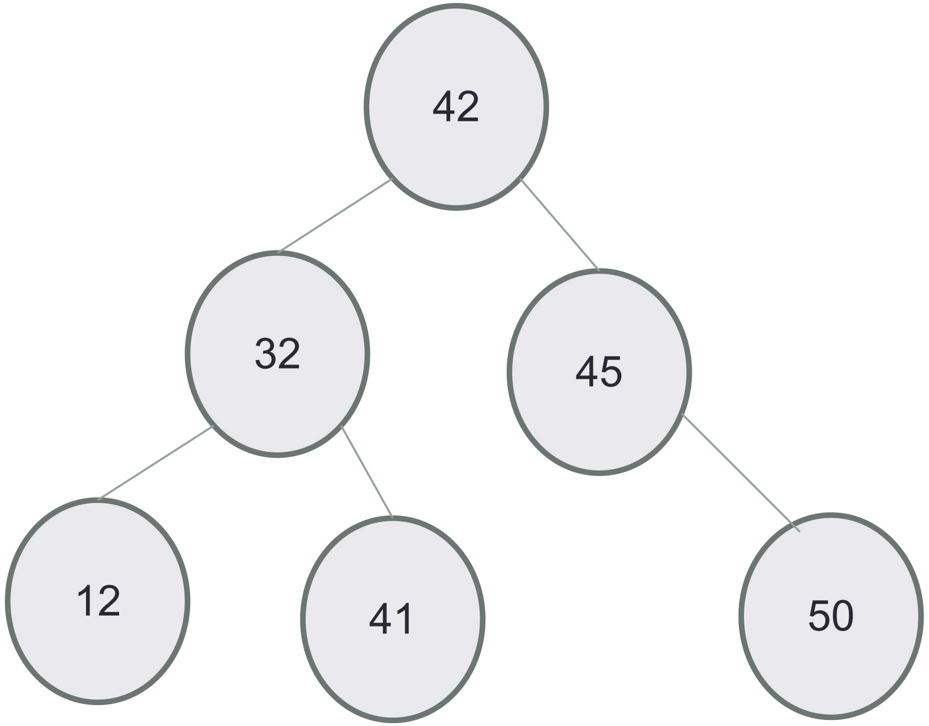
BINARY SEARCH TREES - PART 2

Problem Solving with Computers-II

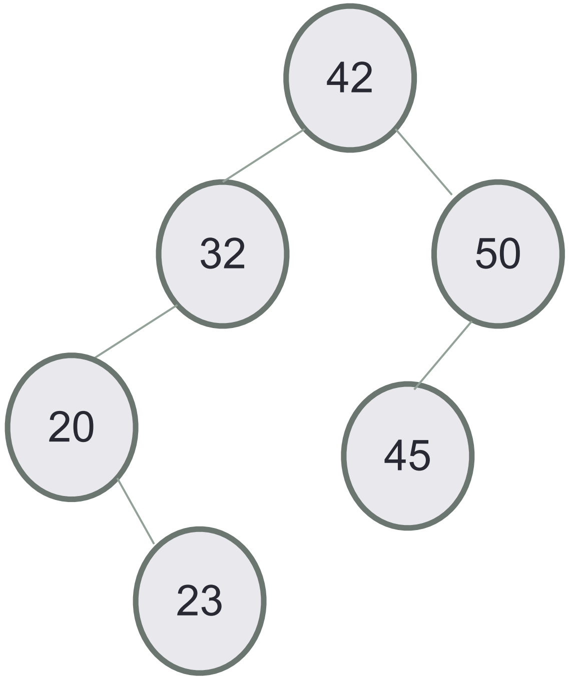
```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;

}
```

Define the BST ADT

Operations
Search
Insert
Min
Max
Successor
Predecessor
Delete
Print elements In order Preorder, Post order



Predecessor: Next smallest element



```
int bst::predecessor(BSTNode* n, int value) const{
   if(!n) return std::numeric_limits<int>::min();
   if(n->left){
        //Case 1
        return ________________________;

    }else{
        //Case 2


    }
}
```

- What is the predecessor of 32?
- What is the predecessor of 45?

Fill in the blank for case 1 using min/max helper functions

A. `n->left;`

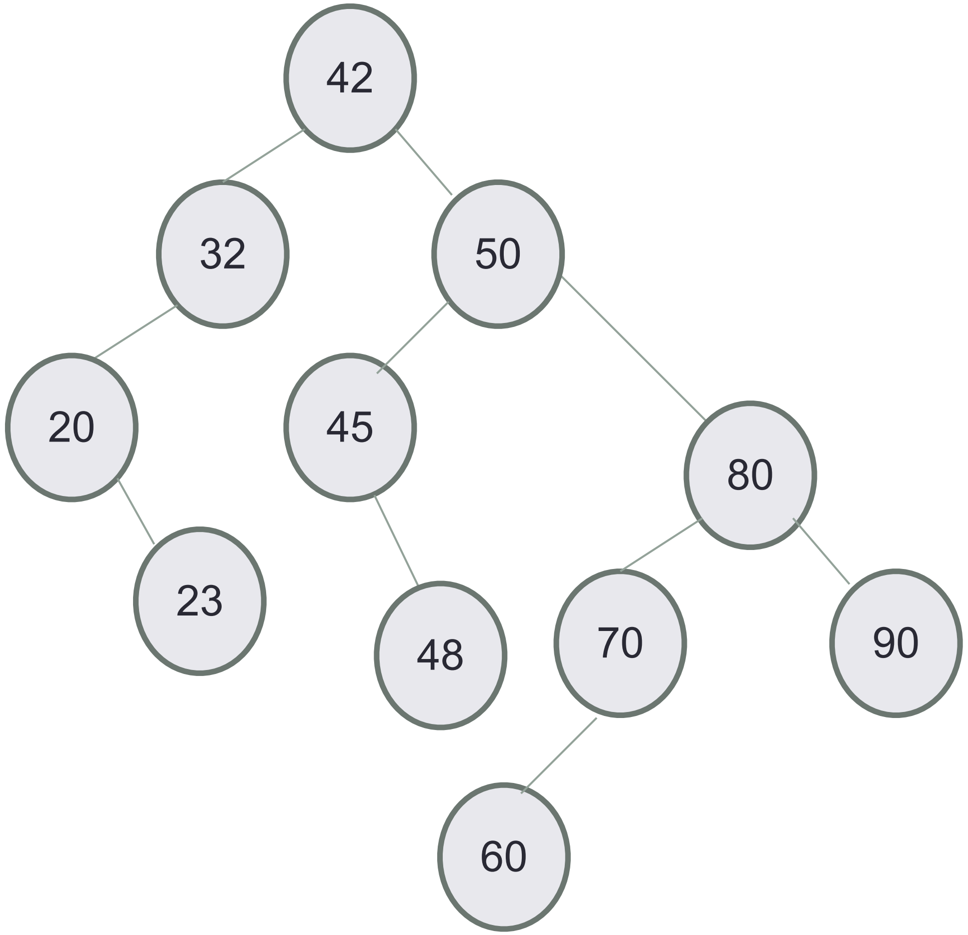
B. `min(n)`

C. `max(n)`

D. `min(n->left)`

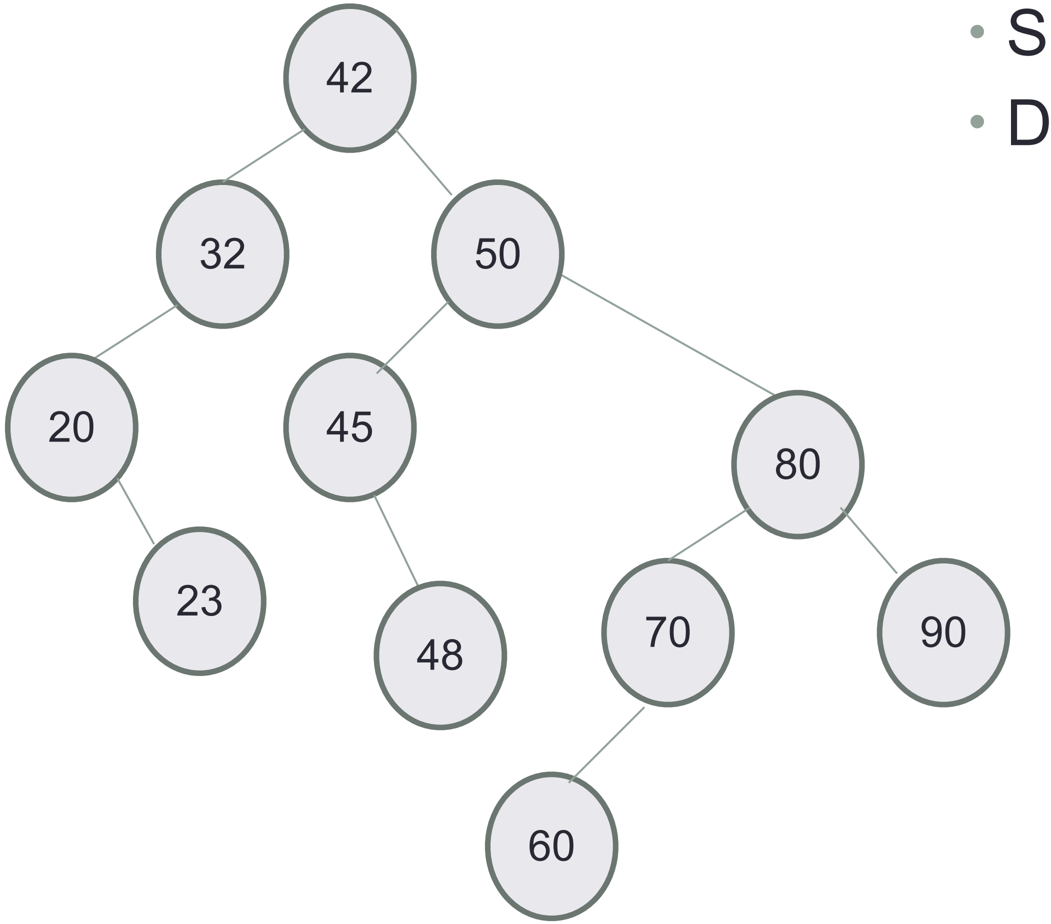
E. `max(n->left)`

Successor: Next largest element



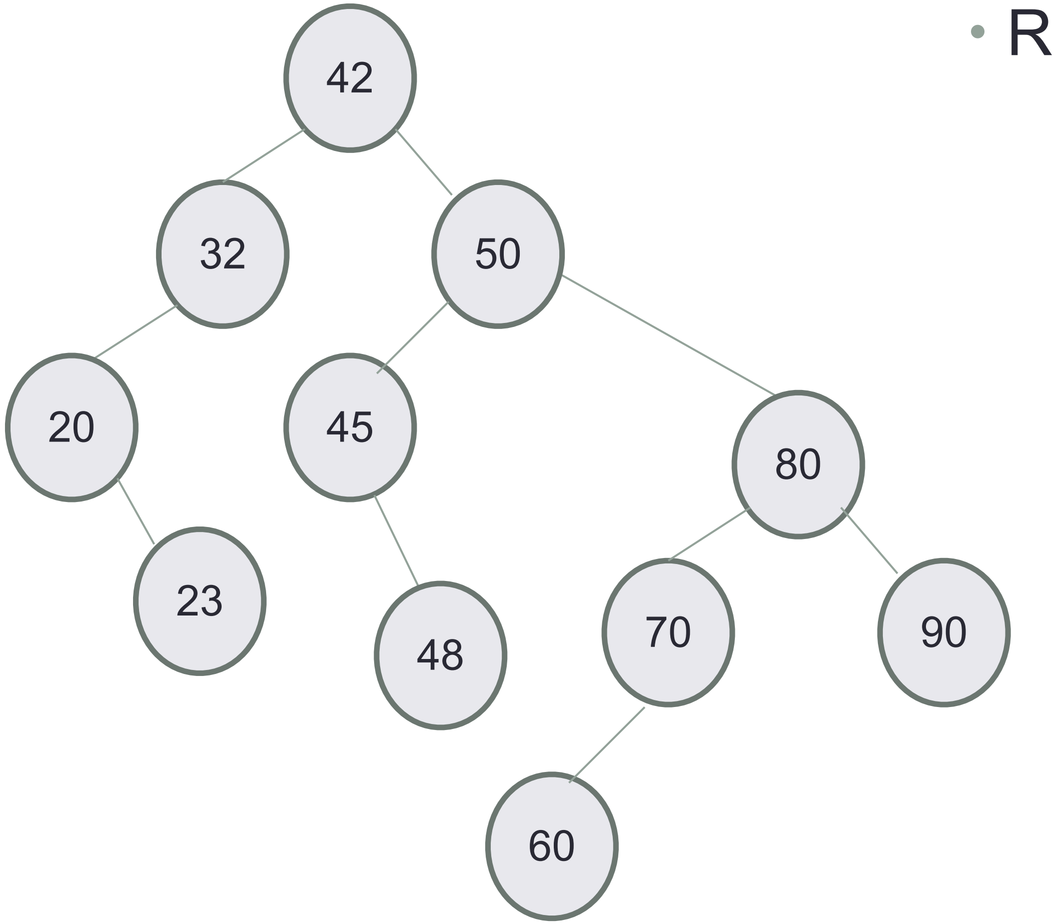
- What is the successor of 45?
- What is the successor of 50?
- What is the successor of 60?

Delete: Case 1 - Node is a leaf node



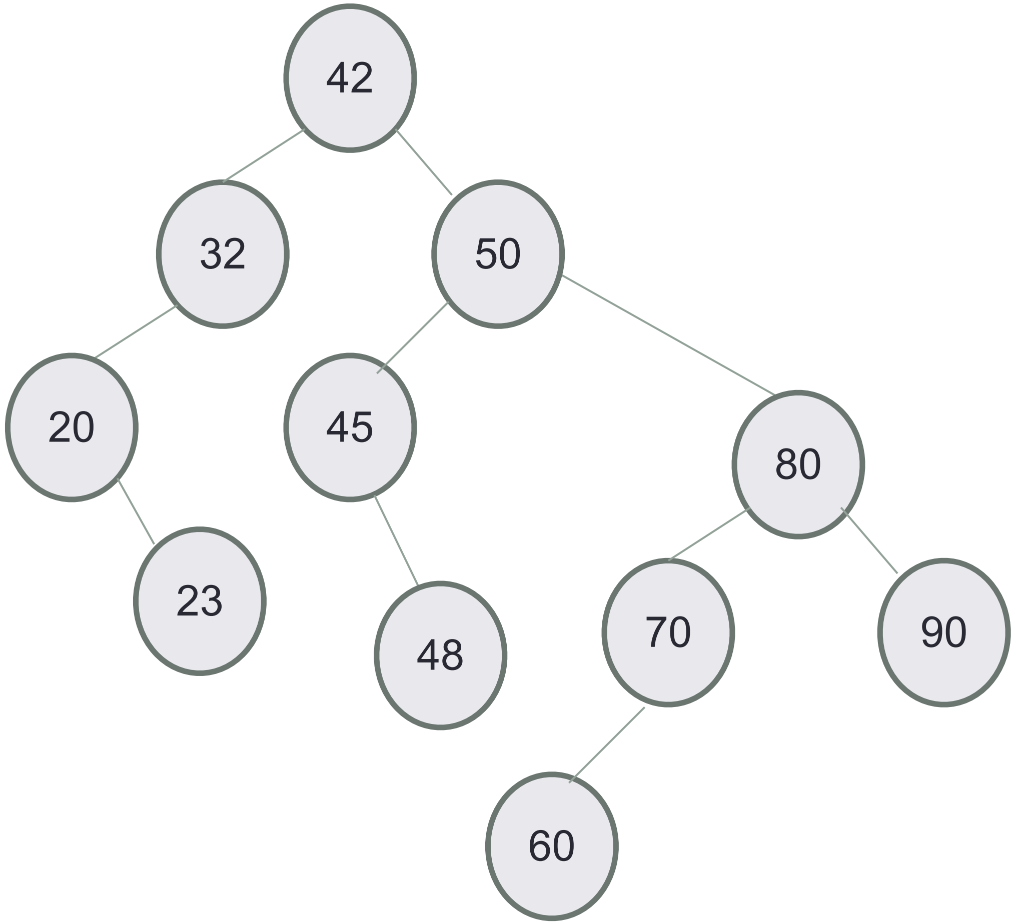
- Set parent’s (left/right) child pointer to null
- Delete the node

Delete: Case 2 - Node has only one child



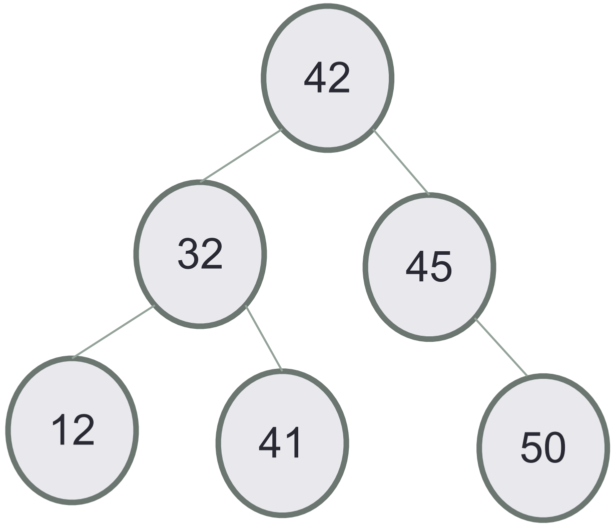
- Replace the node by its only child

Delete: Case 3 - Node has two children



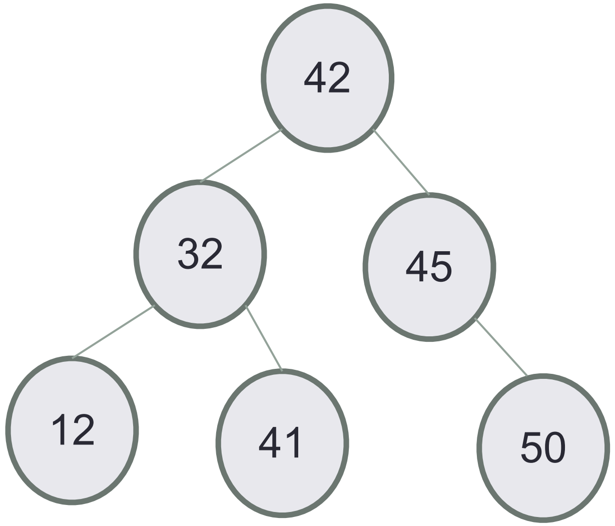
- Can we still replace the node by one of its children? Why or Why not?

In order traversal: print elements in sorted order



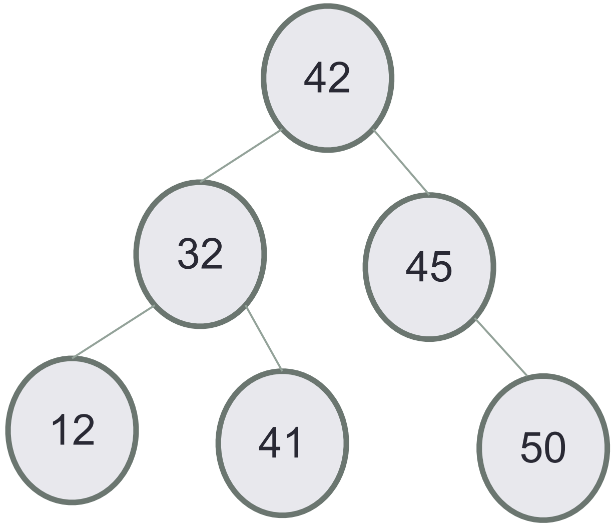
```
Algorithm Inorder(tree)
   1. Traverse the left subtree, i.e., call Inorder(left-subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e., call Inorder(right-subtree)
```

Pre-order traversal: nice way to linearize your tree!



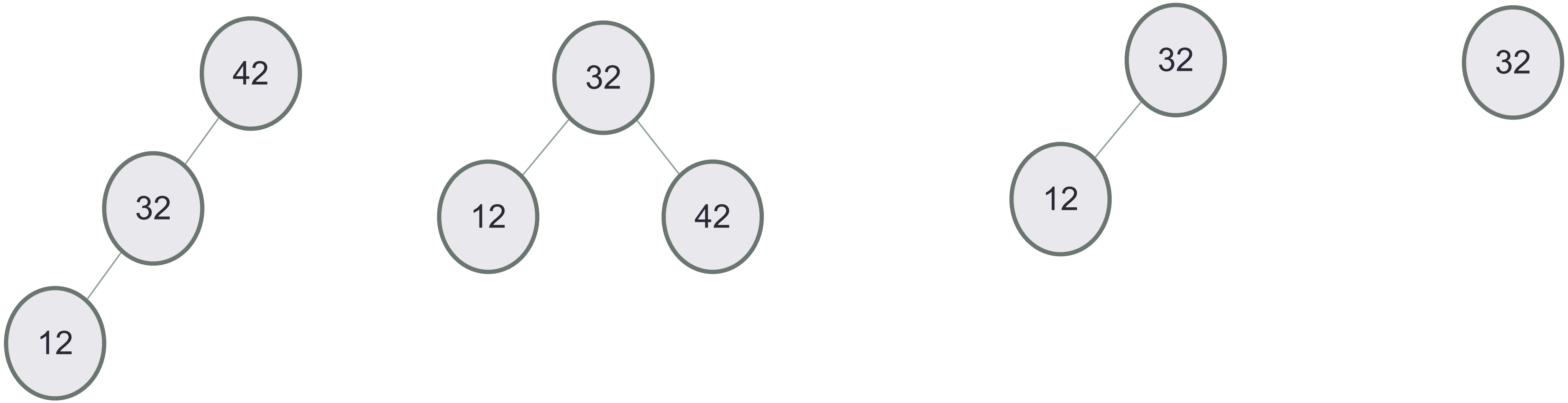
```
Algorithm Preorder(tree)
   1. Visit the root.
   2. Traverse the left subtree, i.e., call Preorder(left-subtree)
   3. Traverse the right subtree, i.e., call Preorder(right-subtree)
```

Post-order traversal: use to recursively clear the tree!



```
Algorithm Postorder(tree)
   1. Traverse the left subtree, i.e., call Postorder(left-subtree)
   2. Traverse the right subtree, i.e., call Postorder(right-subtree)
   3. Visit the root.
```

Write a member function for the BST ADT to compute its height



- How confident are you about your solution and overall approach?

A. Not at all

B. Somewhat confident

C. Very confident

Practice problem

- https://leetcode.com/problems/kth-smallest-element-in-a-bst/description/

```
Input:tree on the right, k = 3
Output: 3
```

Constraints:

- The number of nodes in the tree is `n`.
- `1 <= k <= n <= 10^4`
- `0 <= Node.val <= 10^4`