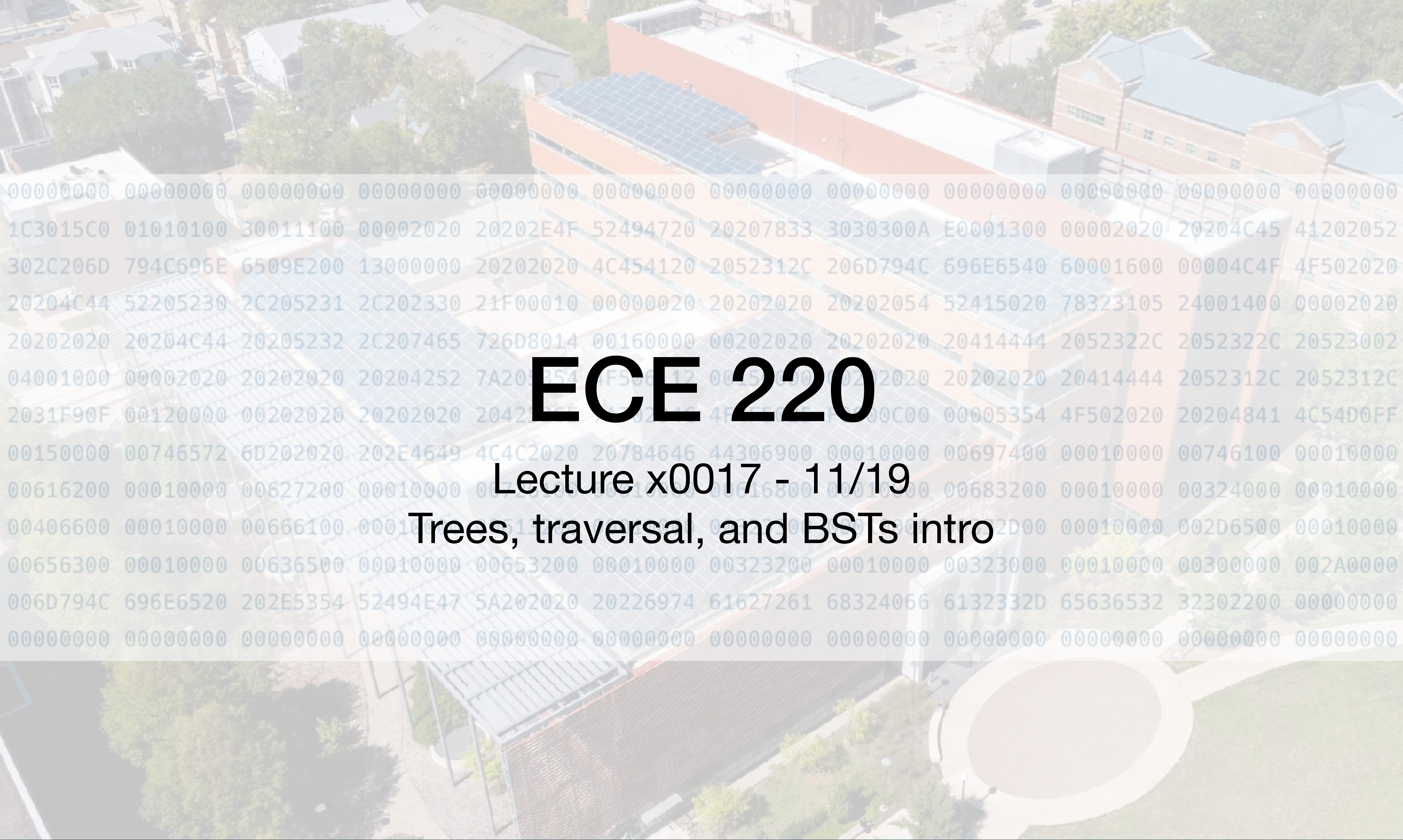


ECE 220

Lecture x0017 - 11/19

Trees, traversal, and BSTs intro

Recap

- Last week
  - OOP Concepts
    - Constructors, destructors, etc.
    - Inheritance, polymorphism, etc.
- Templates
  - Template functions
  - Template classes
- Template library
  - Containers: lists vs. vectors
  - Iterators

Recap

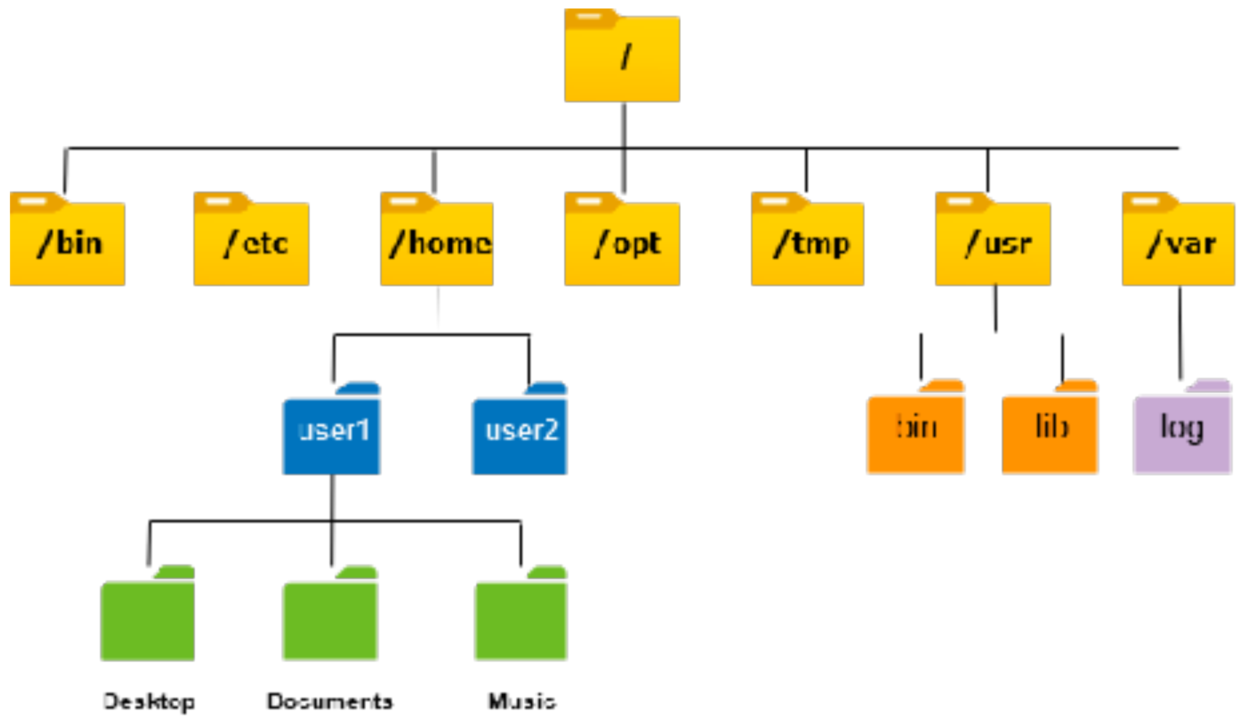
- Scan QR Code

New concept - trees

- Recall linked lists
  - Singly linked lists
  - Doubly linked lists
- Linked lists, queues, stacks: *linear* data structures

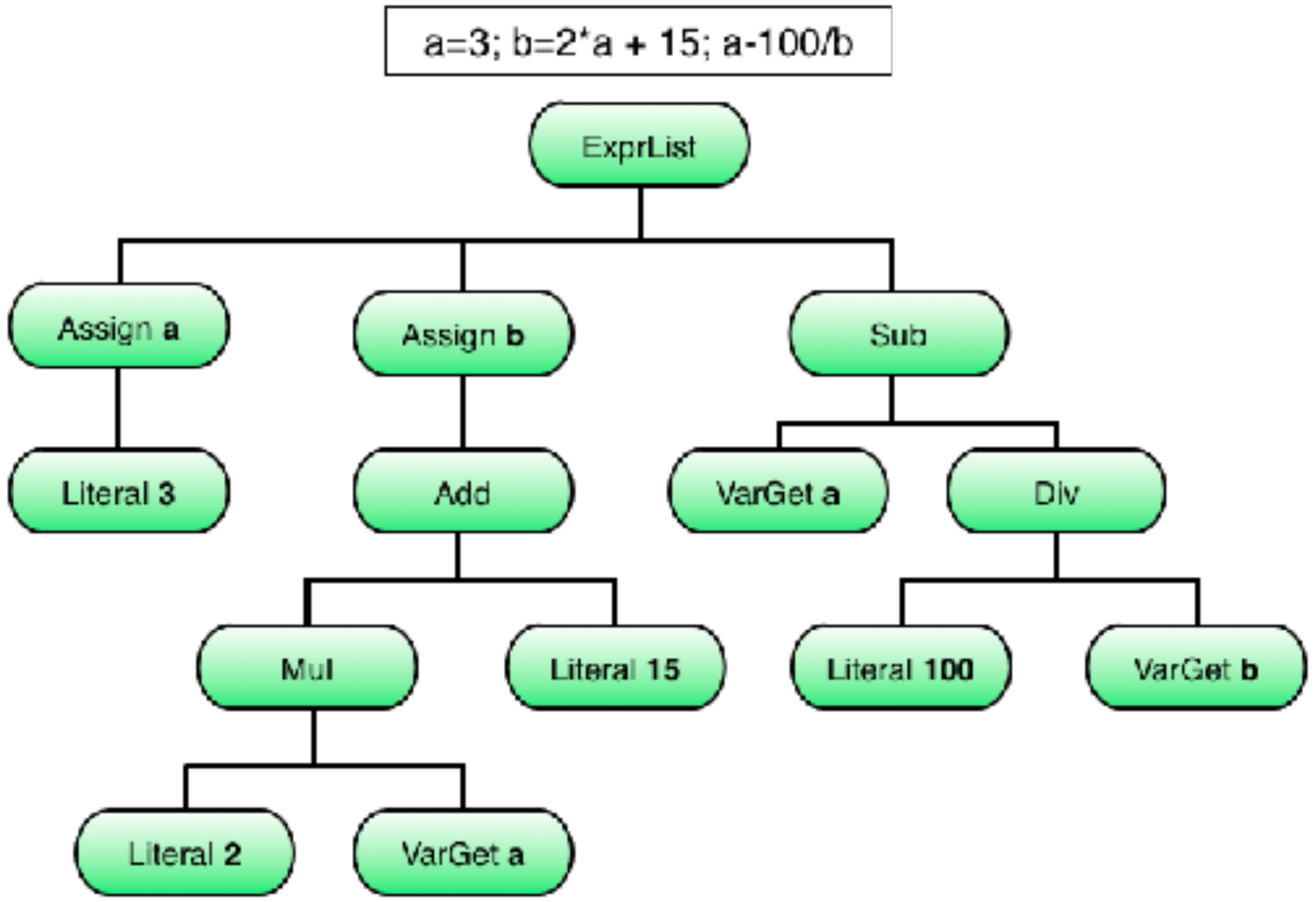
- Trees - are *nonlinear* & hierarchical
  - Think family trees or organizational charts
  - Basic unit ~ node in DLL
  - Difference - functions.

Why trees?



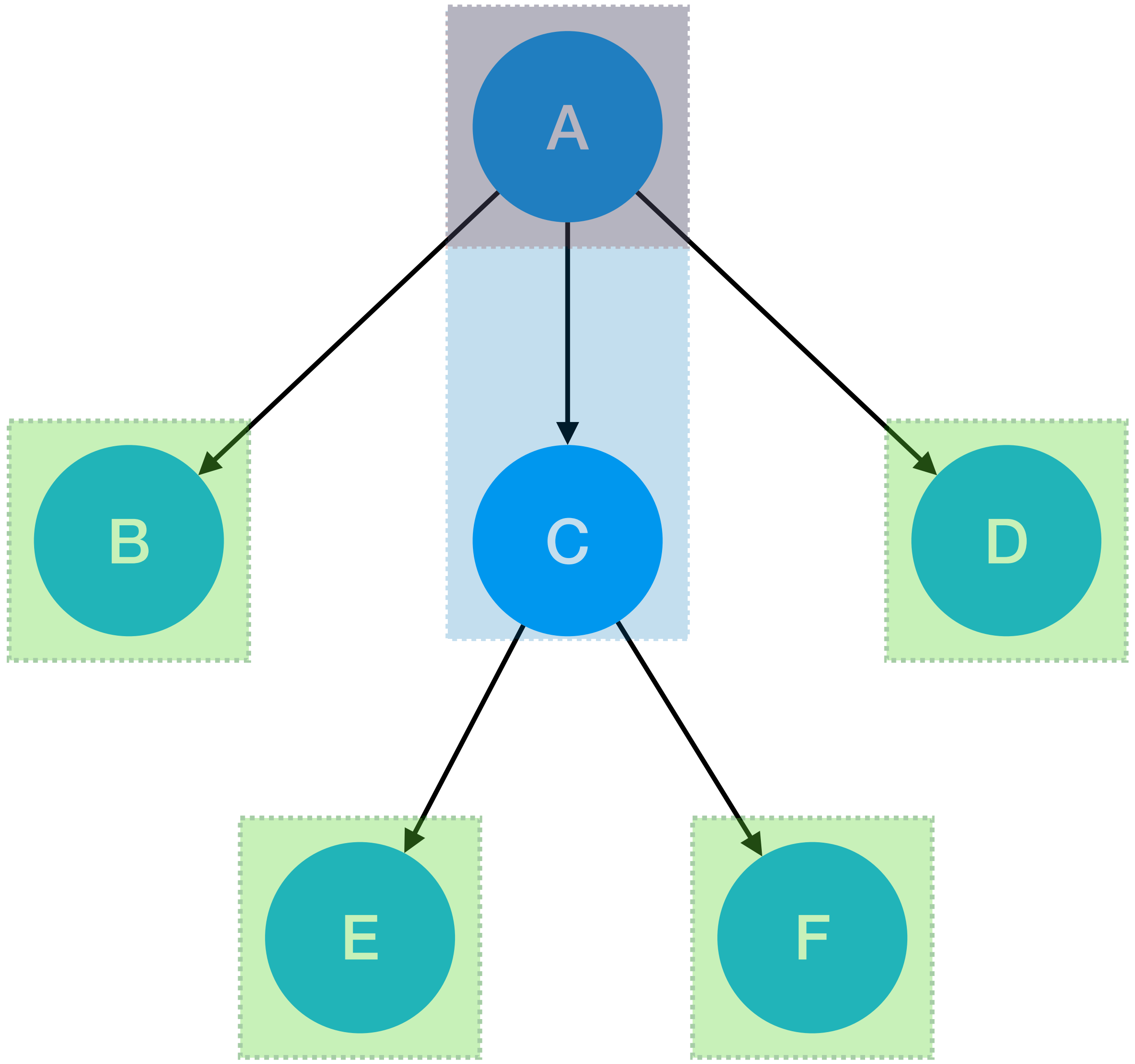
Filesystems, computer graphics, programming languages, taxonomic classifictaion, etc.

QuadTree: https://en.wikipedia.org/wiki/Quadtree



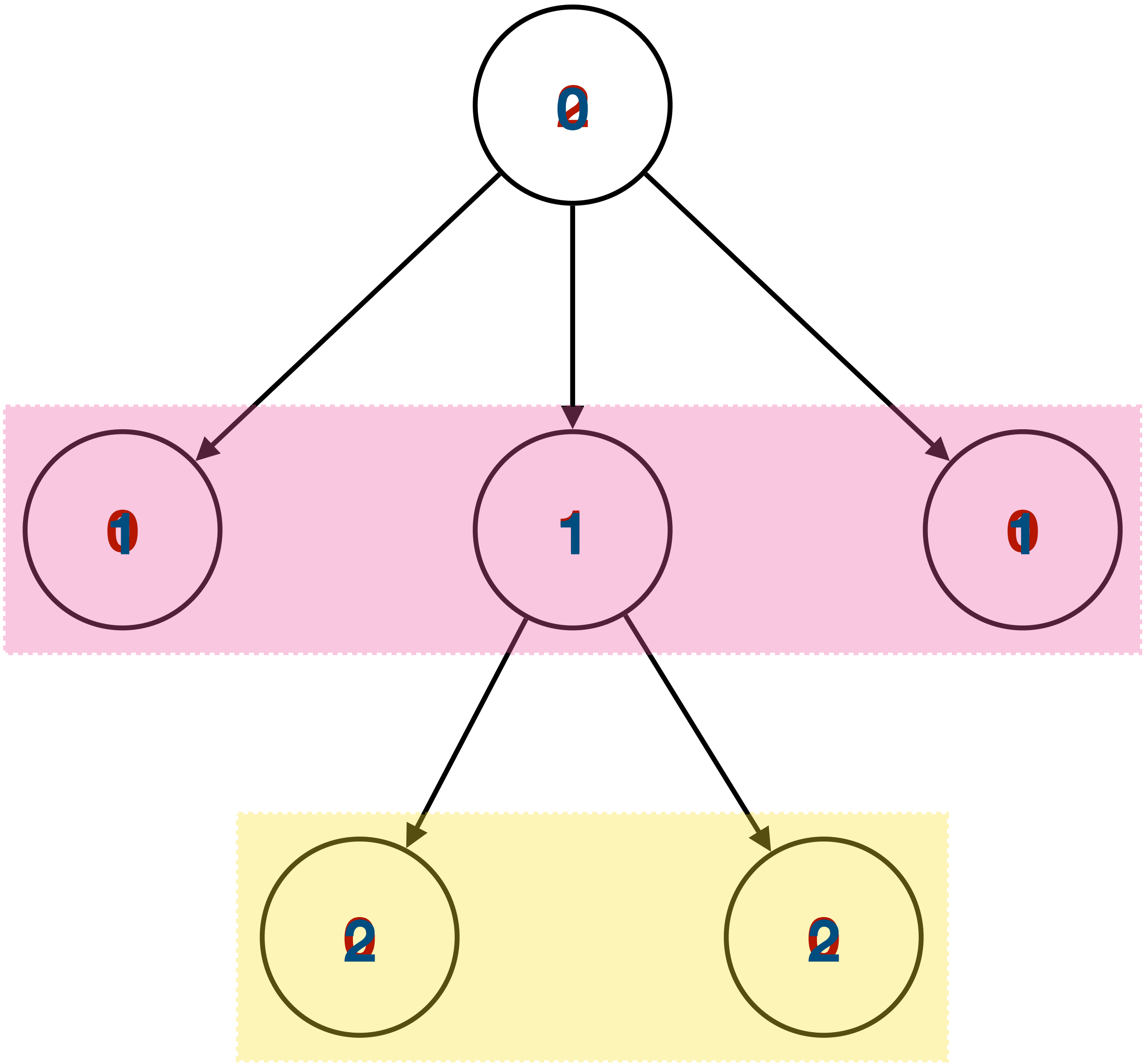
Concepts related to trees

- Root
  - Top most node, no parent.
- Leaf
  - Outermost nodes, no children
- Inner node(s)
  - Has atleast one child



Concepts related to trees

- Siblings
- Height (of a node)
    - Length of *longest* path from given node to a leaf
- Depth (of a node)
    - Length of path from root to given node

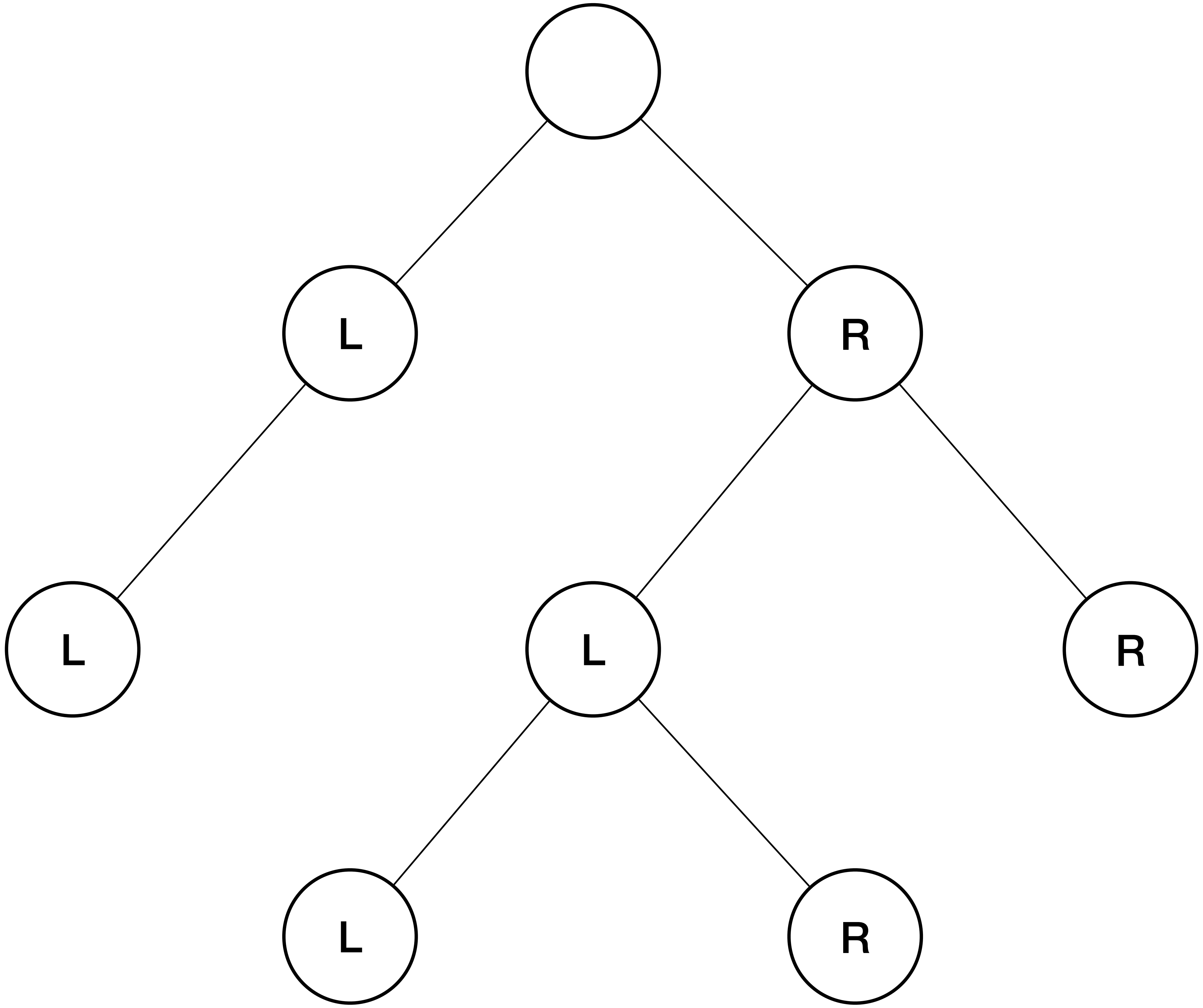


Binary trees

- Trees where every node has *at most* two children.

```
typedef struct person node;
struct person{
  char *name;
  node *next;
  node *prev;
}node;

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



Traversing trees

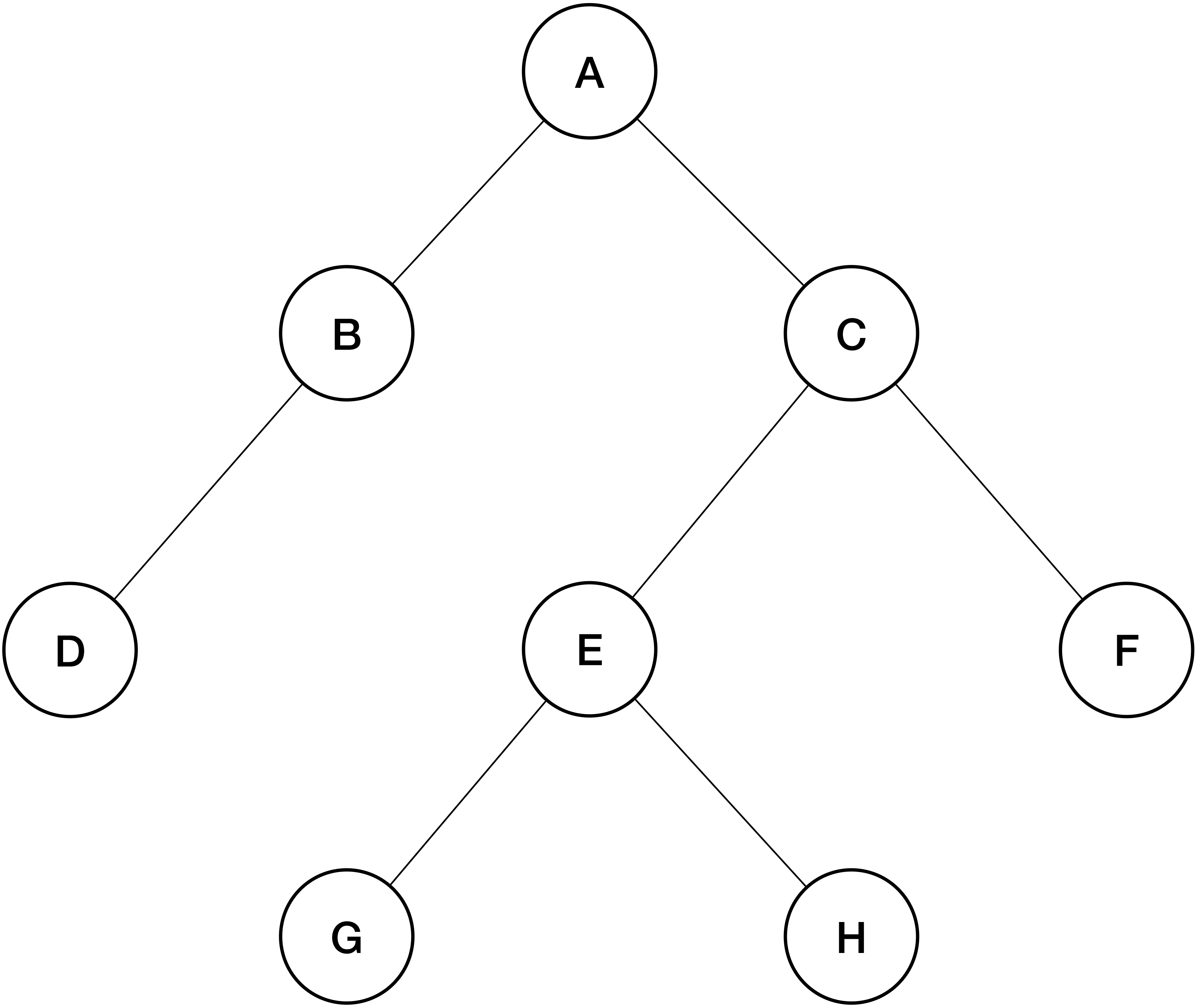
- You can traverse trees in three ways
  - Pre-order
    - Root, Left, Right
  - In-order
    - Left, Root, Right
  - Post-order
    - Left, Right, Root

For each node, read the data of the **node**, then visit the **left** subtree and then the **right** subtree.

For each node, visit the **left** subtree, then read the data of the **node**, then visit the **right** subtree.

For each node, visit the **left** subtree, then visit the **right** subtree, then read the data of the **node**.

Traversing trees



For each node, read the data of the **node**, then visit the **left** subtree and then the **right** subtree.

A → B → D → C → E → G → H → F

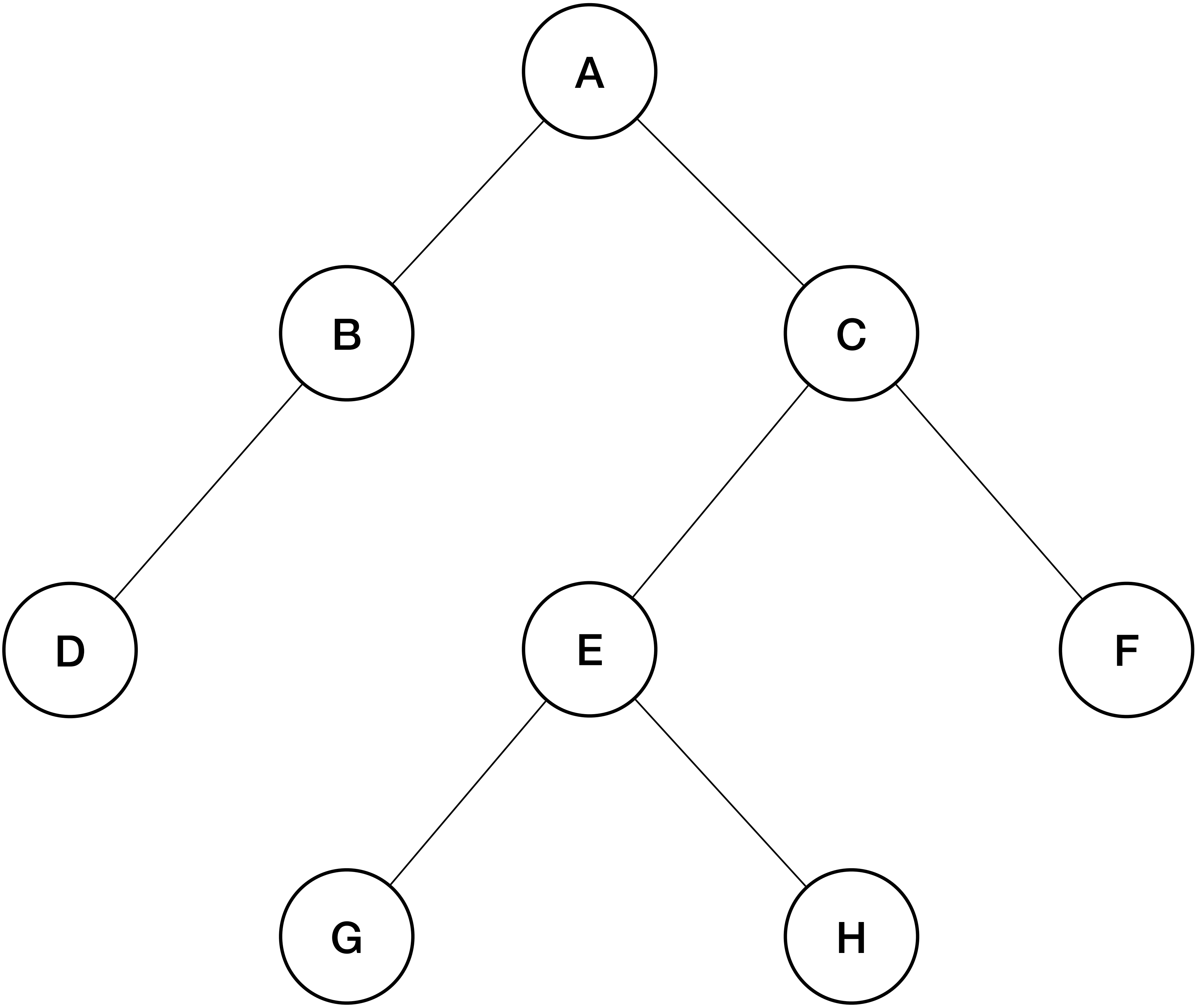
For each node, visit the **left** subtree, then read the data of the **node**, then visit the **right** subtree.

D → B → A → G → E → H → C → F

For each node, visit the **left** subtree, then visit the **right** subtree, then read the data of the **node**.

D → B → G → H → E → F → C → A

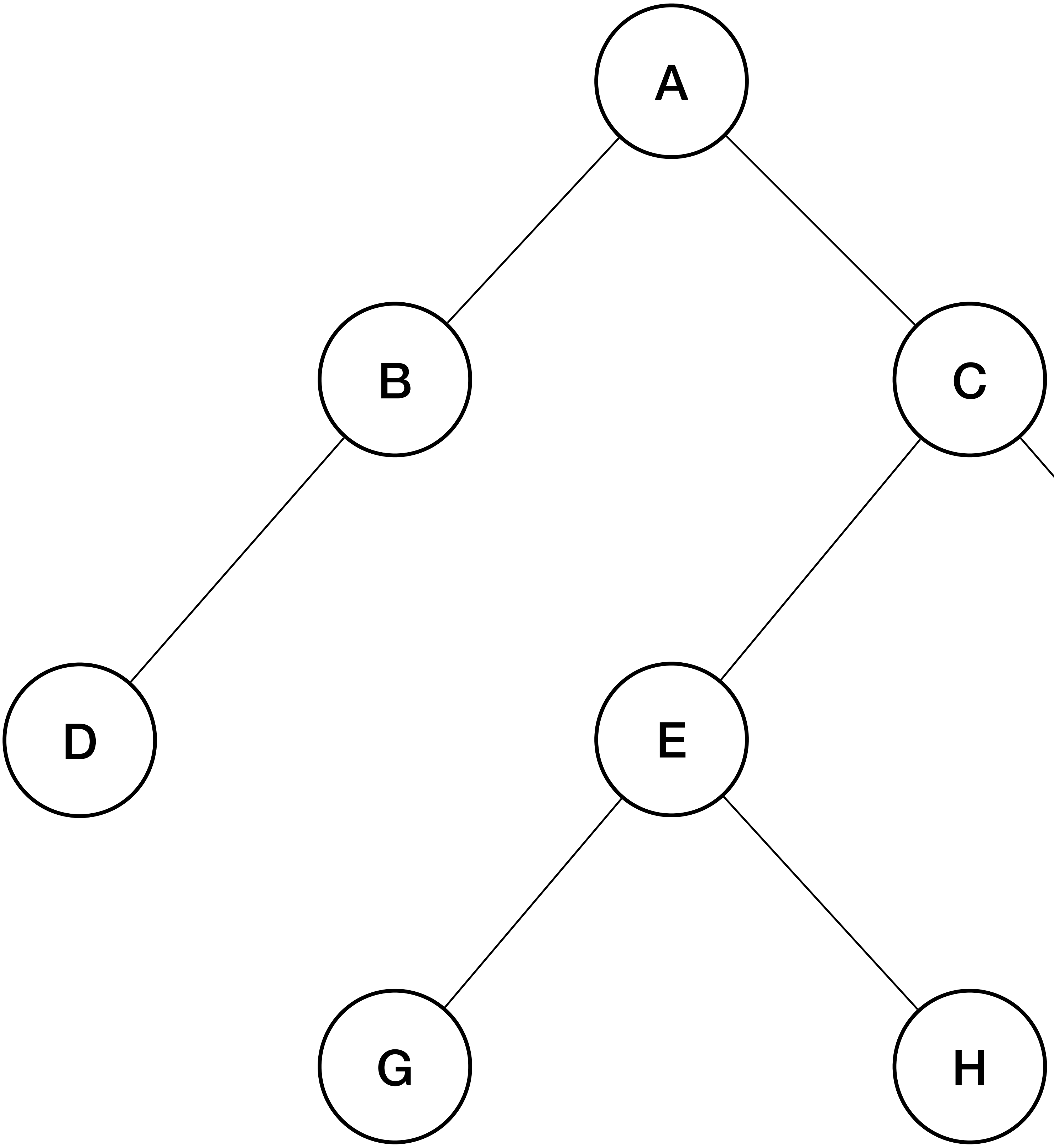
Traversing trees



- The previous are called **depth-first** traversals. Could also do a **breadth-first** traversal.
  - Traverse through all the children of a node, then visit the grandchildren.

A → B → C → D → E → F → G → H

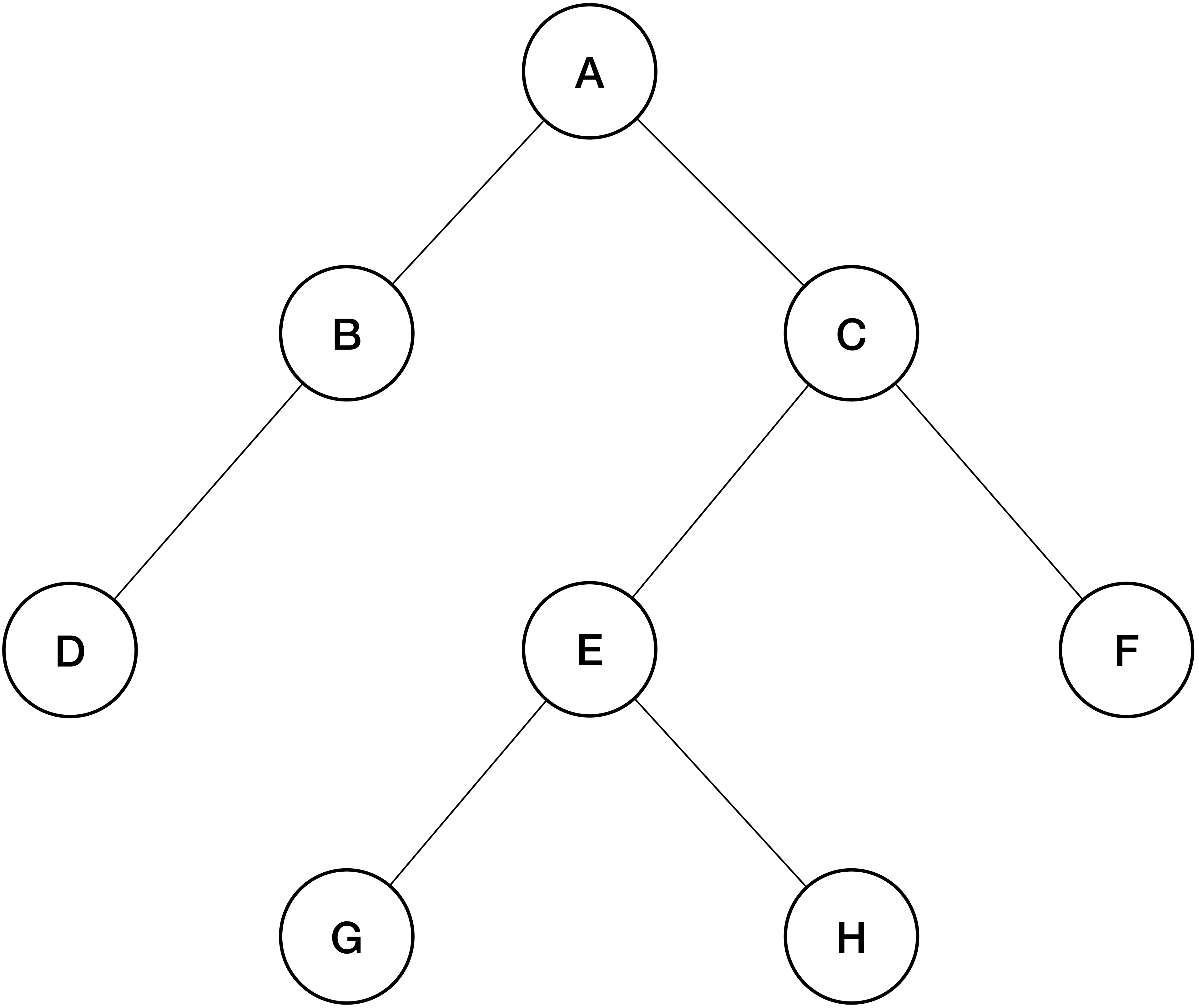
Implementing traversals



```
Stack myStack
myStack.push(root)
while(myStack){
    cursor = myStack.pop()
    cursor.print()
    if (cursor->right)
        myStack.push(cursor->right)
    if (cursor->left)
        myStack.push(cursor->left)
}
```

What does this algorithm do?

Implementing traversals



```
Queue myQueue
myQueue.enqueue(root)
while(myQueue){
    cursor = myQueue.dequeue()
    cursor.print()
    if (cursor->left)
        myQueue.enqueue(cursor->left)
    if (cursor->right)
        myQueue.enqueue(cursor->right)
}
```

What does this algorithm do?

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

Practice time

- Write functions to:
  - Add to the left and right of a node.
  - Implement preorder, inorder, and postorder traversals.
  - Delete a tree.

```
int main(){
  int arr[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  node * root = (node *) malloc(sizeof(node));
  root->data = arr[0];
  root->left = NULL;
  root->right=NULL;
  node * cursor = root;

  for (int j=0, i=1; i<11; i=i+2, j++){
    add_left(&cursor, arr[i]);
    add_right(&cursor, arr[i+1]);
    cursor = (j%2==0) ? cursor->right : cursor->left;
  }
  print_preorder(root);
  print_inorder(root);
  print_postorder(root);
  delete_tree(root);
}
```

Printing a tree

- Can we print a tree in a human readable way?
  - Focus on pre-order traversal
    - Print node, then go left, then go right
  - Use *depth* to print right amount of indentation

```
void treeprint(node *cursor, int depth){
  if (cursor == NULL)
    return;
  for (int i = 0; i < depth; i++)
    printf(i == depth - 1 ? "|-" : "  ");
  printf("%d\n", cursor->data);
  treeprint(cursor->left, depth + 1);
  treeprint(cursor->right, depth + 1);
}
```

Let us check if we got previous slide right …