CSE 12 Week 9 Discussion

3-2-21

Focus: Heaps & BST Traversal

Reminders

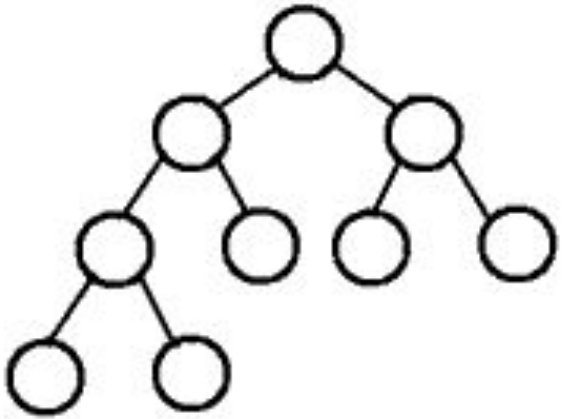
- PA8 **(open!)** due Thursday, March 11th @ 11:59 PM
  - No resubmission

- PA6 Resubmission due Friday, March 5th @ 11:59 PM

- PA7 Resubmission due Friday, March 12th @ 11:59 PM

Heaps

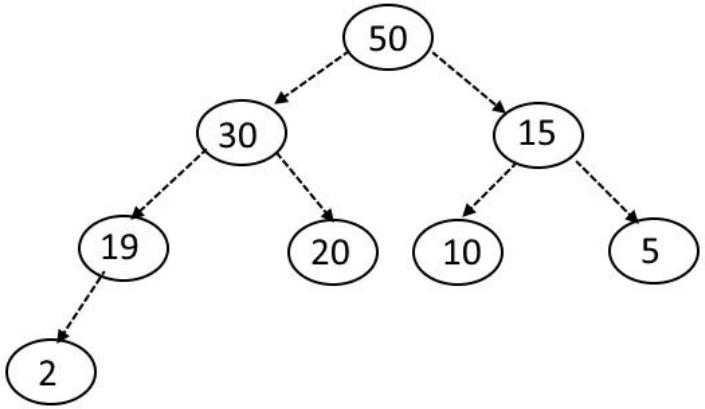
- A heap is a **complete** tree
  - Every level is full except possibly the last, and all nodes are as far left as possible.
- It might not necessarily be a **full** tree
  - Every node other than the leaves have two children



complete tree

Heaps

- Implemented with a list
- min/max heap
  - useful when we care about the next largest/smallest value
- Can we start at 0?



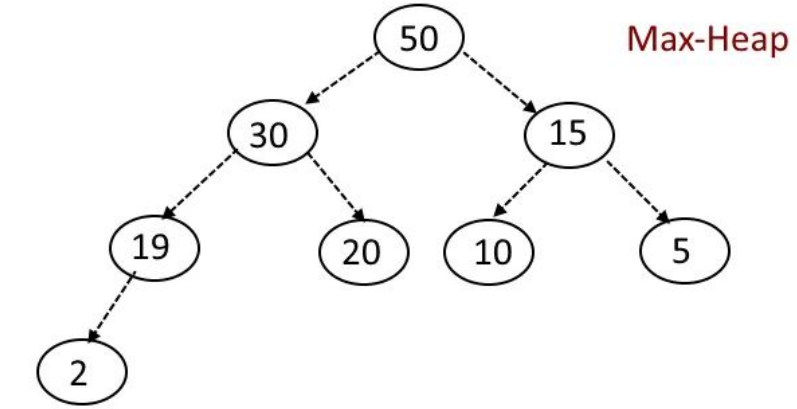
0	1	2	3	4	5	6	7	8
	50	30	15	19	20	10	5	2

for Node at i : Left child will be 2i and right child will be at 2i+1

Question

How can I get the parent node of the following heap?

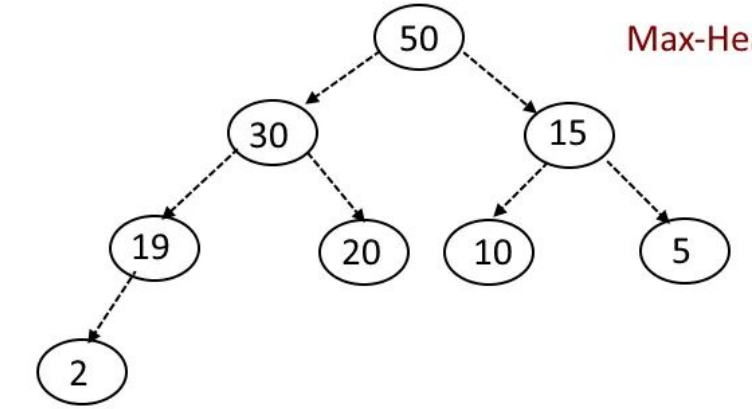
A) i/2
B) i/2 -1
C) i - 2
D) None of these



Question

How can I get the parent node of the following heap?

A) i/2
B) i/2 -1
C) i - 2
D) None of these



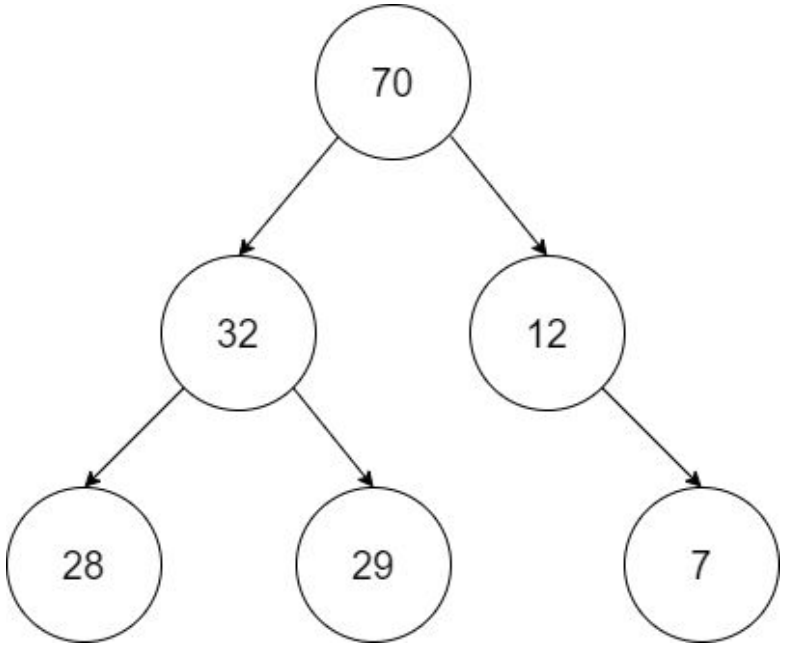
0	1	2	3	4	5	6	7	8
	50	30	15	19	20	10	5	2

for Node at i : Left child will be 2i and right child will be at 2i+1 and parent node will be at [i/2].

Question

Is this a valid Heap?

A) Yes
B) No

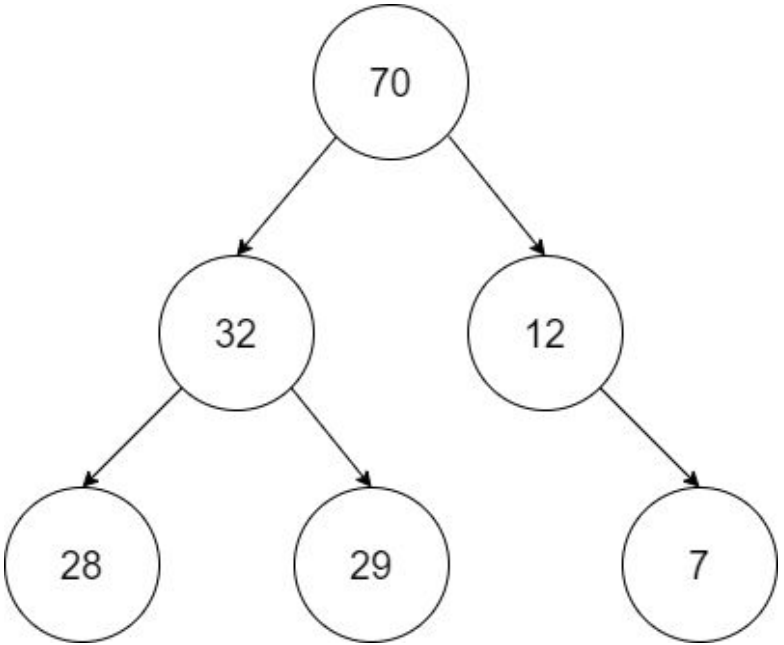


Question

Is this a valid Heap?

A) Yes

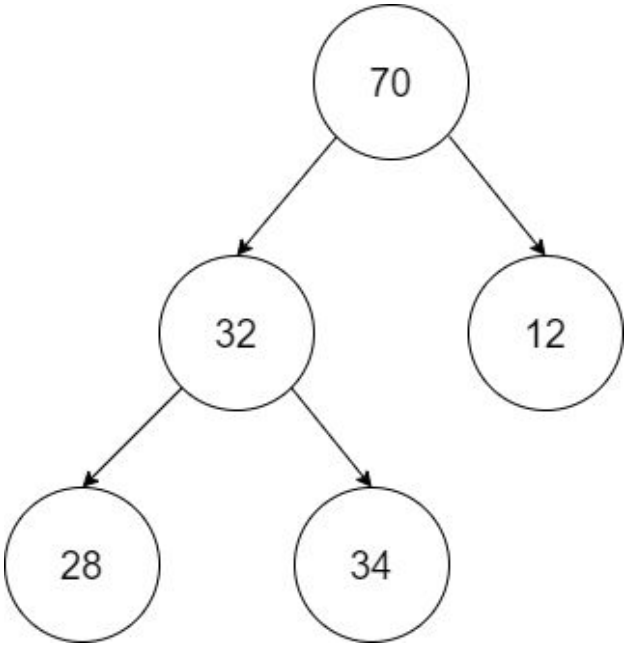
B) No



Question

Is this a valid Heap?

A) Yes
B) No

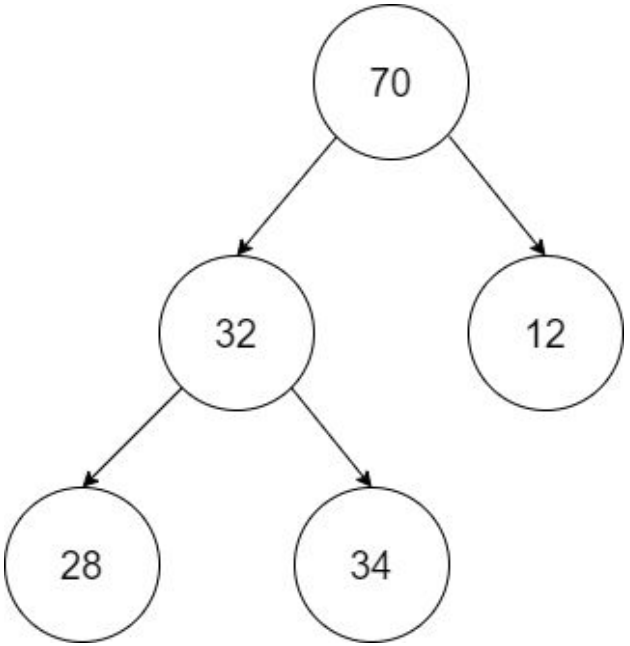


Question

Is this a valid Heap?

A) Yes

B) No



Bubble Down

- Used for deleting an element from the heap
- Take last element of heap and put it at the index of the element to be deleted
- Check and Swap
  a. Min-heap: if replaced element > any child node, swap element with the child that is smaller
  b. Max-heap: if replaced element < any child node, swap element with the child that is greater
- Keep repeating till conditions are not met

Bubble Up

- Used for inserting an element into the heap
- Insert element at the last leaf of the tree
- Check and Swap
  a. Min-heap: if inserted element < parent node, swap element with parent node
  b. Max-heap: if inserted element > parent node, swap element with parent node
- Keep repeating till the inserted element is in place
- http://btv.melezinek.cz/binary-heap.html

Tree Traversal

Recursion

Definition: A function that calls itself.

Recursive functions break bigger problems into smaller problems until we reach the base case, which is simple to solve.

Example: Write a program to calculate n!.

Recall n! = n * (n-1)!

(n-1)! = (n-1) * (n-2)!

…

2! = 2 * 1! = 2

Recursion

Example: Write a program to calculate n!.

Recall:

n! = n * (n-1)! ← bigger problem

(n-1)! = (n-1) * (n-2)!

…

2! = 2 * 1! = 2 ← smaller problem

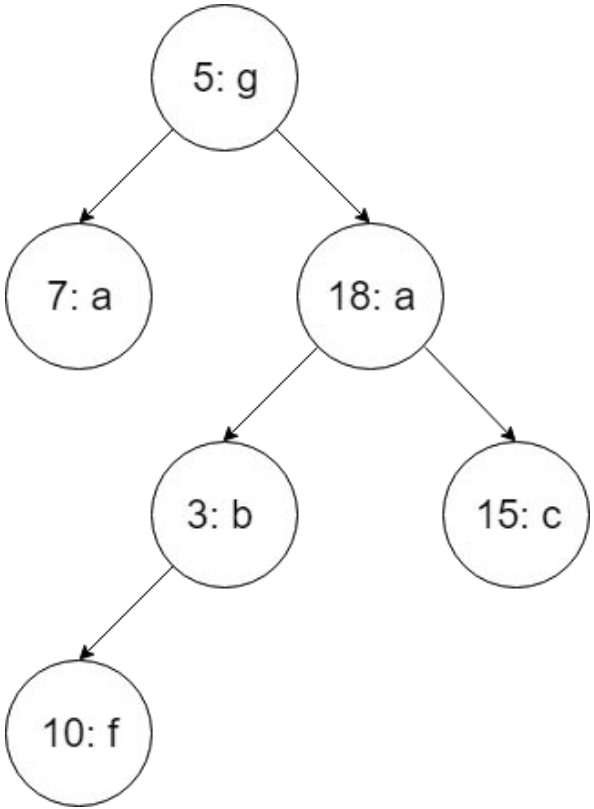
1! = 1 ← base case

Implementing inorder tree traversal

Step 1. Visit left subtree
Step 2: Visit Node
Step 3. Visit right subtree

Single Nodes are subtrees as well

An approach to traverse through entire tree

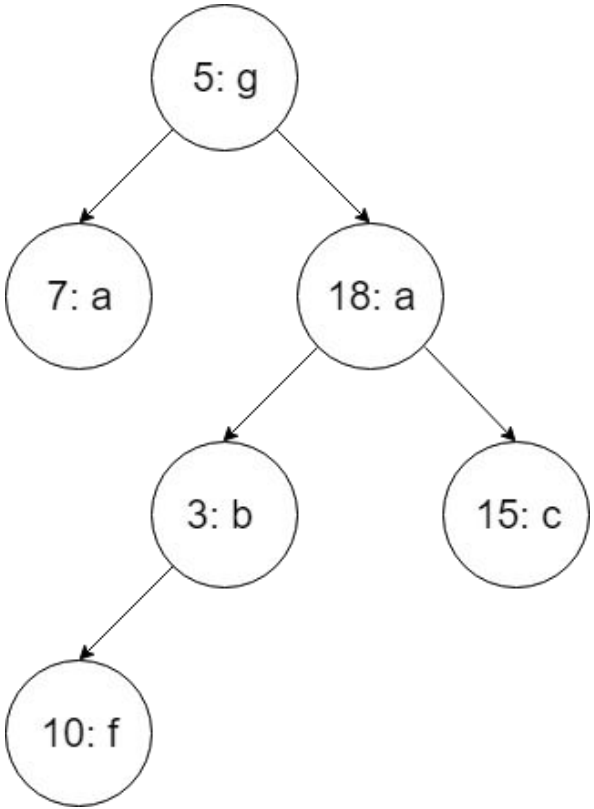


Implementing inorder tree traversal

Step 1. Starting at root node, visit leftmost subtree (single Nodes can be subtrees too). Which Node should we visit first then?

A. 5: g
B. 7: a
C. 10: f
D. 3: b
E. null

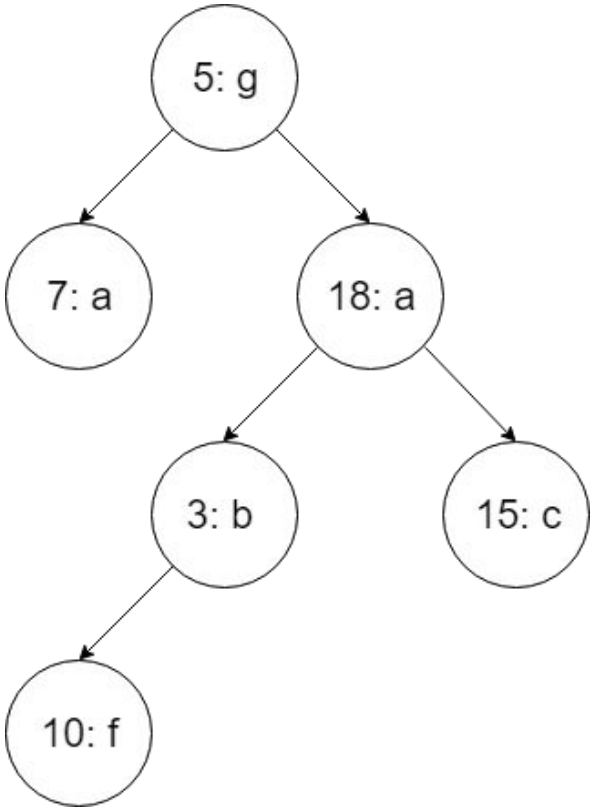
Visited Nodes: { }



Implementing inorder tree traversal

Stack trace to the rescue:

- Step 1. Visit left subtree 7: a
  - Step 1. Visit left subtree (null Node)

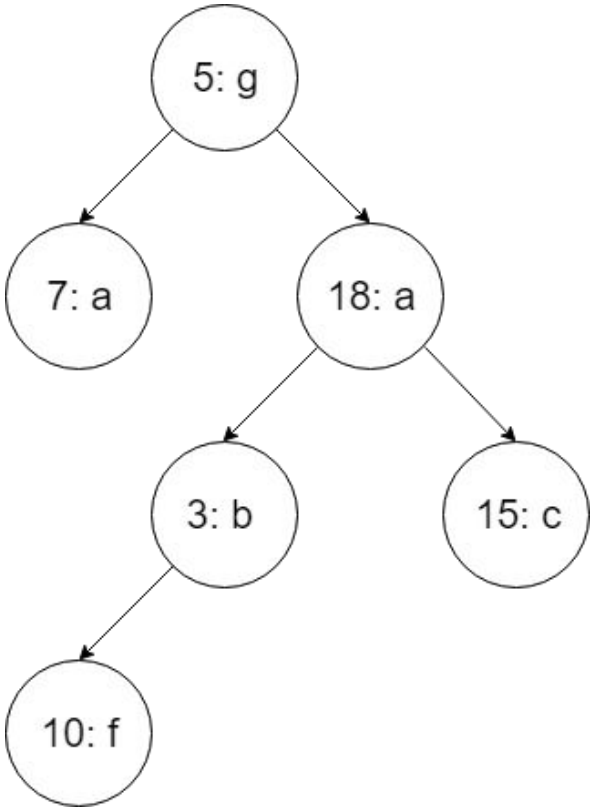


Inorder traversal visual example

Step 2. Visit root of subtree. Which one should we visit next then?

A. 5: g
B. 7: a
C. 10: f
D. 3: b
E. null

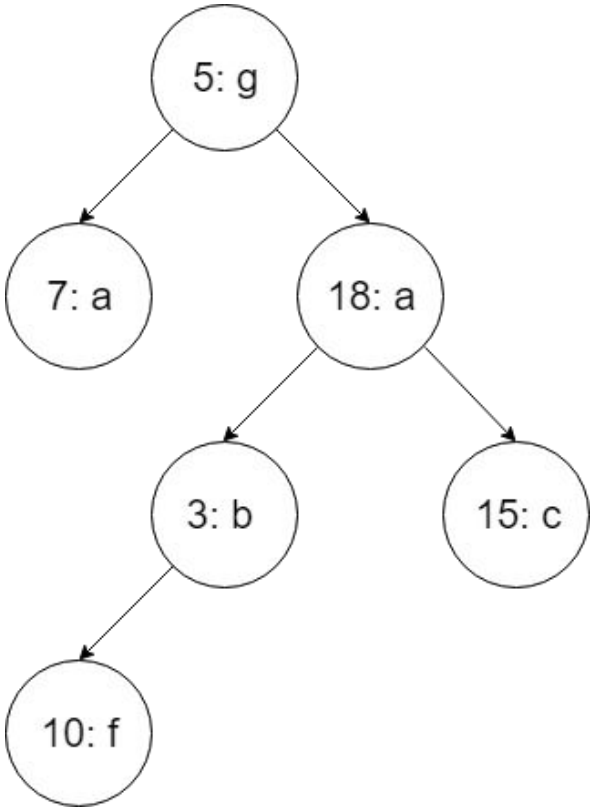
Visited Nodes: { }



Implementing inorder tree traversal

Stack trace to the rescue:

- Step 1. Visit left subtree 7: a
  - Step 1. Visit left subtree (null Node)
  - Step 2. Visit Node 7: a

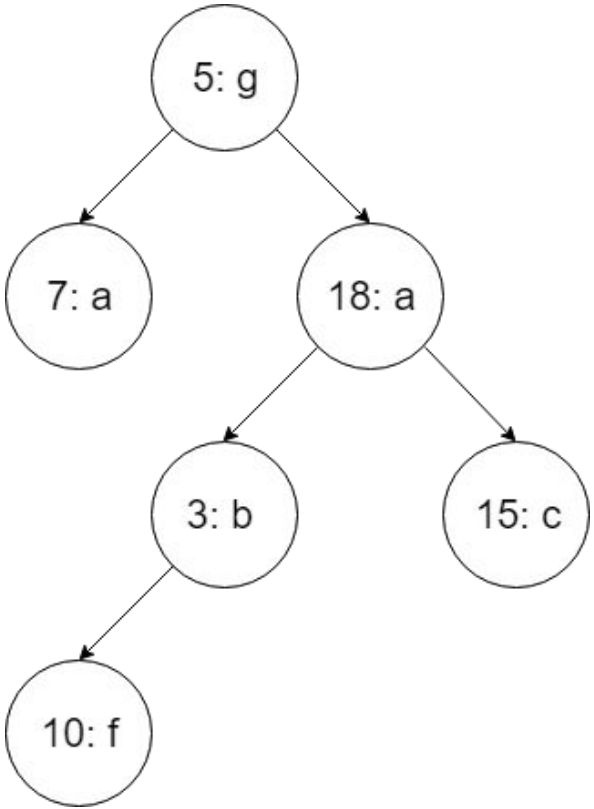


Inorder traversal visual example

Step 3. Visit rightmost subtree (single Nodes can be subtrees too). Which one should we visit then?

A. 15: c
B. 18: a
C. 3: b
D. 10: f
E. null

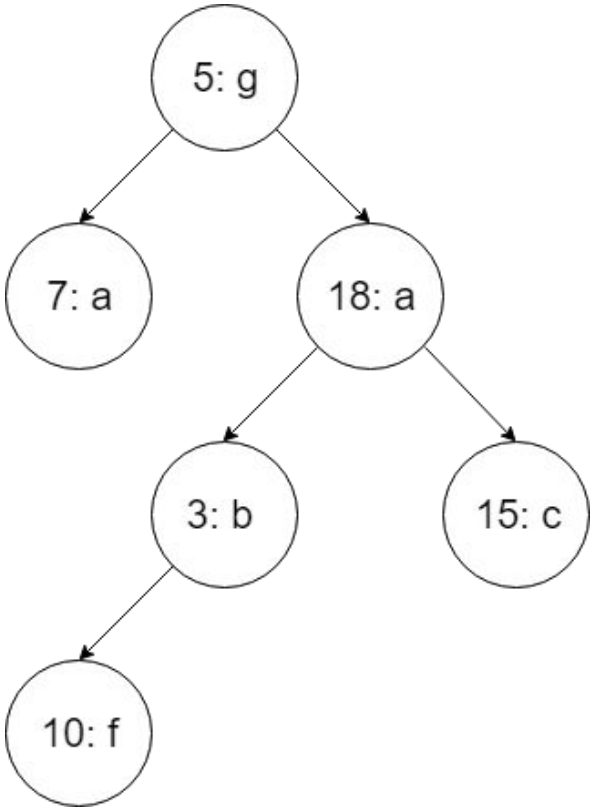
Visited Nodes: {7: a}



Implementing inorder tree traversal

Stack trace to the rescue:

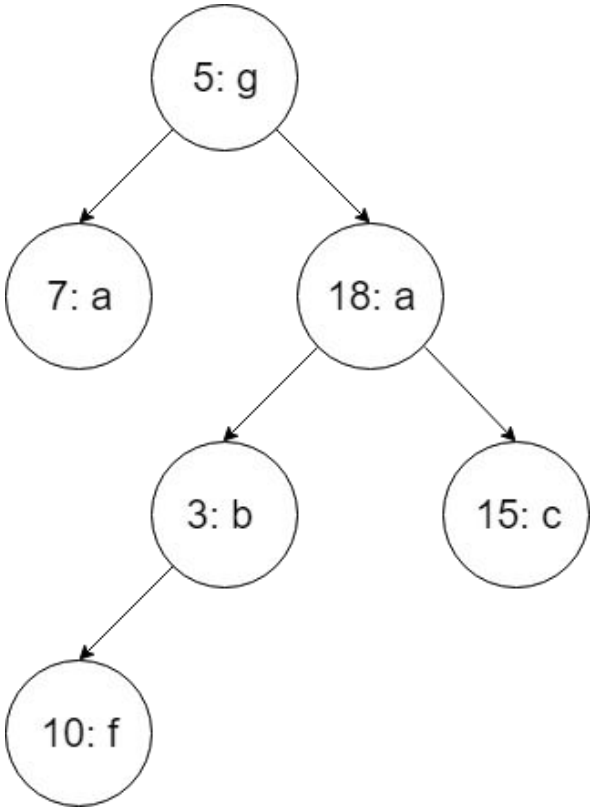
- Step 1. Visit left subtree 7: a
  - Step 1. Visit left subtree (null Node)
  - Step 2. Visit Node 7: a
  - Step 3. Visit right subtree (null Node)



Inorder traversal visual example

After traversing through all of the left subtree of 5: g, what do we visit now?

A. 5: g
B. 18: a
C. 3: b
D. 10: f
E. null

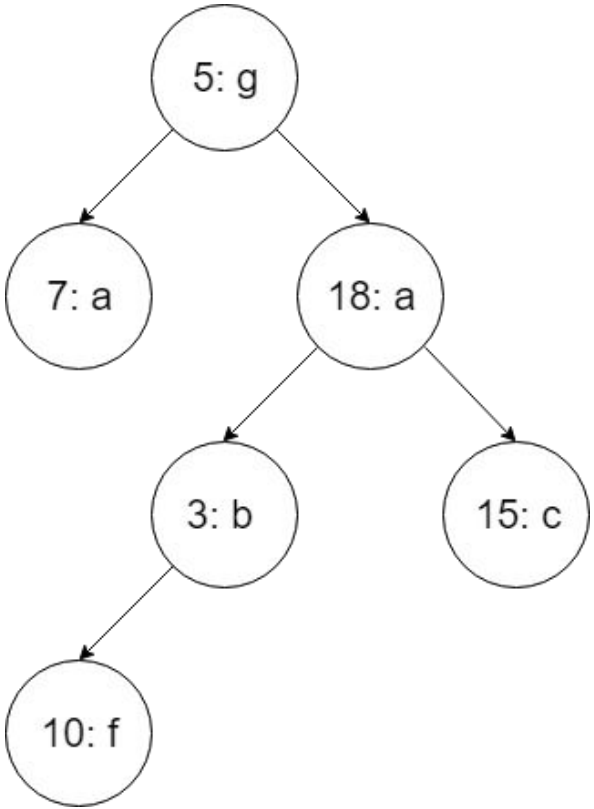


Visited Nodes: {7: a}

Implementing inorder tree traversal

Stack trace to the rescue:

- Step 1. Visit left subtree 7: a
  - Step 1. Visit left subtree (null Node)
  - Step 2. Visit Node 7: a
  - Step 3. Visit right subtree (null Node)
- Step 2. Visit Node 5: g

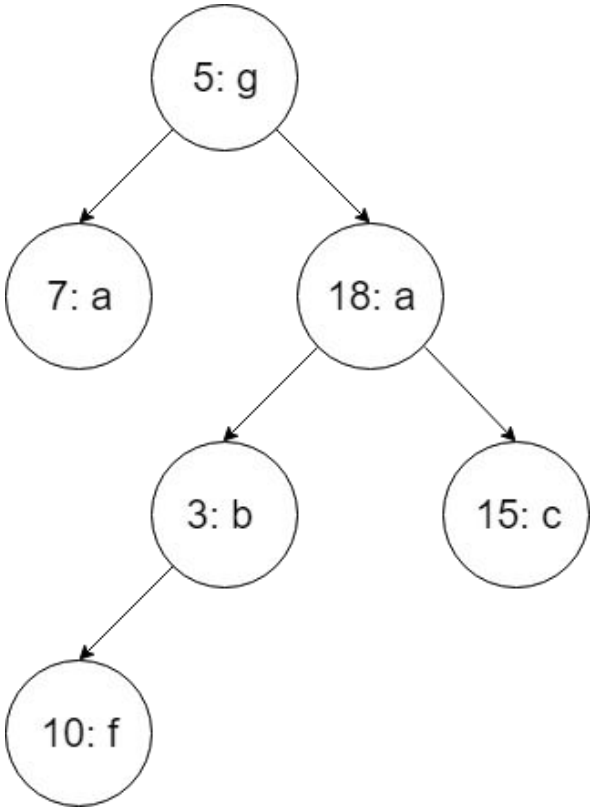


Inorder traversal visual example

After visiting root (5: g), what do we visit next?

A. 15: c
B. 18: a
C. 3: b
D. 10: f
E. null

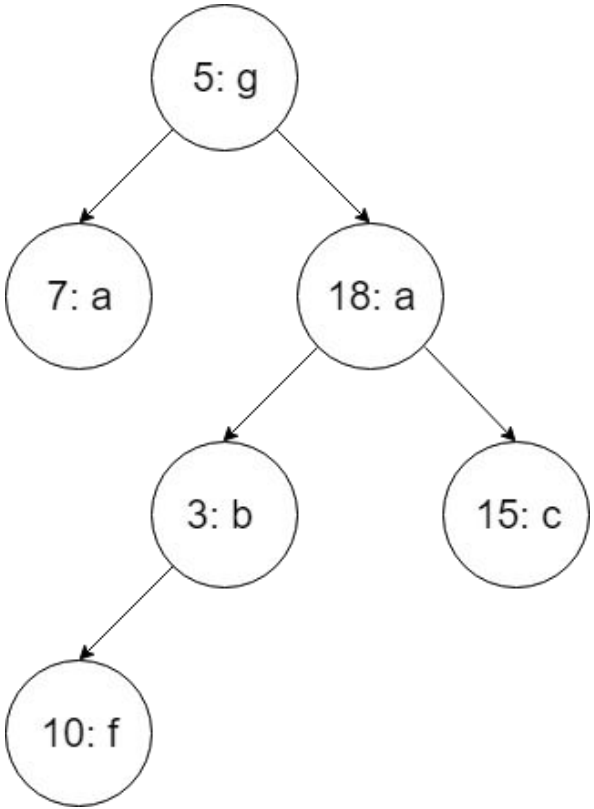
Visited Nodes: {7: a, 5: g}



Implementing inorder tree traversal

Stack trace to the rescue:

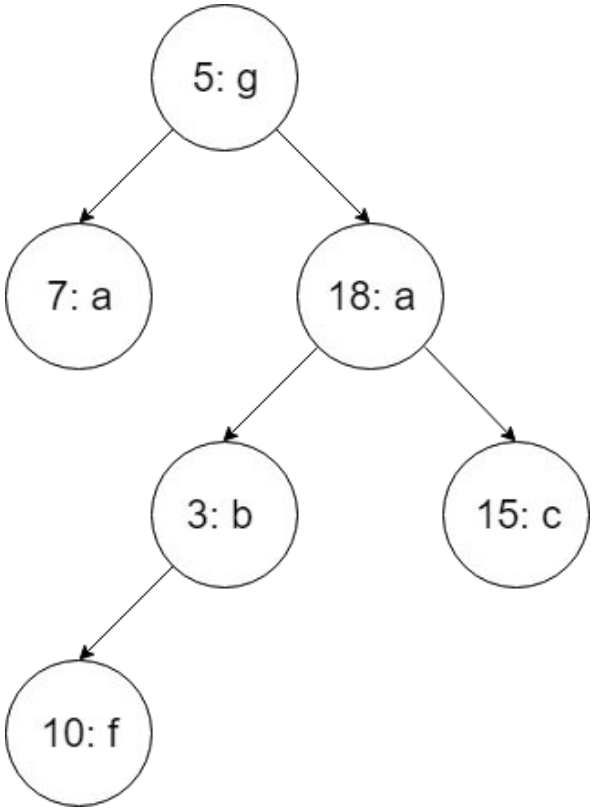
- Step 1. Visit left subtree 7: a
  - Step 1. Visit left subtree (null Node)
  - Step 2. Visit Node 7: a
  - Step 3. Visit right subtree (null Node)
- Step 2. Visit Node 5: g
- Step 3. Visit right subtree 18: a
  - Step 1. Visit left subtree 3: b
    - Step 1. Visit left subtree 10: f
      - Step 1. Visit left subtree (null Node)



Inorder traversal visual example

Following this procedure, what do we visit next?

A. 15: c
B. 18: a
C. 3: b
D. 10: f
E. null

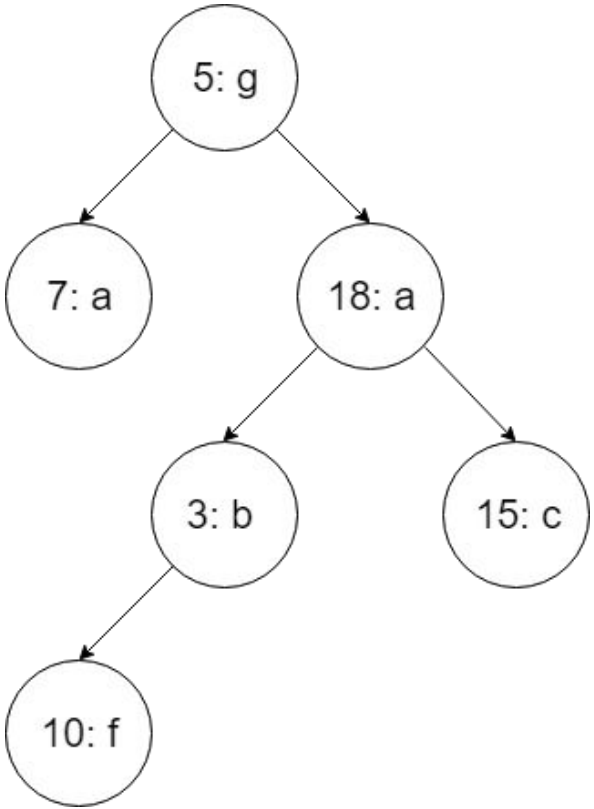


Visited Nodes: {7: a, 5: g}

Implementing inorder tree traversal

Stack trace to the rescue:

- Step 1. Visit left subtree 7: a
  - Step 1. Visit left subtree (null Node)
  - Step 2. Visit Node 7: a
  - Step 3. Visit right subtree (null Node)
- Step 2. Visit Node 5: g
- Step 3. Visit right subtree 18: a
  - Step 1. Visit left subtree 3: b
    - Step 1. Visit left subtree 10: f
      - Step 1. Visit left subtree (null Node)
      - Step 2. Visit Node 10: f

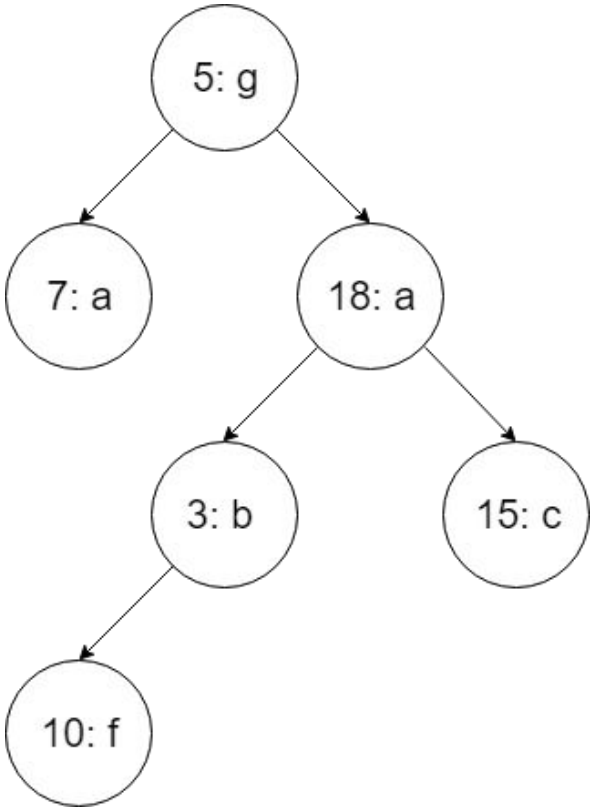


Inorder traversal visual example

Following this procedure, in what order do we visit remaining Nodes?

A. 3: b, 15: c, 18: a
B. 15: c, 3: b, 18: a
C. 3: b, 18: a, 15: c
D. 18: a, 3: b, 15: c
E. 15: c, 18: a, 3: b

Visited Nodes: {7: a, 5: g, 10: f}

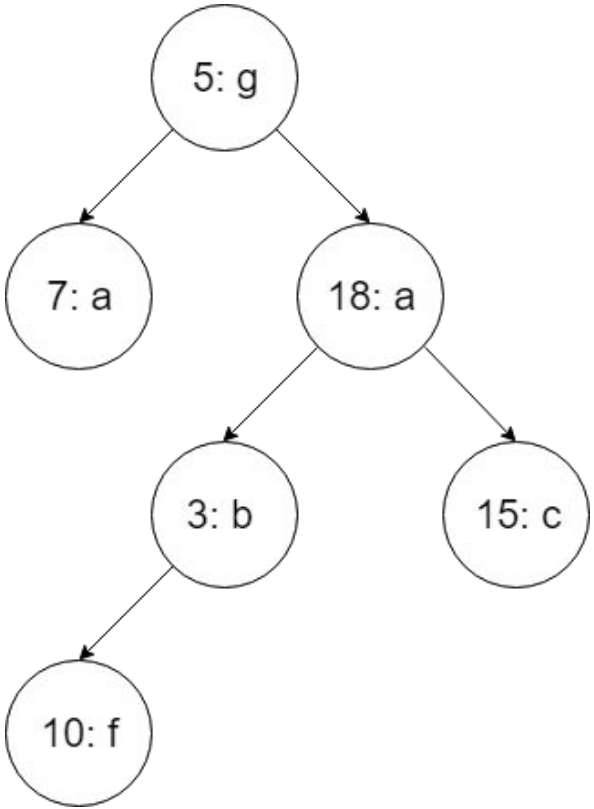


Answer - C

Repeating Steps 1-3 at each subtree at the lowest level possible, we then visit:

3: b, null, 18: a, null, 15: c, null

Which is just 3: b, 18: a, 15: c

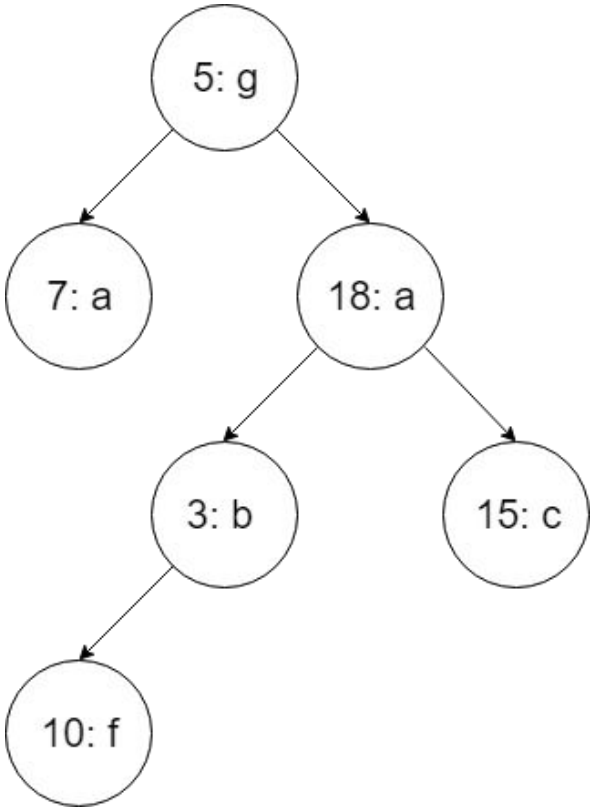


Visited Nodes: {7: a, 5: g, 10: f, 3: b, 18: a, 15: c}

Implementing inorder tree traversal

Stack trace to the rescue:

- Step 1. Visit left subtree 7: a
  - Step 1. Visit left subtree (null Node)
  - Step 2. Visit Node 7: a
  - Step 3. Visit right subtree (null Node)
- Step 2. Visit Node 5: g
- Step 3. Visit right subtree 18: a
  - Step 1. Visit left subtree 3: b
    - Step 1. Visit left subtree 10: f
      - Step 1. Visit left subtree (null Node)
      - Step 2. Visit Node 10: f
      - Step 3. Visit right subtree (null Node)
    - Step 2. Visit Node 3: b
    - Step 3. Visit right subtree (null Node)
  - Step 2. Visit Node 18: a
  - Step 3. Visit right subtree 15: c
    - Step 1. Visit left subtree (null Node)
    - Step 2. Visit Node 15: c
    - Step 3. Visit right subtree (null Node)

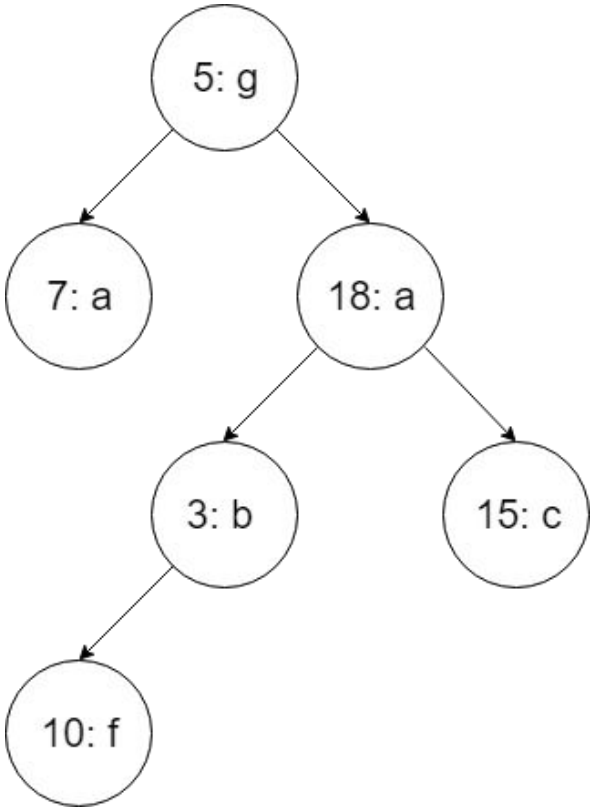


Implementing inorder tree traversal

```
public inorder(Node<K, V> node) {
if (node == null) return;
inorder(node.left);
System.out.println(node.key + “: “ + Node.value);
inorder(node.right);

}
```

We will check if node is null before trying to access left and right subtree because we will receive a null pointer exception otherwise.

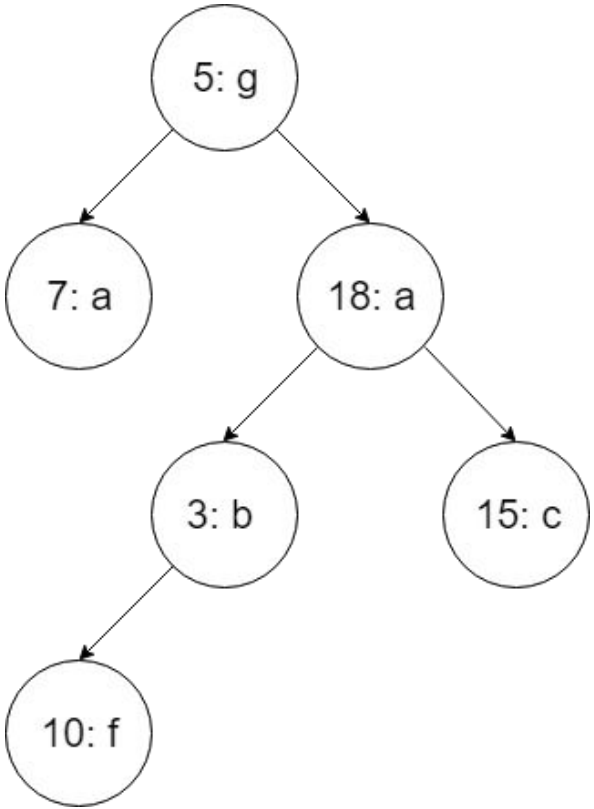


Implementing preorder tree traversal

Step 1. Visit Node
Step 2: Visit left subtree
Step 3. Visit right subtree

Single Nodes are subtrees as well

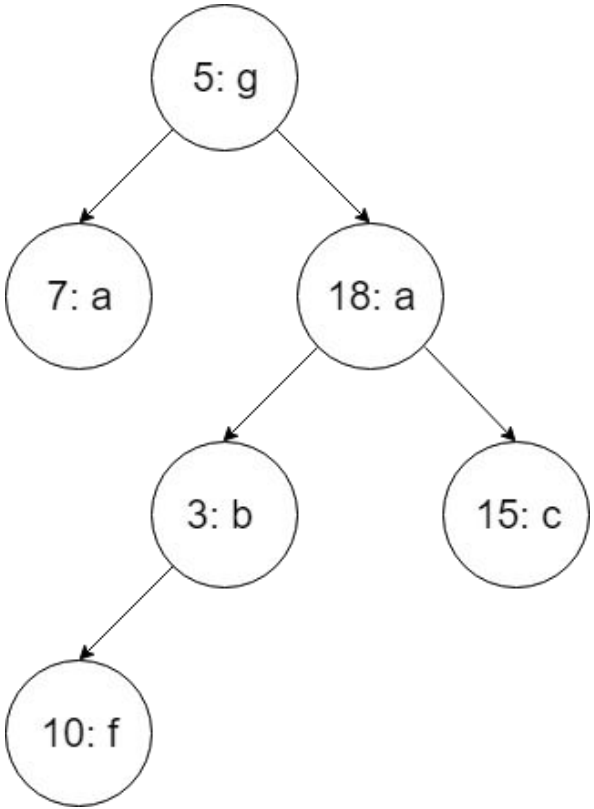
Another approach to traverse through entire tree



Implementing preorder tree traversal

Stack trace to the rescue:

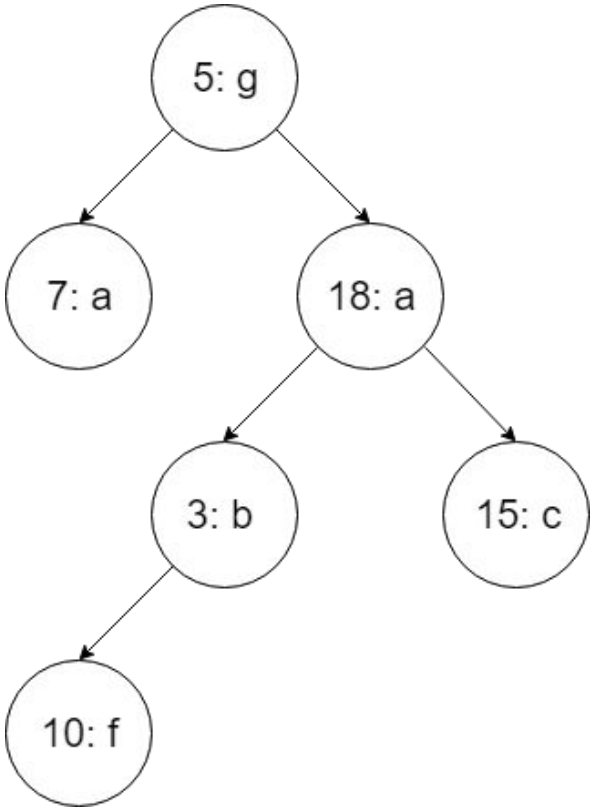
- Step 1. Visit root 5: g



Implementing preorder tree traversal

Stack trace to the rescue:

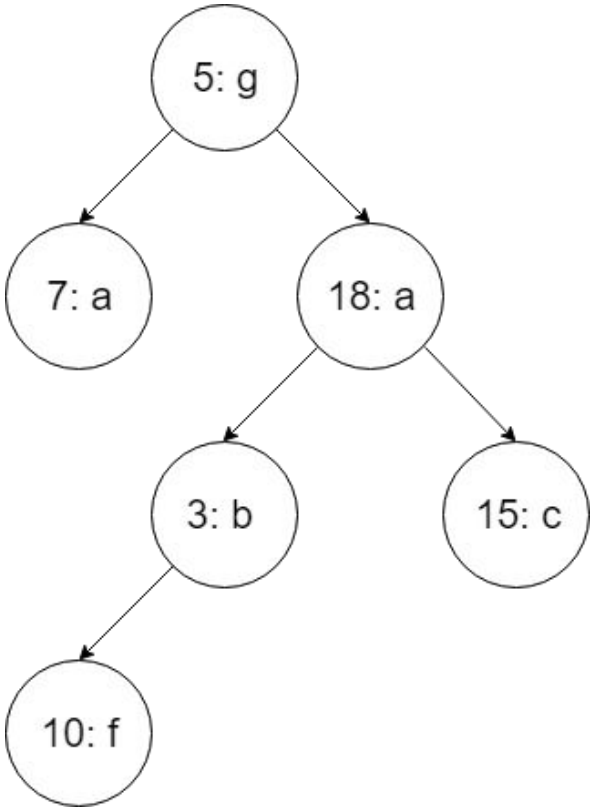
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a



Implementing preorder tree traversal

Stack trace to the rescue:

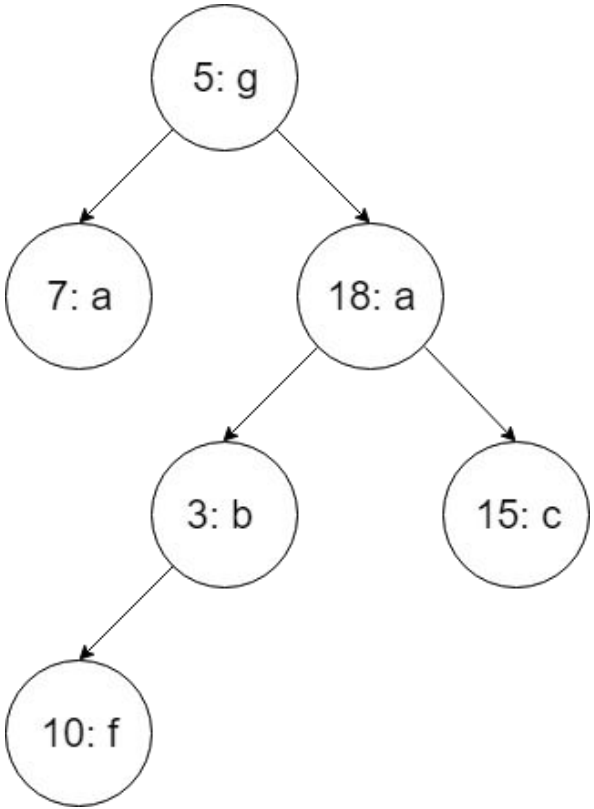
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a



Implementing preorder tree traversal

Stack trace to the rescue:

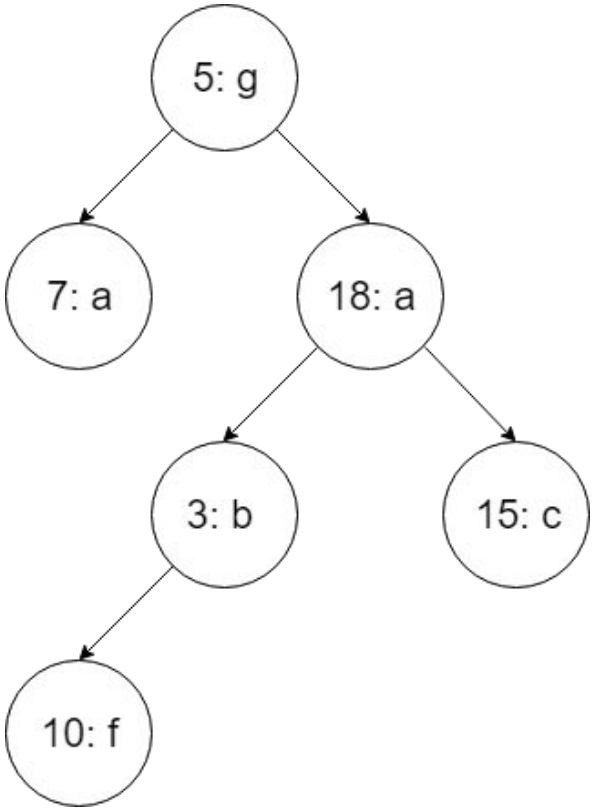
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)



Implementing preorder tree traversal

Stack trace to the rescue:

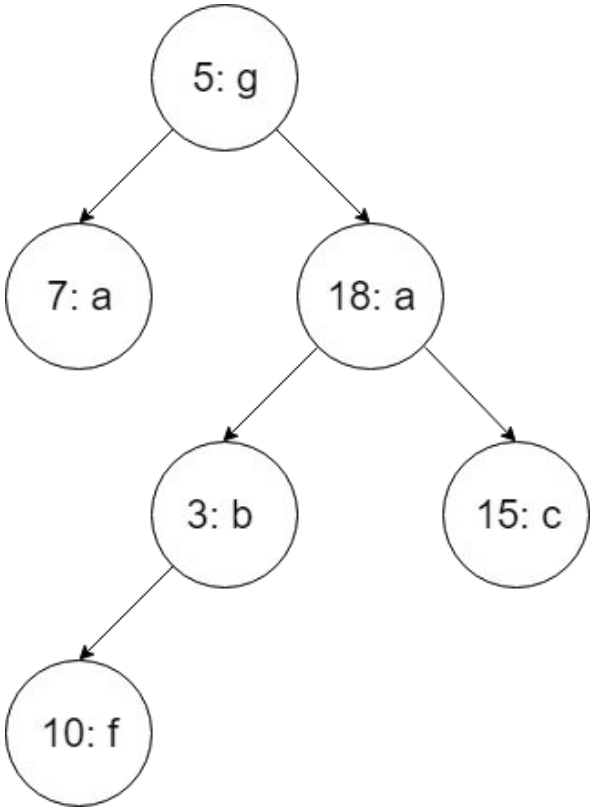
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a



Implementing preorder tree traversal

Stack trace to the rescue:

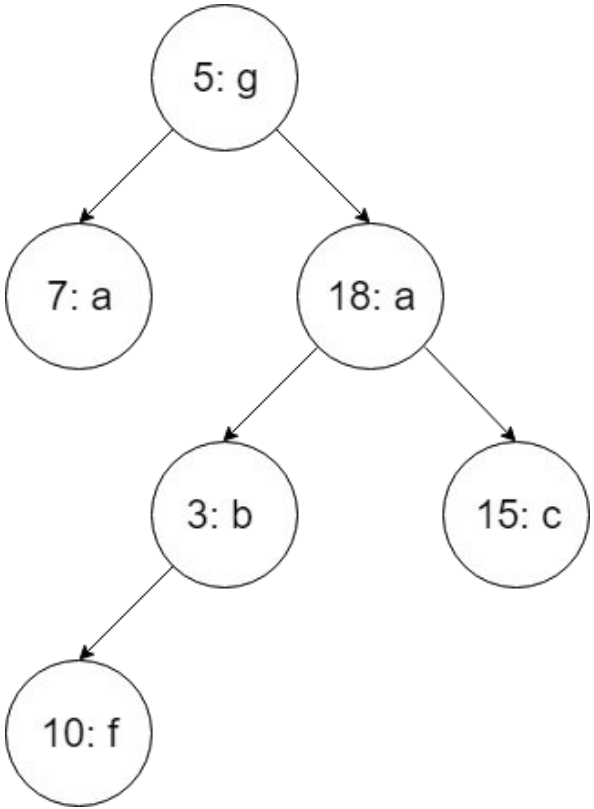
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a



Implementing preorder tree traversal

Stack trace to the rescue:

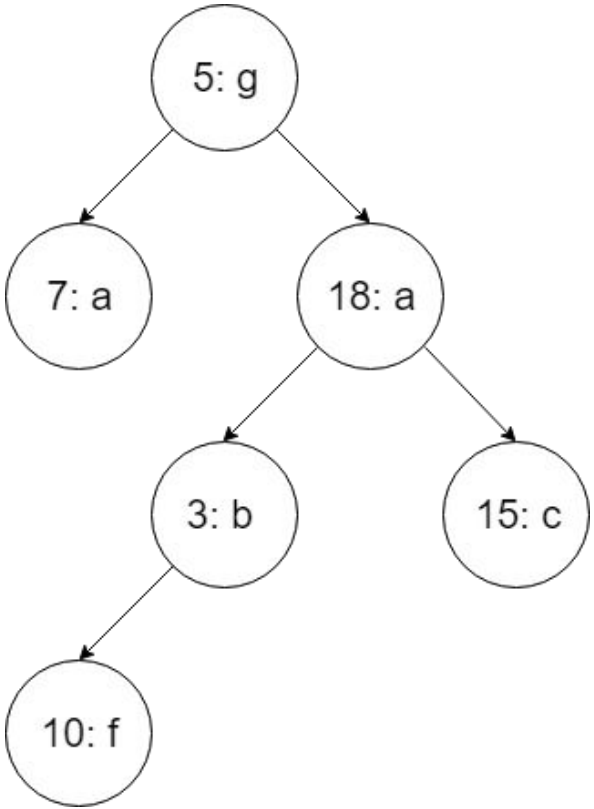
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b



Implementing preorder tree traversal

Stack trace to the rescue:

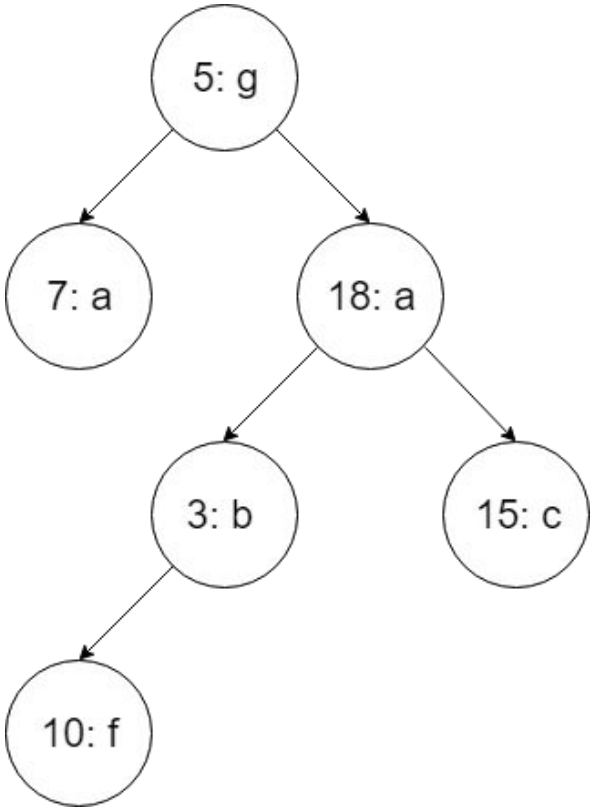
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b



Implementing preorder tree traversal

Stack trace to the rescue:

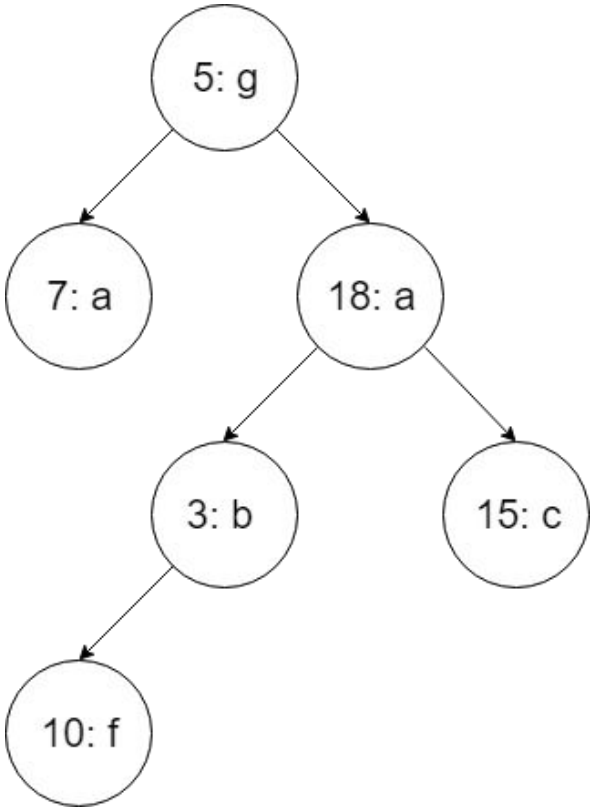
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f



Implementing preorder tree traversal

Stack trace to the rescue:

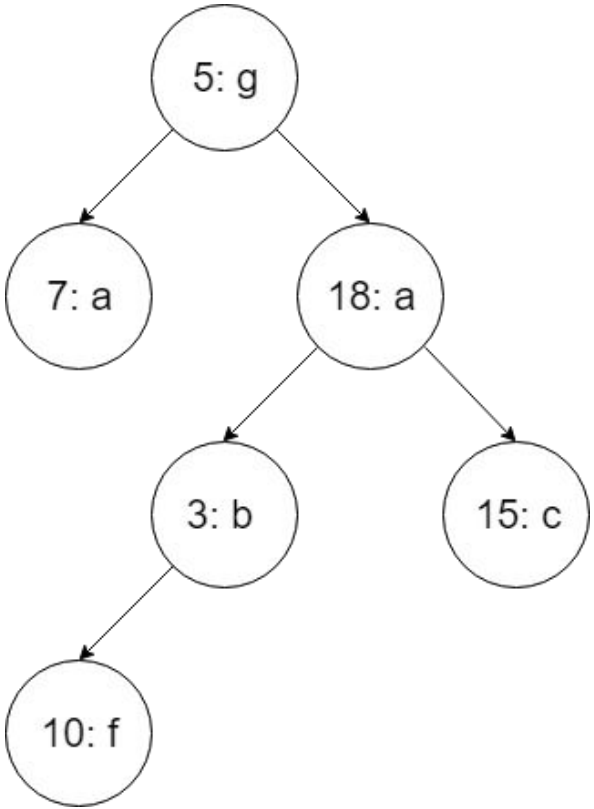
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f
      - Step 1. Visit root 10: f



Implementing preorder tree traversal

Stack trace to the rescue:

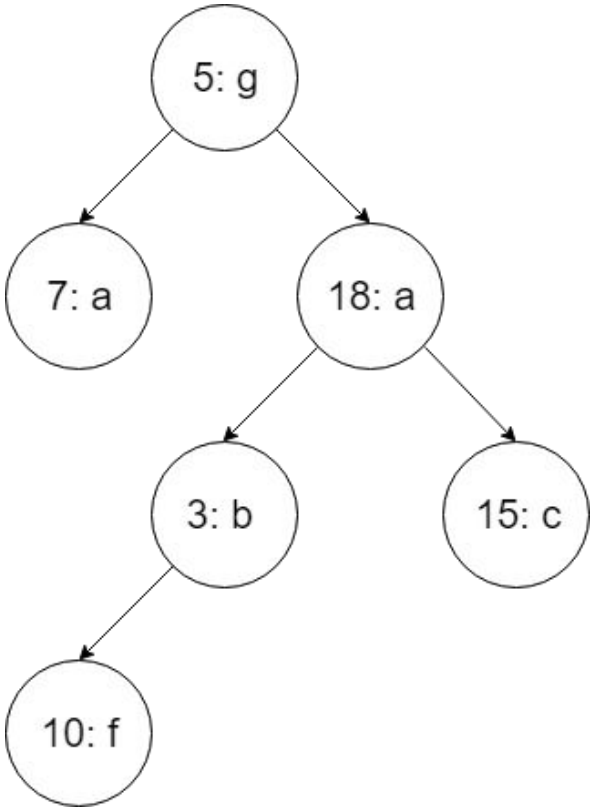
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f
      - Step 1. Visit root 10: f
      - Step 2. Visit left subtree (null Node)



Implementing preorder tree traversal

Stack trace to the rescue:

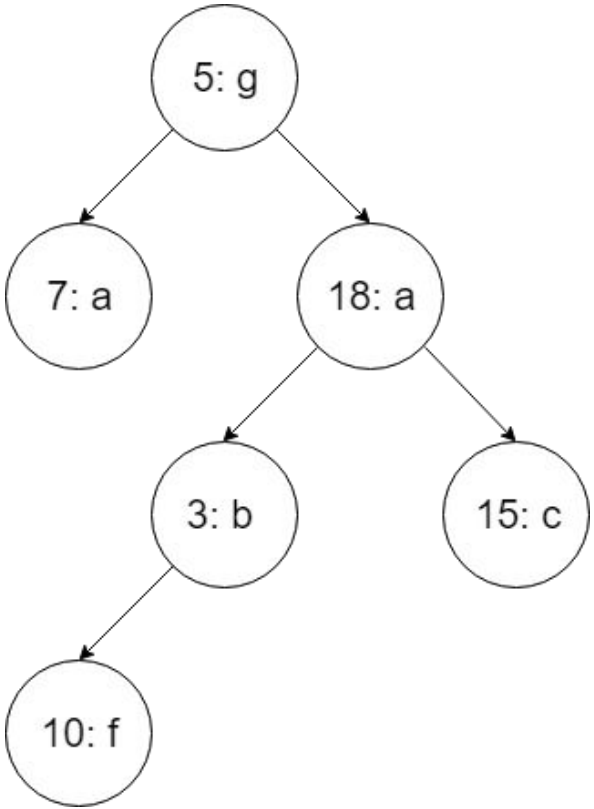
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f
      - Step 1. Visit root 10: f
      - Step 2. Visit left subtree (null Node)
      - Step 3. Visit left subtree (null Node)



Implementing preorder tree traversal

Stack trace to the rescue:

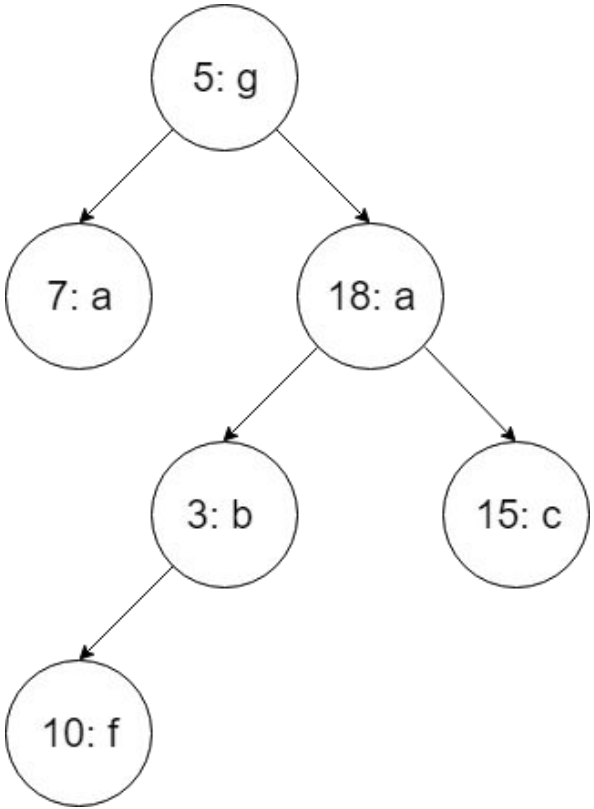
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f
      - Step 1. Visit root 10: f
      - Step 2. Visit left subtree (null Node)
      - Step 3. Visit left subtree (null Node)
    - Step 3. Visit right subtree (null Node)



Implementing preorder tree traversal

Stack trace to the rescue:

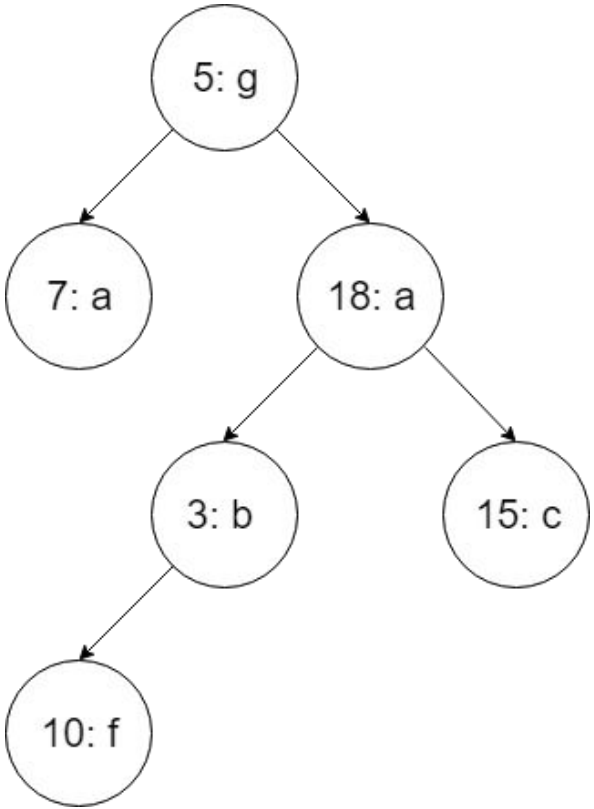
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f
      - Step 1. Visit root 10: f
      - Step 2. Visit left subtree (null Node)
      - Step 3. Visit left subtree (null Node)
    - Step 3. Visit right subtree (null Node)
  - Step 3. Visit right subtree 15: c



Implementing preorder tree traversal

Stack trace to the rescue:

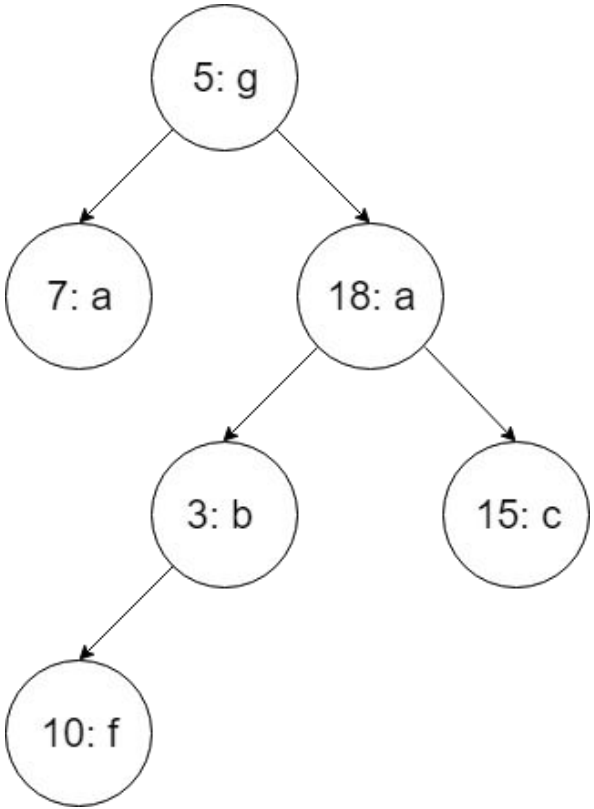
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f
      - Step 1. Visit root 10: f
      - Step 2. Visit left subtree (null Node)
      - Step 3. Visit left subtree (null Node)
    - Step 3. Visit right subtree (null Node)
  - Step 3. Visit right subtree 15: c
    - Step 1. Visit root 15: c



Implementing preorder tree traversal

Stack trace to the rescue:

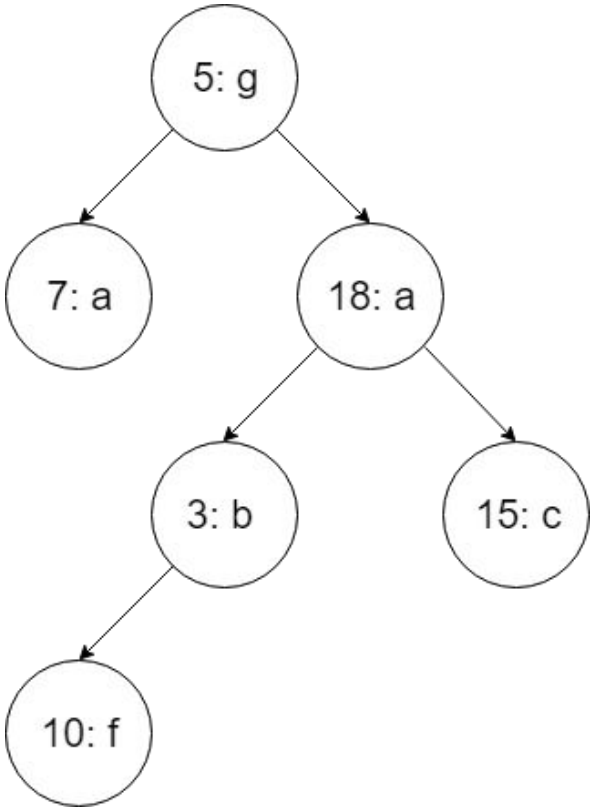
- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f
      - Step 1. Visit root 10: f
      - Step 2. Visit left subtree (null Node)
      - Step 3. Visit left subtree (null Node)
    - Step 3. Visit right subtree (null Node)
  - Step 3. Visit right subtree 15: c
    - Step 1. Visit root 15: c
    - Step 2. Visit left subtree (null Node)



Implementing preorder tree traversal

Stack trace to the rescue:

- Step 1. Visit root 5: g
- Step 2. Visit left subtree 7: a
  - Step 1. Visit root 7: a
  - Step 2. Visit left subtree (null Node)
  - Step 3. Visit right subtree (null Node)
- Step 3. Visit right subtree 18: a
  - Step 1. Visit root 18: a
  - Step 2. Visit left subtree 3: b
    - Step 1. Visit root 3: b
    - Step 2. Visit left subtree 10: f
      - Step 1. Visit root 10: f
      - Step 2. Visit left subtree (null Node)
      - Step 3. Visit left subtree (null Node)
    - Step 3. Visit right subtree (null Node)
  - Step 3. Visit right subtree 15: c
    - Step 1. Visit root 15: c
    - Step 2. Visit left subtree (null Node)
    - Step 3. Visit right subtree (null Node)

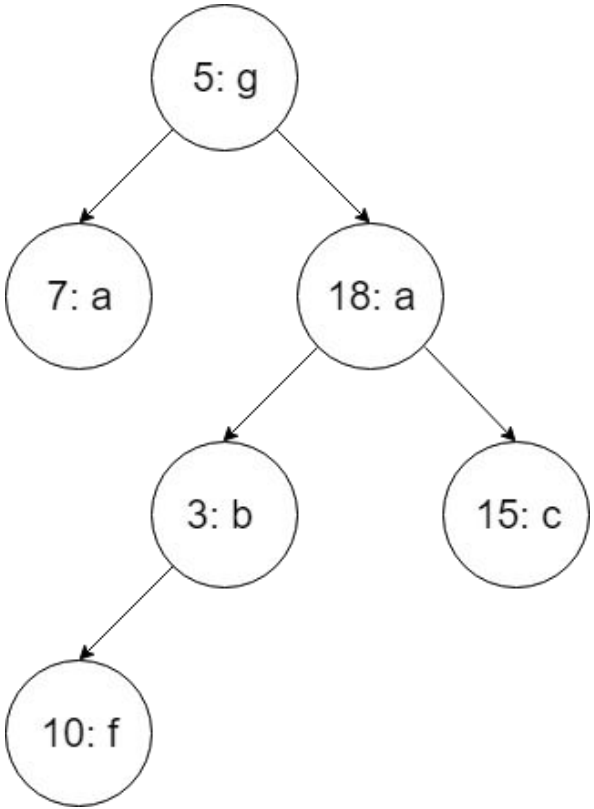


Implementing preorder tree traversal

```
public preorder(Node<K, V> node) {
if (node == null) return;
System.out.println(node.key + “: “ + Node.value);
preorder(node.left);
preorder(node.right);

}
```

We will check if node is null before trying to access left and right subtree because we will receive a null pointer exception otherwise.

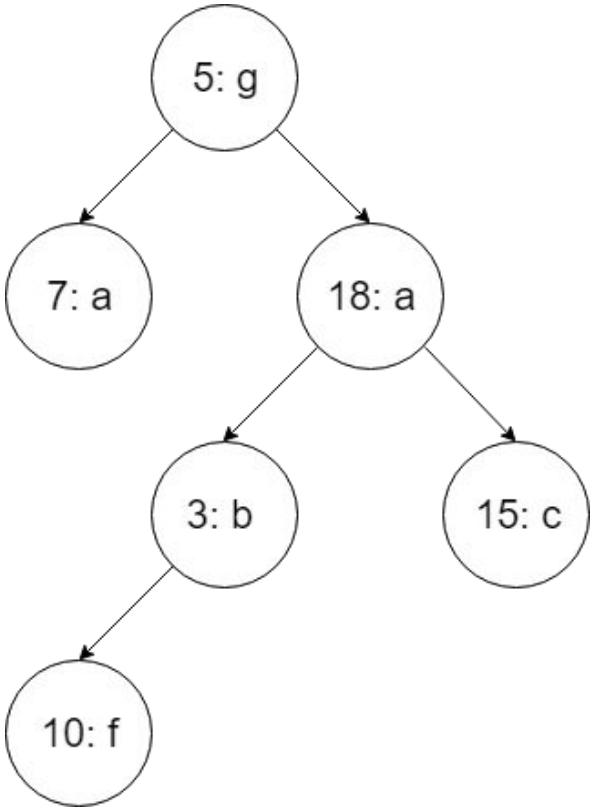


Implementing postorder tree traversal

Step 1: Visit left subtree
Step 2: Visit right subtree
Step 3: Visit Node

Single Nodes are subtrees as well

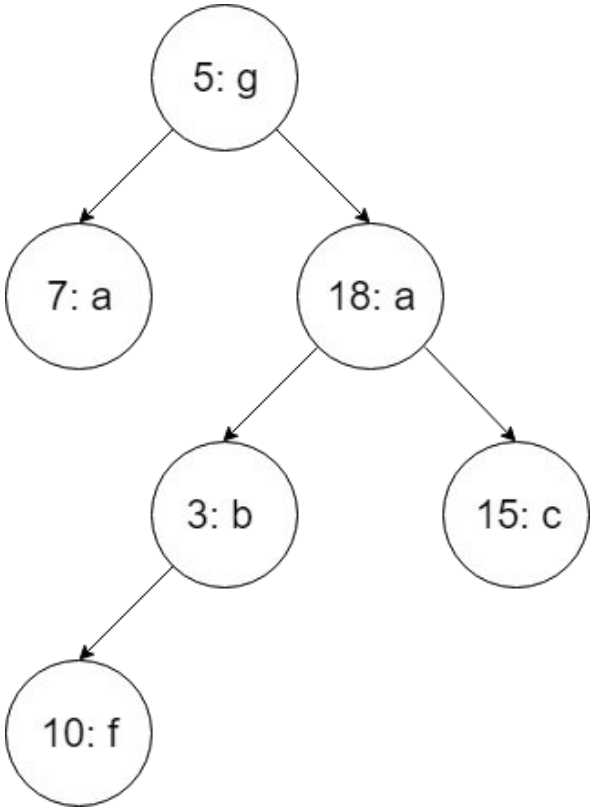
Another approach to traverse through entire tree



Implementing postorder tree traversal

Can we implement postorder by just switching the order of statements in the inorder and preorder traversal methods?

A. Yes
B. No
C. I don't know



Implementing postorder tree traversal

```
public postorder(Node<K, V> node) {
if (node == null) return;
postorder(node.left);
postorder(node.right);
System.out.println(node.key + “: “ + Node.value);

}
```

We will check if node is null before trying to access left and right subtree because we will receive a null pointer exception otherwise.