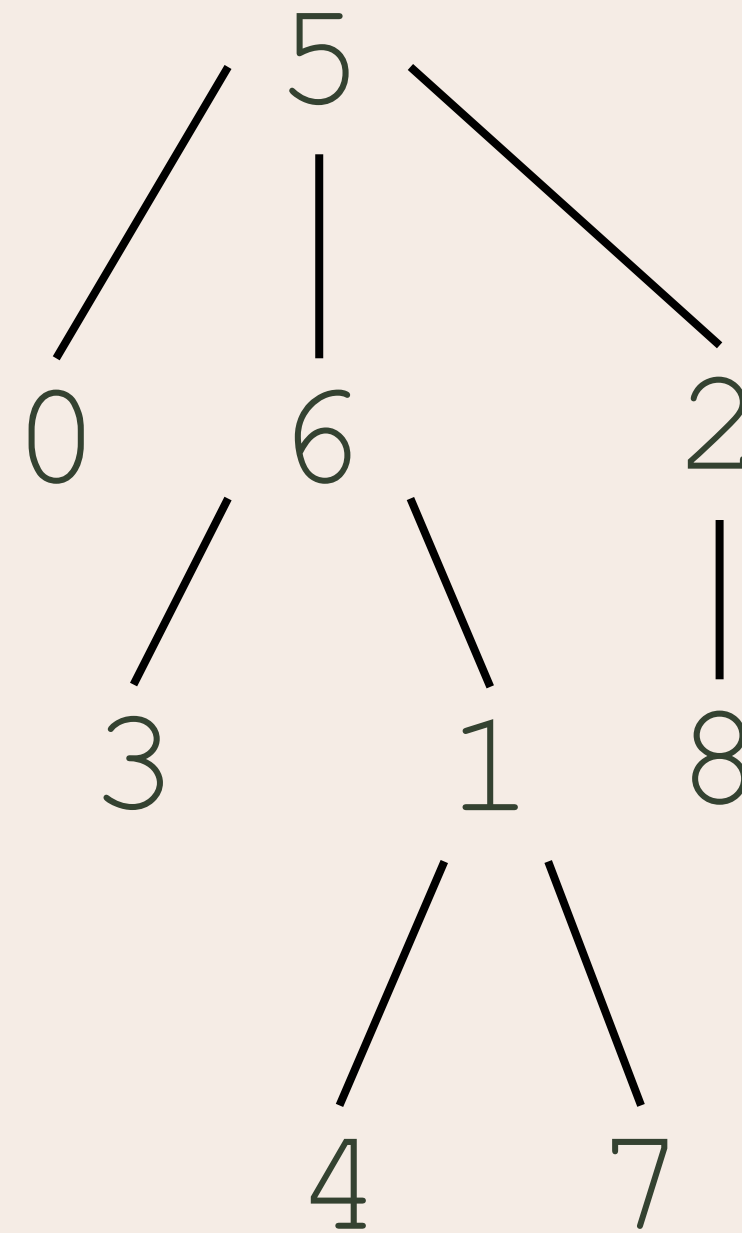
## OPERATIONS



- `parent(node n, tree T)`
  - returns the parent node of a given child node
- `left(tree T)`
  - returns the leftmost element
- `right(tree T)`
  - returns the rightmost element
- `label()`
- `create()`
- `initialize()`

# IMPLEMENTATION

0	5
1	6
2	5
3	6
4	1
5	-1
6	5
7	1
8	2
9	-2



## PARENT POINTER REPRESENTATION

- Array of parents
- indexes are the child node
- contents are the parent node

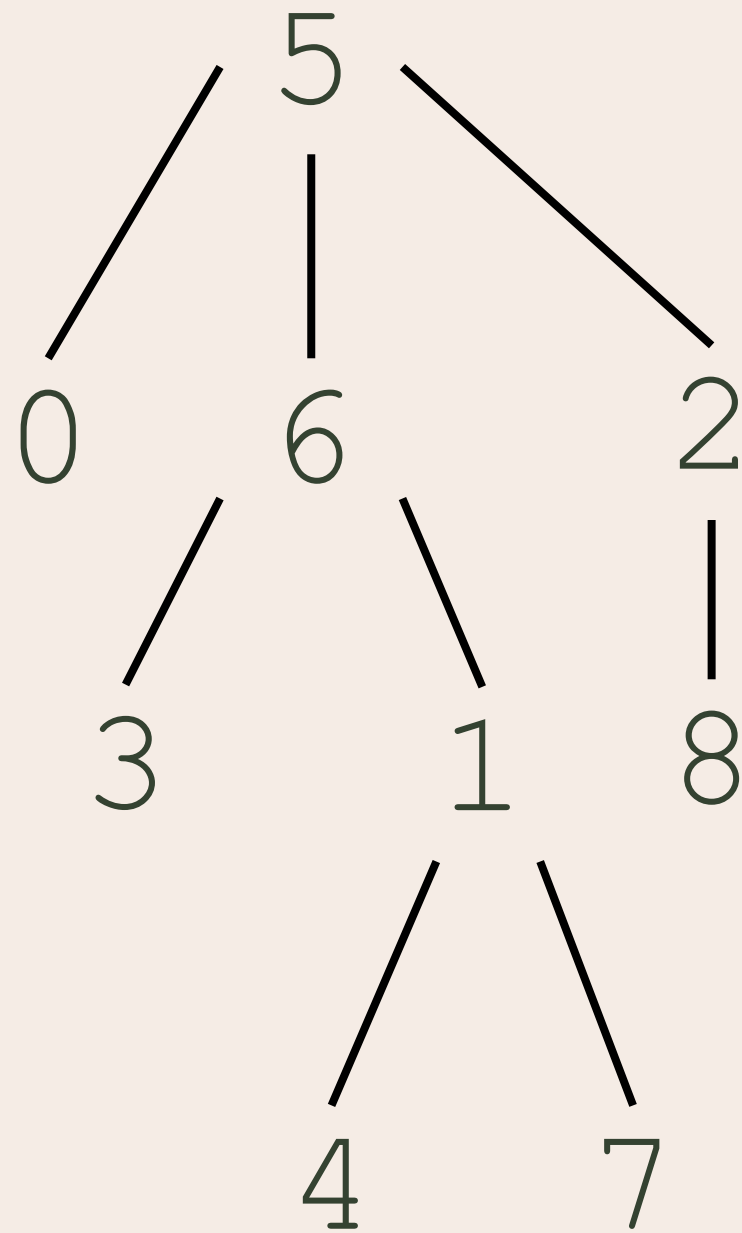
`T[x] == Parent`

`x == Child`

`T[x] = -1 //Root`

`T[x] = -2 //DNE`

# IMPLEMENTATION

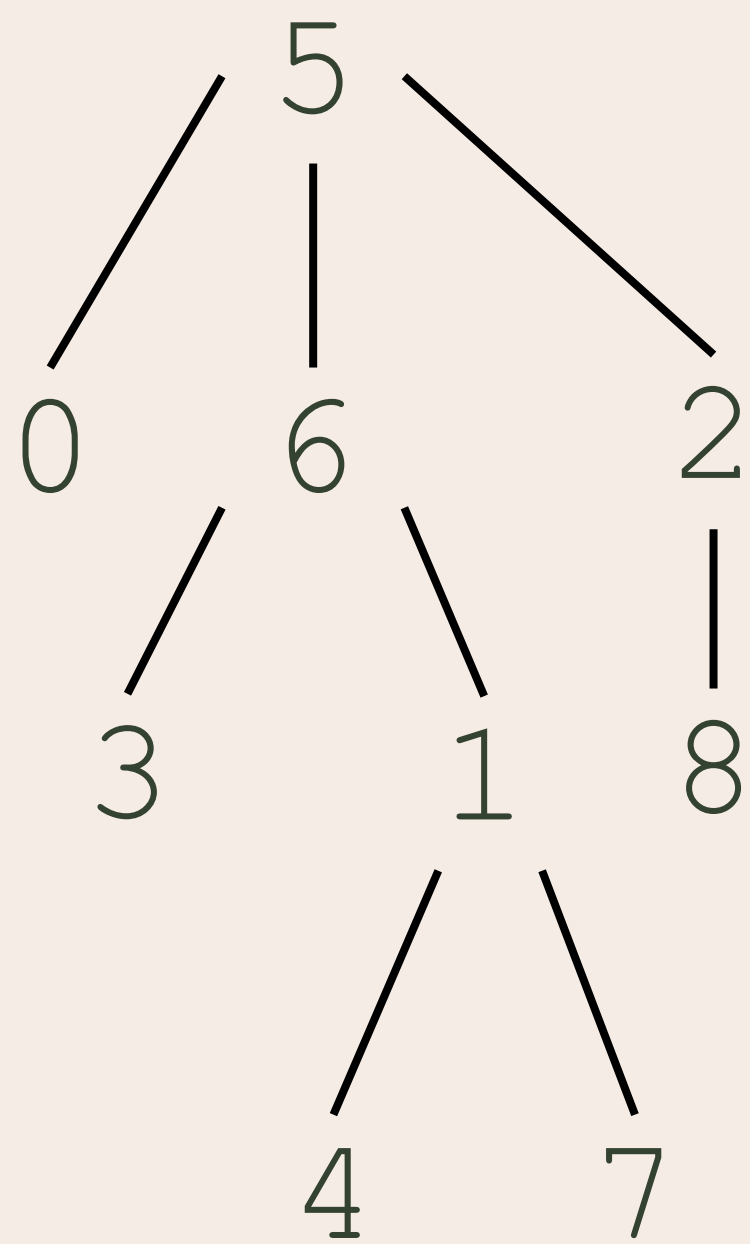


## REPRESENTATION BY LIST OF CHILDREN

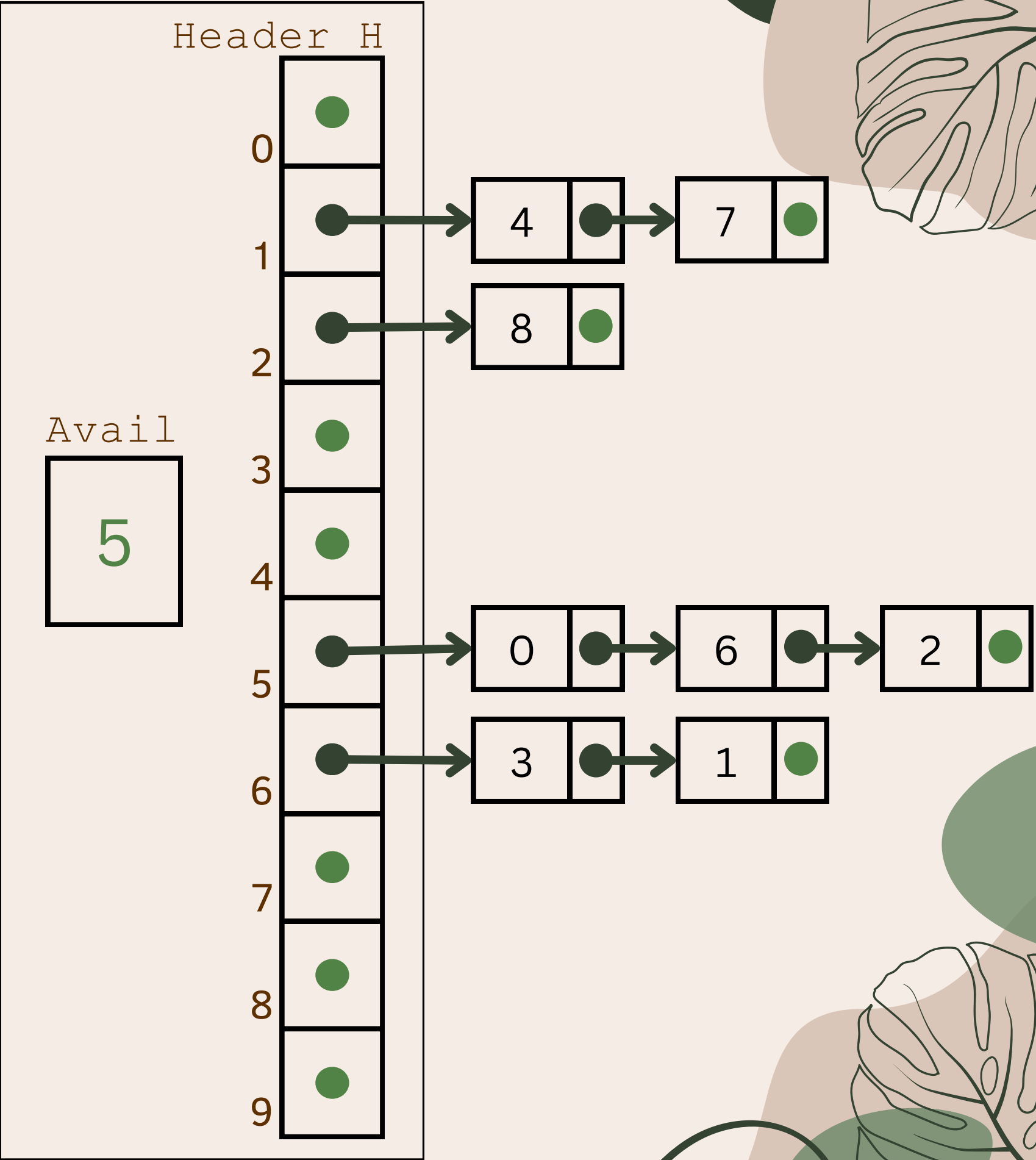
- Array of linked list
- hard to distinguish the root therefore the root field is created



# REPRESENTATION BY LIST OF CHILDREN



Trees T







# TRAVERSING TREES



# TREE TRAVERSAL

Breadth-First Traversal

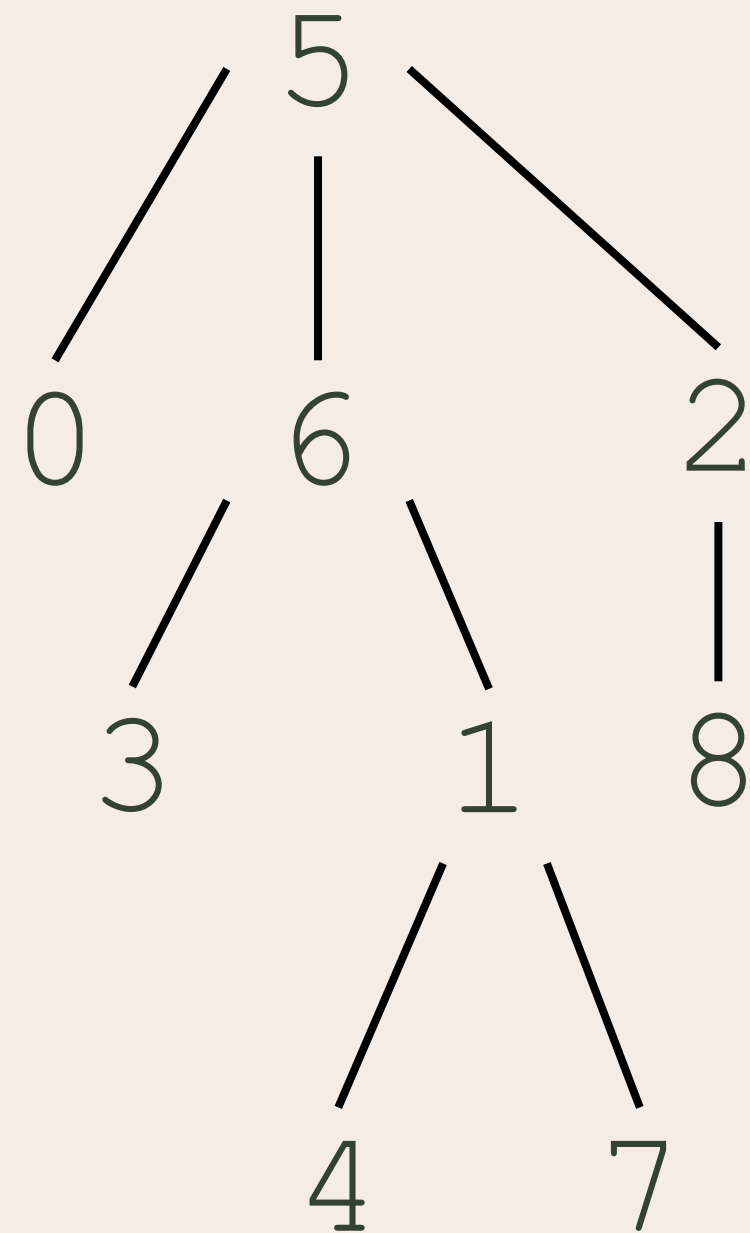
each node is explored level-by-level

Depth-First Traversal

explore nodes as far as possible

# TRAVERSALS

0	5
1	6
2	5
3	6
4	1
5	-1
6	5
7	1
8	2
9	-2



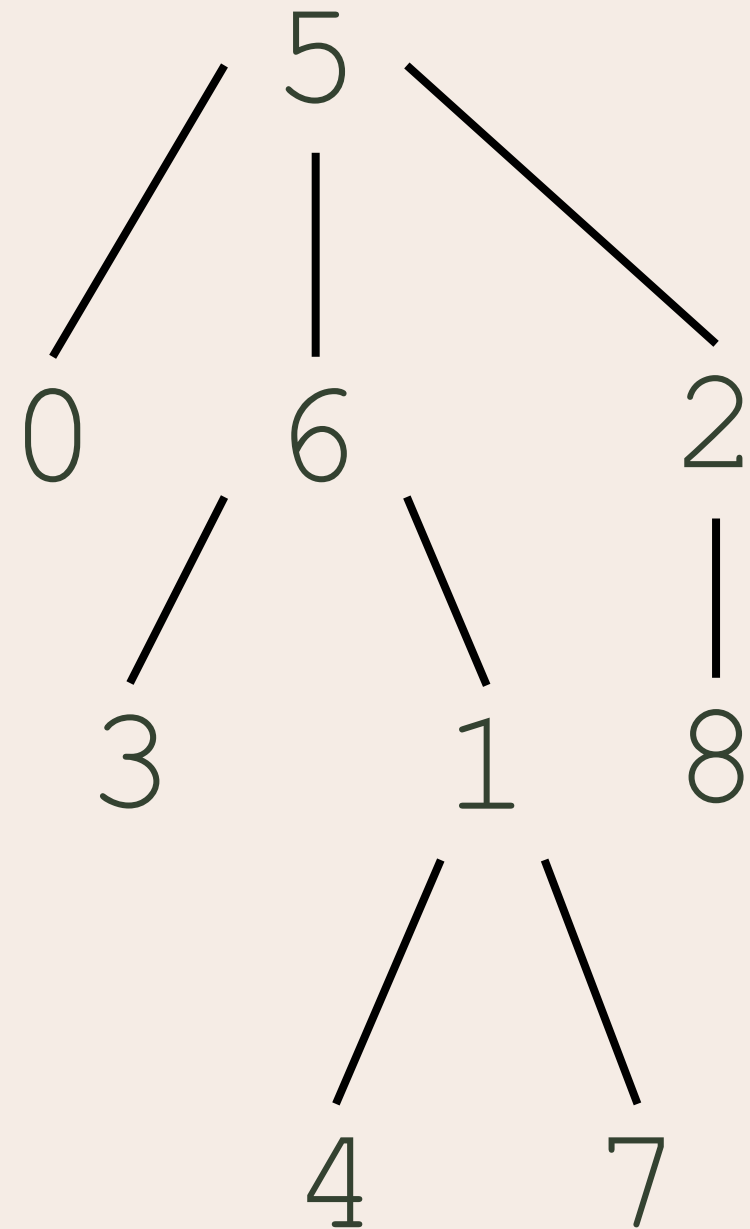
## Recursion

use recursive functions to traverse.  
Easier to implement, somewhat  
harder to trace

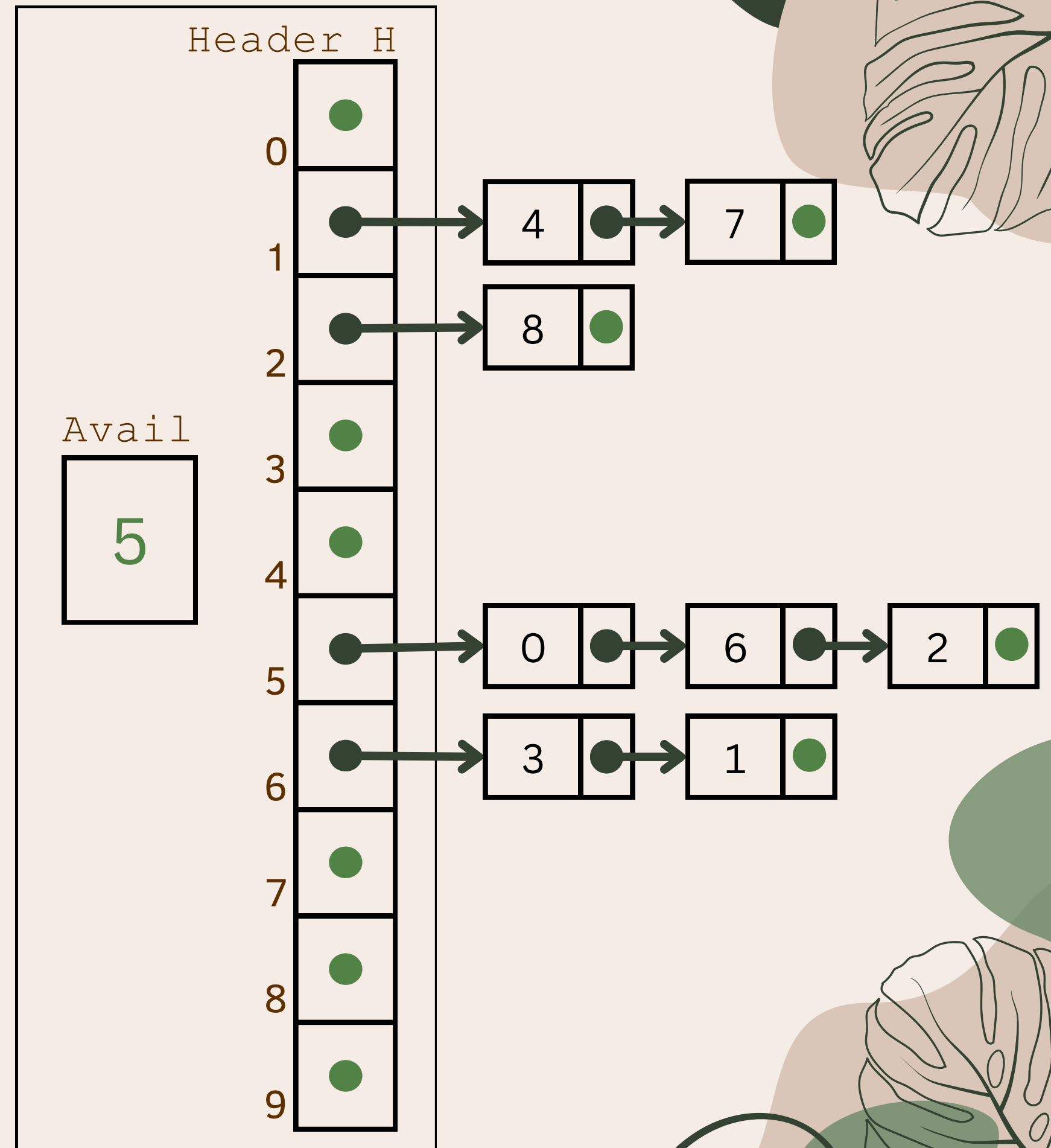
## Iterative

use loops to traverse the tree.  
Easier to trace, somewhat harder to  
implement.

# REPRESENTATION BY LIST OF CHILDREN



Trees T

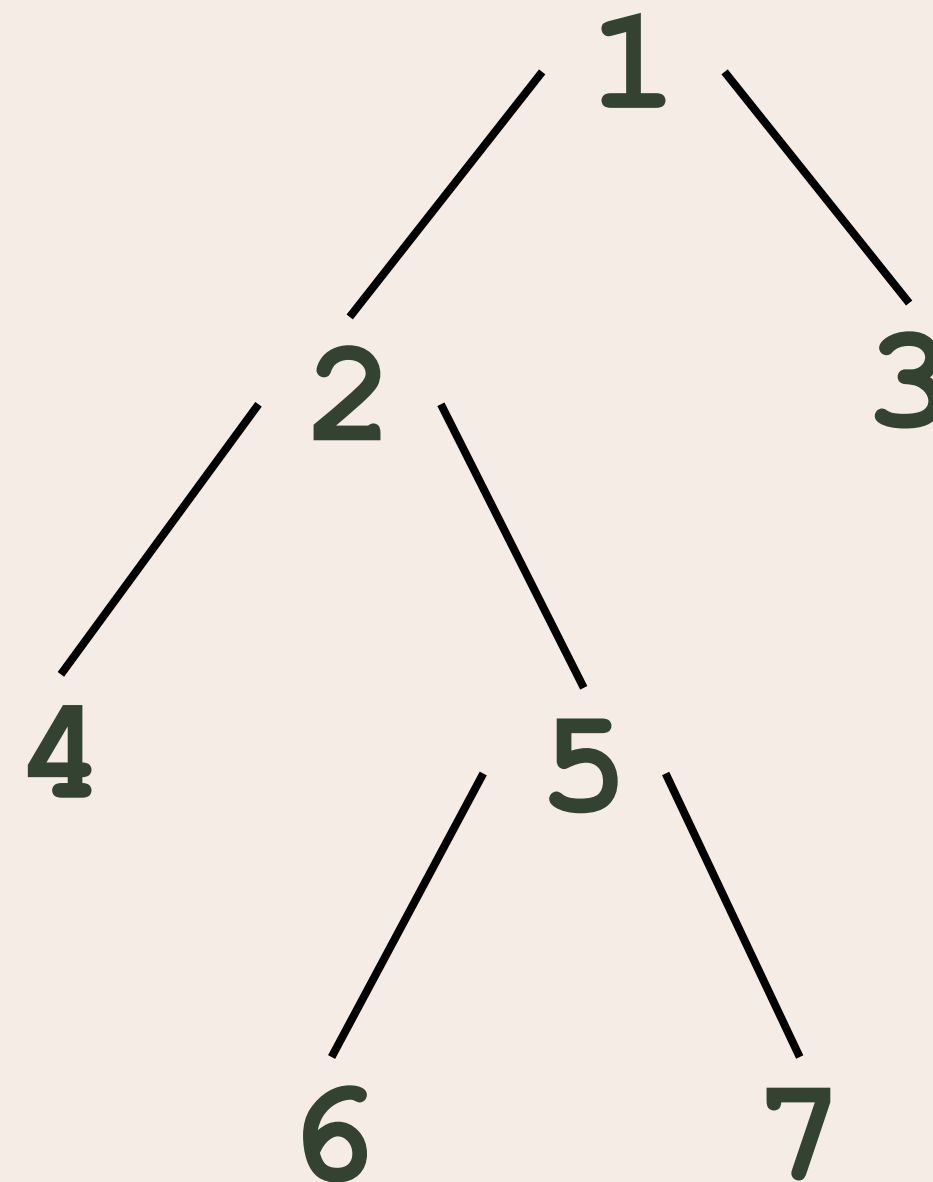




# BINARY TREES

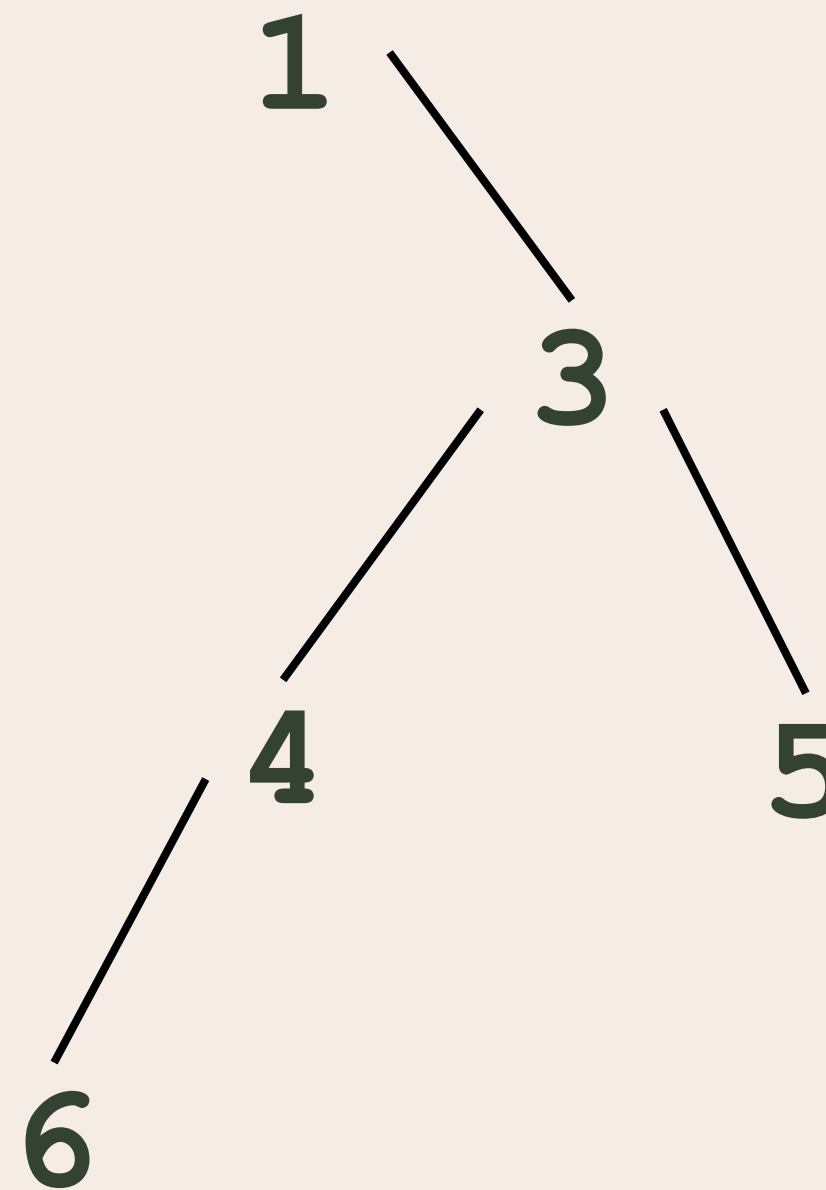
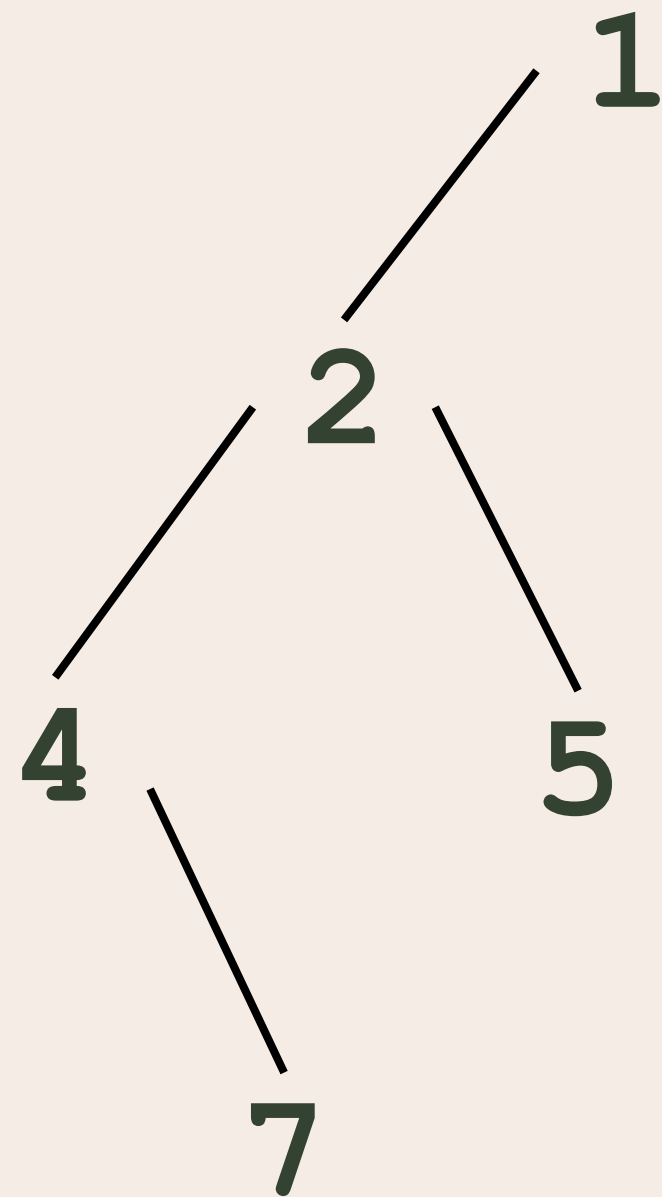
# BINARY TREES

- a tree that is empty OR
- a tree which every node has either
  - no child node
  - a left child
  - a right child
  - both a left and right child

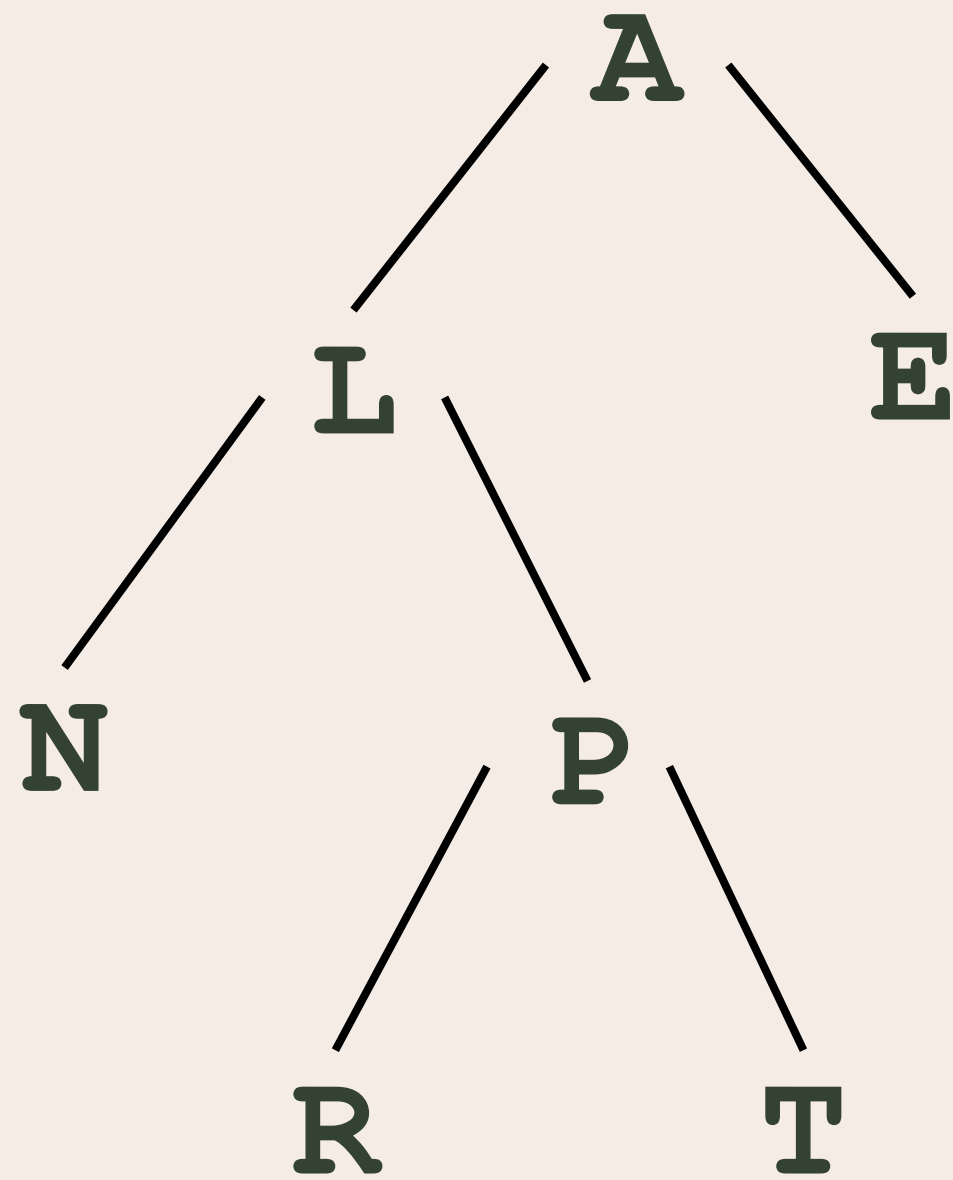




# BINARY TREES



# BINARY TREES



The image features a minimalist design with stylized tropical leaves in the corners. The leaves are rendered in a muted green color with black outlines, set against a light beige background. The leaves are arranged in a way that they appear to be framing the central text. The text "THANK YOU" is written in a bold, serif font, centered on the page. The overall aesthetic is clean and modern, with a focus on natural elements.

THANK  
YOU