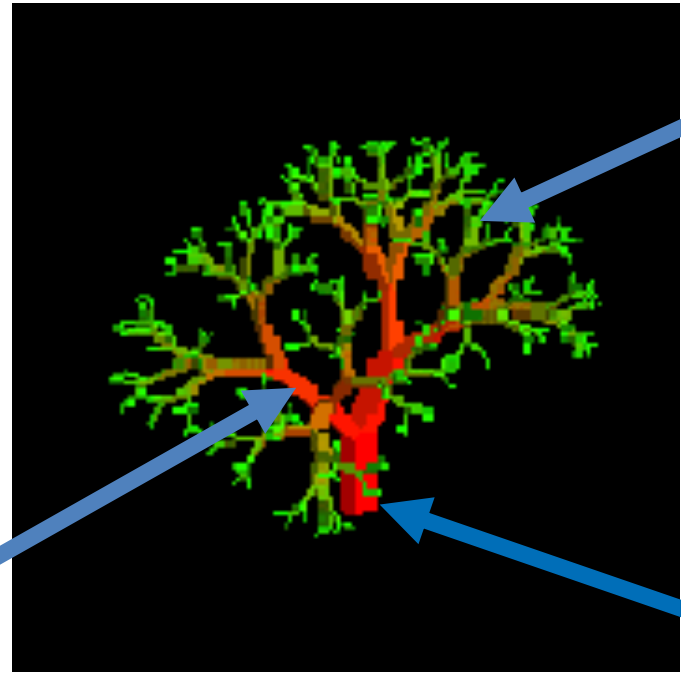
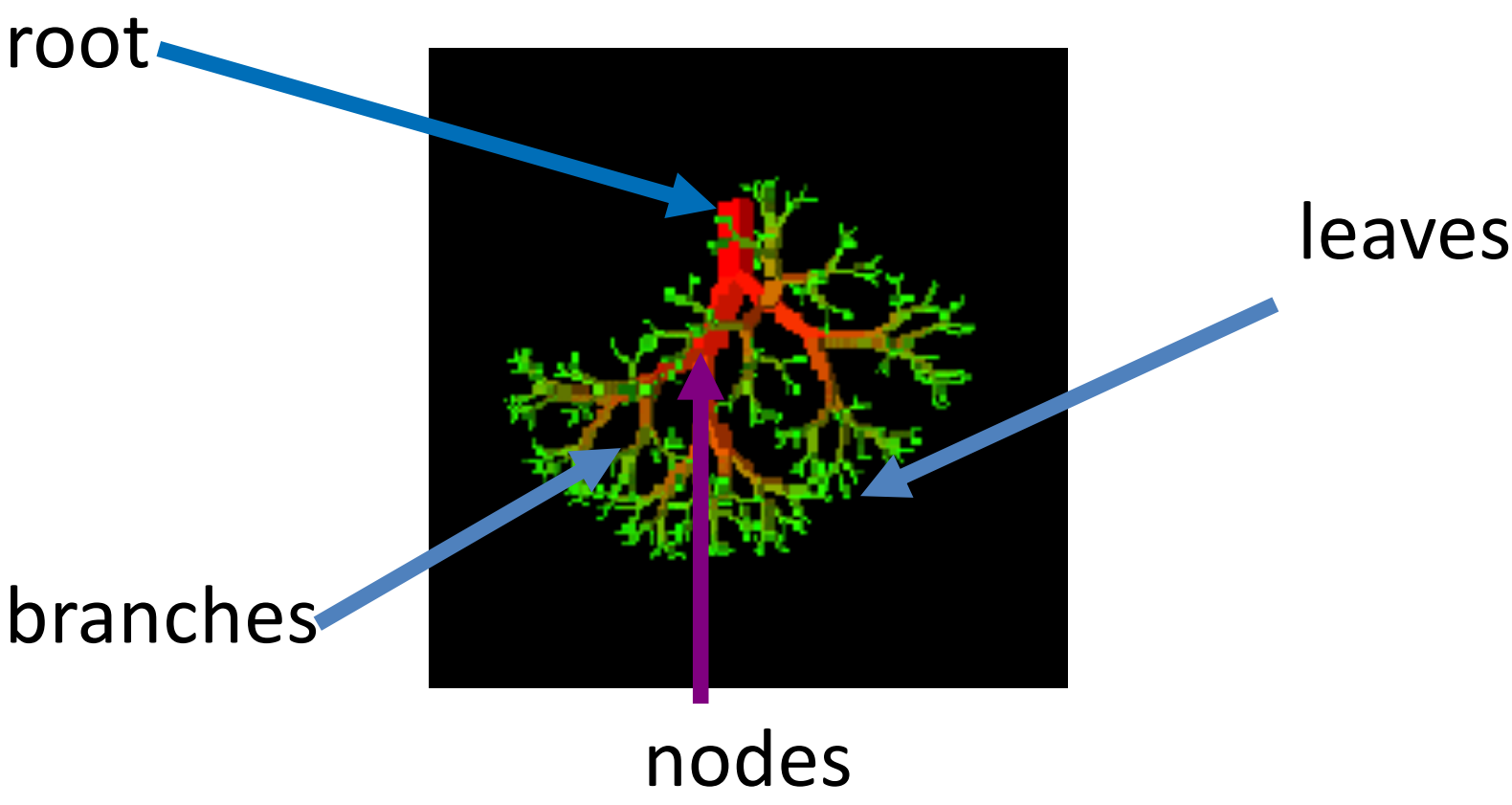
Tree

IIITS

Nature View of a Tree



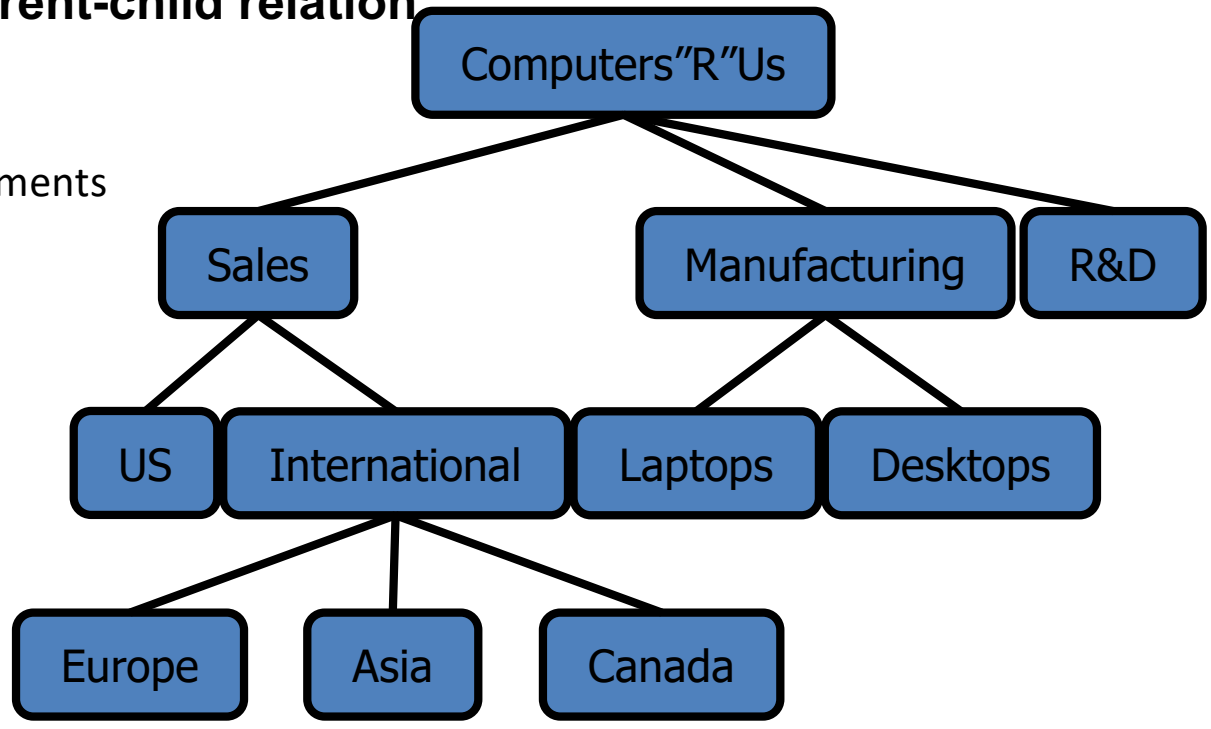
Computer Scientist's View



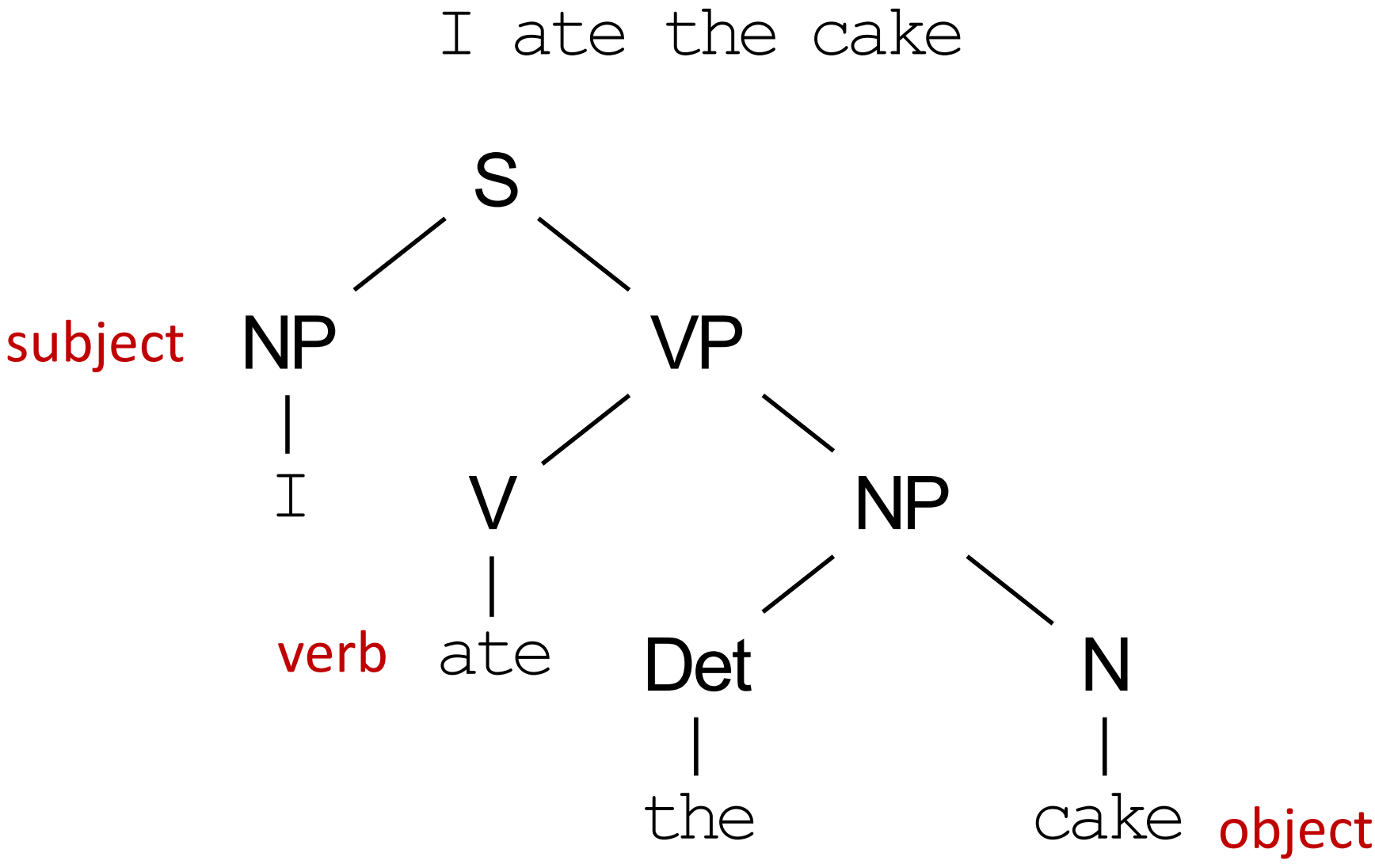
What is a Tree?

A tree is a finite nonempty set of elements.
It is an abstract model of a **hierarchical structure**.
consists of nodes with a **parent-child relation**.
Applications:

- Organization charts
- File systems
- Programming environments

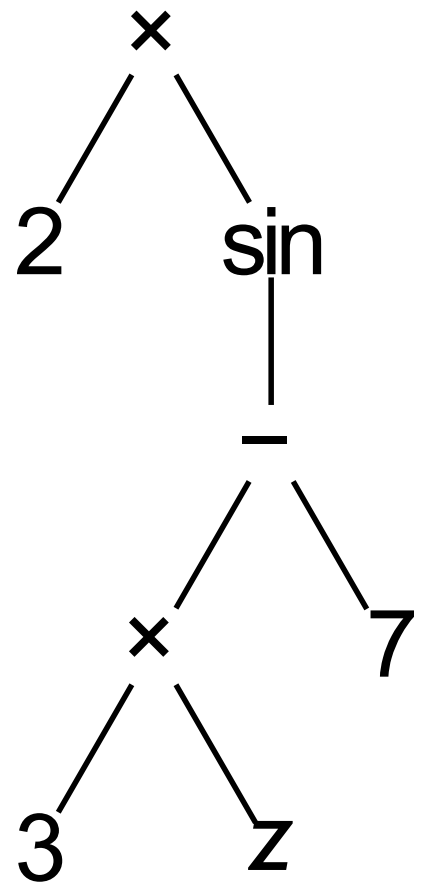


Syntax Tree for a Sentence

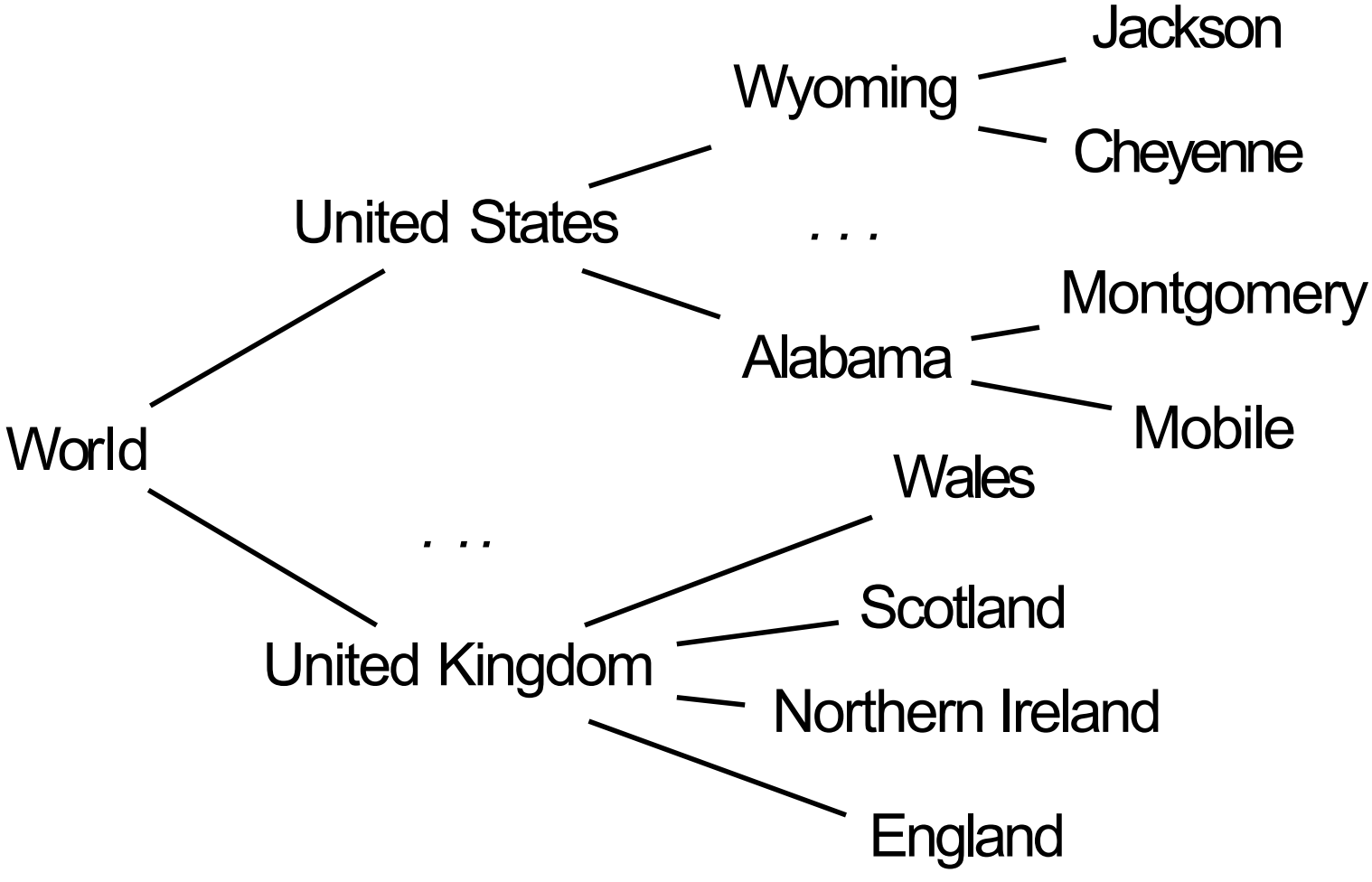


Syntax tree for an Expression

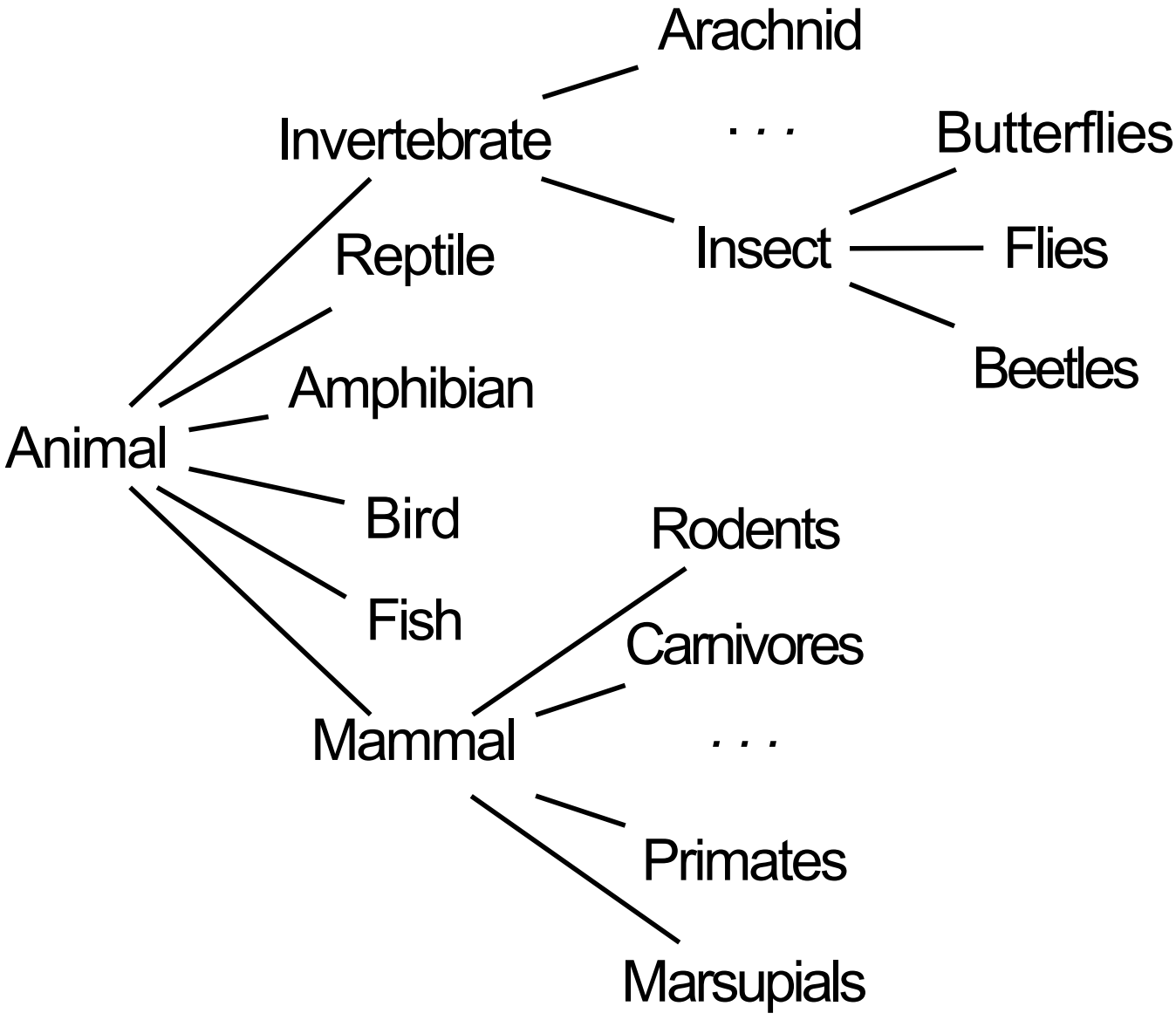
$$2\sin(3z - 7)$$



Geography Hierarchy

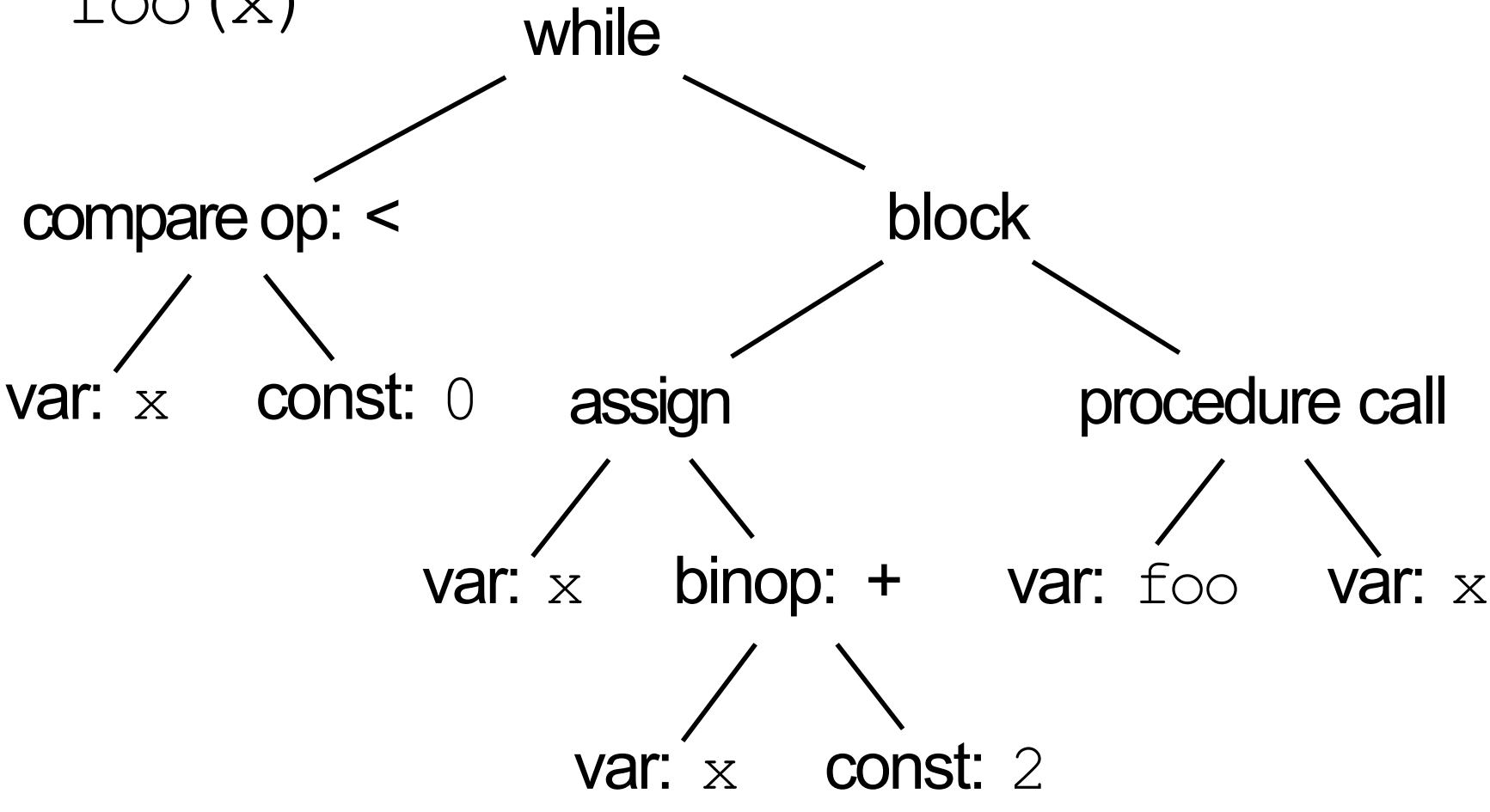


Animal Kingdom



Abstract Syntax Tree for Code

```
while x < 0:
  x = x + 2
  foo(x)
```



Definition – A recursive definition

A Tree is:

- empty, or
- a node with:
  - a key, and
  - a list of child trees.

Simple Tree

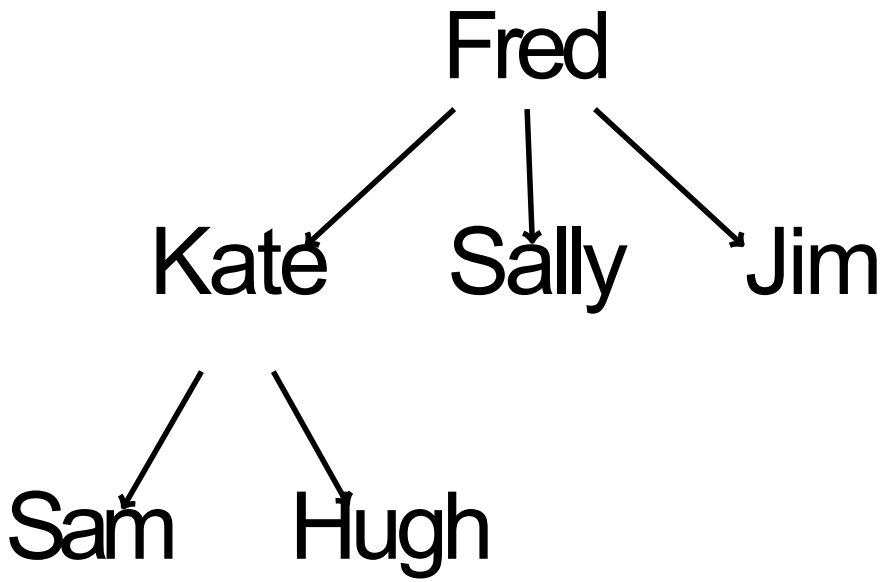
Empty tree:

Tree with one node:

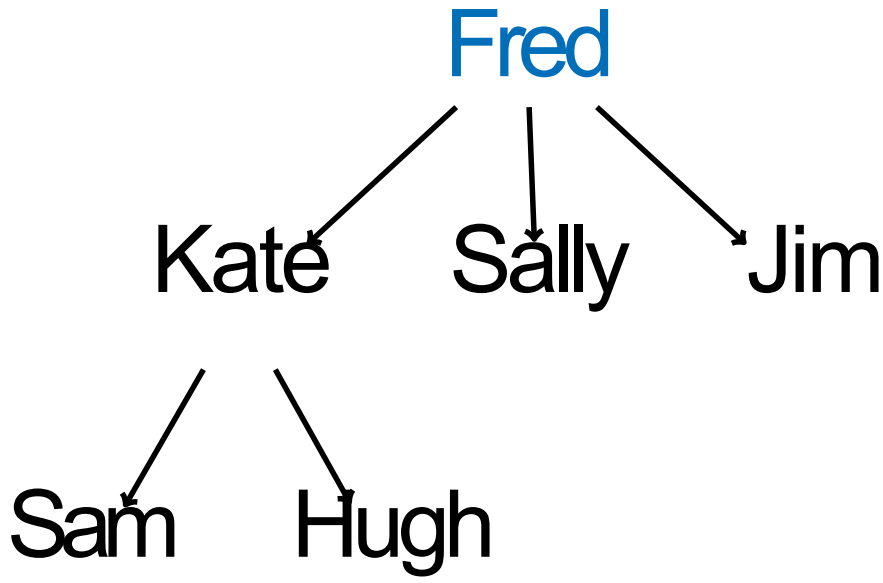
Fred

Tree with two nodes:

Terminology



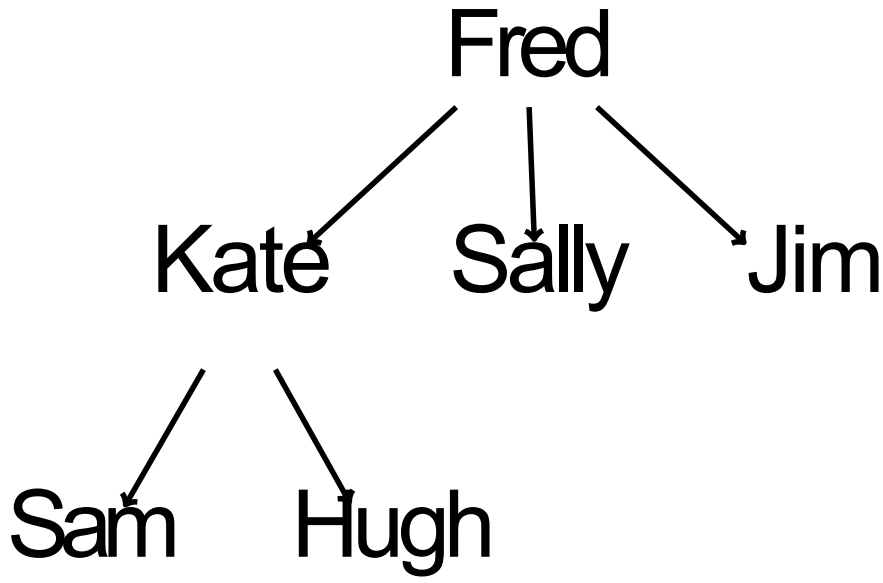
Terminology



***Root*:**

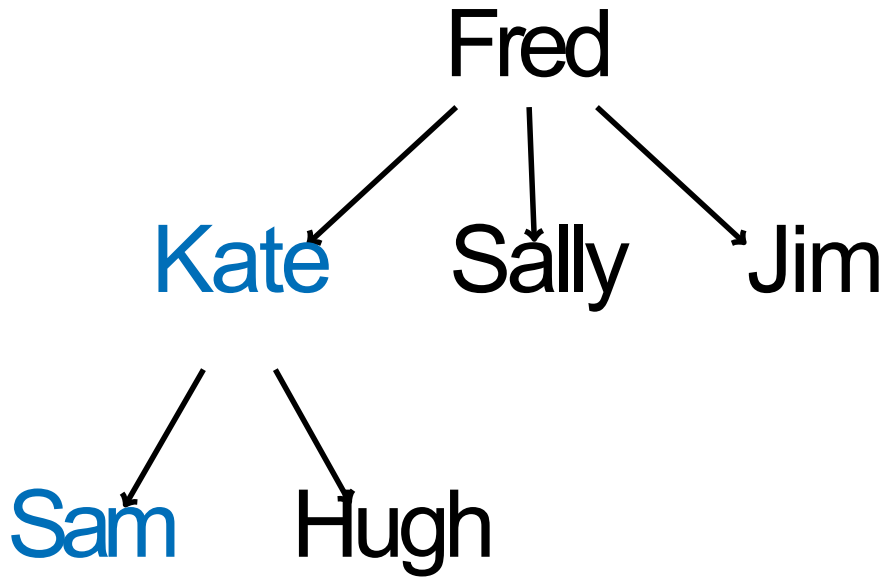
top node in the tree

Terminology



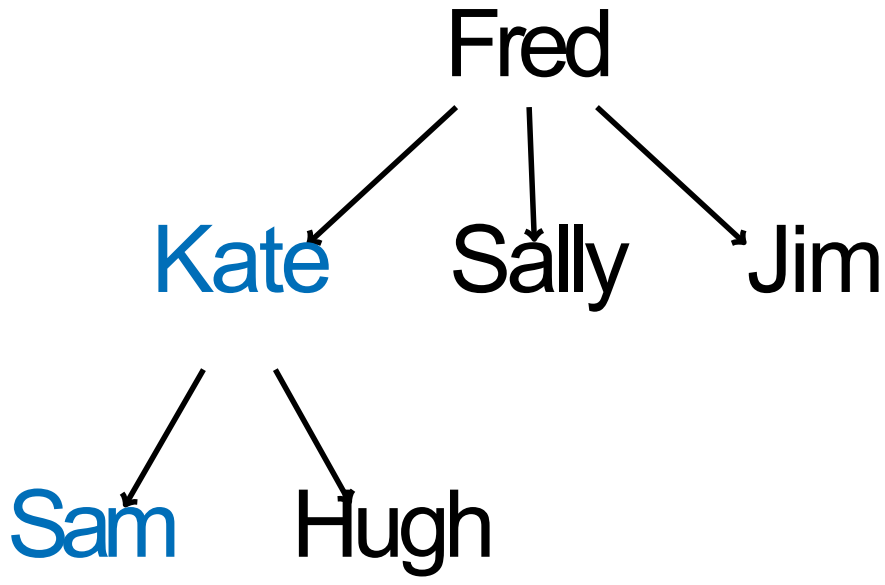
A *child* has a line down directly from a *parent*

Terminology



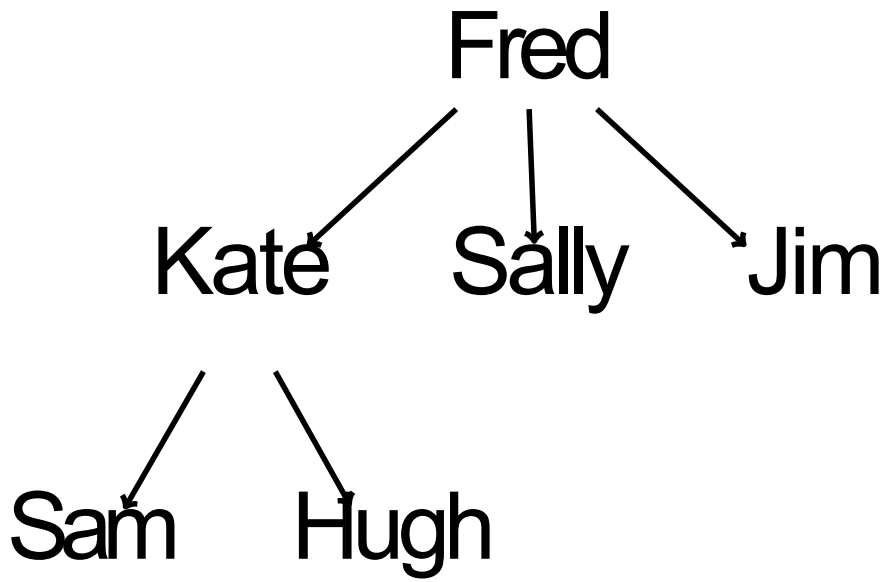
Kate is a *parent* of Sam

Terminology



Sam is a *child* of Kate

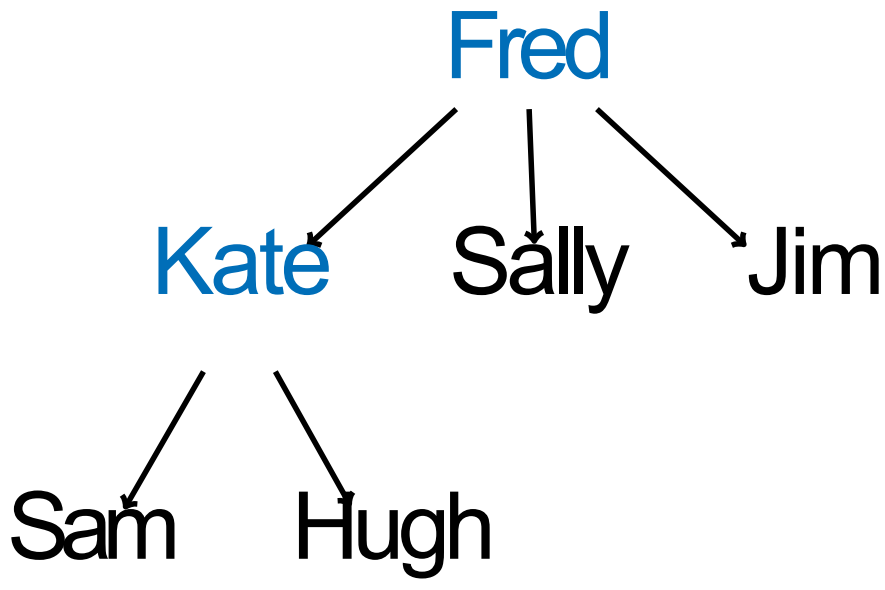
Terminology



***Ancestor*:**

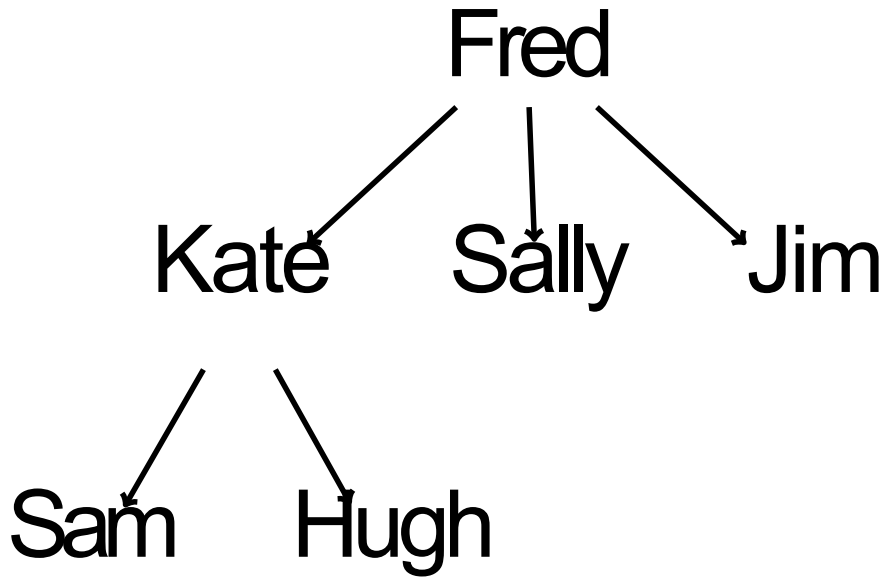
parent, or parent of parent, etc.

Terminology



***Ancestors* of Sam**

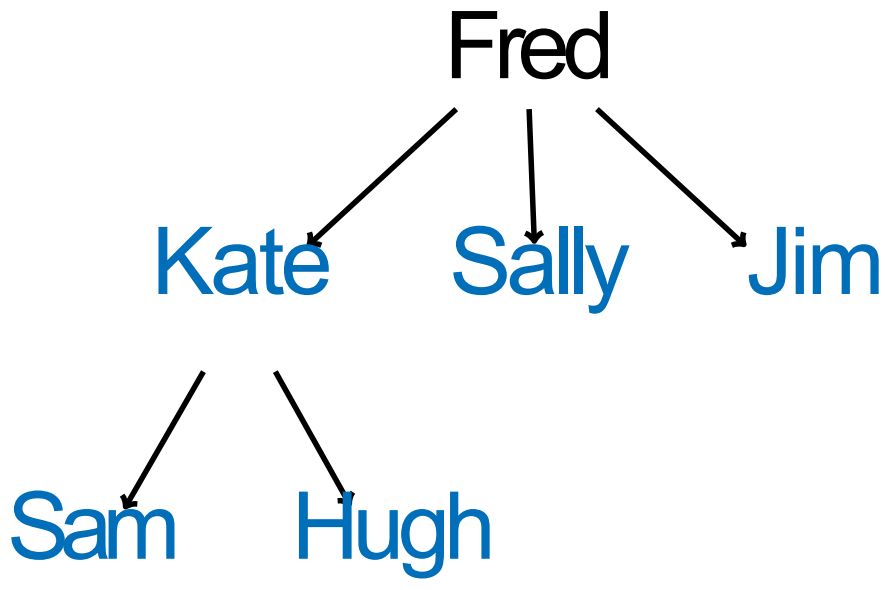
Terminology



***Descendant*:**

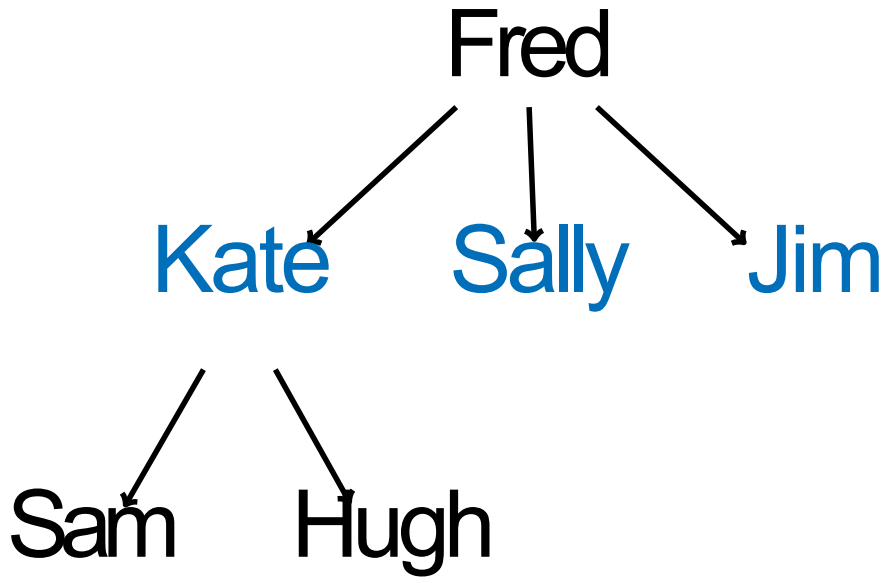
child, or child of child, etc.

Terminology



***Descendants* of Fred**

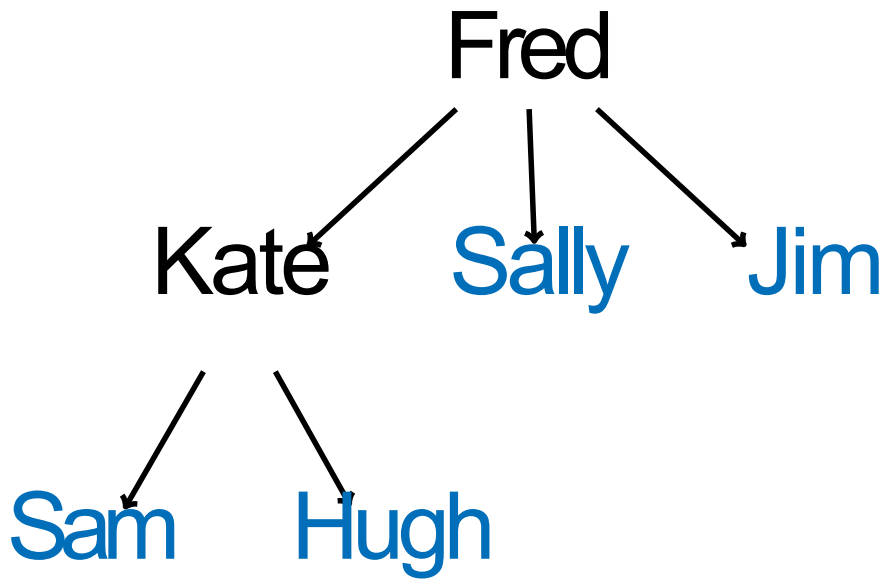
Terminology



Sibling **:**

sharing the same parent

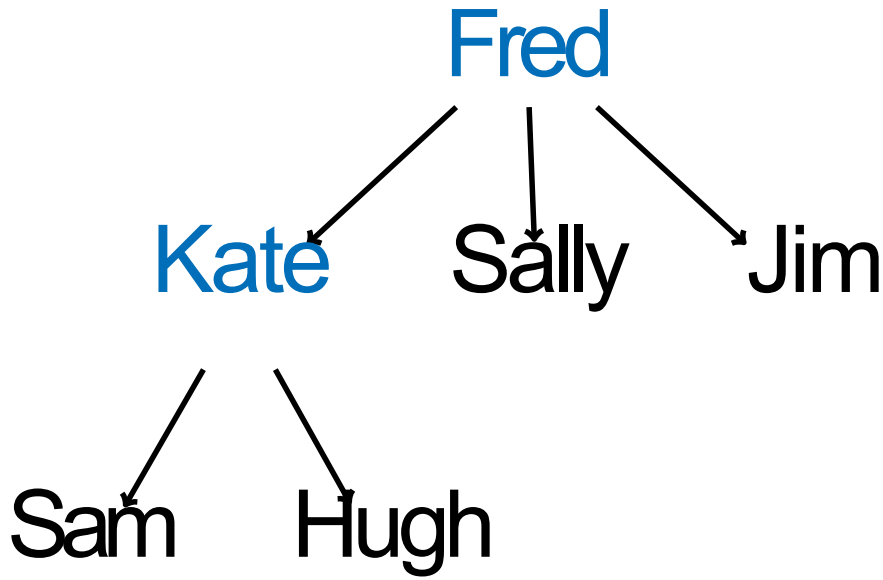
Terminology



Leaf **:**

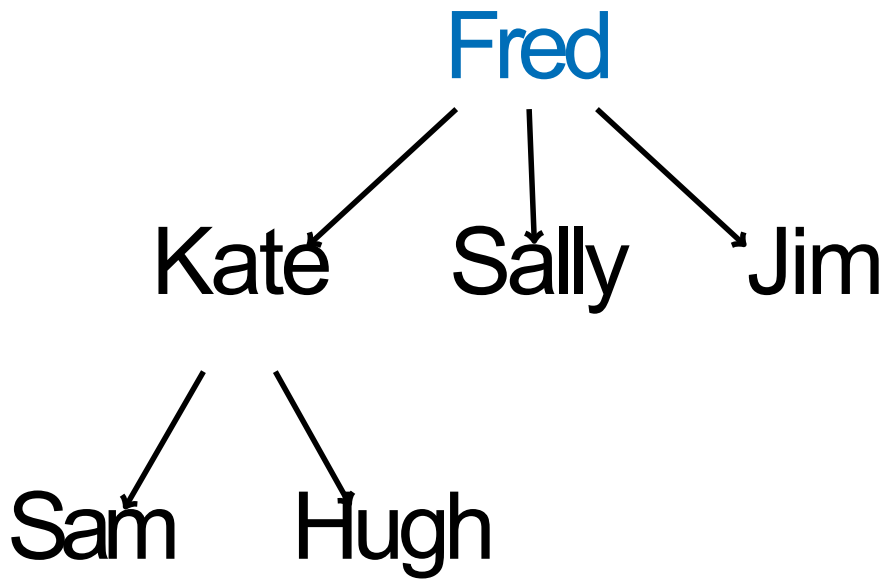
node with no children

Terminology



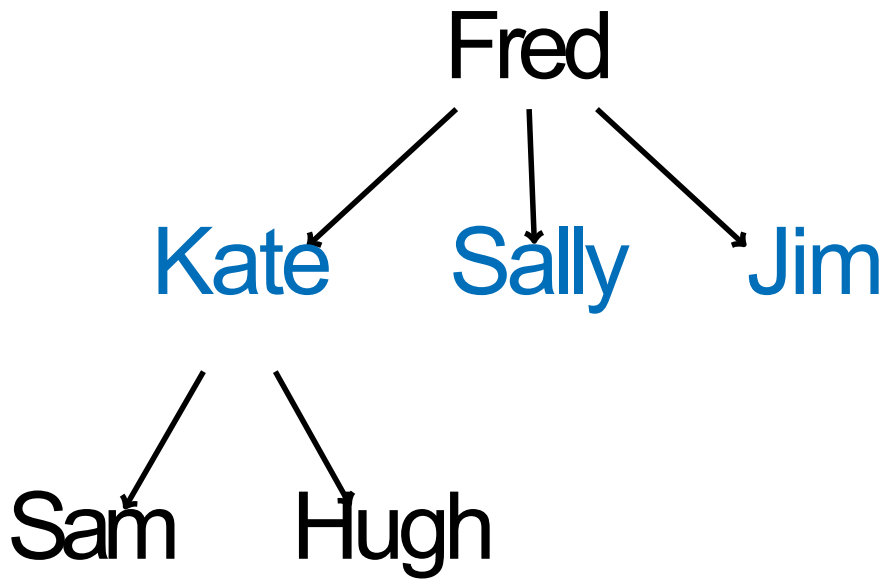
Interior node (non-leaf)

Terminology



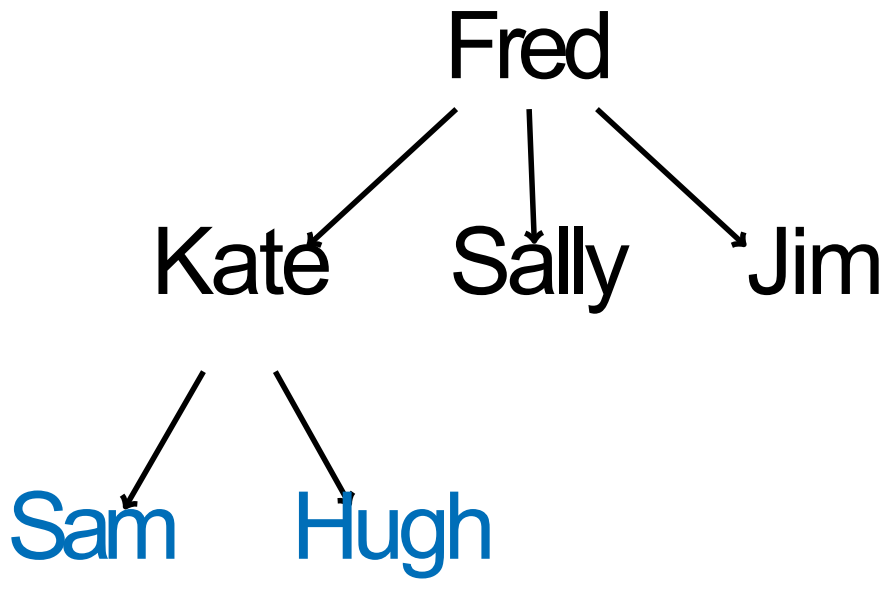
Level 1

Terminology



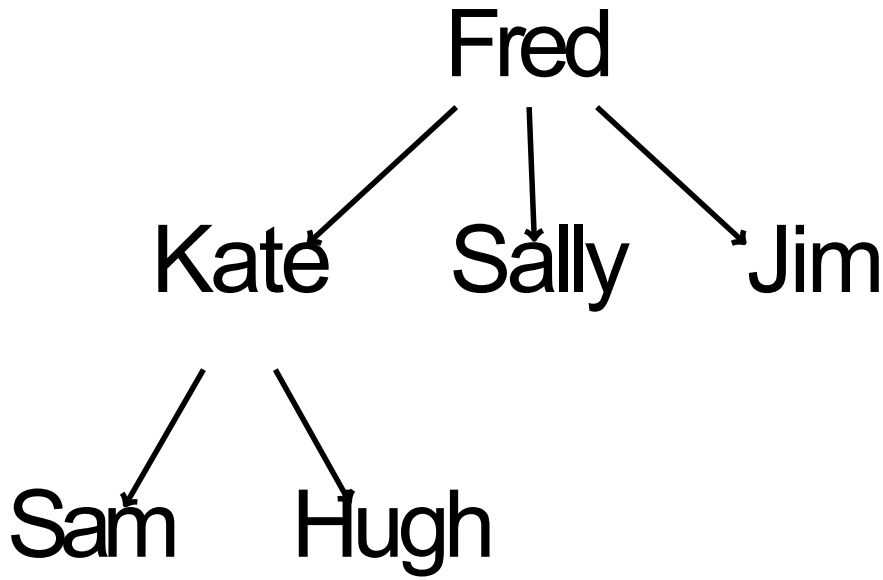
Level 2

Terminology



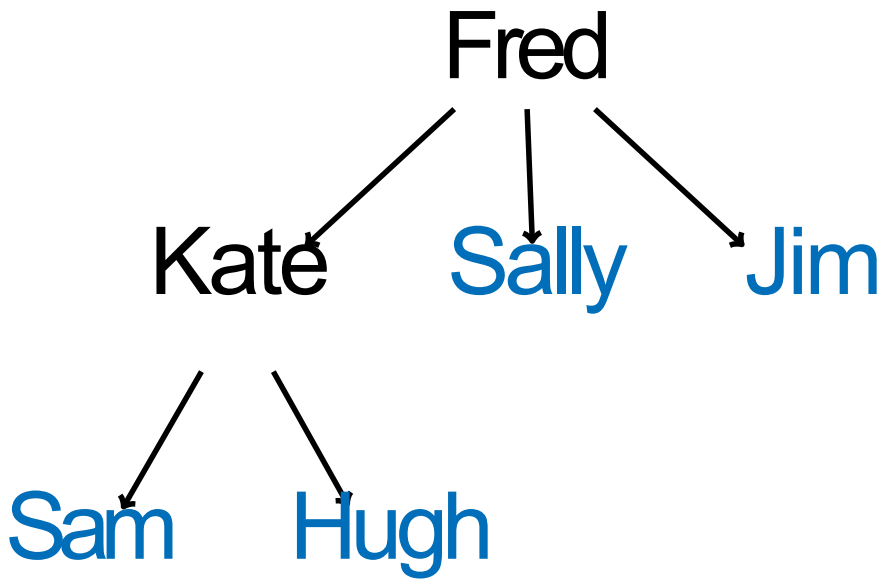
Level 3

Terminology



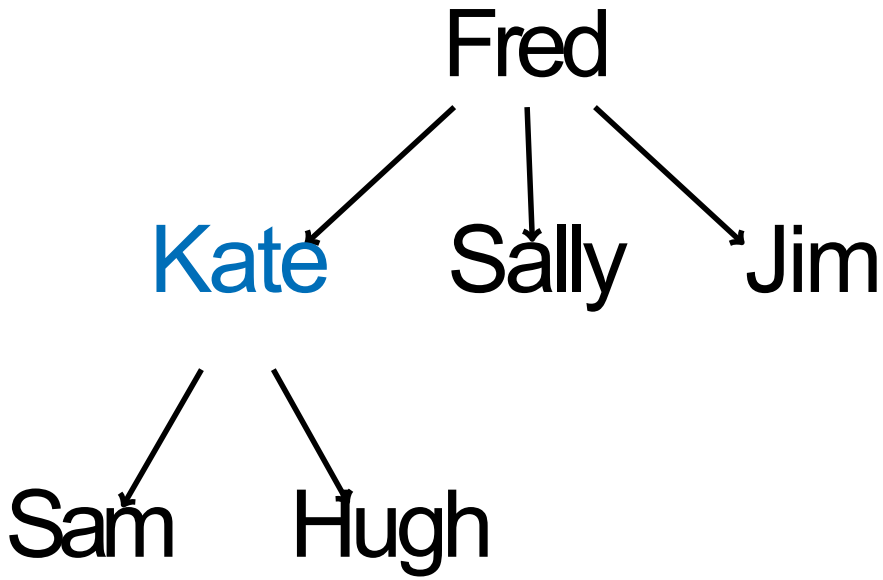
***Height*: maximum depth of subtree node and farthest leaf**

Terminology



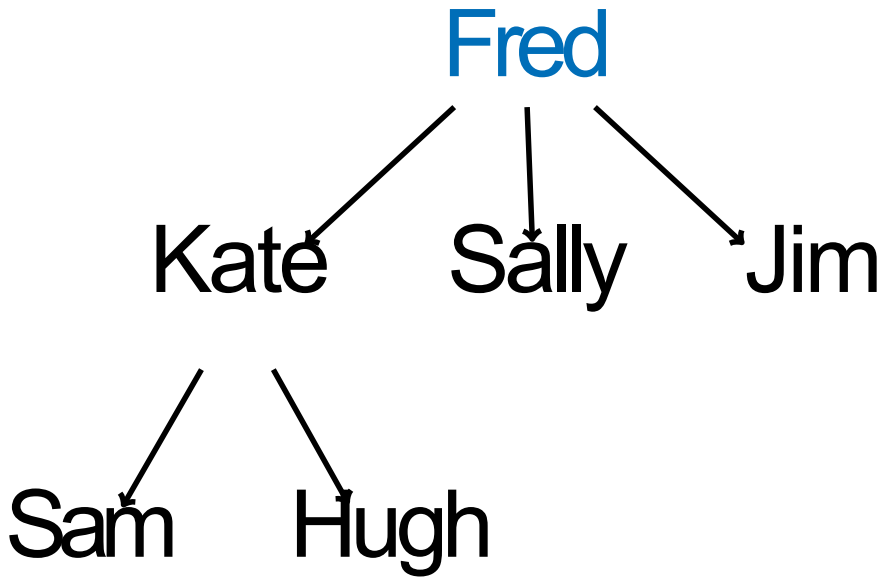
Height 1

Terminology



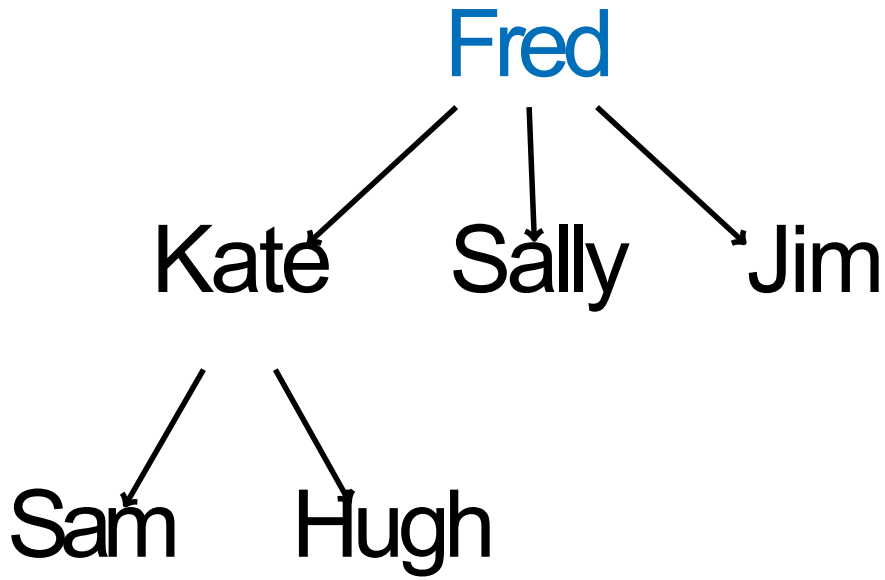
Height 2

Terminology



Height 3

Terminology



Height of the tree is 3

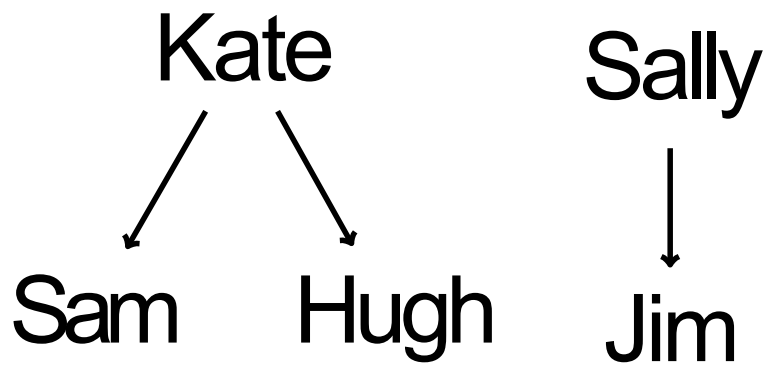
Height(*tree*)

```
if tree = nil:
   return 0
return 1 + Max(Height(tree.left),
                Height(tree.right))
```

Size(*tree*)

```
if tree = nil
   return 0
return 1 + Size(tree.left) +
           Size(tree.right)
```

Terminology

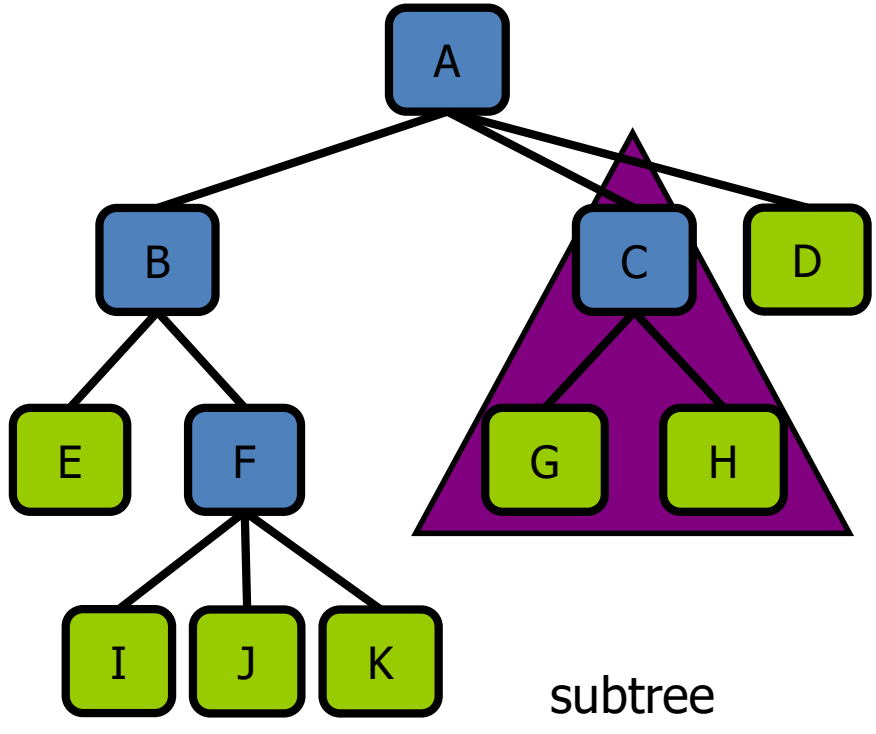


***Forest*:**

collection of trees

Tree Terminologies - Summary

- **Root**: node without parent (A)
- **Siblings**: nodes share the same parent
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)
- **Degree** of a node: the number of its children
- **Subtree**: tree consisting of a node and its descendants



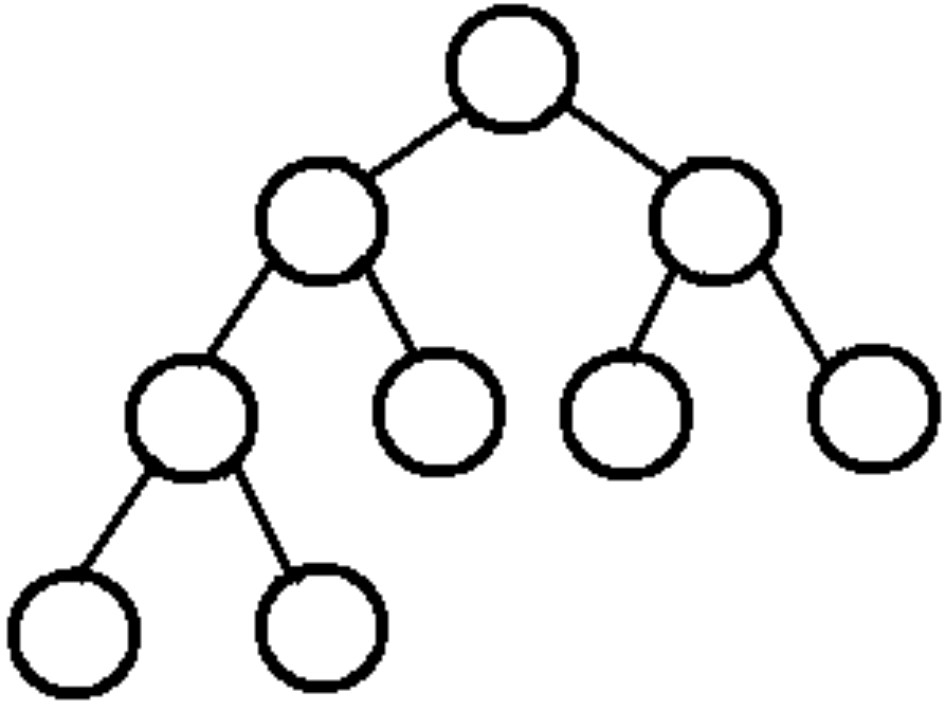
Binary Tree

For binary tree, node contains:

- `key`
- `left`
- `right`
- (optional) `parent`

Complete Binary Tree

A **complete binary tree** is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

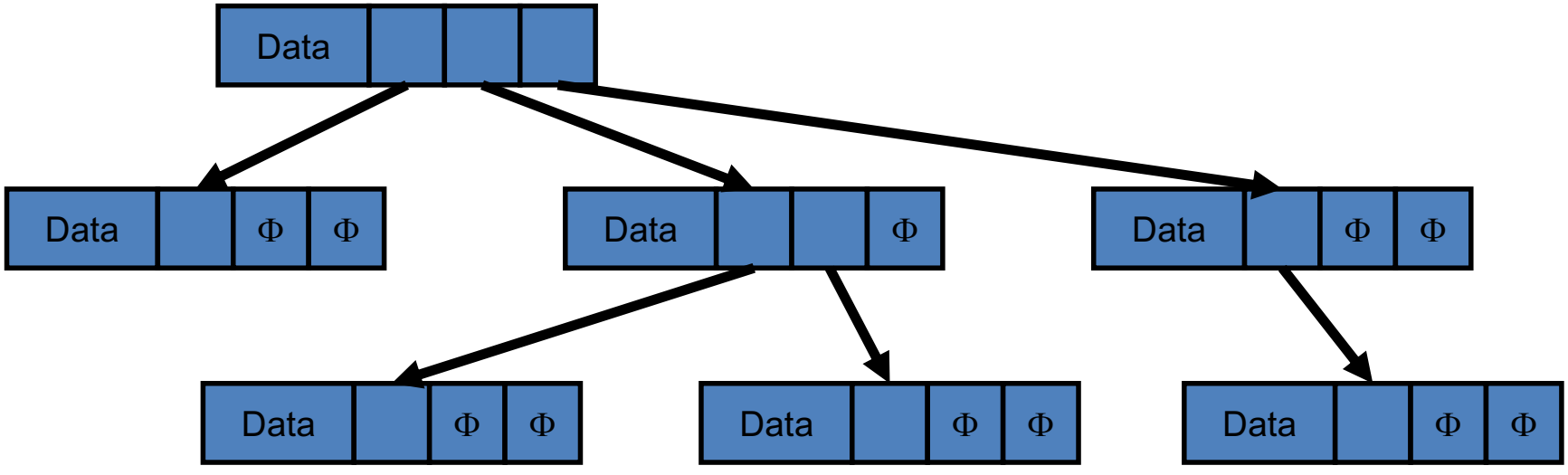


Trees - ADT

Every tree node:

object – useful information or data

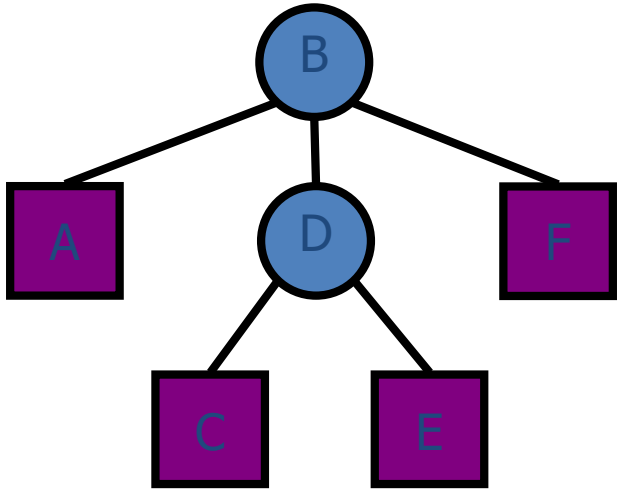
children – pointers to its children

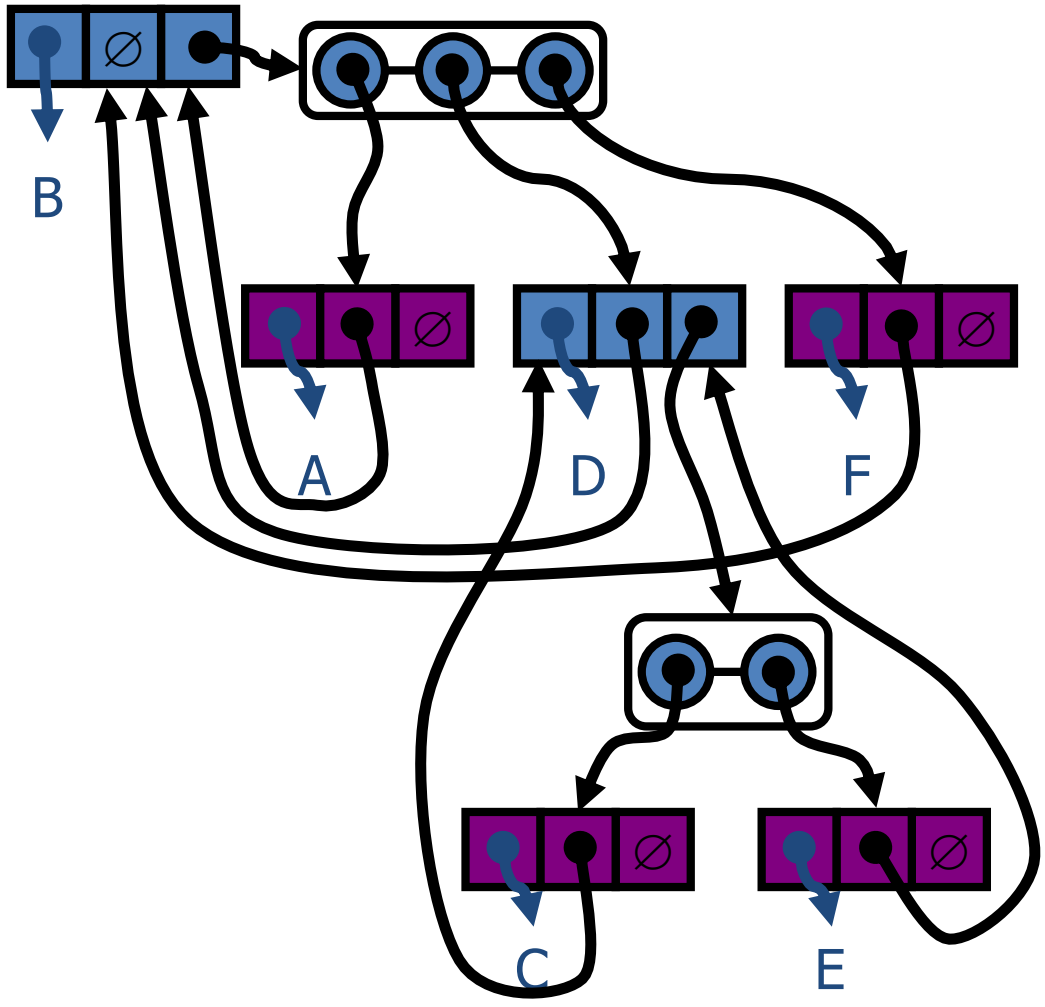


A Tree Representation

A node is represented by an object storing

- Element
- Parent node
- Sequence of children nodes





Assignment

```
struct node
{
    int data;
    struct node *left;
    struct node *right;
};

/* newNode() allocates a new node with the given data and NULL left and
   right pointers. */
struct node* newNode(int data)
{
  // Allocate memory for new node
  struct node* node = (struct node*)malloc(sizeof(struct node));

  // Assign data to this node
  node->data = data;

  // Initialize left and right children as NULL
  node->left = NULL;
  node->right = NULL;
  return(node);
}
```

Assignment

```
int main()
{
  /*create root*/
  struct node *root = newNode(1);
  /* following is the tree after above statement

        1
      /   \
    NULL  NULL
  */


  root->left        = newNode(2);
  root->right       = newNode(3);
  /* 2 and 3 become left and right children of 1
           1
         /   \
        2     3
      /   \   /  \
    NULL NULL NULL NULL
  */


  root->left->left  = newNode(4);
  /* 4 becomes left child of 2
           1
         /       \
        2         3
      /   \      /  \
     4    NULL  NULL  NULL
   /  \
NULL NULL
*/

  getchar();
  return 0;
}
```

Walking a Tree

Often we want to visit the nodes of a tree in a particular order.

For example, print the nodes of the tree.

Walking a Tree

Often we want to visit the nodes of a tree in a particular order.

For example, print the nodes of the tree.

- Depth-first: We completely traverse one sub-tree before exploring a sibling sub-tree.

Walking a Tree

Often we want to visit the nodes of a tree in a particular order.

For example, print the nodes of the tree.

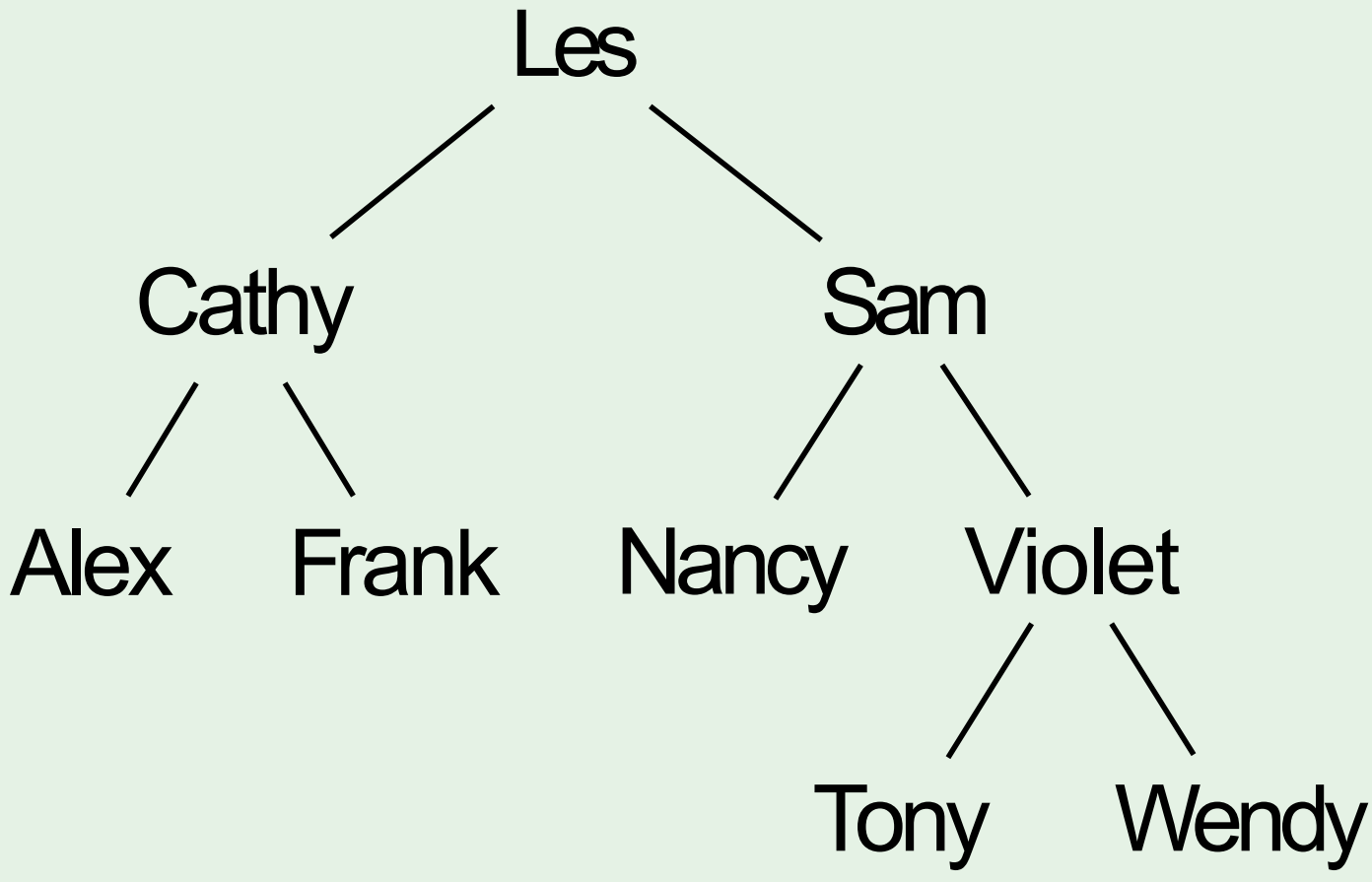
- Depth-first: We completely traverse one sub-tree before exploring a sibling sub-tree.
- Breadth-first: We traverse all nodes at one level before progressing to the next level.

Depth-first

InOrderTraversal(*tree*)

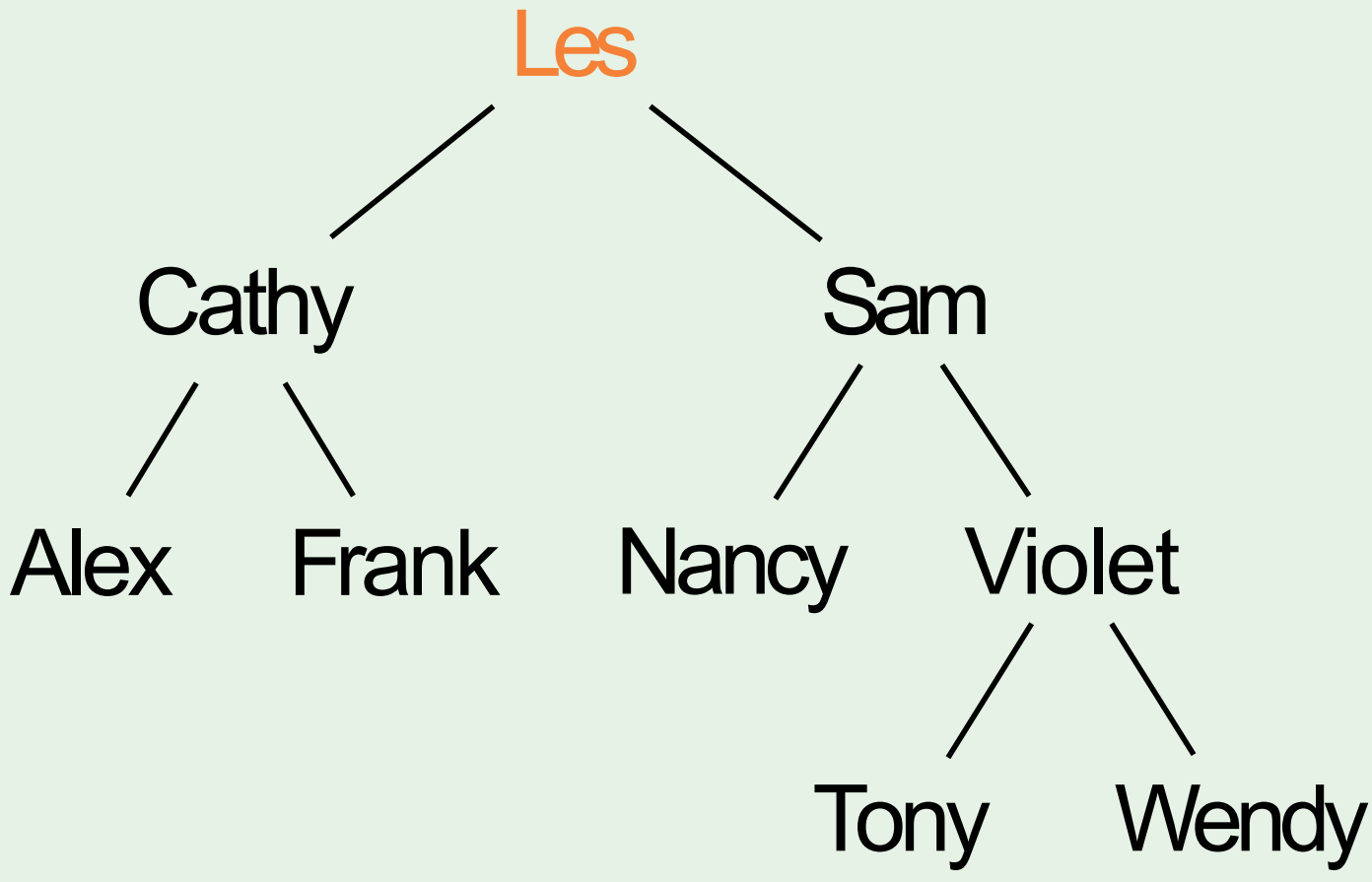
```
if tree = nil:
   return
InOrderTraversal(tree.left)
Print(tree.key)
InOrderTraversal(tree.right)
```

InOrderTraversal



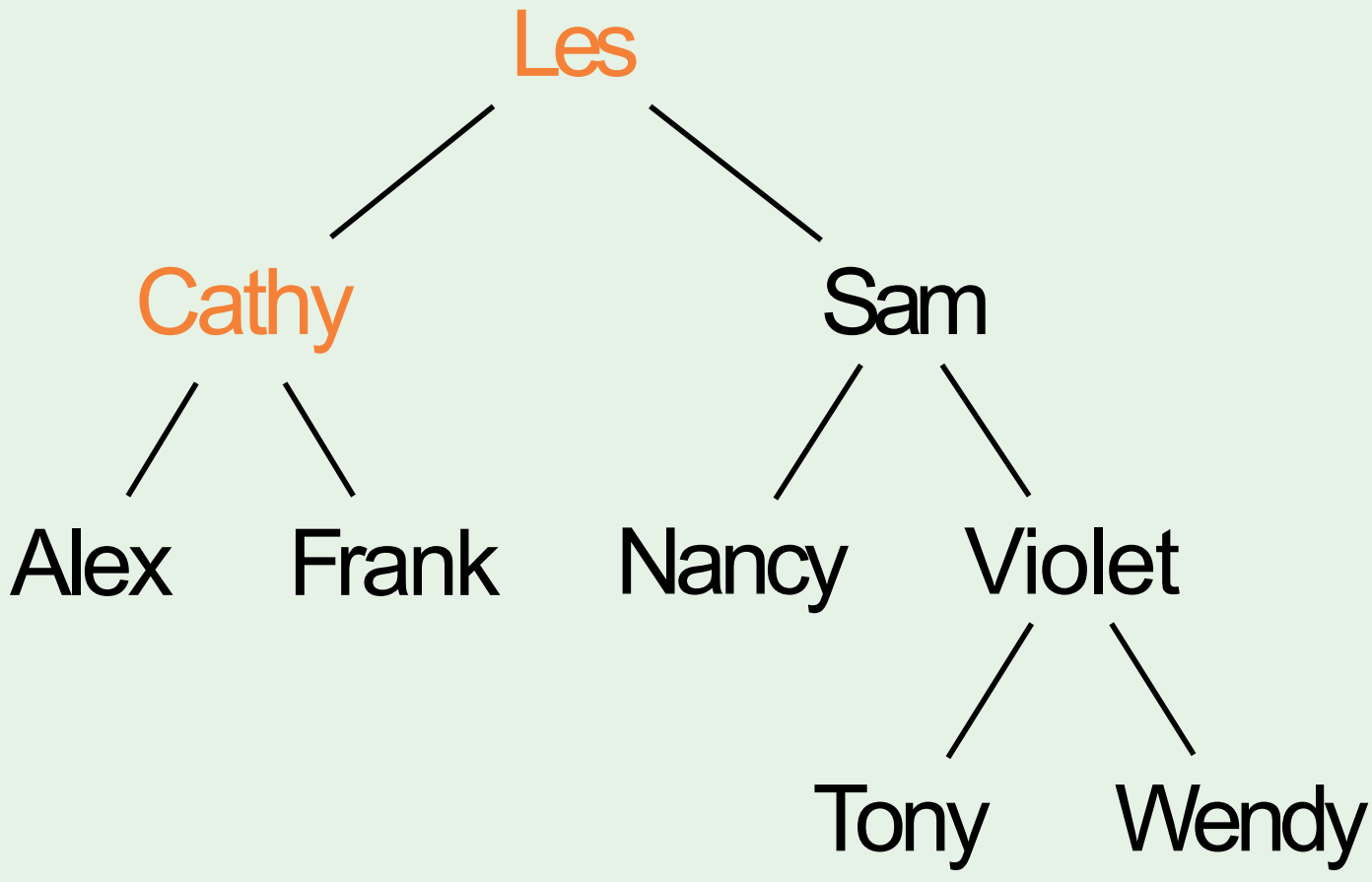
Output:

InOrderTraversal



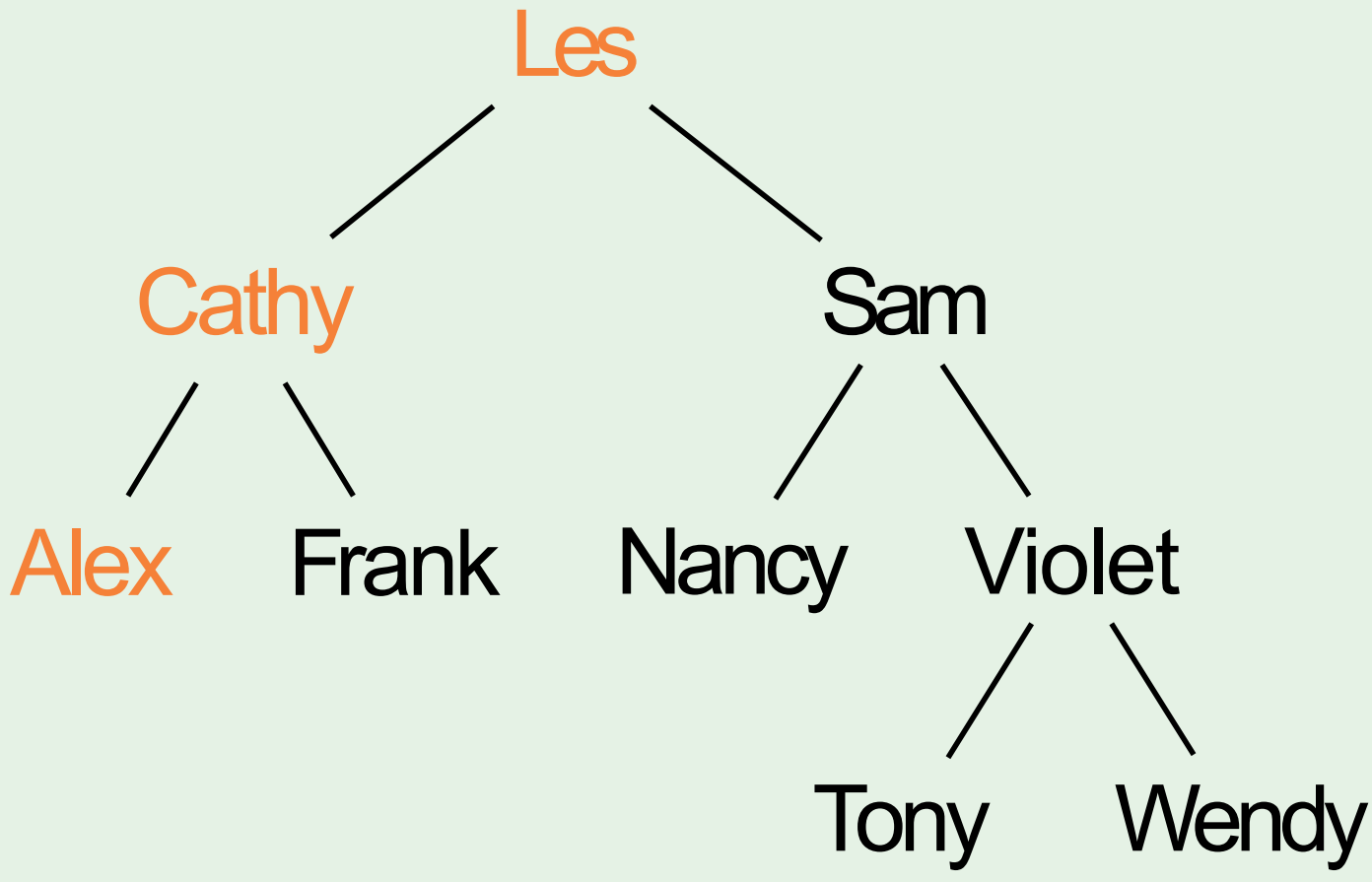
Output:

InOrderTraversal



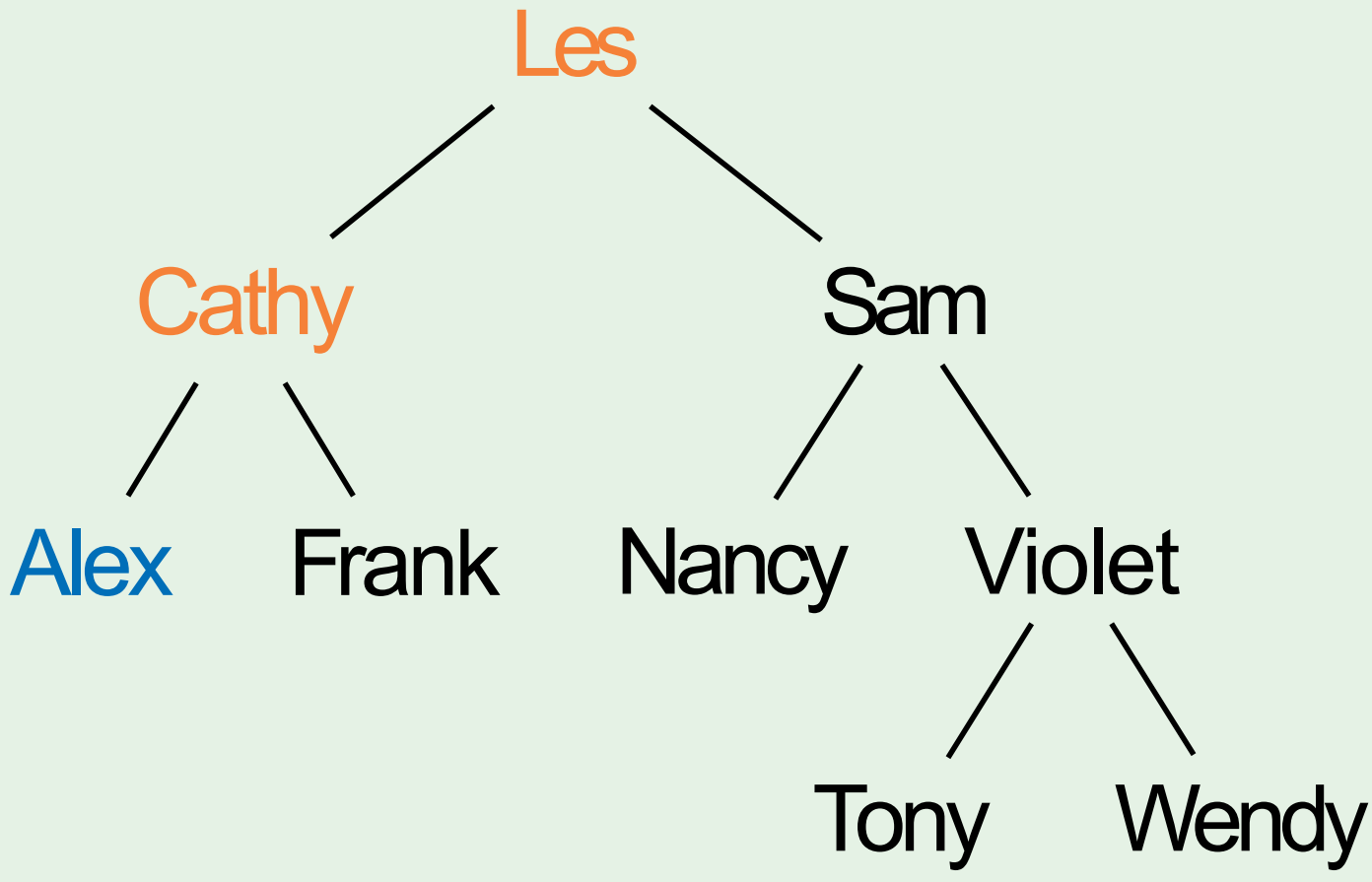
Output:

InOrderTraversal



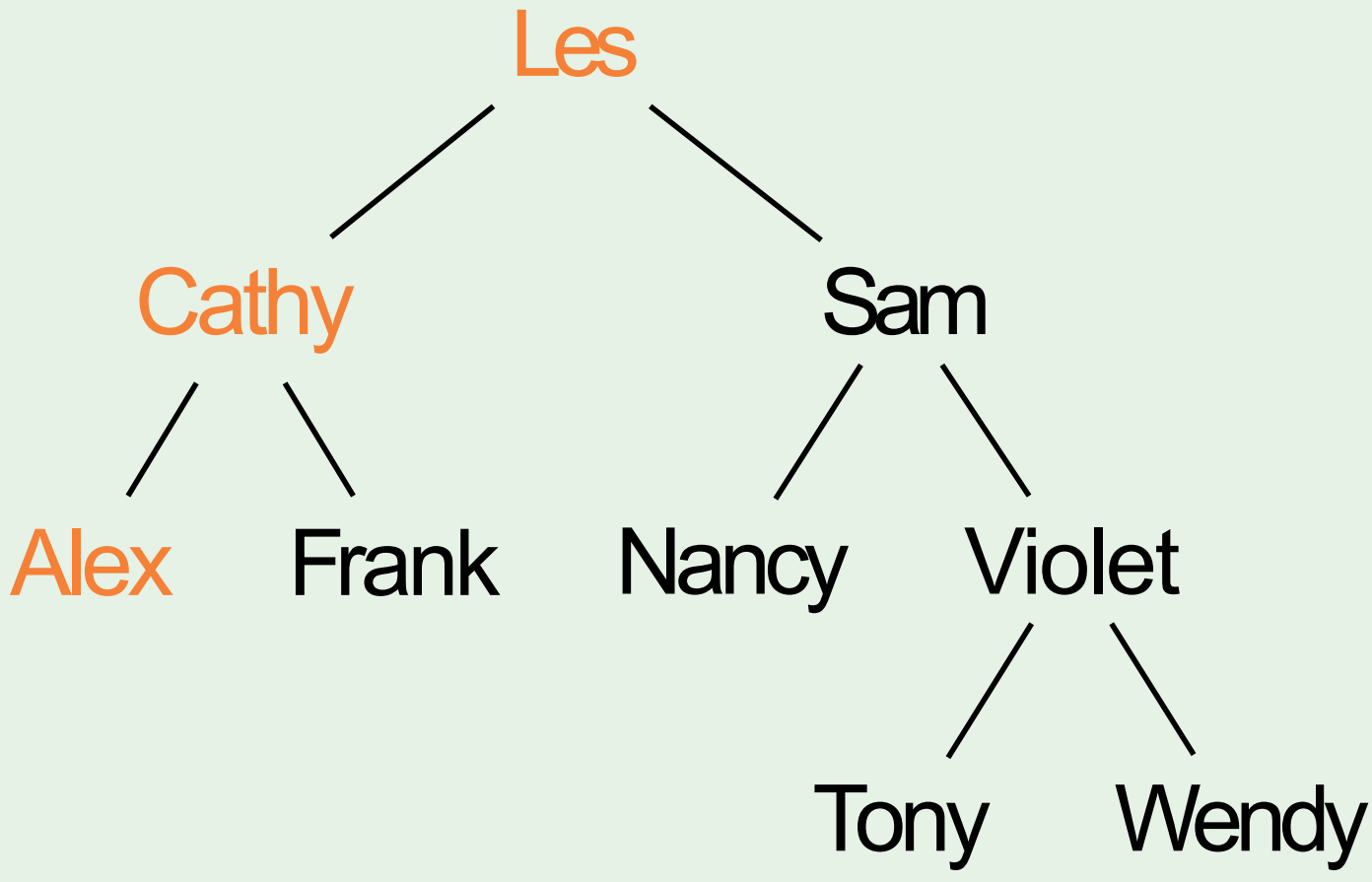
Output:

InOrderTraversal



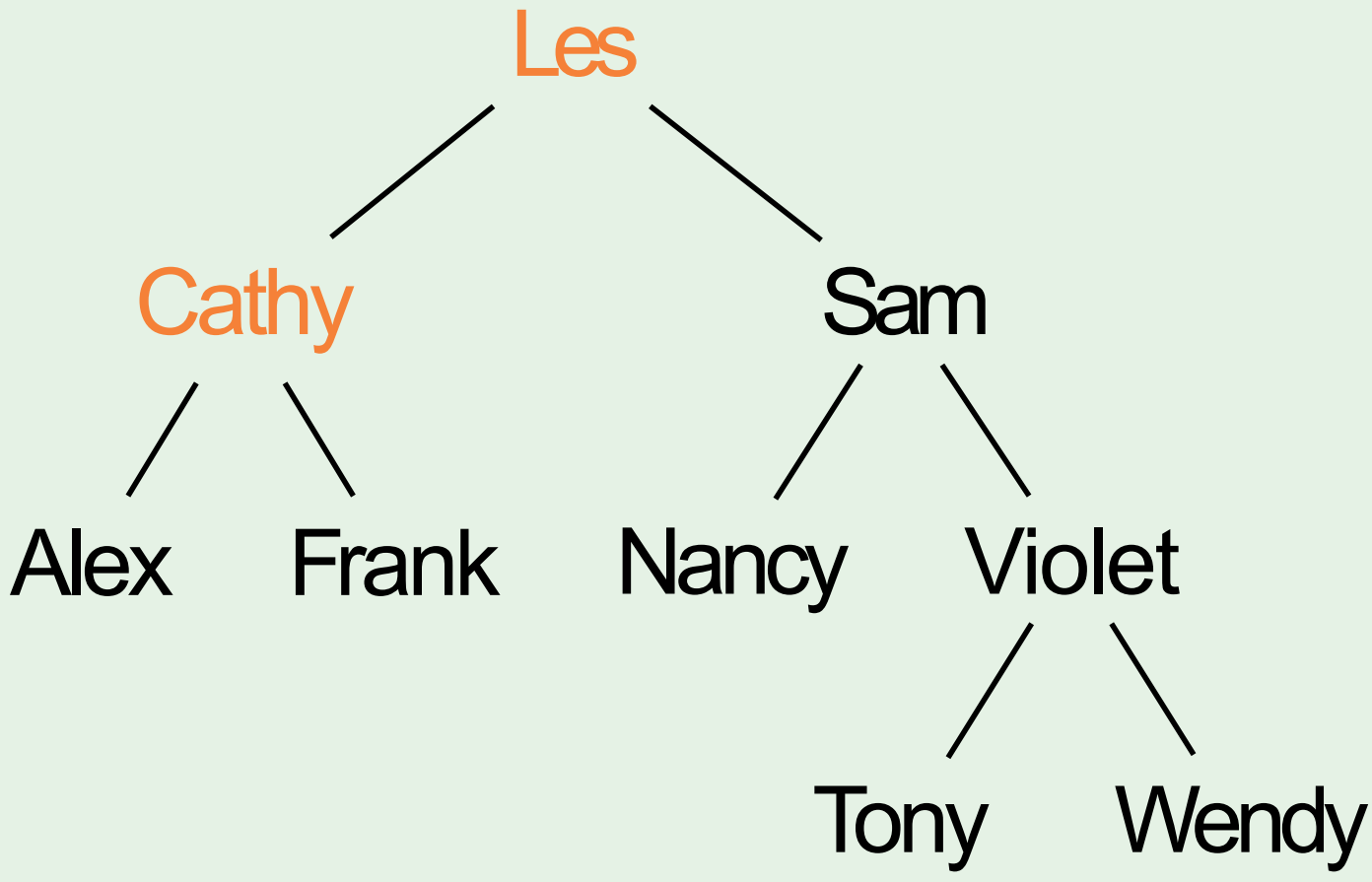
Output: Alex

InOrderTraversal



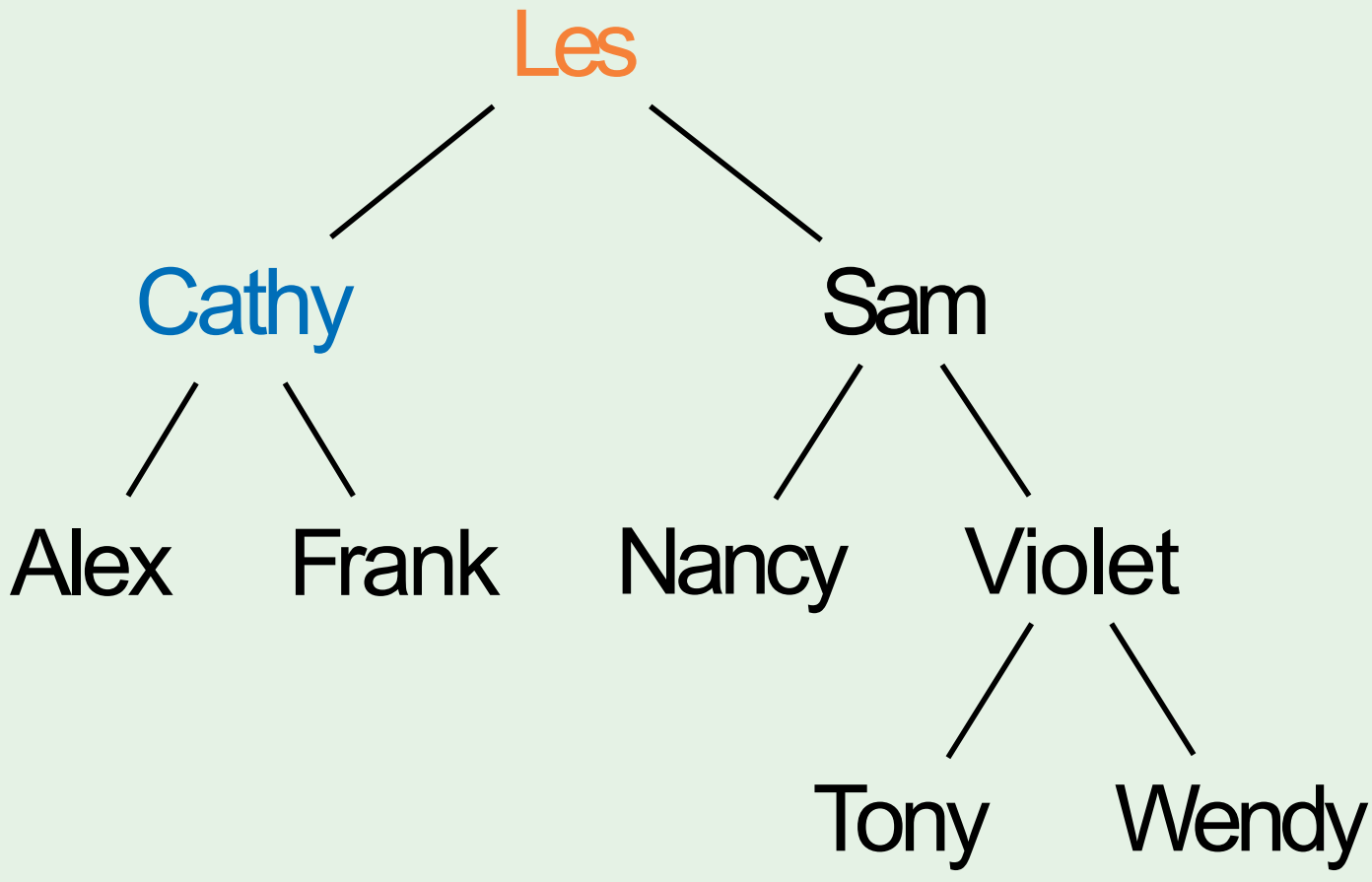
Output: Alex

InOrderTraversal



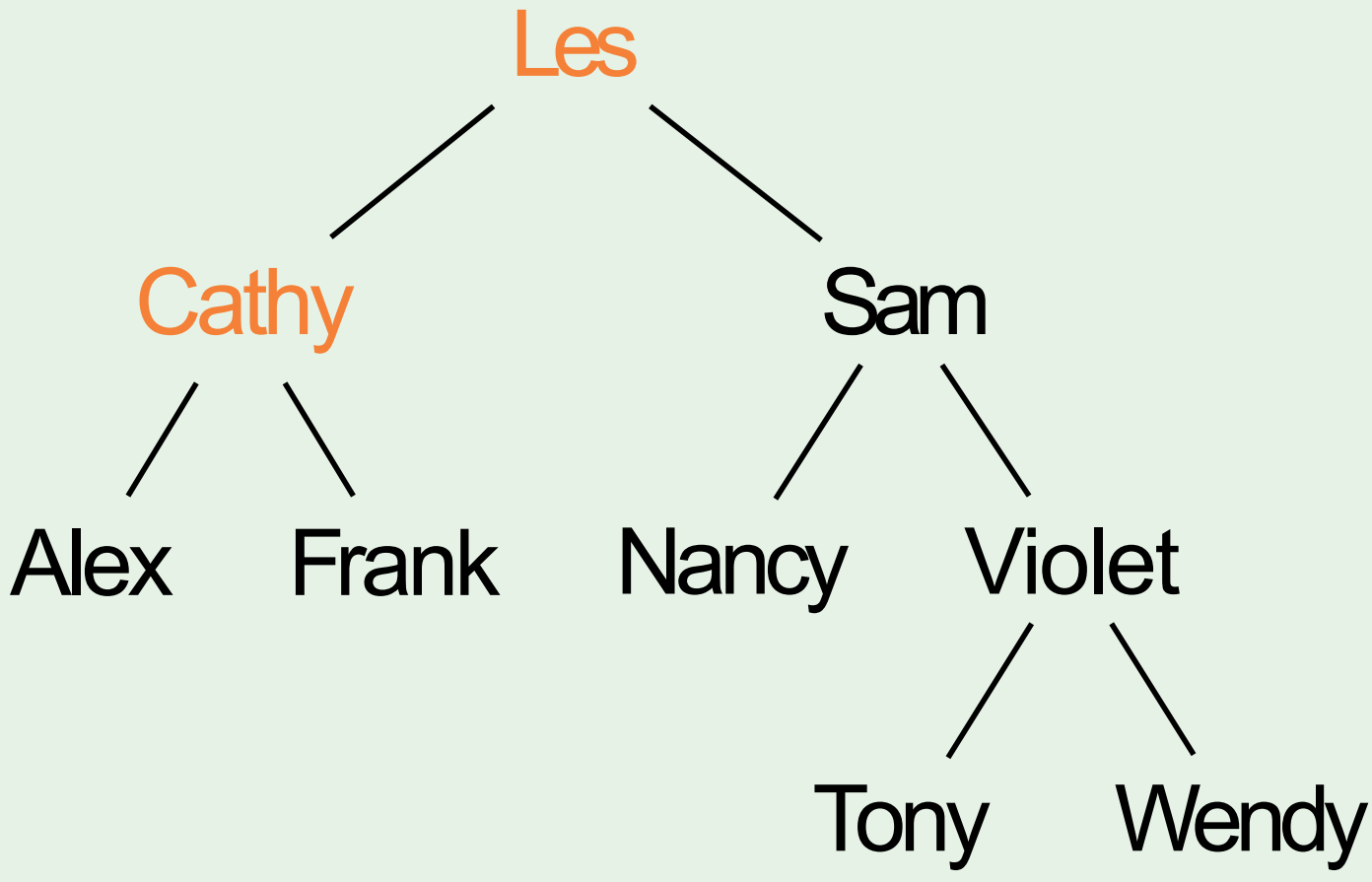
Output: Alex

InOrderTraversal



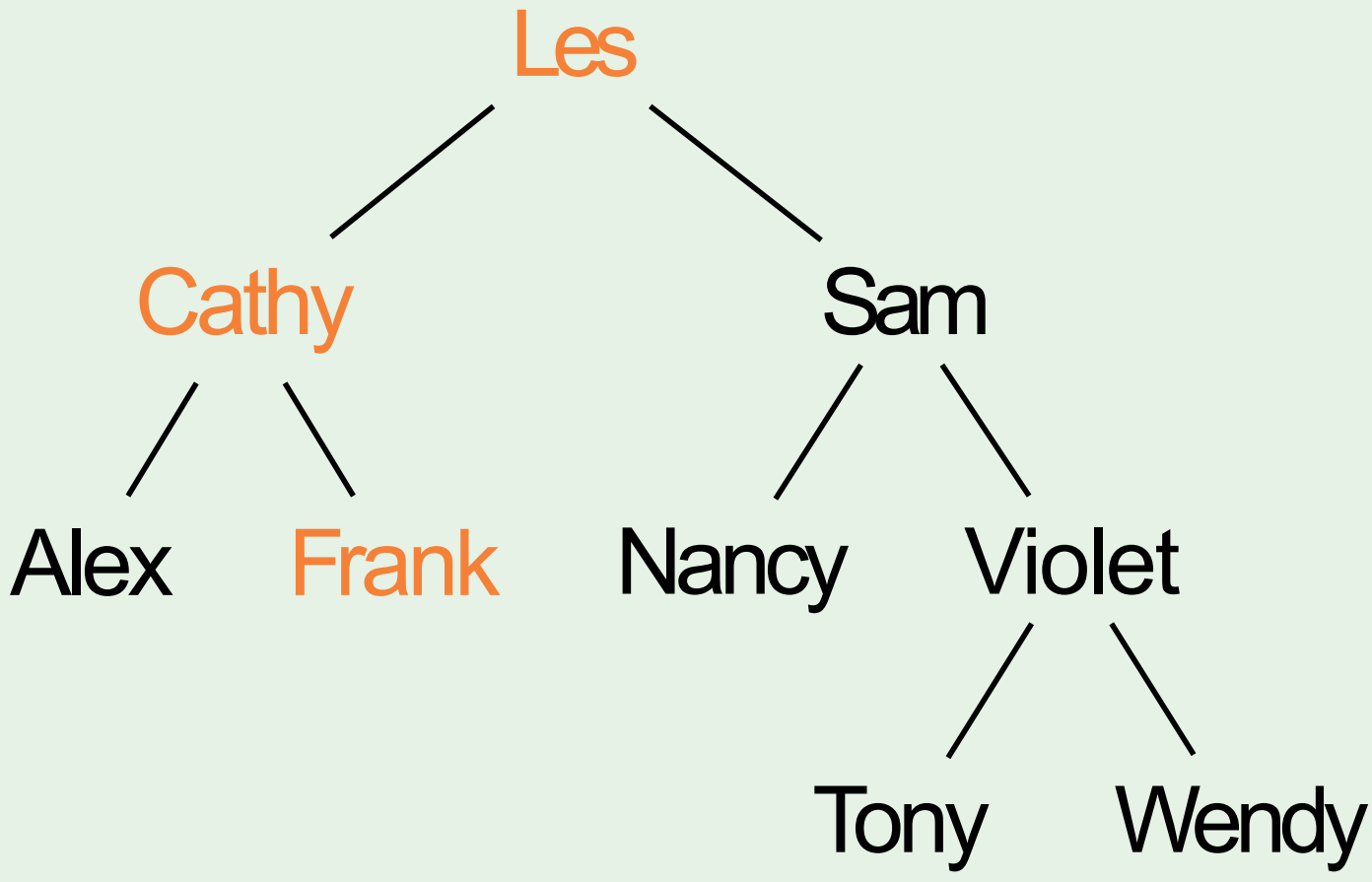
Output: Alex Cathy

InOrderTraversal



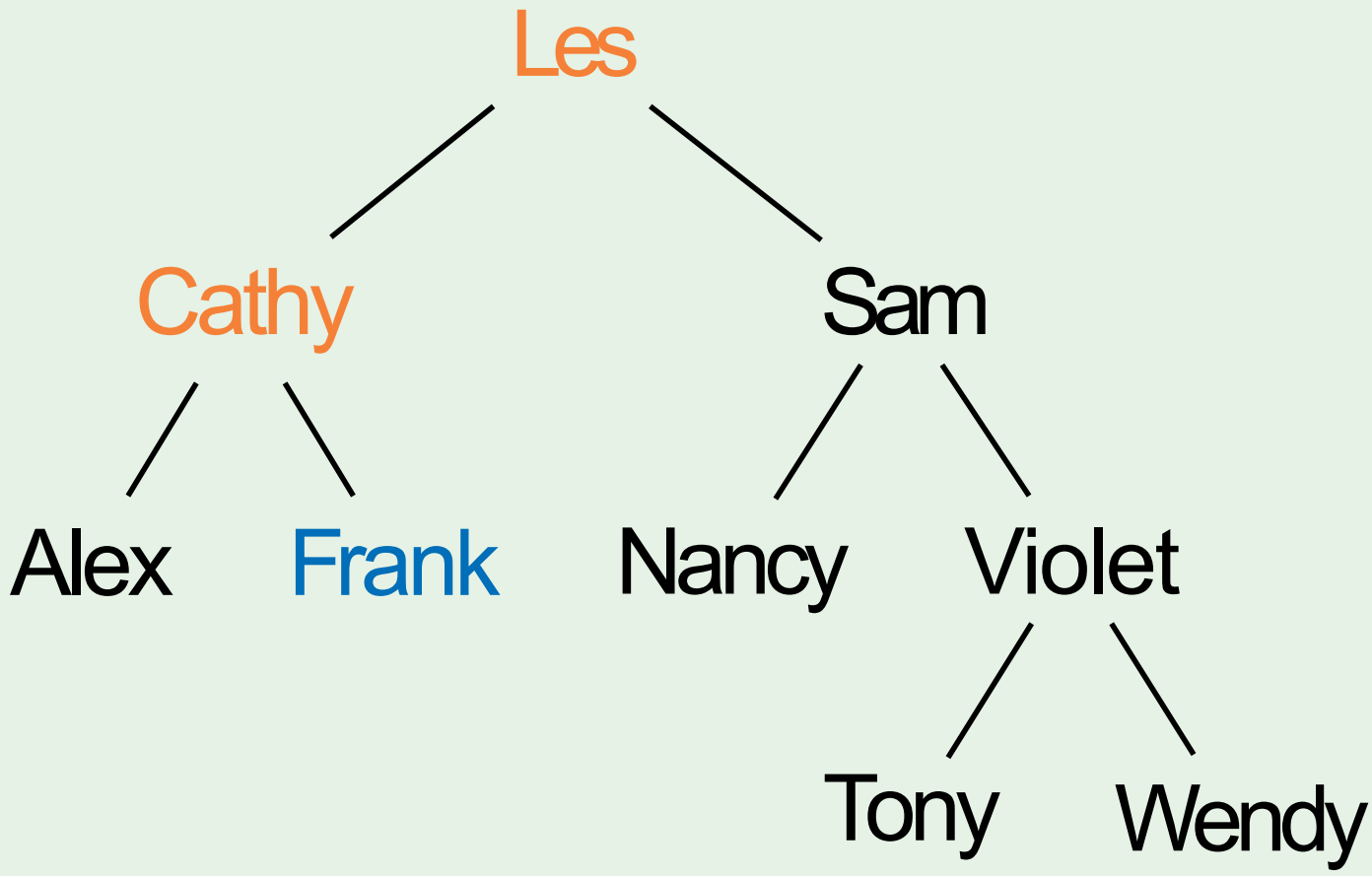
Output: Alex Cathy

InOrderTraversal



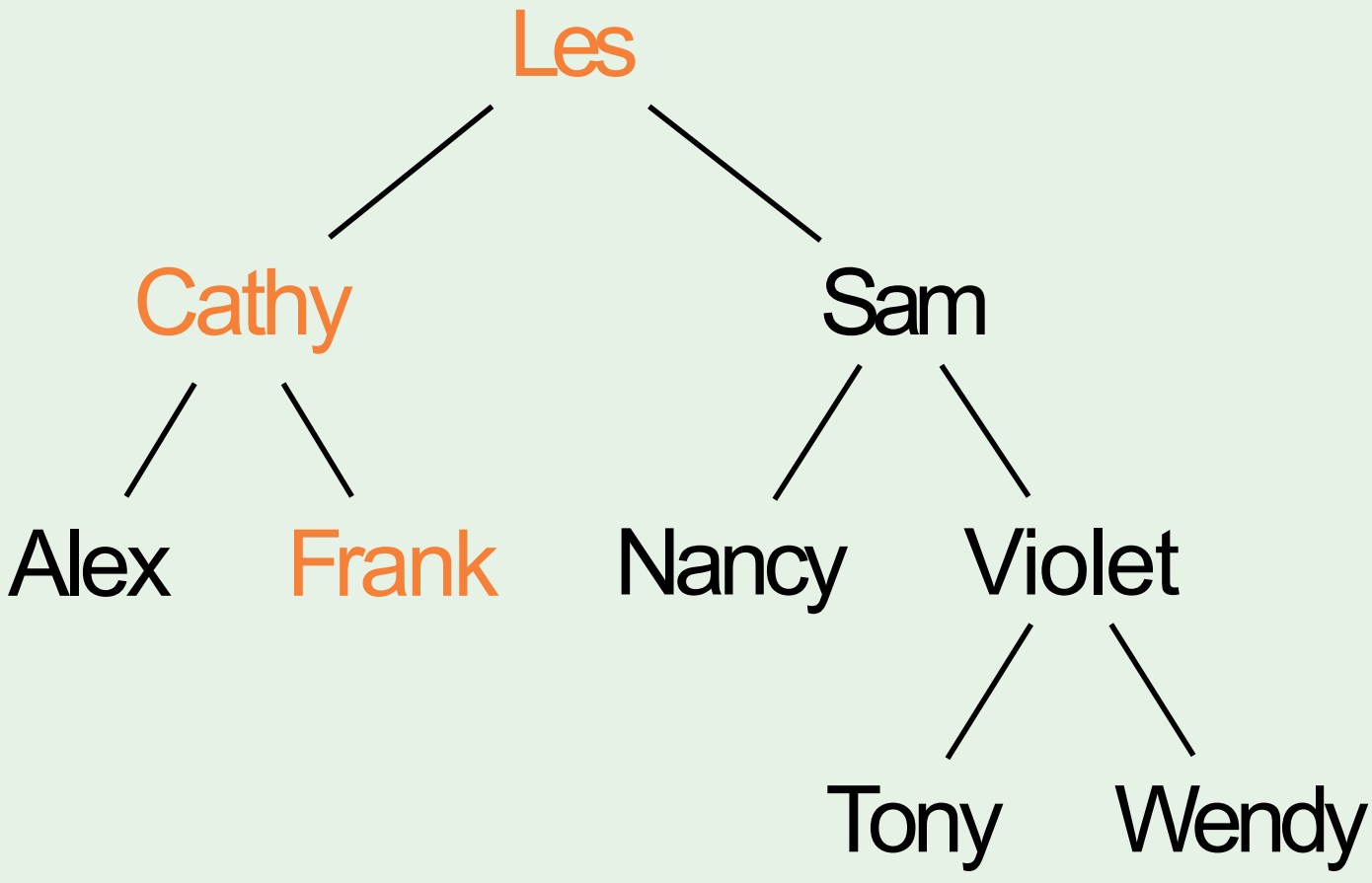
Output: AlexCathy

InOrderTraversal



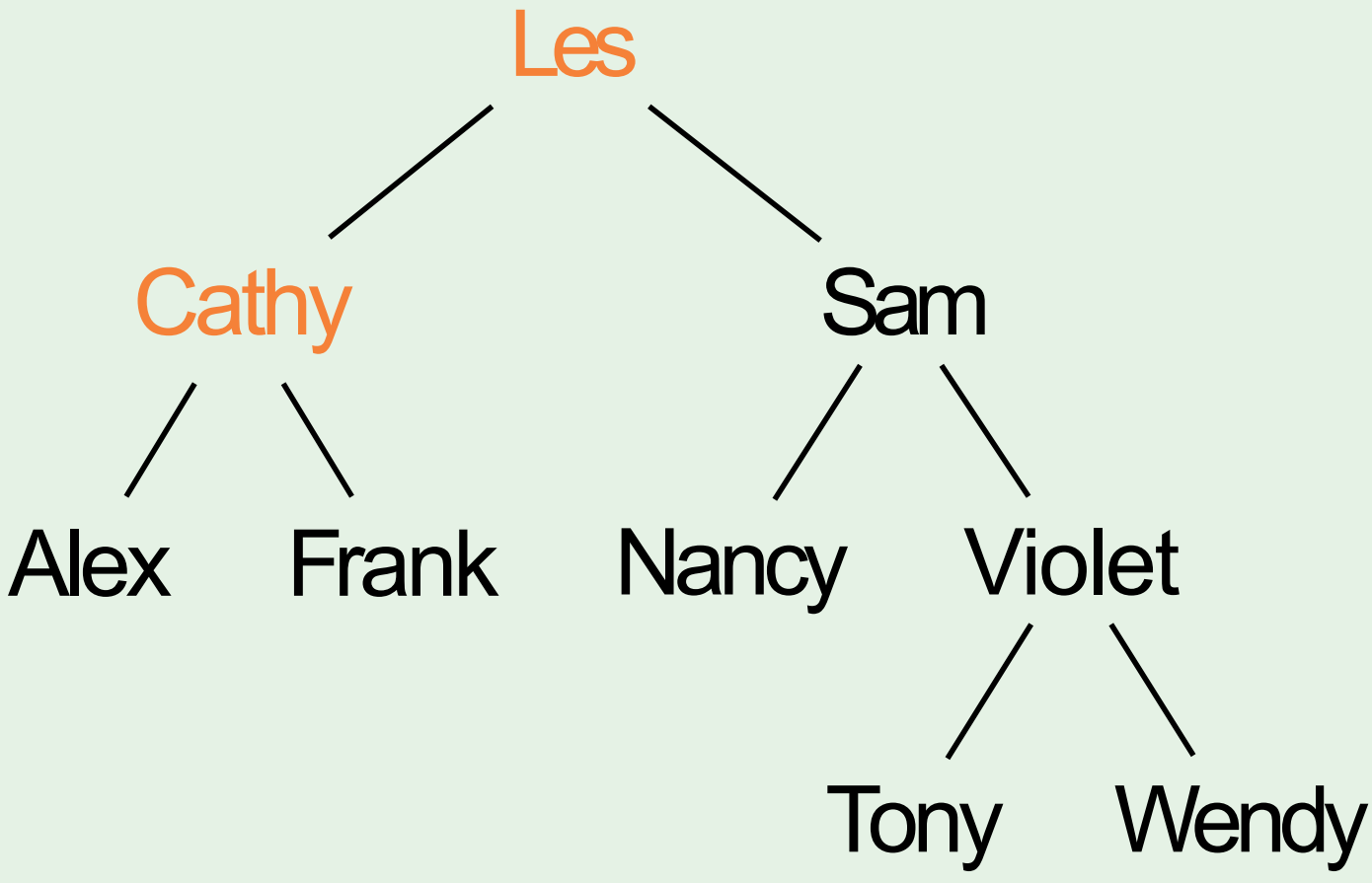
Output: Alex Cathy Frank

InOrderTraversal



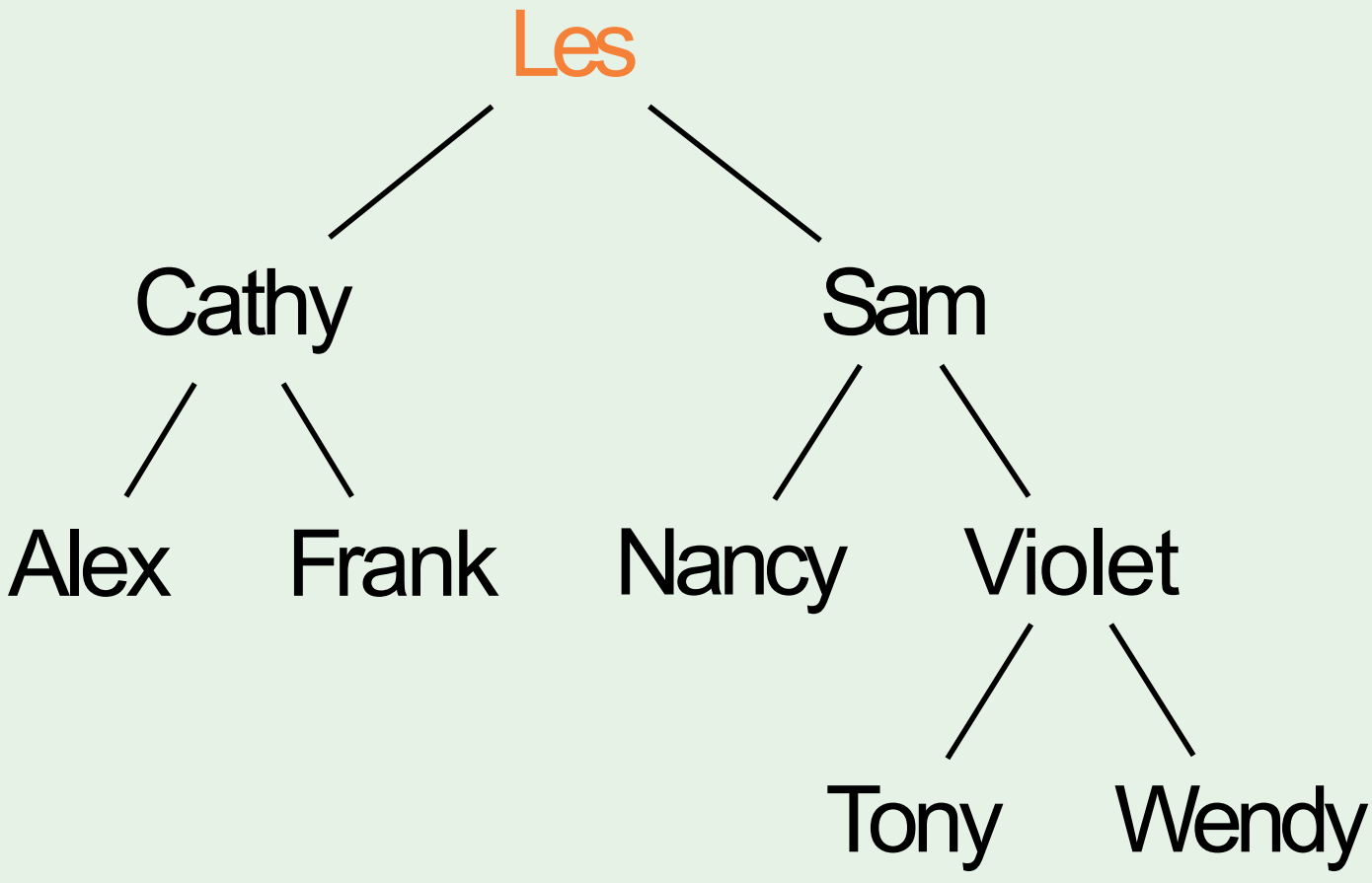
Output: Alex Cathy Frank

InOrderTraversal



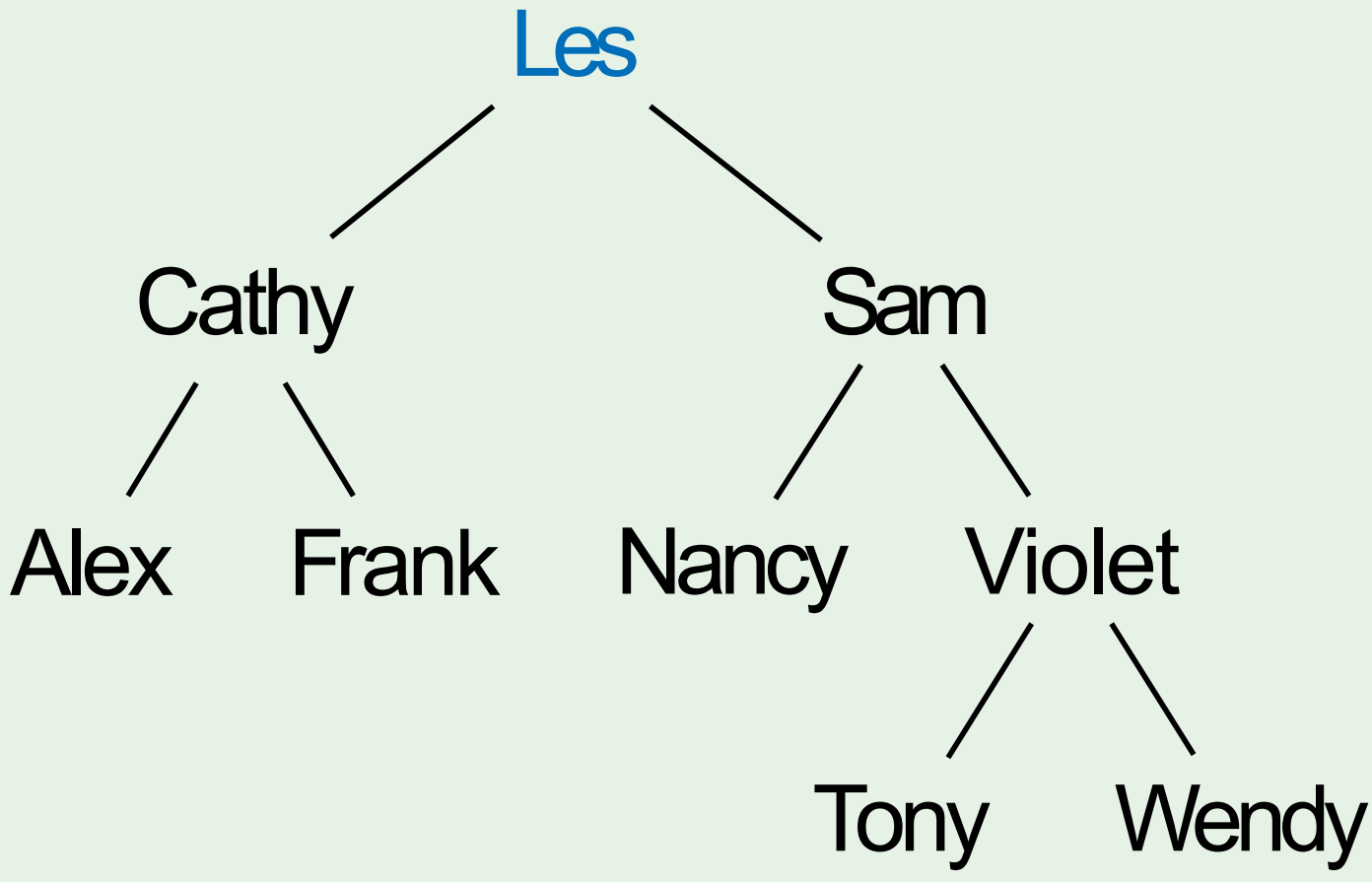
Output: Alex Cathy Frank

InOrderTraversal



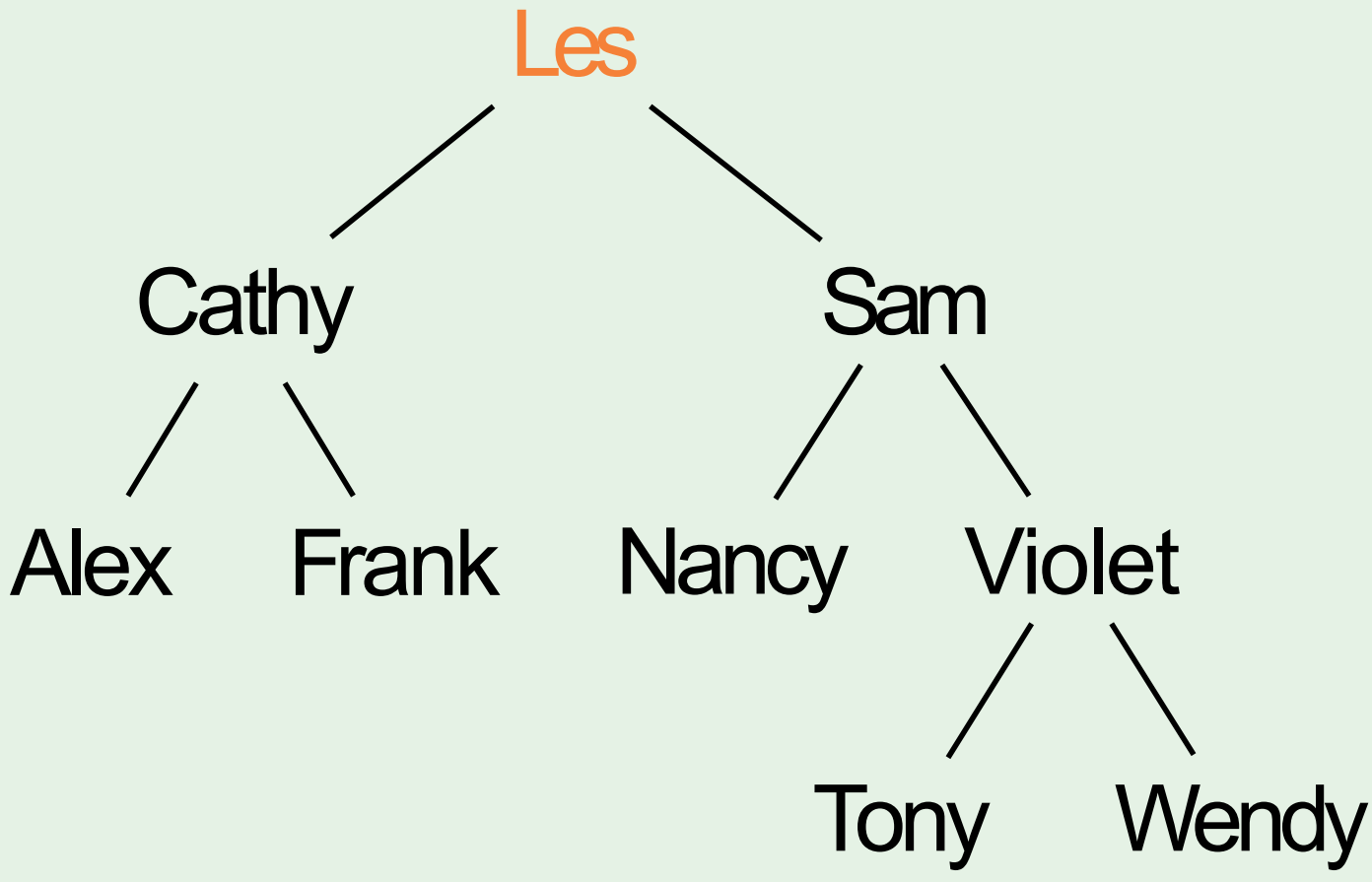
Output: Alex Cathy Frank

InOrderTraversal



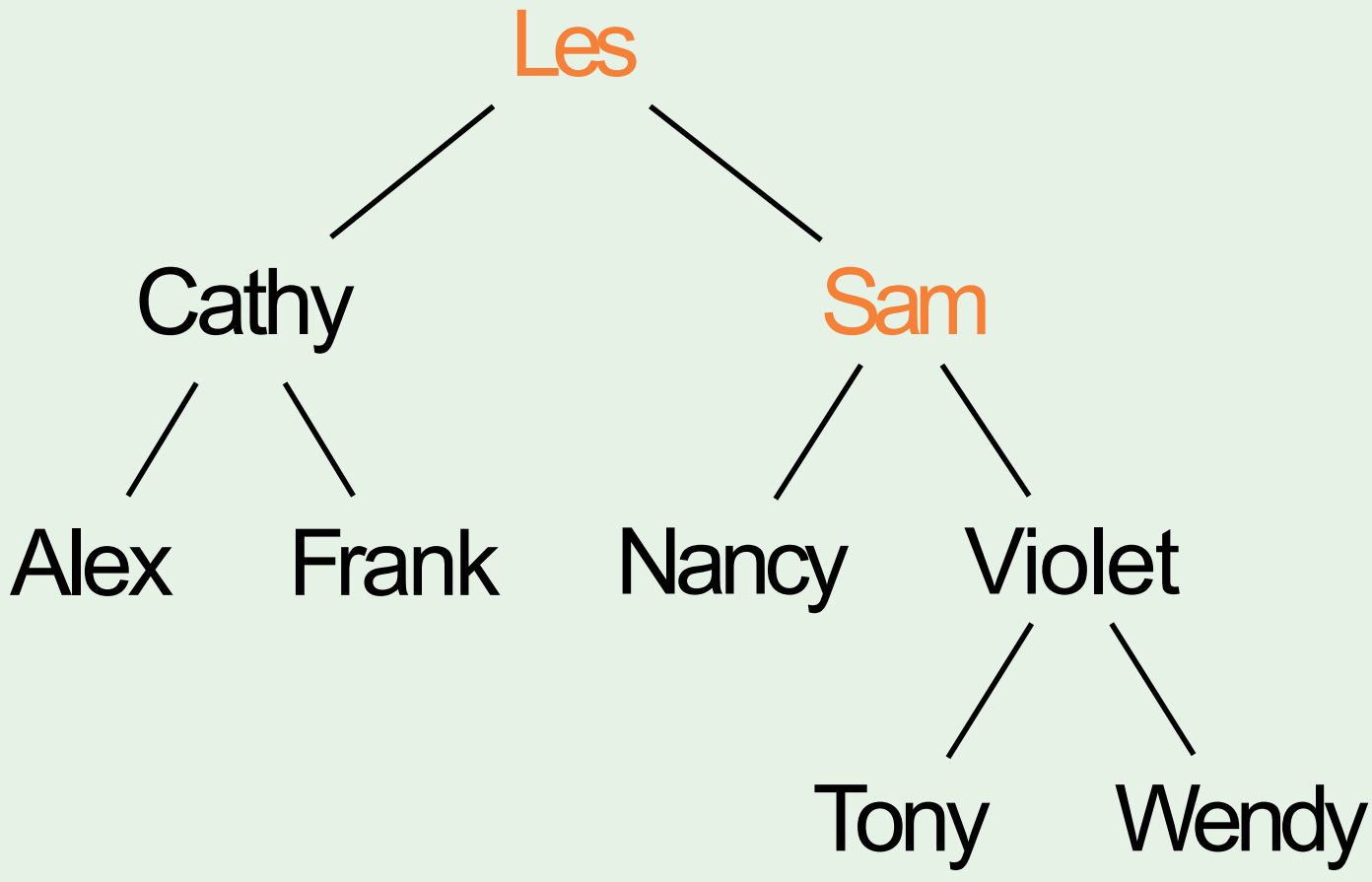
Output: Alex Cathy Frank Les

InOrderTraversal



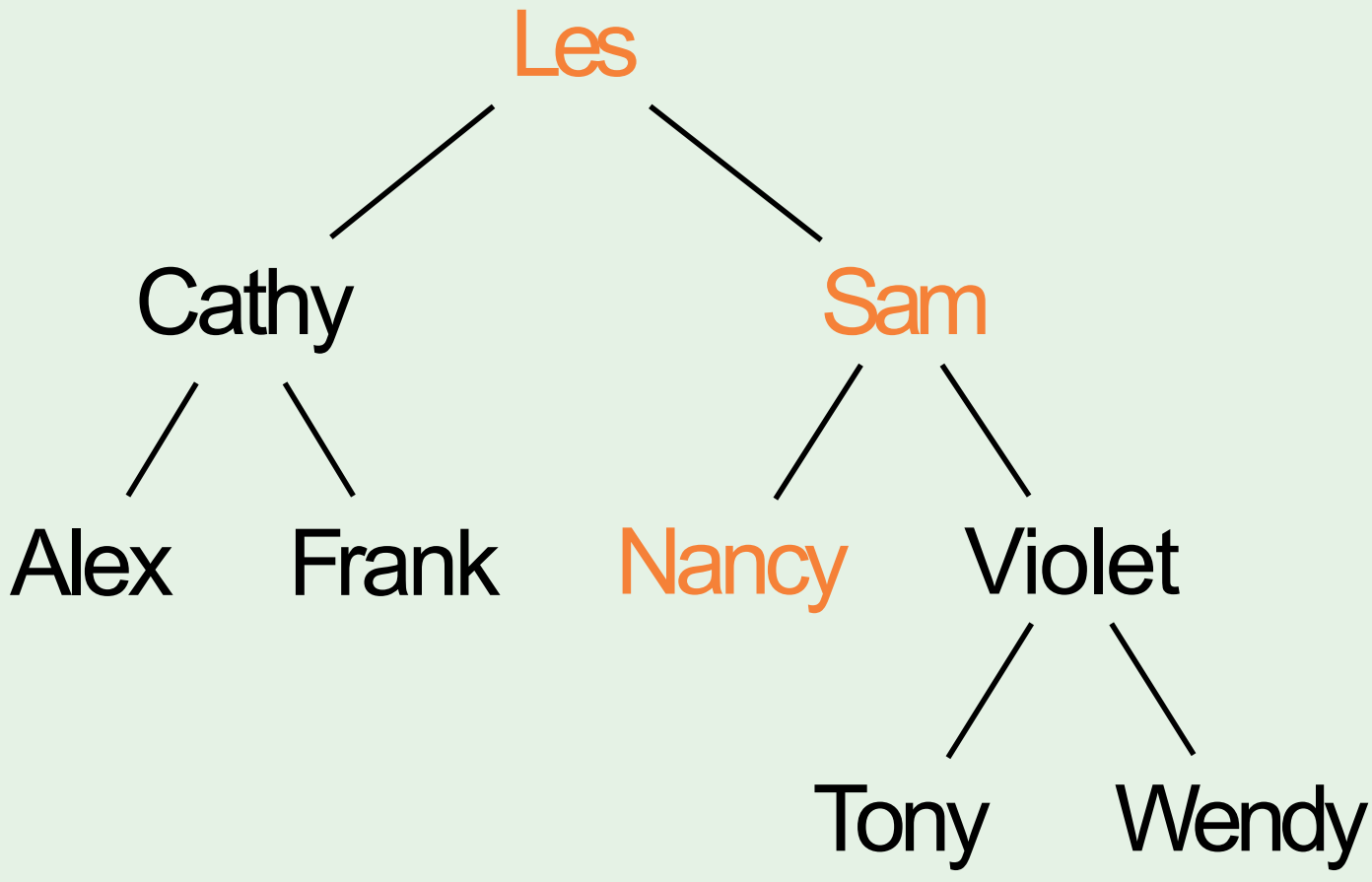
Output: Alex Cathy Frank Les

InOrderTraversal



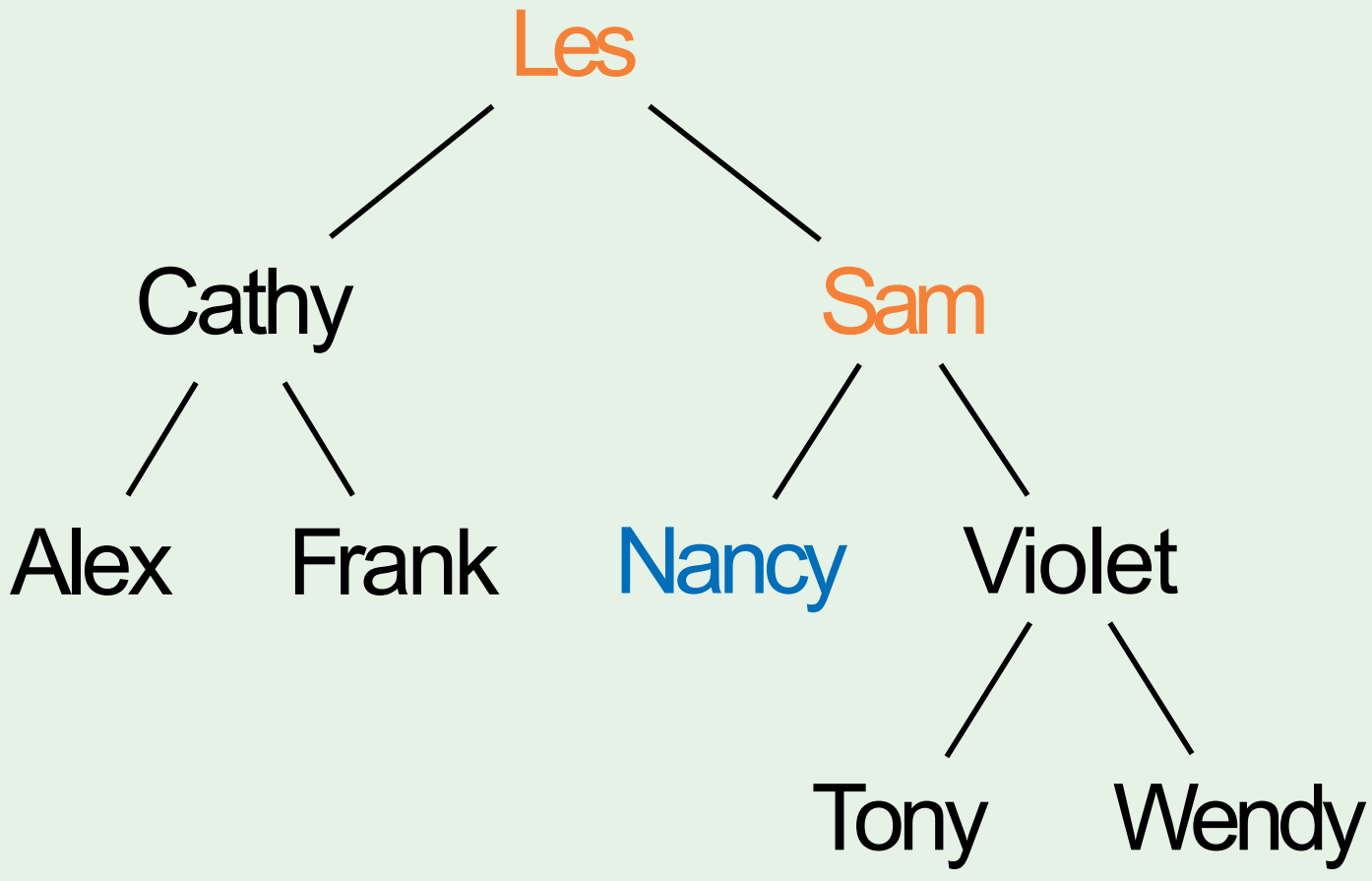
Output: Alex Cathy Frank Les

InOrderTraversal



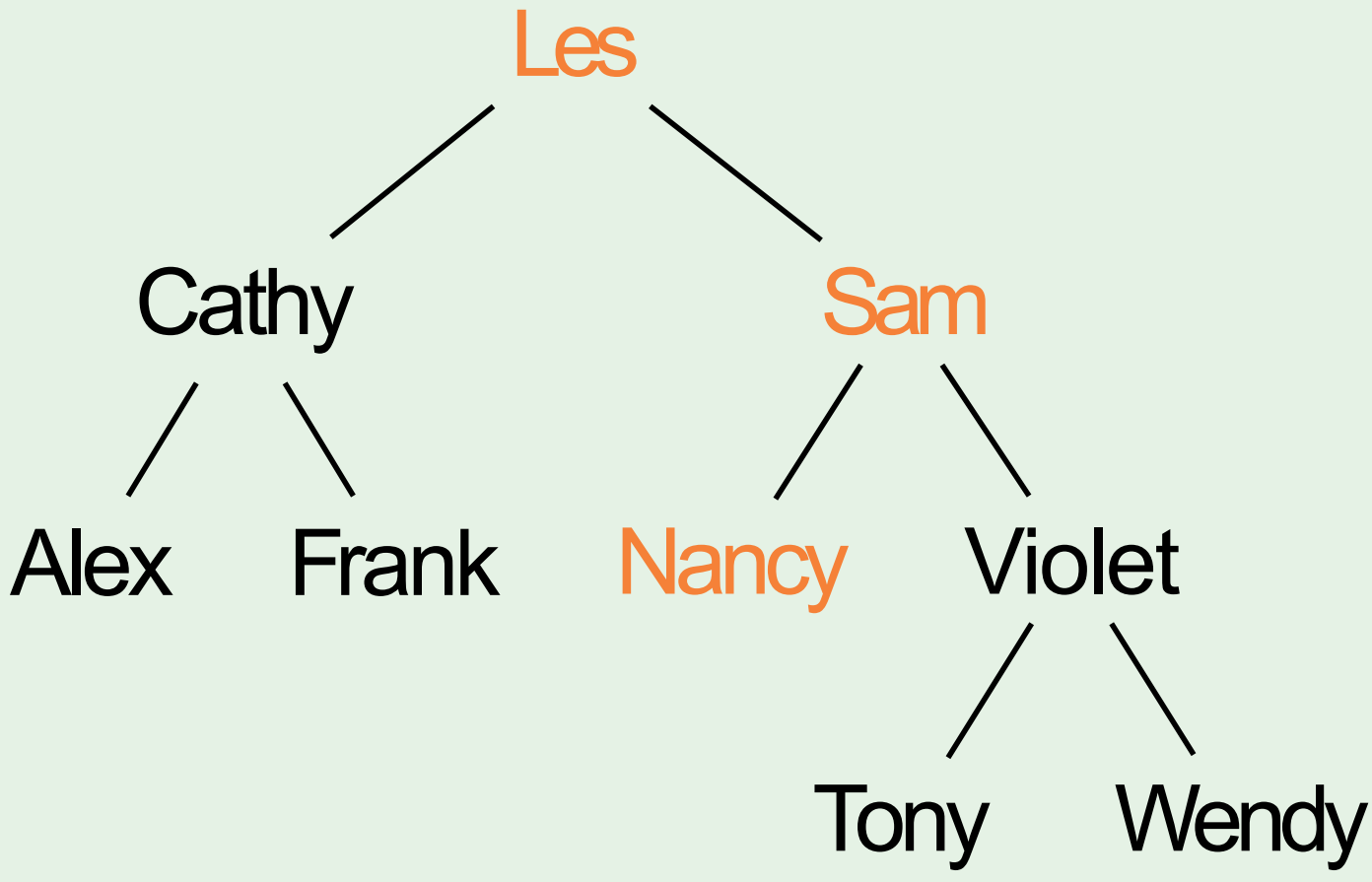
Output: Alex Cathy Frank Les

InOrderTraversal



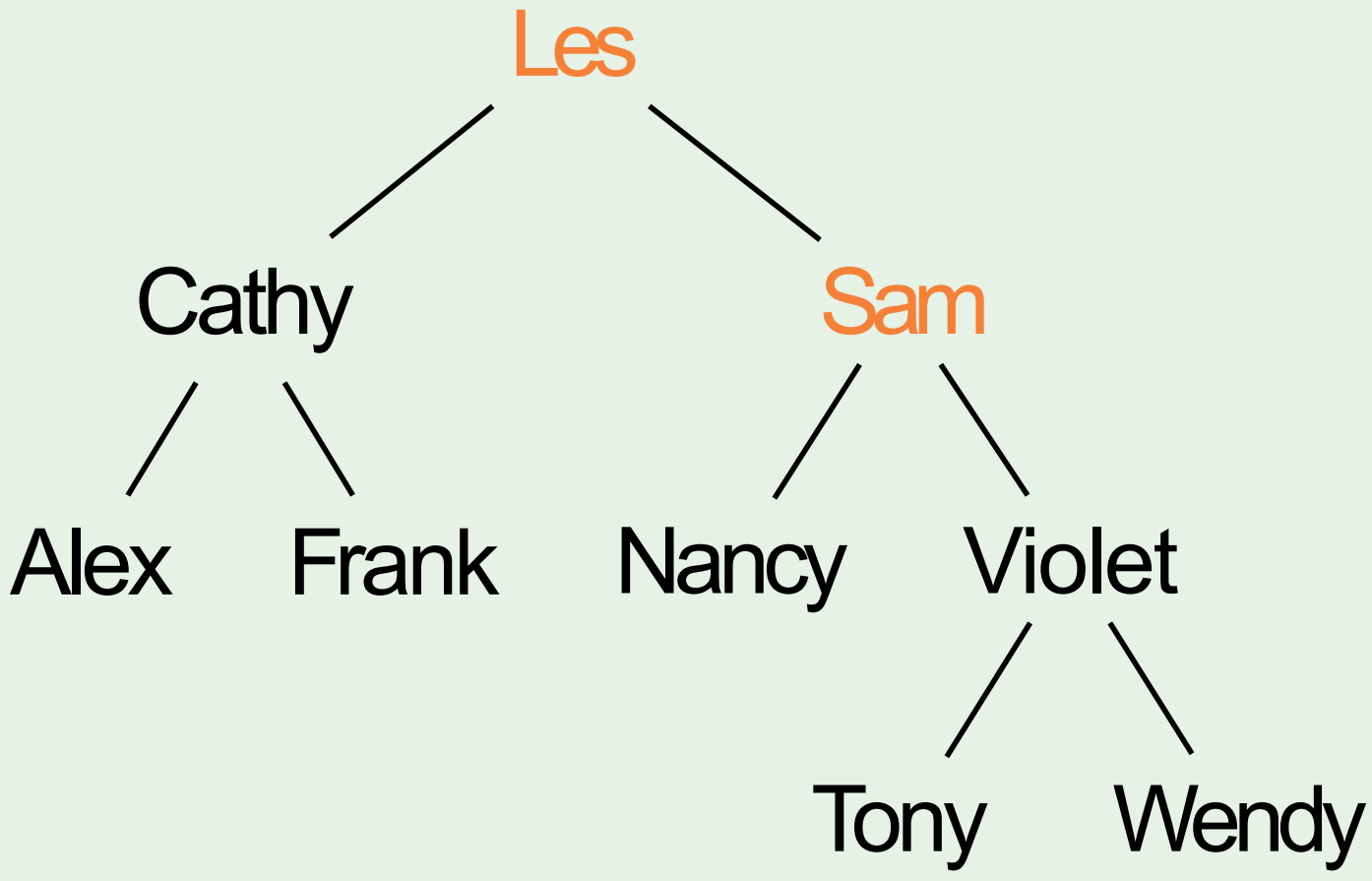
Output: Alex Cathy Frank Les Nancy

InOrderTraversal



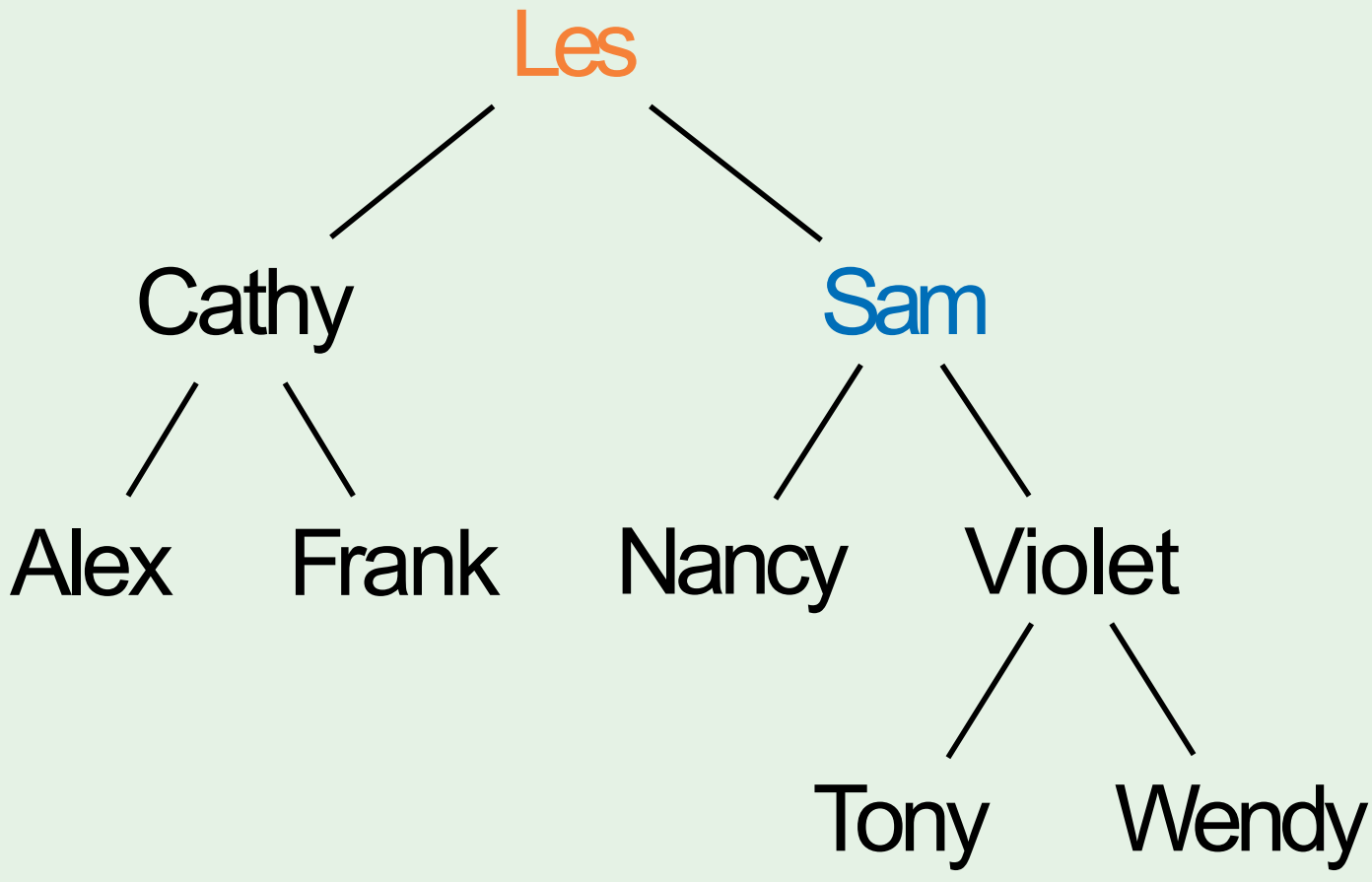
Output: Alex Cathy Frank Les Nancy

InOrderTraversal



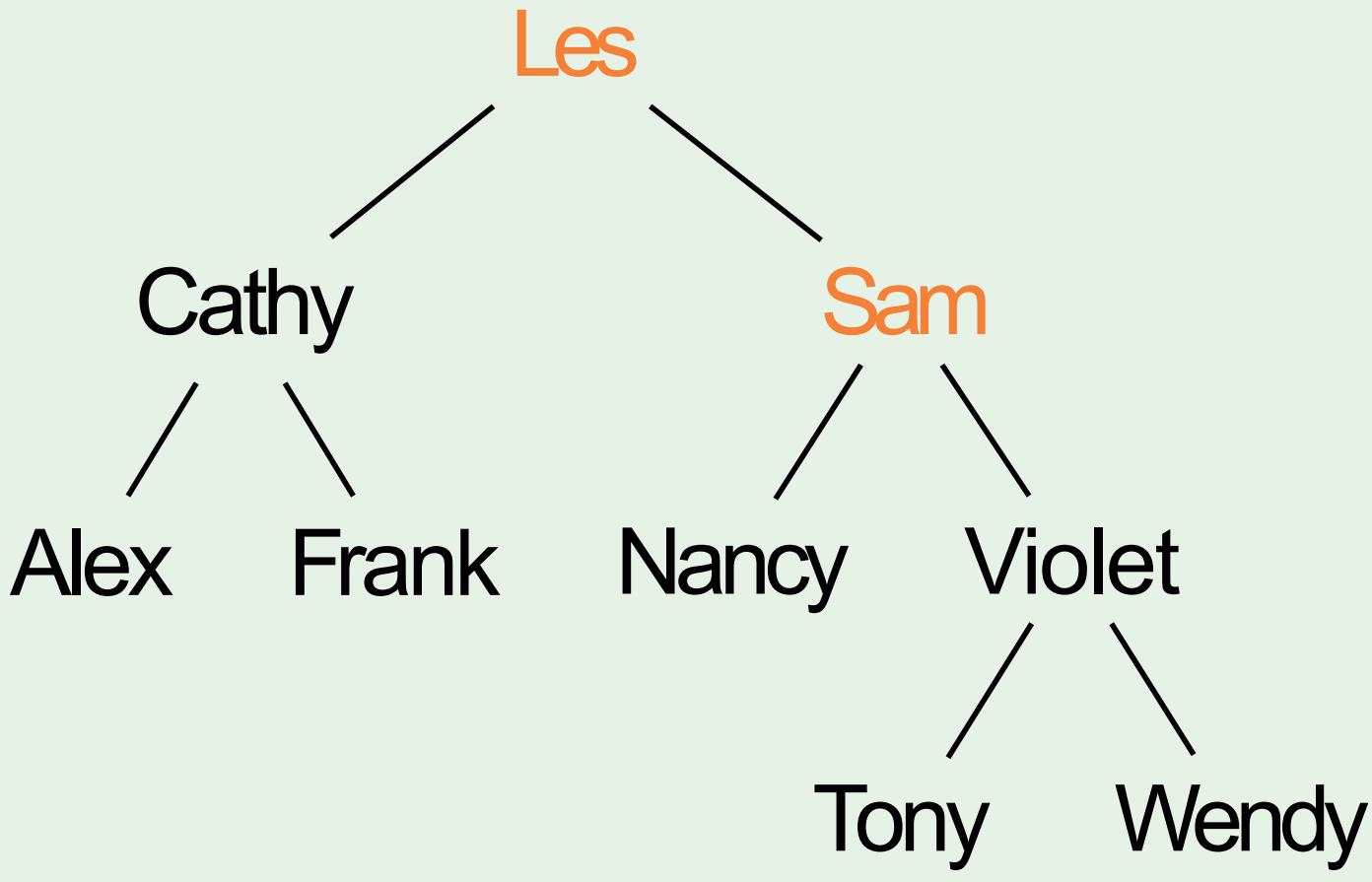
Output: Alex Cathy Frank Les Nancy

InOrderTraversal



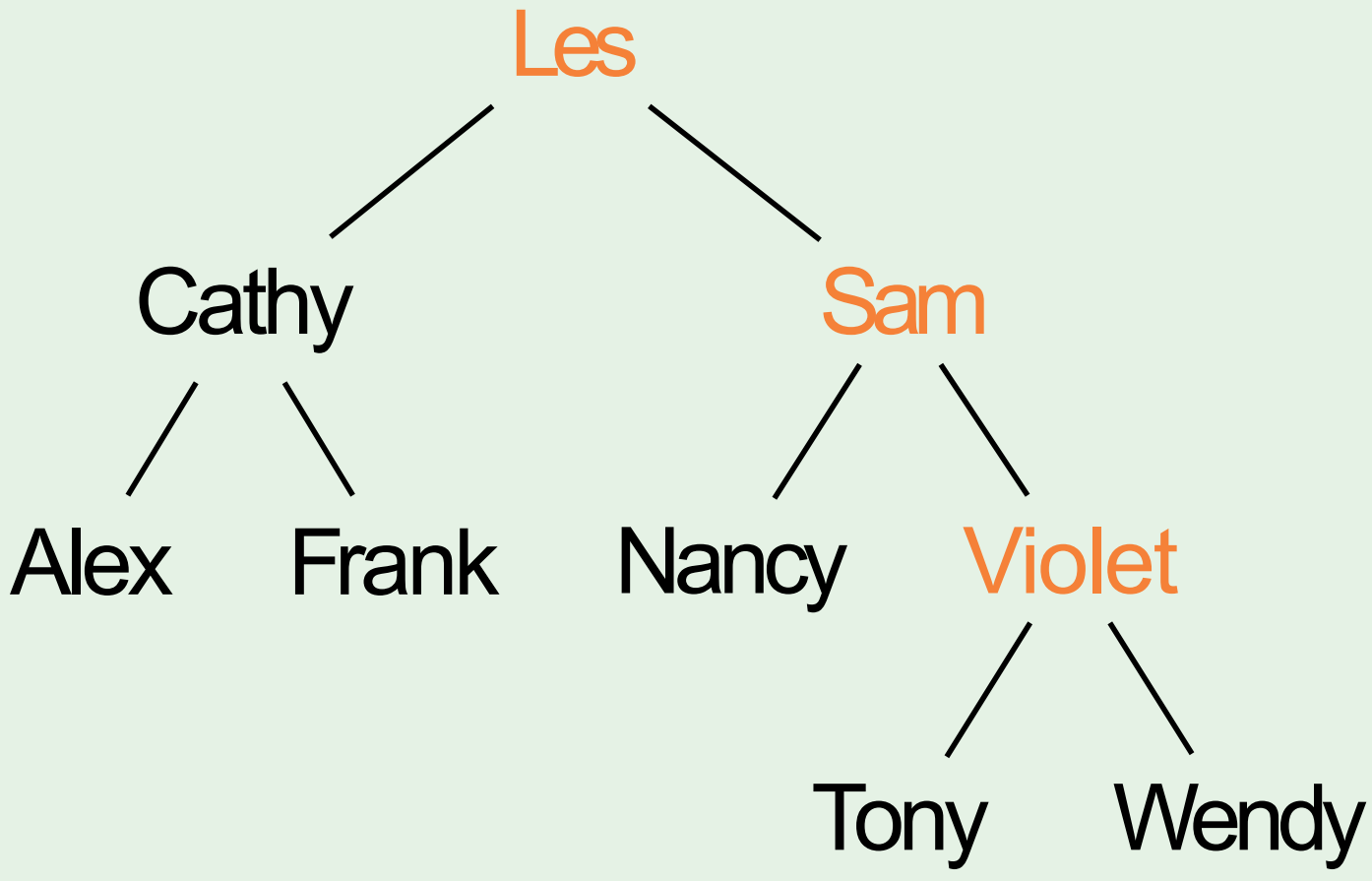
Output: Alex Cathy Frank Les Nancy Sam

InOrderTraversal



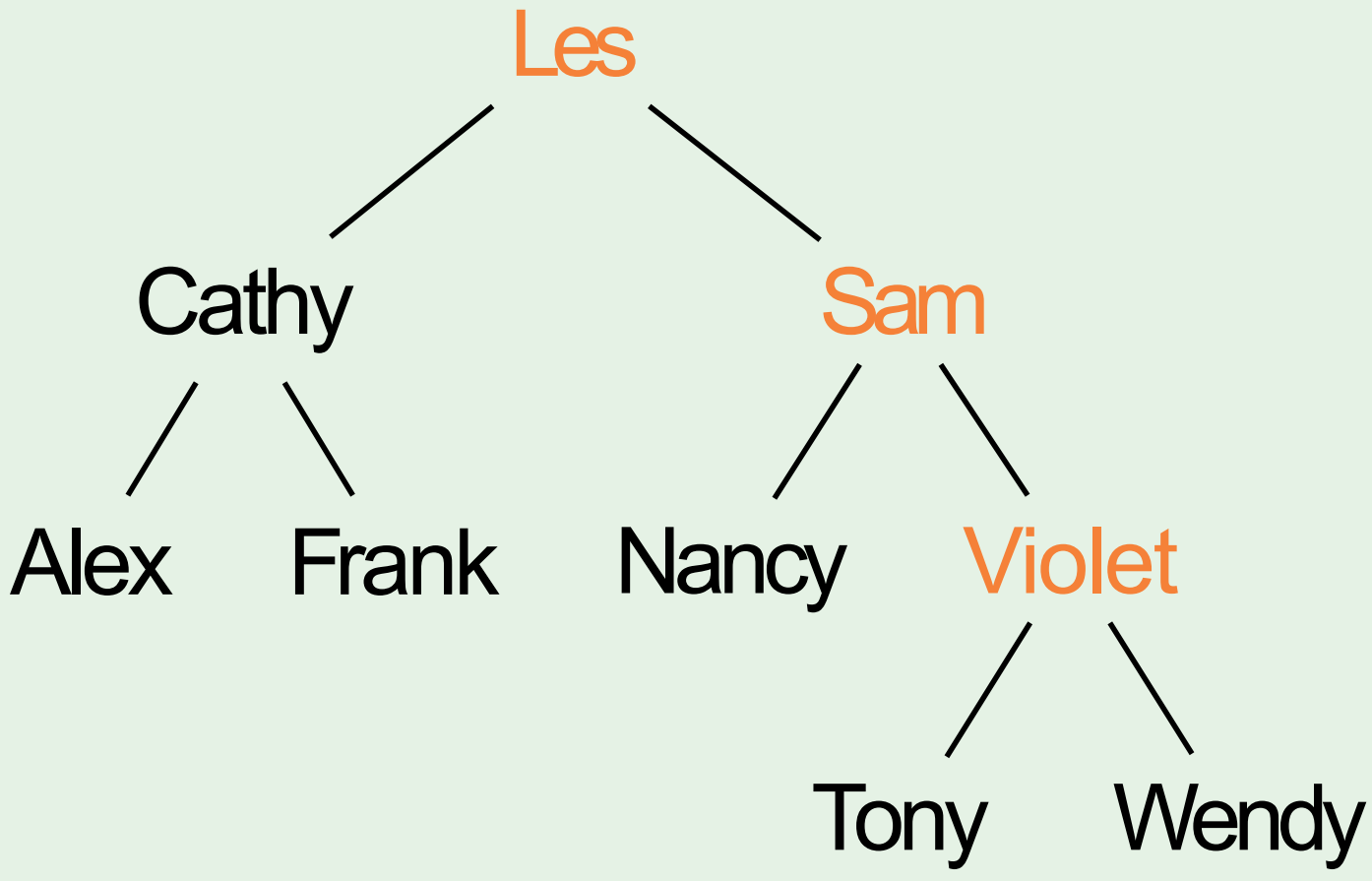
Output: Alex Cathy Frank Les Nancy Sam

InOrderTraversal



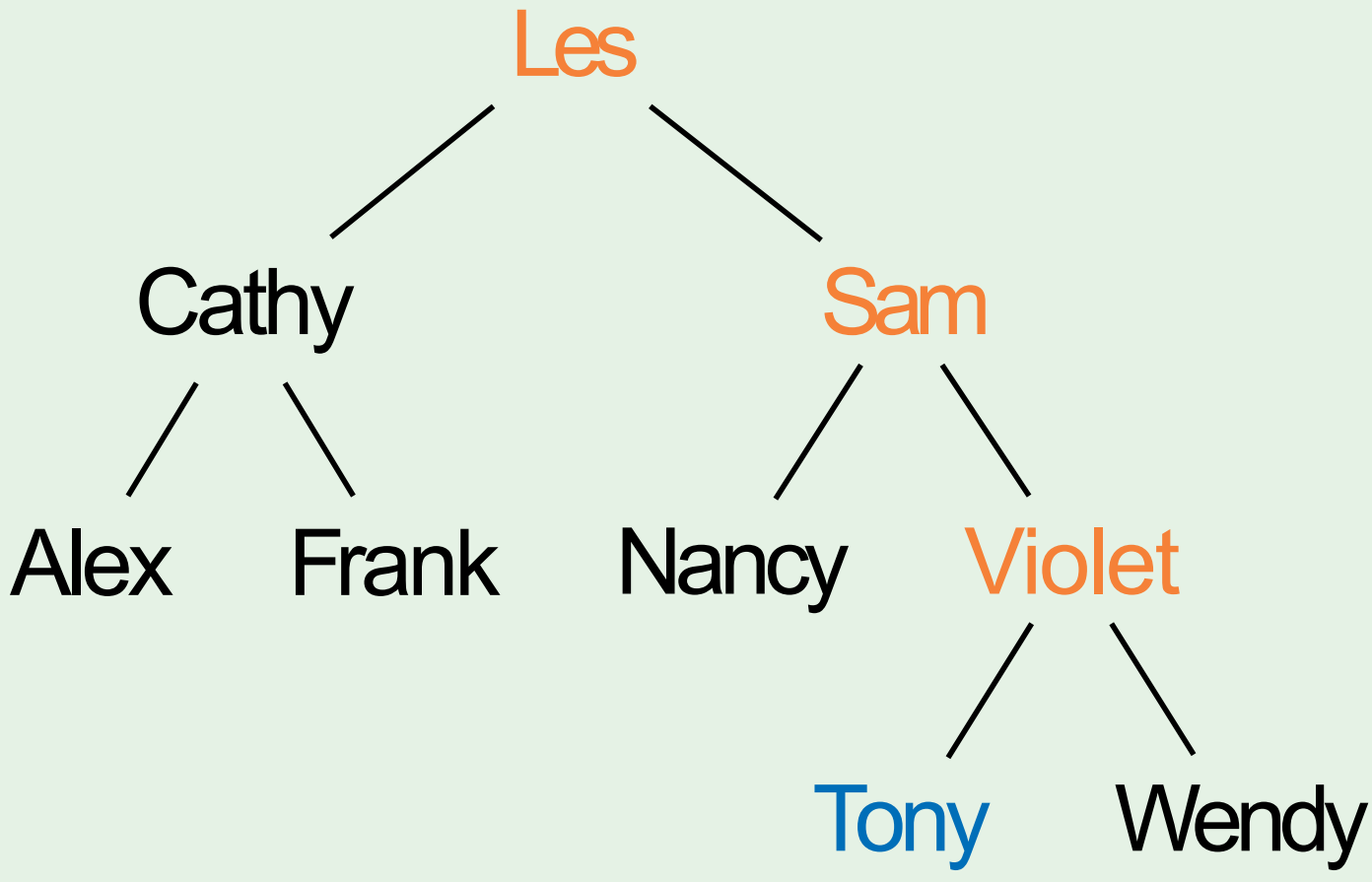
Output: Alex Cathy Frank Les Nancy Sam

InOrderTraversal



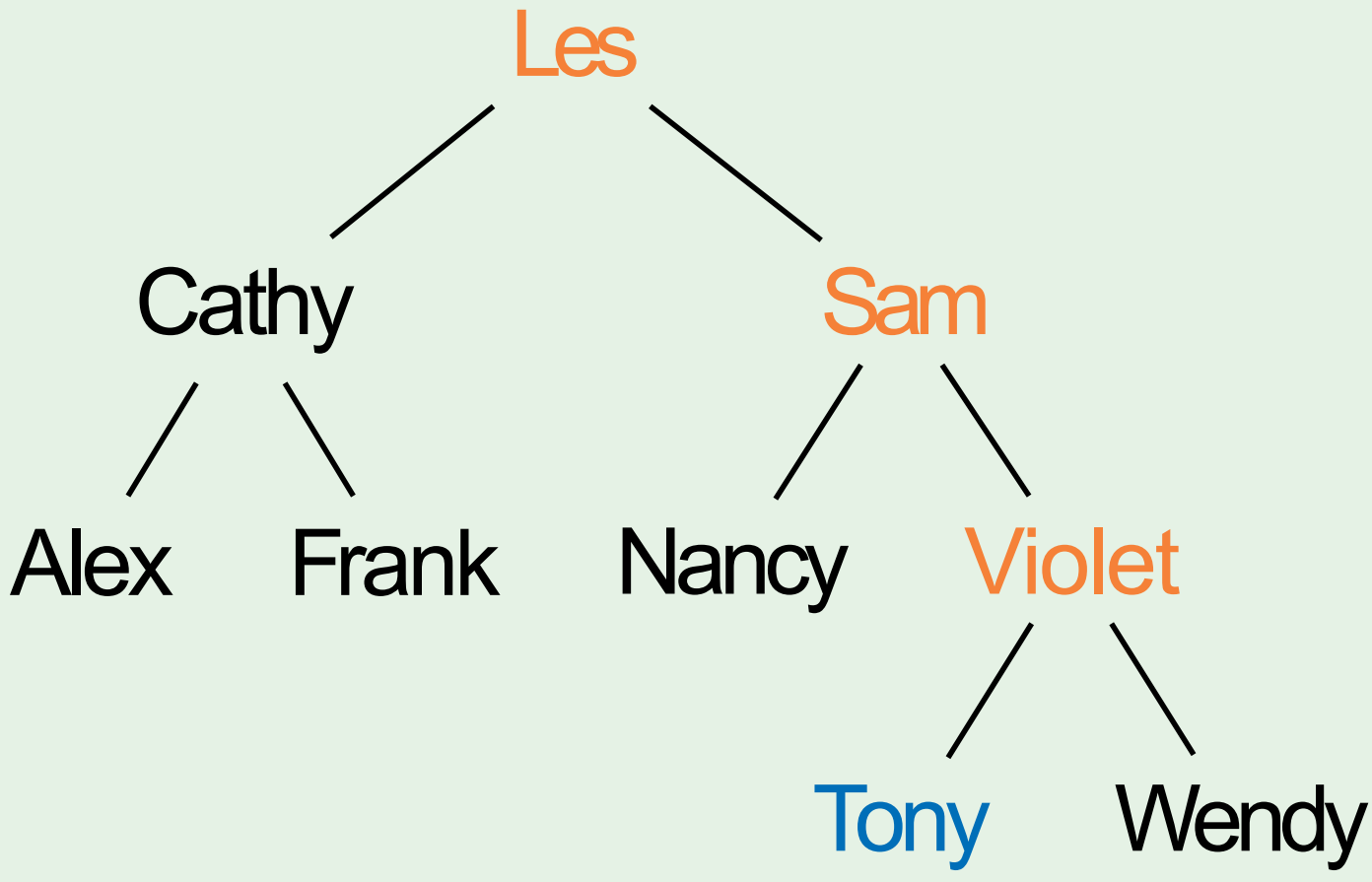
Output: Alex Cathy Frank Les Nancy Sam

InOrderTraversal



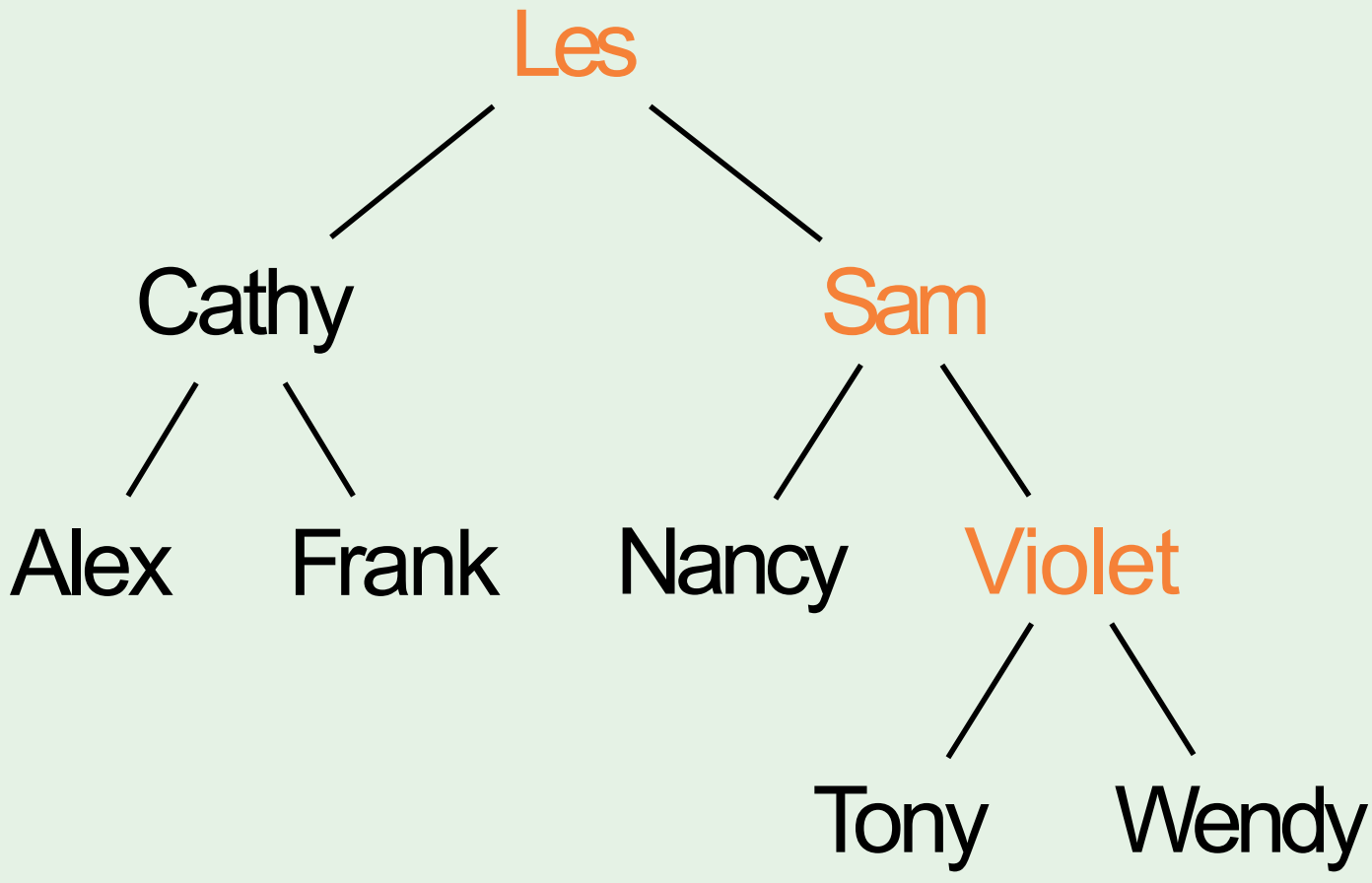
Output: Alex Cathy Frank Les Nancy Sam Tony

InOrderTraversal



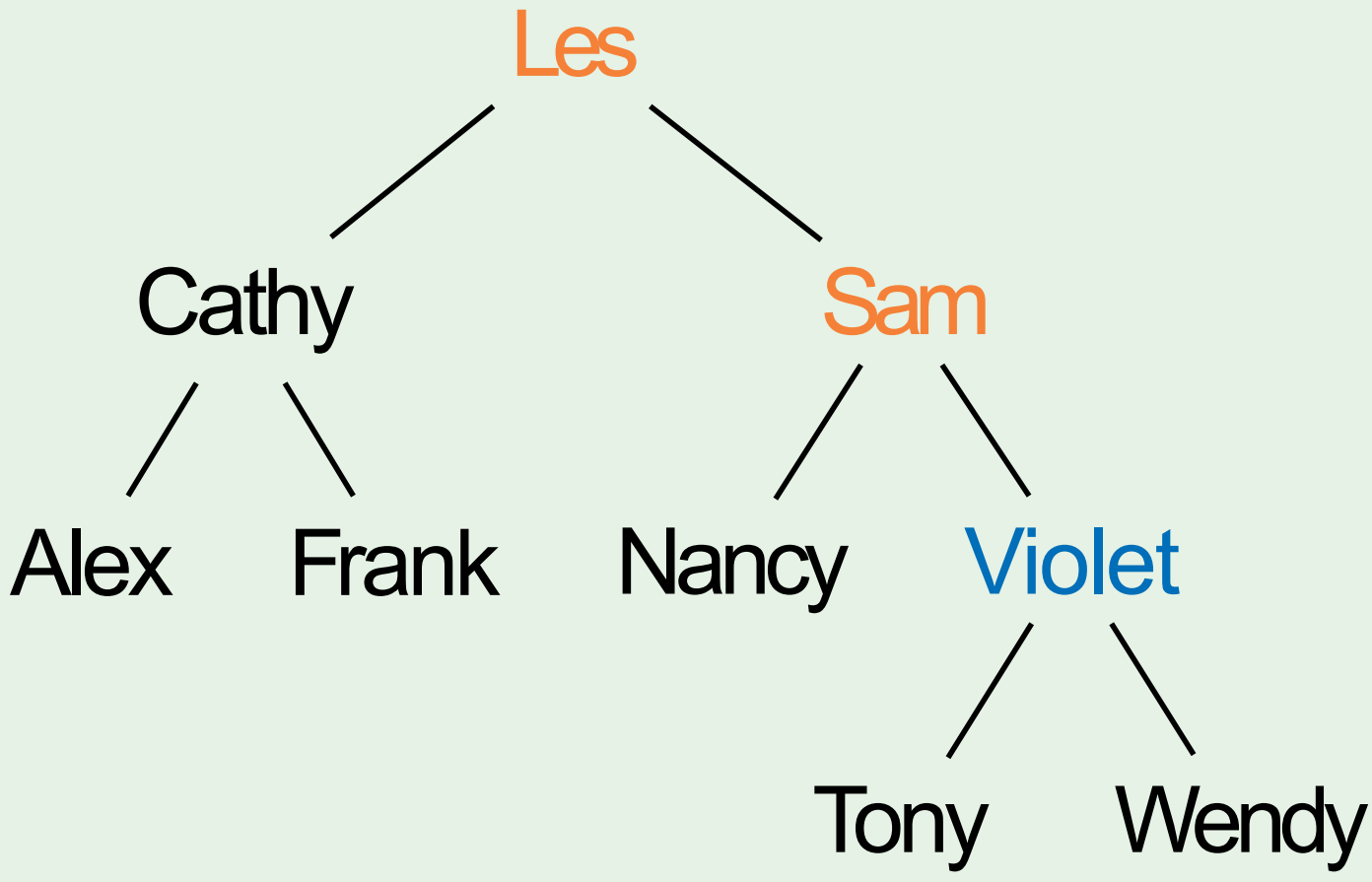
Output: Alex Cathy Frank Les Nancy Sam Tony

InOrderTraversal



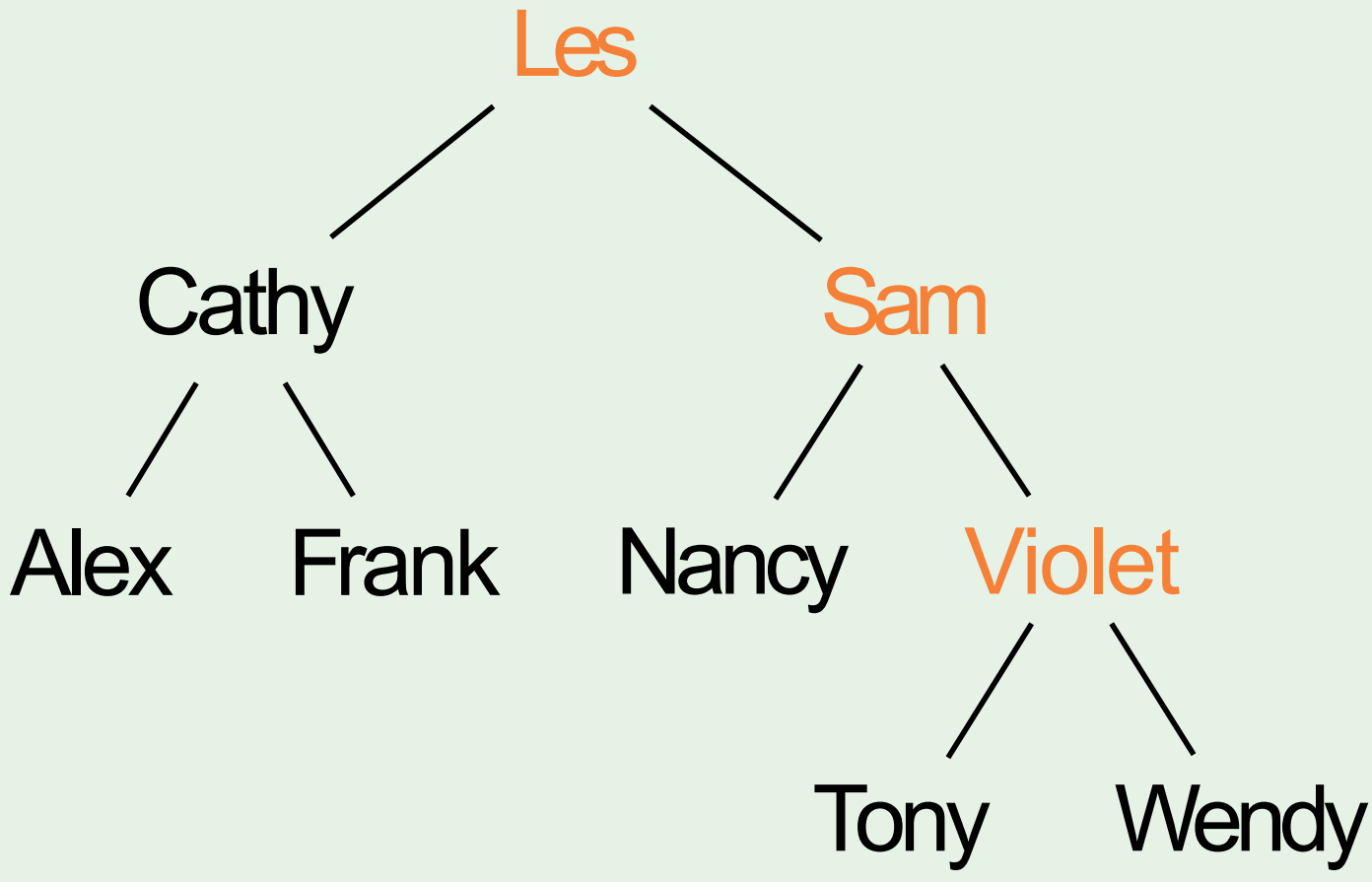
Output: Alex Cathy Frank Les Nancy Sam Tony

InOrderTraversal



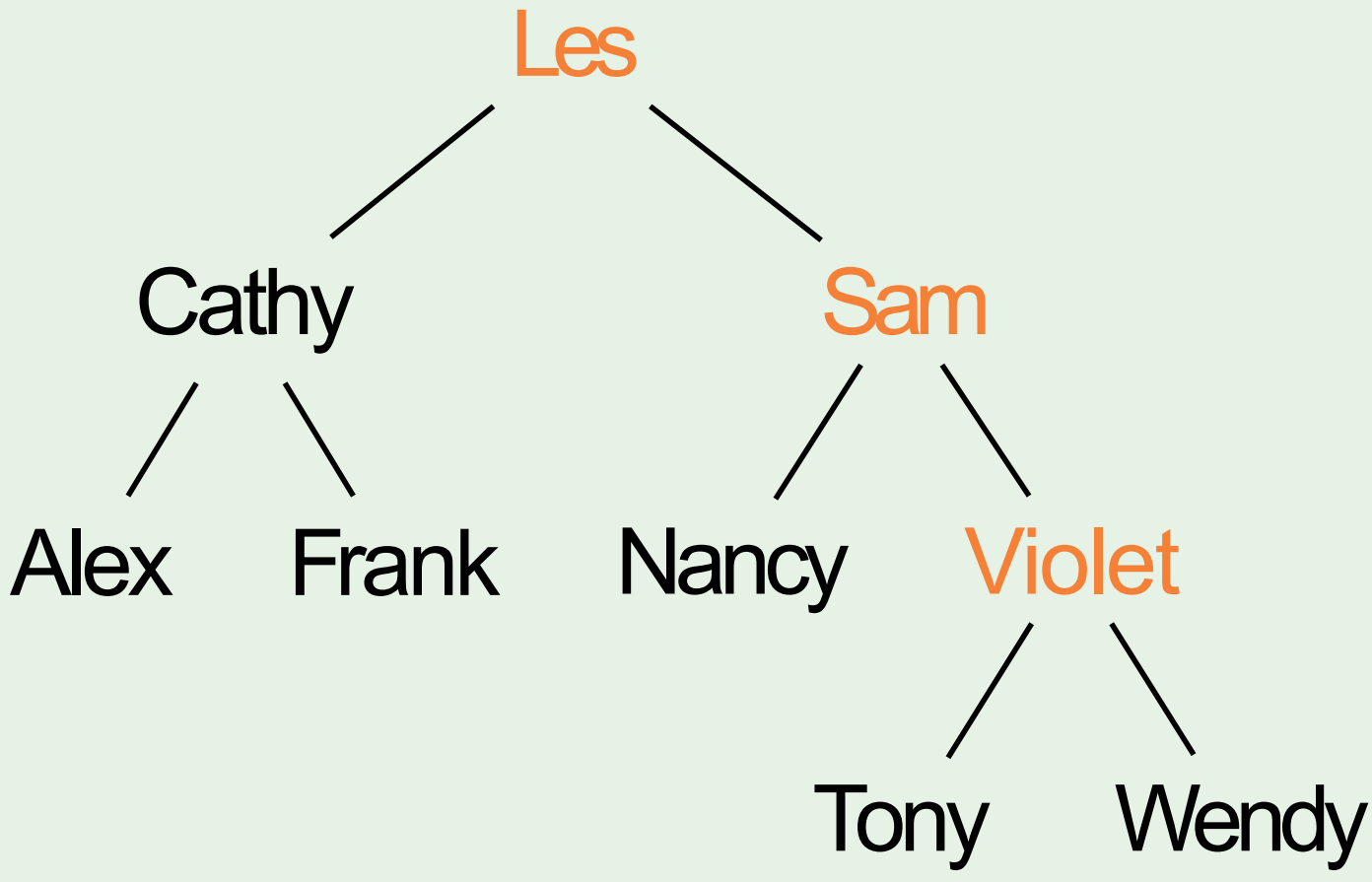
Output: Alex Cathy Frank Les Nancy Sam Tony Violet

InOrderTraversal



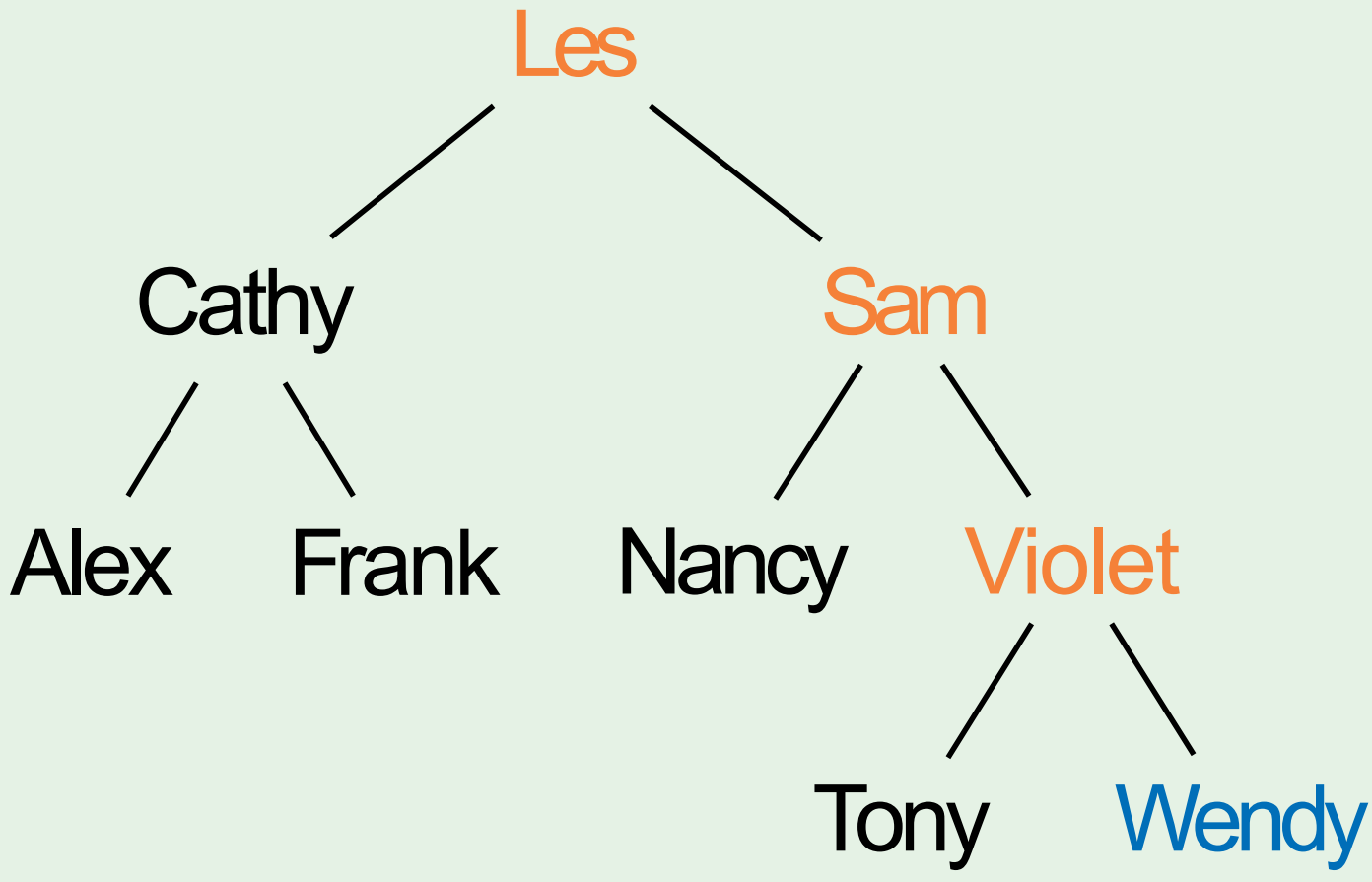
Output: Alex Cathy Frank Les Nancy Sam Tony Violet

InOrderTraversal



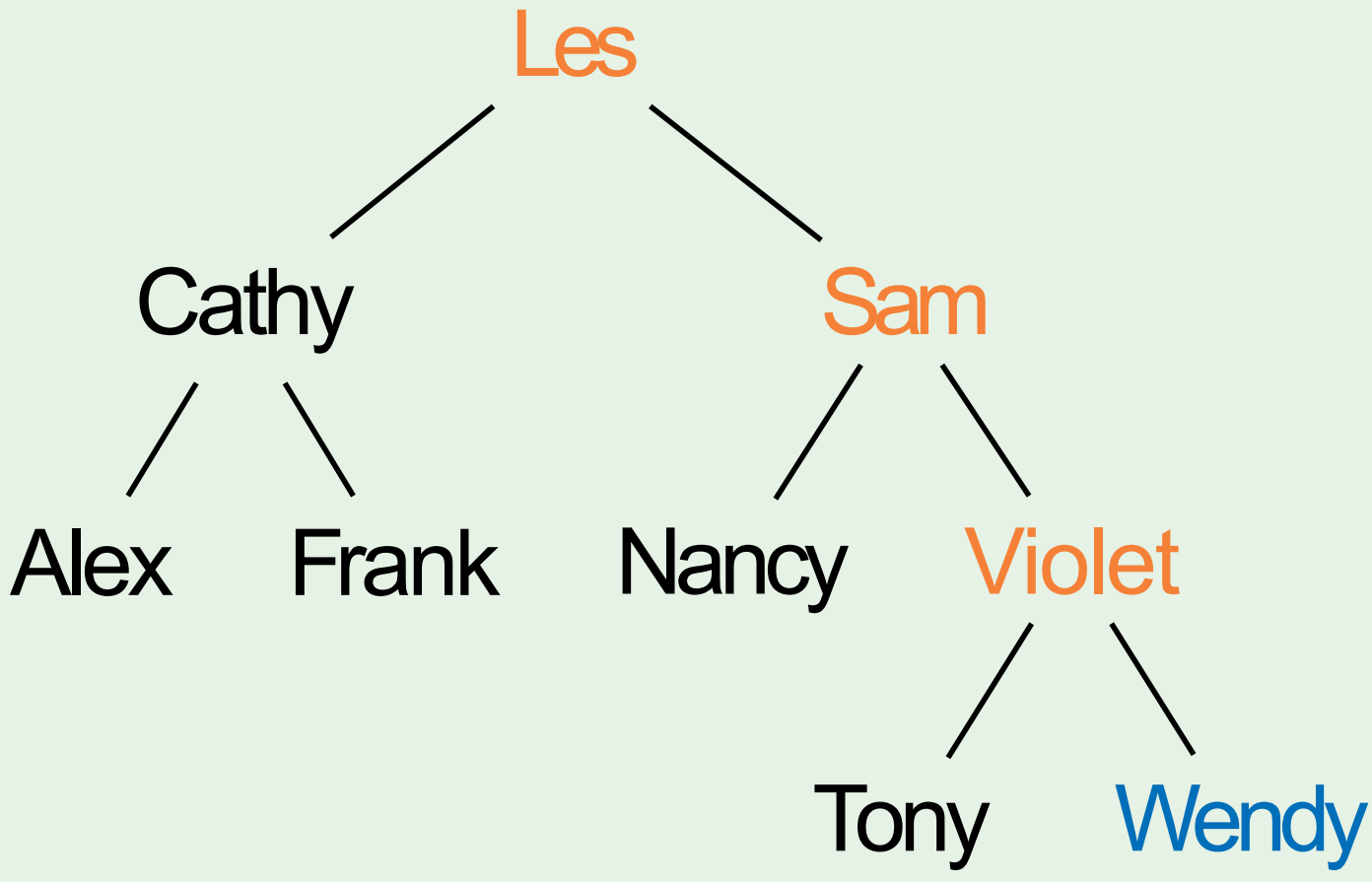
Output: Alex Cathy Frank Les Nancy Sam Tony Violet

InOrderTraversal



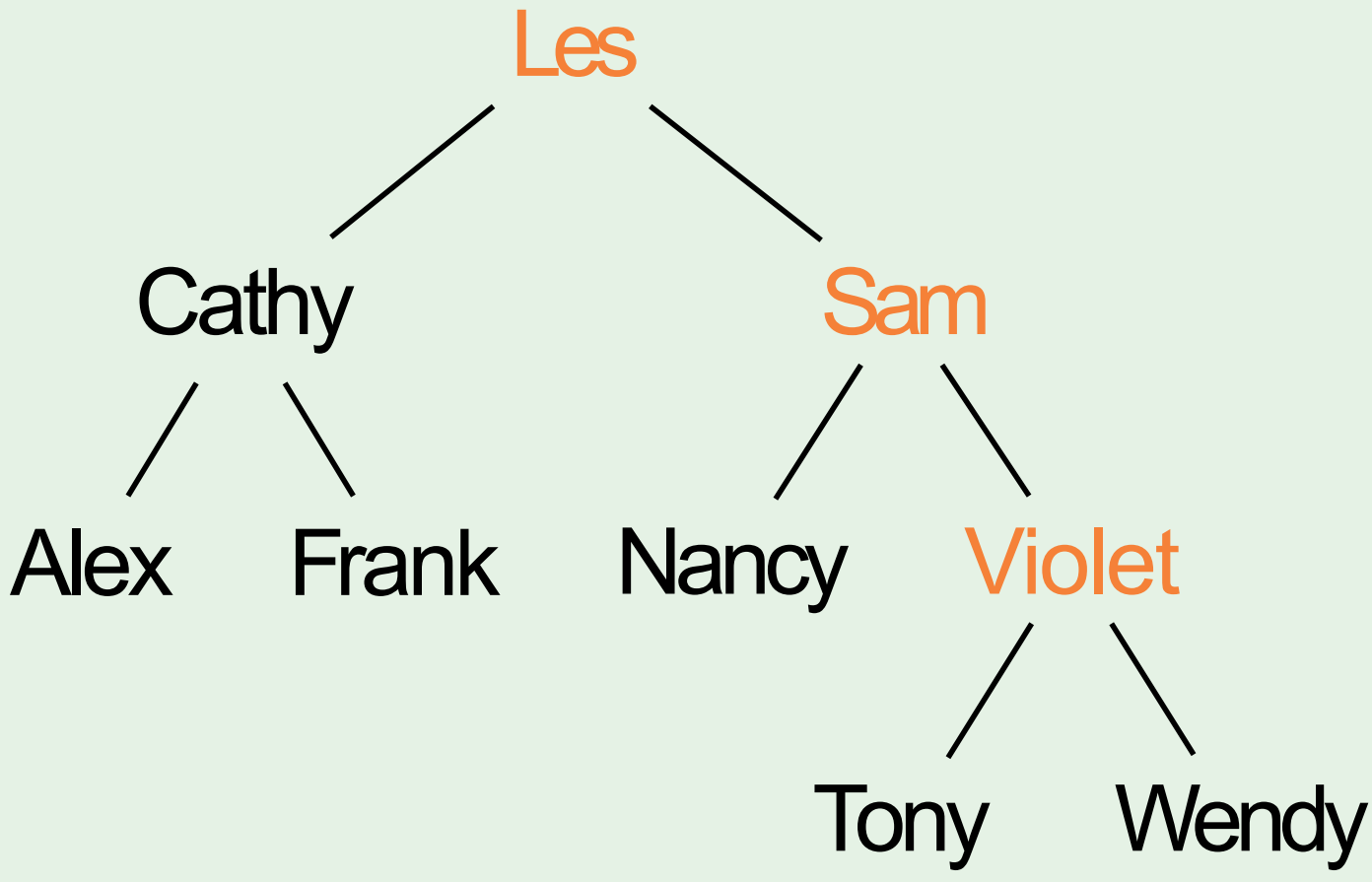
Output: Alex Cathy Frank Les Nancy Sam Tony Violet Wendy

InOrderTraversal



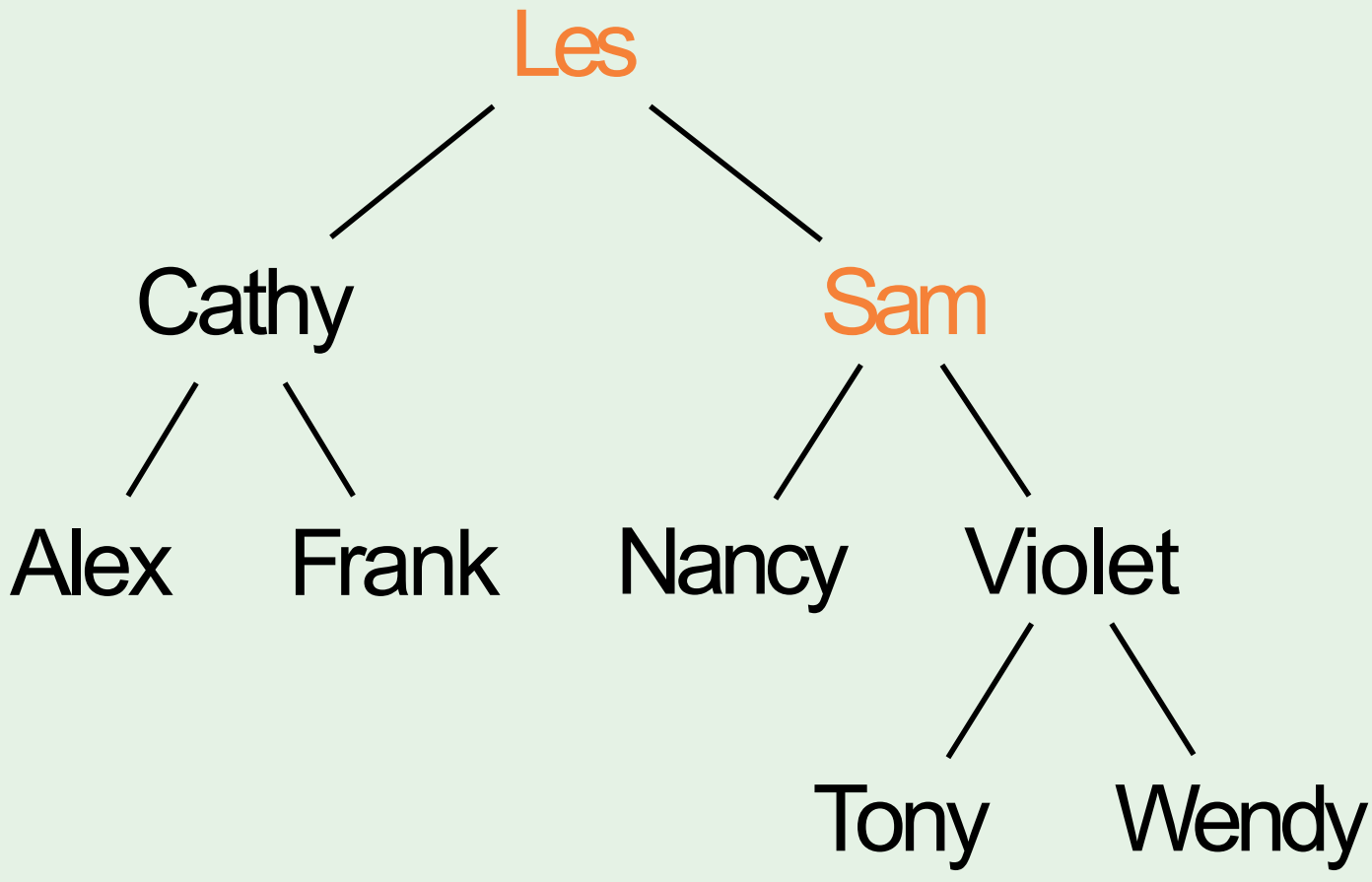
Output: Alex Cathy Frank Les Nancy Sam Tony Violet Wendy

InOrderTraversal



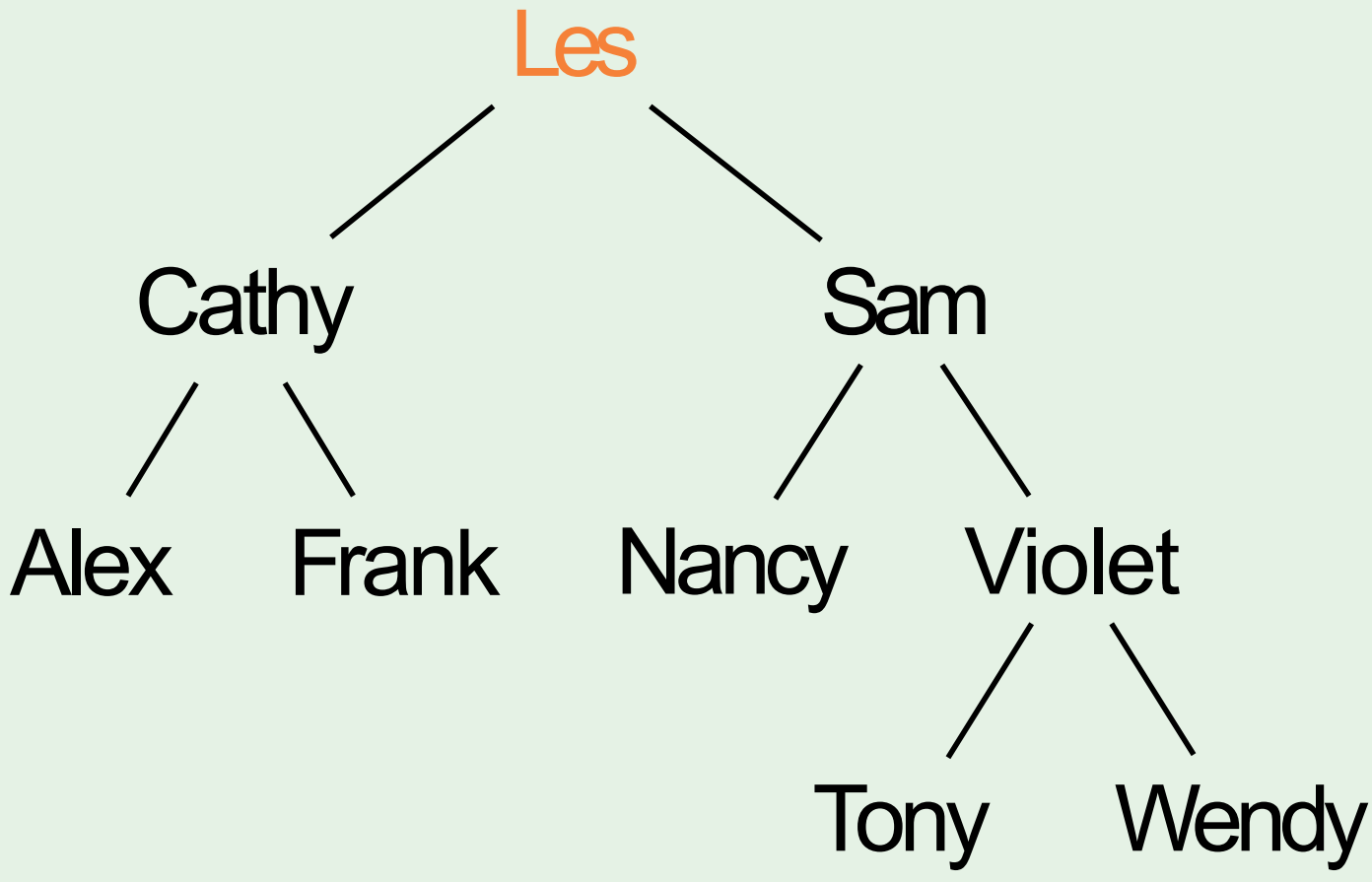
Output: Alex Cathy Frank Les Nancy Sam Tony Violet Wendy

InOrderTraversal



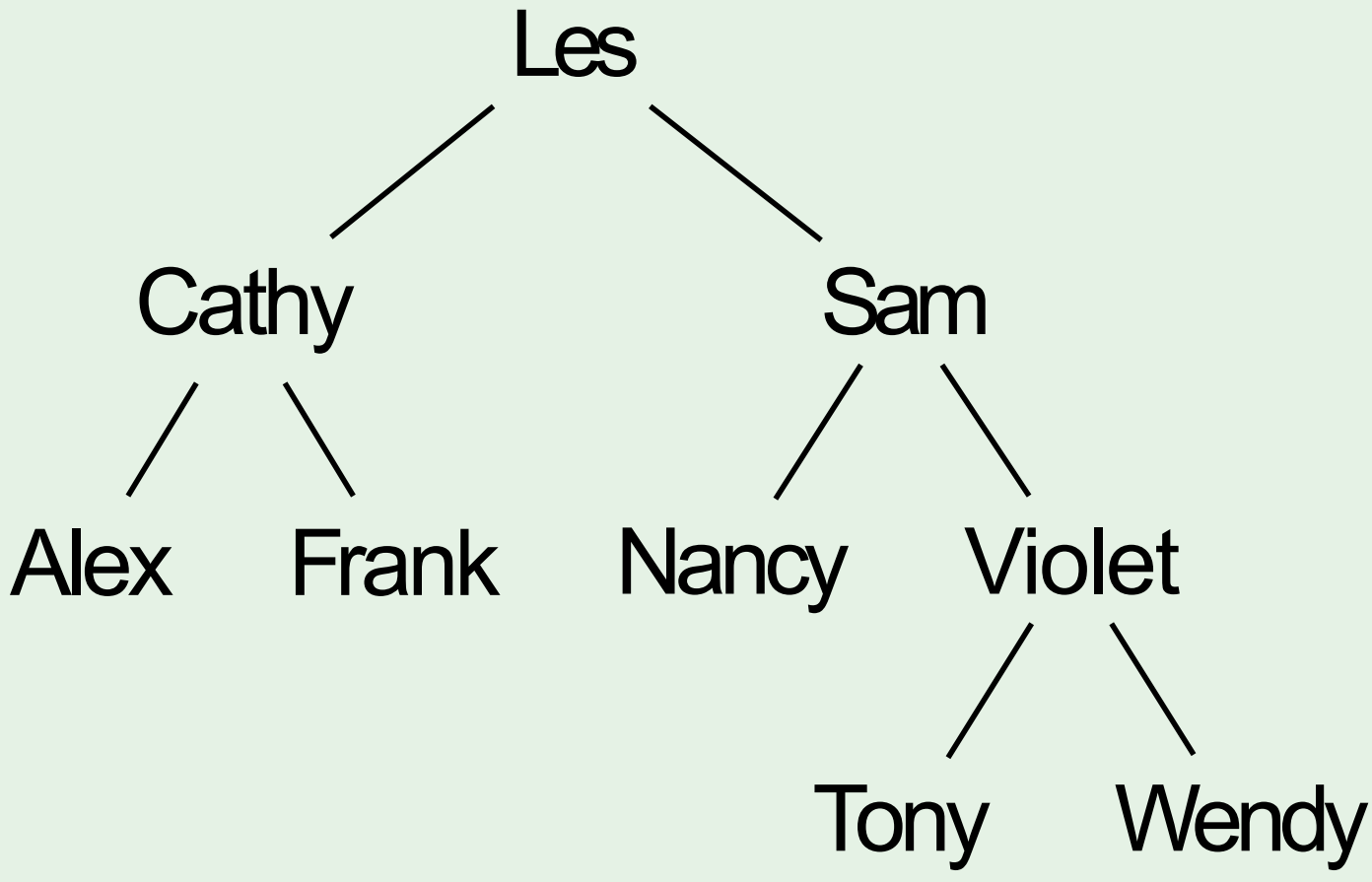
Output: Alex Cathy Frank Les Nancy Sam Tony Violet Wendy

InOrderTraversal



Output: Alex Cathy Frank Les Nancy Sam Tony Violet Wendy

InOrderTraversal



Output: Alex Cathy Frank Les Nancy Sam Tony Violet Wendy

Assignment

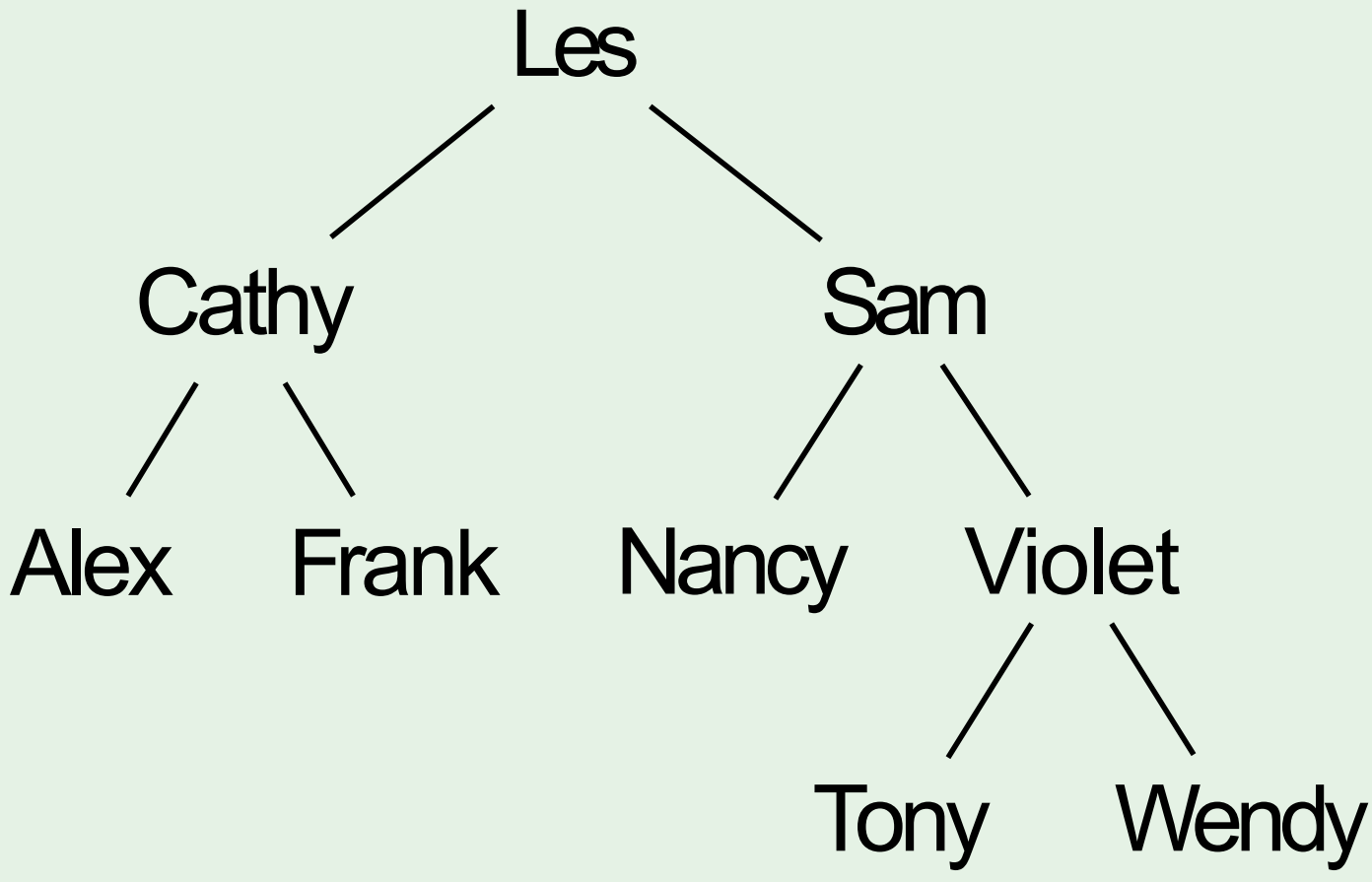
- Take input from user in InOrder and create the binary tree.
- Draw the Tree

Depth-first

PreOrderTraversal(*tree*)

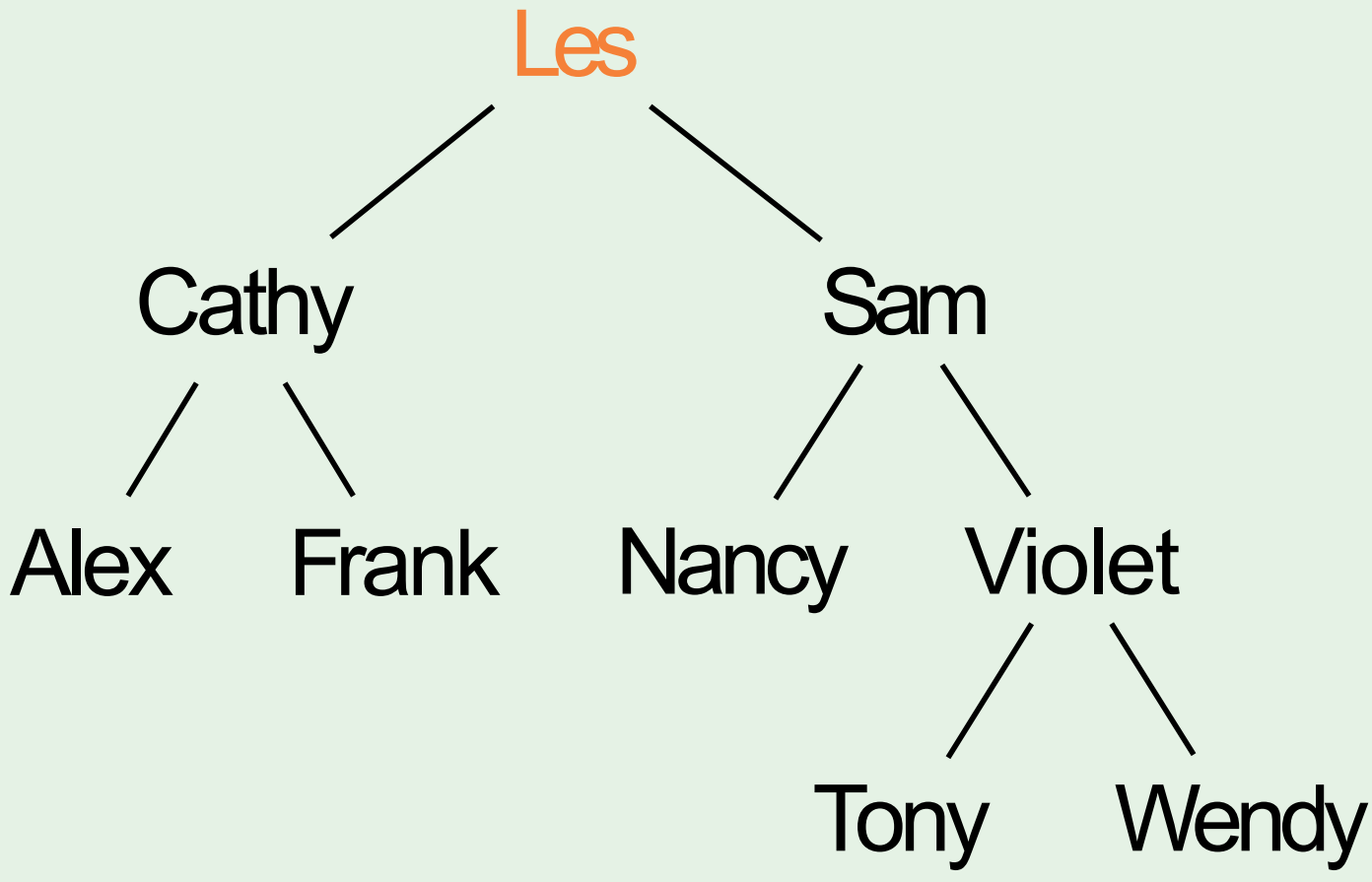
```
if tree = nil:
   return
Print(tree.key)
PreOrderTraversal(tree.left)
PreOrderTraversal(tree.right)
```

PreOrderTraversal



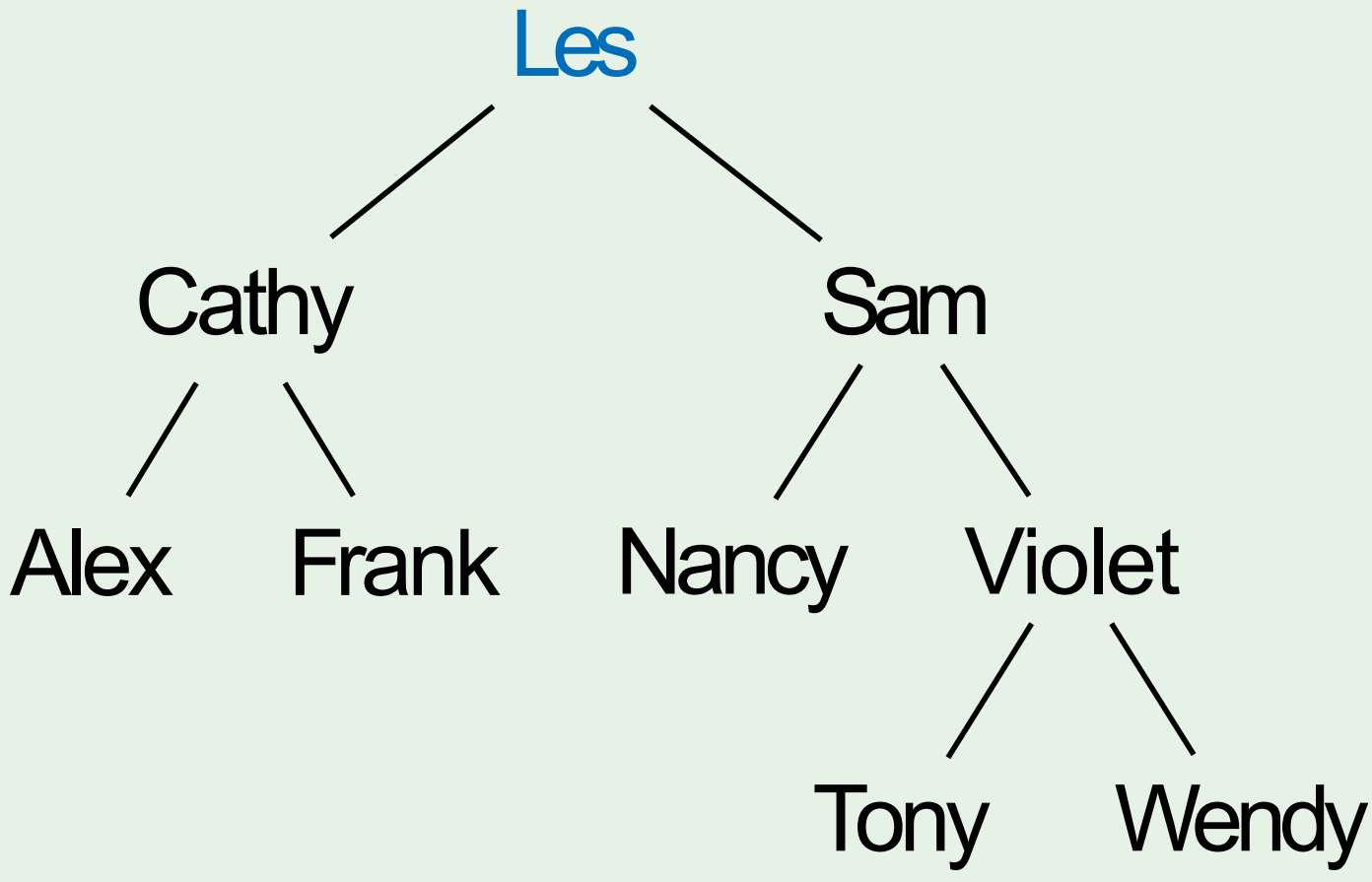
Output:

PreOrderTraversal



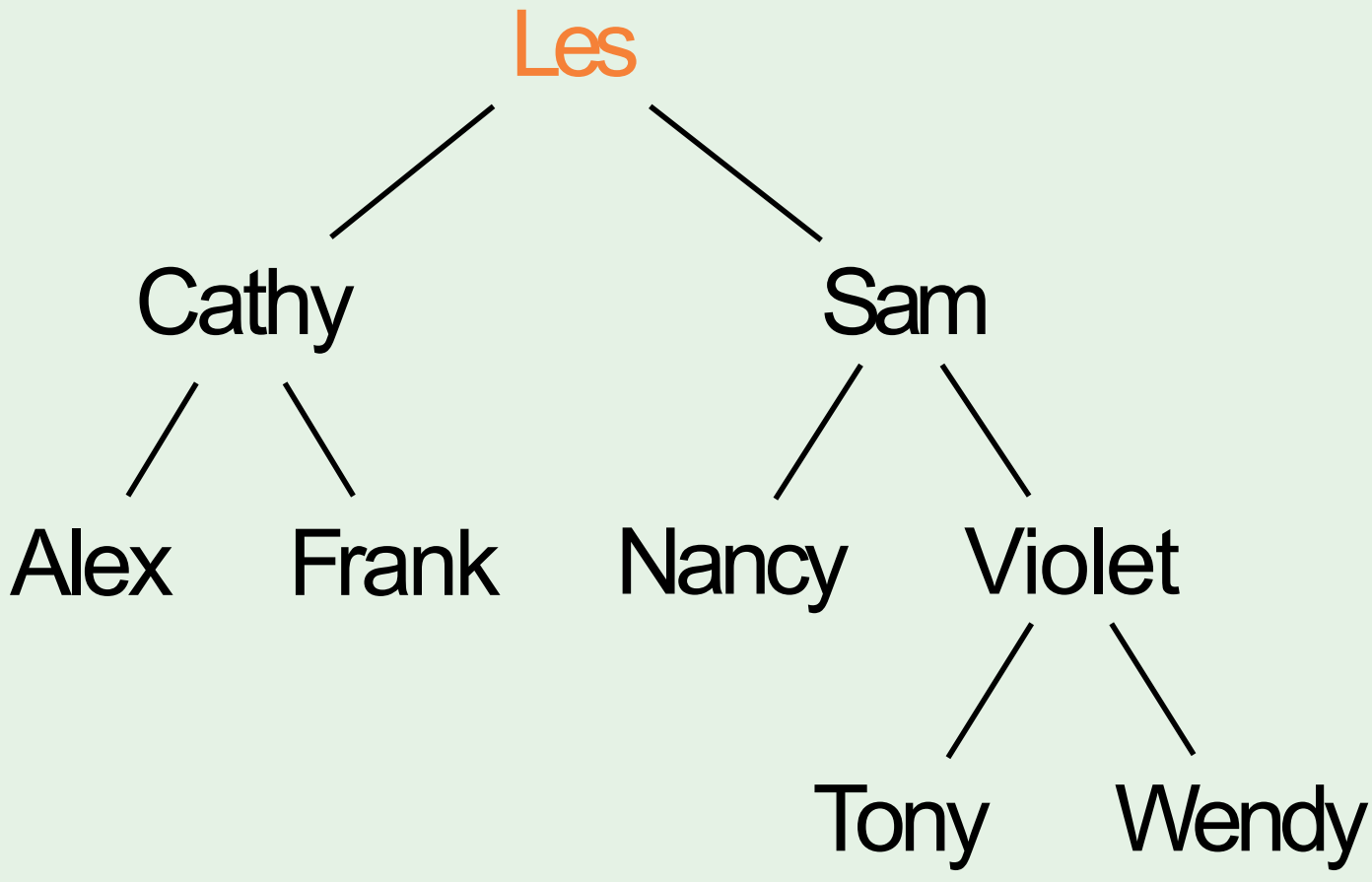
Output:

PreOrderTraversal



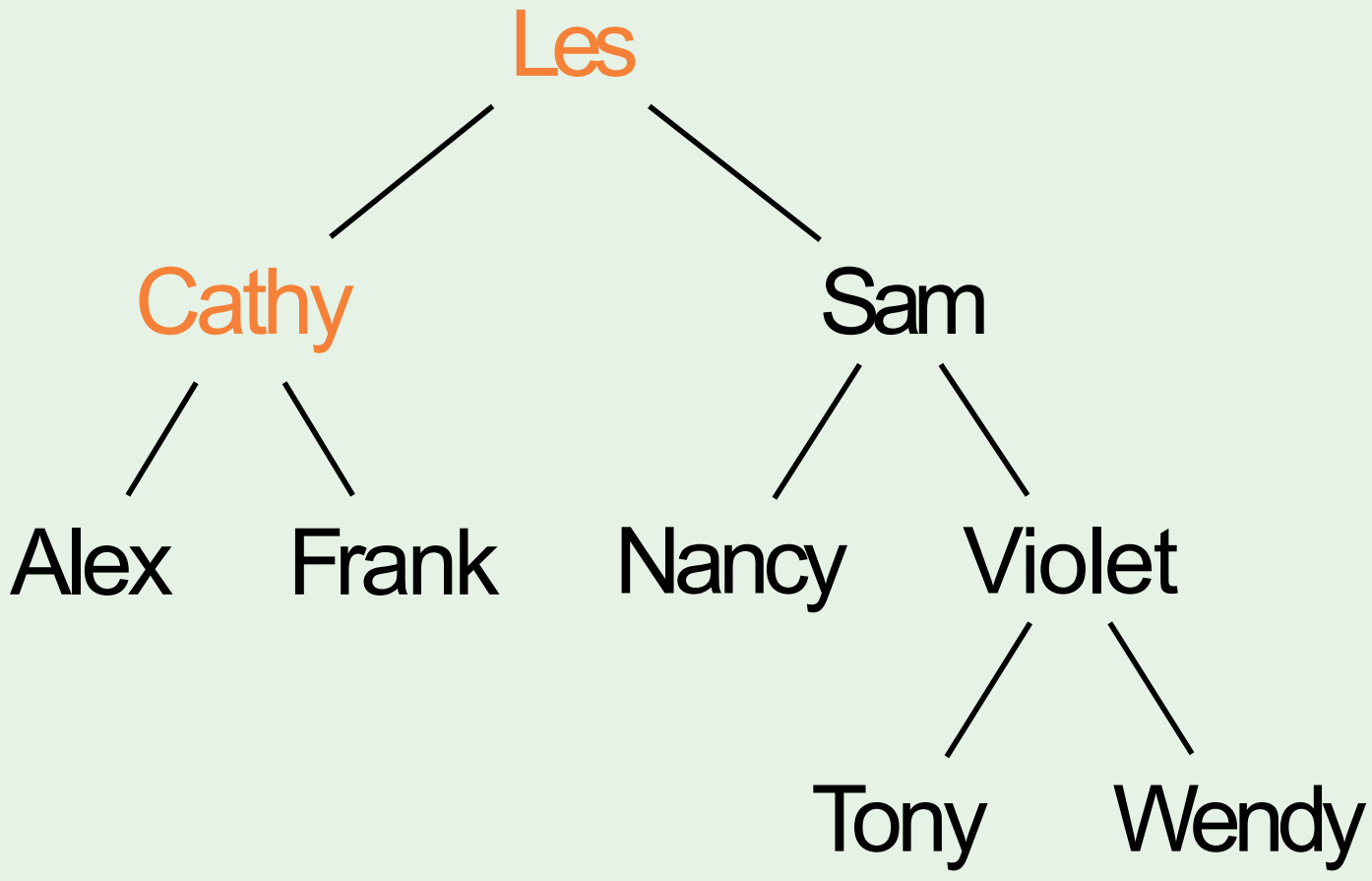
Output: Les

PreOrderTraversal



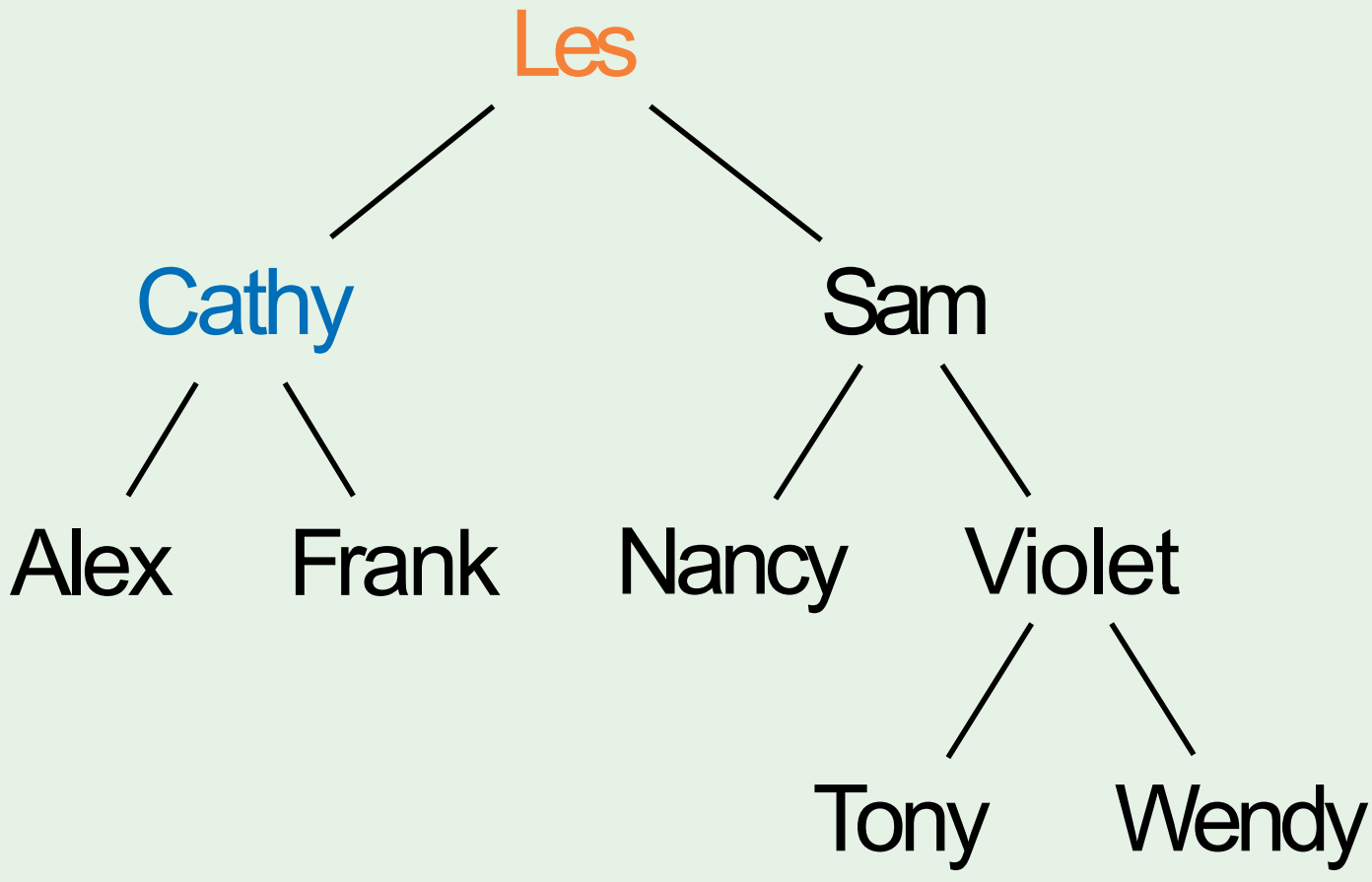
Output: Les

PreOrderTraversal



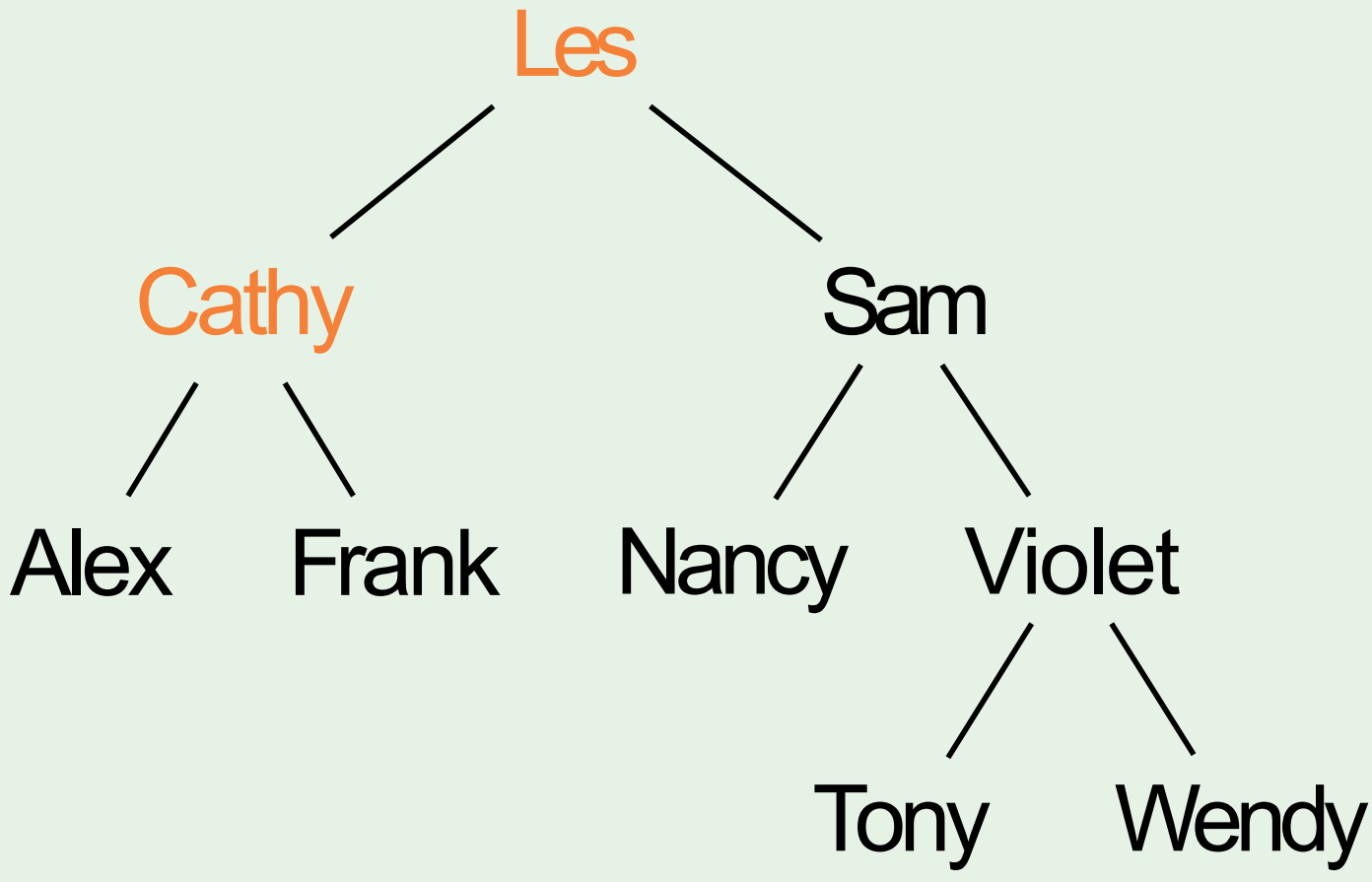
Output: Les

PreOrderTraversal



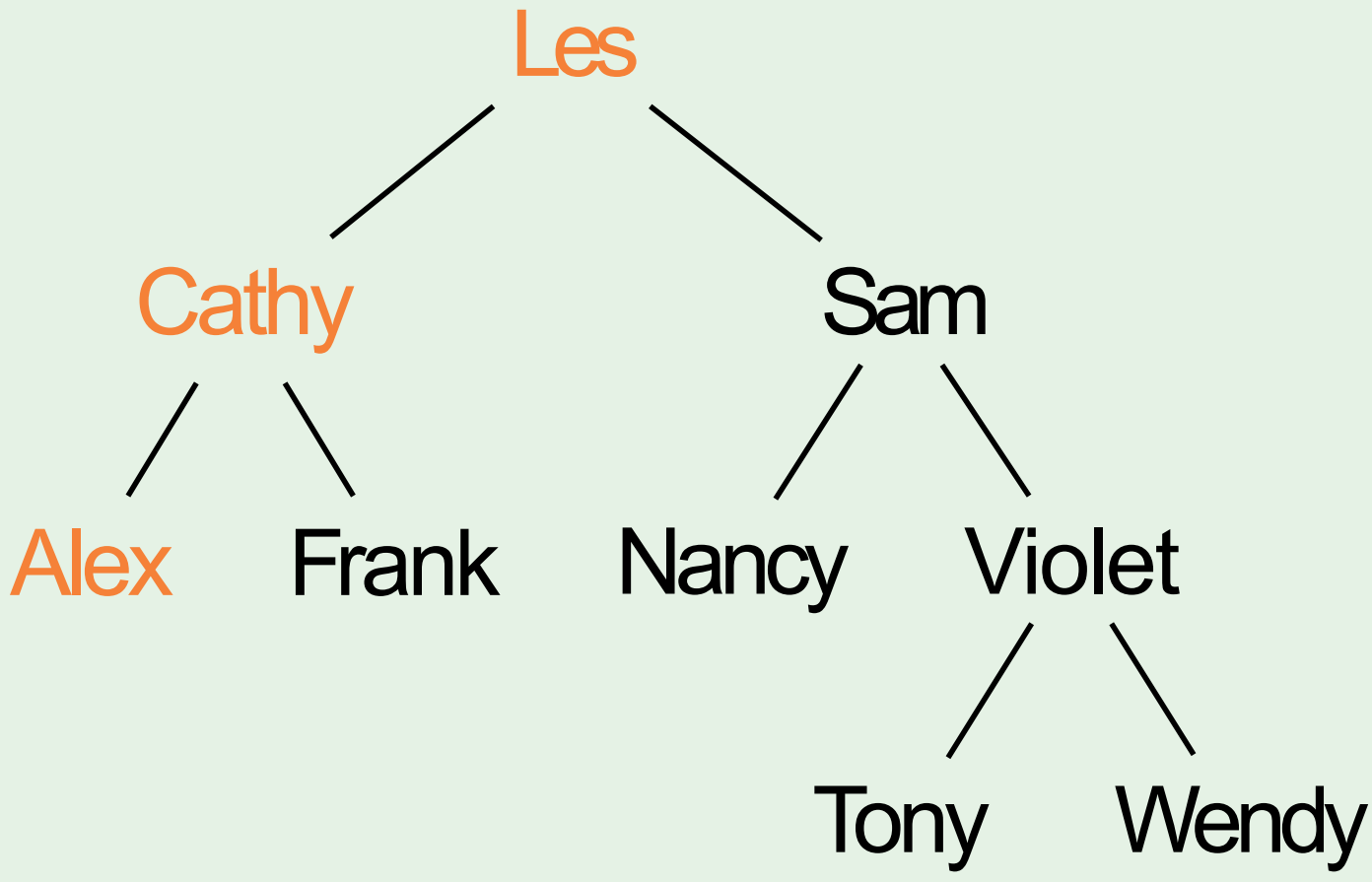
Output: Les Cathy

PreOrderTraversal



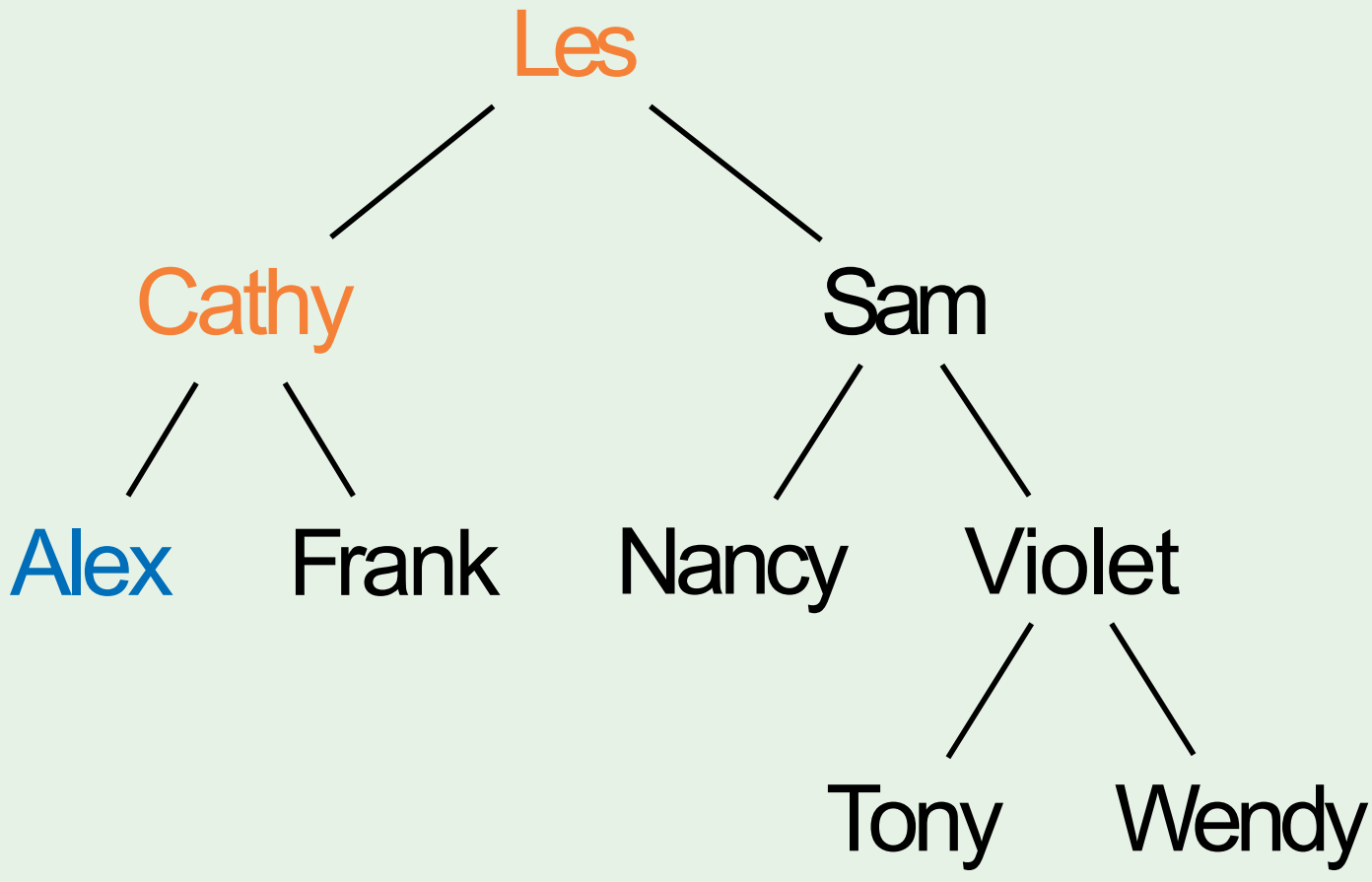
Output: Les Cathy

PreOrderTraversal



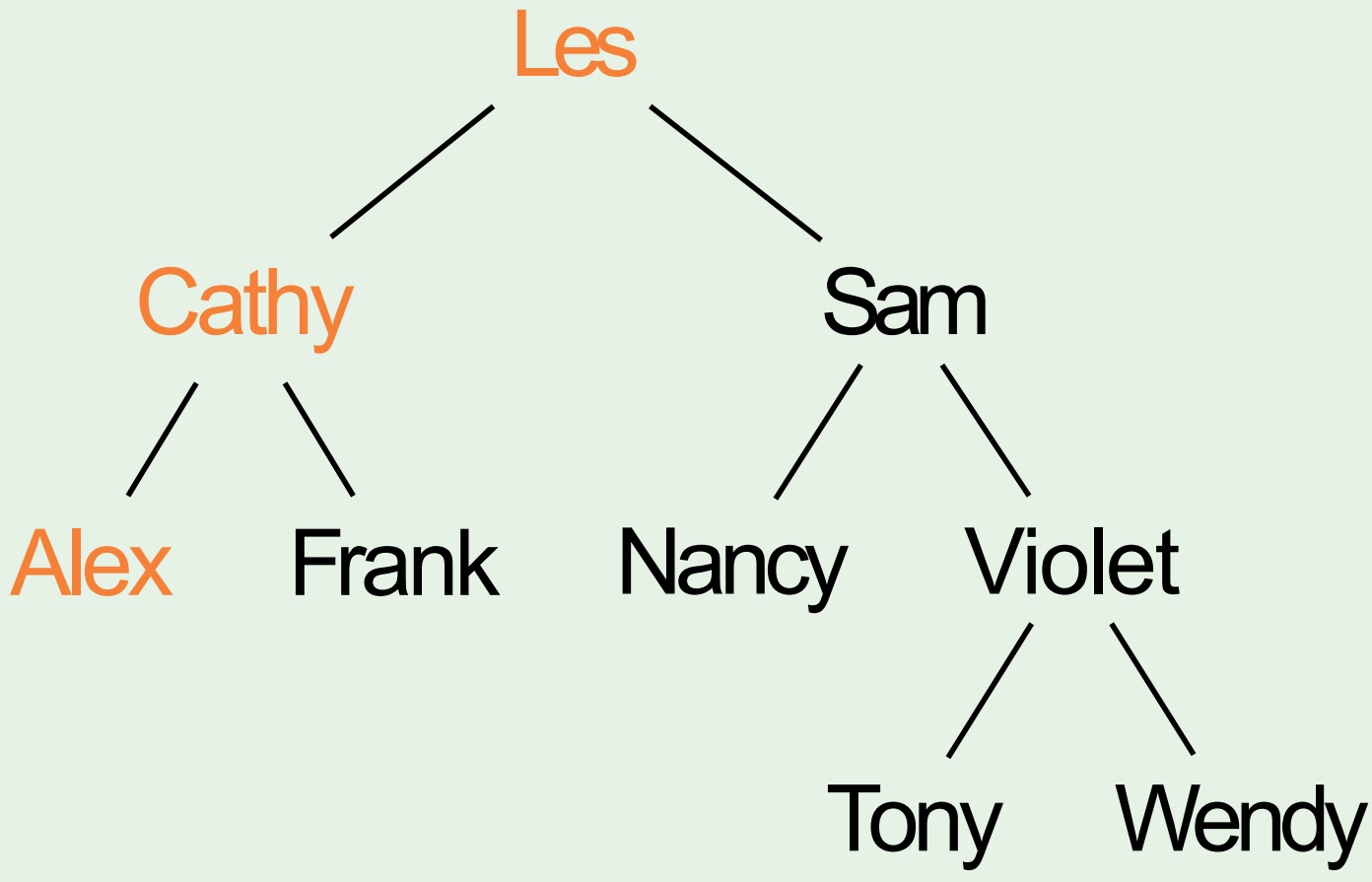
Output: LesCathy

PreOrderTraversal



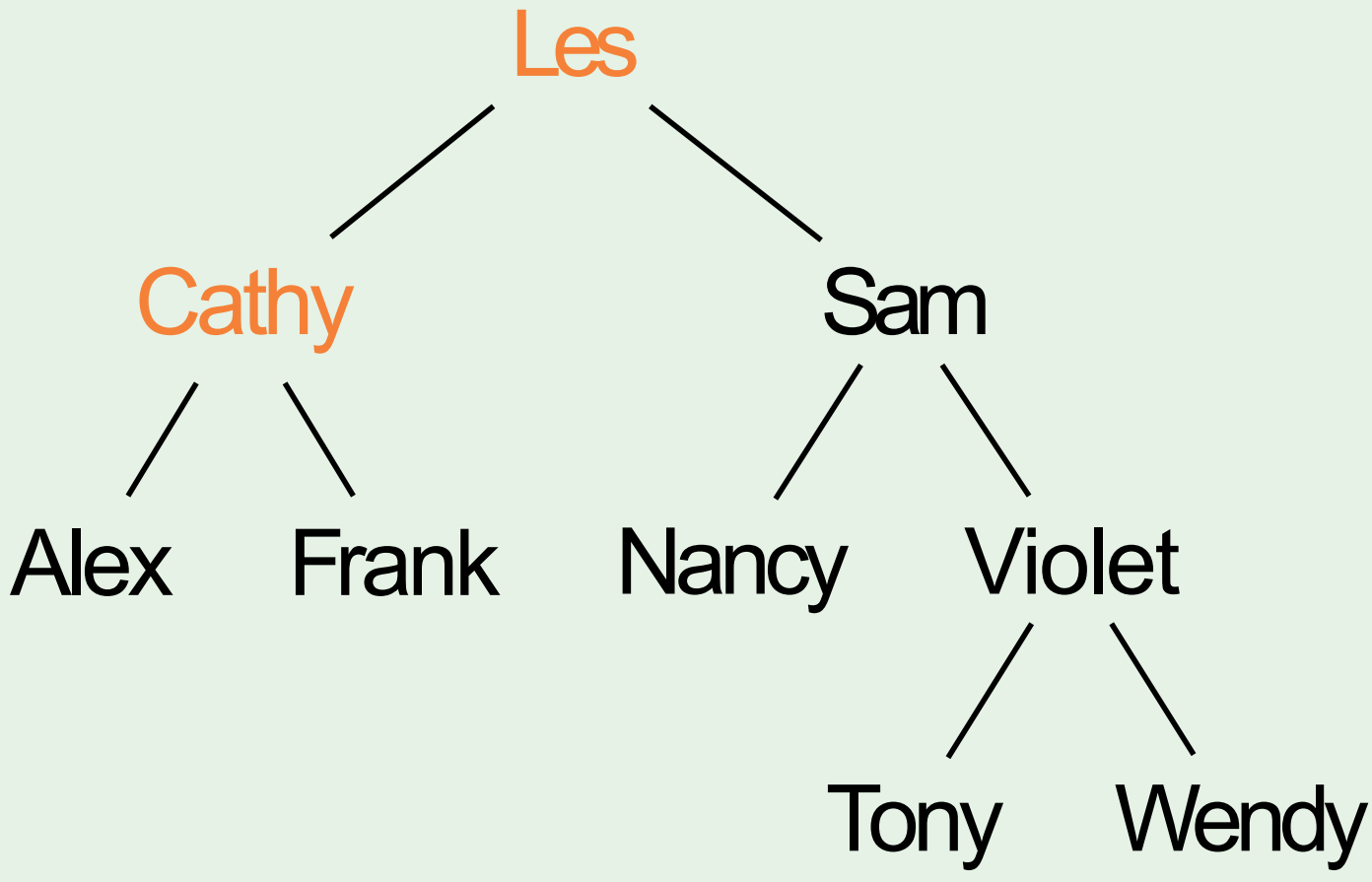
Output: Les Cathy Alex

PreOrderTraversal



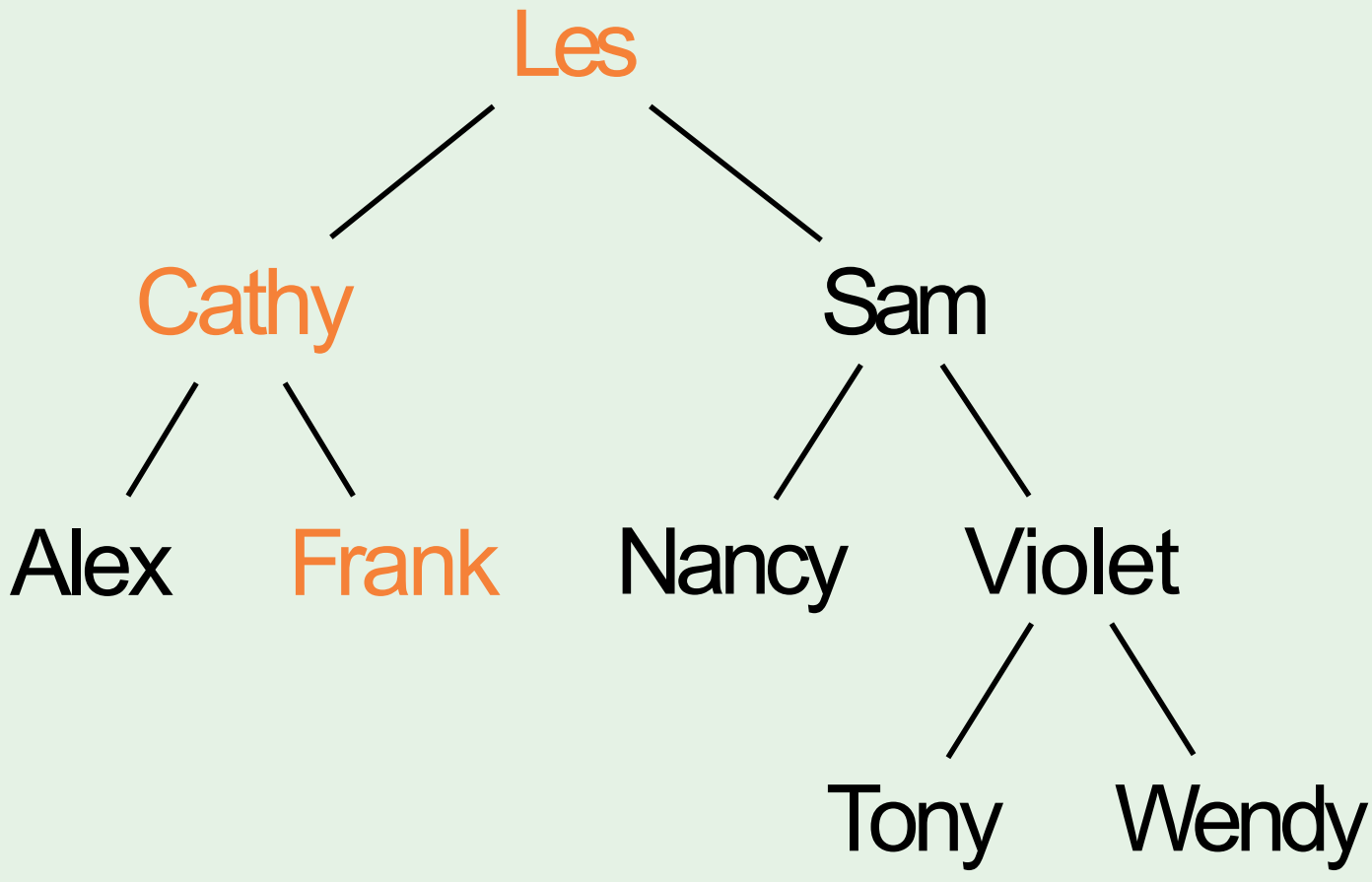
Output: Les Cathy Alex

PreOrderTraversal



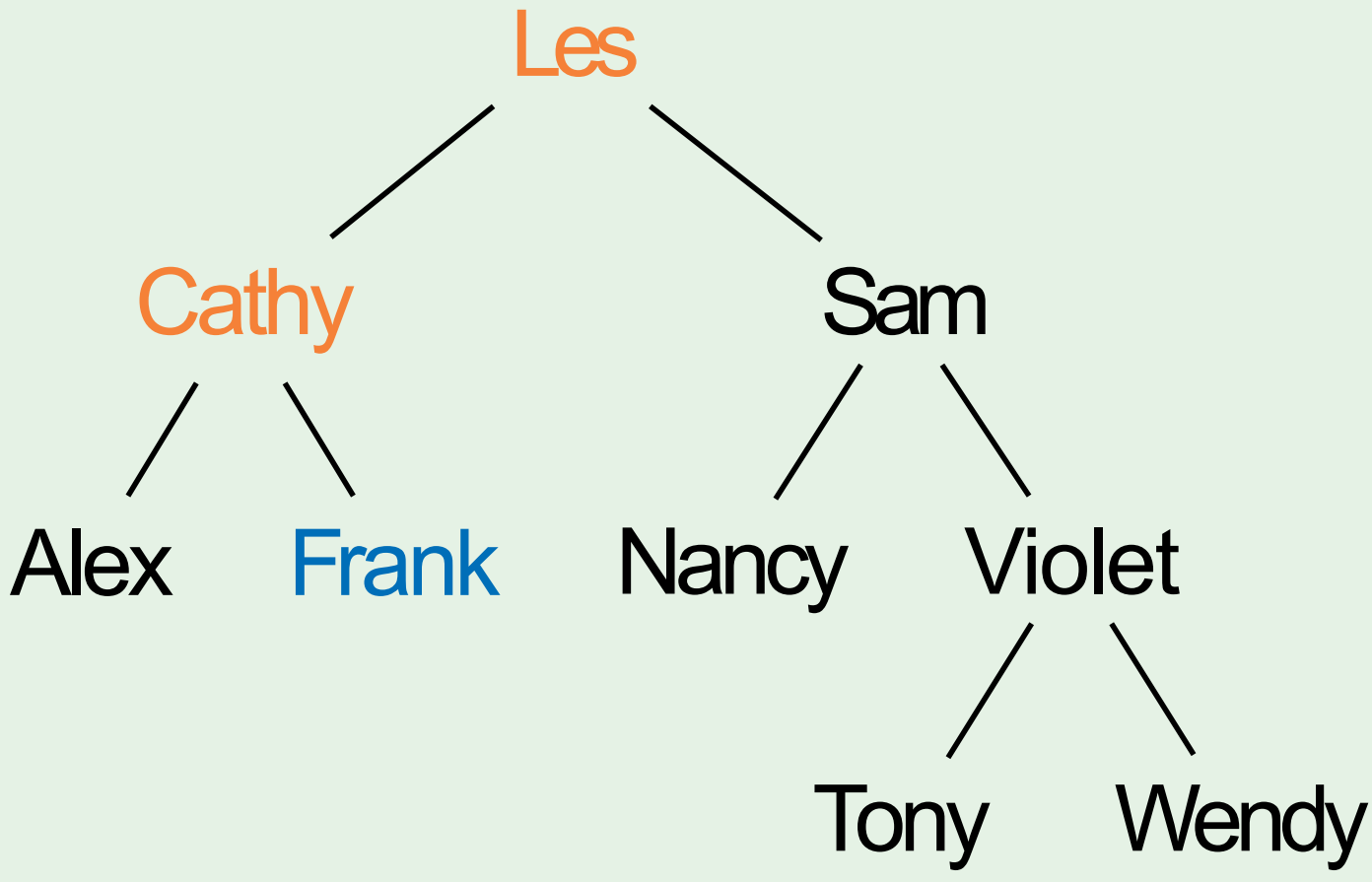
Output: Les Cathy Alex

PreOrderTraversal



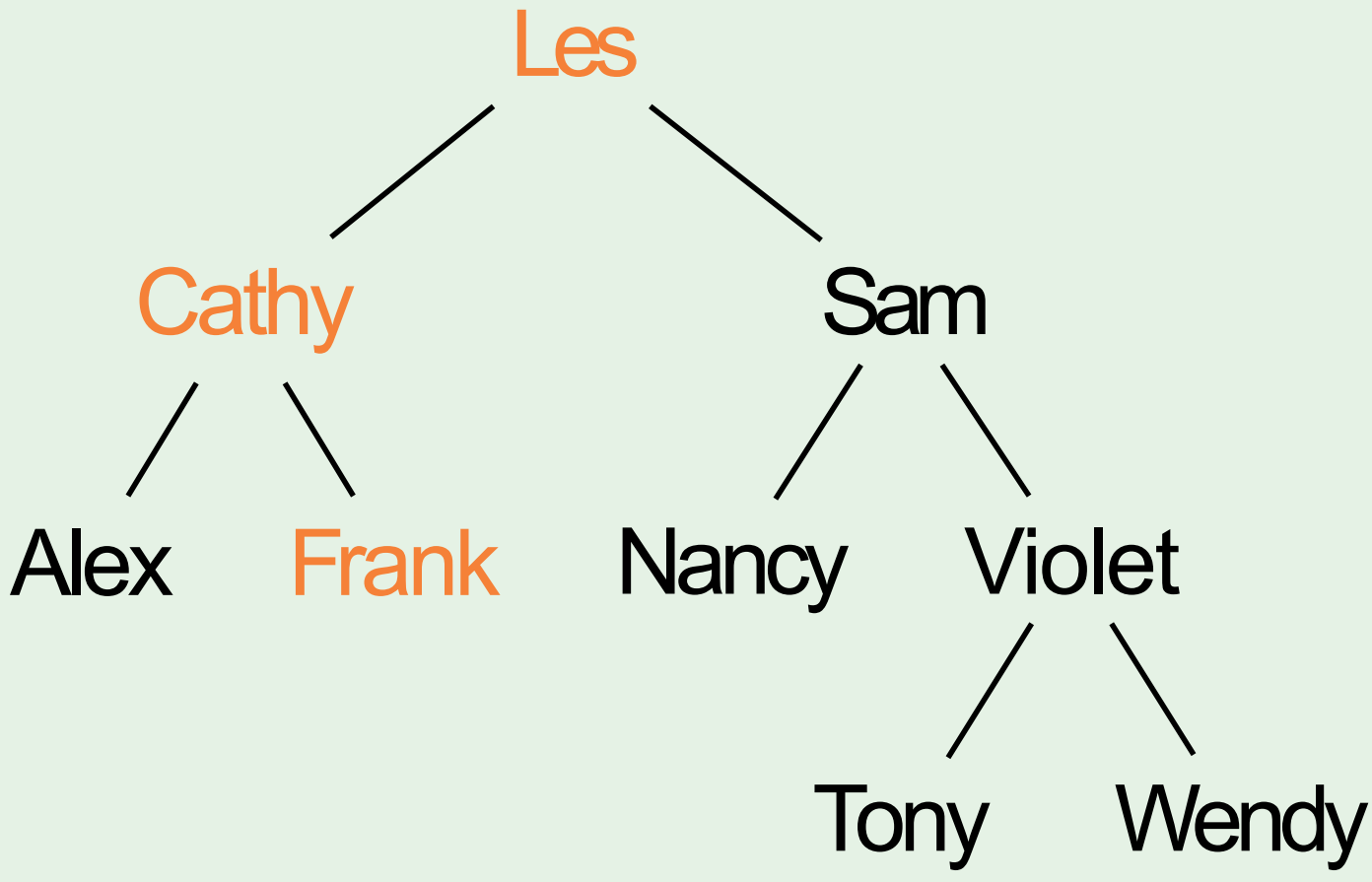
Output: Les Cathy Alex

PreOrderTraversal



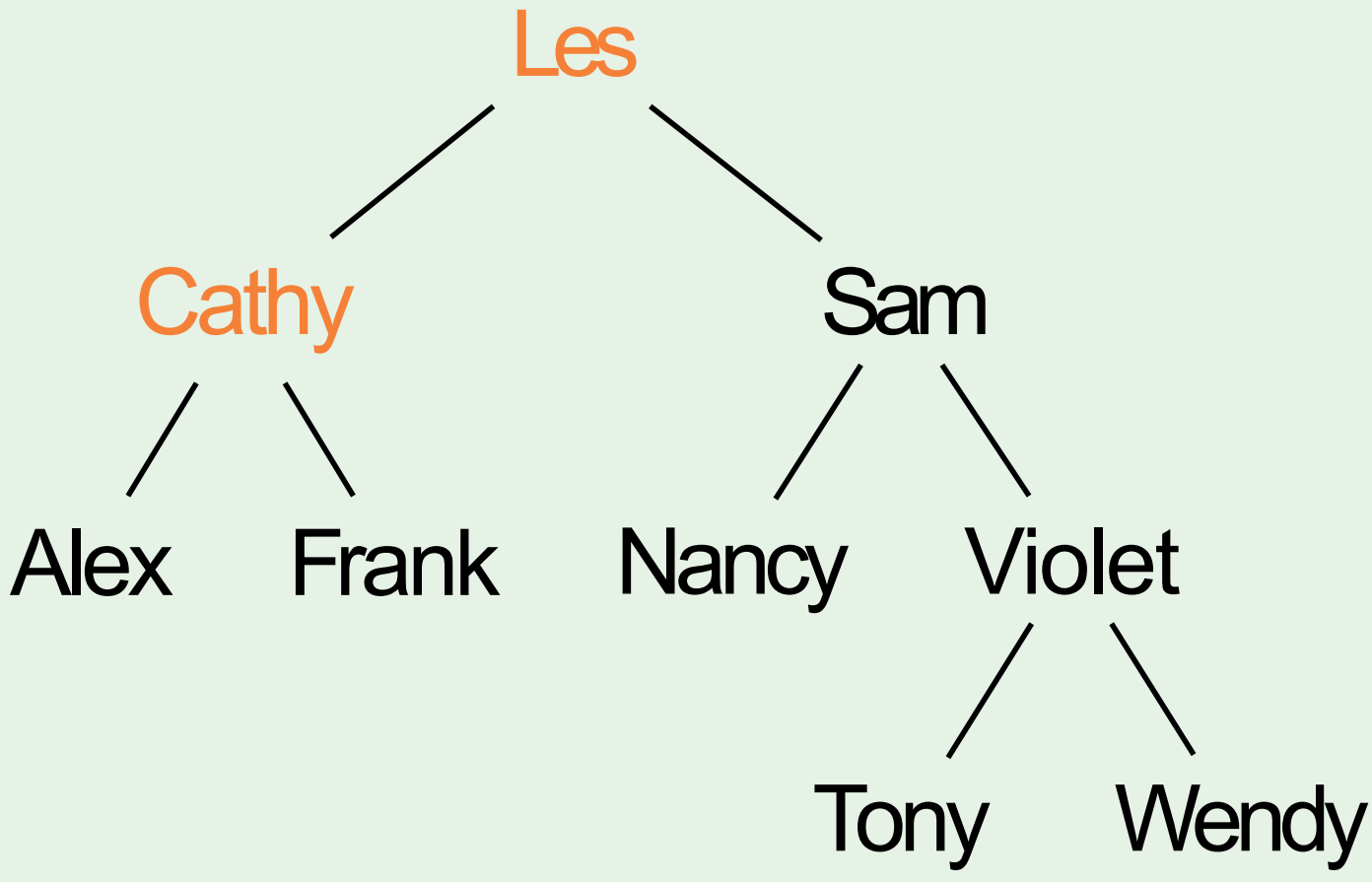
Output: Les Cathy Alex Frank

PreOrderTraversal



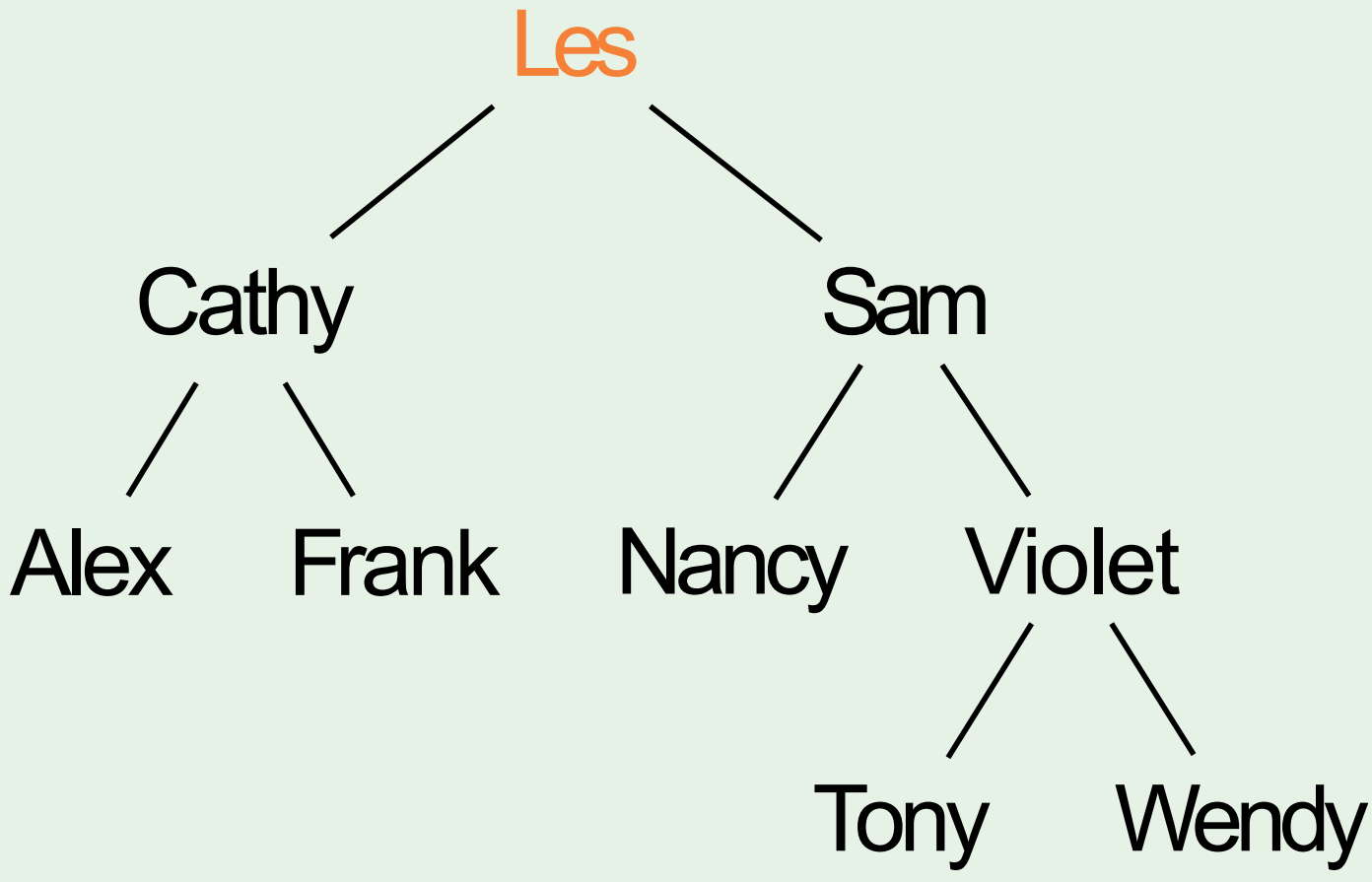
Output: Les Cathy Alex Frank

PreOrderTraversal



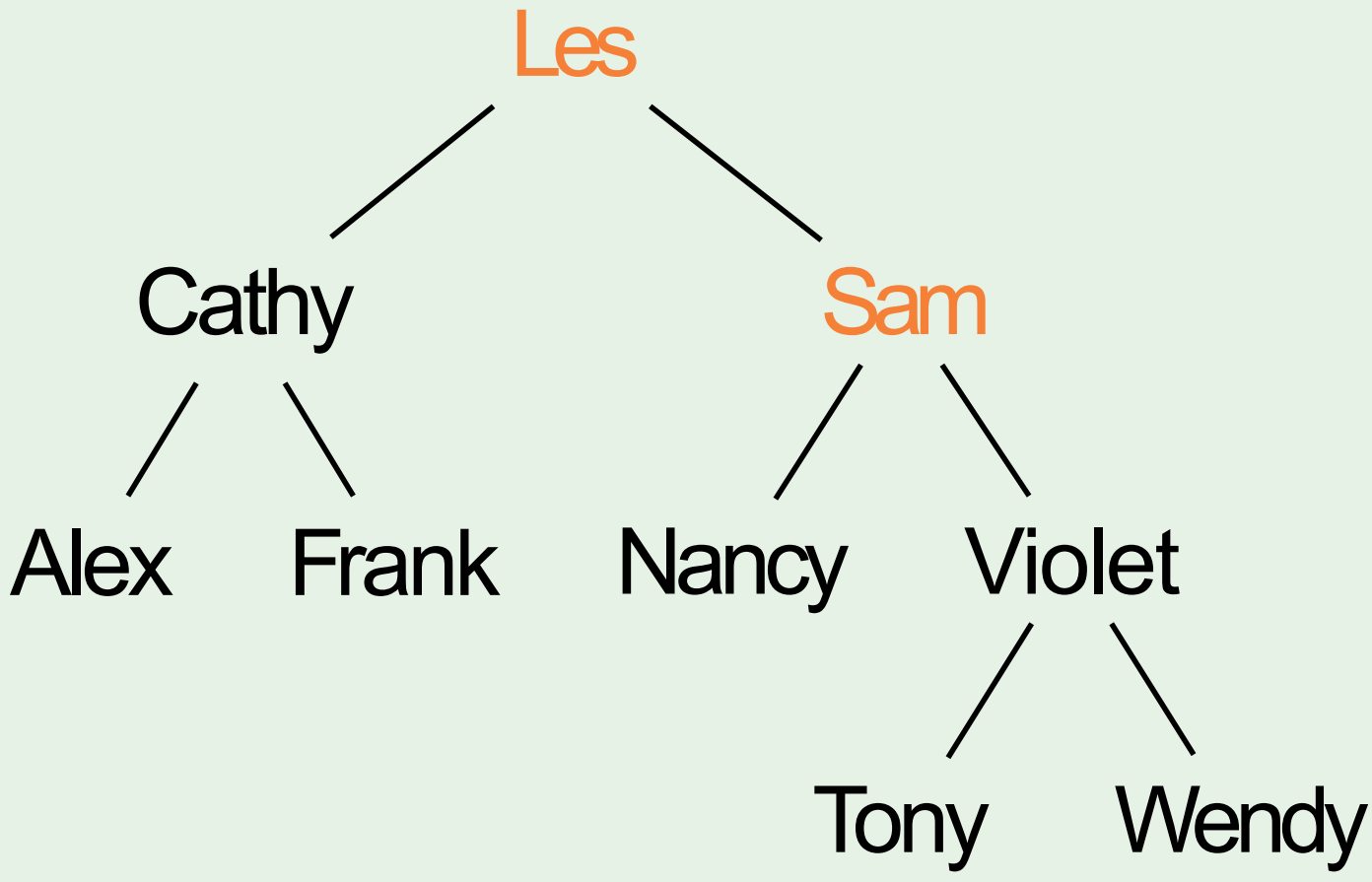
Output: Les Cathy Alex Frank

PreOrderTraversal



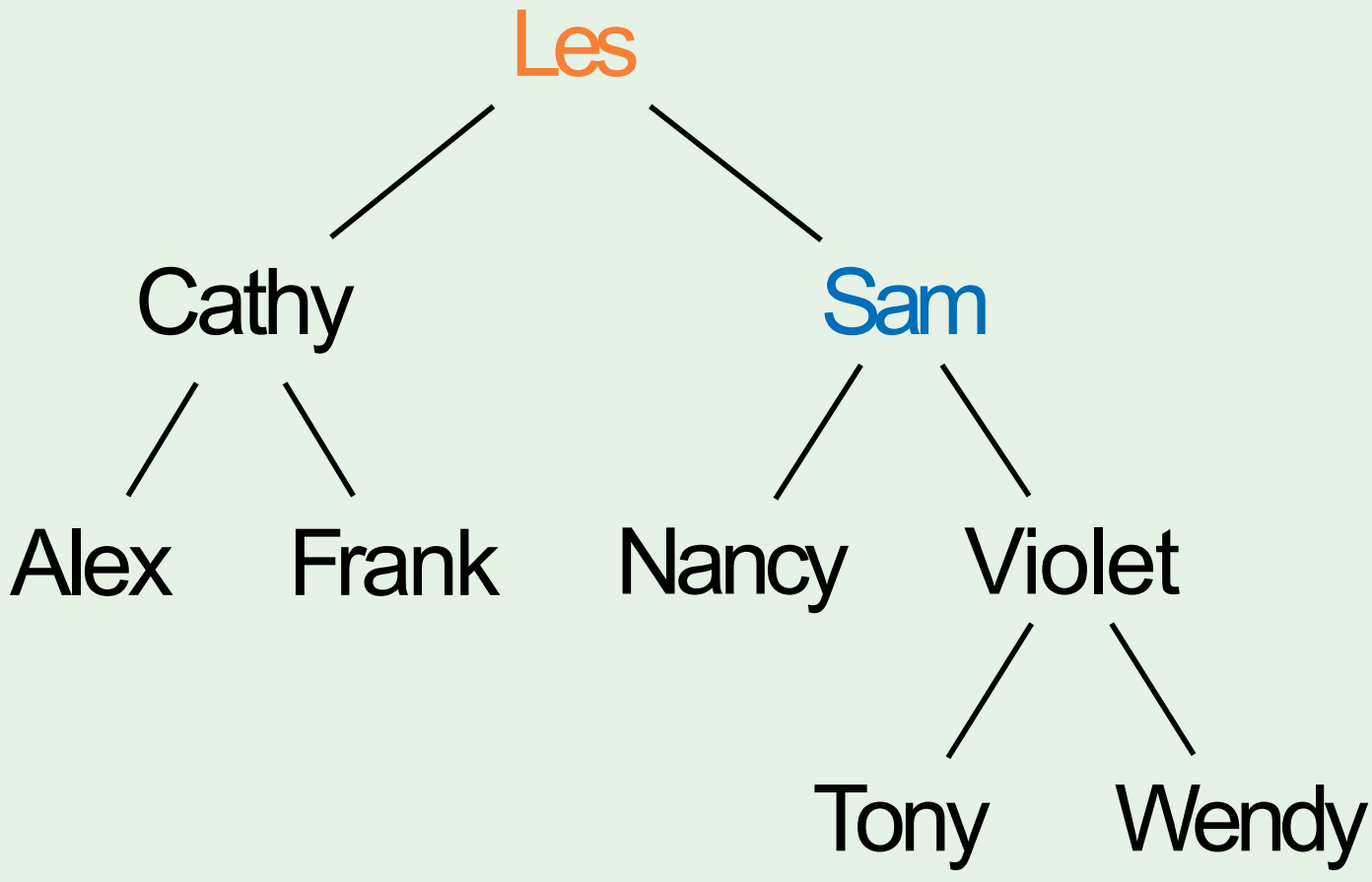
Output: Les Cathy Alex Frank

PreOrderTraversal



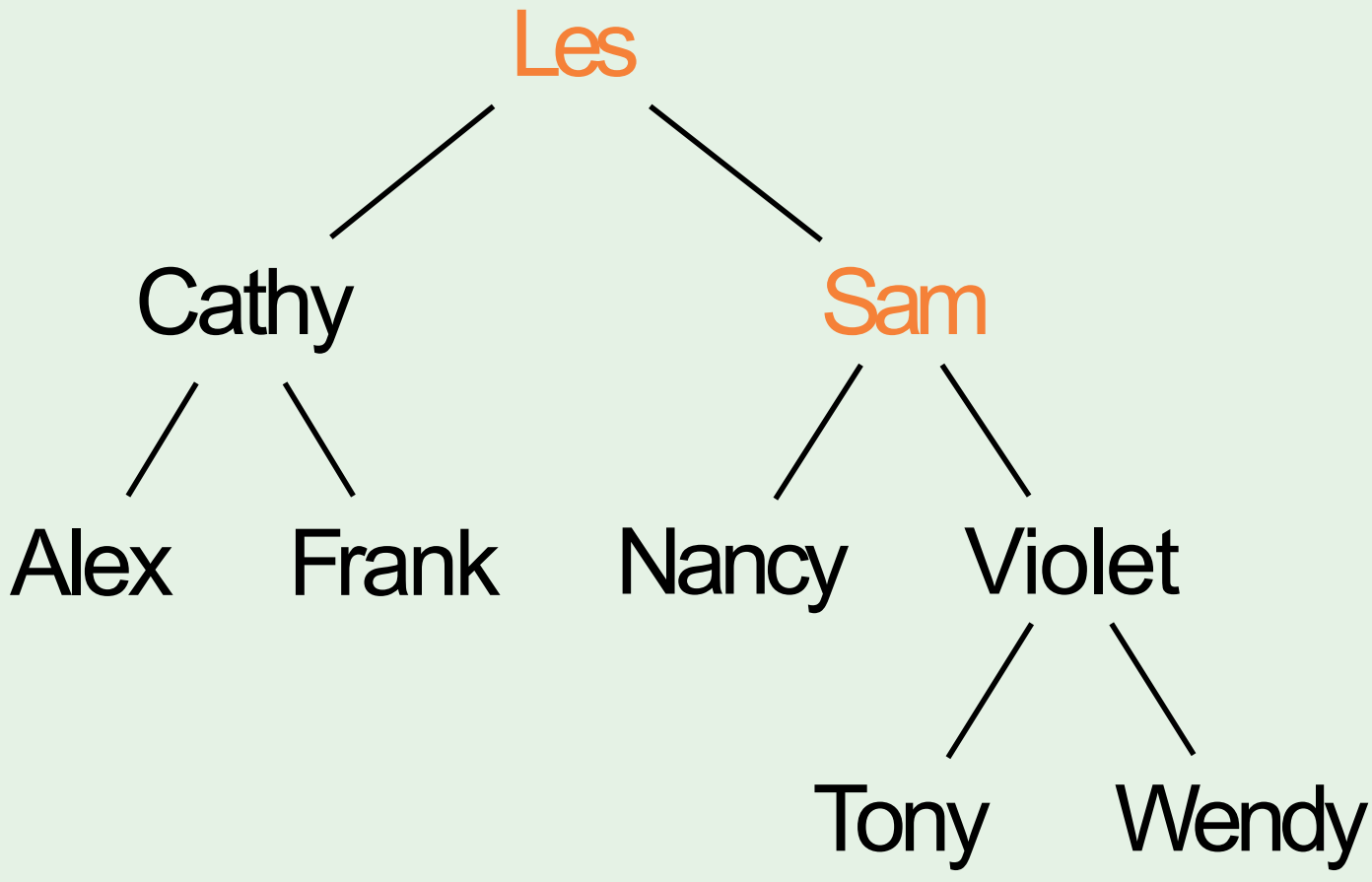
Output: Les Cathy Alex Frank

PreOrderTraversal



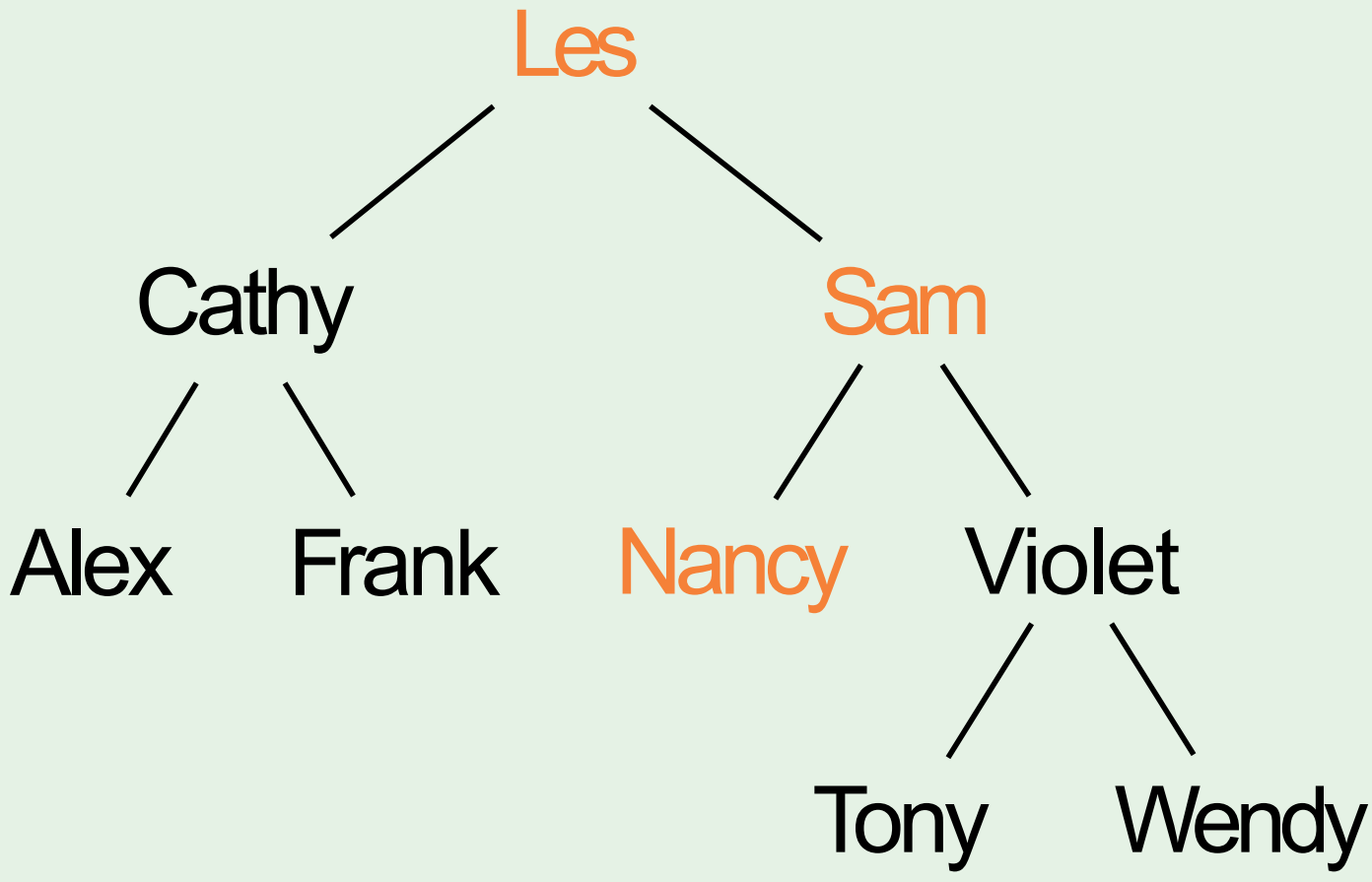
Output: Les Cathy Alex Frank Sam

PreOrderTraversal



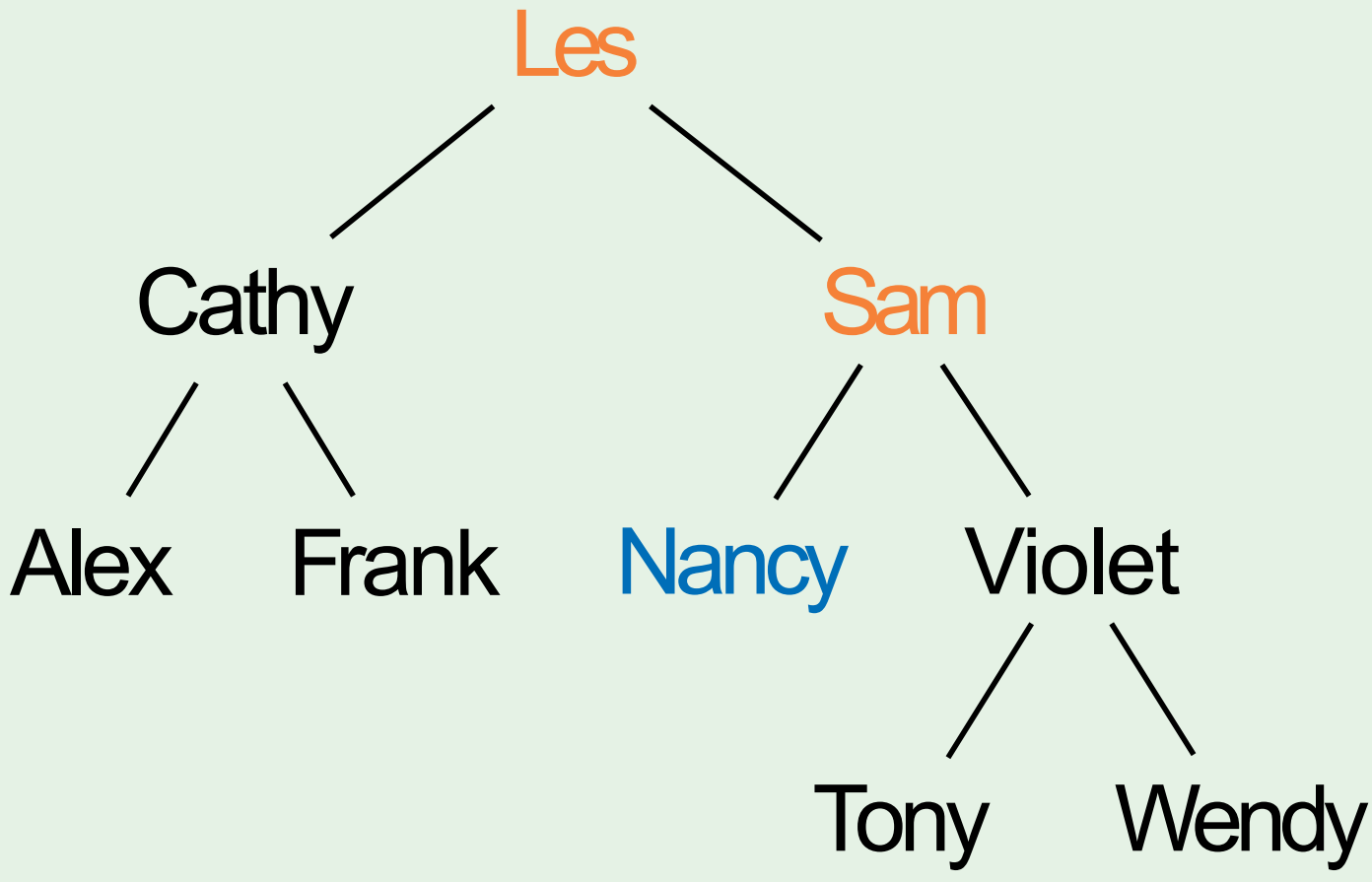
Output: Les Cathy Alex Frank Sam

PreOrderTraversal



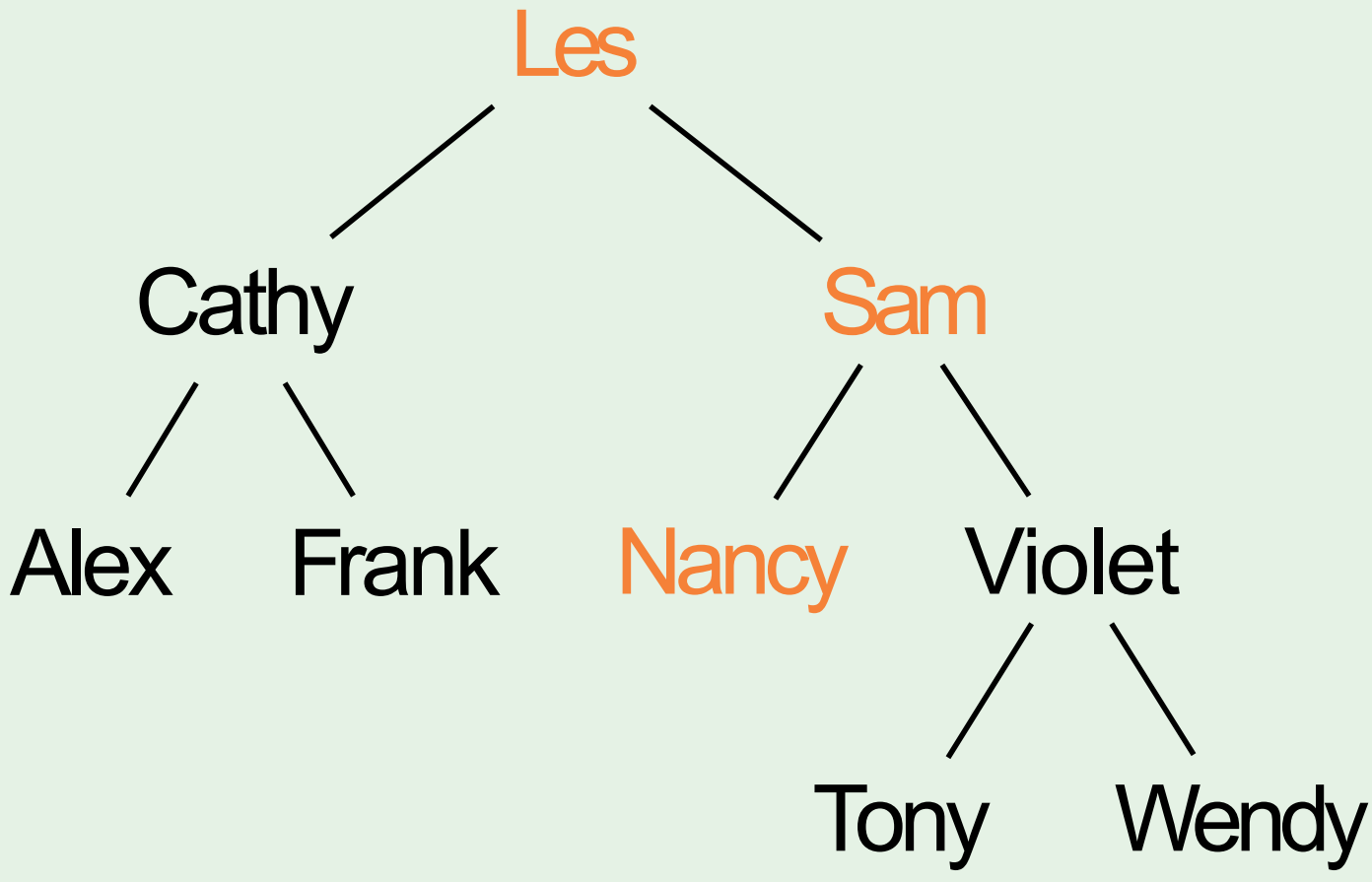
Output: Les Cathy Alex Frank Sam

PreOrderTraversal



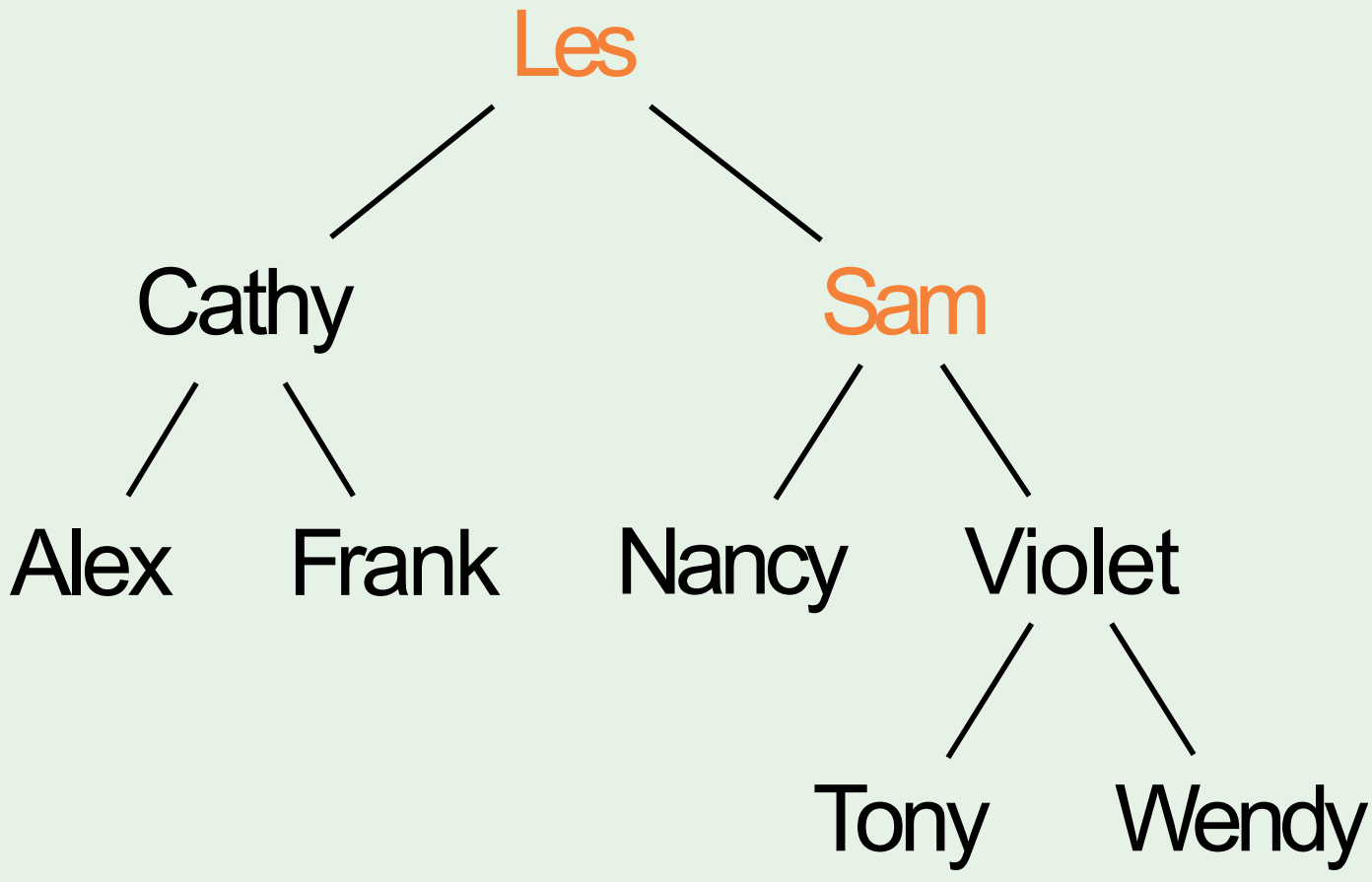
Output: Les Cathy Alex Frank Sam Nancy

PreOrderTraversal



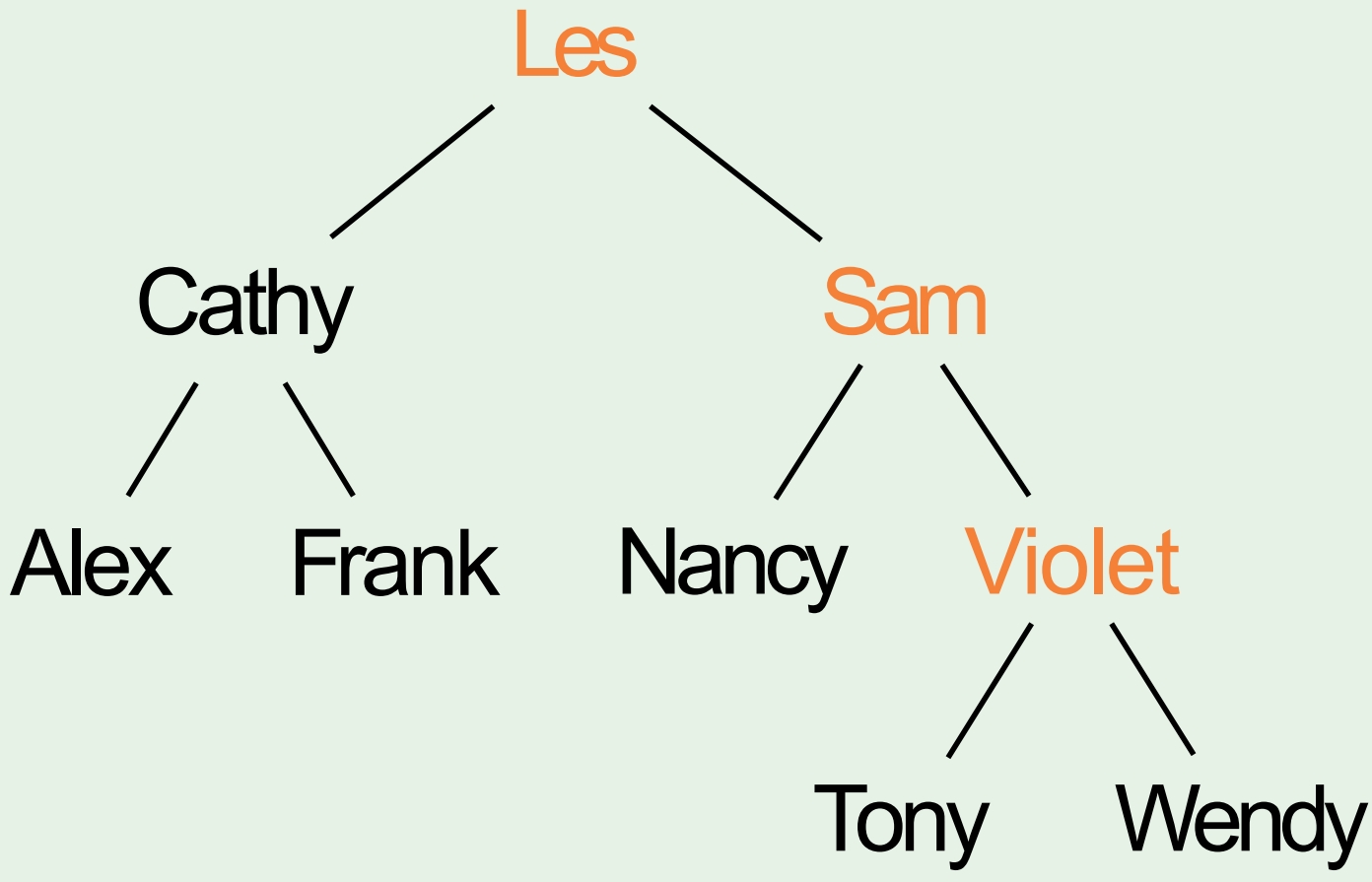
Output: Les Cathy Alex Frank Sam Nancy

PreOrderTraversal



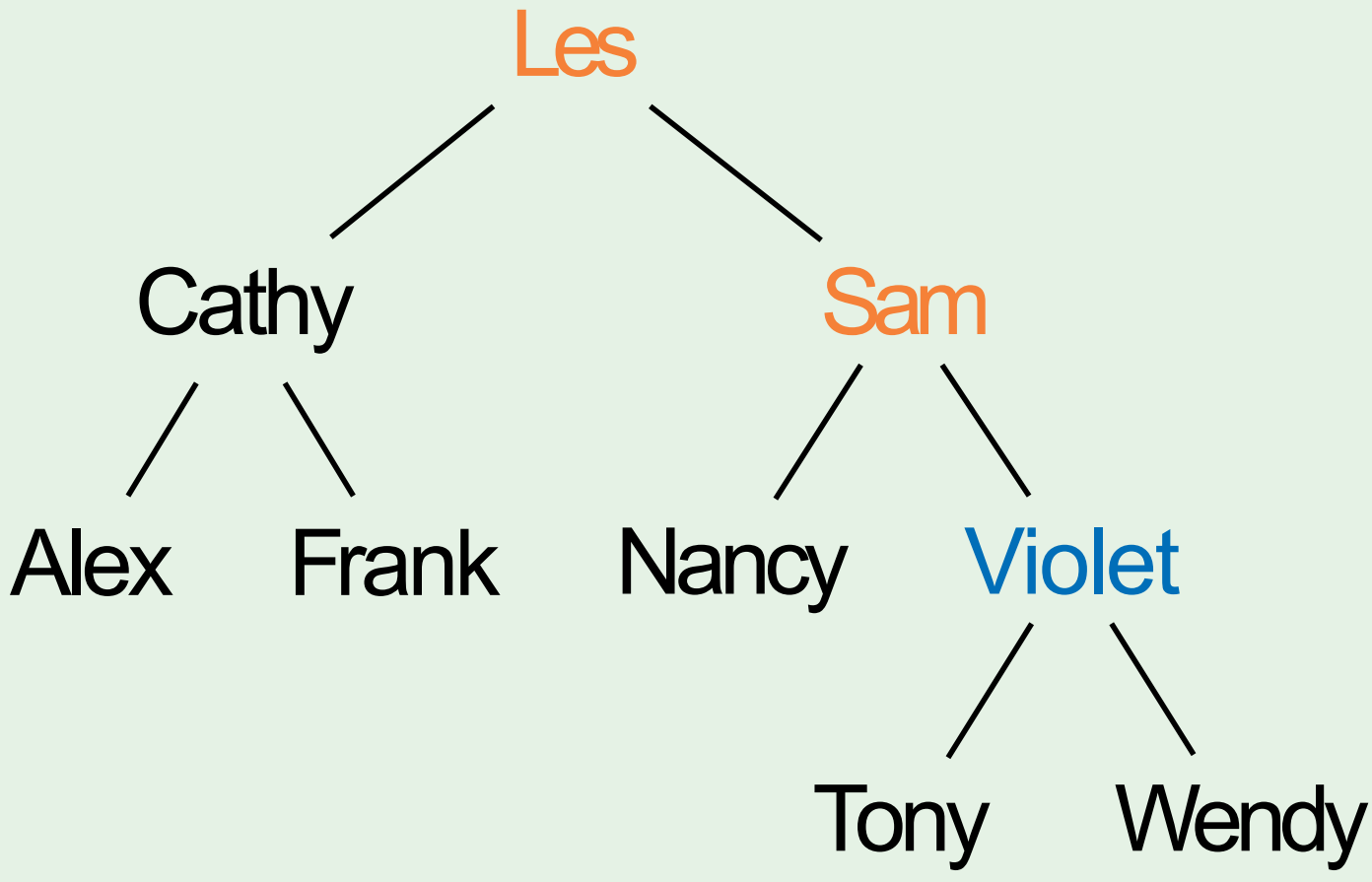
Output: Les Cathy Alex Frank Sam Nancy

PreOrderTraversal



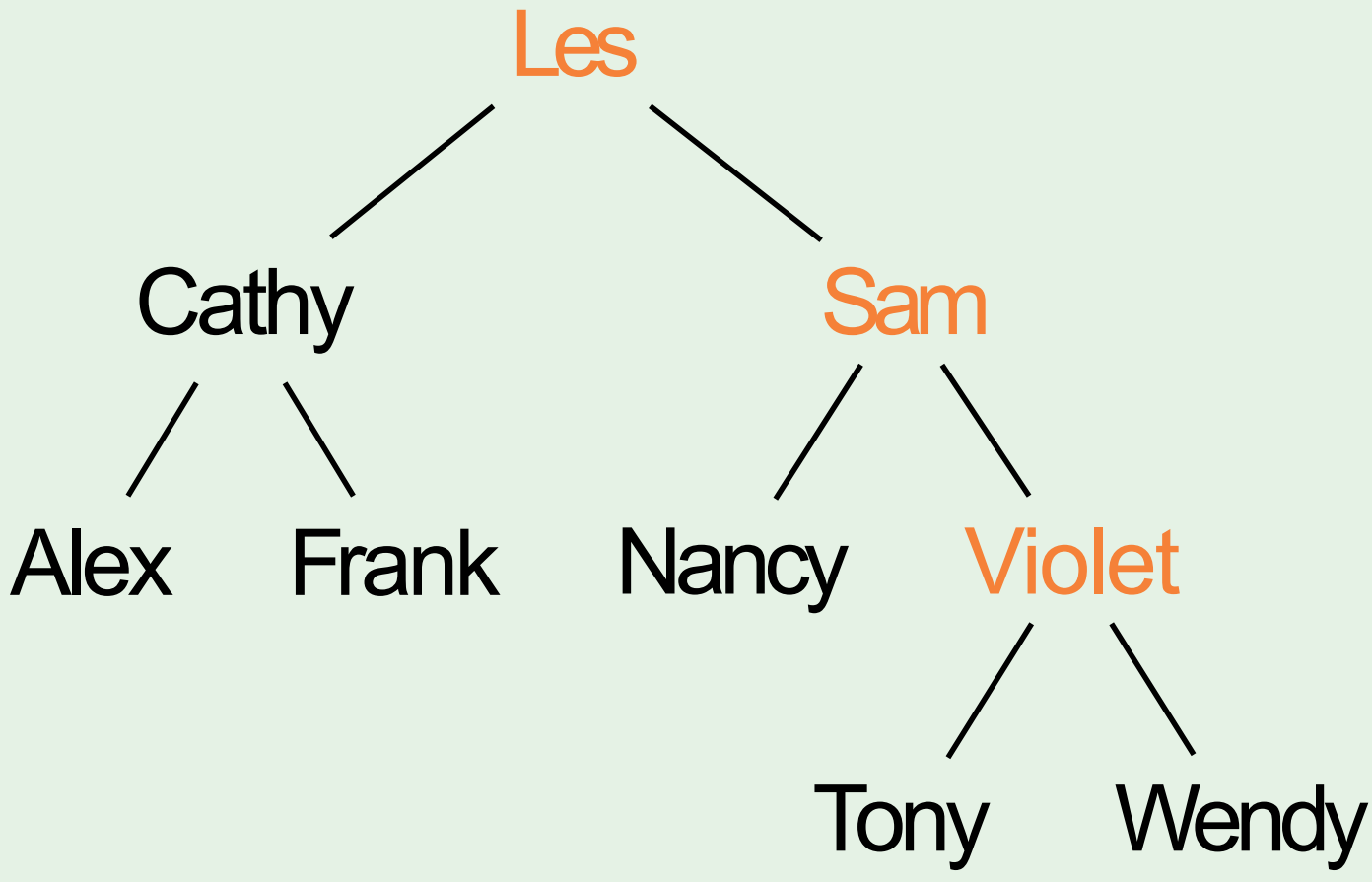
Output: Les Cathy Alex Frank Sam Nancy

PreOrderTraversal



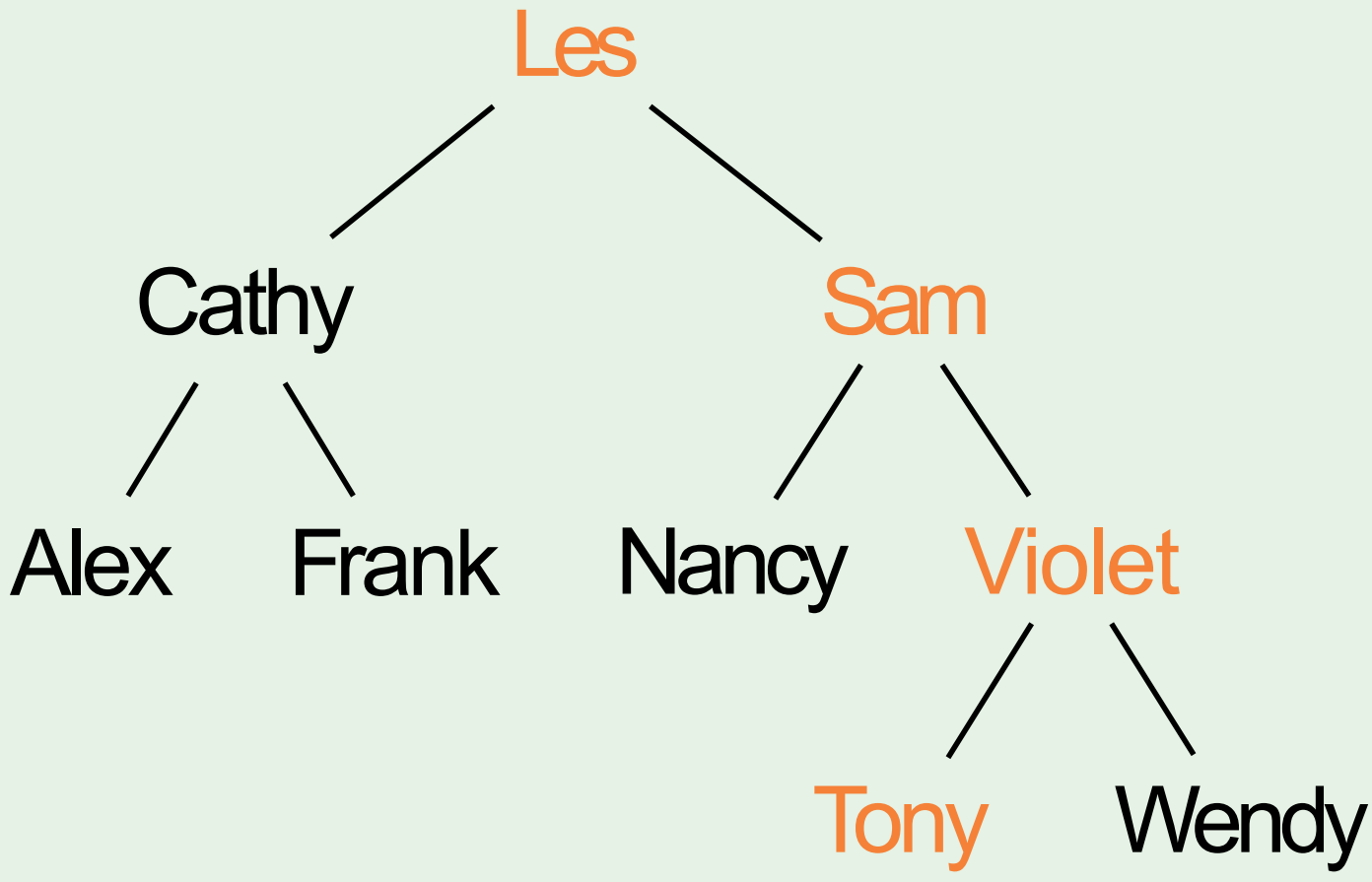
Output: Les Cathy Alex Frank Sam Nancy Violet

PreOrderTraversal



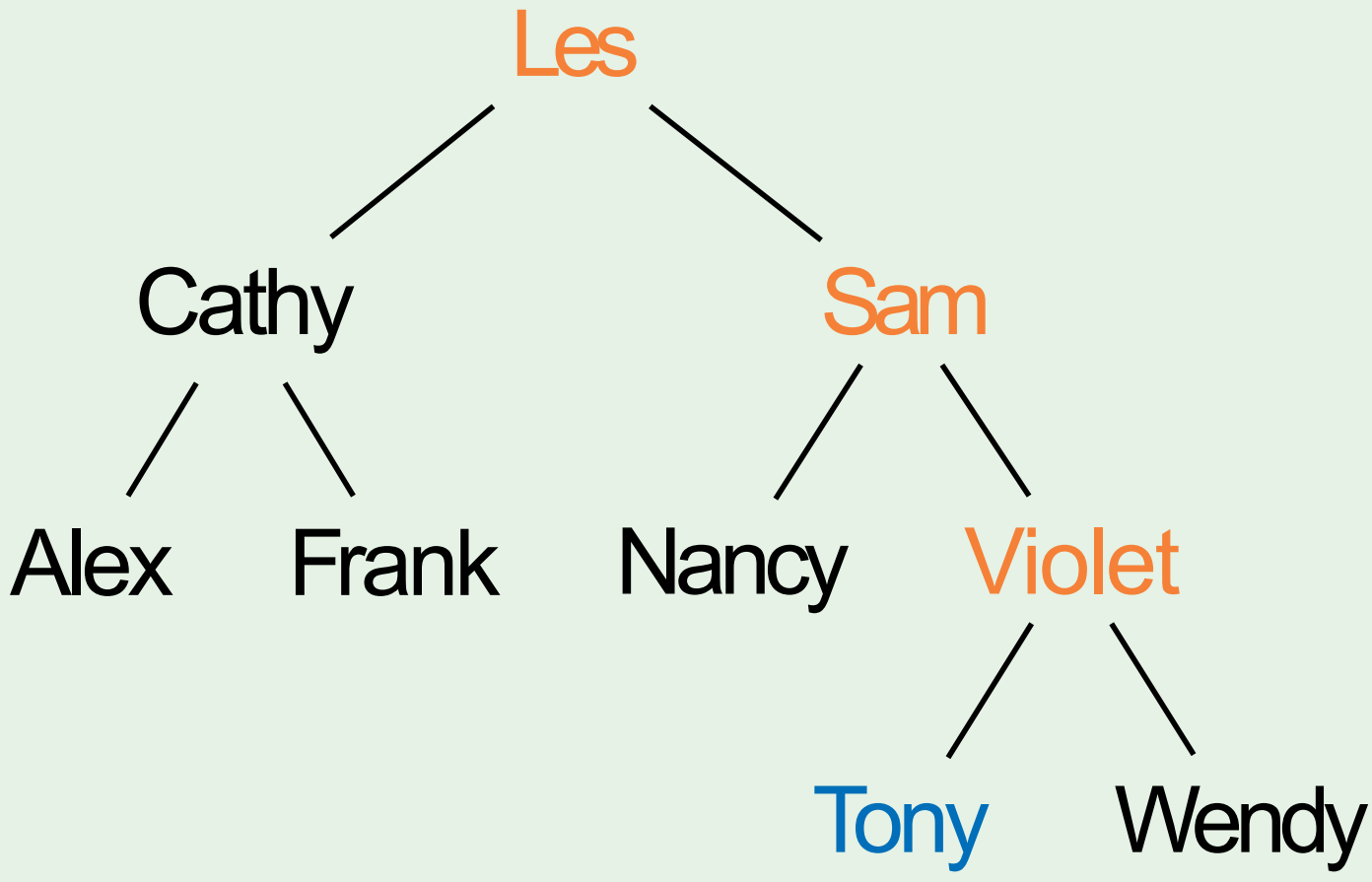
Output: Les Cathy Alex Frank Sam Nancy Violet

PreOrderTraversal



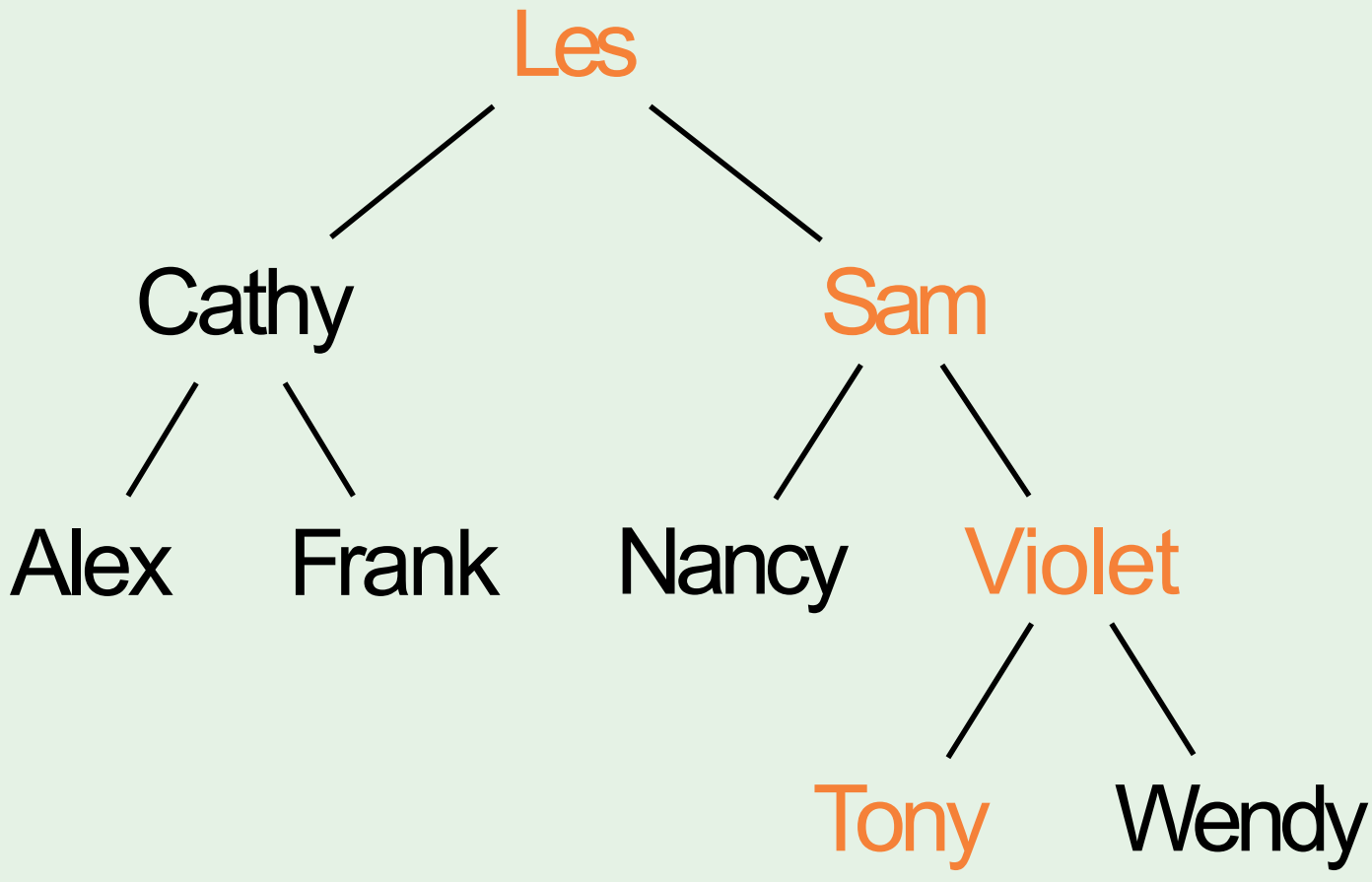
Output: Les Cathy Alex Frank Sam Nancy Violet

PreOrderTraversal



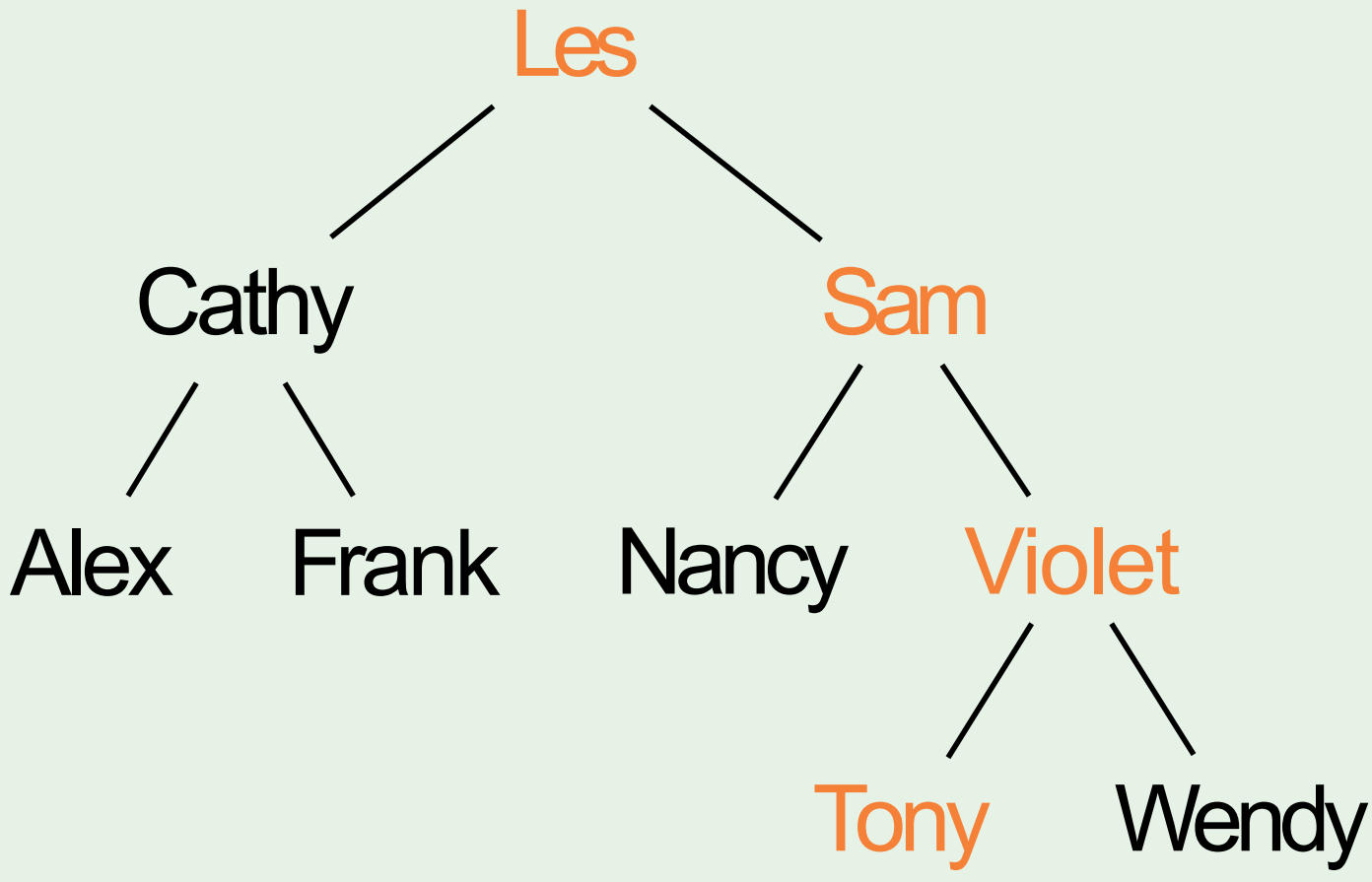
Output: Les Cathy Alex Frank Sam Nancy Violet Tony

PreOrderTraversal



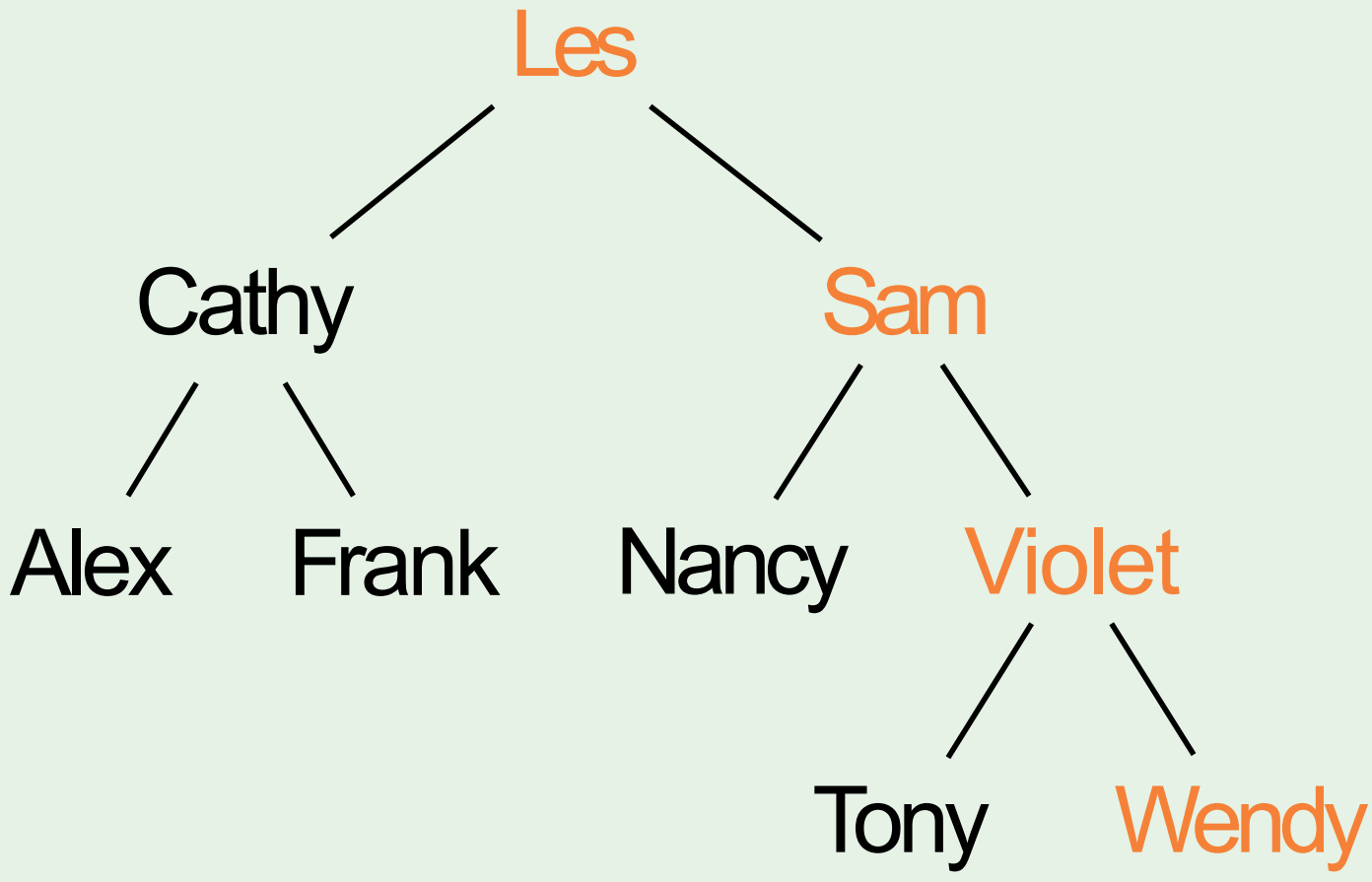
Output: Les Cathy Alex Frank Sam Nancy Violet Tony

PreOrderTraversal



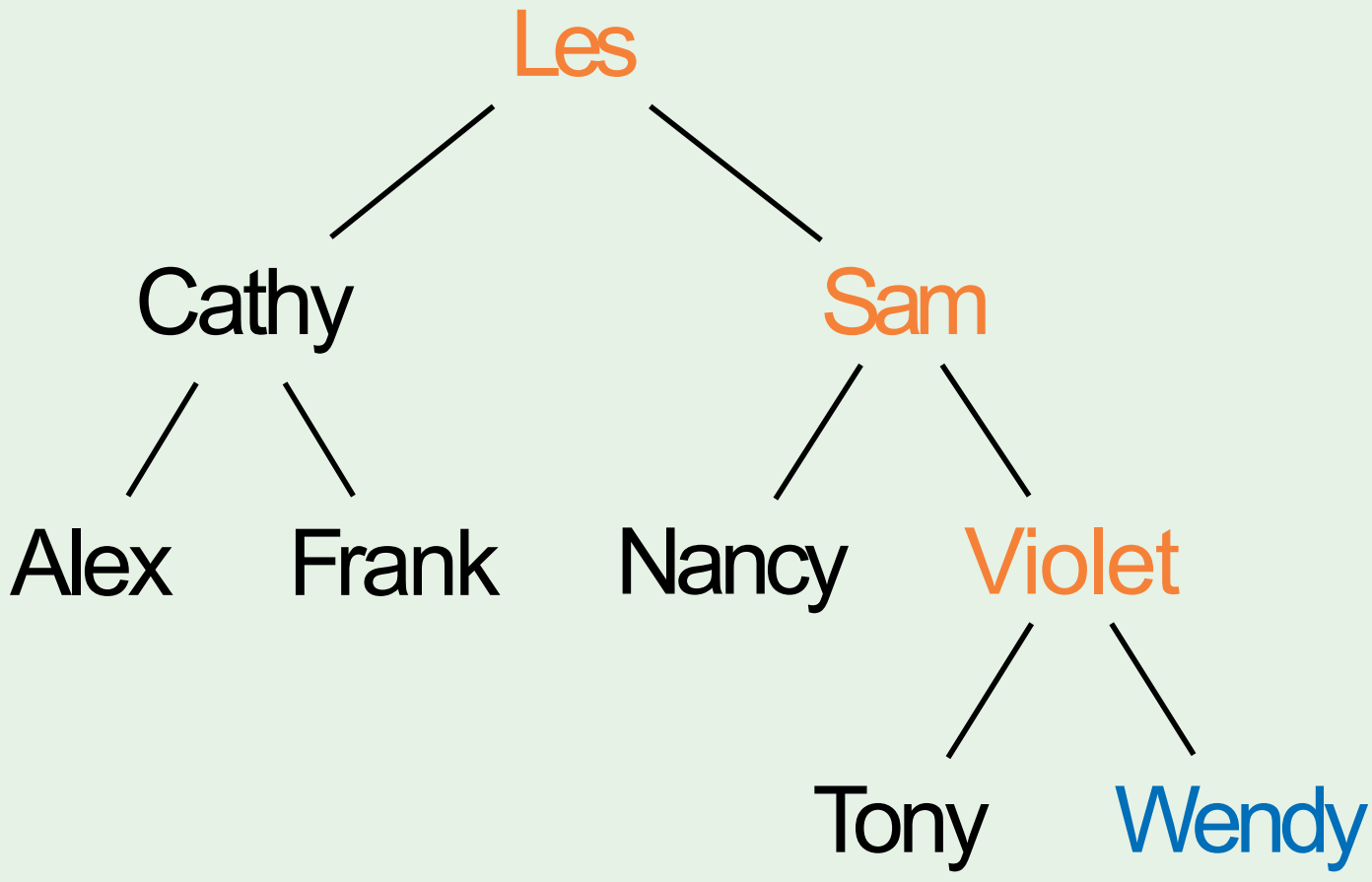
Output: Les Cathy Alex Frank Sam Nancy Violet Tony

PreOrderTraversal



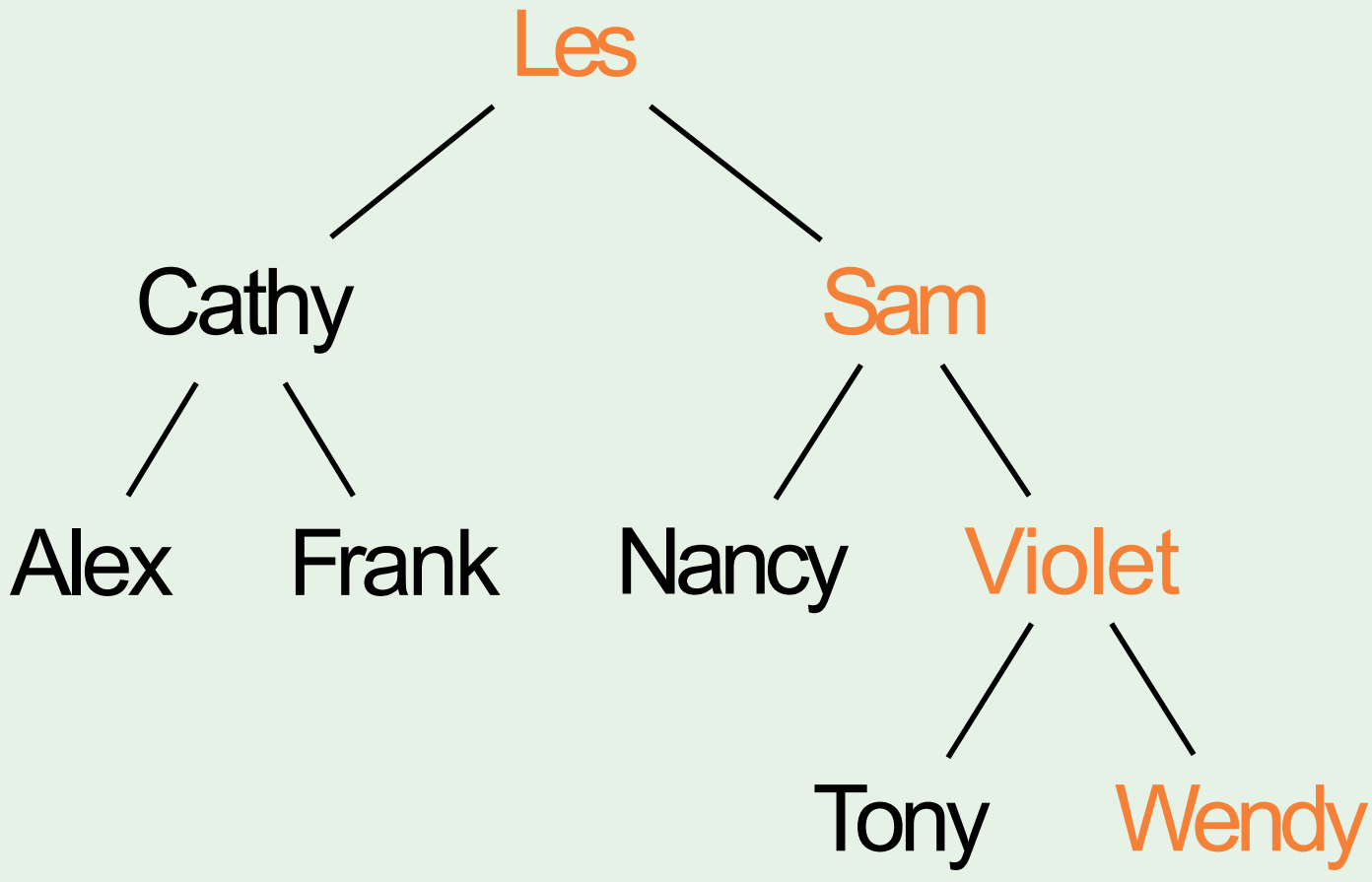
Output: Les Cathy Alex Frank Sam Nancy Violet Tony

PreOrderTraversal



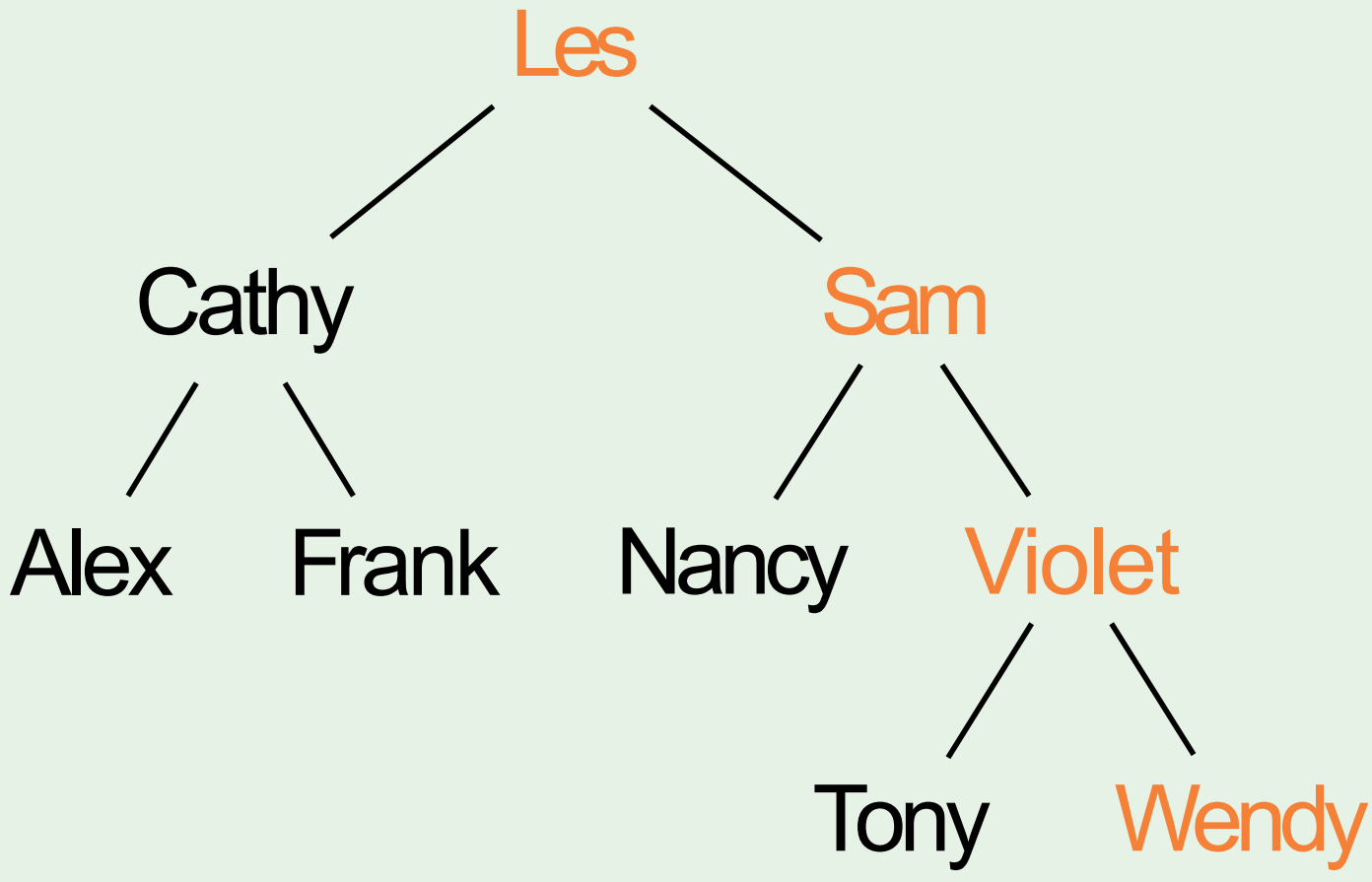
Output: Les Cathy Alex Frank Sam Nancy Violet Tony Wendy

PreOrderTraversal



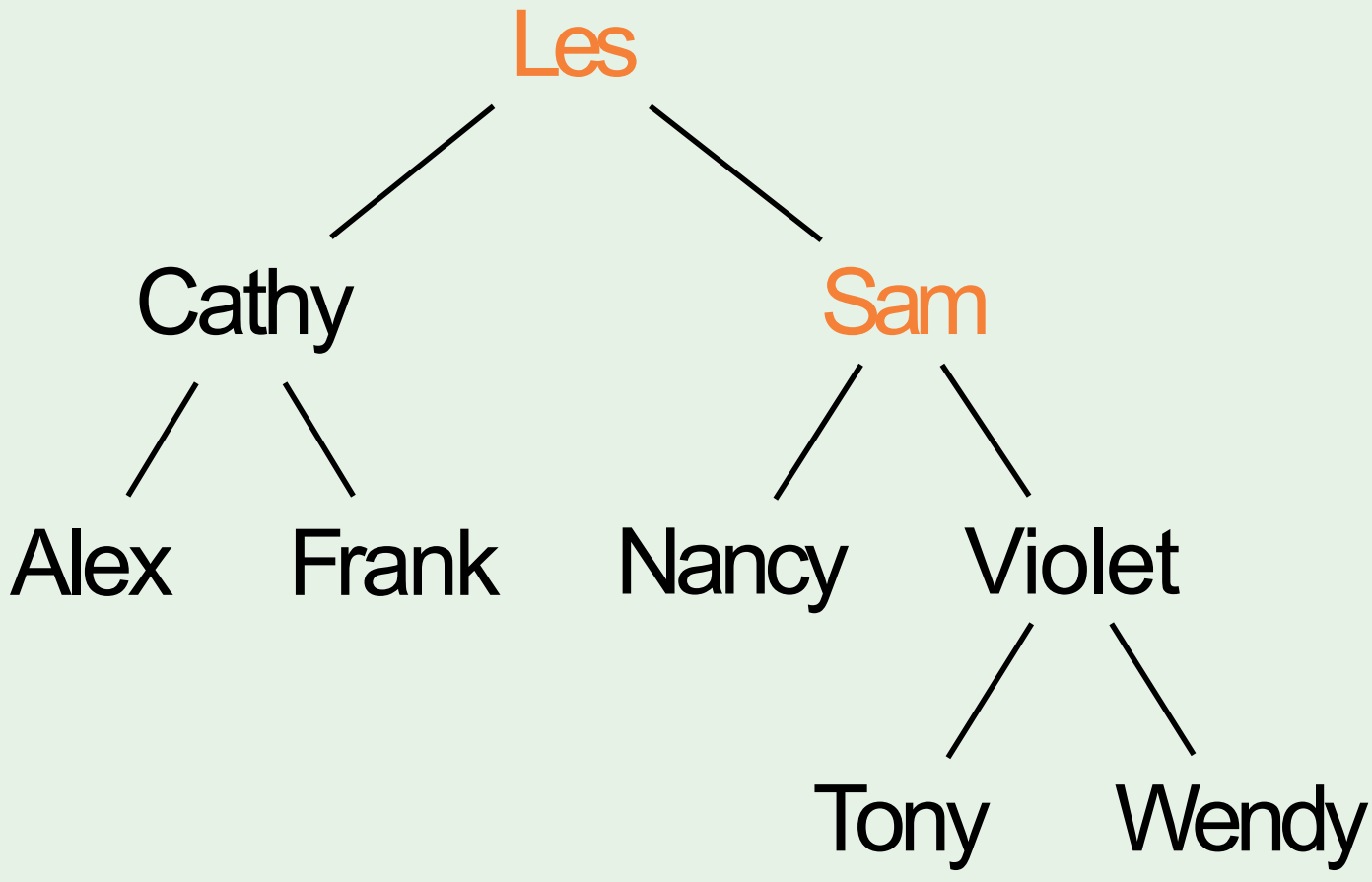
Output: Les Cathy Alex Frank Sam Nancy Violet Tony Wendy

PreOrderTraversal



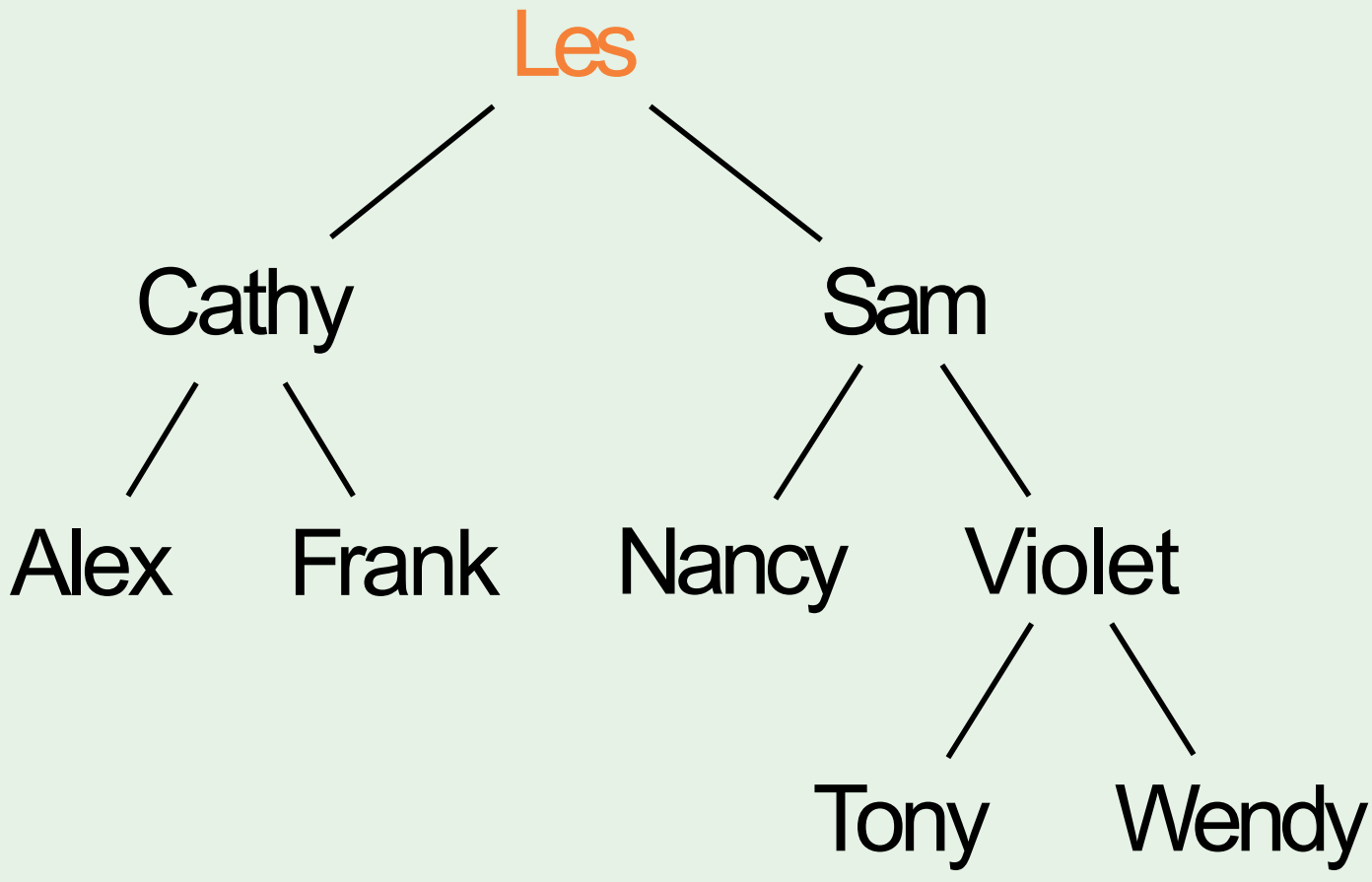
Output: Les Cathy Alex Frank Sam Nancy Violet Tony Wendy

PreOrderTraversal



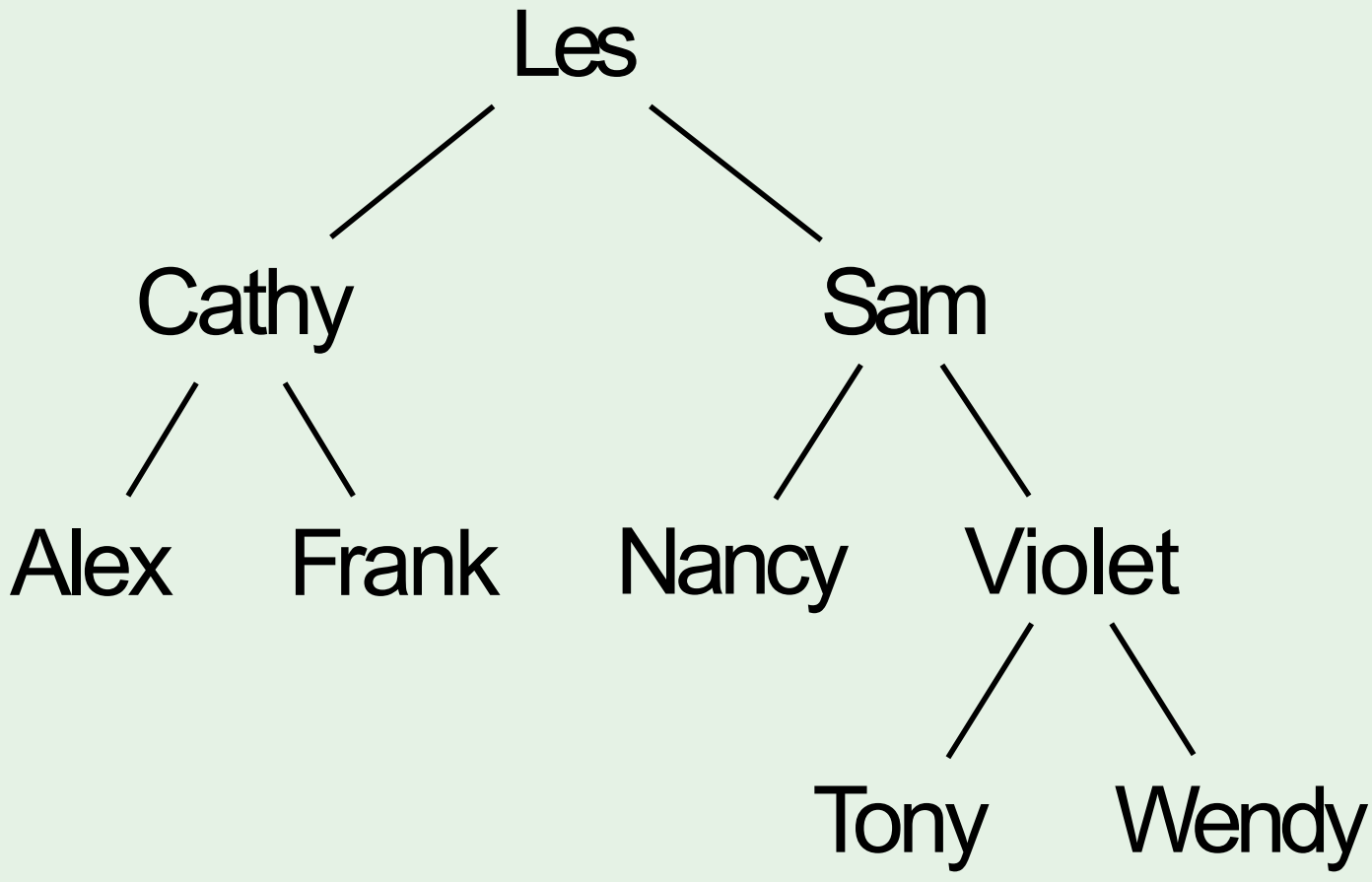
Output: Les Cathy Alex Frank Sam Nancy Violet Tony Wendy

PreOrderTraversal



Output: Les Cathy Alex Frank Sam Nancy Violet Tony Wendy

PreOrderTraversal



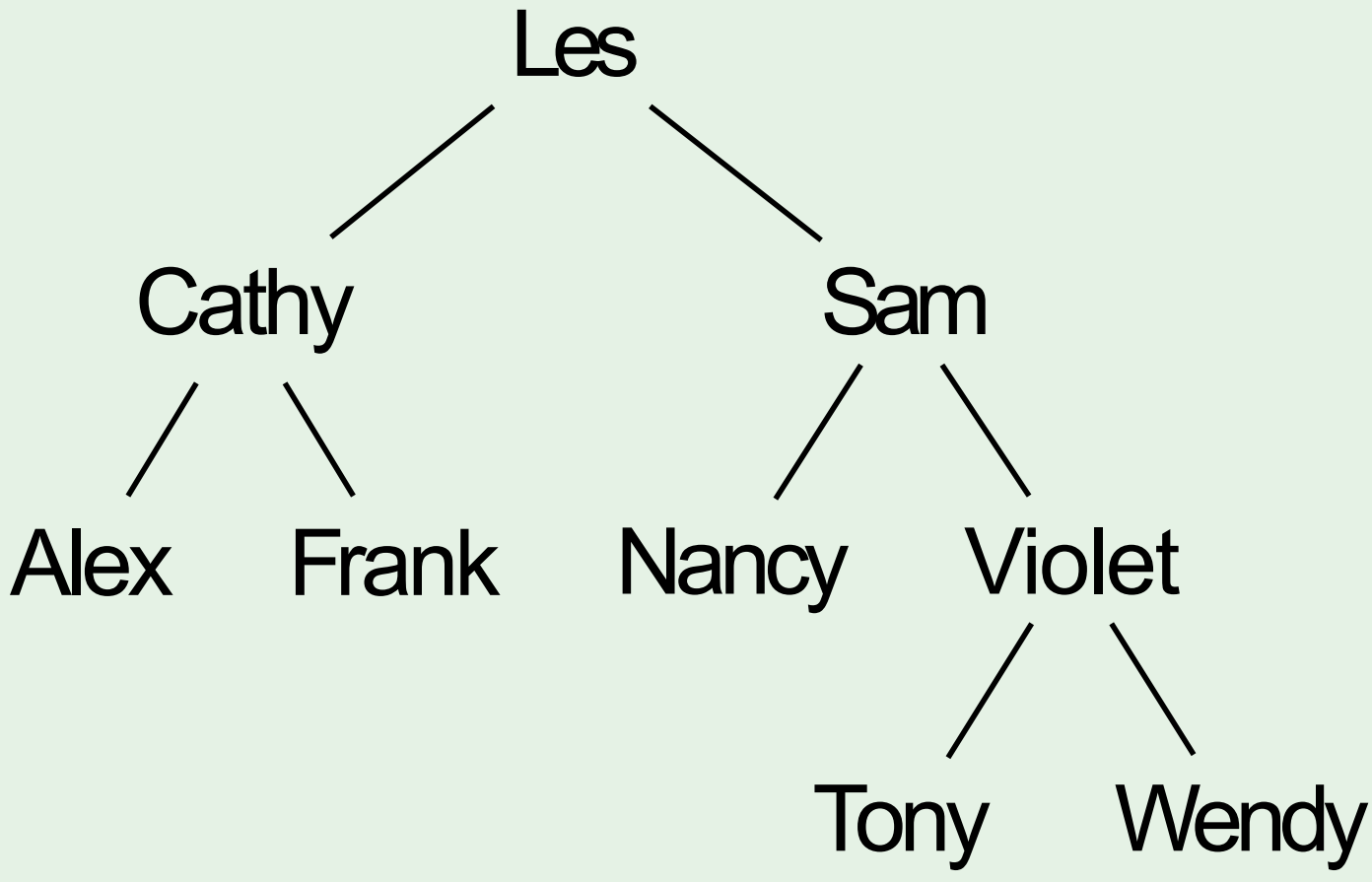
Output: Les Cathy Alex Frank Sam Nancy Violet Tony Wendy

Depth-first

PostOrderTraversal(*tree*)

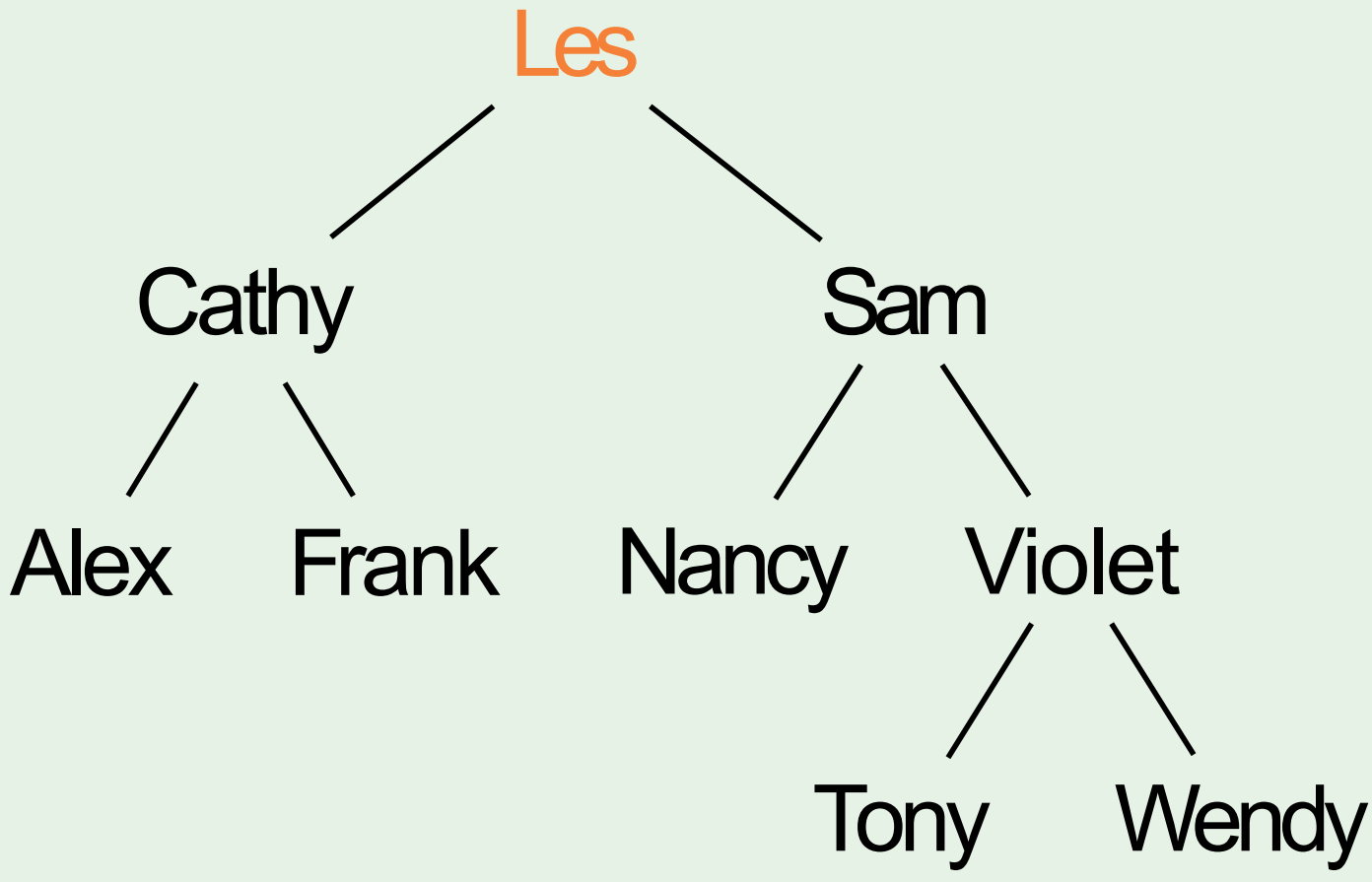
```
if tree = nil:
   return
PostOrderTraversal(tree.left)
PostOrderTraversal(tree.right)
Print(tree.key)
```

PostOrderTraversal



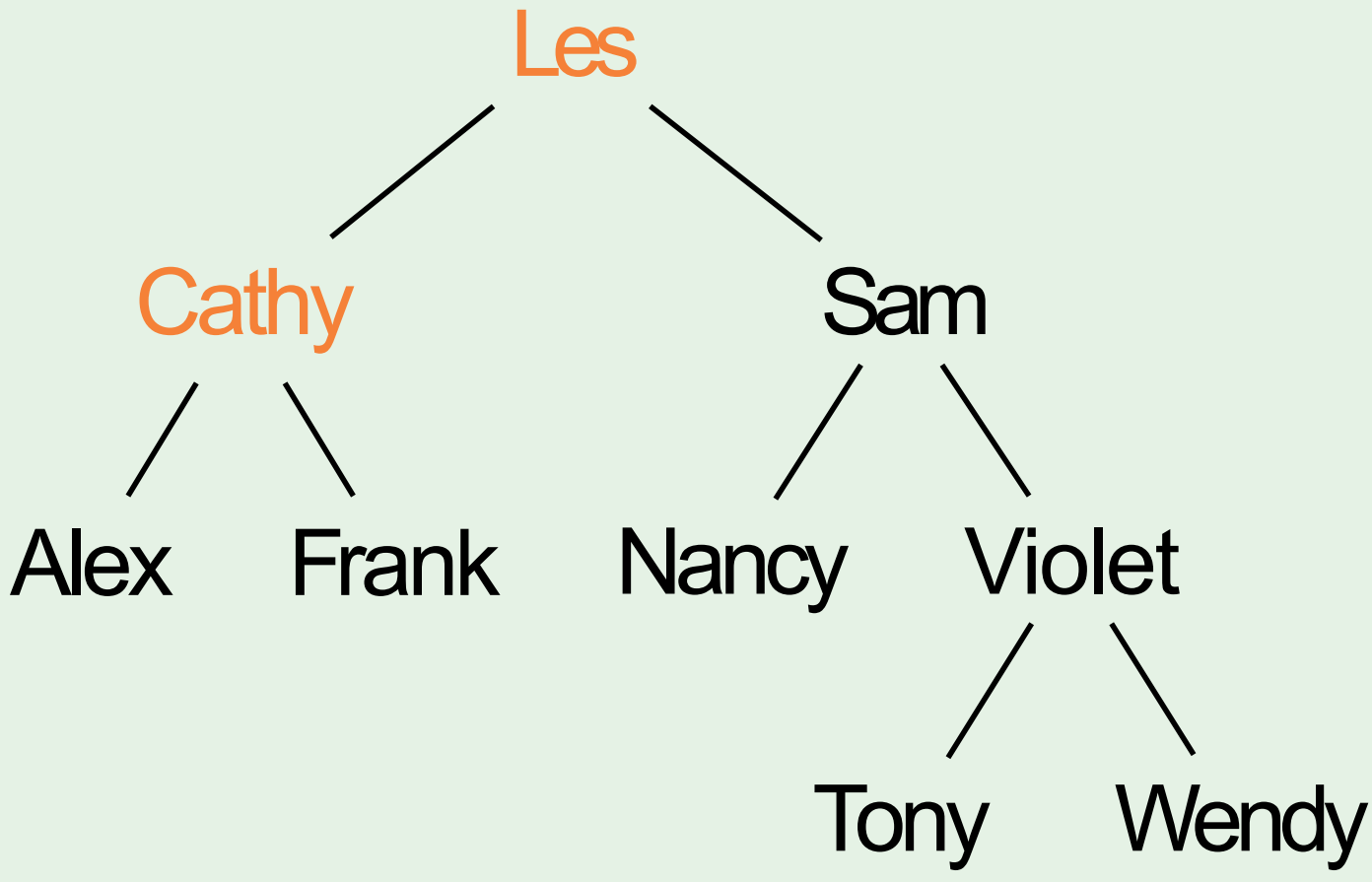
Output:

PostOrderTraversal



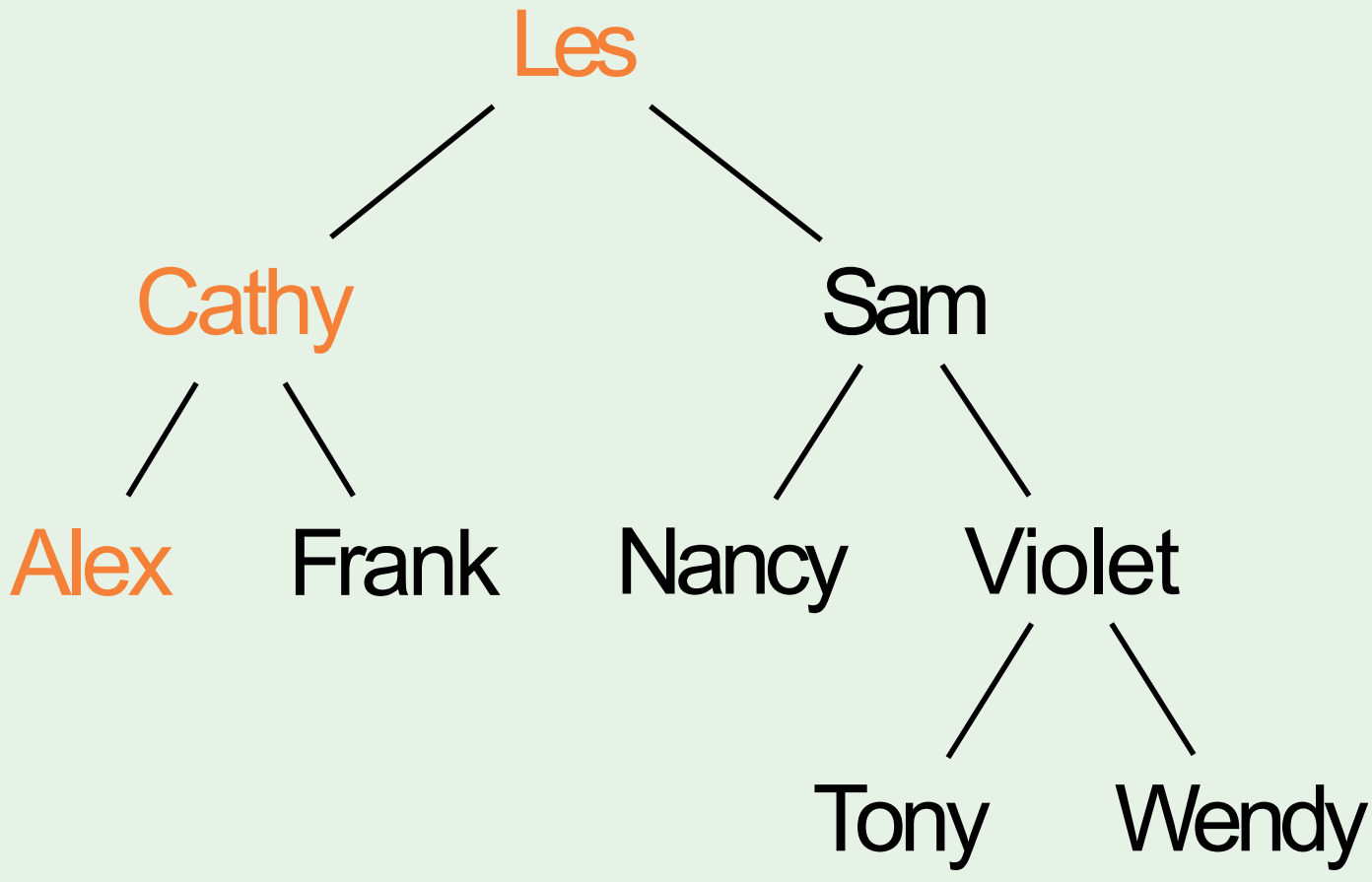
Output:

PostOrderTraversal



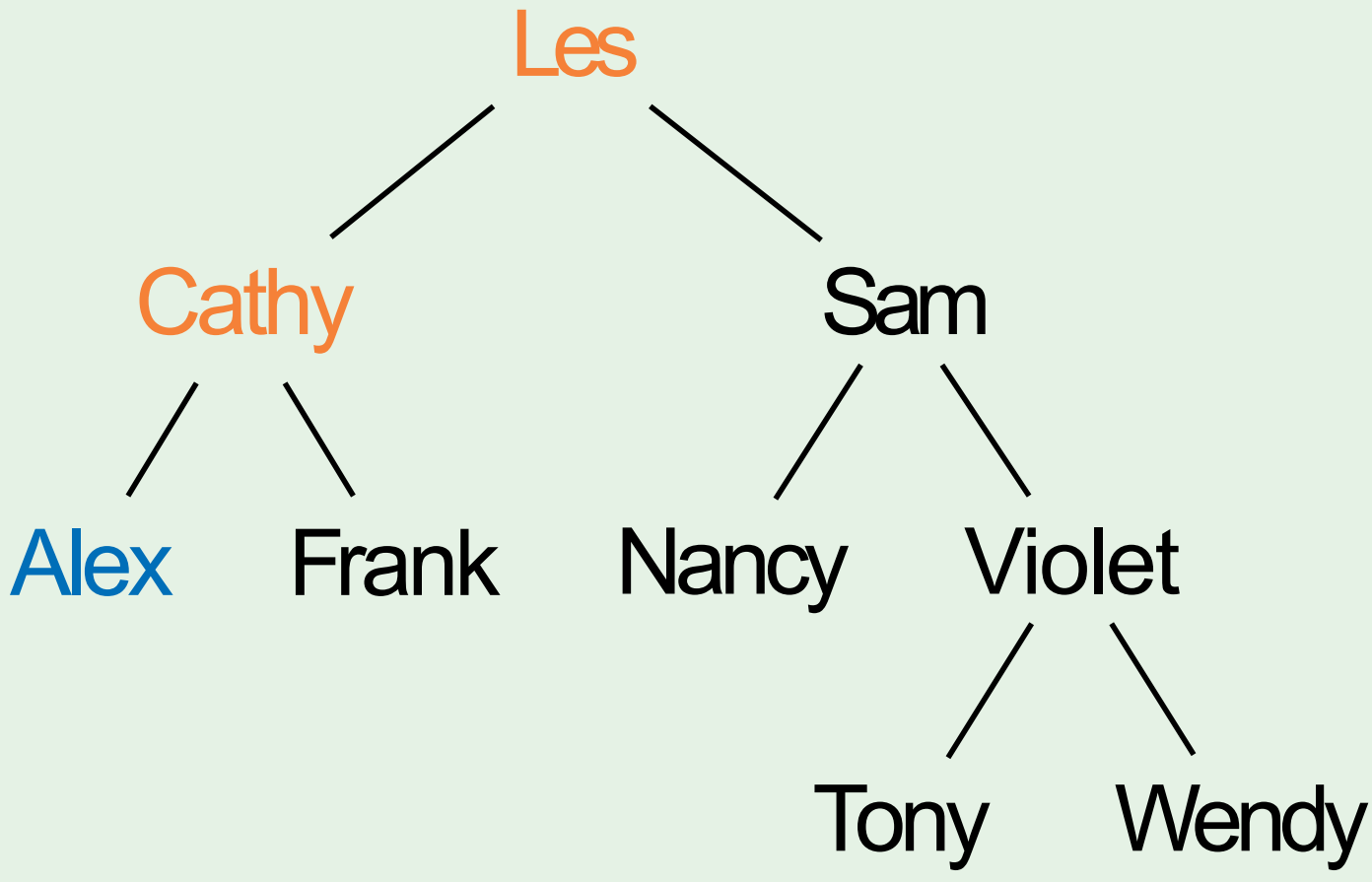
Output:

PostOrderTraversal



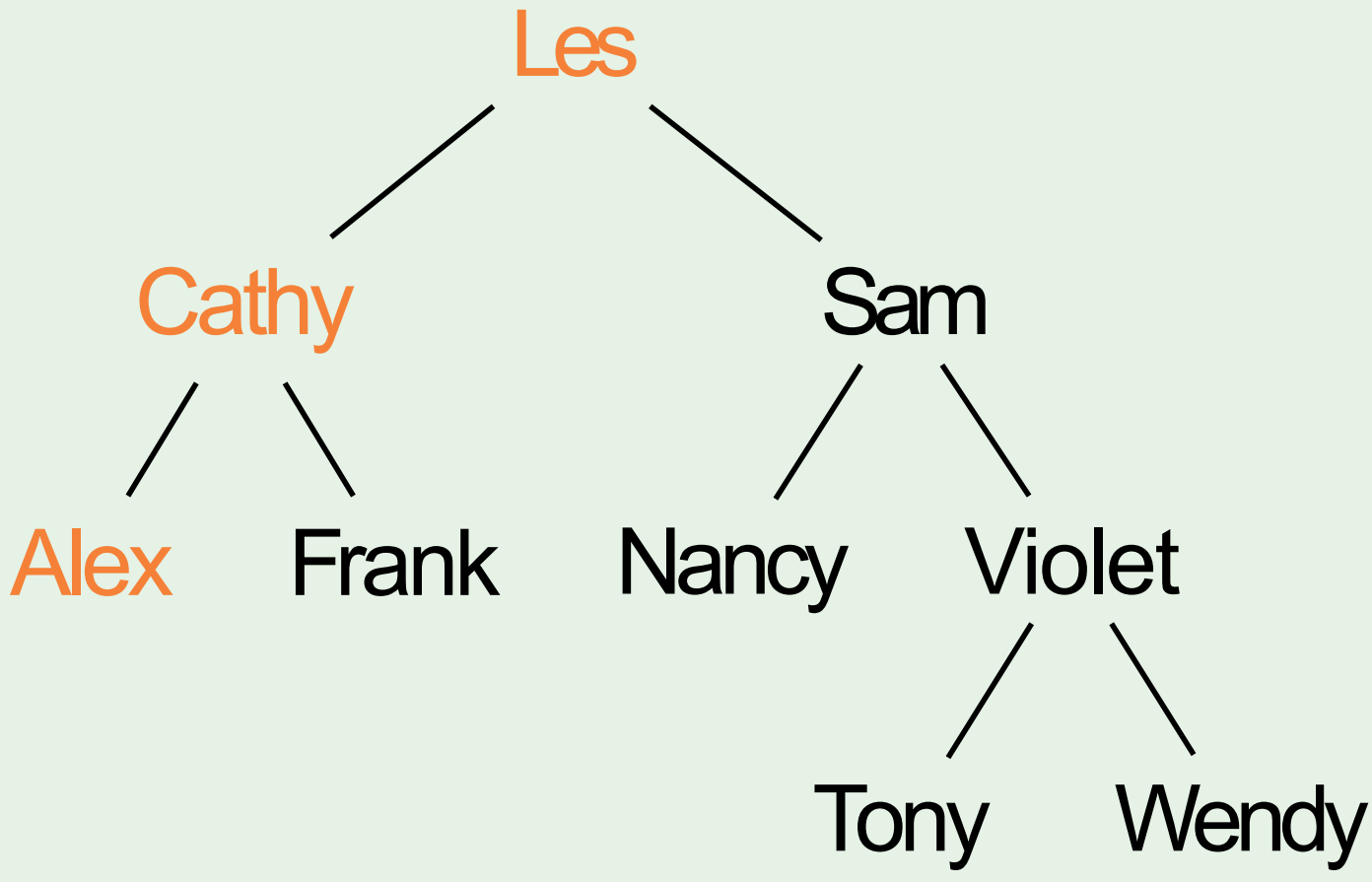
Output:

PostOrderTraversal



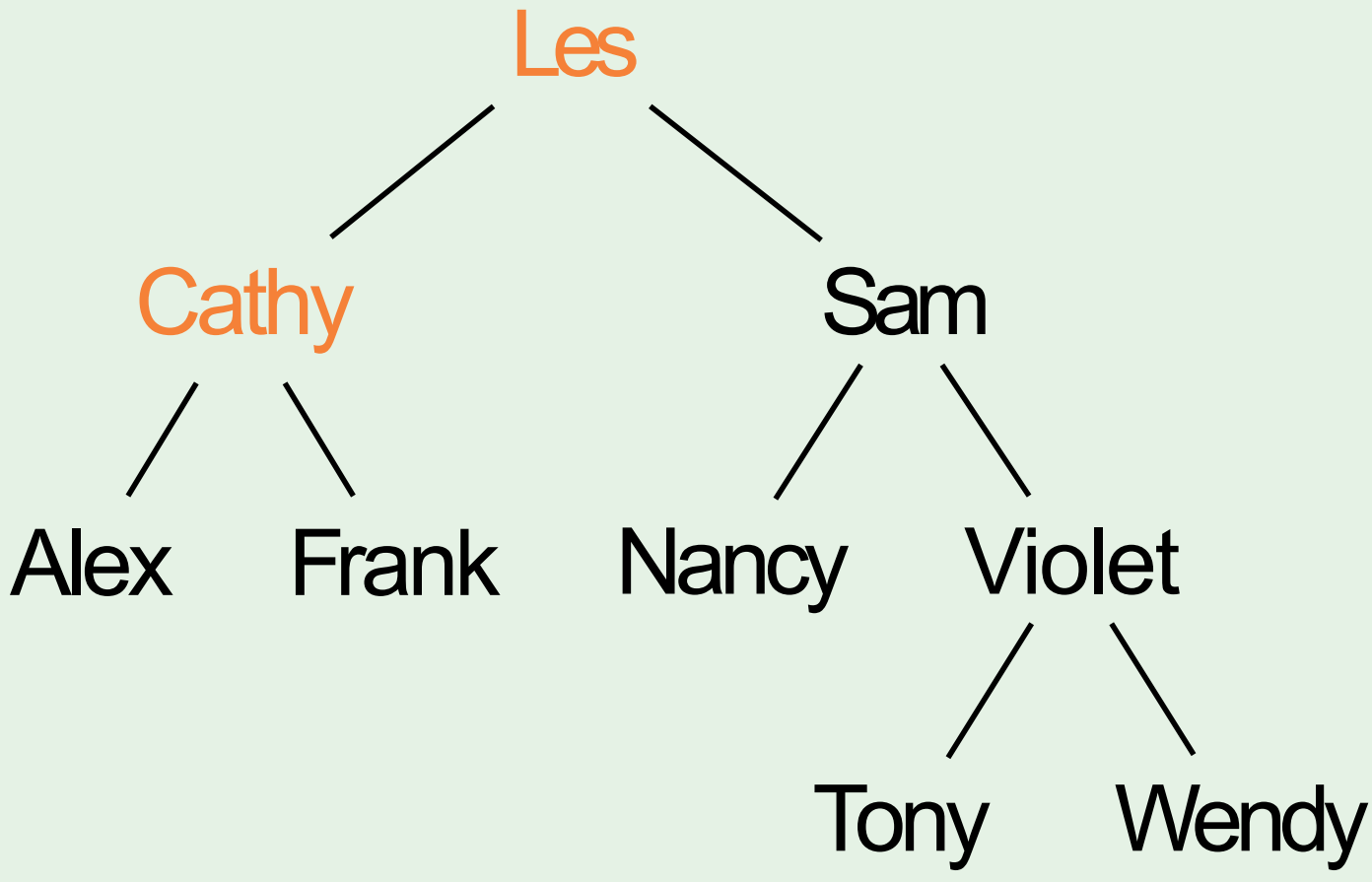
Output: Alex

PostOrderTraversal



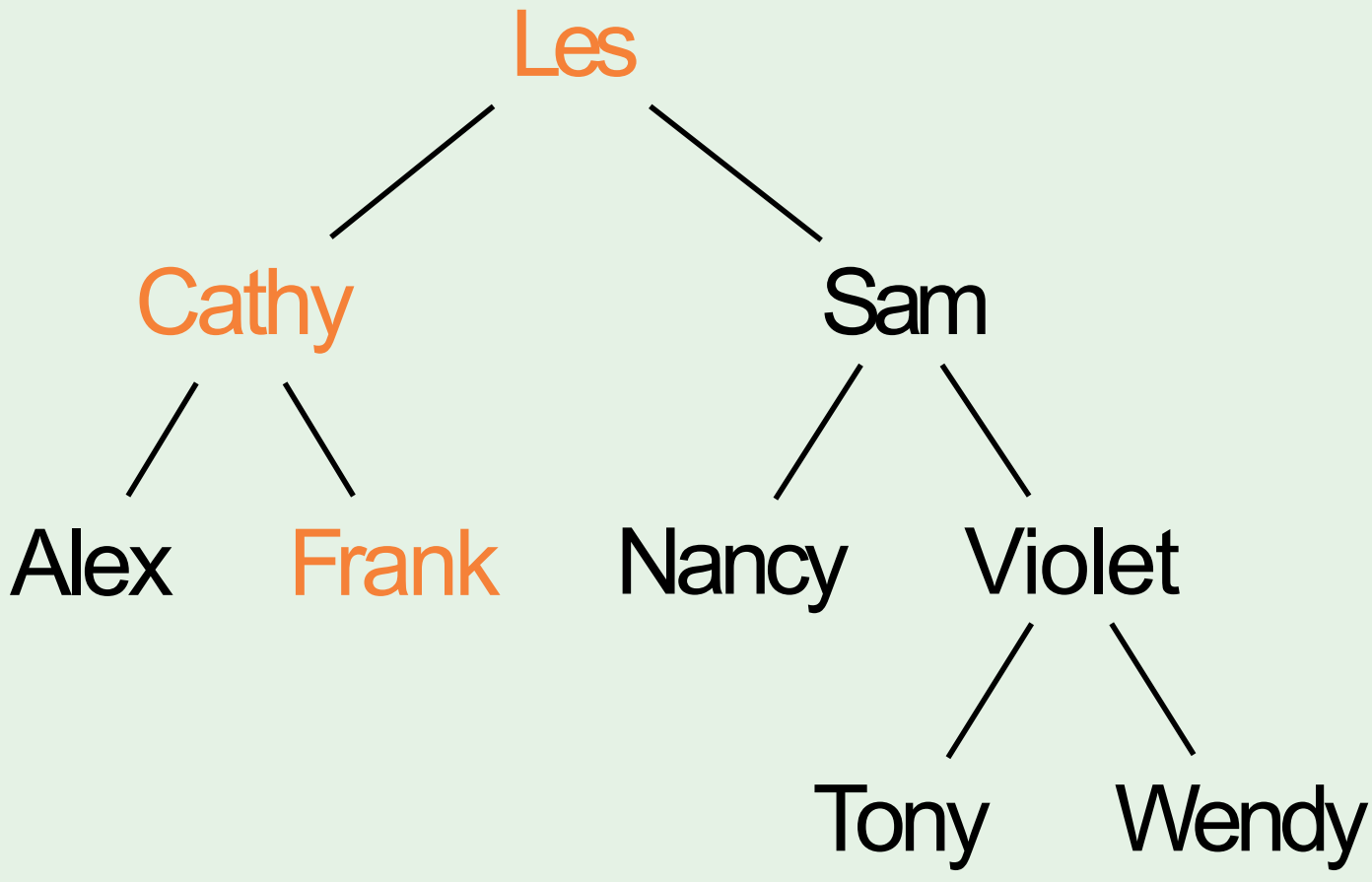
Output: Alex

PostOrderTraversal



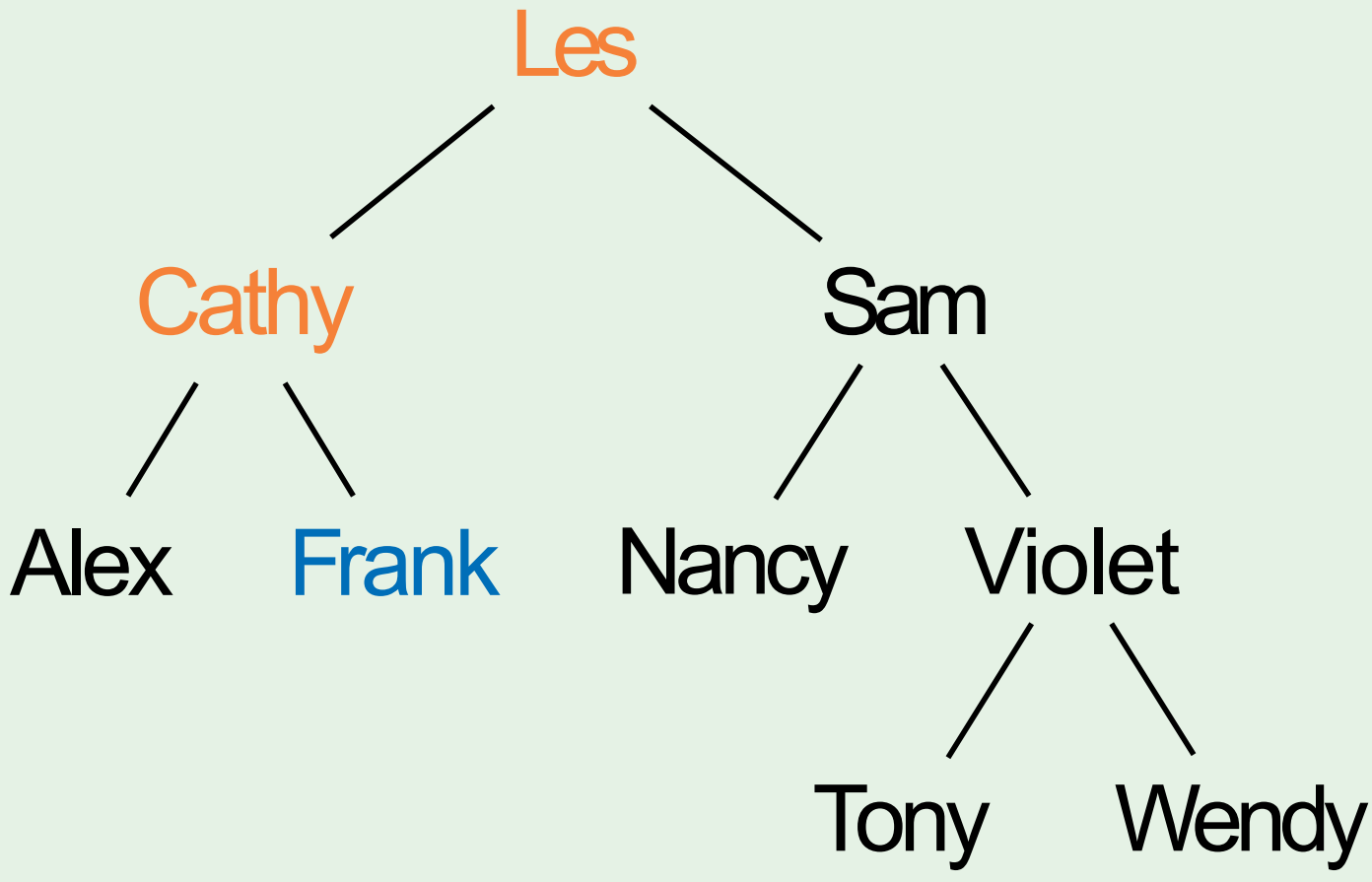
Output: Alex

PostOrderTraversal



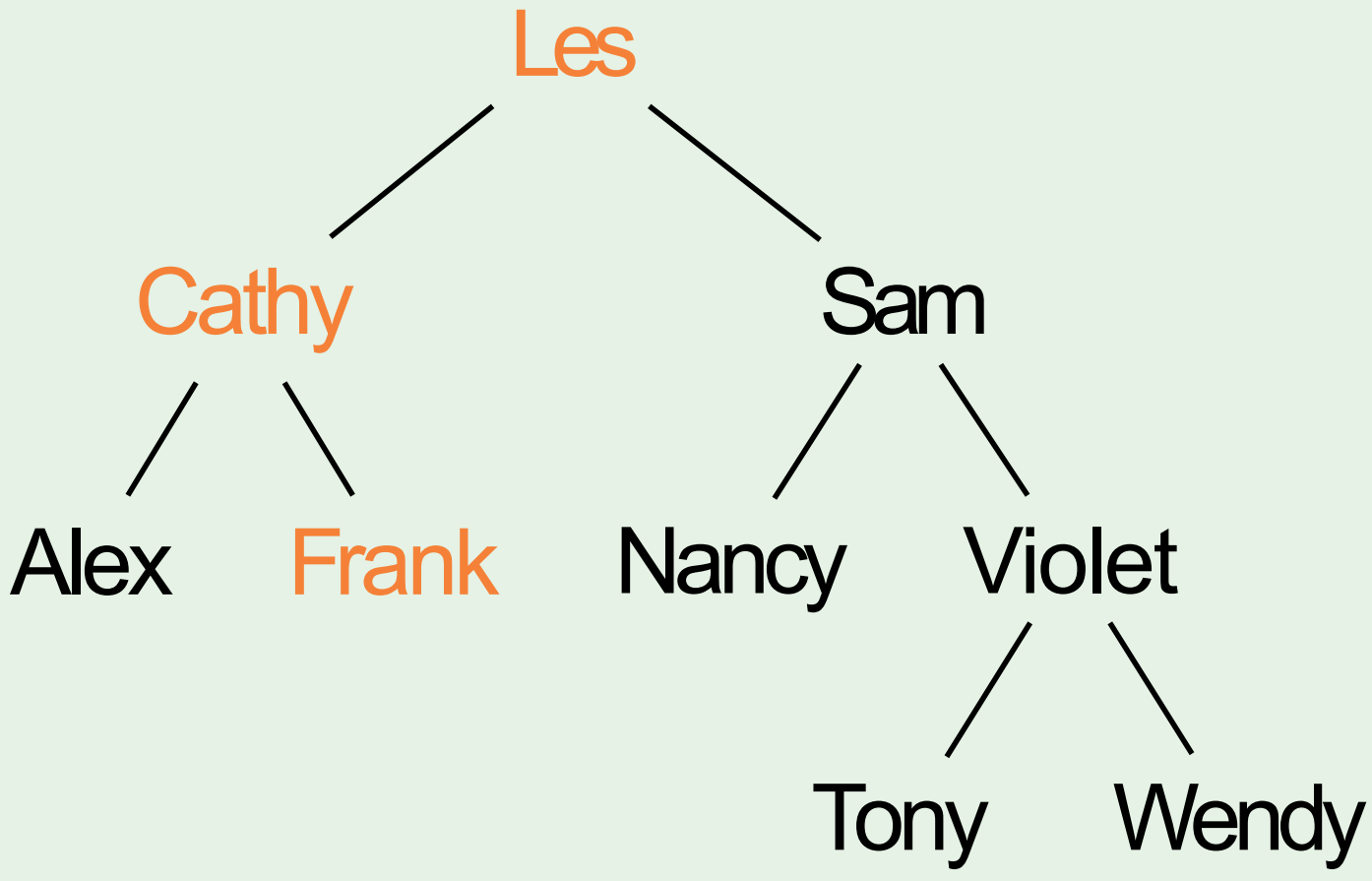
Output: Alex

PostOrderTraversal



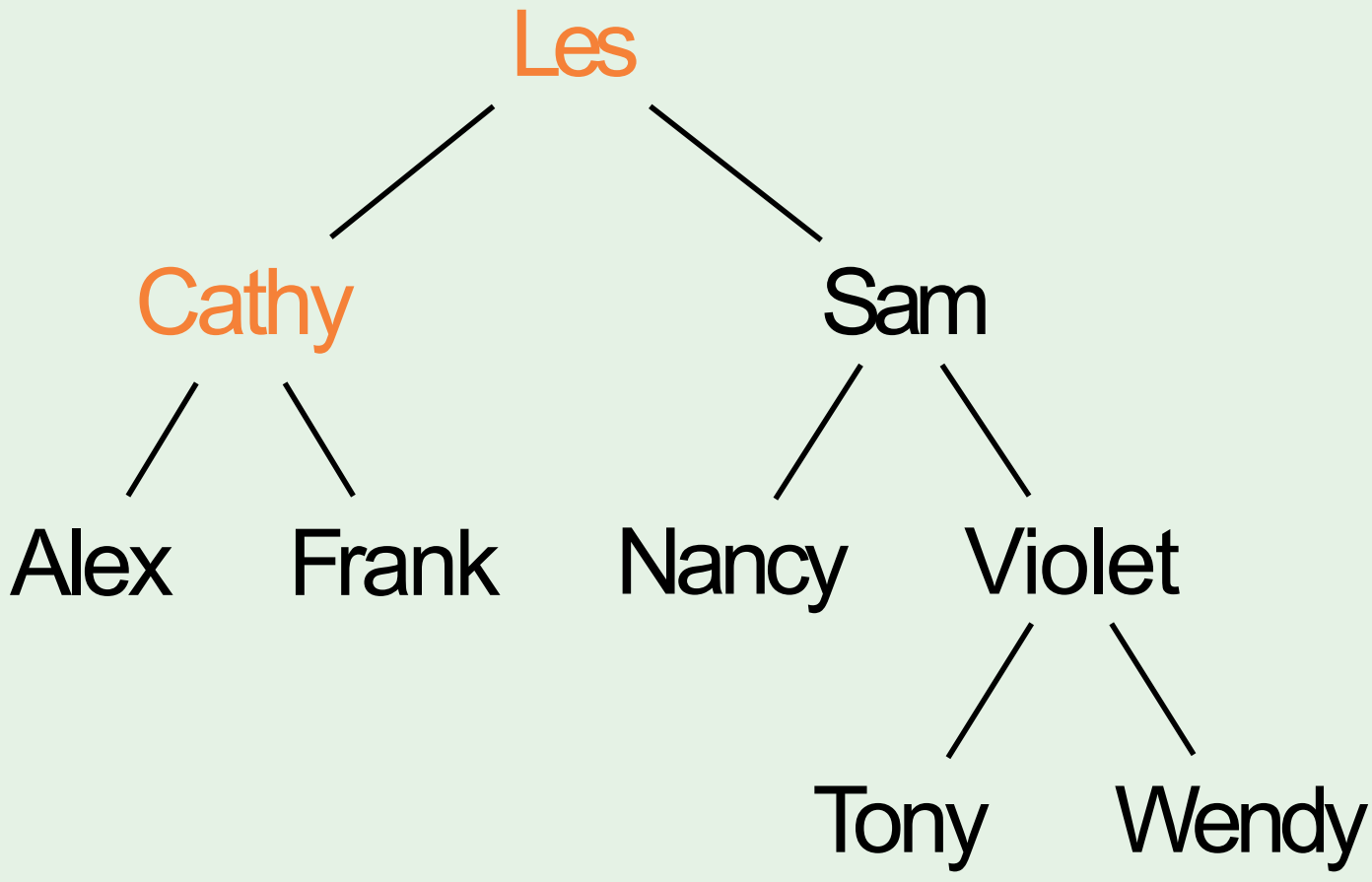
Output: Alex Frank

PostOrderTraversal



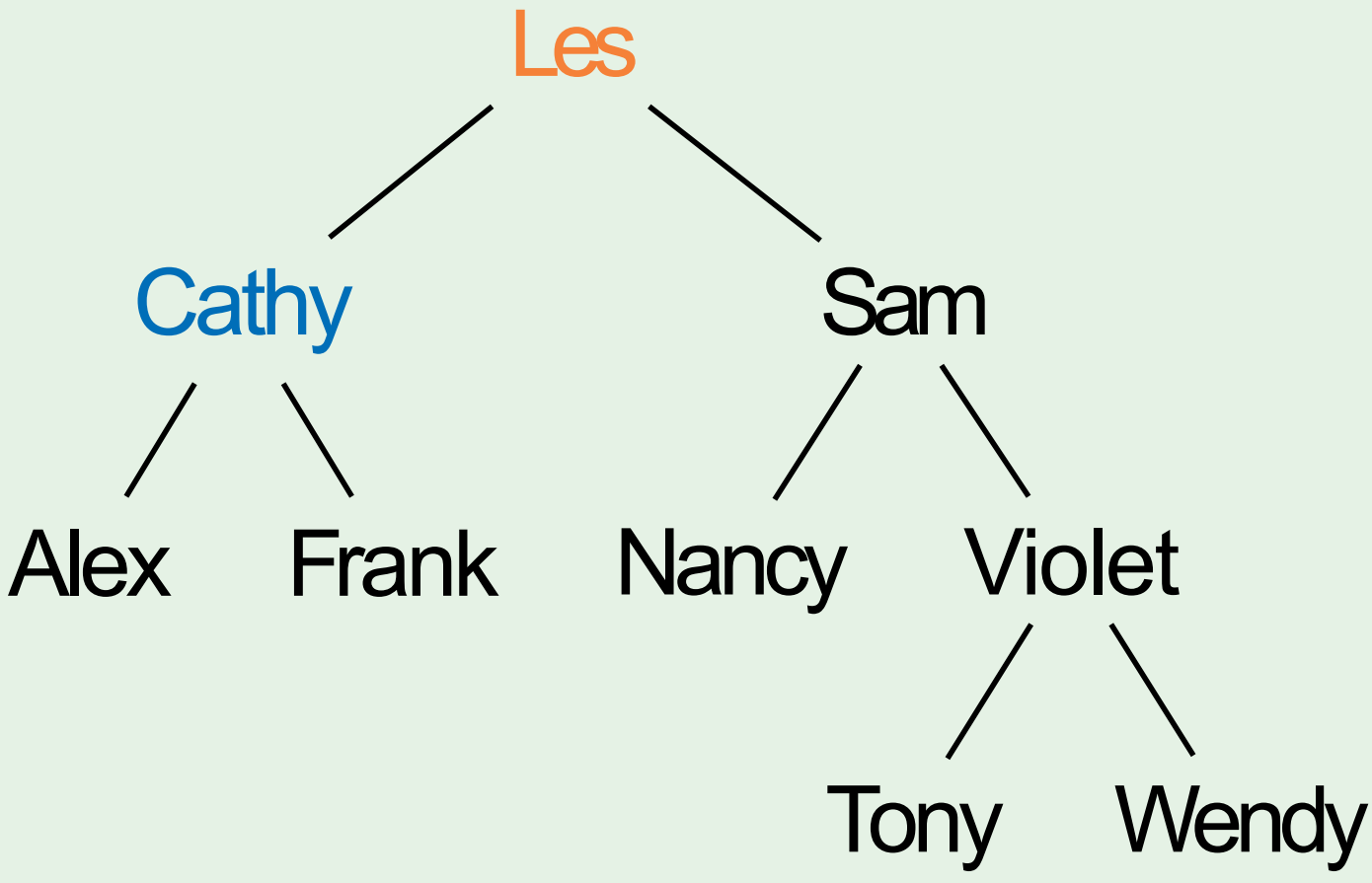
Output: Alex Frank

PostOrderTraversal



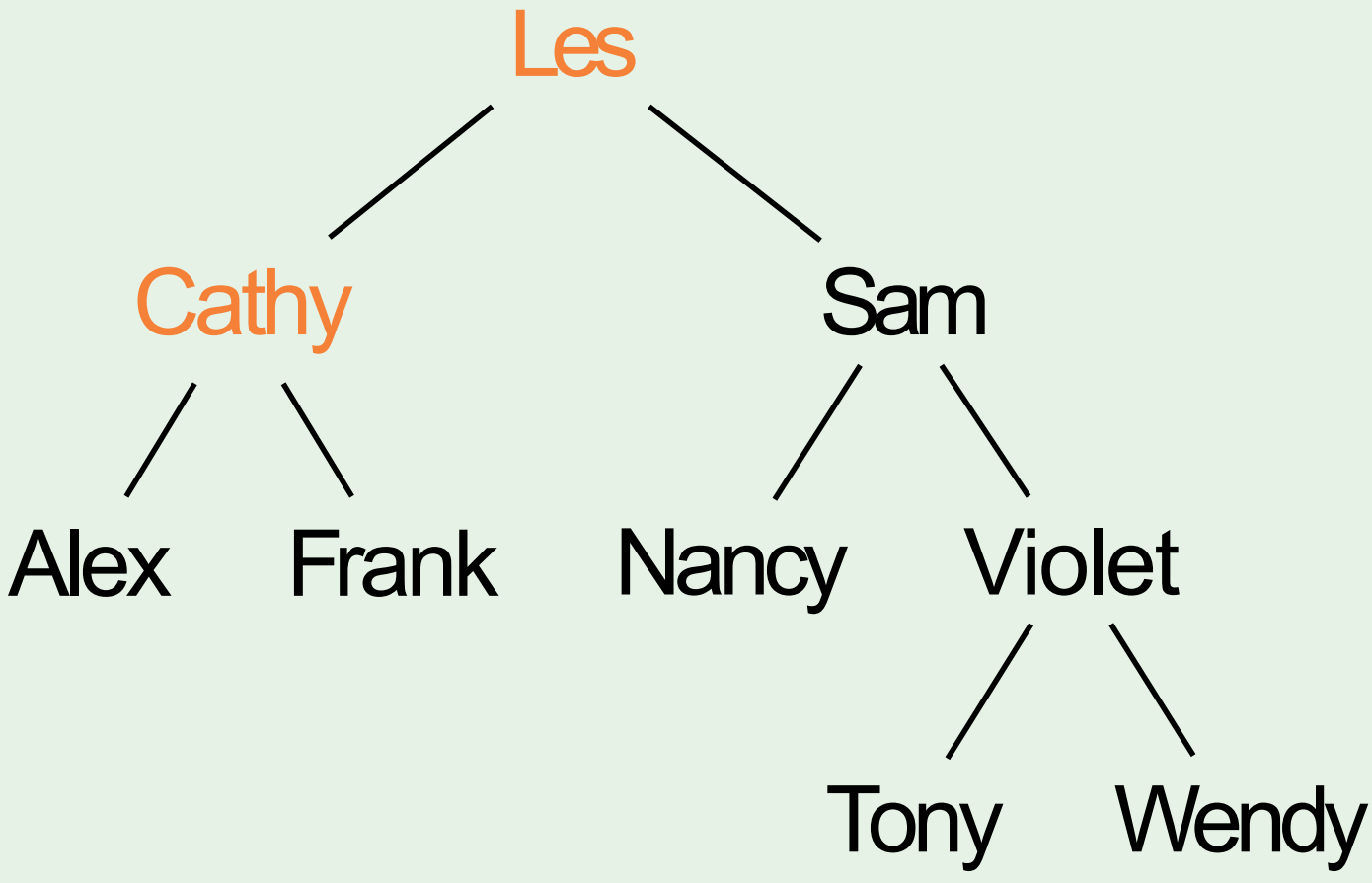
Output: AlexFrank

PostOrderTraversal



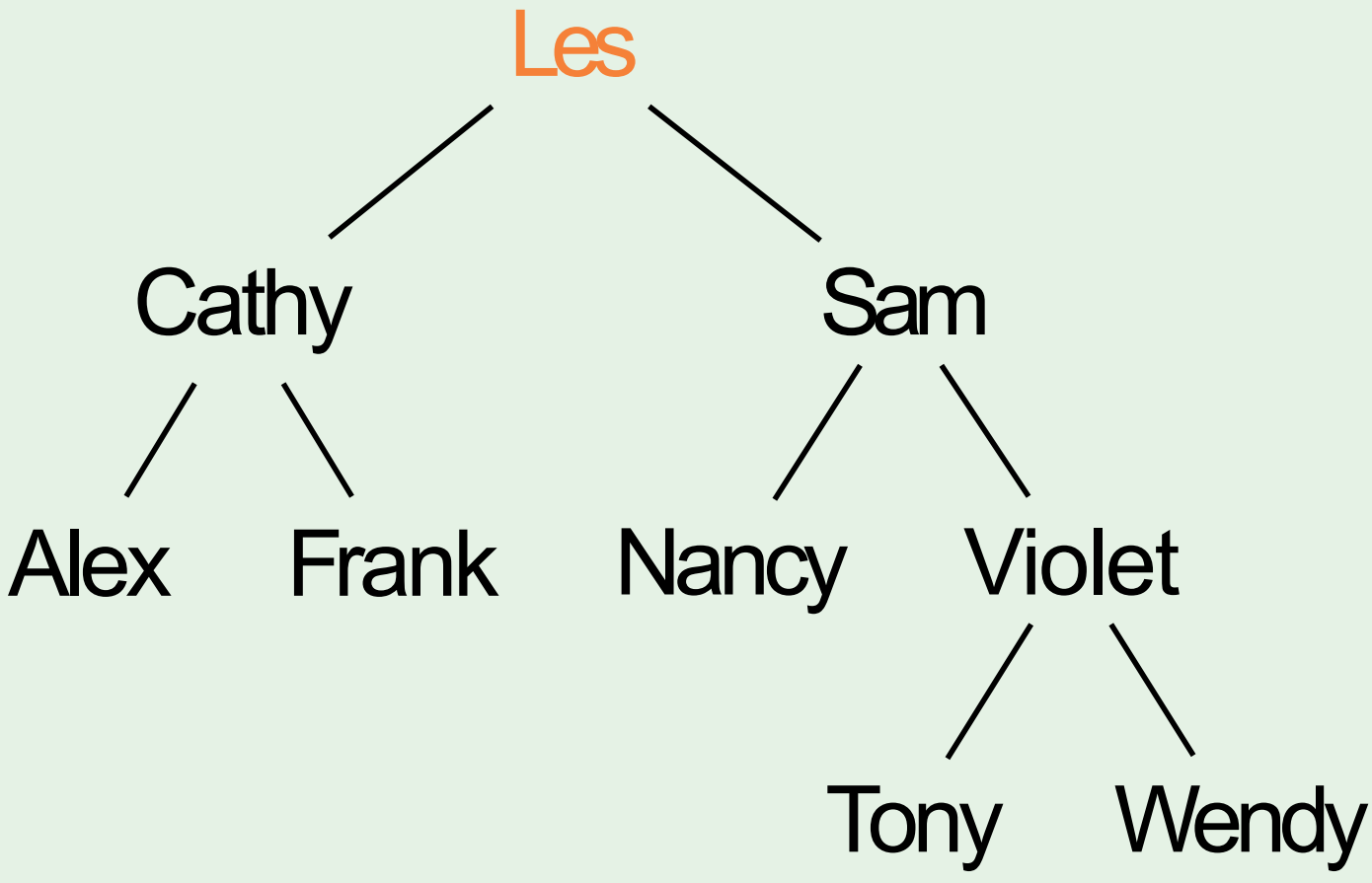
Output: Alex Frank Cathy

PostOrderTraversal



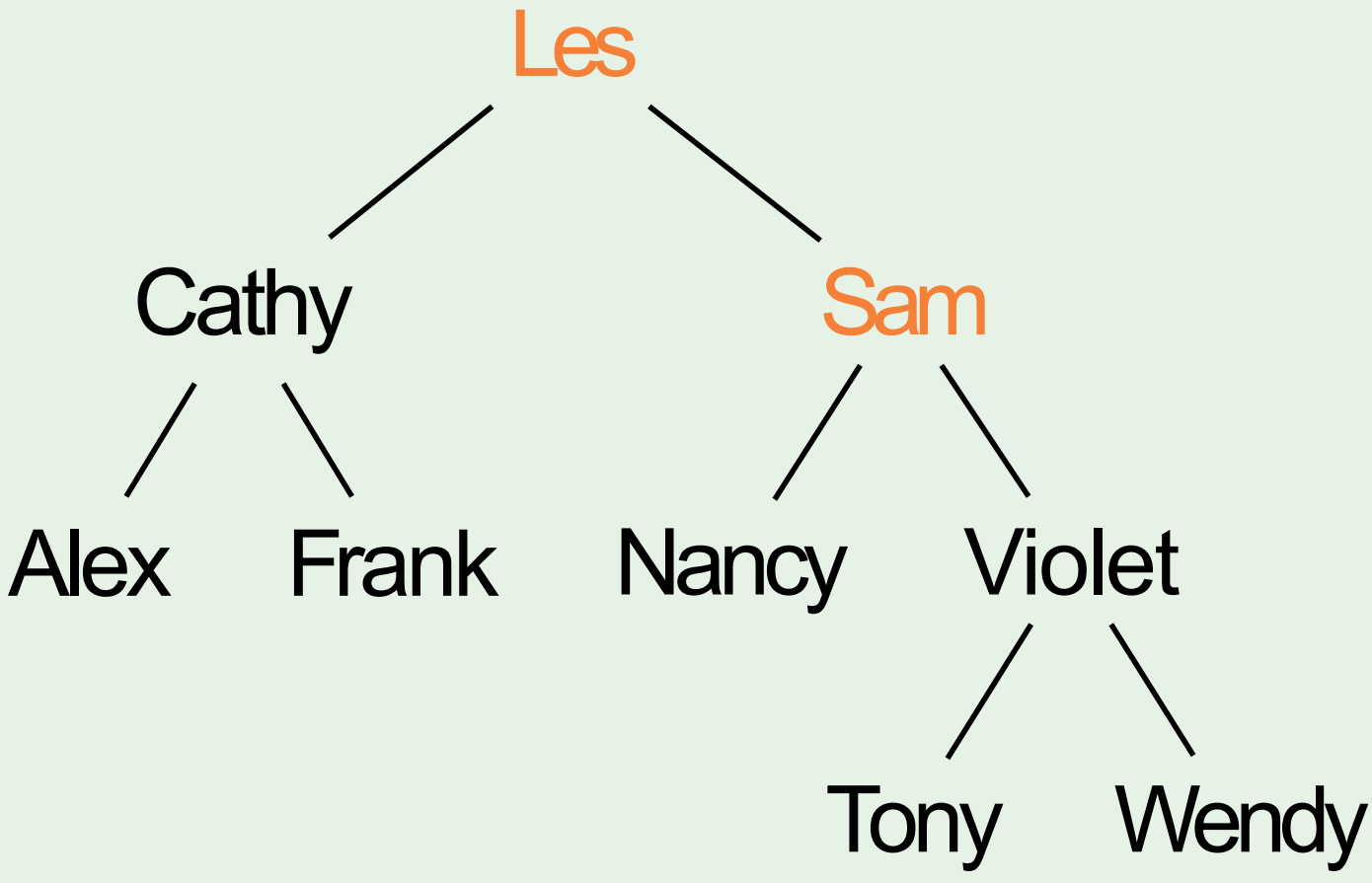
Output: Alex Frank Cathy

PostOrderTraversal



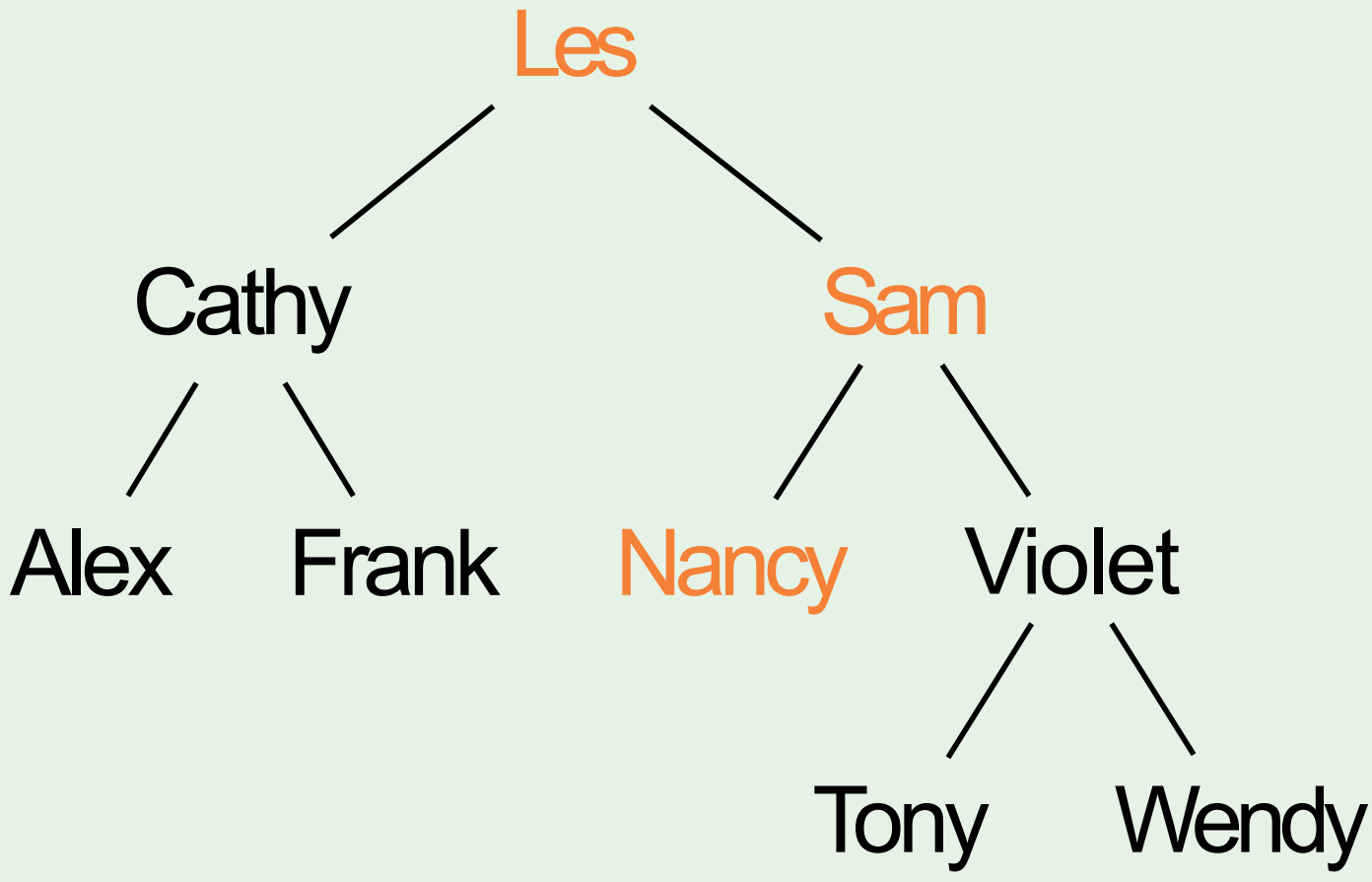
Output: Alex Frank Cathy

PostOrderTraversal



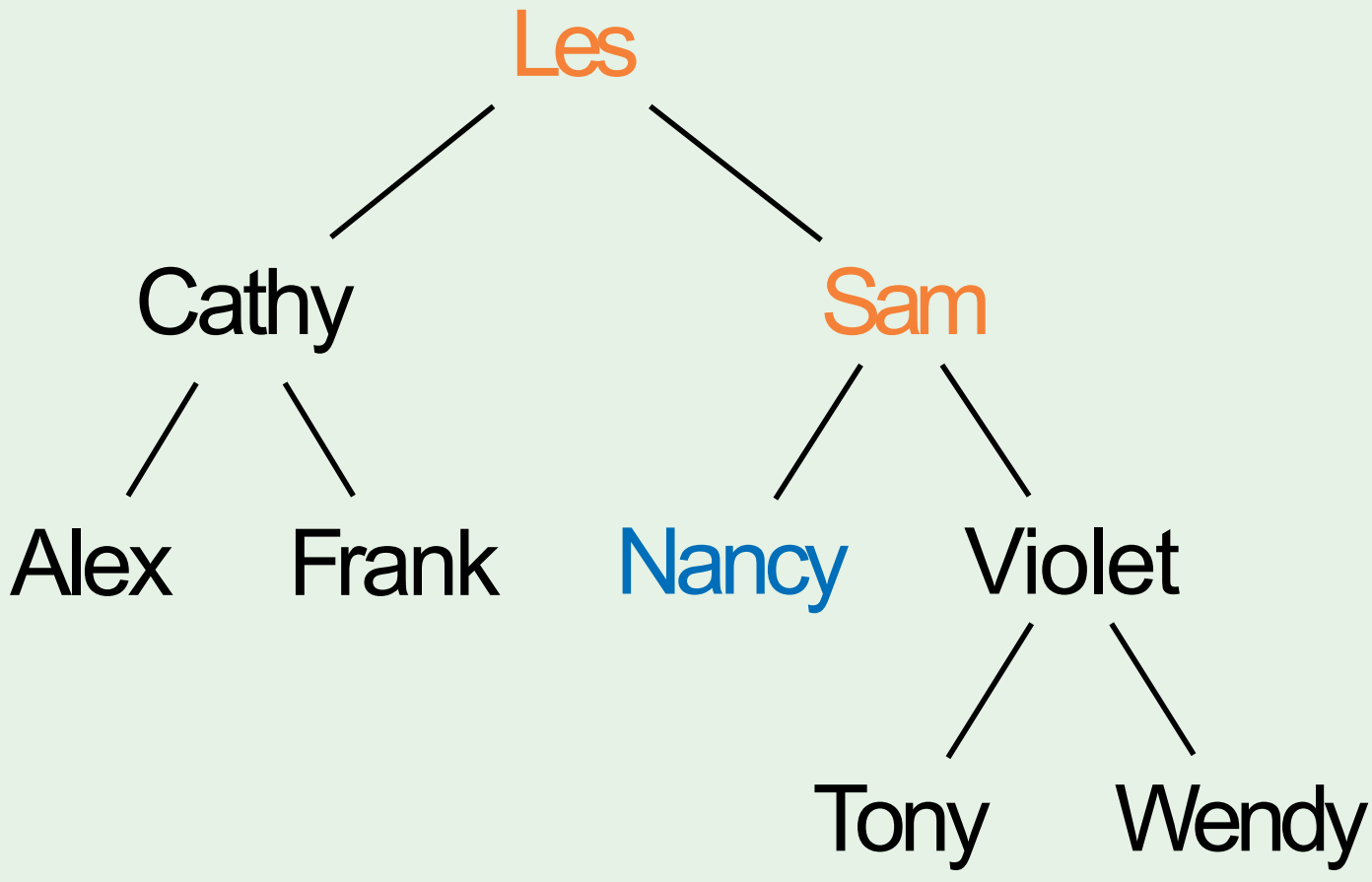
Output: Alex Frank Cathy

PostOrderTraversal



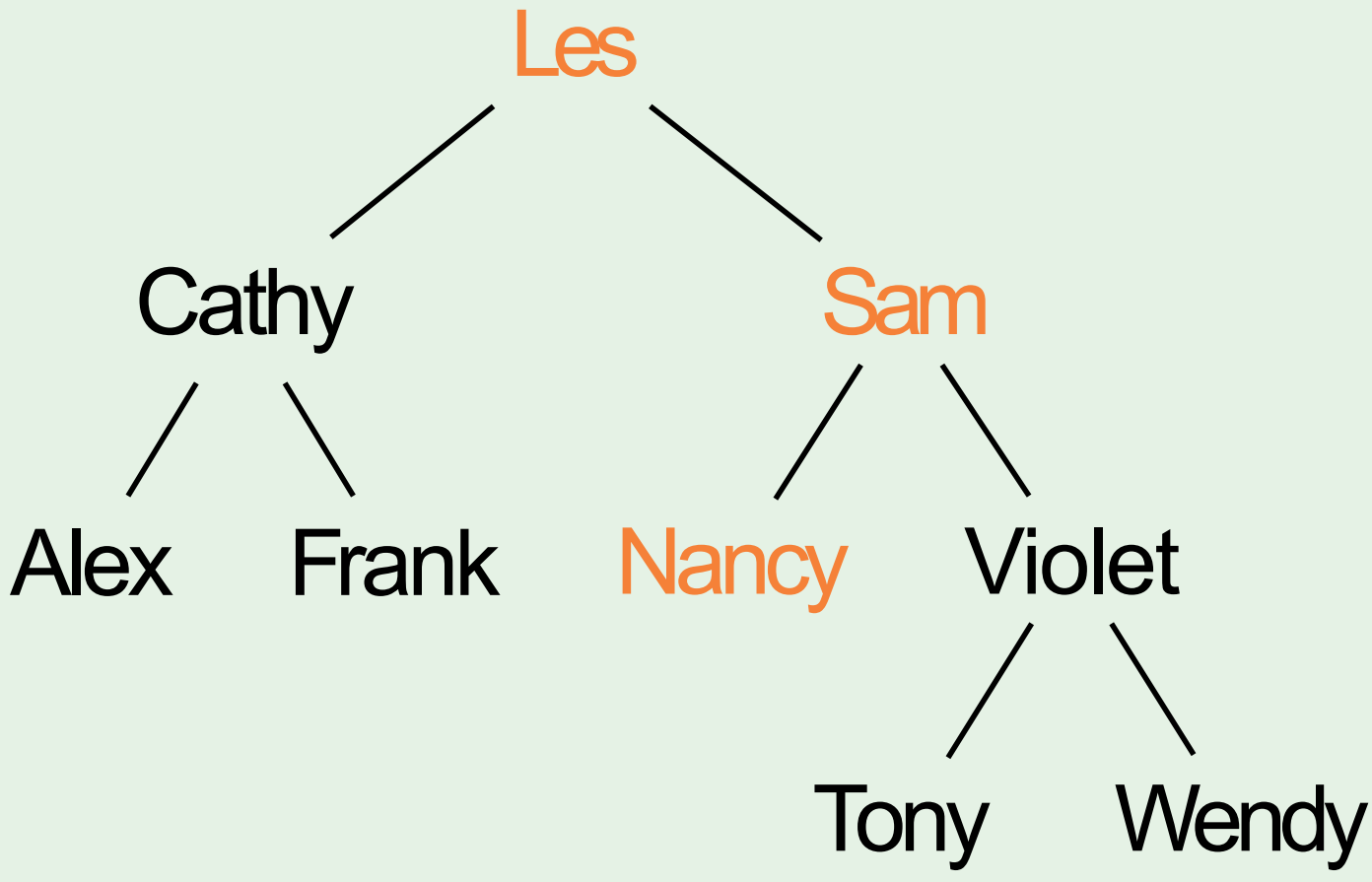
Output: Alex Frank Cathy

PostOrderTraversal



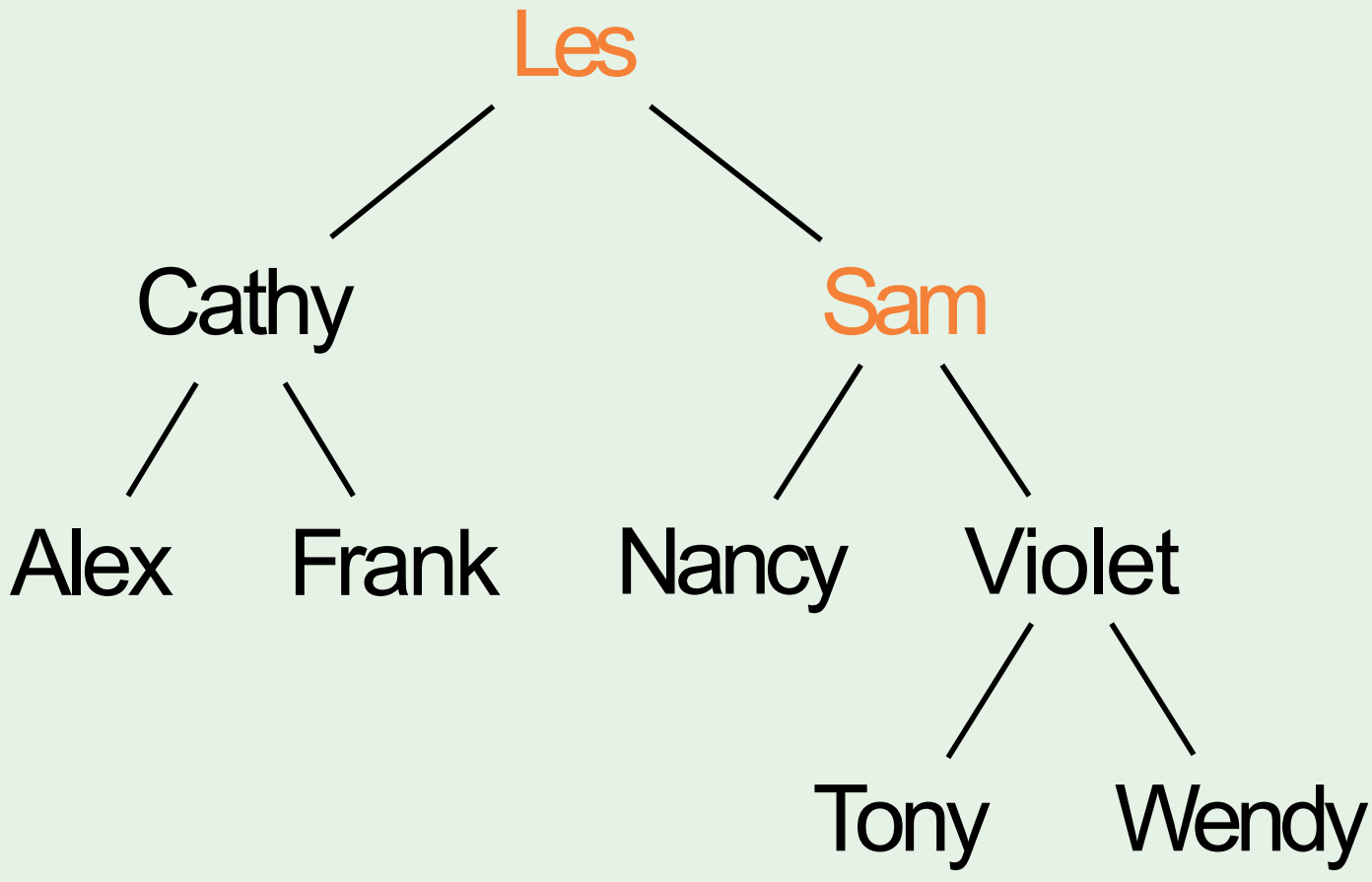
Output: Alex Frank Cathy Nancy

PostOrderTraversal



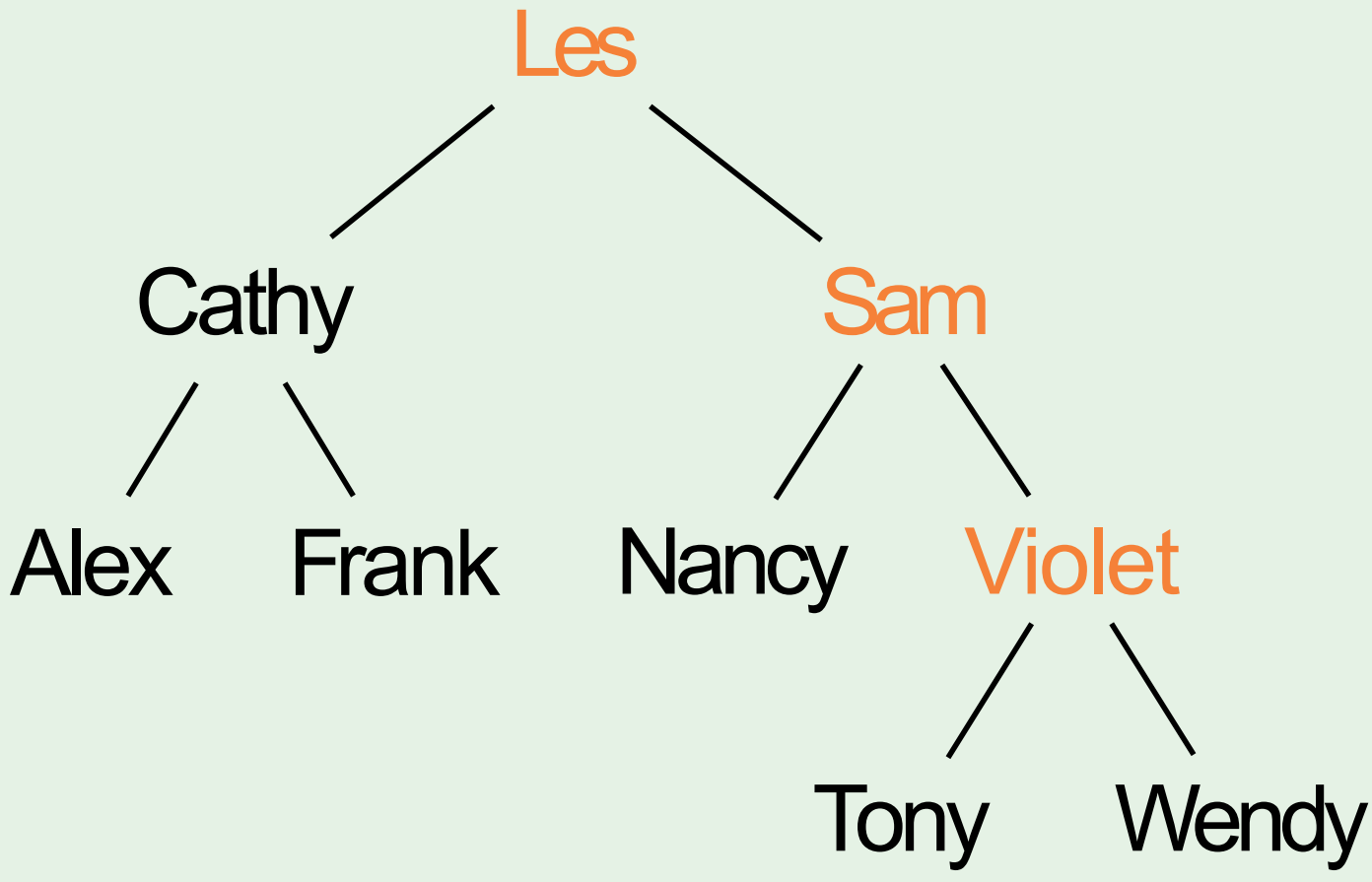
Output: Alex Frank Cathy Nancy

PostOrderTraversal



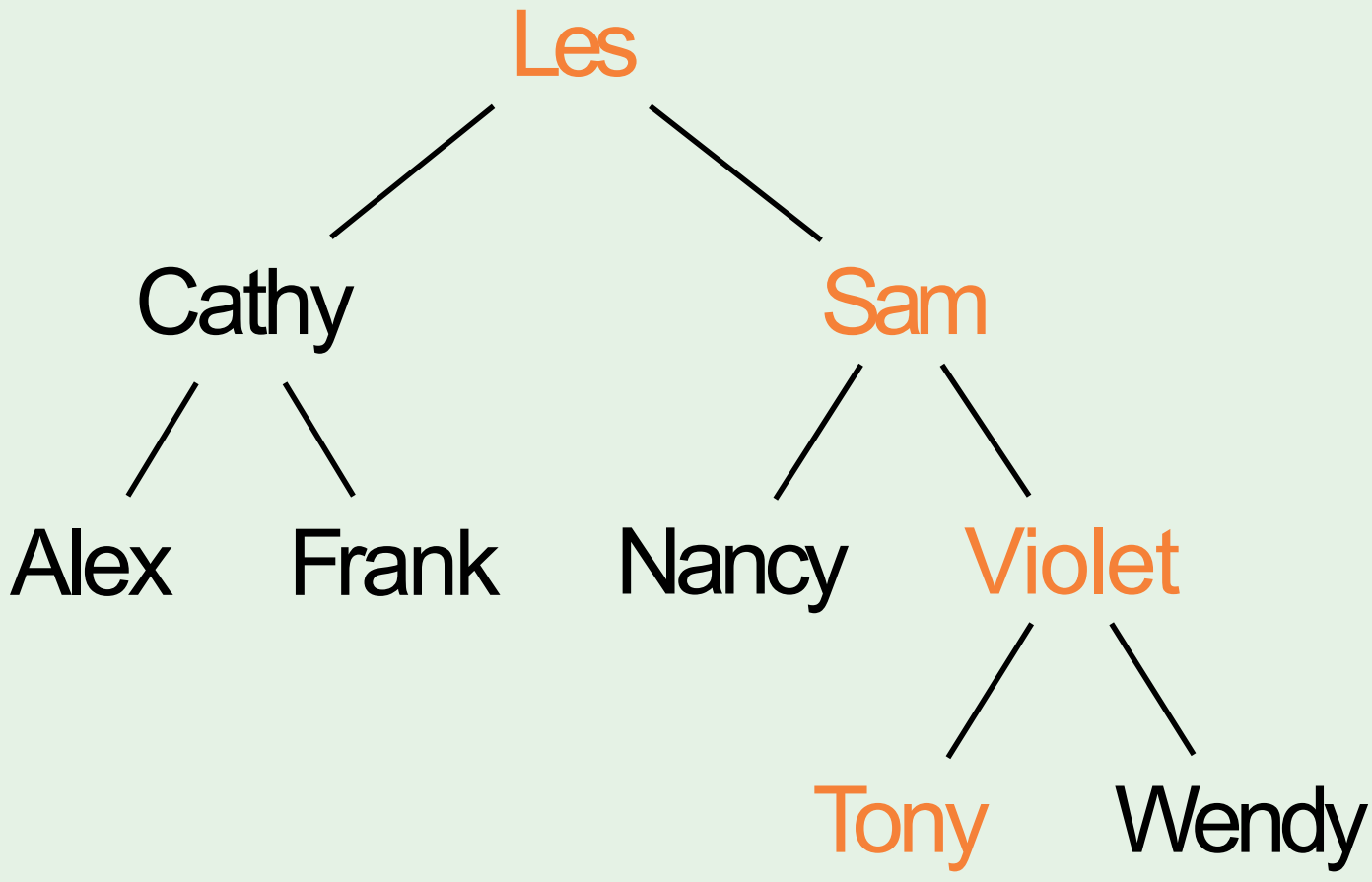
Output: Alex Frank Cathy Nancy

PostOrderTraversal



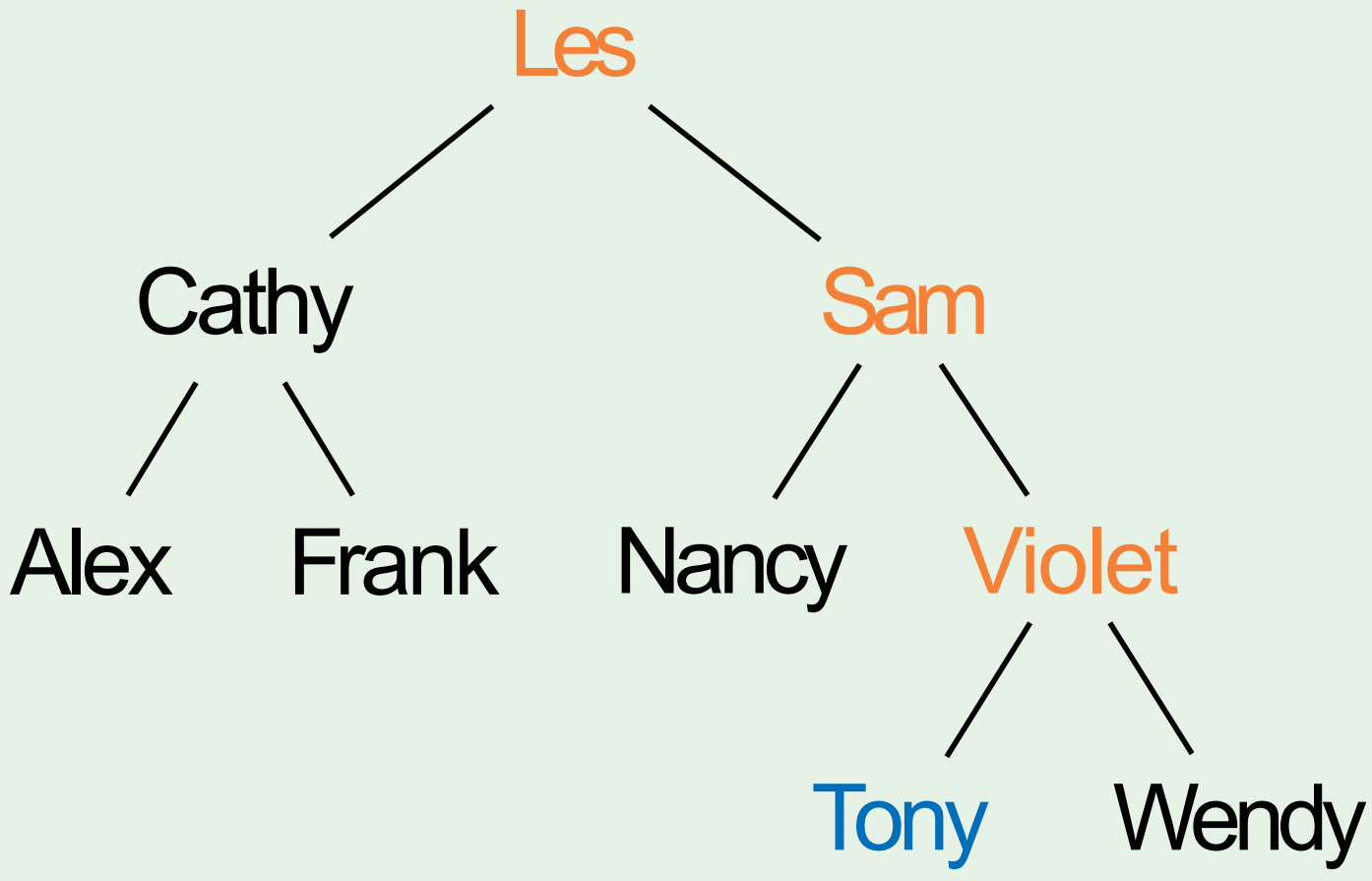
Output: Alex Frank Cathy Nancy

PostOrderTraversal



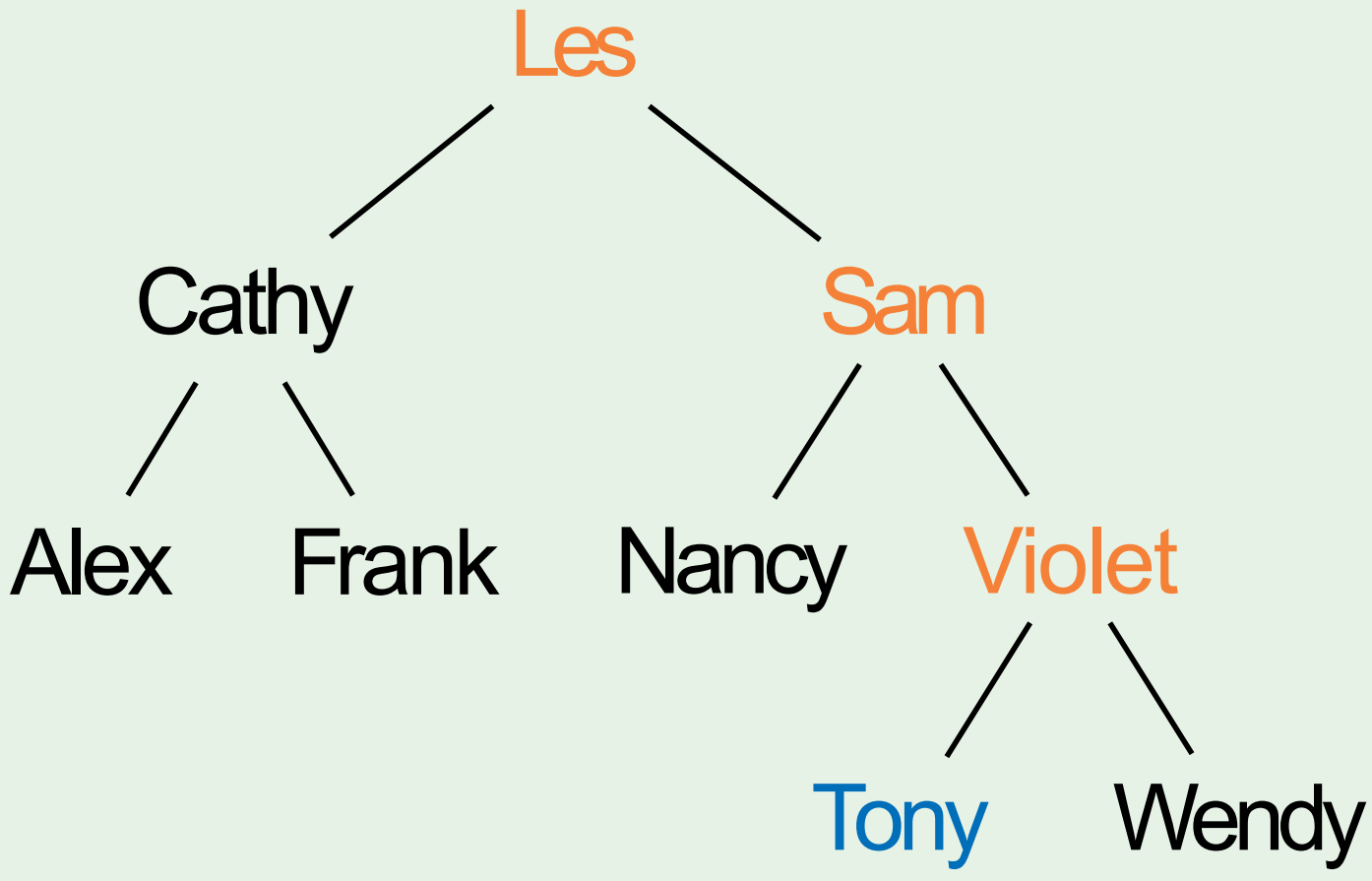
Output: Alex Frank Cathy Nancy

PostOrderTraversal



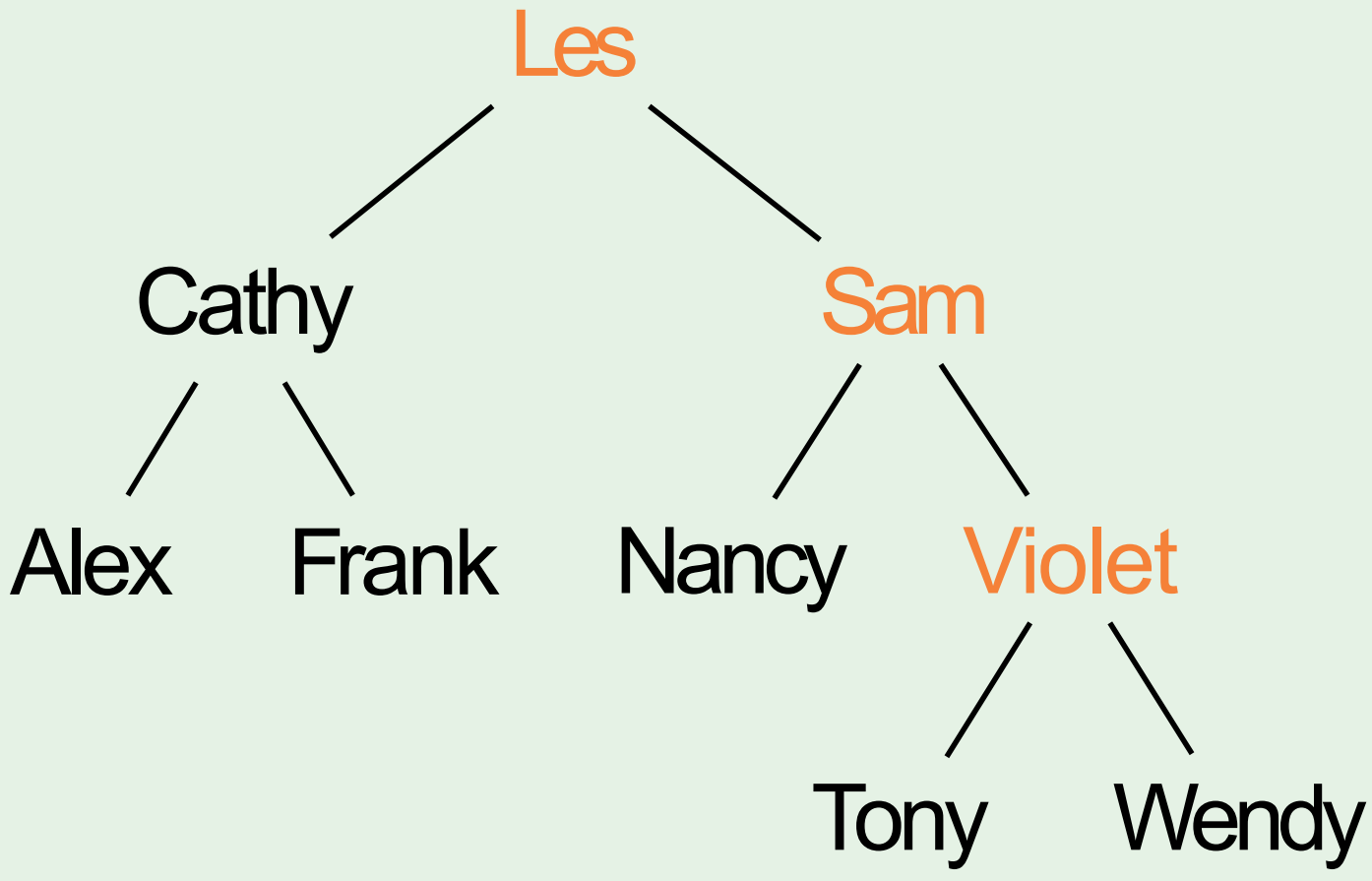
Output: Alex Frank Cathy Nancy Tony

PostOrderTraversal



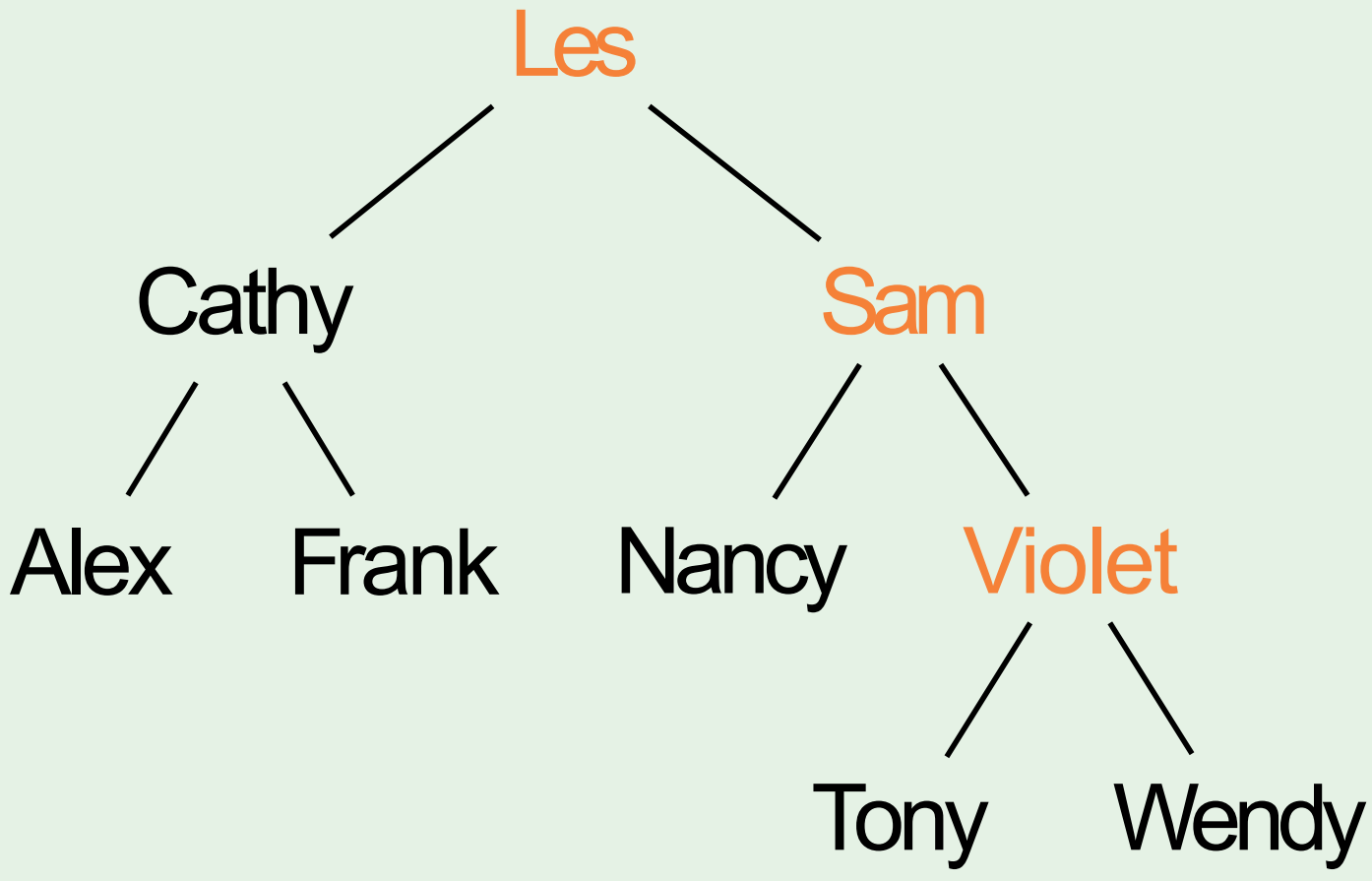
Output: Alex Frank Cathy Nancy Tony

PostOrderTraversal



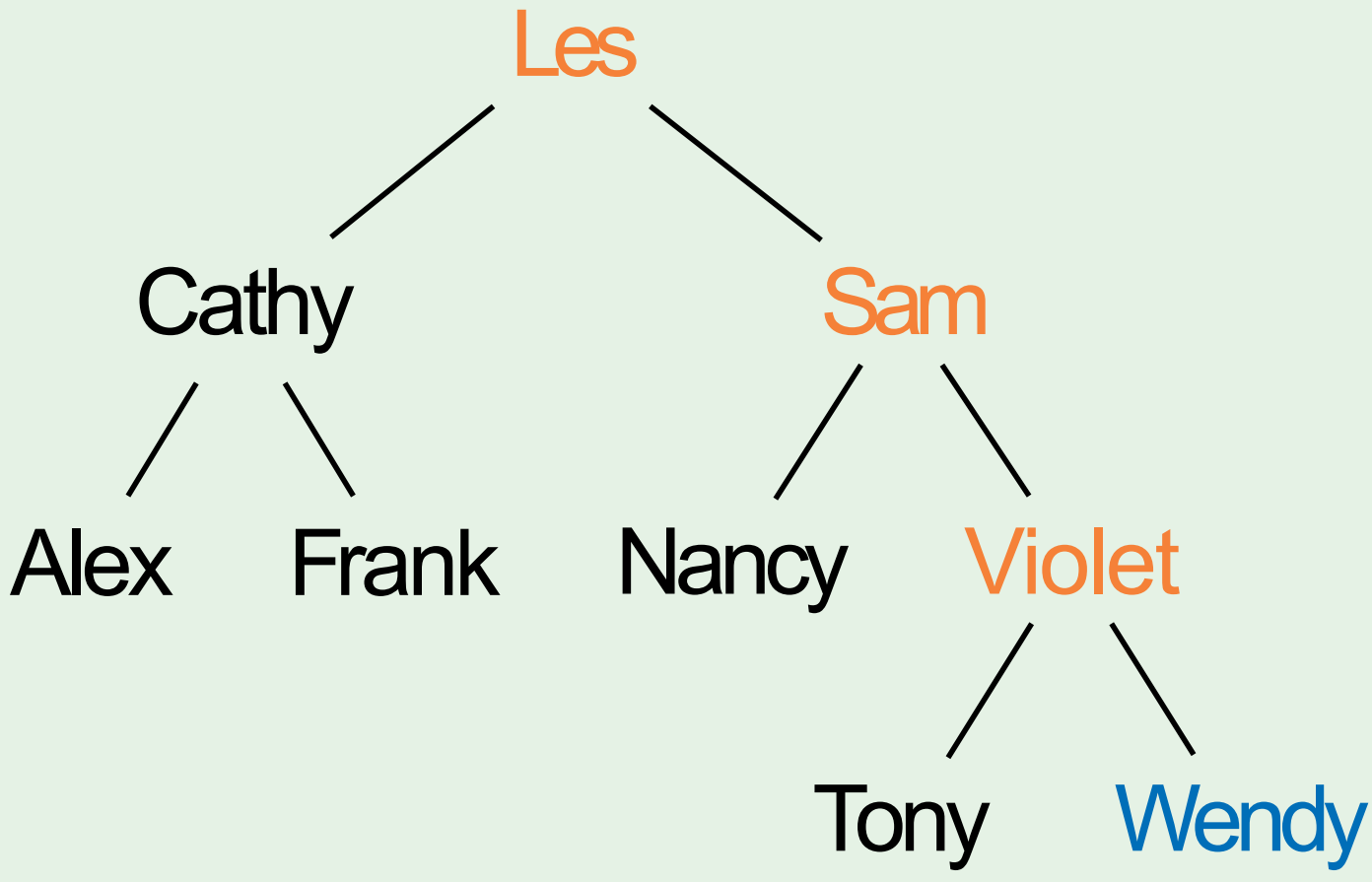
Output: Alex Frank Cathy Nancy Tony

PostOrderTraversal



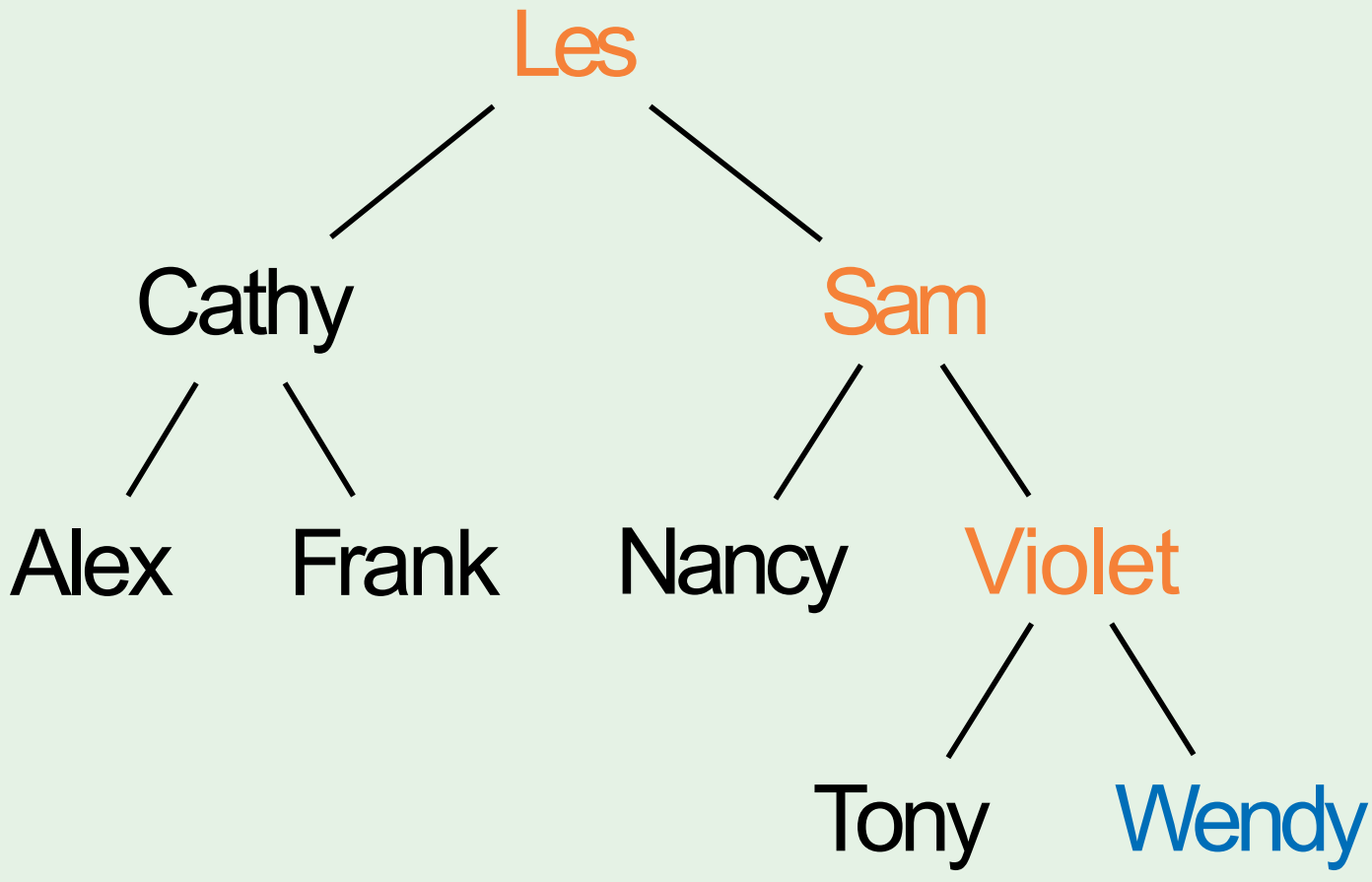
Output: Alex Frank Cathy Nancy Tony

PostOrderTraversal



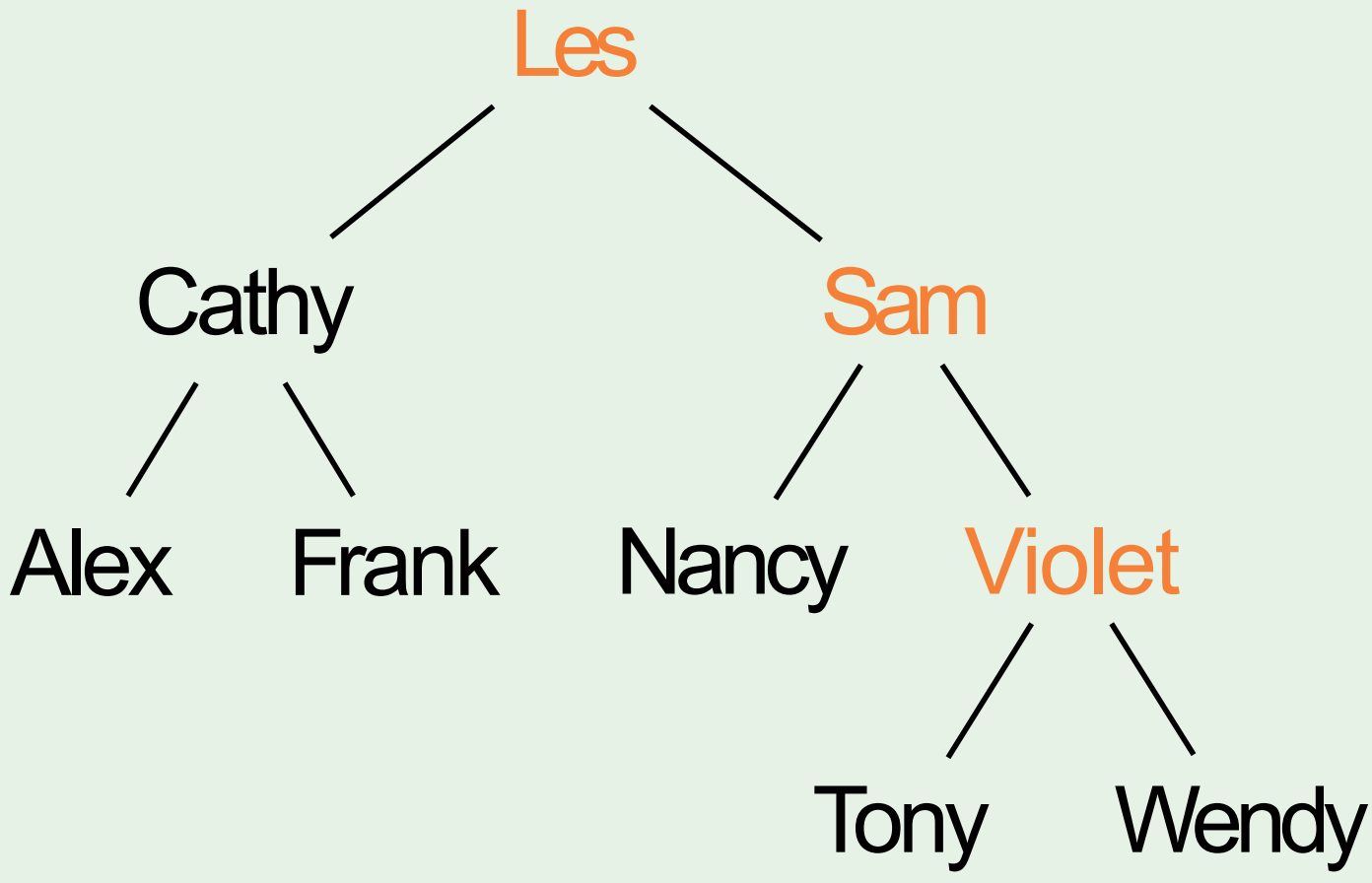
Output: Alex Frank Cathy Nancy Tony Wendy

PostOrderTraversal



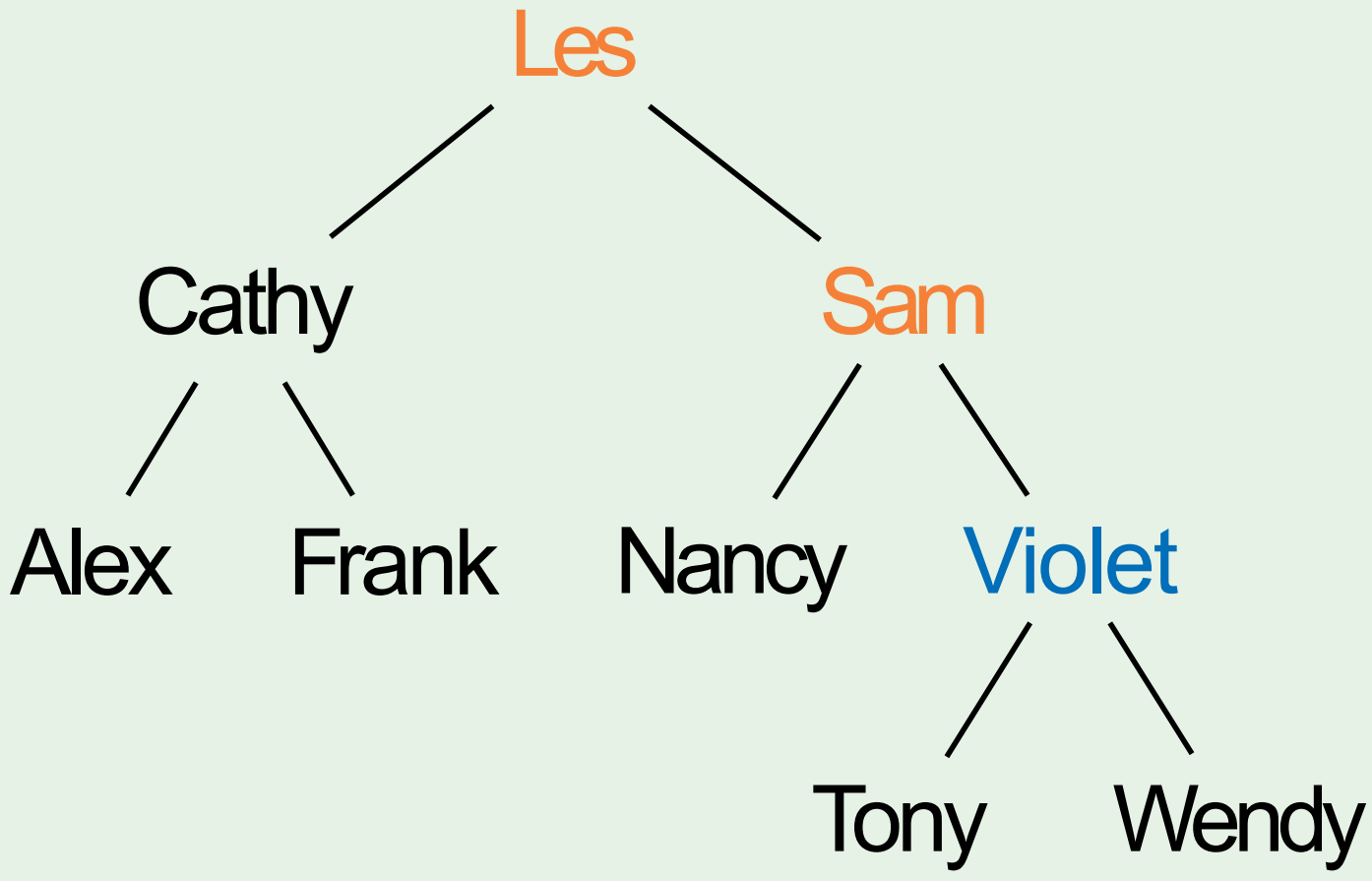
Output: Alex Frank Cathy Nancy Tony Wendy

PostOrderTraversal



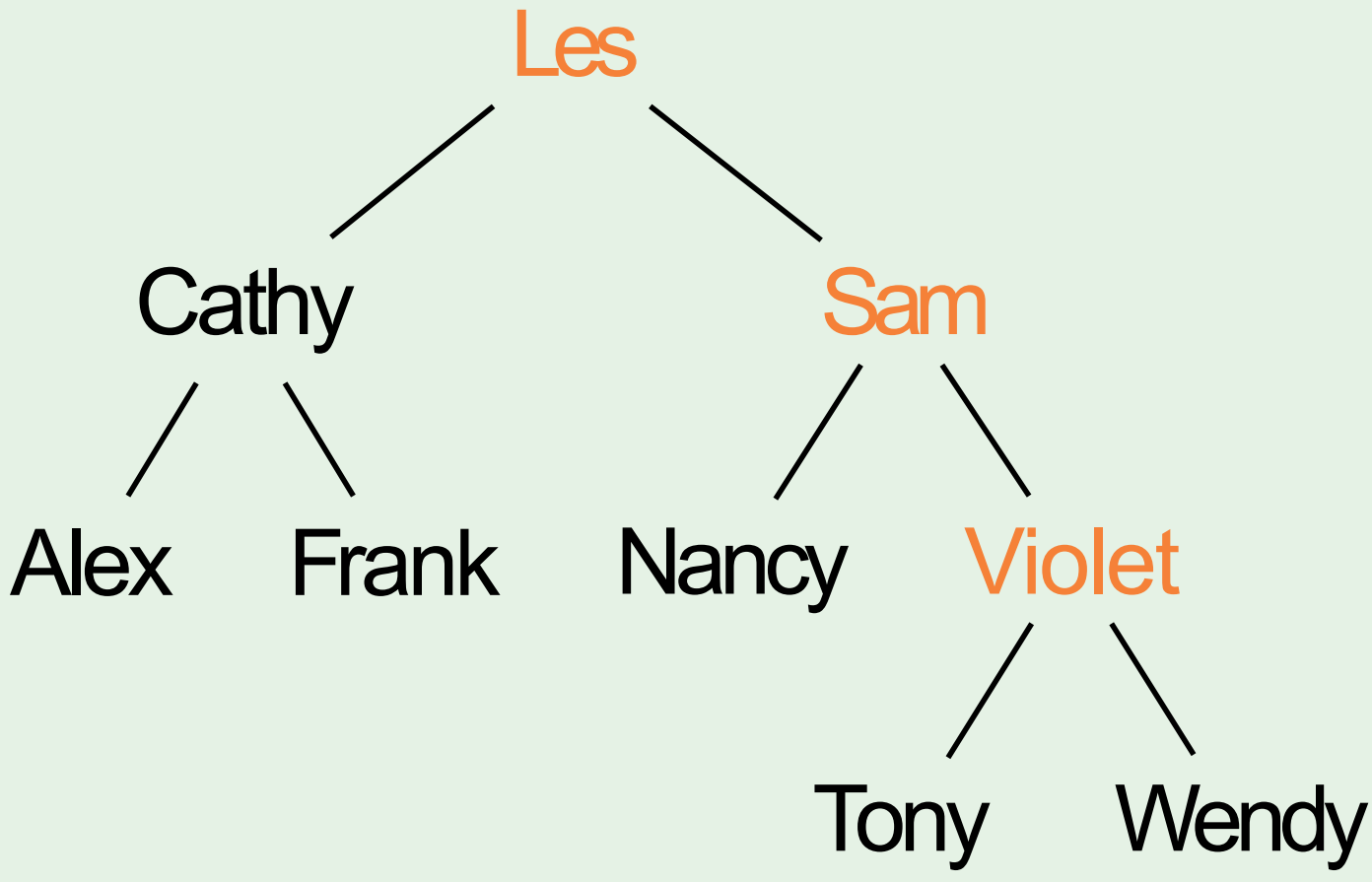
Output: Alex Frank Cathy Nancy Tony Wendy

PostOrderTraversal



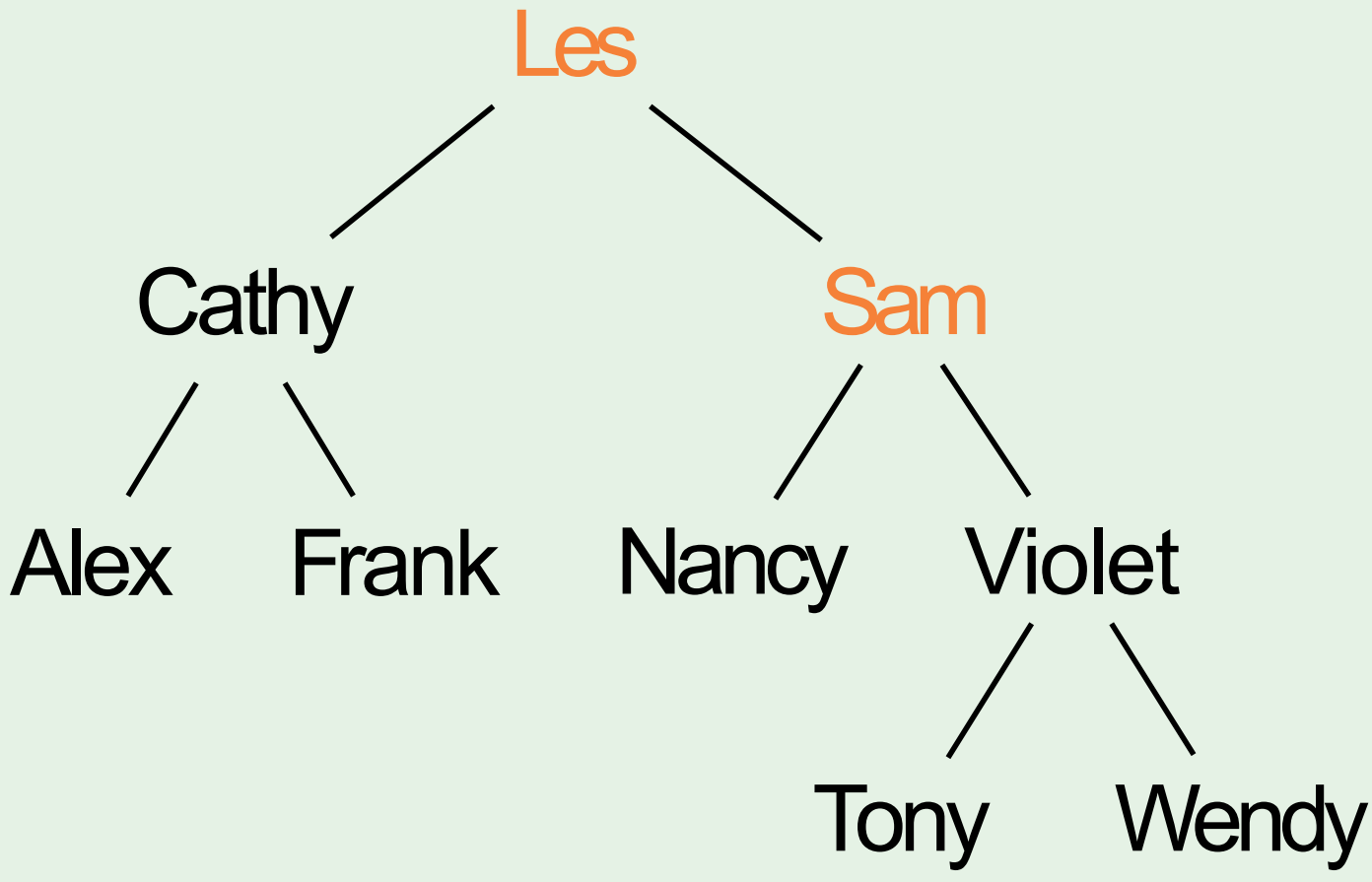
Output: Alex Frank Cathy Nancy Tony Wendy Violet

PostOrderTraversal



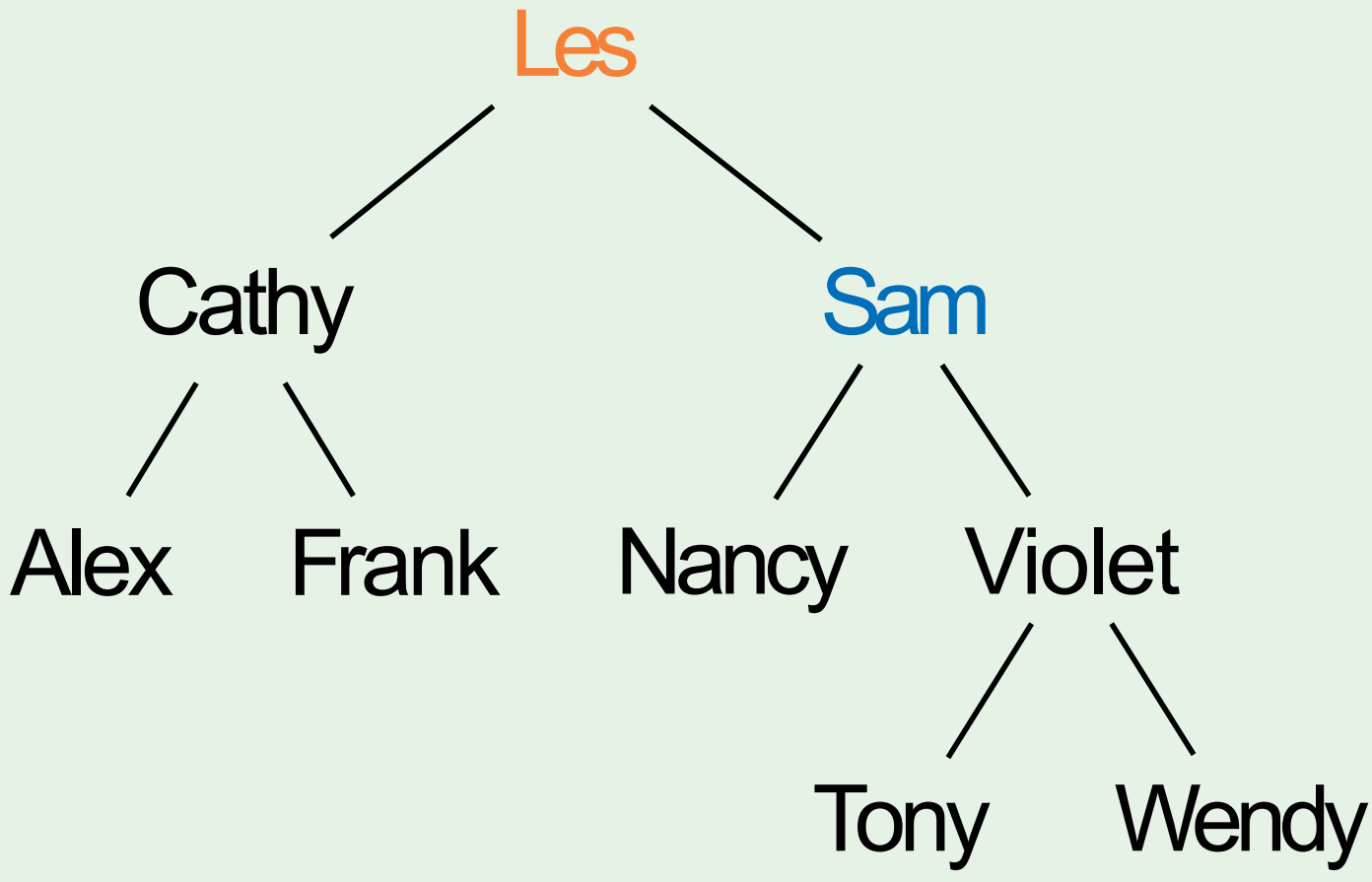
Output: Alex Frank Cathy Nancy Tony Wendy Violet

PostOrderTraversal



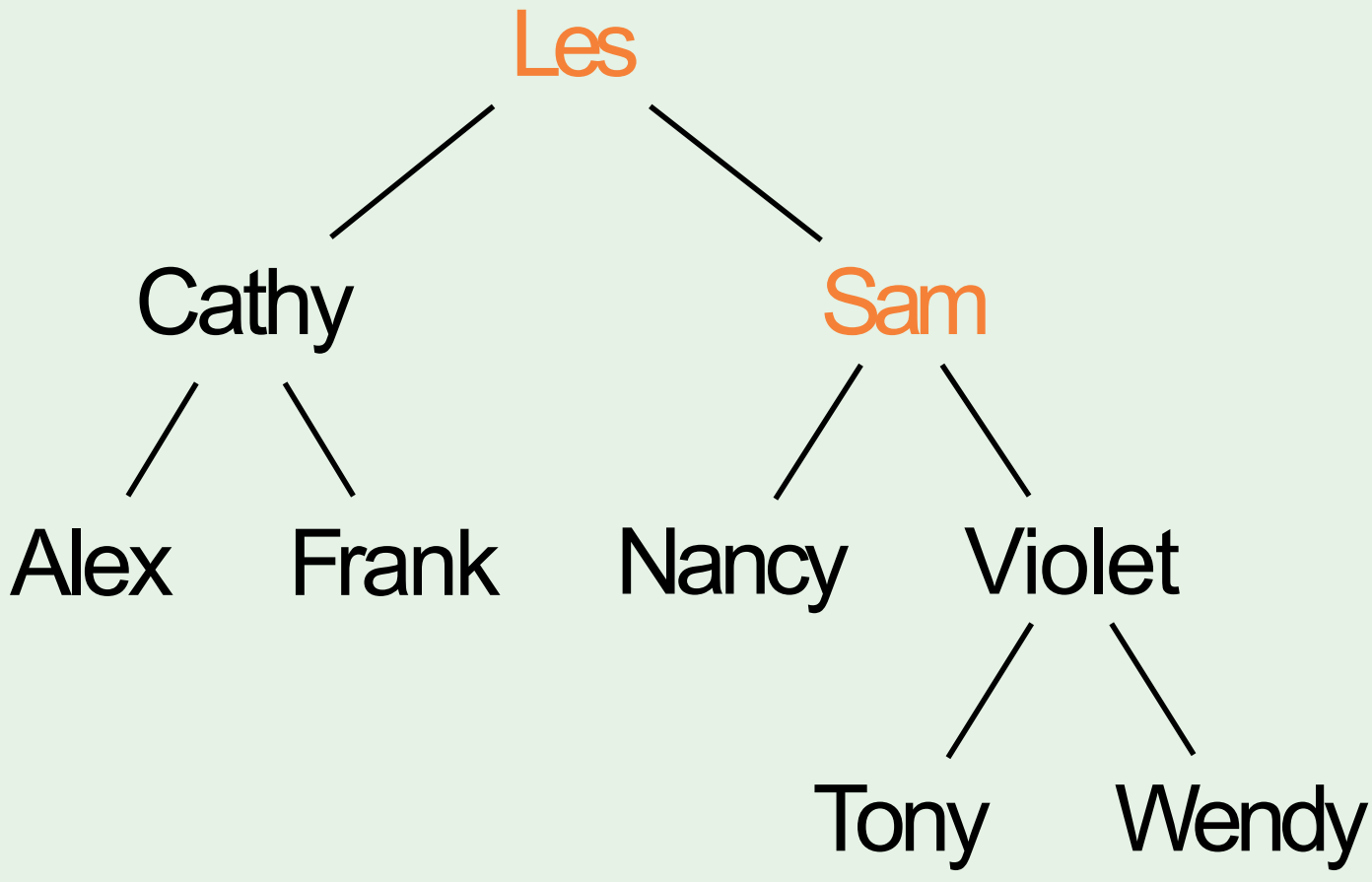
Output: Alex Frank Cathy Nancy Tony Wendy Violet

PostOrderTraversal



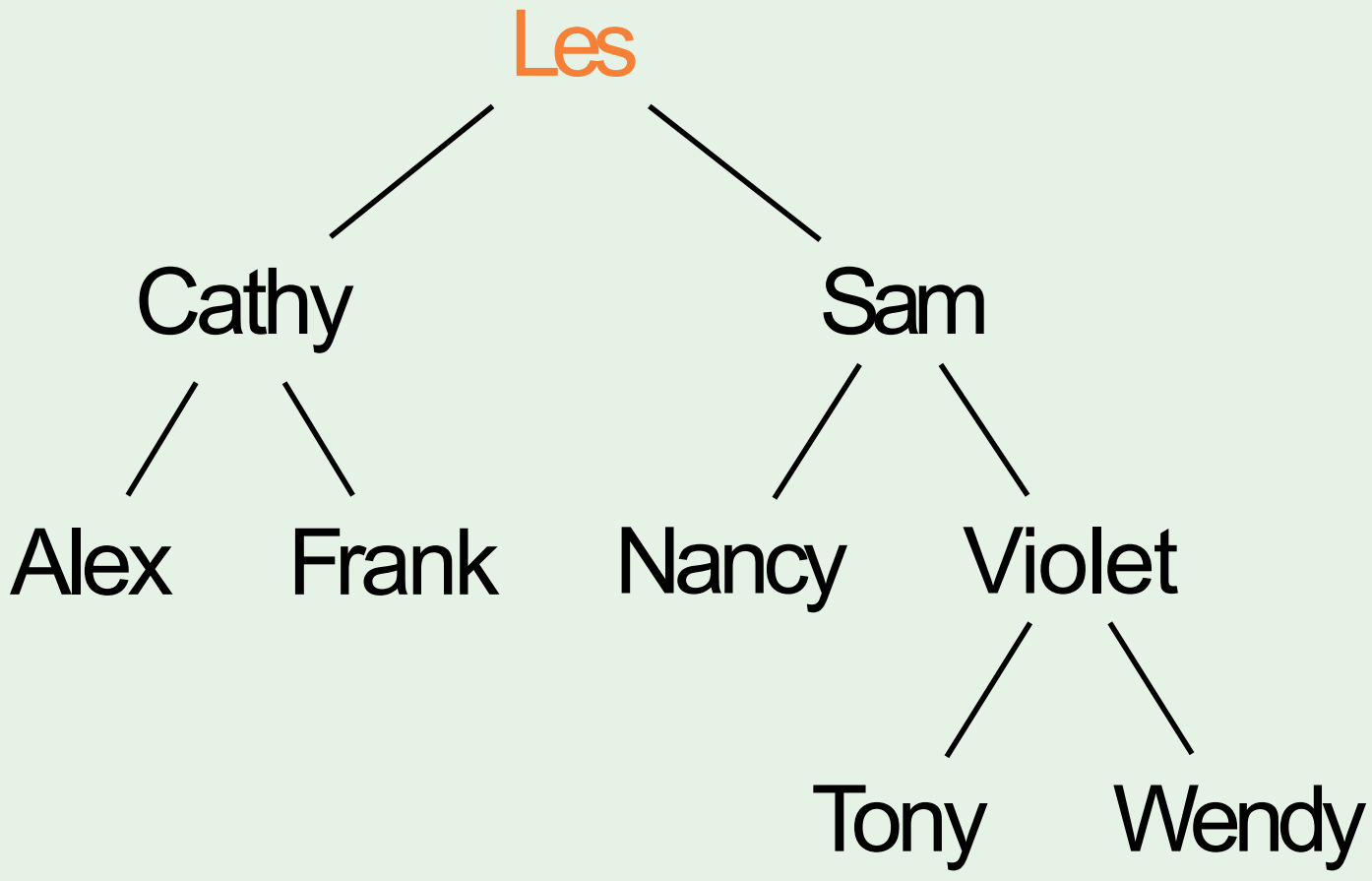
Output: Alex Frank Cathy Nancy Tony Wendy Violet Sam

PostOrderTraversal



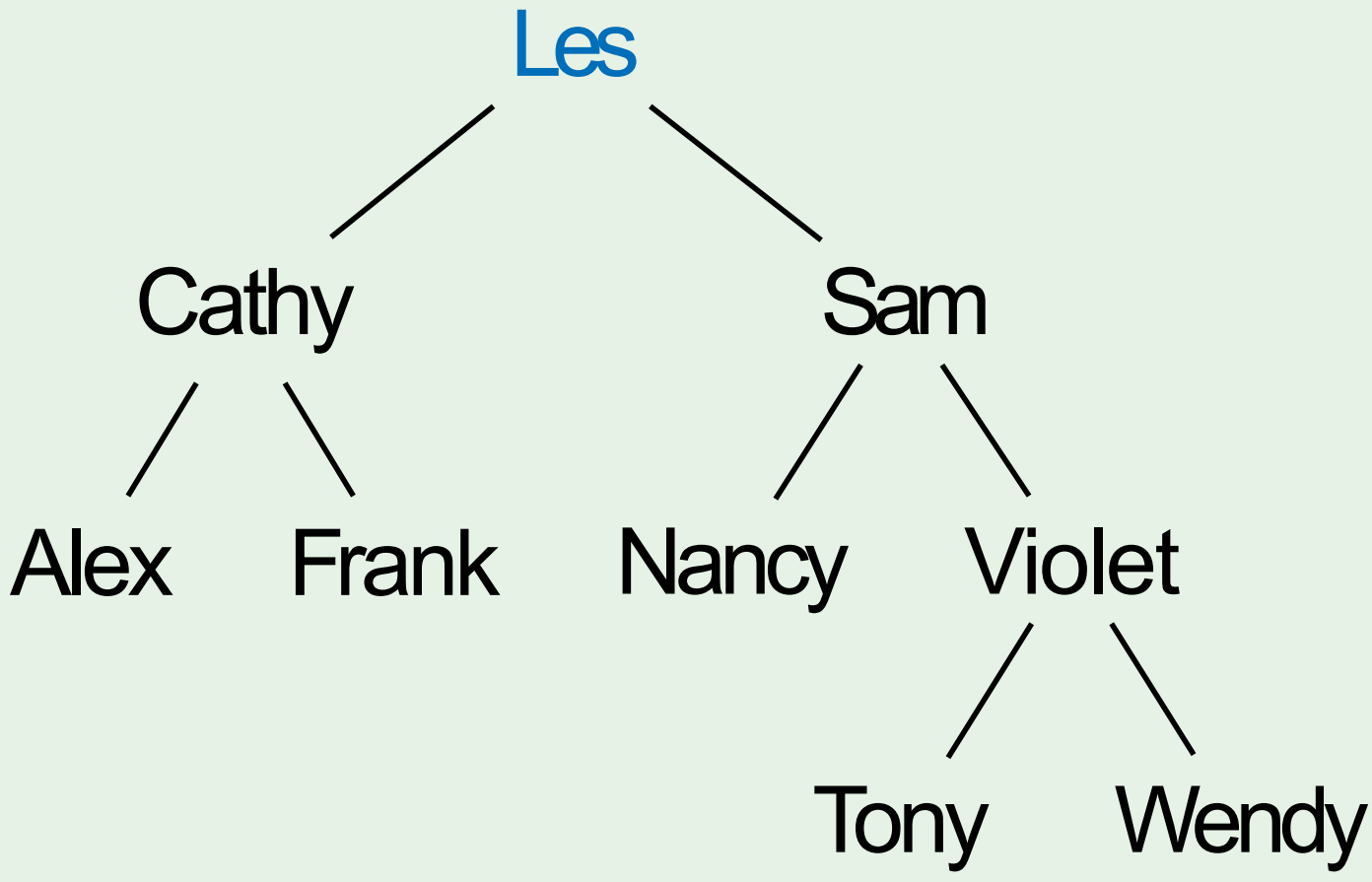
Output: Alex Frank Cathy Nancy Tony Wendy Violet Sam

PostOrderTraversal



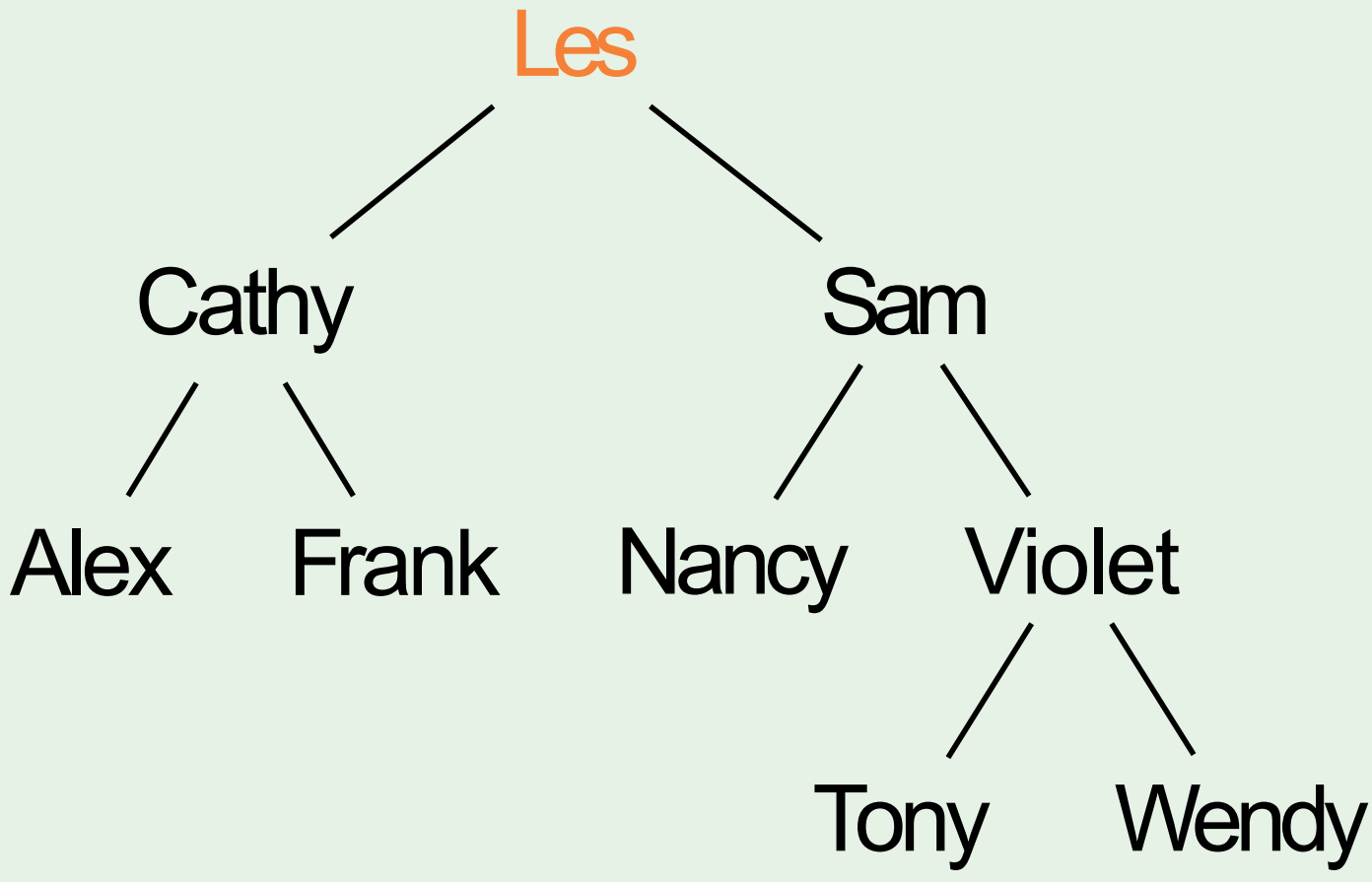
Output: Alex Frank Cathy Nancy Tony Wendy Violet Sam

PostOrderTraversal



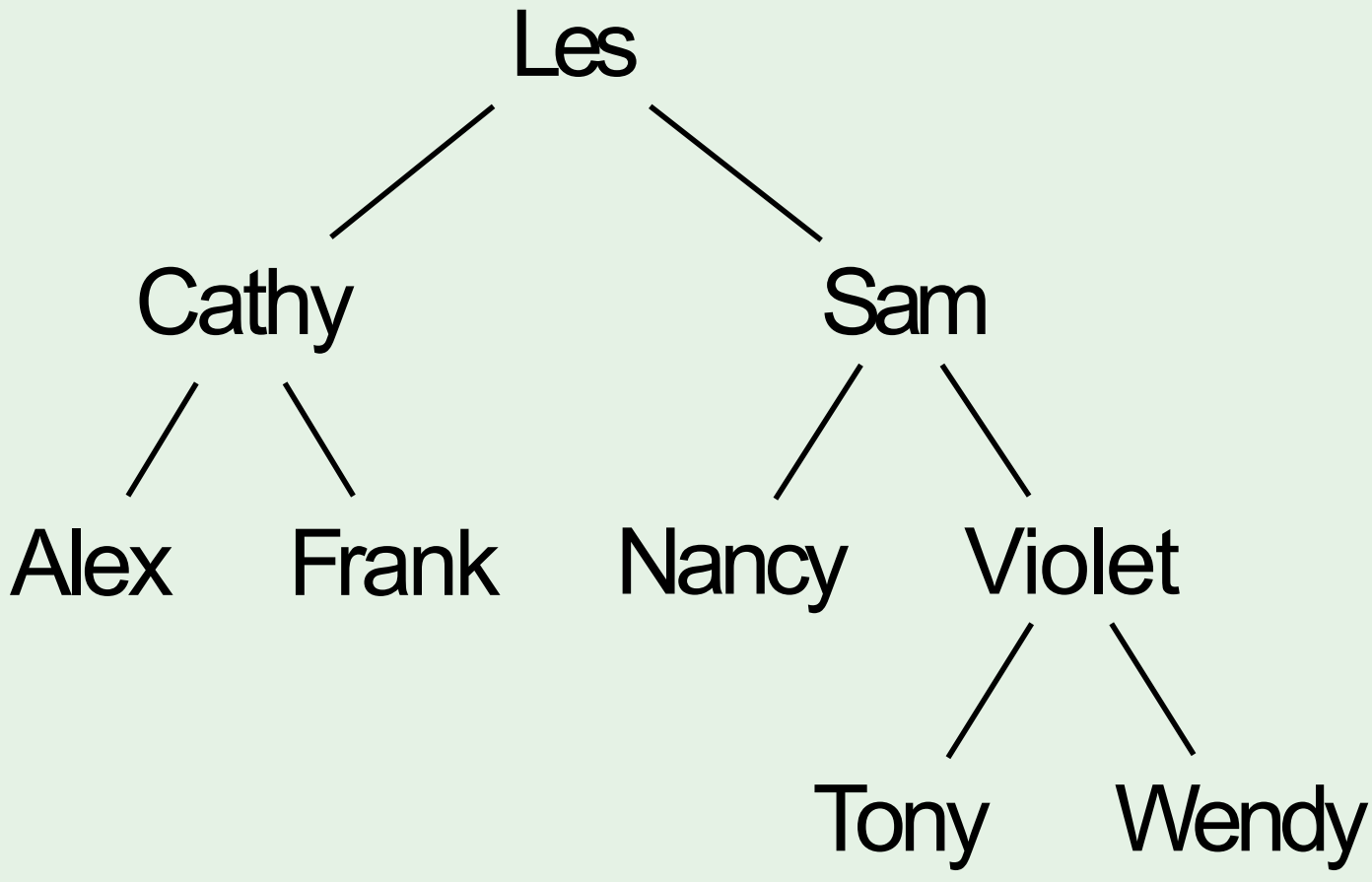
Output: Alex Frank Cathy Nancy Tony Wendy Violet Sam Les

PostOrderTraversal



Output: Alex Frank Cathy Nancy Tony Wendy Violet Sam Les

PostOrderTraversal



Output: Alex Frank Cathy Nancy Tony Wendy Violet Sam Les

Construct a Tree

Inorder sequence: D B E A F C

Can we construct a tree?

Which one is the root node?

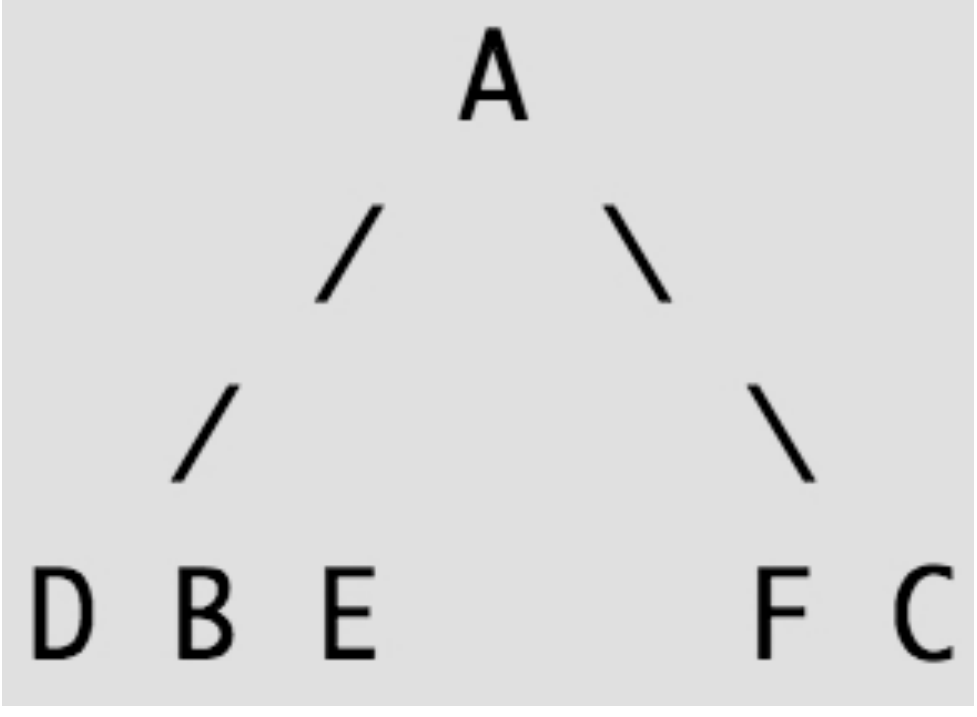
Construct a Tree

Inorder sequence: D B E A F C
Preorder sequence: A B D E C F

- In a Preorder sequence, leftmost element is the root of the tree.
- So we know ‘A’ is root for given sequences.

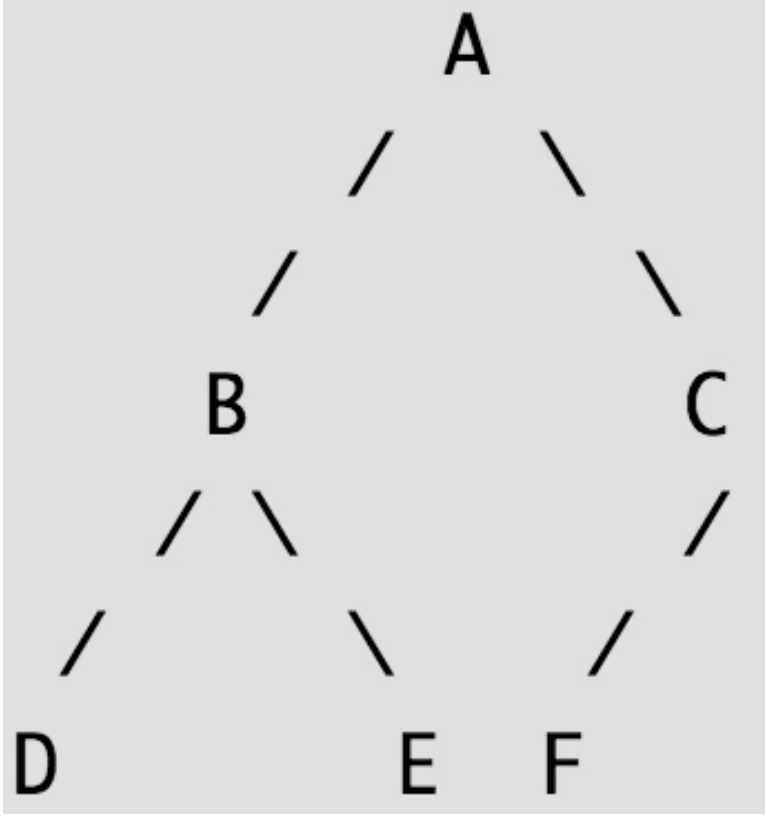
Construct a Tree

Inorder sequence: D B E A F C
Preorder sequence: A B D E C F



Construct a Tree

Inorder sequence: D B E A F C
Preorder sequence: A B D E C F



Construct a Tree

Inorder sequence: D B E A F C
Preorder sequence: A B D E C F

Algorithm: buildTree()

1. First element in preorder[] will be the root of the tree, here its A.
2. Now the search element A in inorder[], say you find it at position i, once you find it, make note of elements which are left to *i* (this will construct the left-subtree) and elements which are right to *i* (this will construct the right-subtree).
3. See this step above and recursively construct left subtree and link it root.left and recursively construct right subtree and link it root.right.

Construct a Tree

Inorder sequence: D B E A F C
Preorder sequence: A B D E C F

```
/* Recursive function to construct binary of size len from
   Inorder traversal in[] and Preorder traversal pre[].  Initial values
   of inStrt and inEnd should be 0 and len -1.  The function doesn't
   do any error checking for cases where inorder and preorder
   do not form a tree */
struct node* buildTree(char in[], char pre[], int inStrt, int inEnd)
{
  static int preIndex = 0;

  if(inStrt > inEnd)
     return NULL;

  /* Pick current node from Preorder traversal using preIndex
    and increment preIndex */
  struct node *tNode = newNode(pre[preIndex++]);

  /* If this node has no children then return */
  if(inStrt == inEnd)
    return tNode;

  /* Else find the index of this node in Inorder traversal */
  int inIndex = search(in, inStrt, inEnd, tNode->data);

  /* Using index in Inorder traversal, construct left and
     right subtress */
  tNode->left = buildTree(in, pre, inStrt, inIndex-1);
  tNode->right = buildTree(in, pre, inIndex+1, inEnd);

  return tNode;
}
```

Construct a Tree

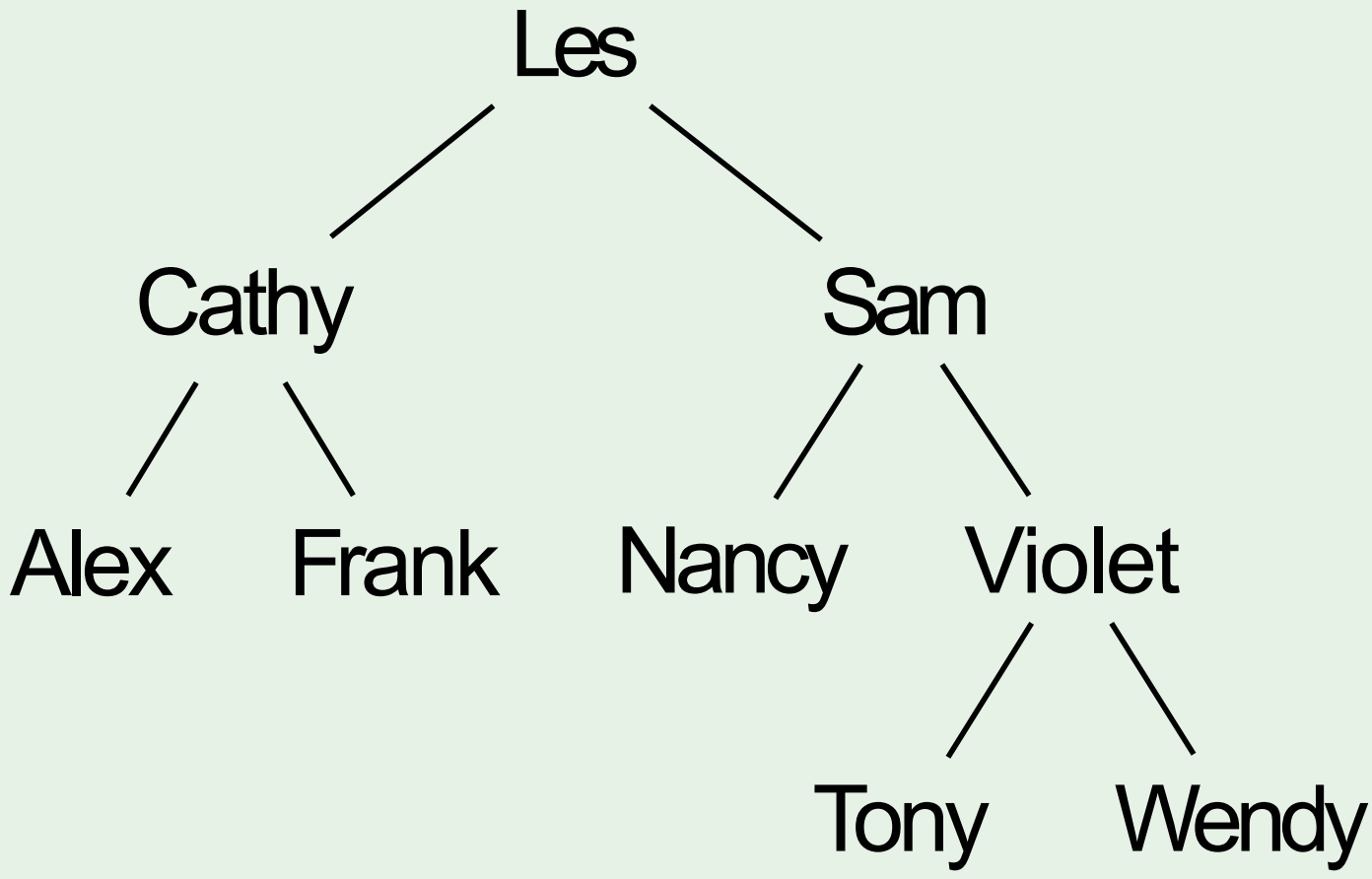
Inorder sequence: D B E A F C
Preorder sequence: A B D E C F

```
/* UTILITY FUNCTIONS */
/* Function to find index of value in arr[start...end]
   The function assumes that value is present in in[] */
int search(char arr[], int strt, int end, char value)
{
  int i;
  for(i = strt; i <= end; i++)
  {
    if(arr[i] == value)
      return i;
  }
}
```

Assignment

- Can you do it for InOrder and PostOrder?
- Validity check - InOrder and PreOrder are given, but tree is not valid!
- Validity check – InOrder, and PostOrder are given, but tree is not valid!

LevelTraversal



Output: Les Cathy Sam Alex Frank Nancy Violet Tony Wendy

Assignment

- Can you do it for InOrder and PostOrder?
- Validity check - InOrder and PreOrder are given, but tree is not valid!
- Validity check – InOrder, and PostOrder are given, but tree is not valid!
- Implement Breadth-First traversal