Which of the following is true?

A. Tree is an ADT just like list.
B. Tree is a data structure just like linked list, vector, and array.
C. A and B
D. None of the above
E. Whatever
Today’s Outline

- Binary Tree
- Internal Structure
- Properties
- Traversals
Implementing BinaryTree

- BinaryTree<E> class
  - Instance variables
    - BinaryTree: parent, left, right
    - E: value
  - left and right are never null
    - If no child, they point to an “empty” tree
    - Empty tree T has value null, parent null, left = right = T
  - Only empty tree nodes have null value
Example

```
*  
/  
4   2

null

"*

left   right

parent

"4"

left   right

parent

"2"

left   right

EMPTY   EMPTY   EMPTY   EMPTY   EMPTY
```
Representing Arbitrary Trees

• What if nodes can have many children?
  • Example: Game trees

• Replace left/right node references with a list of children (Vector, SLL, etc)

\[ \text{getLeft()} \rightarrow \text{getChildren()} \]
Representing Knowledge

- Trees can be used to represent knowledge
- We often call these decision trees
  - Leaf: object
  - Internal node: question to distinguish objects
  - Move down decision tree until we reach a leaf node
Today’s Outline

• Binary Tree
  • Internal Structure
  • Properties
  • Traversals
BT Questions/Proofs

• Some of the properties
  • The number of nodes at depth $n$ is at most $2^n$.
  • A tree with $n$ nodes has exactly $n-1$ edges.
  • The number of nodes in a tree of height $n$ is at most $2^{(n+1)}-1$. 
Prove: Number of nodes at depth \( n \) is at most \( 2^n \).

Base case: \( n = 0 \), \( 2^0 = 1 \) \( \checkmark \)

Assume at depth \( n \), there are at most \( 2^n \) nodes.

- Each of the nodes at depth \( n \) has at most 2 children.

\[ \sum_{i=0}^{n} 2^i \text{ nodes by IH} \]

Thus, at depth \( n+1 \), there are at most \( 2 \times 2^n = 2^{n+1} \) nodes.
Today’s Outline

• Binary Tree
  • Internal Structure
  • Properties
• Traversals
Tree Traversals

• In linear structures, there are only a few basic ways to traverse the data structure
  • Start at one end and visit each element
  • Start at the other end and visit each element

• How do we traverse binary trees?
  • (At least) four reasonable mechanisms
Tree Traversals

In-order: “left, node, right”
- Kurt, Arselyne, Candy, Wyatt, Isaki, Alex

Pre-order: “node, left, right”
- Wyatt, Arselyne, Kurt, Candy, Isaki, Alex

Post-order: “left, right, node”
- Kurt, Candy, Arselyne, Alex, Isaki, Wyatt

Level-order: visit all nodes at depth i before depth i+1
- Wyatt, Arselyne, Isaki, Kurt, Candy, Alex
Tree Traversals

In-order: 
2 * 3 + 7

Pre-order: 
+ * 2 3 7

Post-order: 
2 3 * 7 +

Level-order: 
+ * 7 2 3
public void preOrder(BinaryTree t) {
    if (t.isEmpty())
        return;
    doSomething(t);
    preOrder(t.left());
    preOrder(t.right());
}

Tree Traversals
Level-Order Traversal

1. Green
   2. Blue
   3. Violet
      4. Orange
      5. Yellow
         6. Indigo
         7. Red

GBVOURIR
public static <E> void levelOrder(BinaryTree<E> root) {
    if (t.isEmpty()) return;

    // The queue holds nodes for in-order processing
    Queue<BinaryTree<E>> q = new QueueList<BinaryTree<E>>();
    q.enqueue(root); // put root of tree in queue

    while (!q.isEmpty()) {
        BinaryTree<E> next = q.dequeue();
        doSomething(next);
        if (!next.left().isEmpty()) q.enqueue(next.left());
        if (!next.right().isEmpty()) q.enqueue(next.right());
    }
}