R-Way Trie

de la Briandais (1959) and Fredkin (1960)

Introduction

- An r-way trie (retrieval) is an alternate way to implement a hash table. Therefore, items are inserted, removed, and retrieved from the trie based on <key,value> pairs.

- Each successive character of the key such as:
  - a digit where the key is a number
  - a character where the key is a string

  determines which of r ways to proceed down the tree.

- Values are stored at nodes that are associated with a key.

Data structure

```
class Trie<T> : ITrie<T>
{
    private Node root;           // Root node of the Trie

    private class Node
    {
        public T value;          // Value associated with a key;
                                 // otherwise, default
        public int numValues;    // Number of descendent values of a Node
        public Node[] child;     // Branch for each letter 'a' .. ‘z’
        ...
    }
    ...
}
```

Key methods (pun intended)

bool Insert (string key, T value)

Insert a <key,value> pair and return true if successful; false otherwise

Duplicate keys are not permitted

bool Remove (string key)

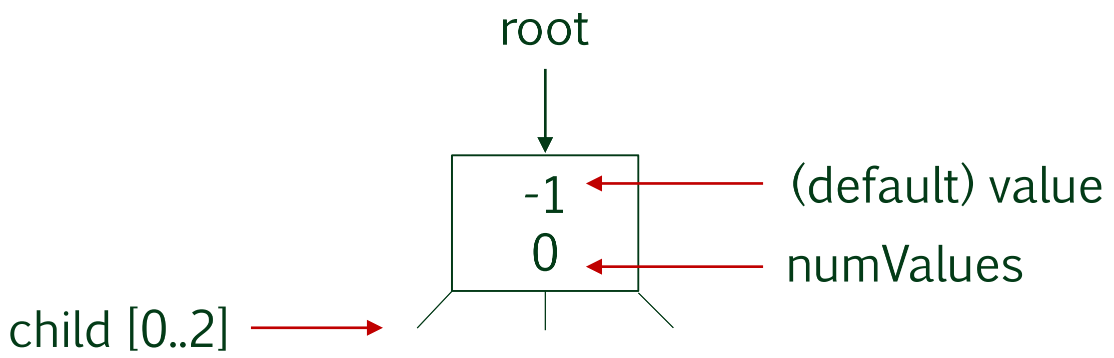
Remove the <key,value> and return true if successful; false otherwise

T Value (string key)

Return the value associated with the given key

Constructor where <key, value> is <string, int>

Assume keys are made up of the letters ‘a’, ‘b’ and ‘c’ only

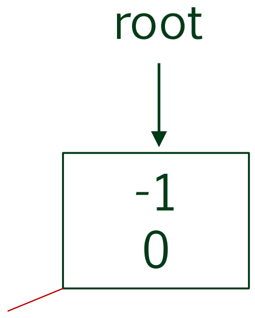


Note: The root node is never removed
(cf header node in a linked list)

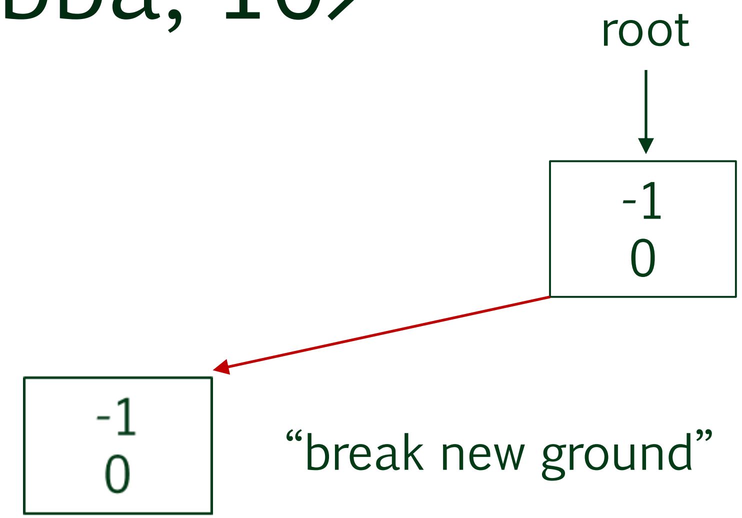
Insert

- Basic strategy
  - Follow the path laid out by the key, “breaking new ground” if need be (i.e. creating new nodes) until the full key has been explored
  - The value is then inserted at the final destination (node) unless the key has already been used (i.e. a value already exists for that key)

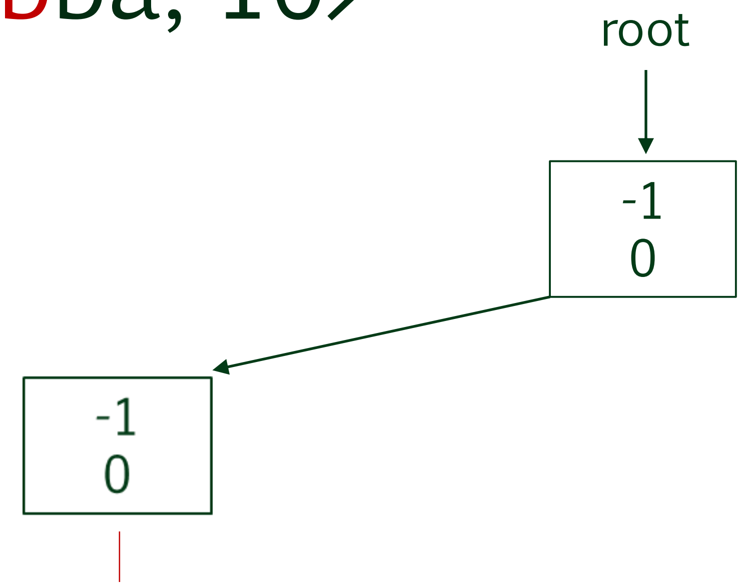
Insert ⟨abba, 10⟩



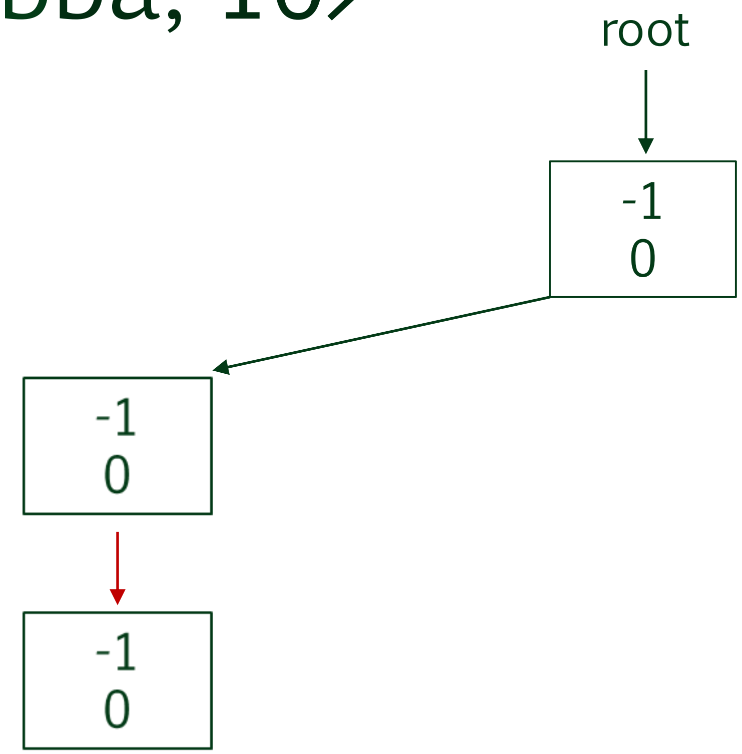
Insert <abba, 10>



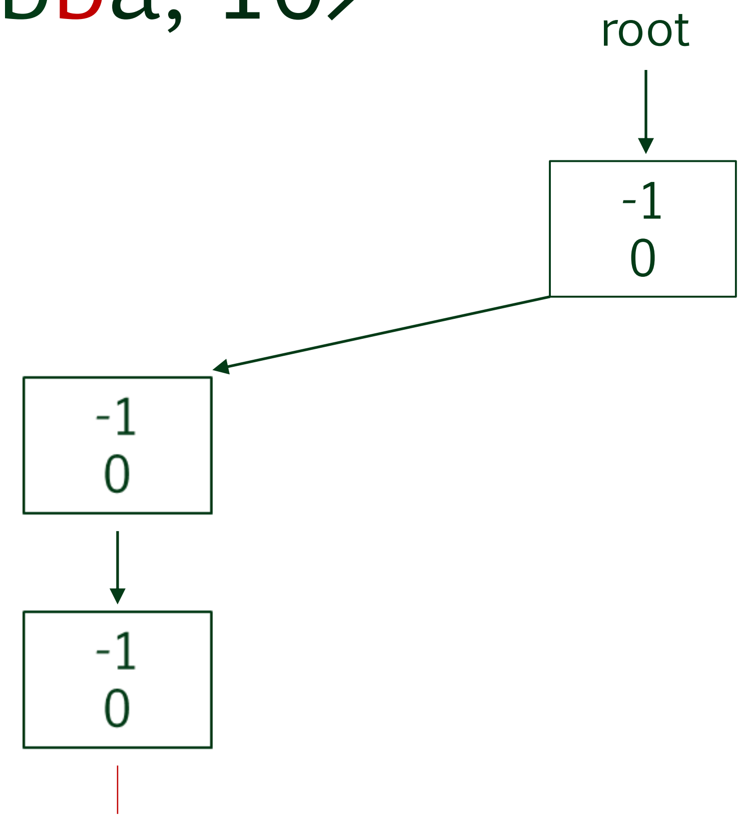
Insert <abba, 10>



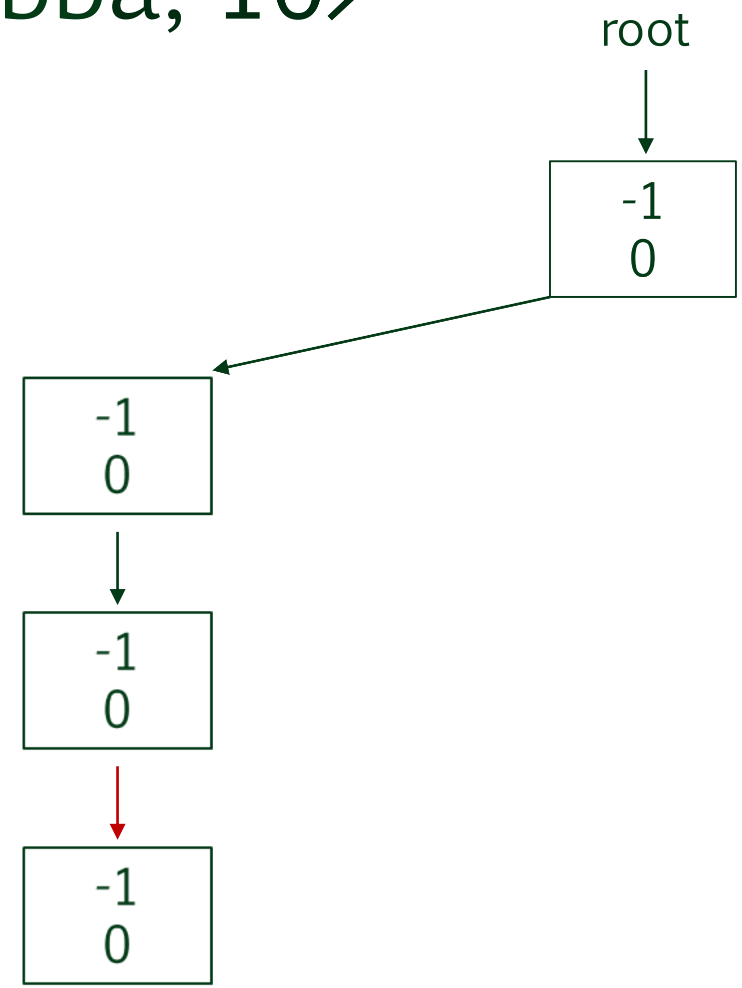
Insert <abba, 10>



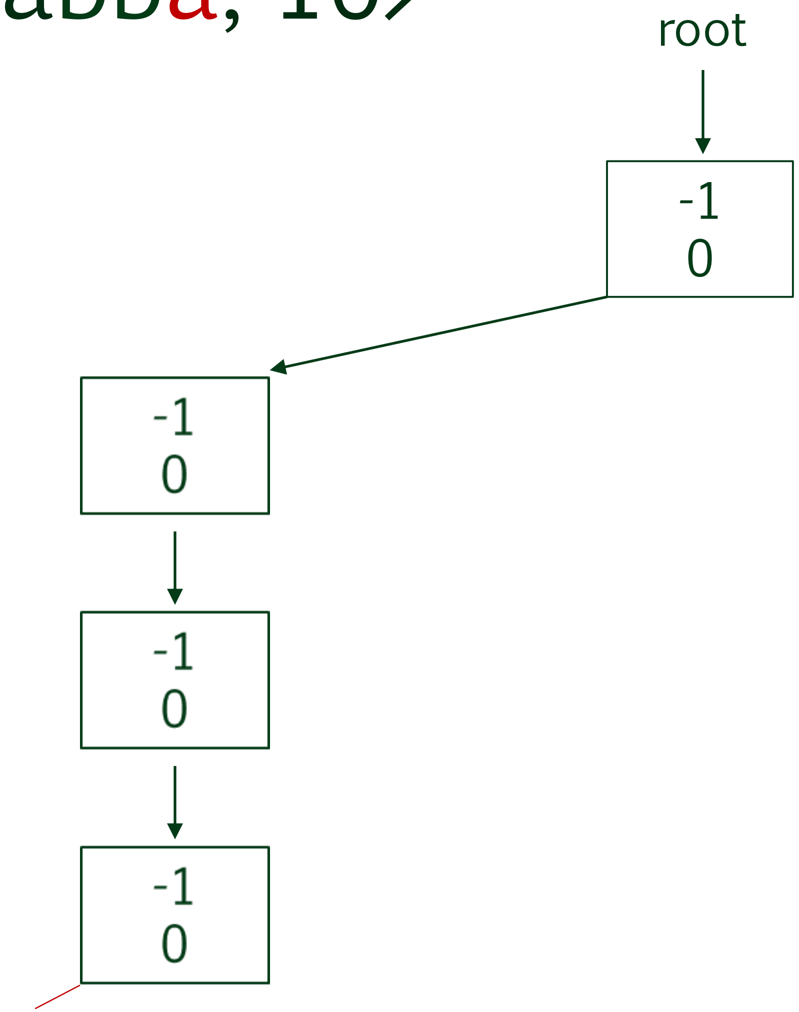
Insert <abba, 10>



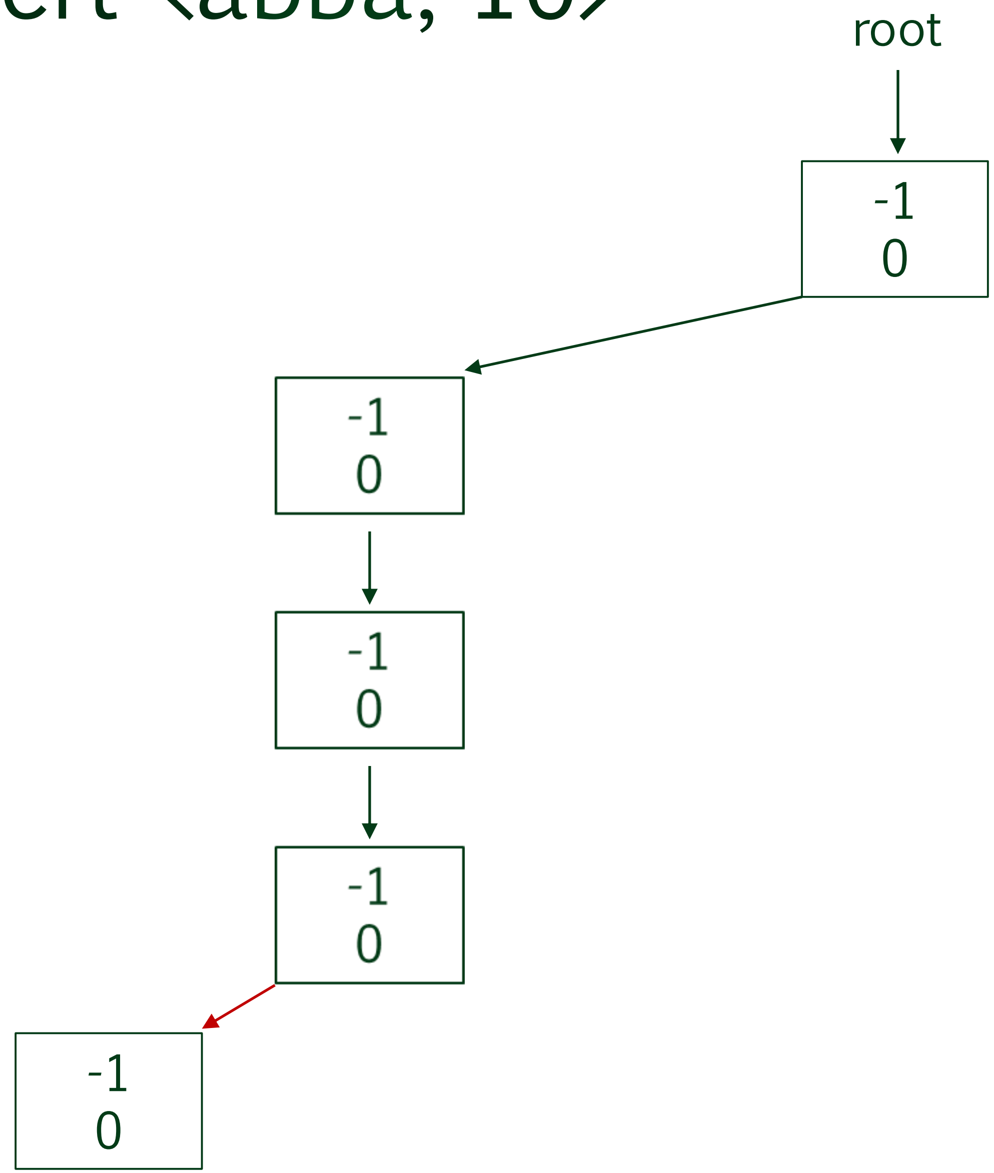
Insert <abba, 10>



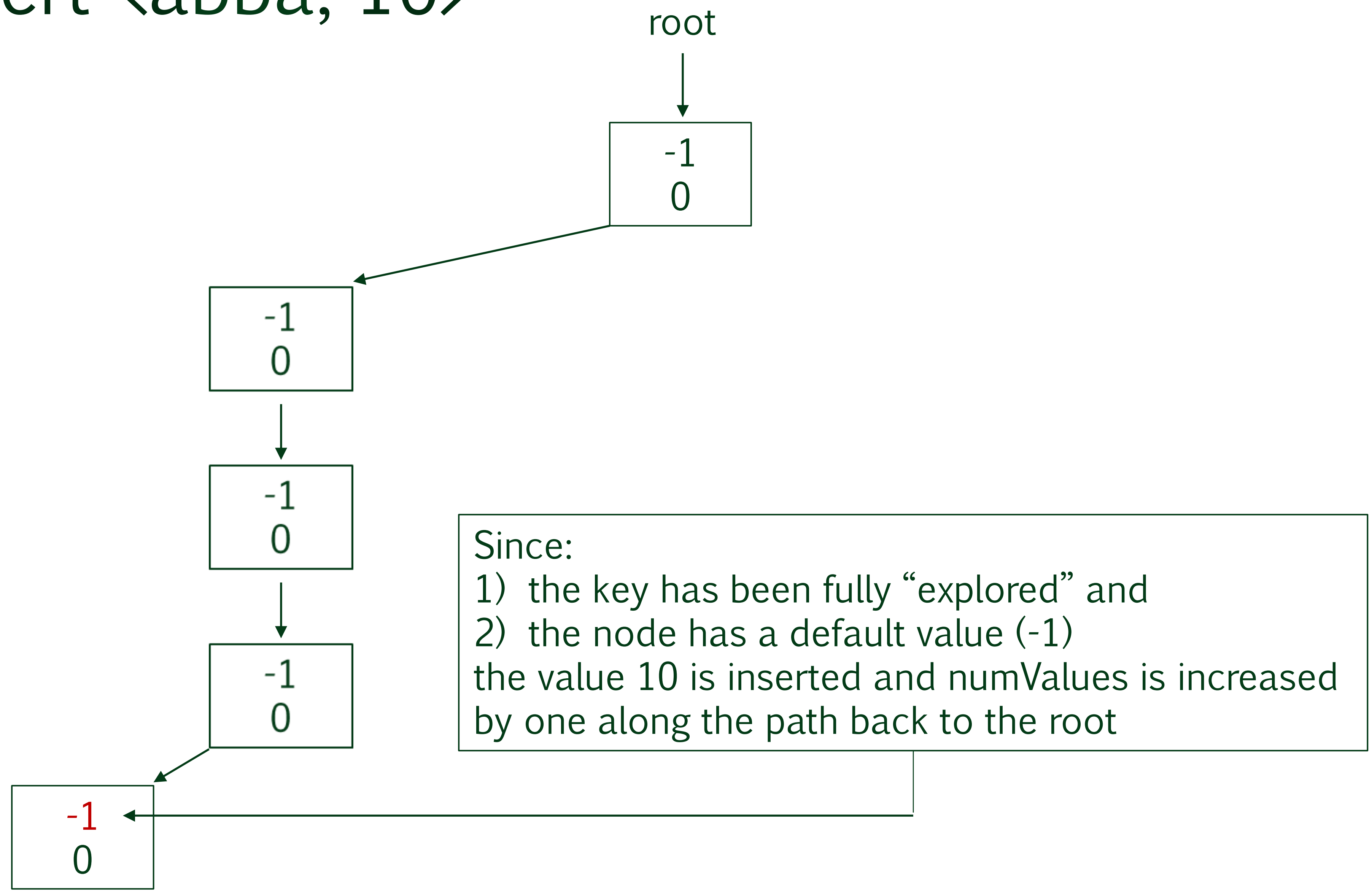
Insert <abba, 10>



Insert <abba, 10>

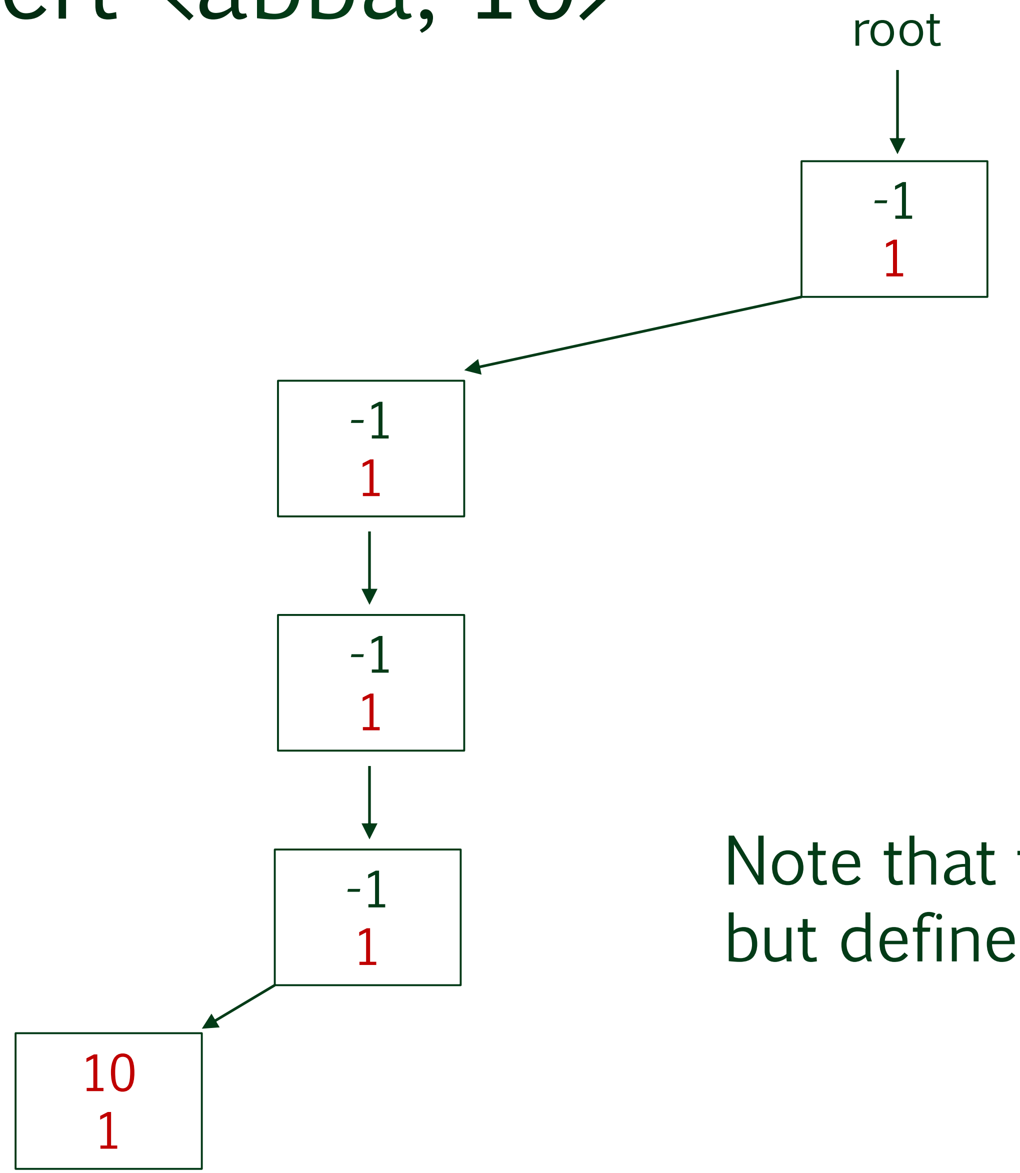


Insert <abba, 10>



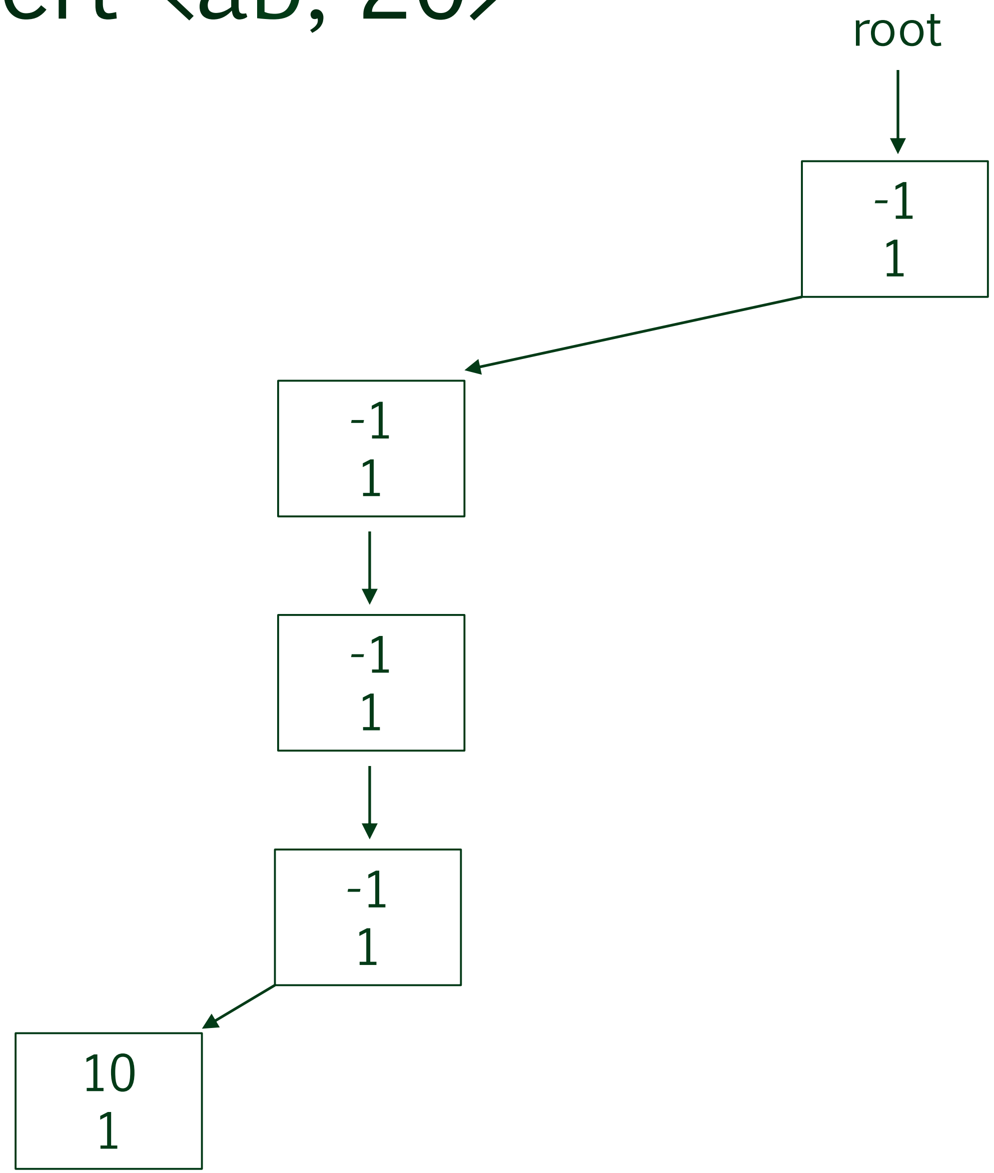
Since:
1) the key has been fully "explored" and
2) the node has a default value (-1)
the value 10 is inserted and numValues is increased by one along the path back to the root

Insert <abba, 10>

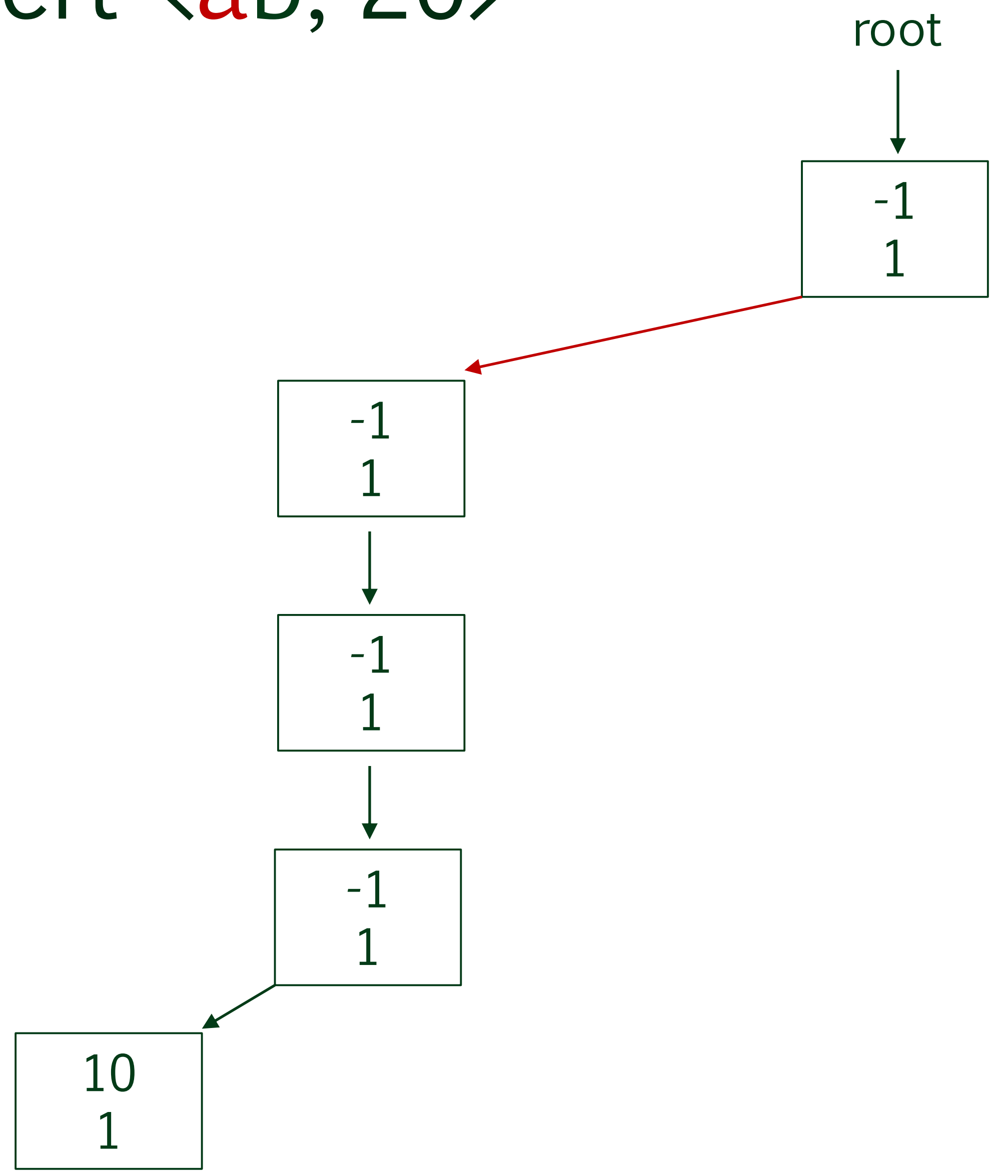


Note that the key itself is NOT stored, but defines the path to a value

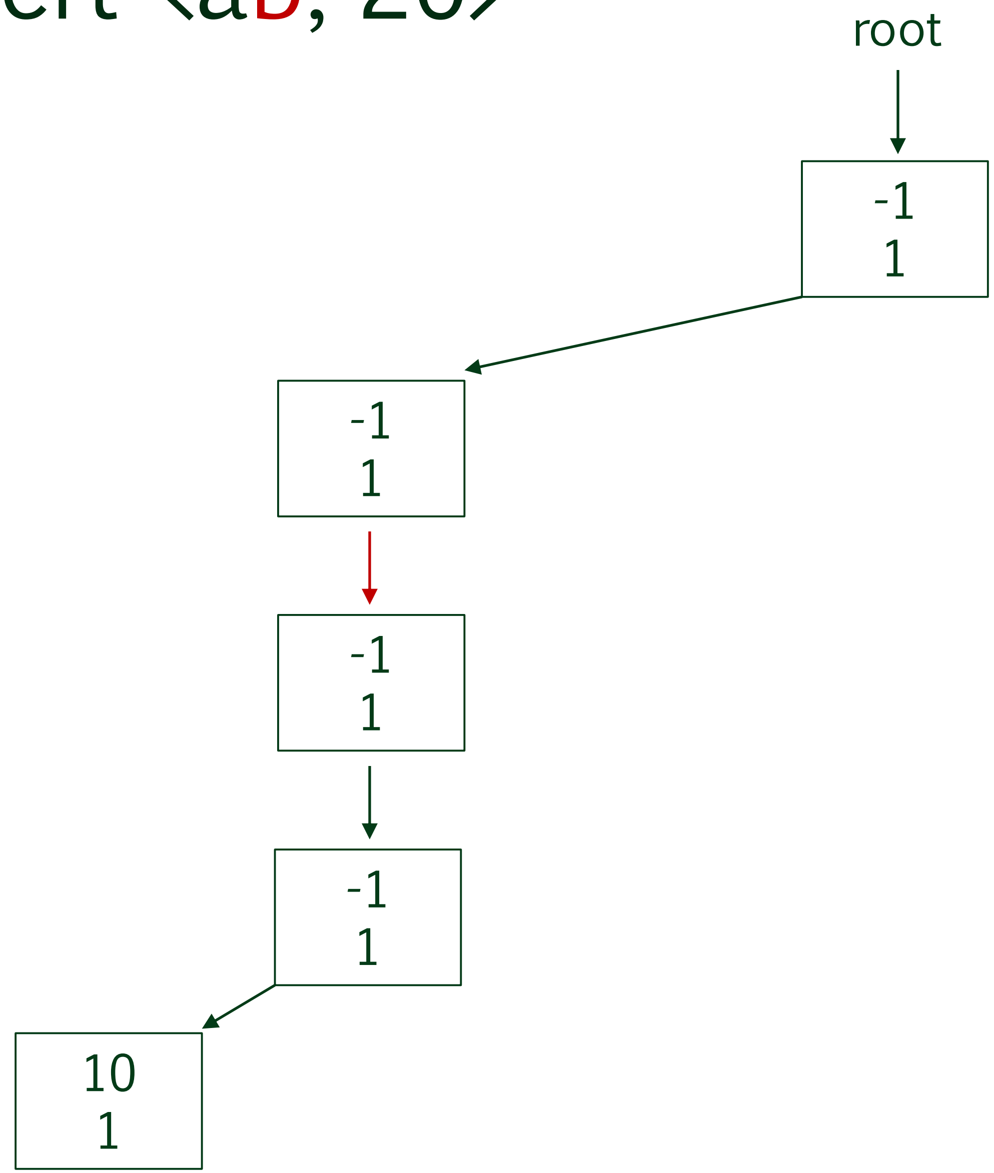
Insert <ab, 20>



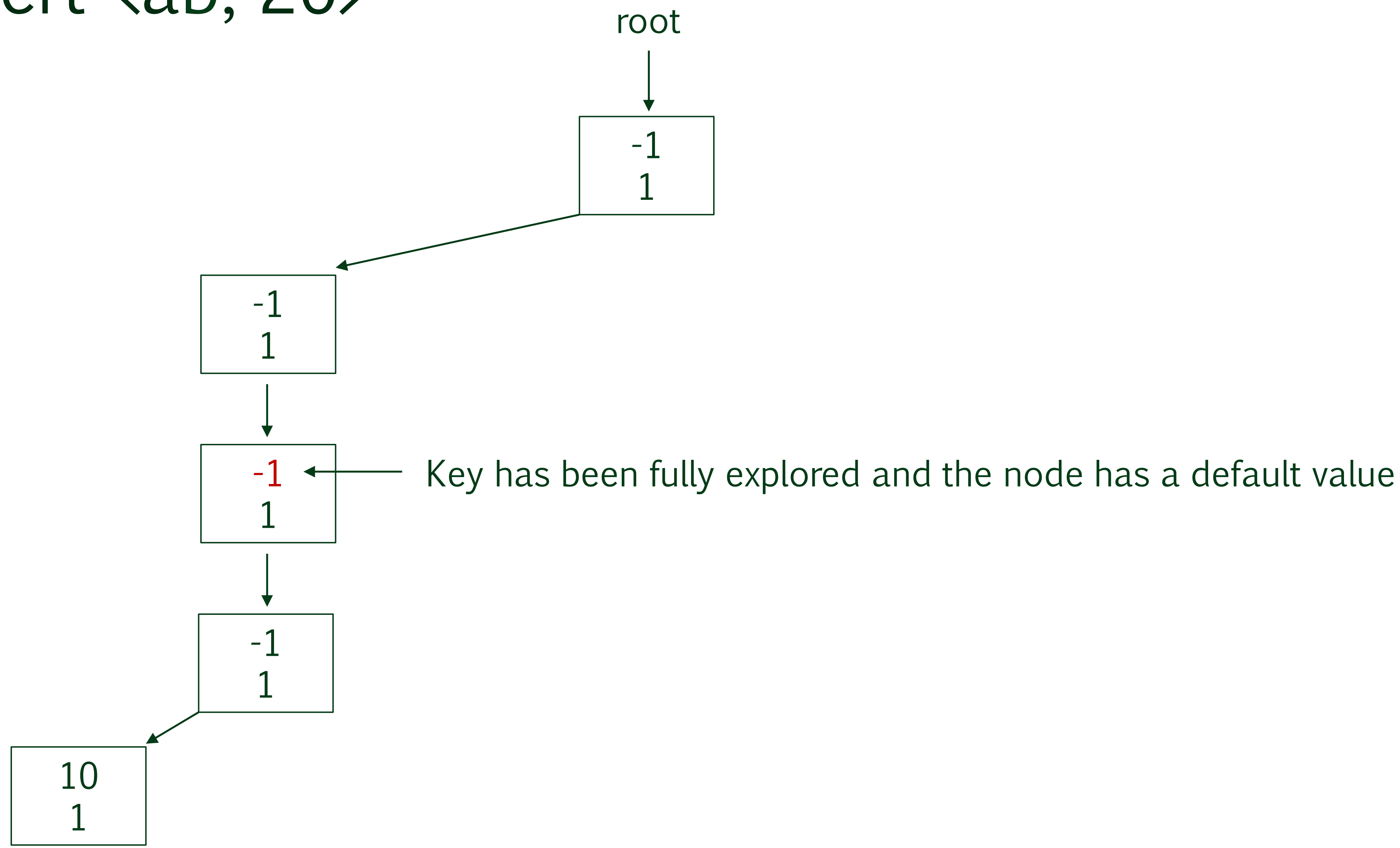
Insert <ab, 20>



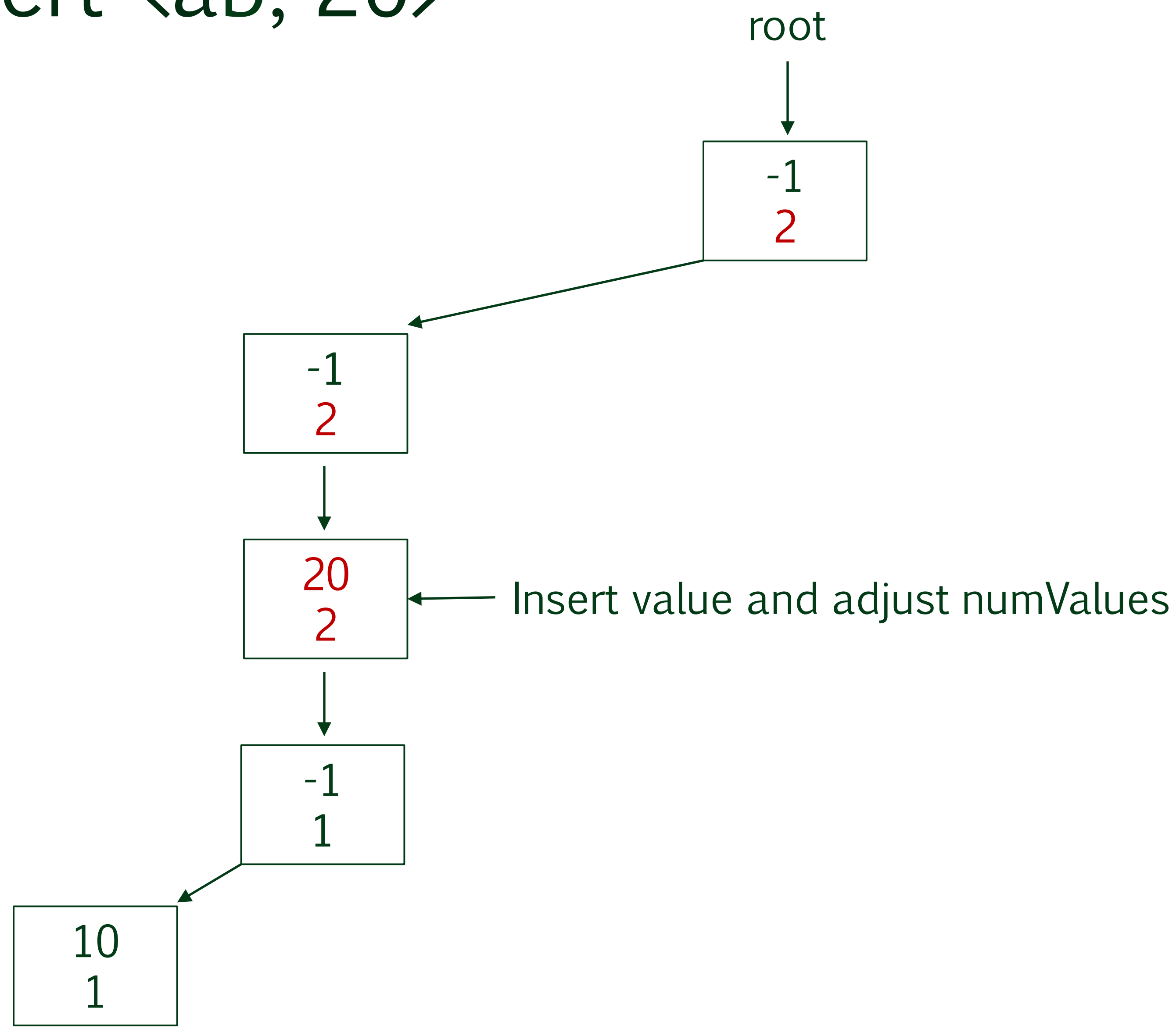
Insert <ab, 20>



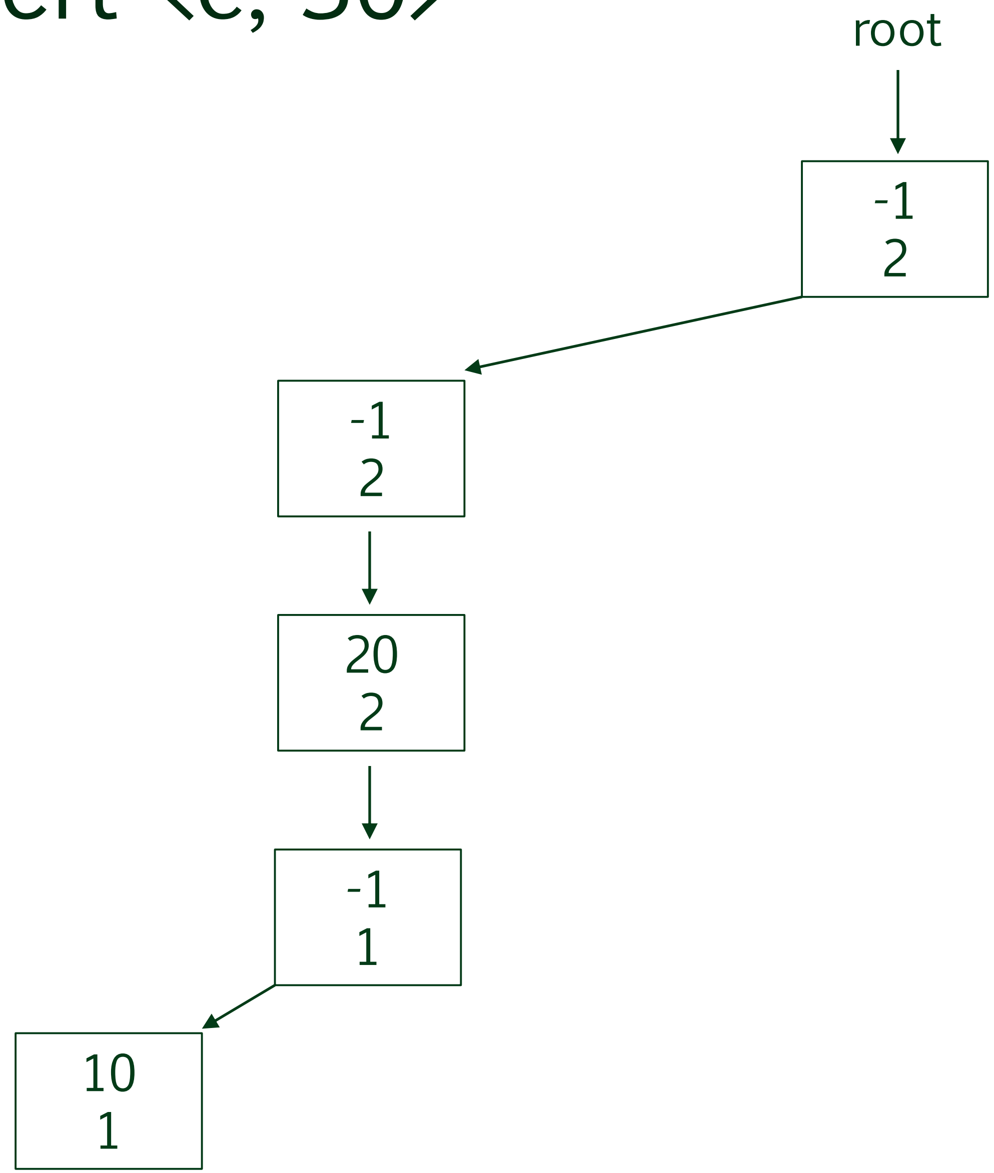
Insert <ab, 20>


root
-1
1
-1
1
-1
1
Key has been fully explored and the node has a default value
-1
1
10
1

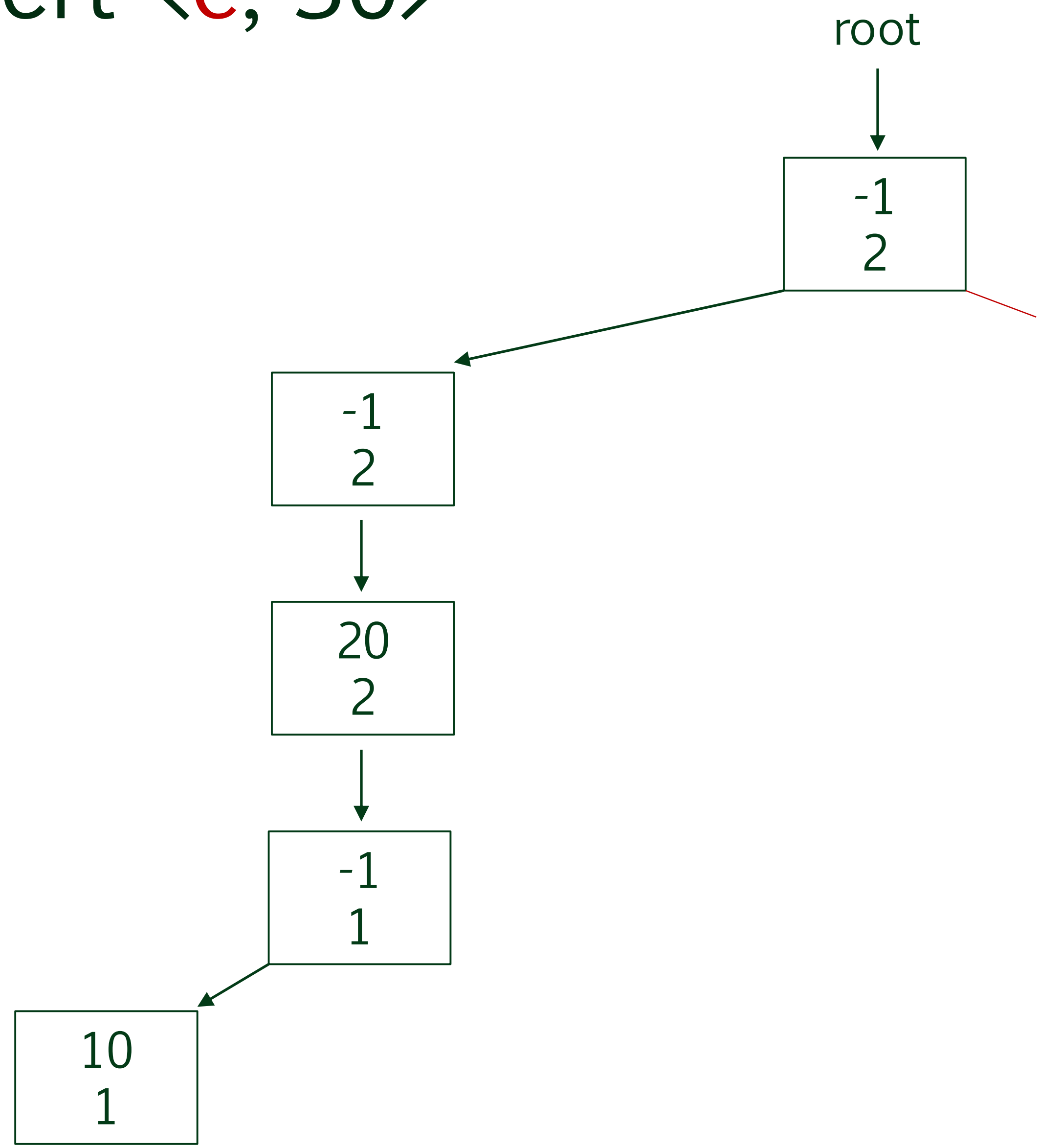
Insert <ab, 20>



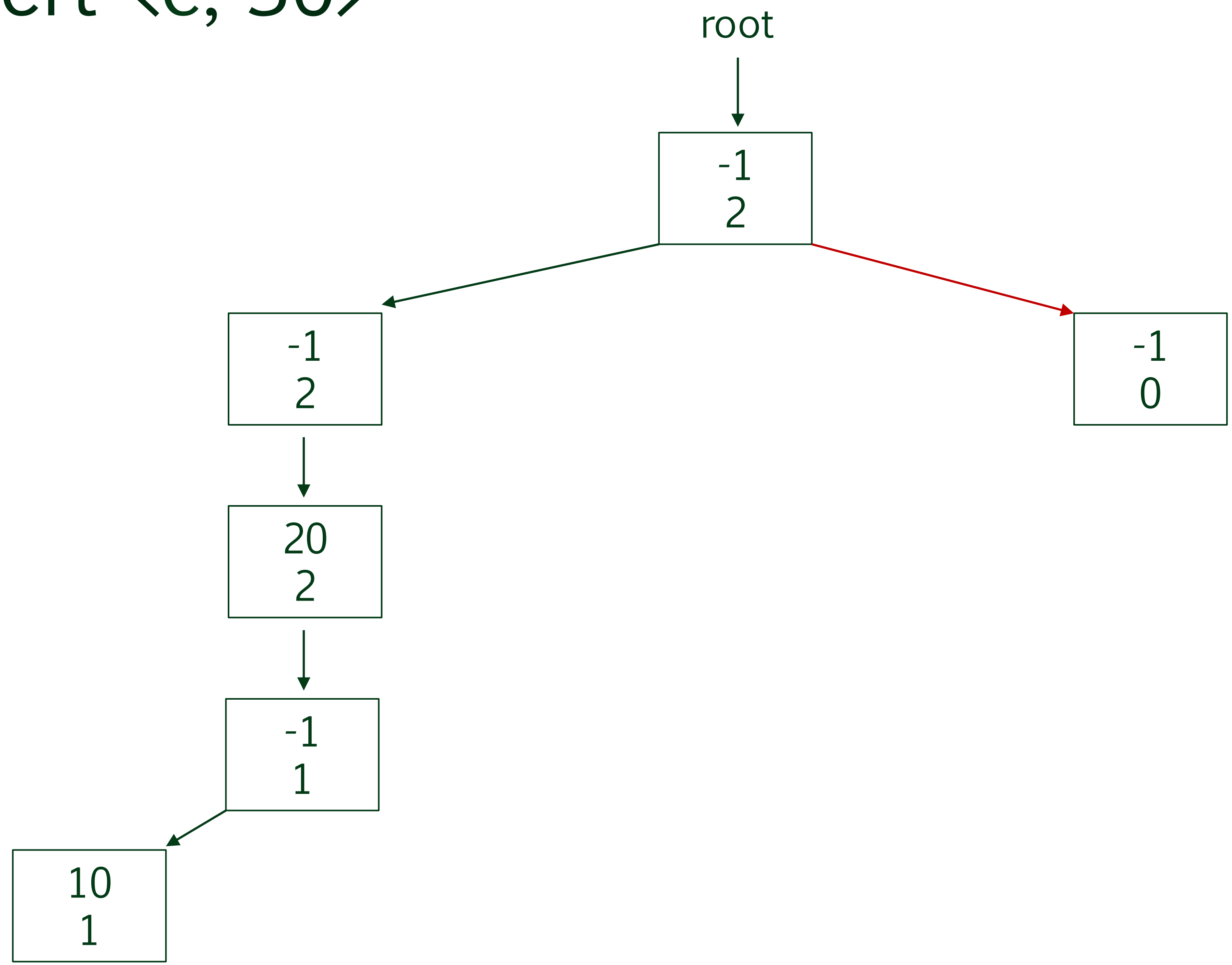
Insert <c, 30>



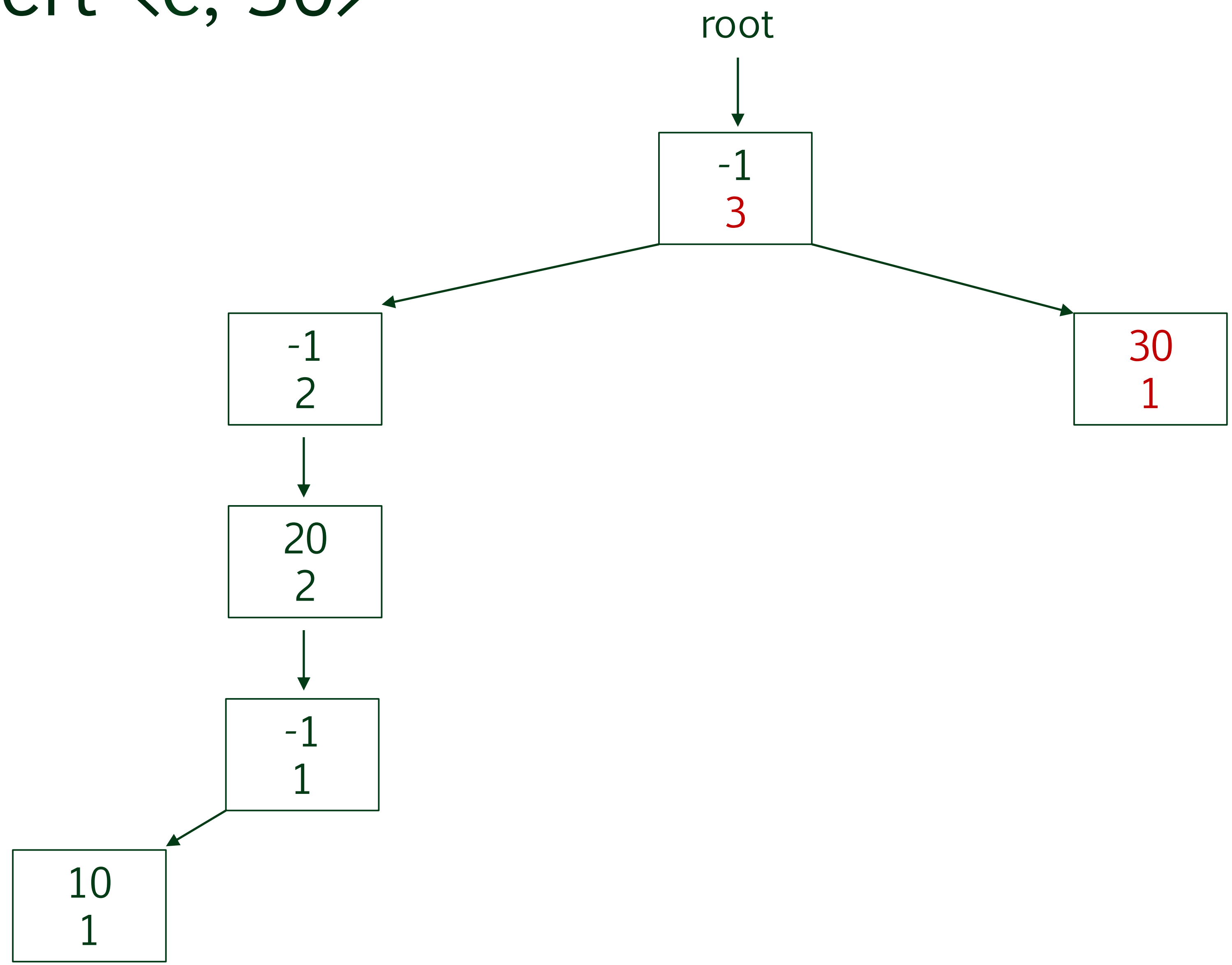
Insert <c, 30>



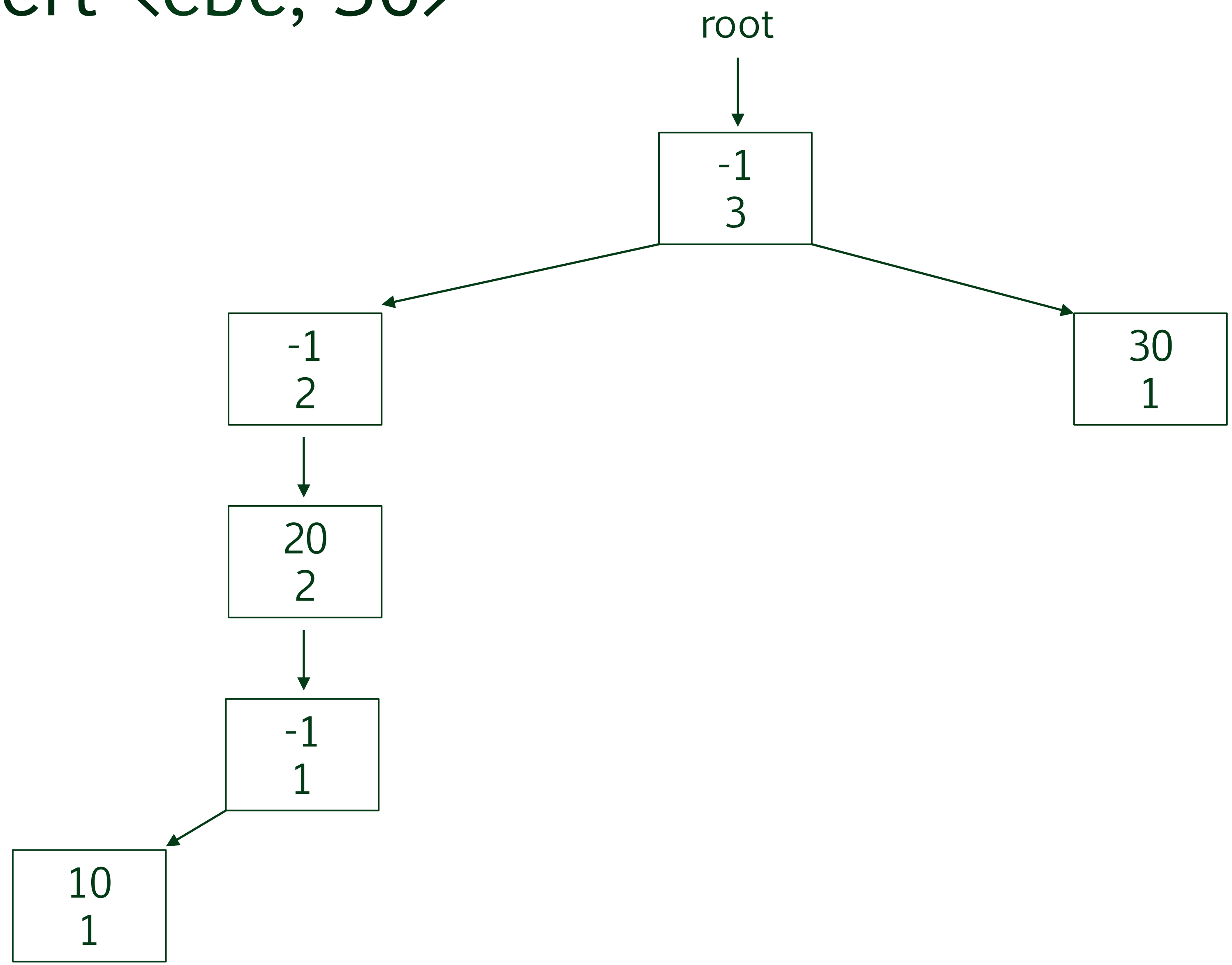
Insert <c, 30>



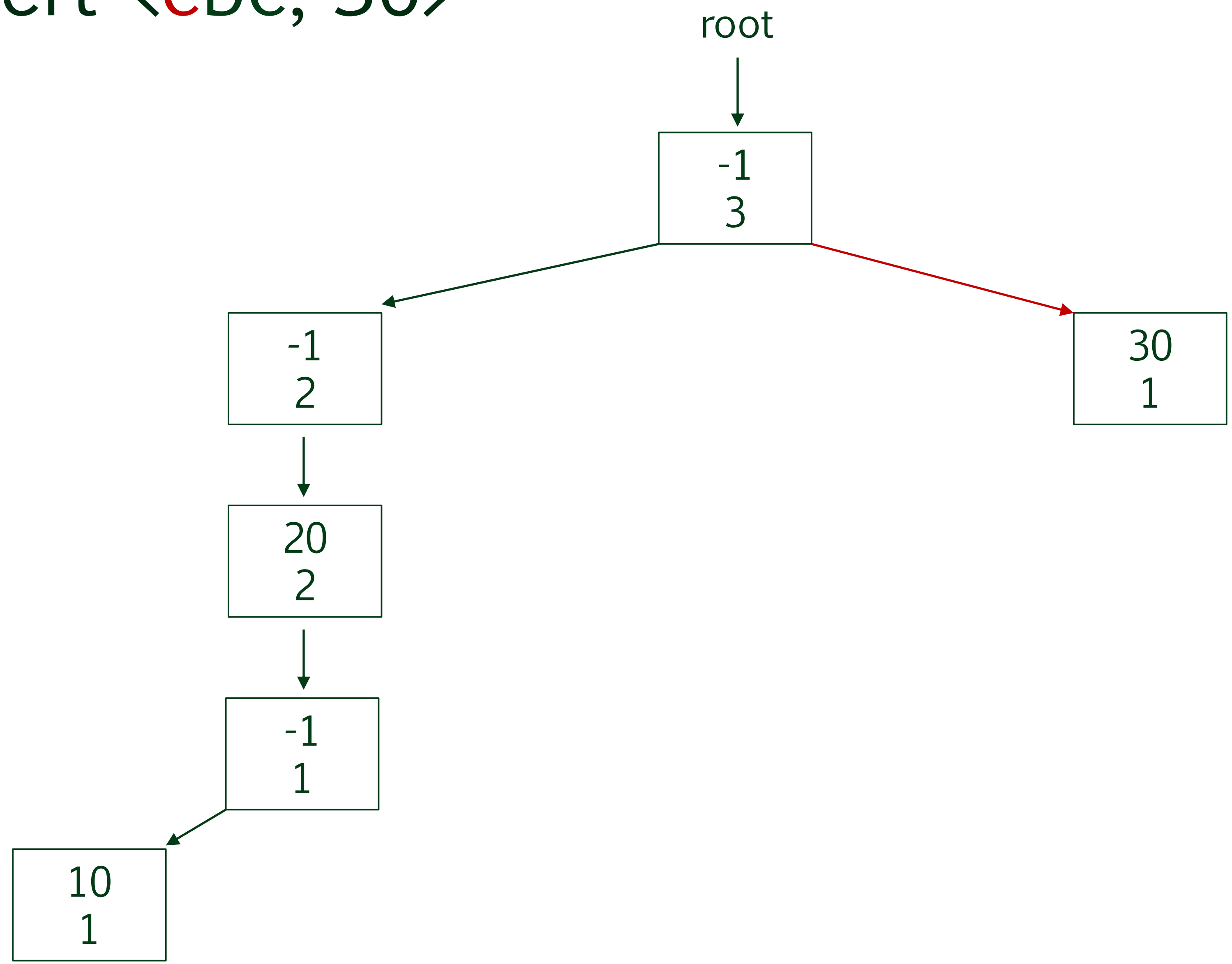
Insert <c, 30>



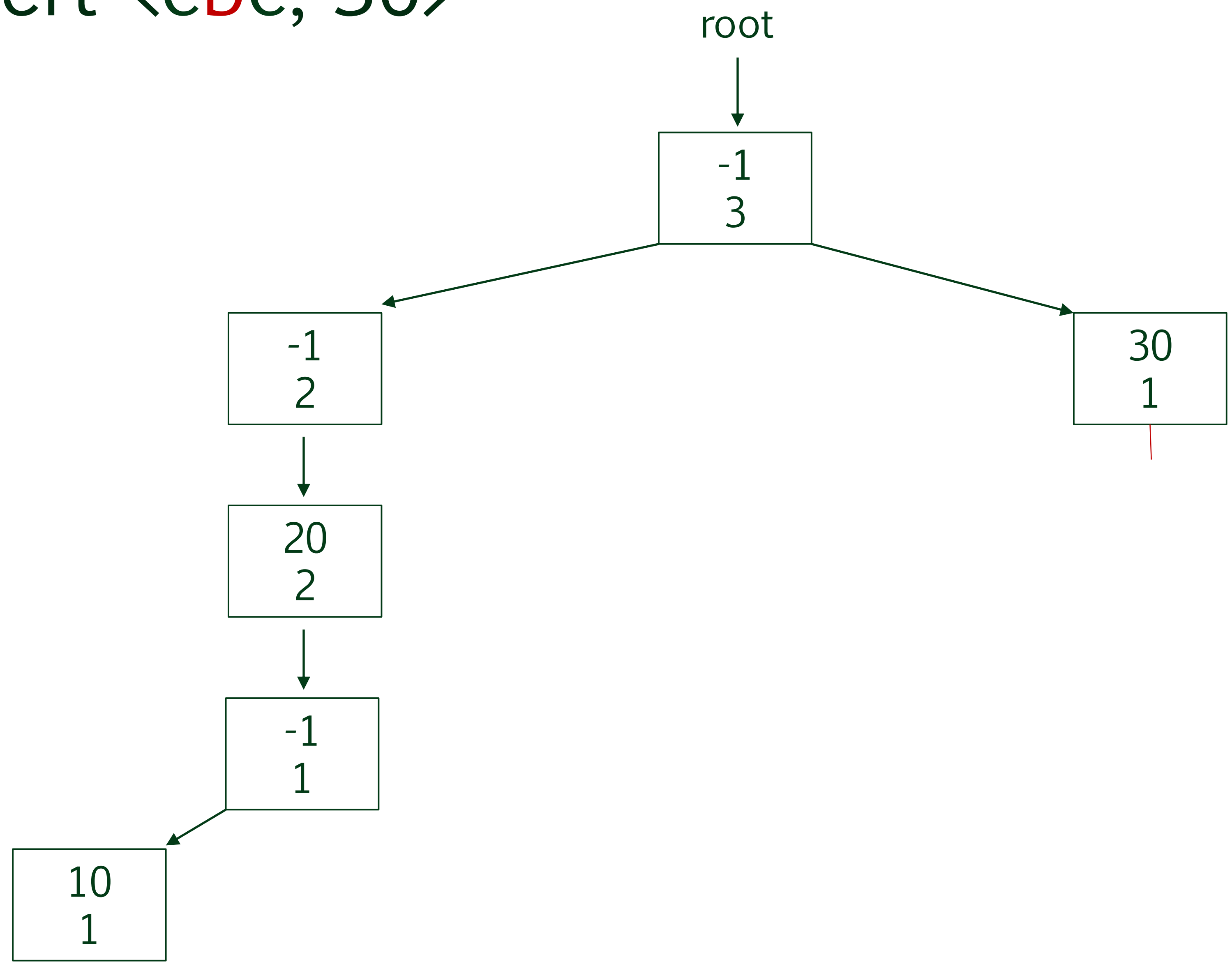
Insert <cbc, 30>



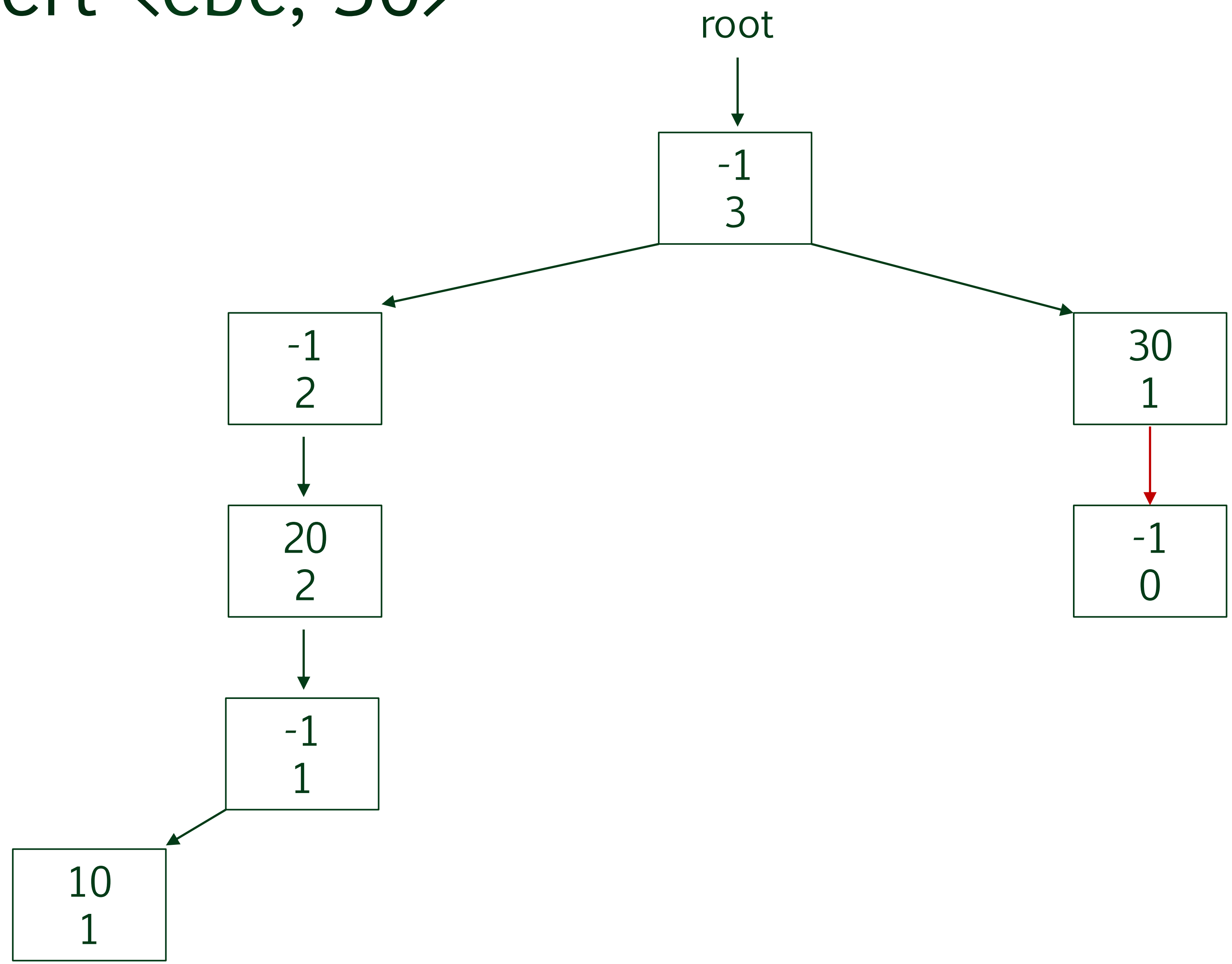
Insert <cbc, 30>



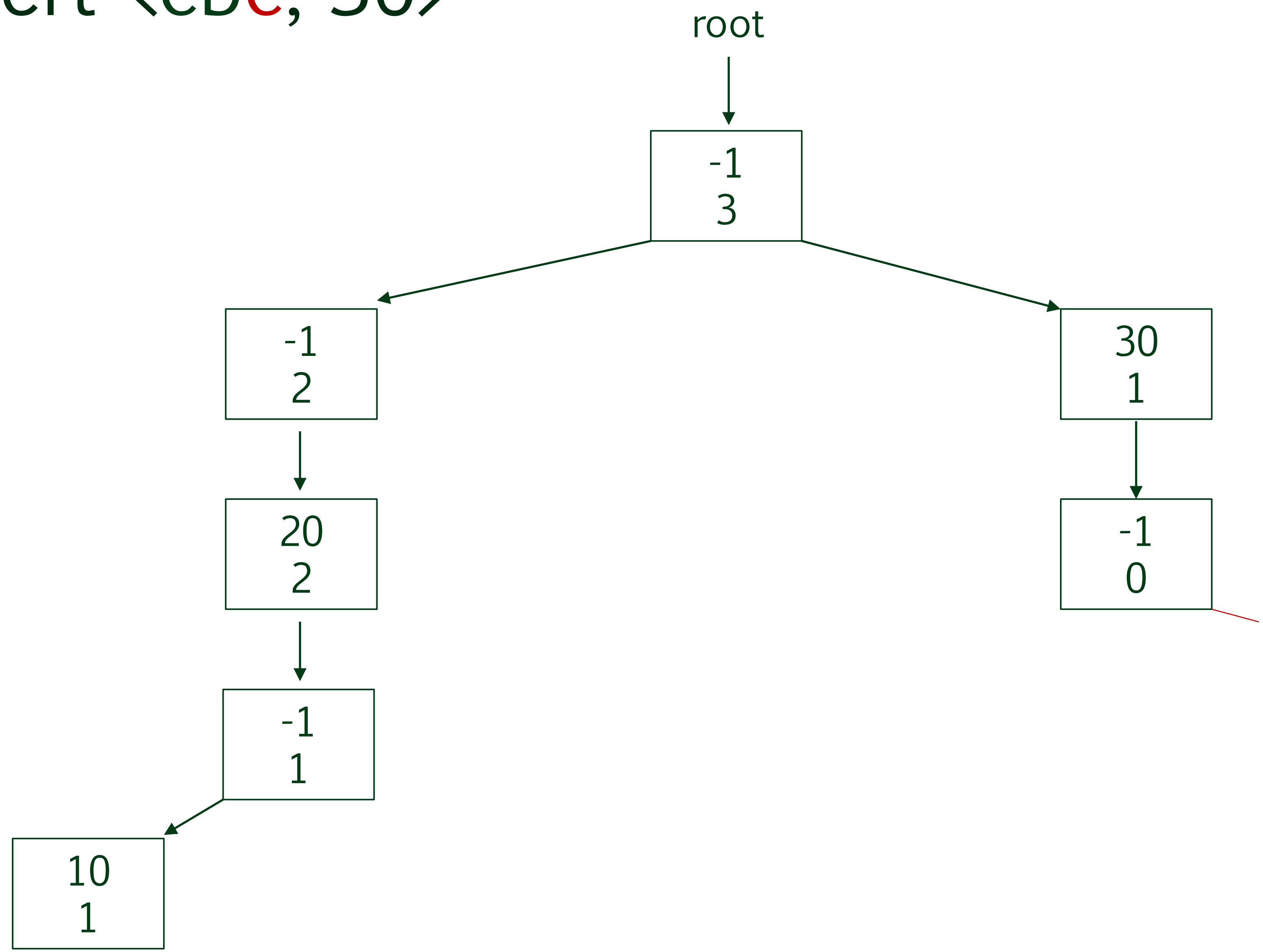
Insert <cbc, 30>



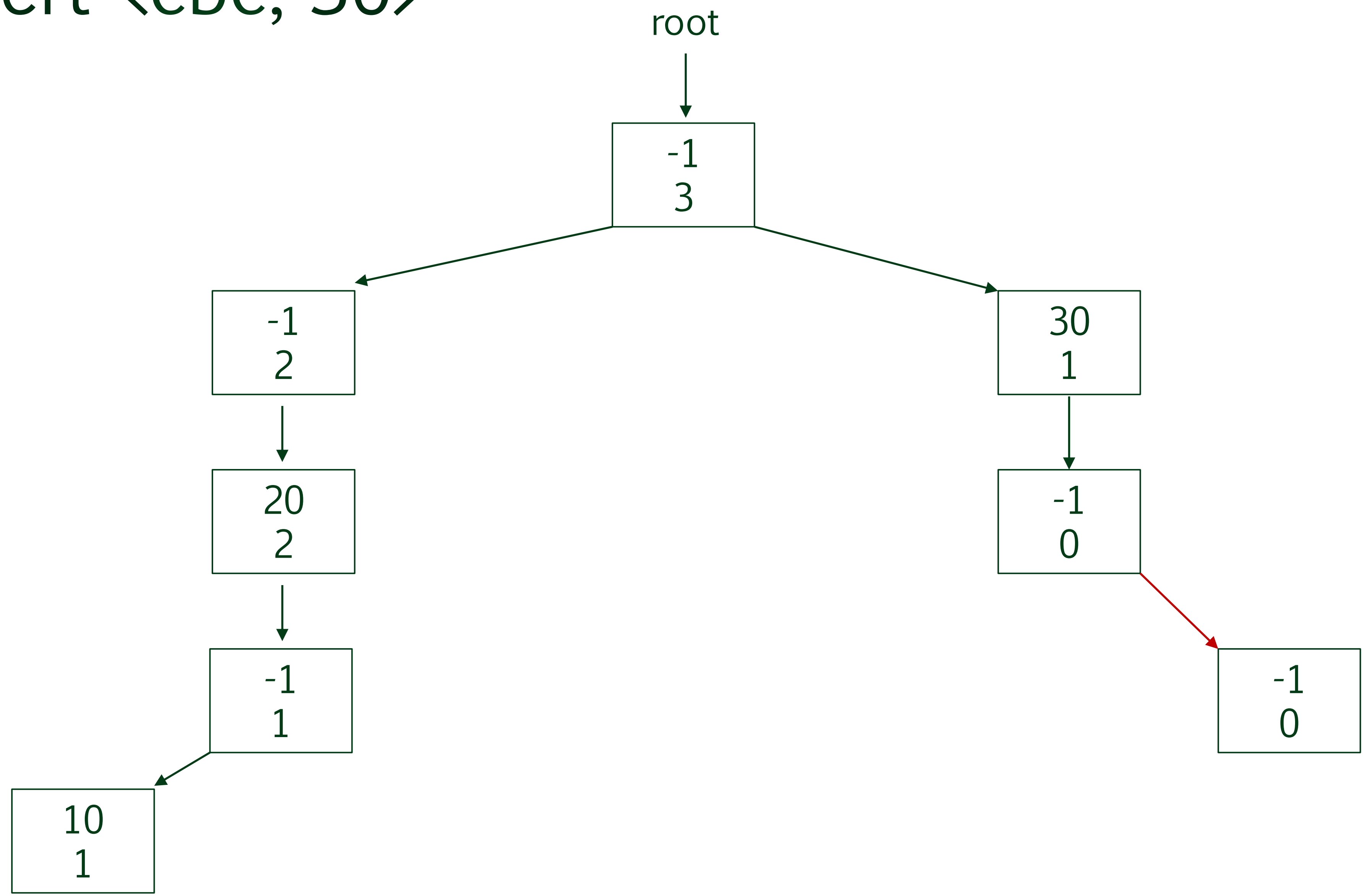
Insert <cbc, 30>



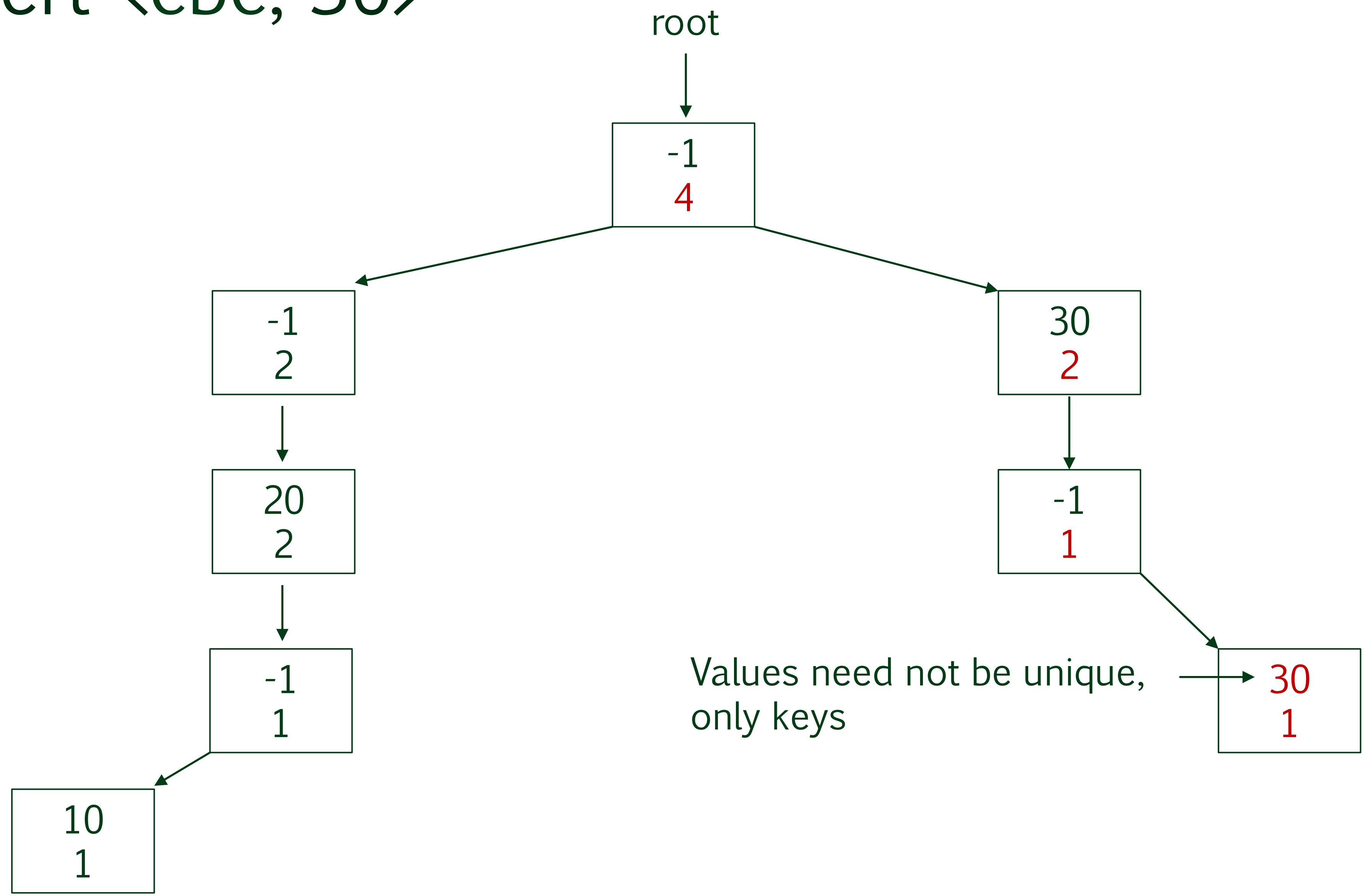
Insert <cbc, 30>



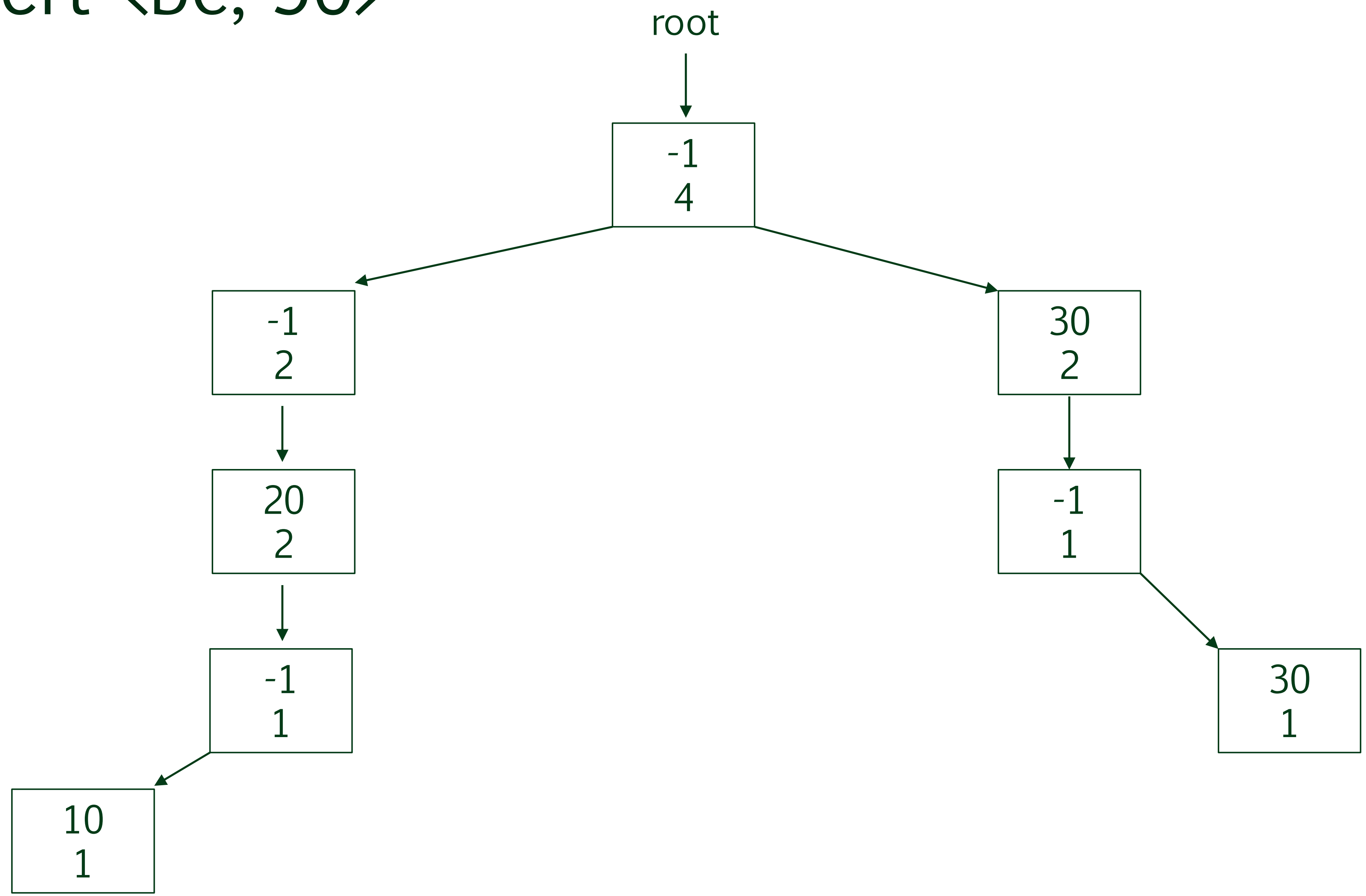
Insert <cbc, 30>



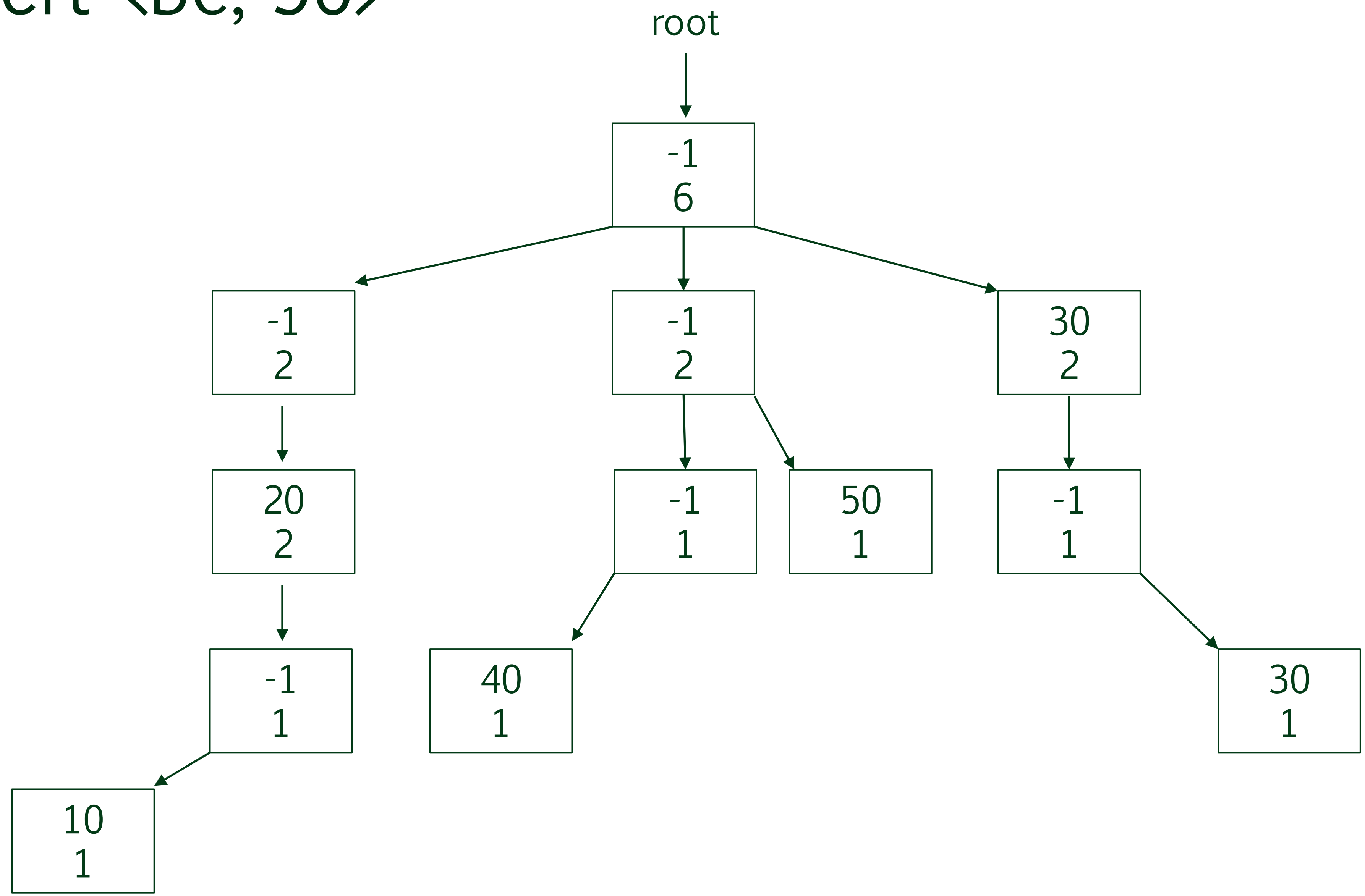
Insert <cbc, 30>



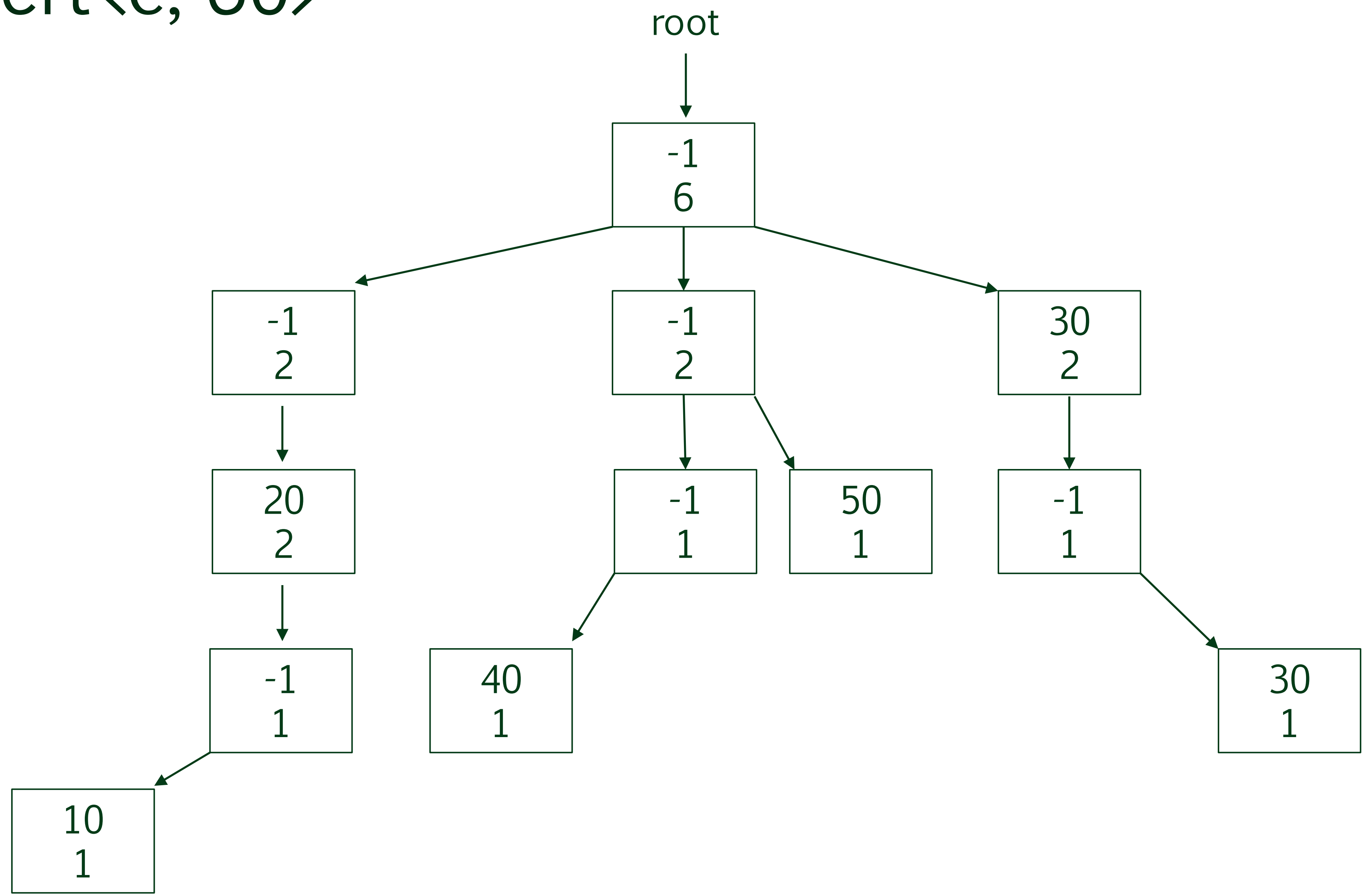
Insert <bba, 40>
Insert <bc, 50>



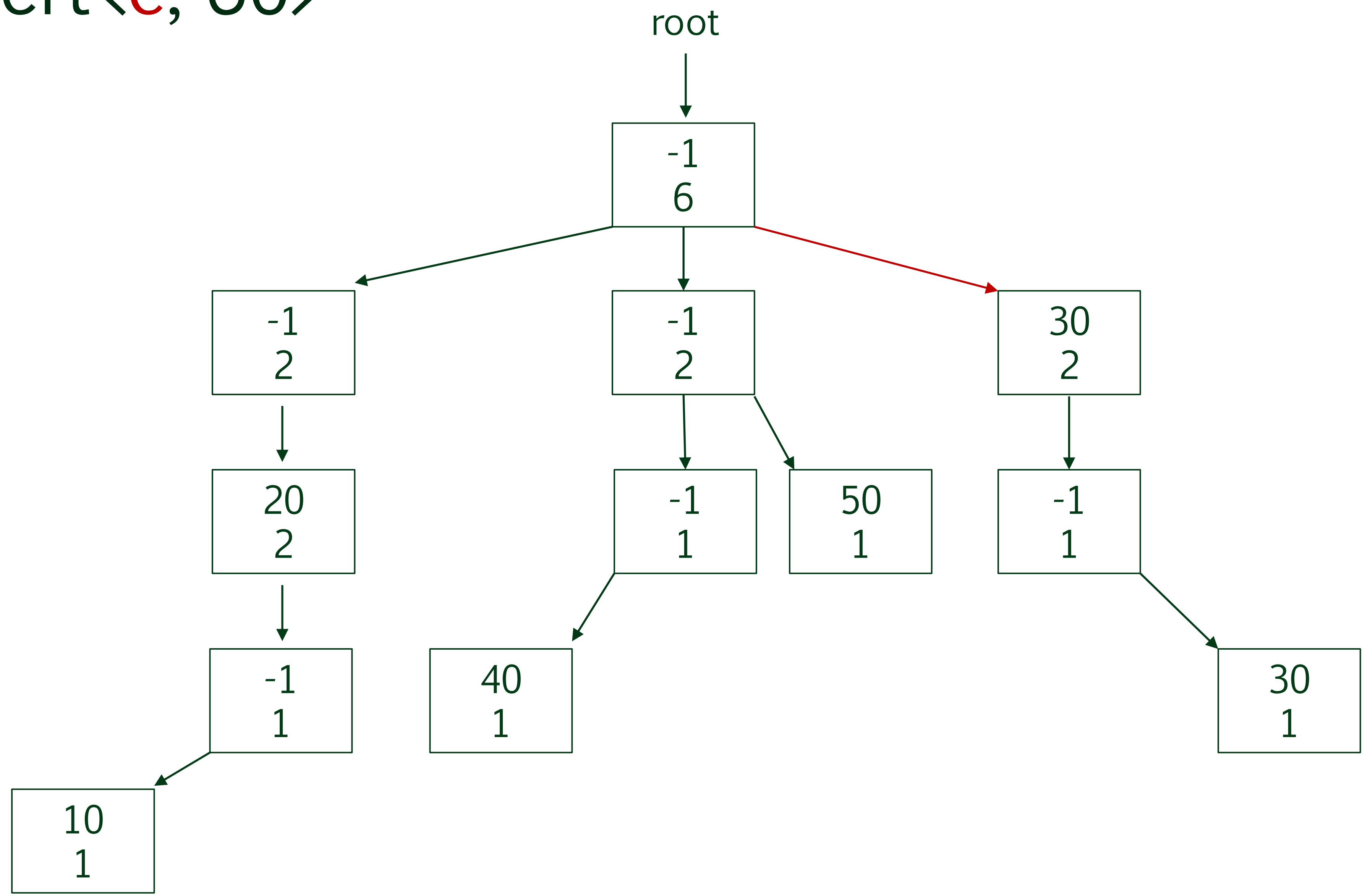
Insert <bba, 40>
Insert <bc, 50>



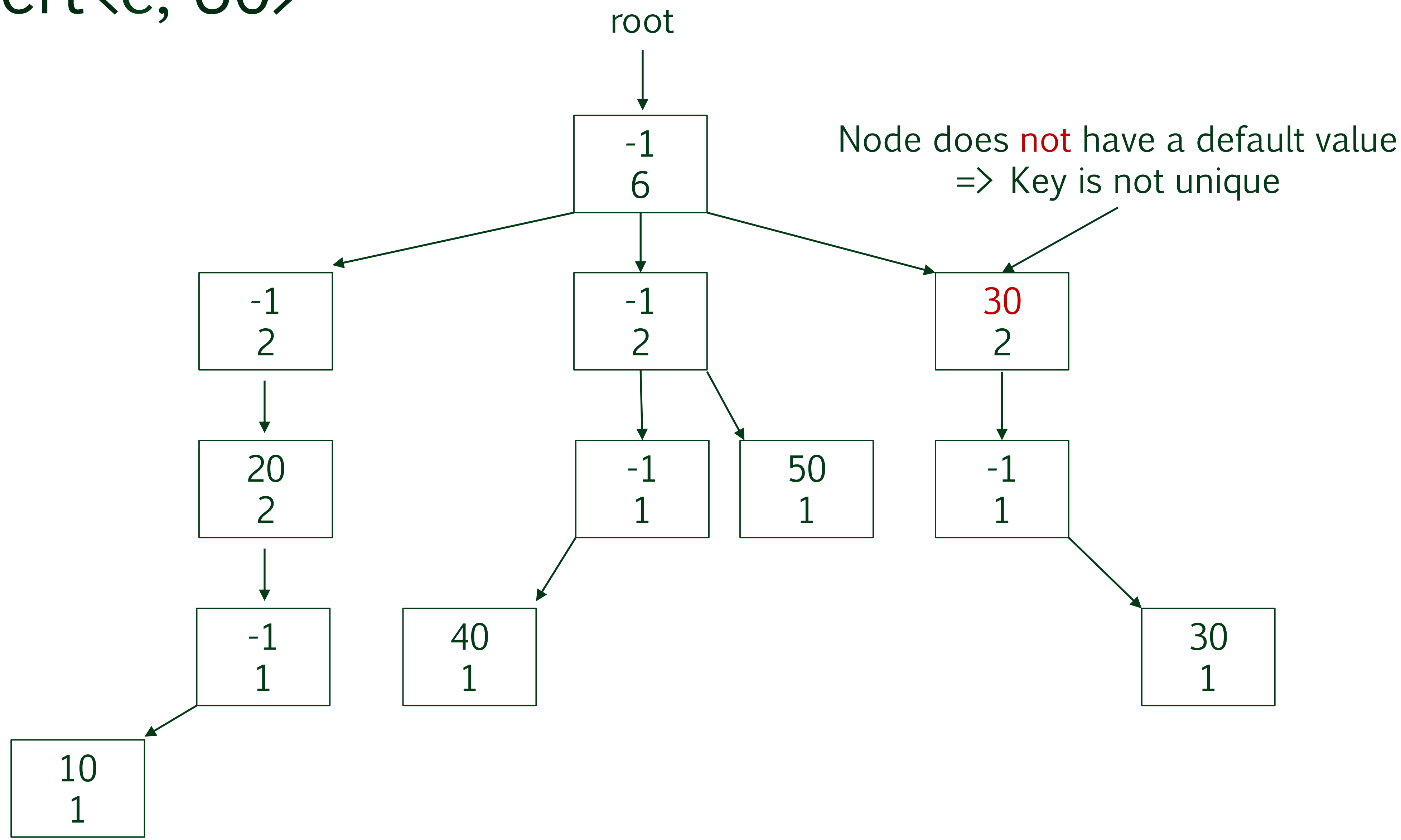
Insert<c, 60>



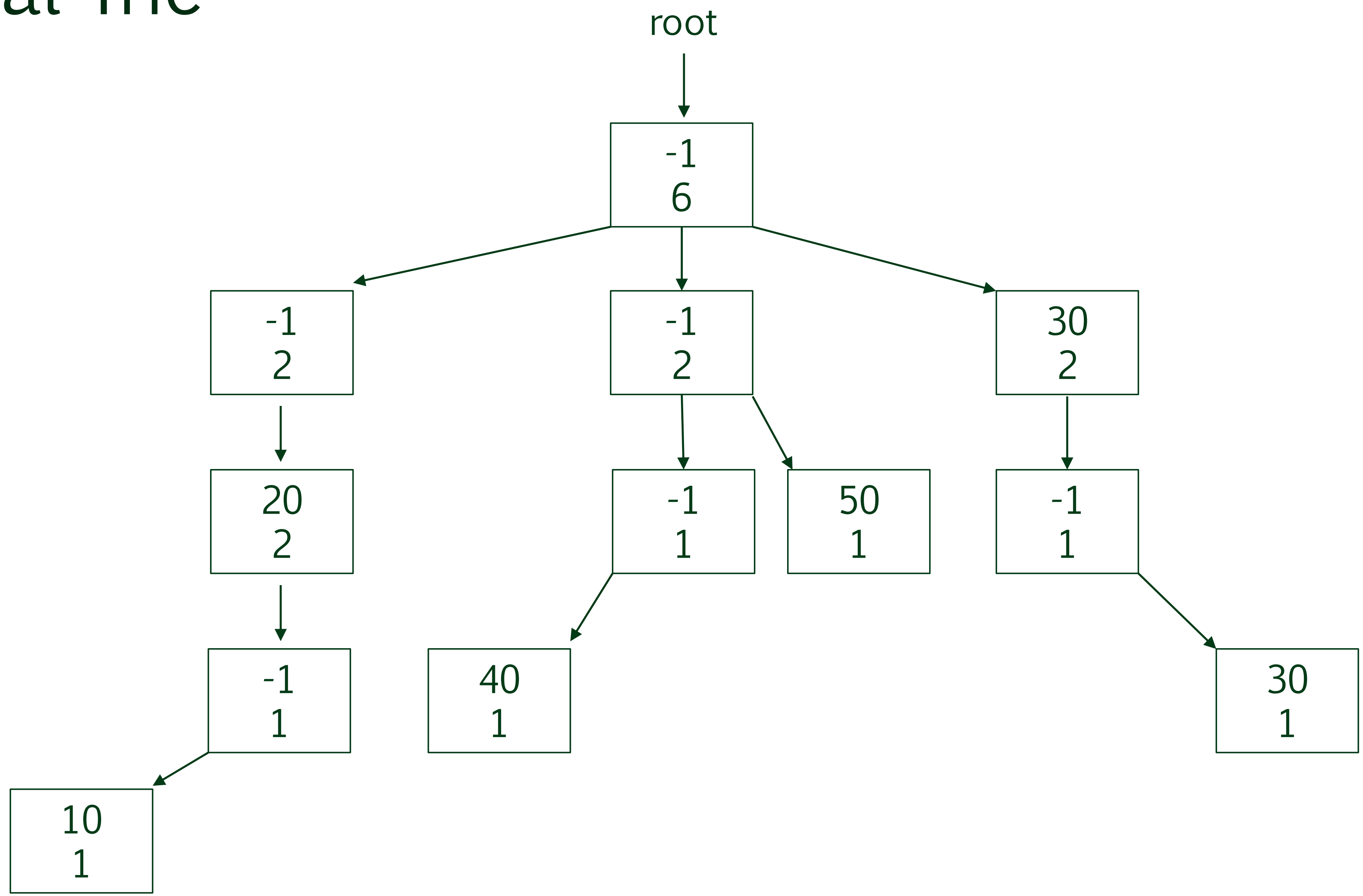
Insert<c, 60>



Insert<c, 60>


root
-1
6
Node does not have a default value
=> Key is not unique
-1
2
-1
2
30
2
20
2
-1
1
50
1
-1
1
-1
1
40
1
30
1
10
1

Final Trie



Time complexity

- The path to insert a value is equal to the length of the key
- Therefore, the time complexity of Insert is O(L) where L is the length of the key

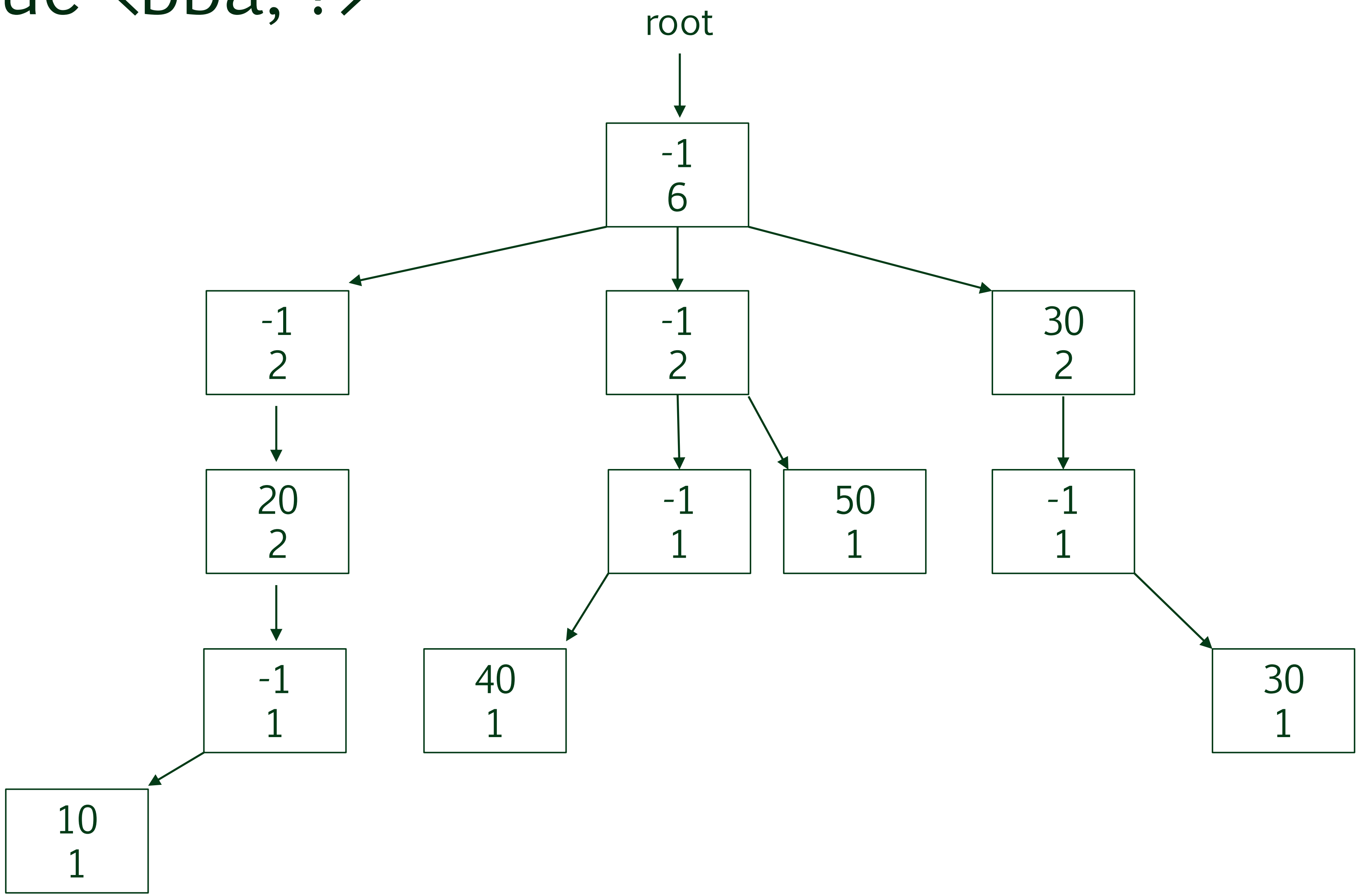
Exercises

- What happens if the key is the empty string? Can the Insert method still work? Show how or why not.
- Is it necessary to examine an entire key before a value is inserted? Show how or why not.
- Insert the following <key,value> pairs into the final trie
  - <bbc, 70>
  - <b, 80>
  - <caa, 90>

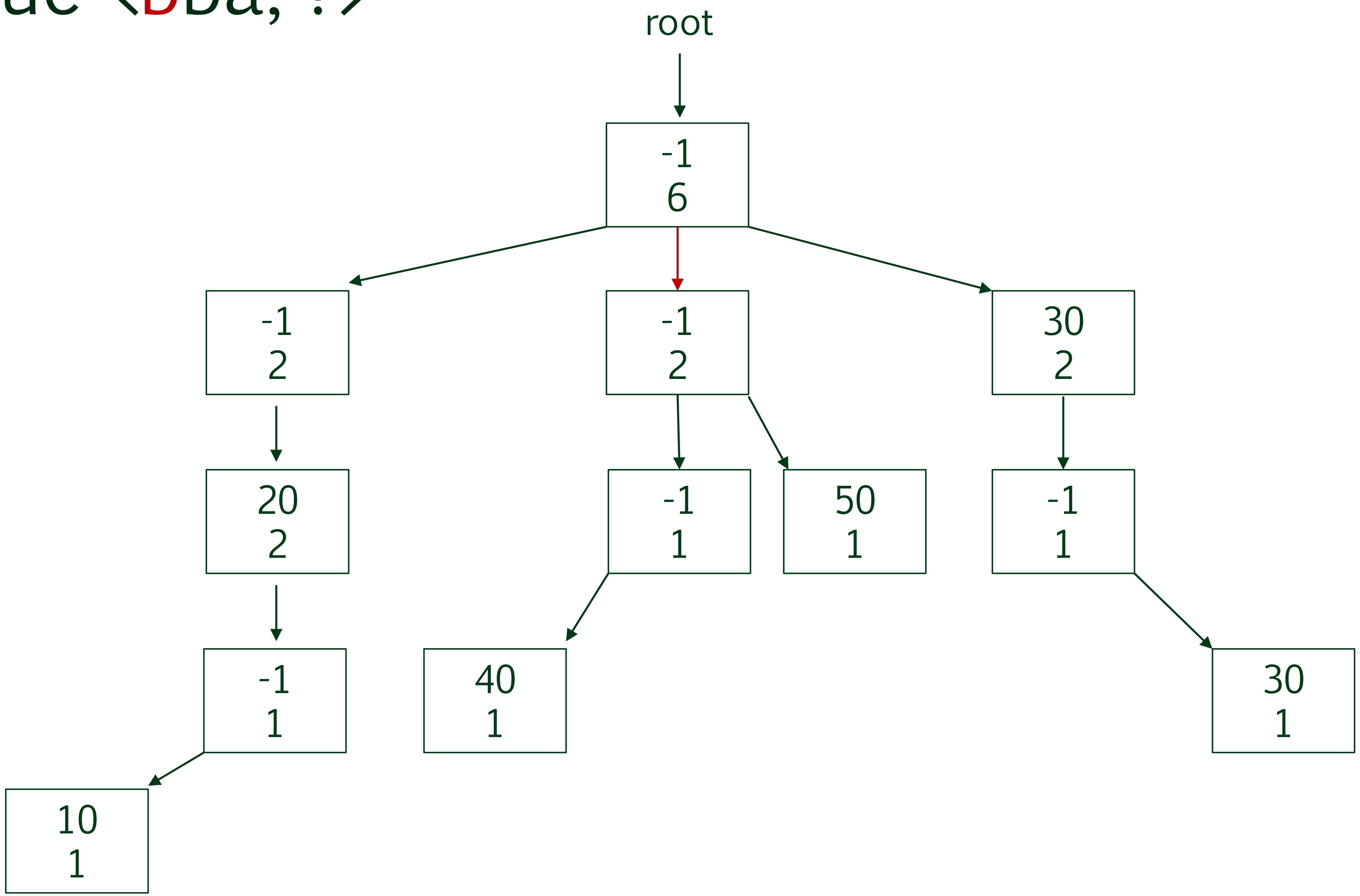
Value

- Basic strategy
  - Follow the given key from the root until:
    - A null pointer is reached, and a default value is returned
      How can a null pointer be reached?
    - The key is fully examined, and the value at the final node is returned
      Can a default value be returned?

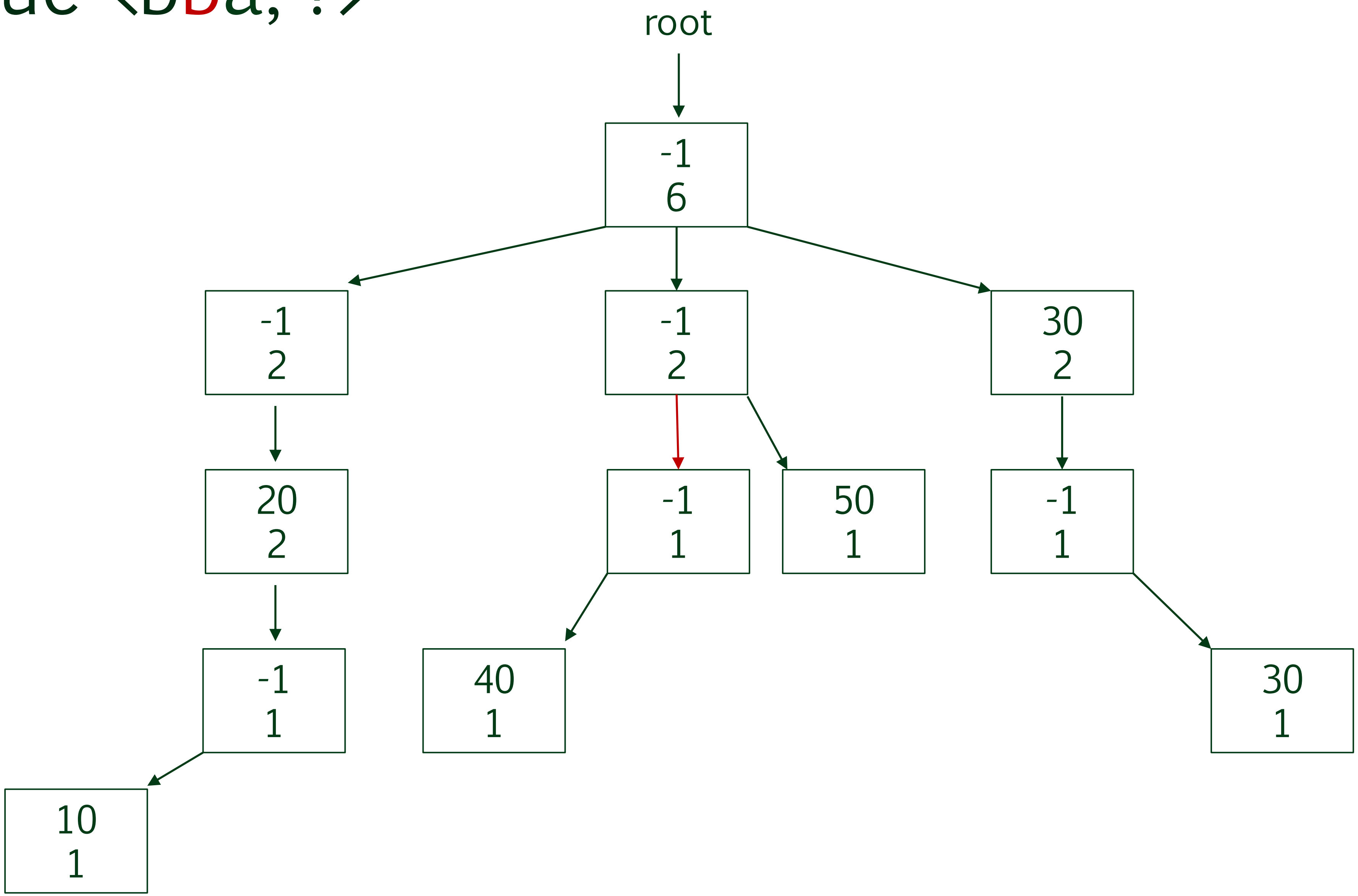
Value <bba, ?>



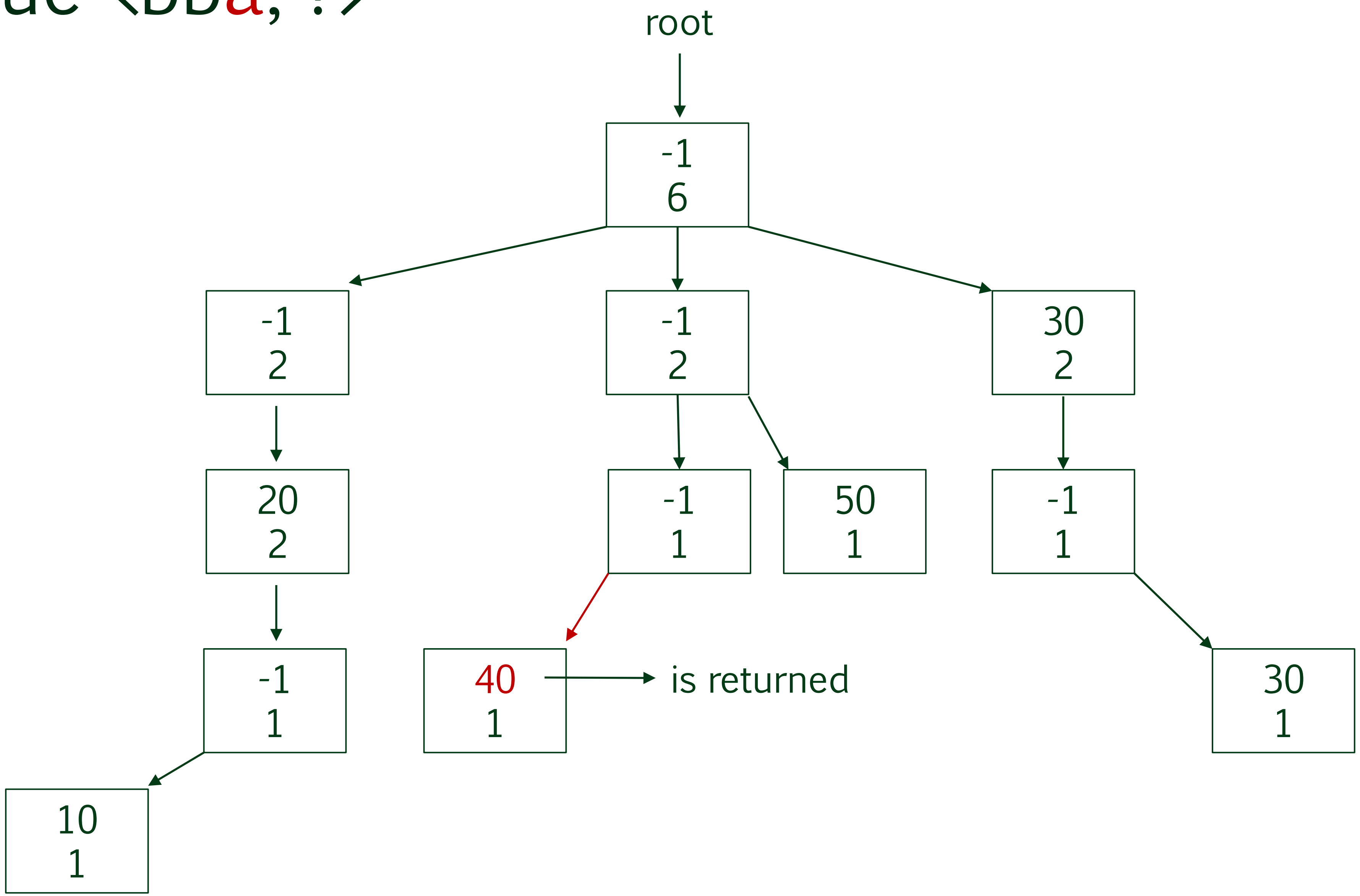
Value <bba, ?>



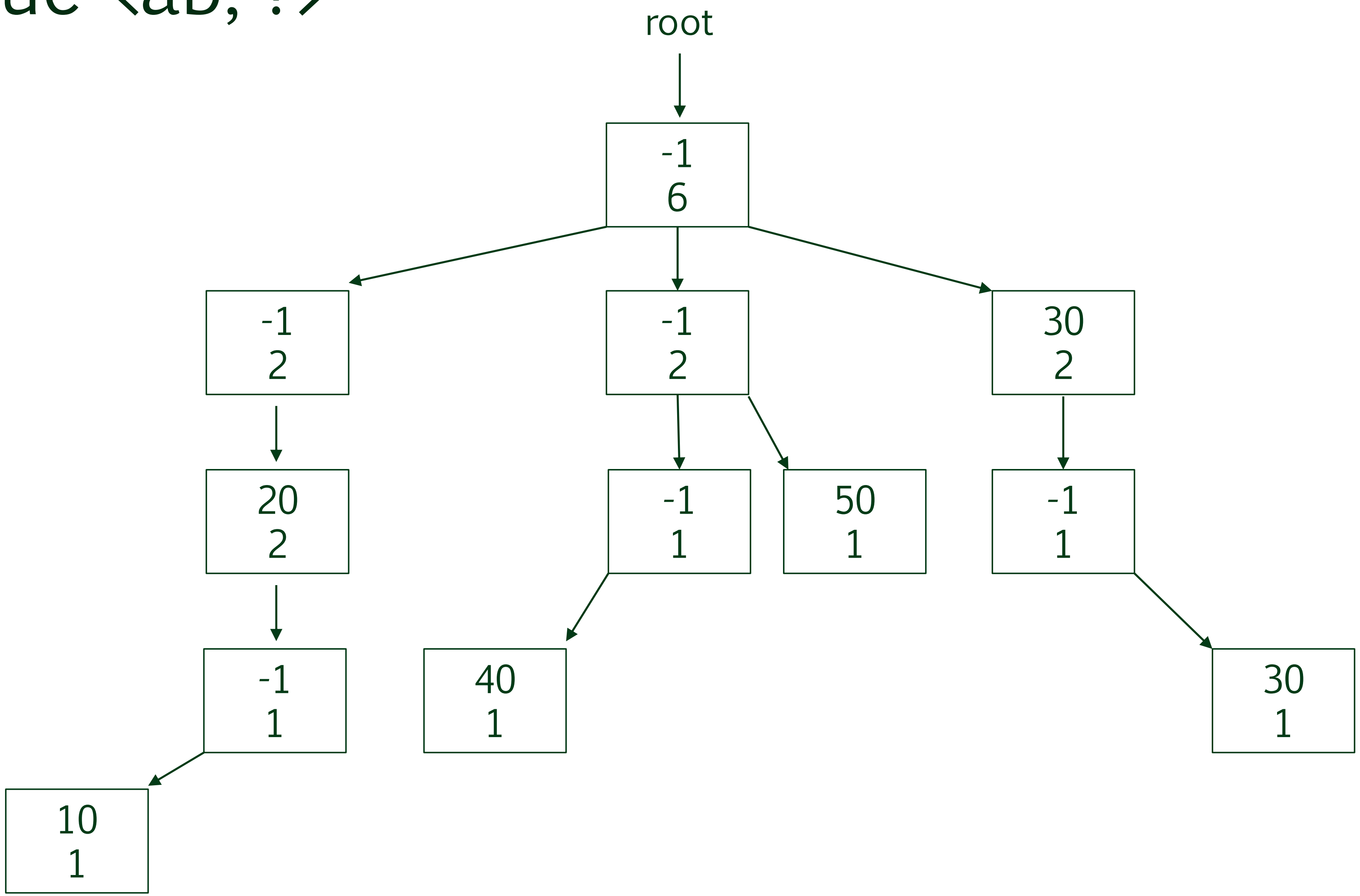
Value <bba, ?>



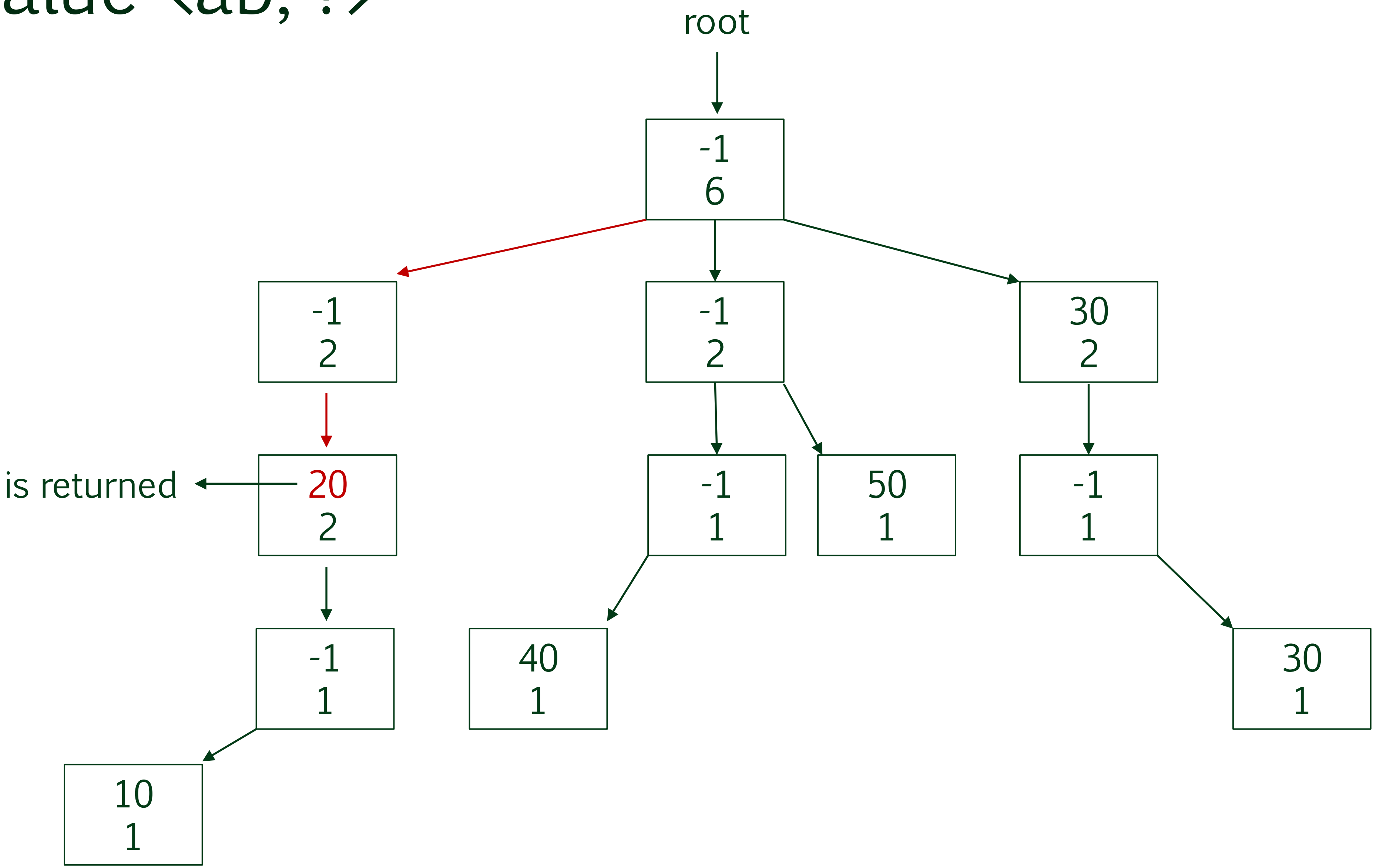
Value <bba, ?>



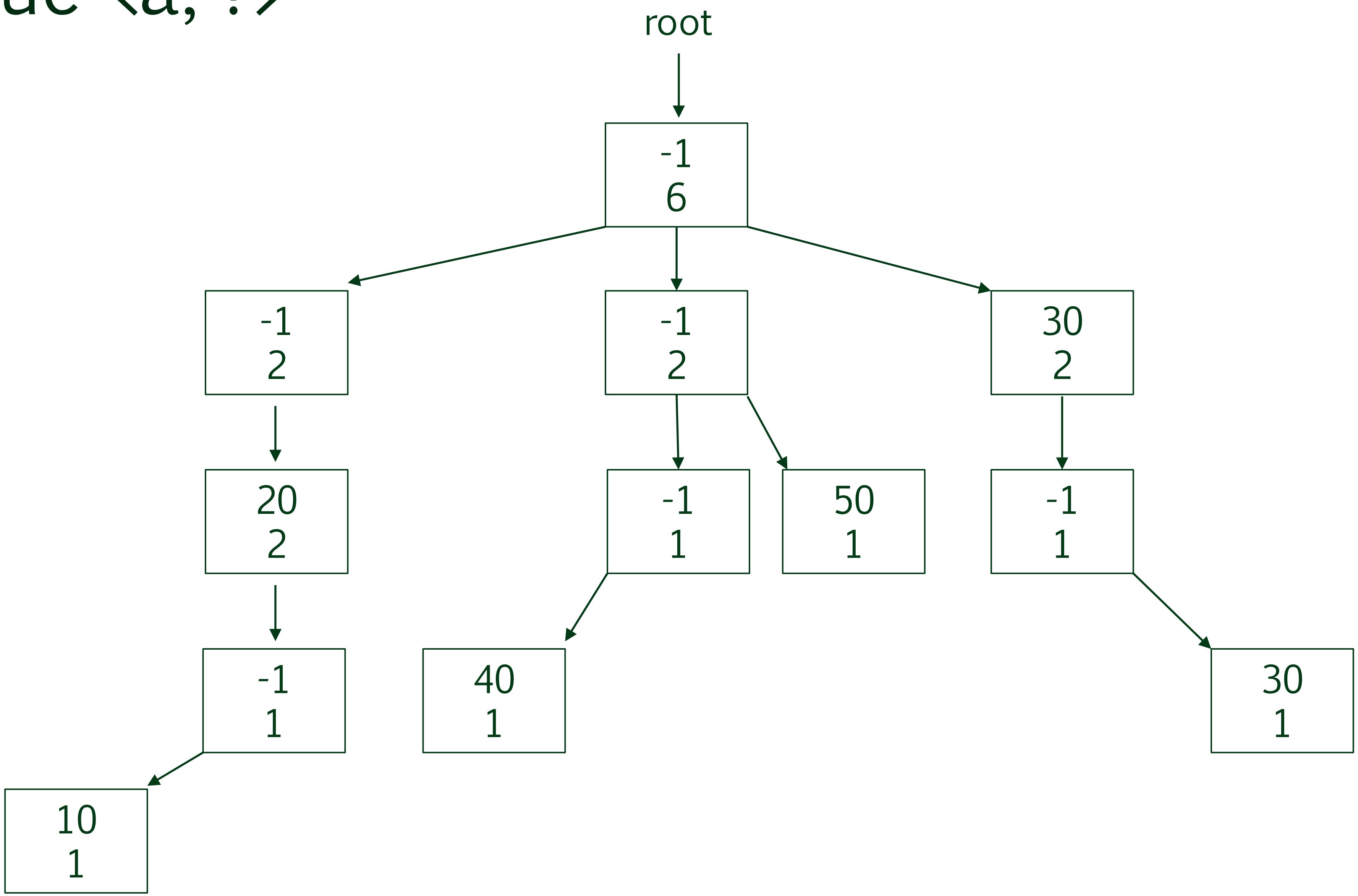
Value <ab, ?>



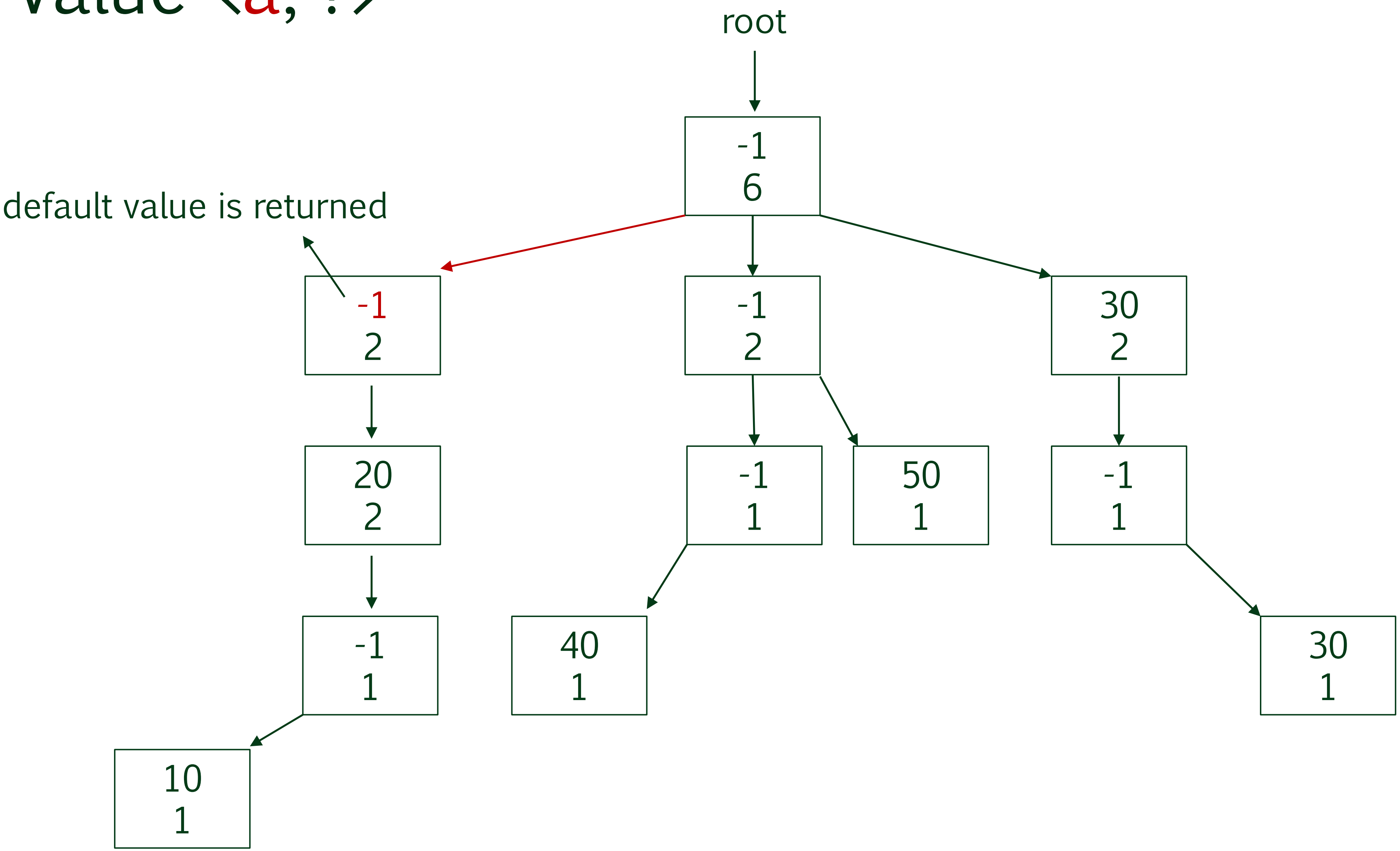
Value <ab, ?>



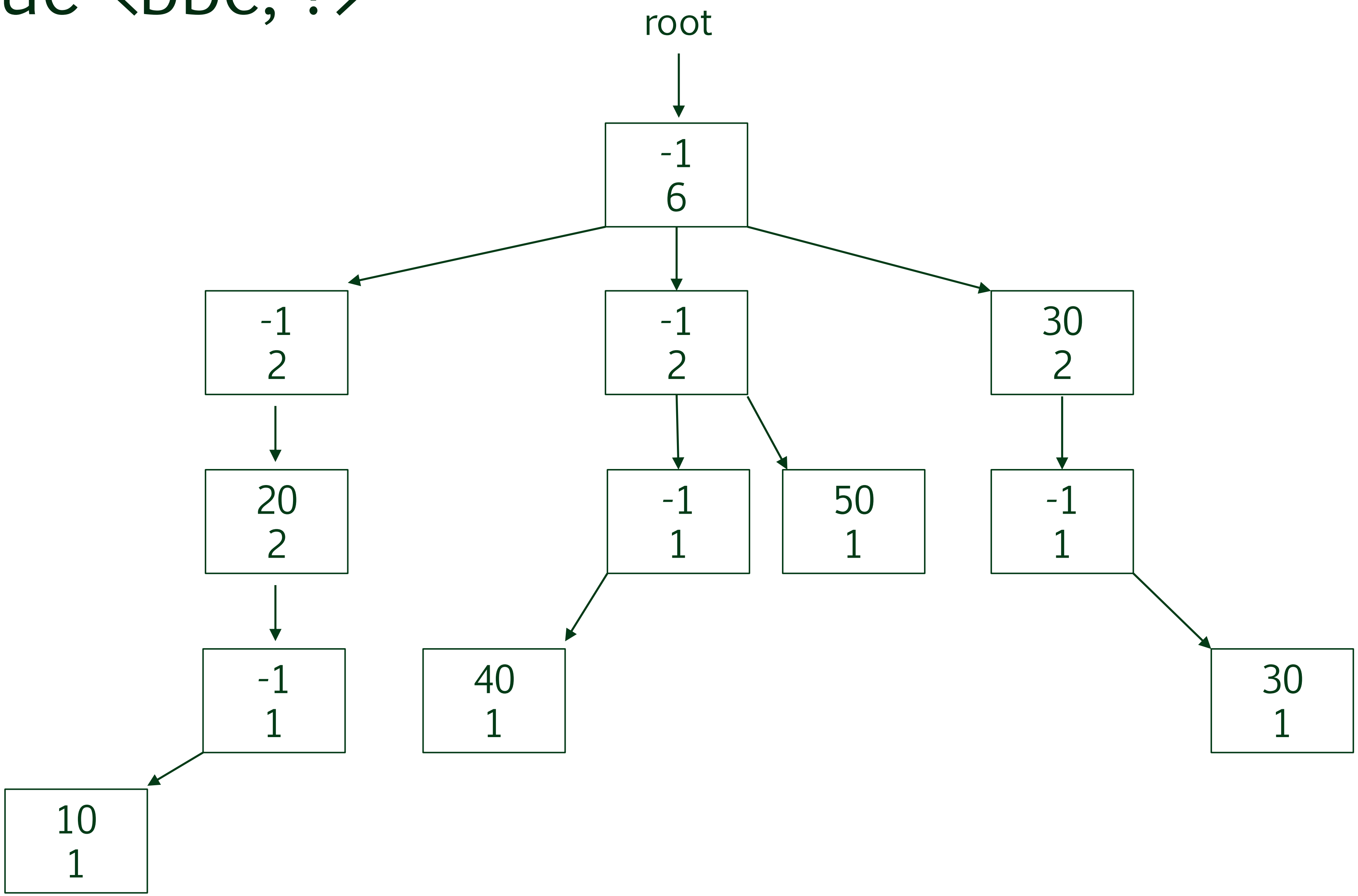
Value <a, ?>



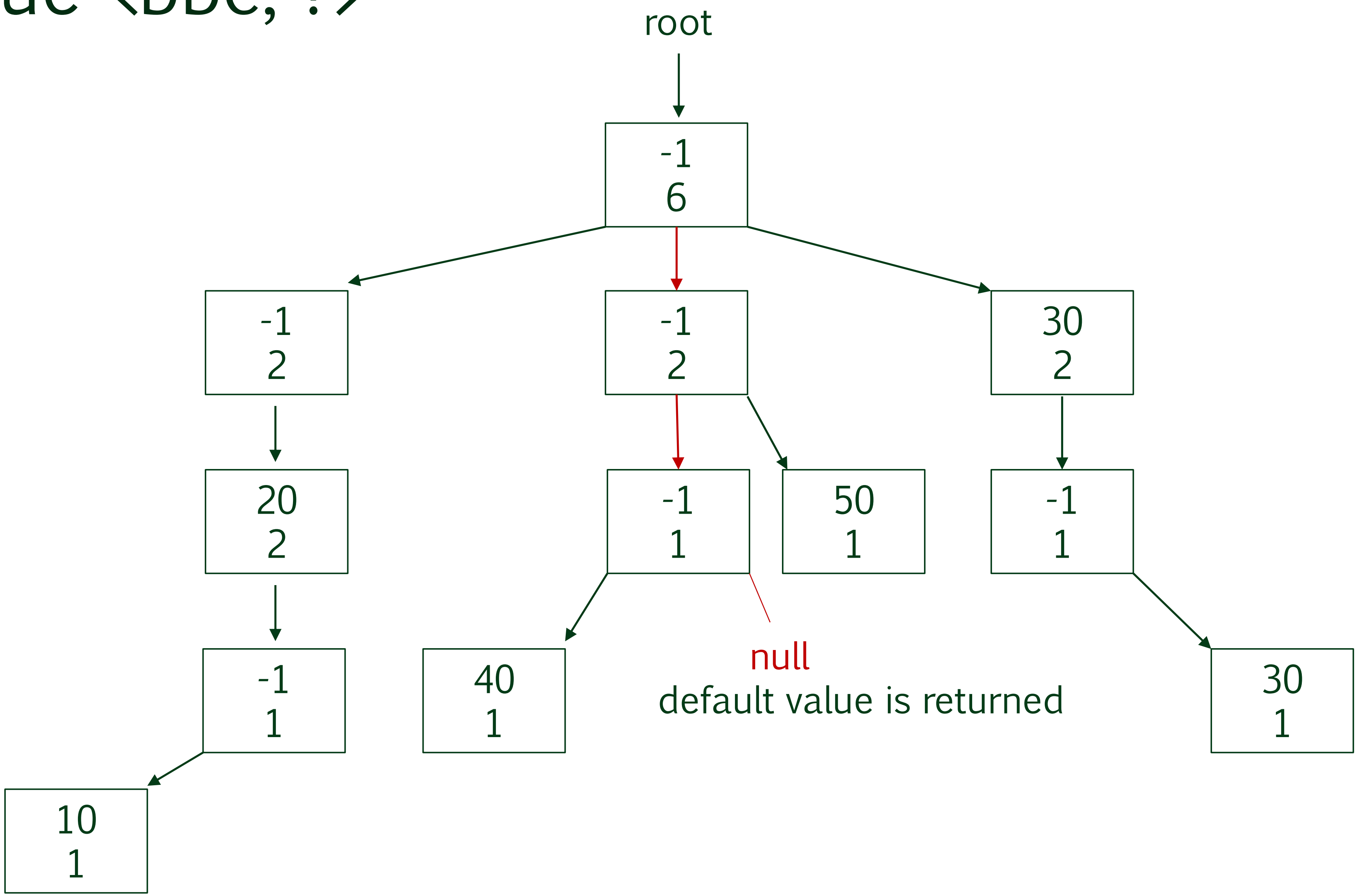
Value <a, ?>


root
-1
6
default value is returned
-1
2
-1
2
30
2
20
2
-1
1
50
1
-1
1
-1
1
40
1
30
1
10
1

Value <bbc, ?>



Value <bbc, ?>


root
-1
6
-1
2
-1
2
30
2
20
2
-1
1
50
1
-1
1
-1
1
40
1
null
default value is returned
30
1
10
1

Time complexity

- Let M be the maximum length of a key in the trie
- The time complexity of Value is O(min(L, M)) where L is the length of the given key

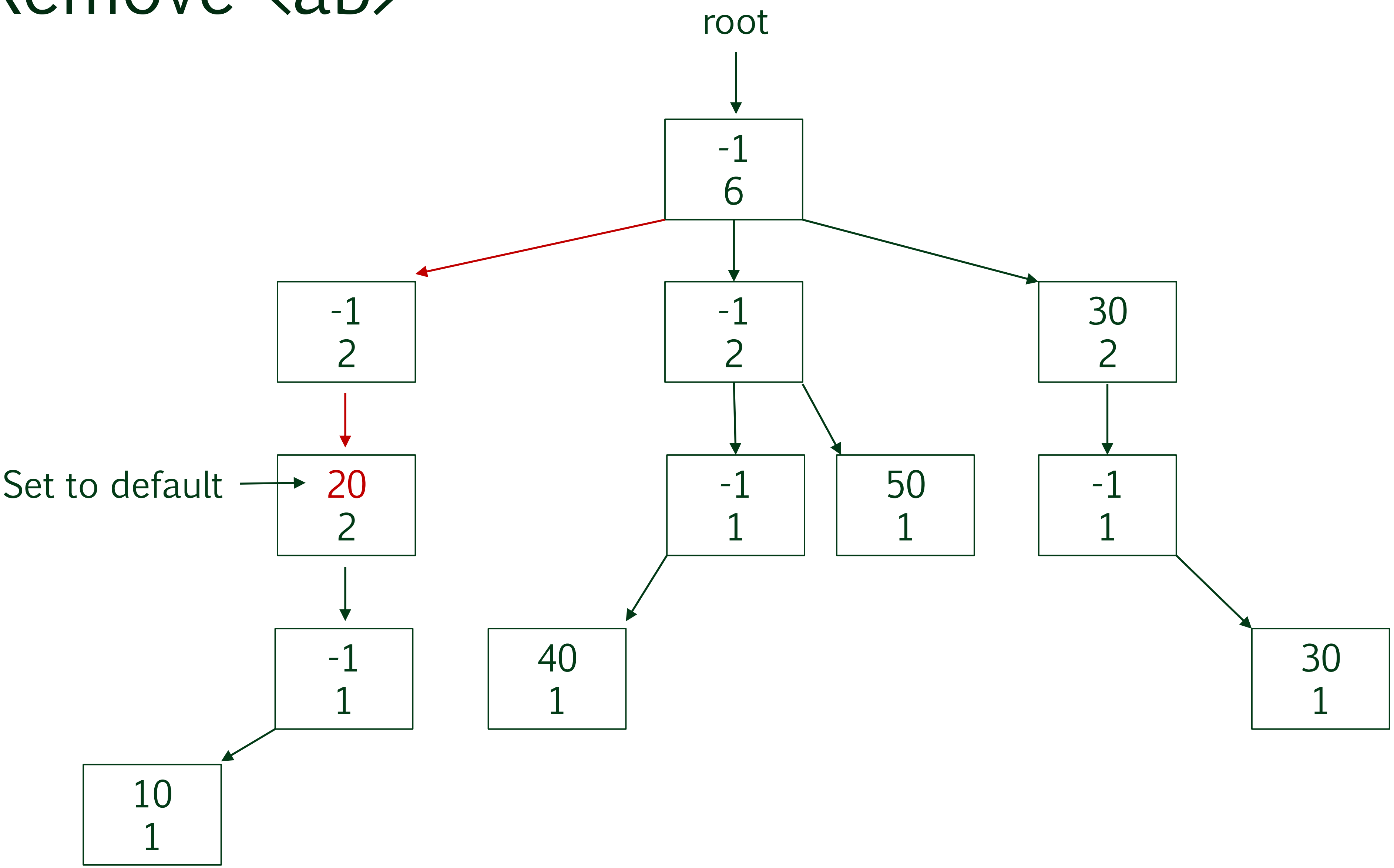
Exercises

- Give a key that will cause the Value method to execute in O(M) time on the final trie.
- According to the Value method posted online, what happens if the key is not made up of letters ‘a’ .. ’z’ ? How would you solve this problem (if there is one)?

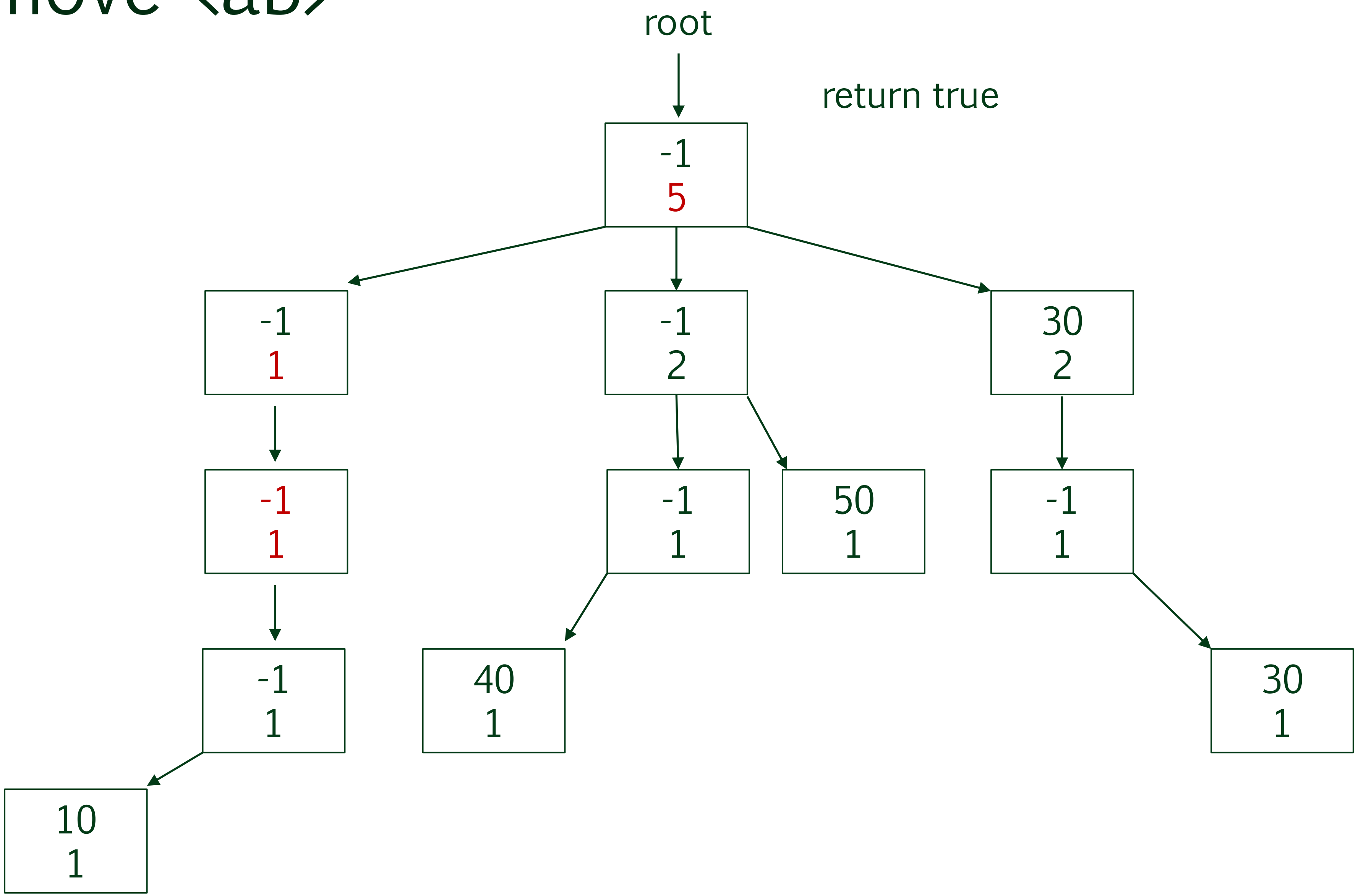
Remove

- Basic strategy
  - Follow the key to the node whose value is to be deleted.
  - If the node is null or contains a default value then false is returned; otherwise, the value at the node is set to default, true is returned, and numValues is reduced by one for each node along the path back to the root.
  - For each child node whose numValues is reduced to 0 on the way back, the link to the child node is set to null (cf Rope compression).

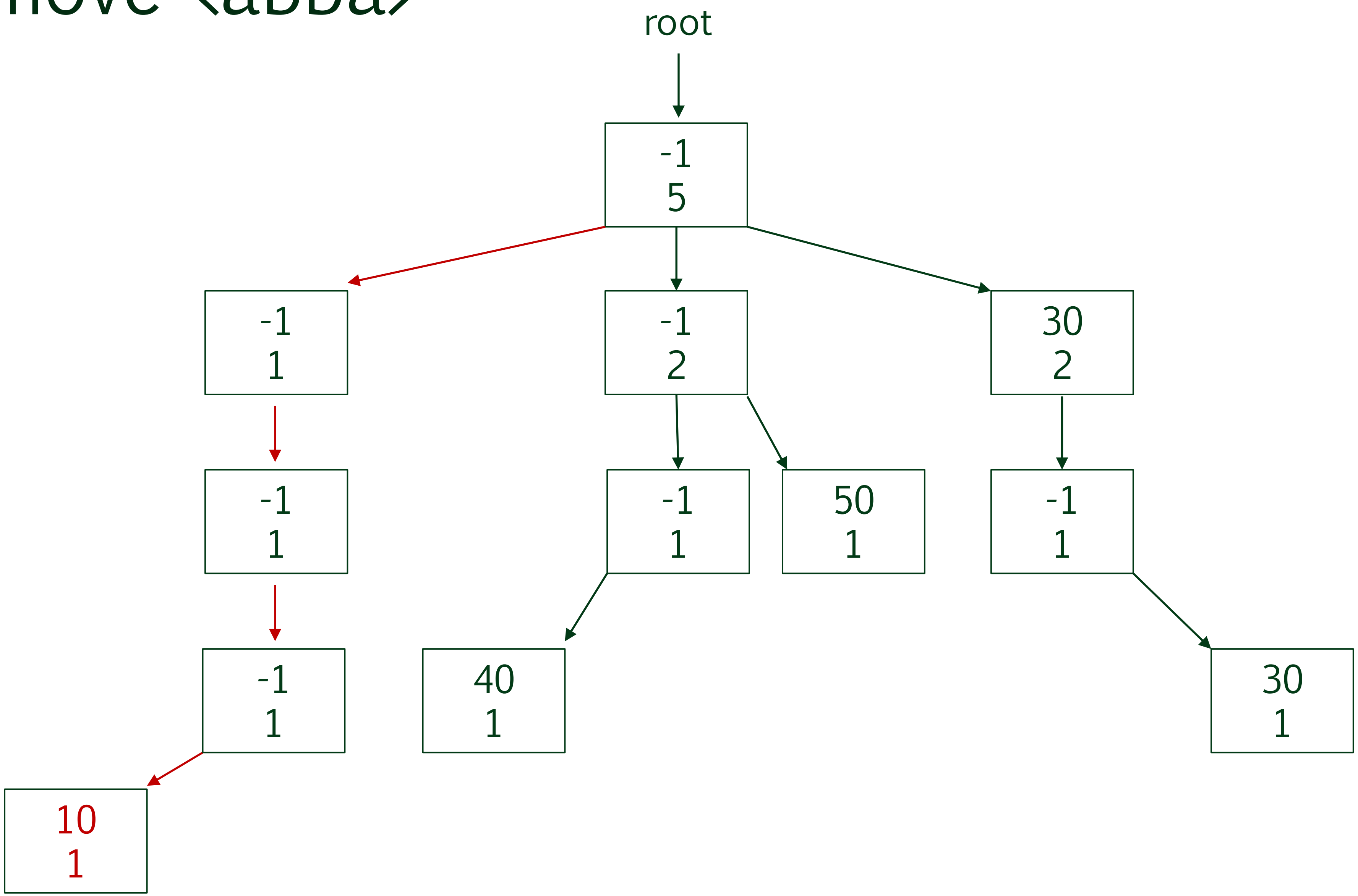
Remove <ab>



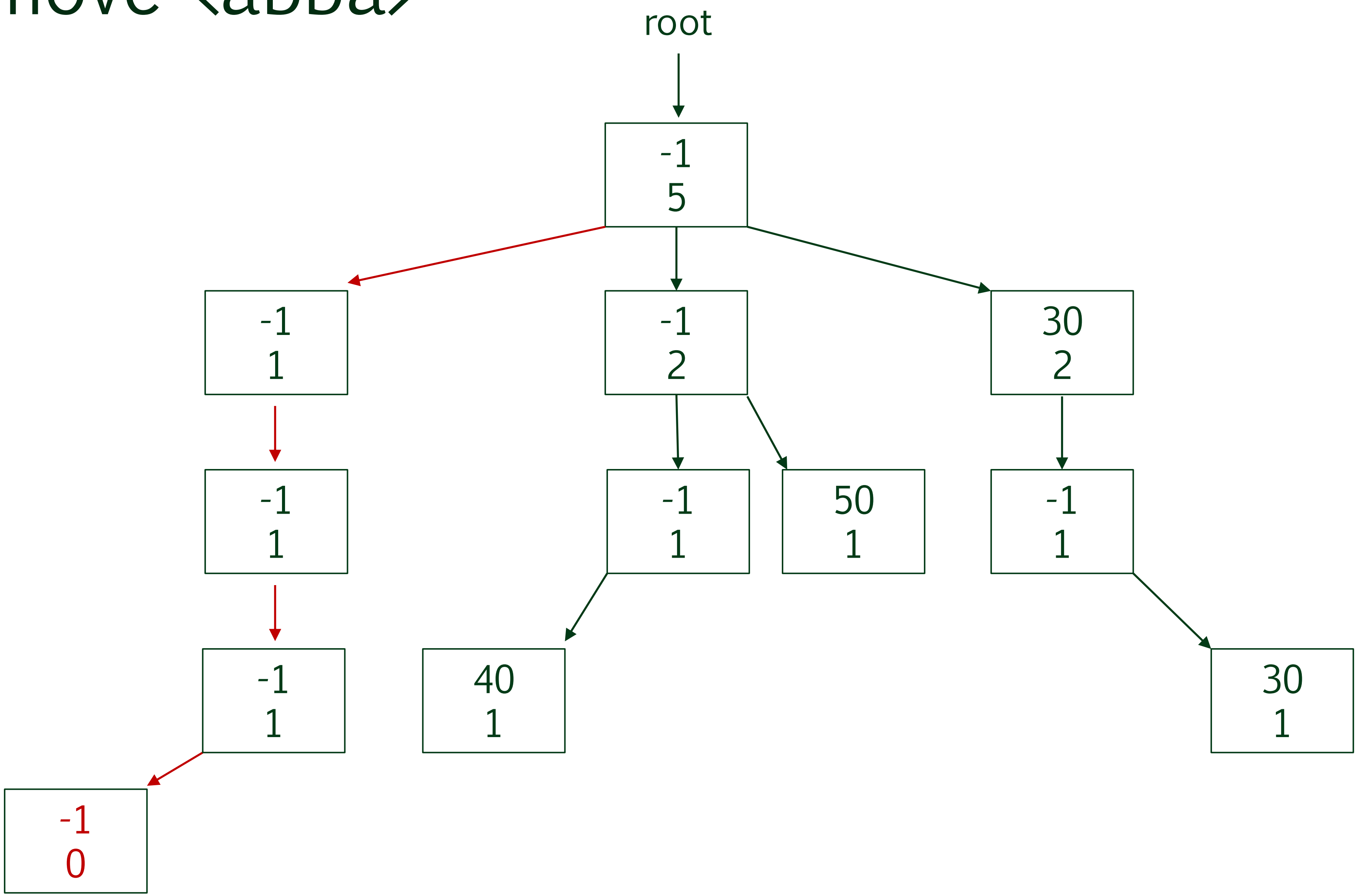
Remove <ab>



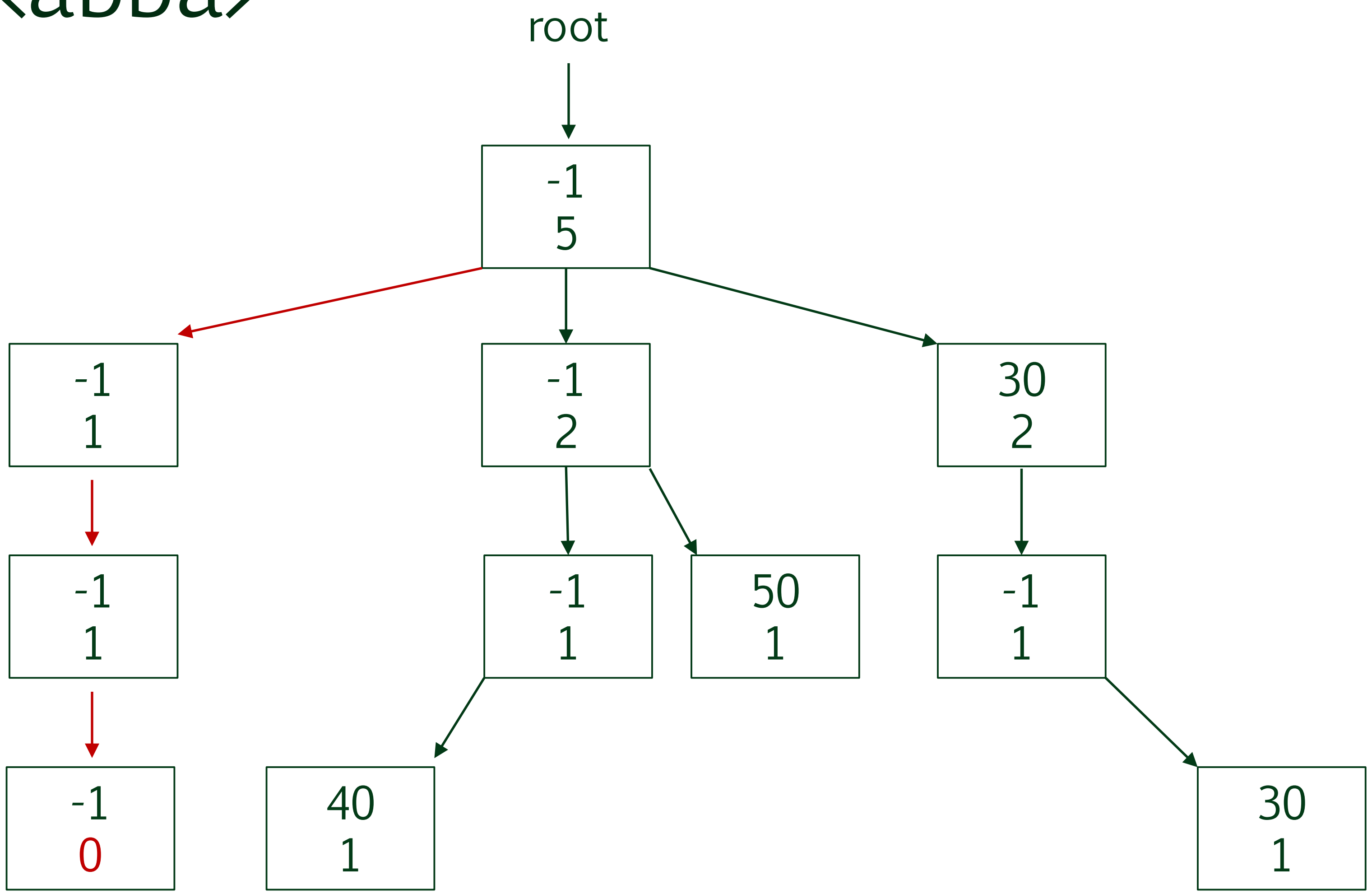
Remove <abba>



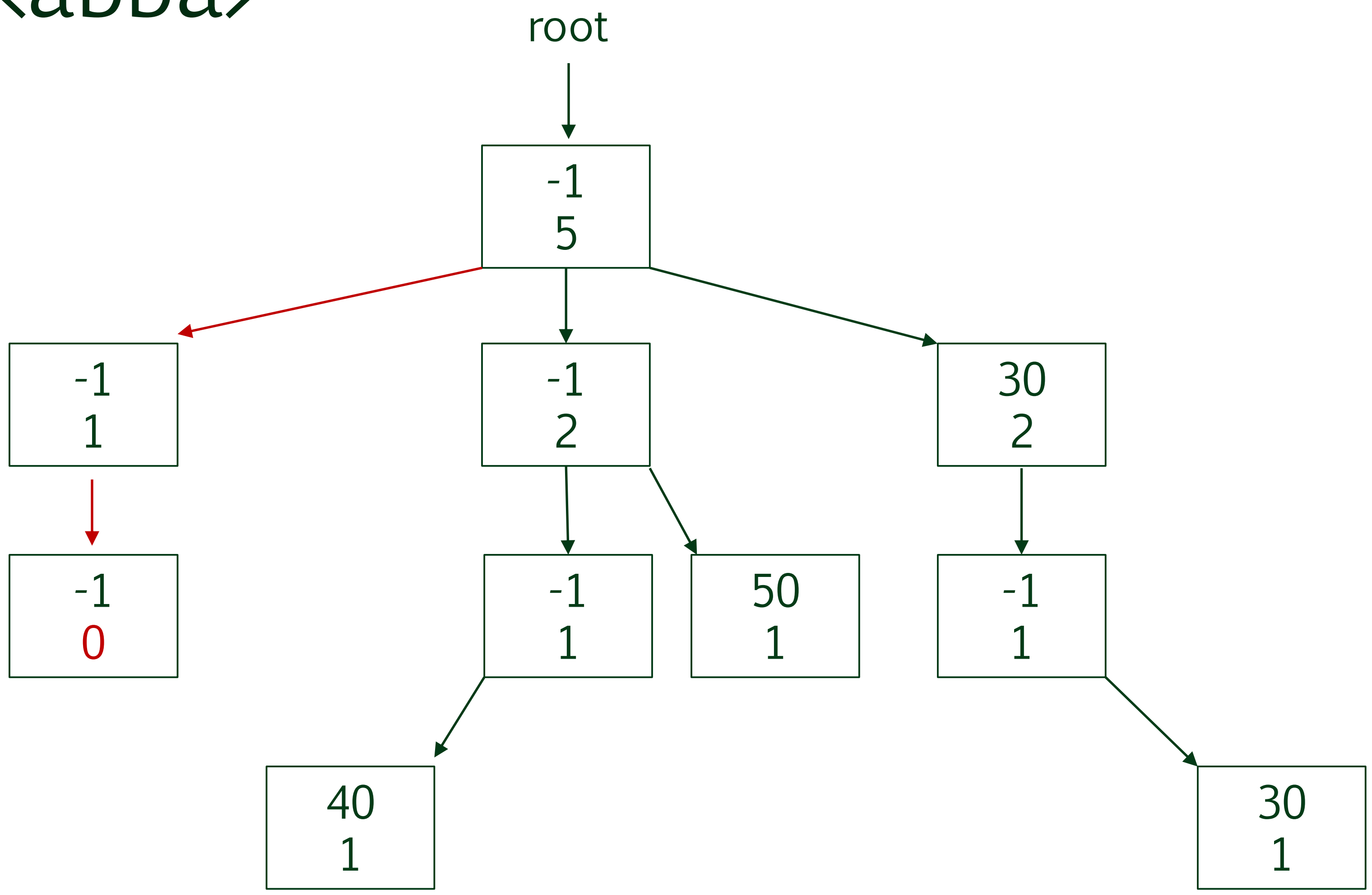
Remove <abba>



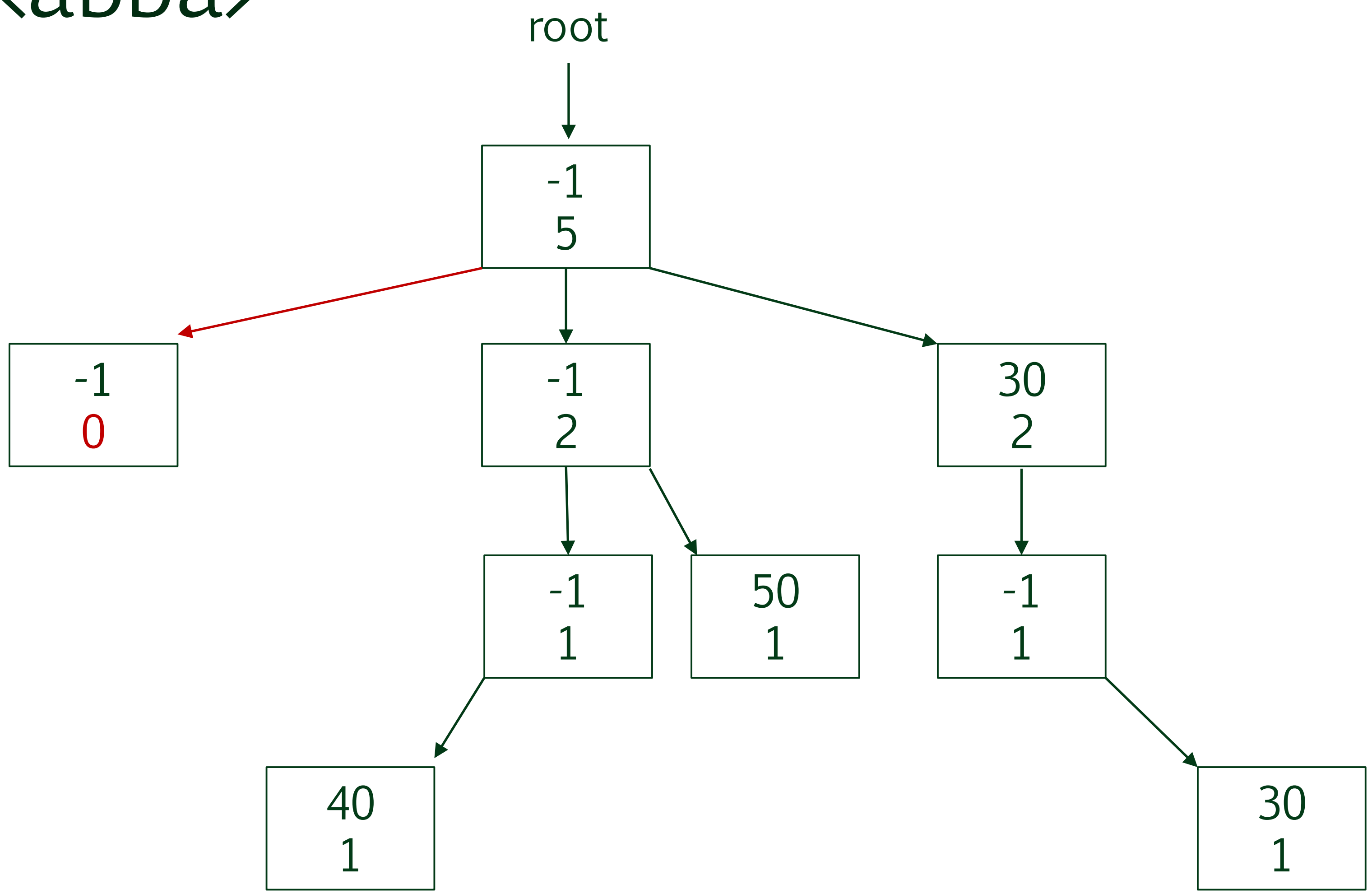
Remove <abba>



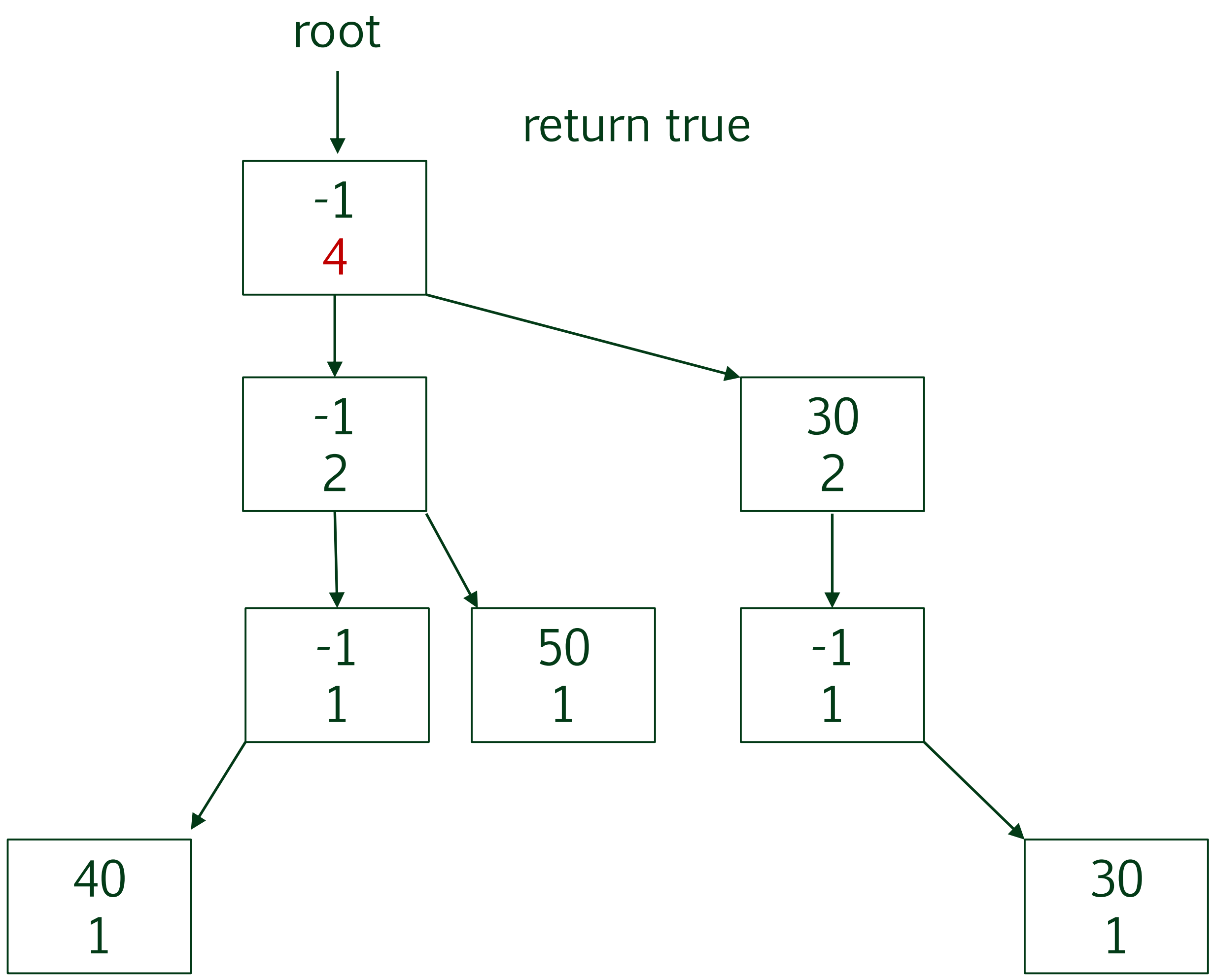
Remove <abba>



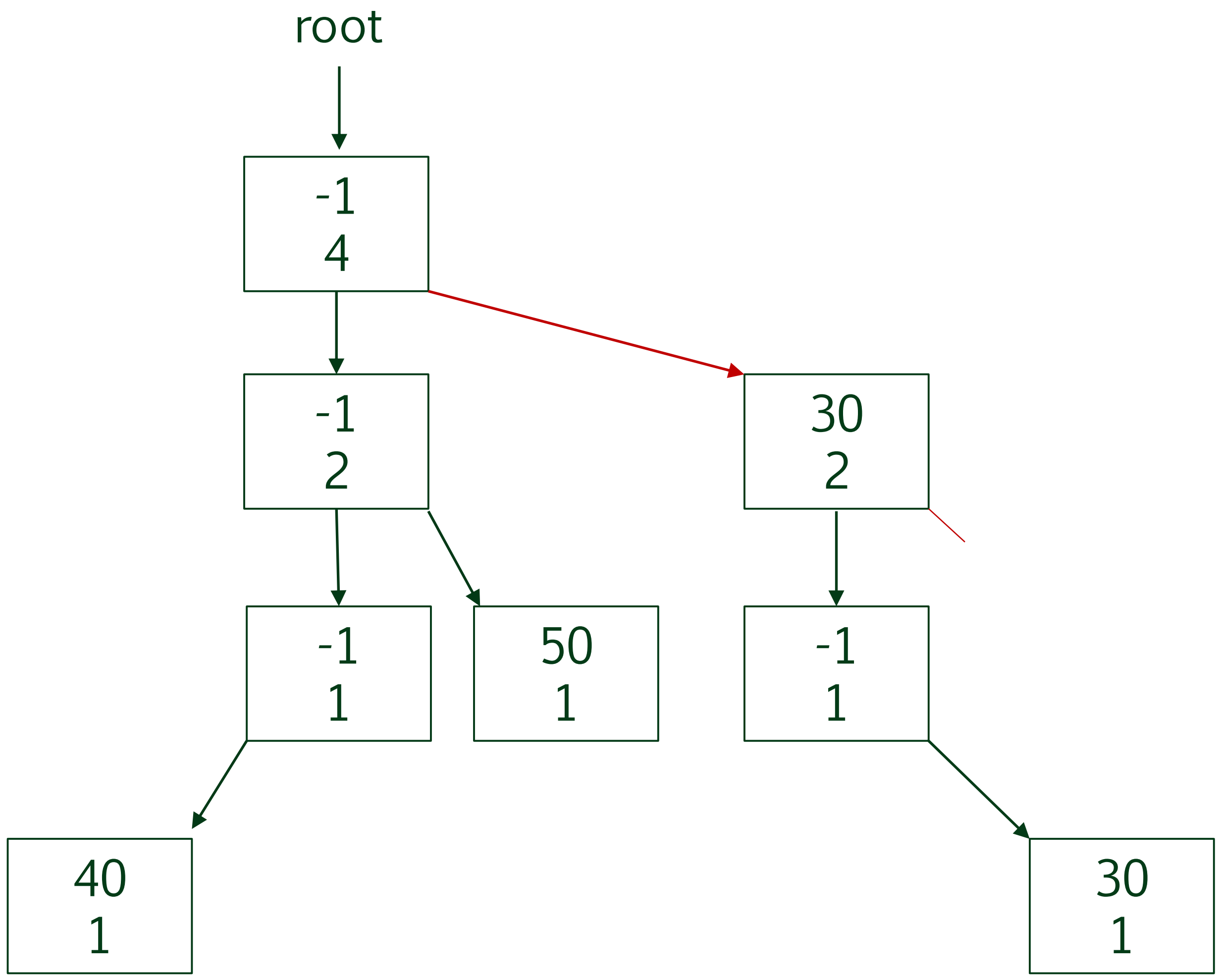
Remove <abba>



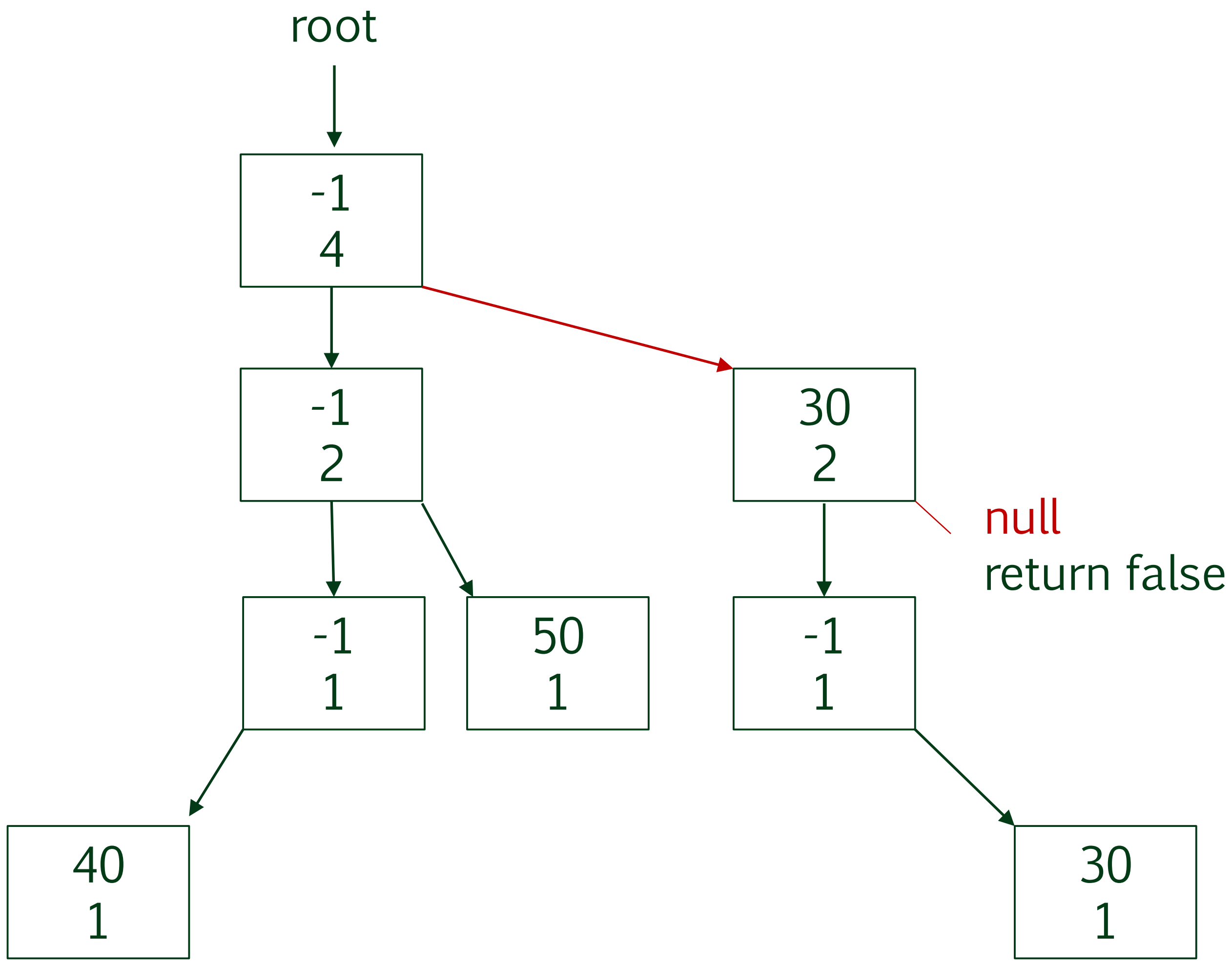
Remove <abba>



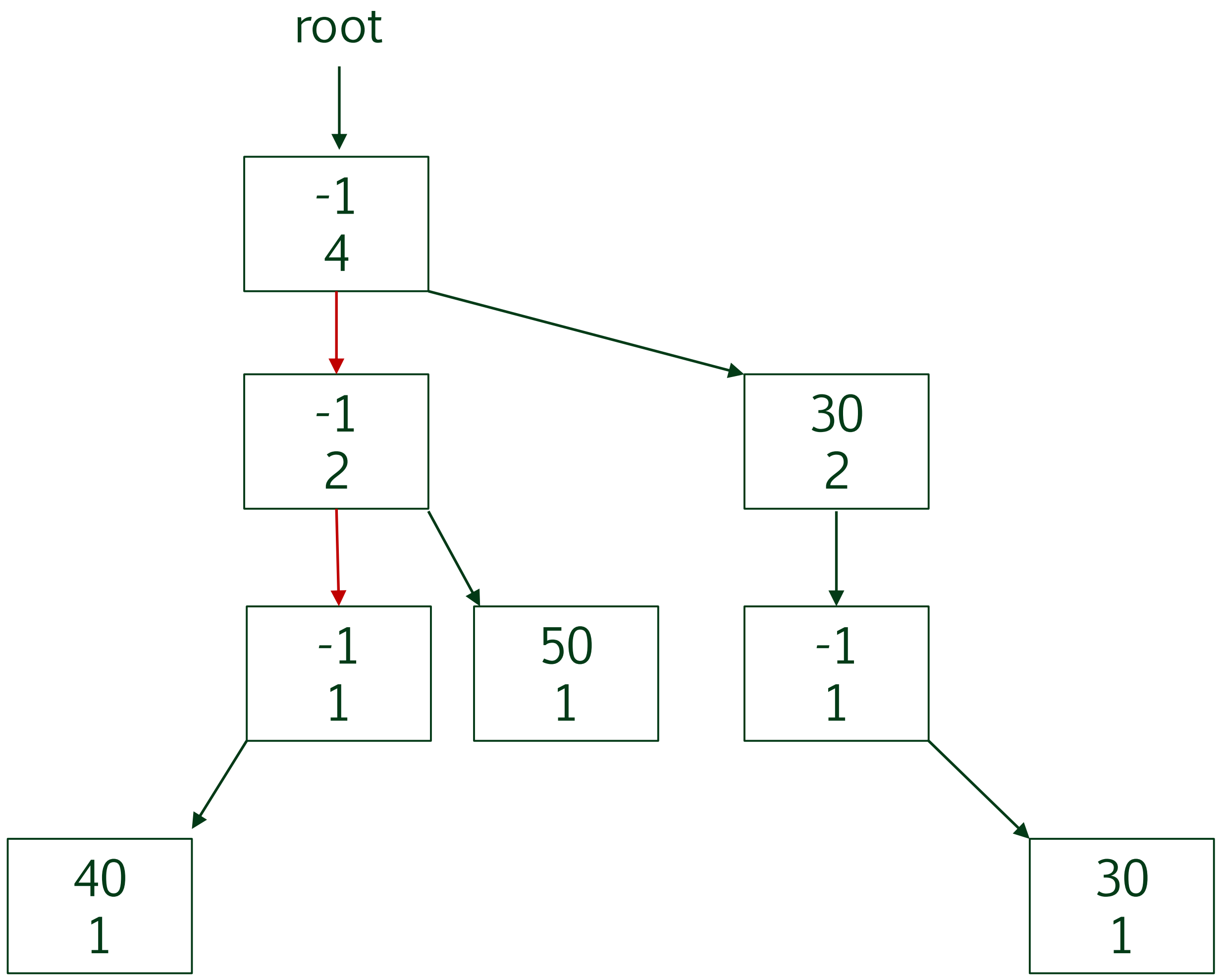
Remove <cc>



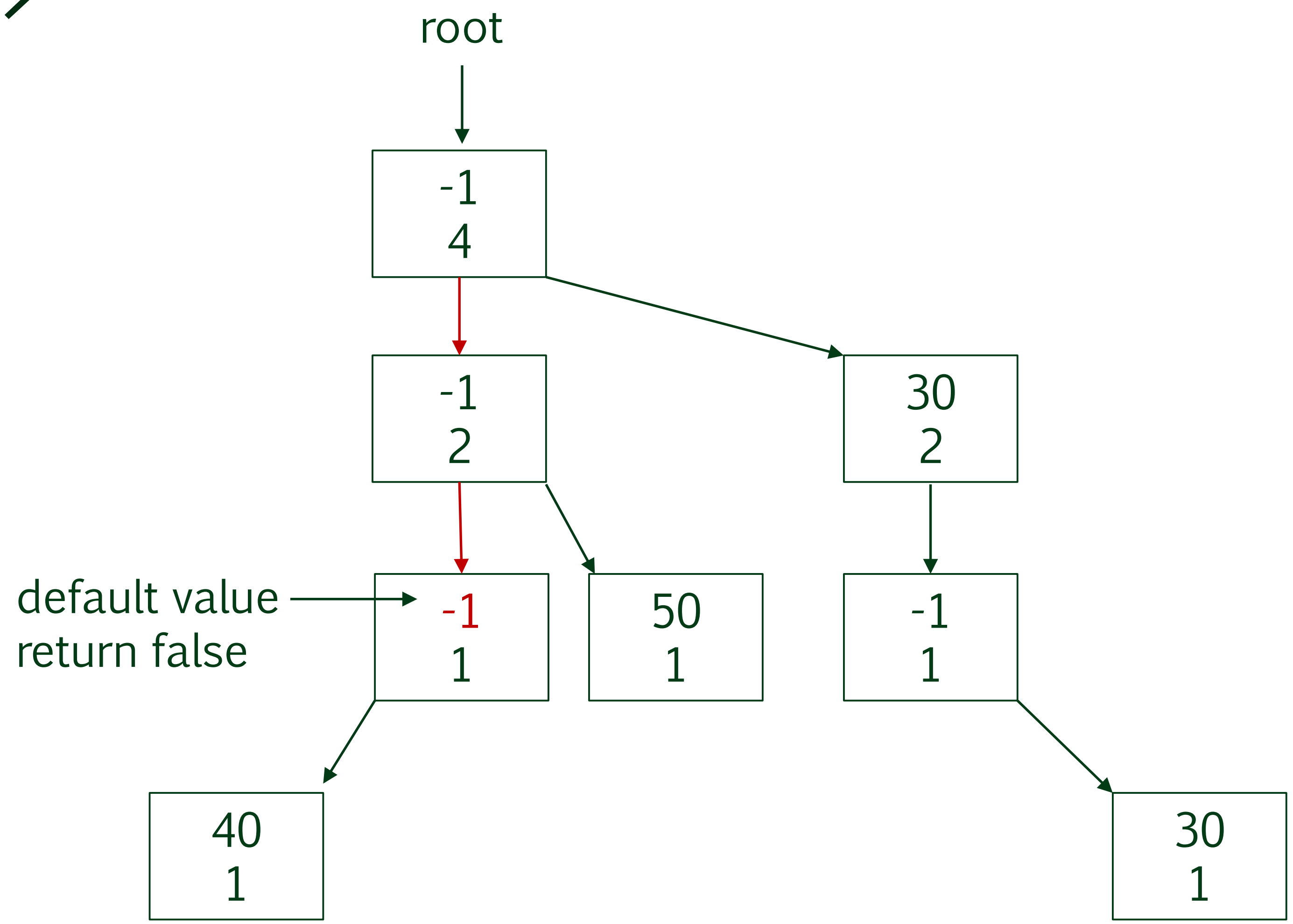
Remove <cc>



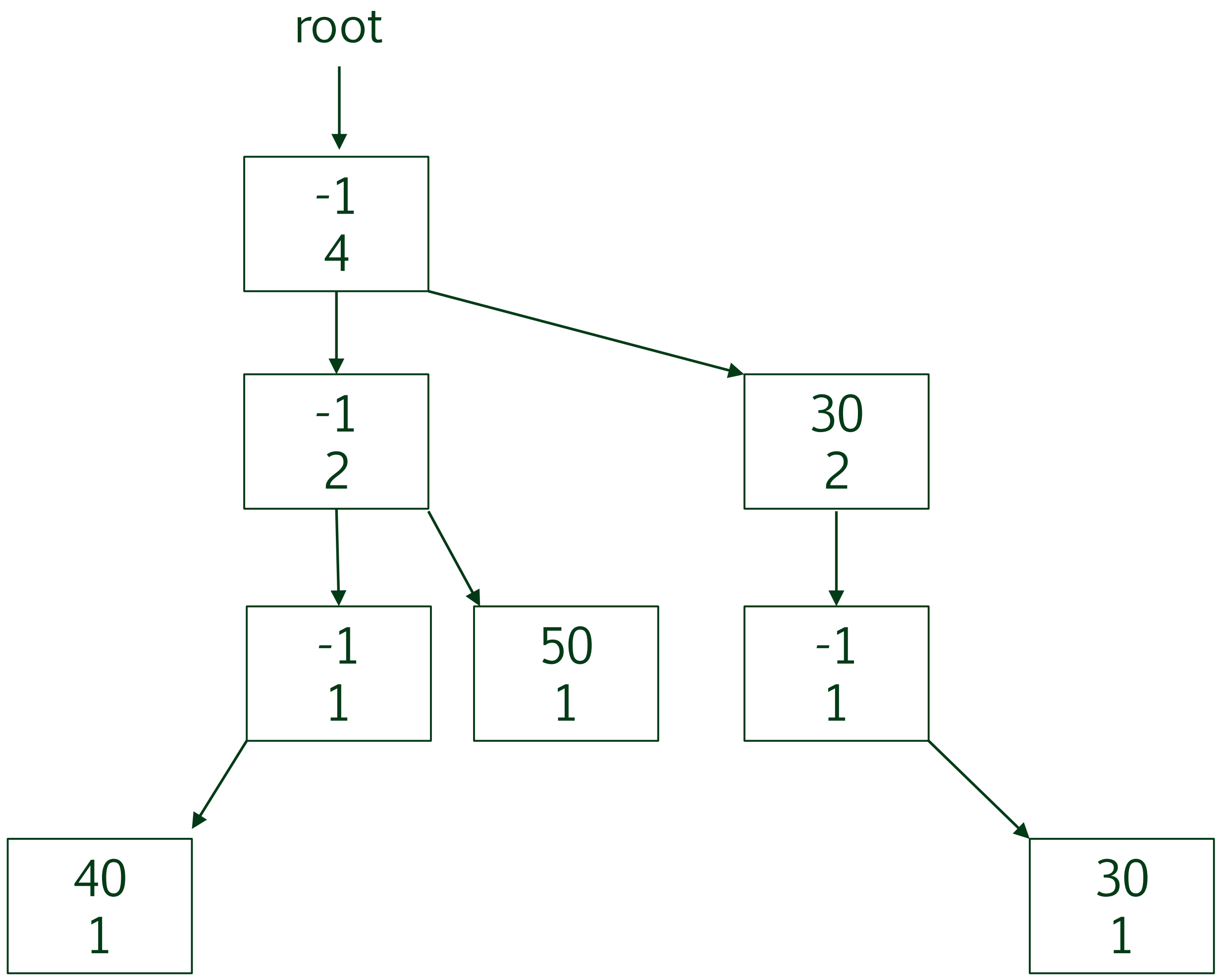
Remove <bb>



Remove <bb>



Final Trie



Time complexity

- Like the method Value, the time complexity of Remove is O(min(L,M)) where L is the length of the given key and M is the maximum length of a key in the trie

Exercises

- Justify the time complexity of the method Remove.
- Continue to remove the remaining <key,value> pairs from the final trie until the empty trie remains.

Print

- Unlike the closed hash table, the r-way trie can easily print out keys in order

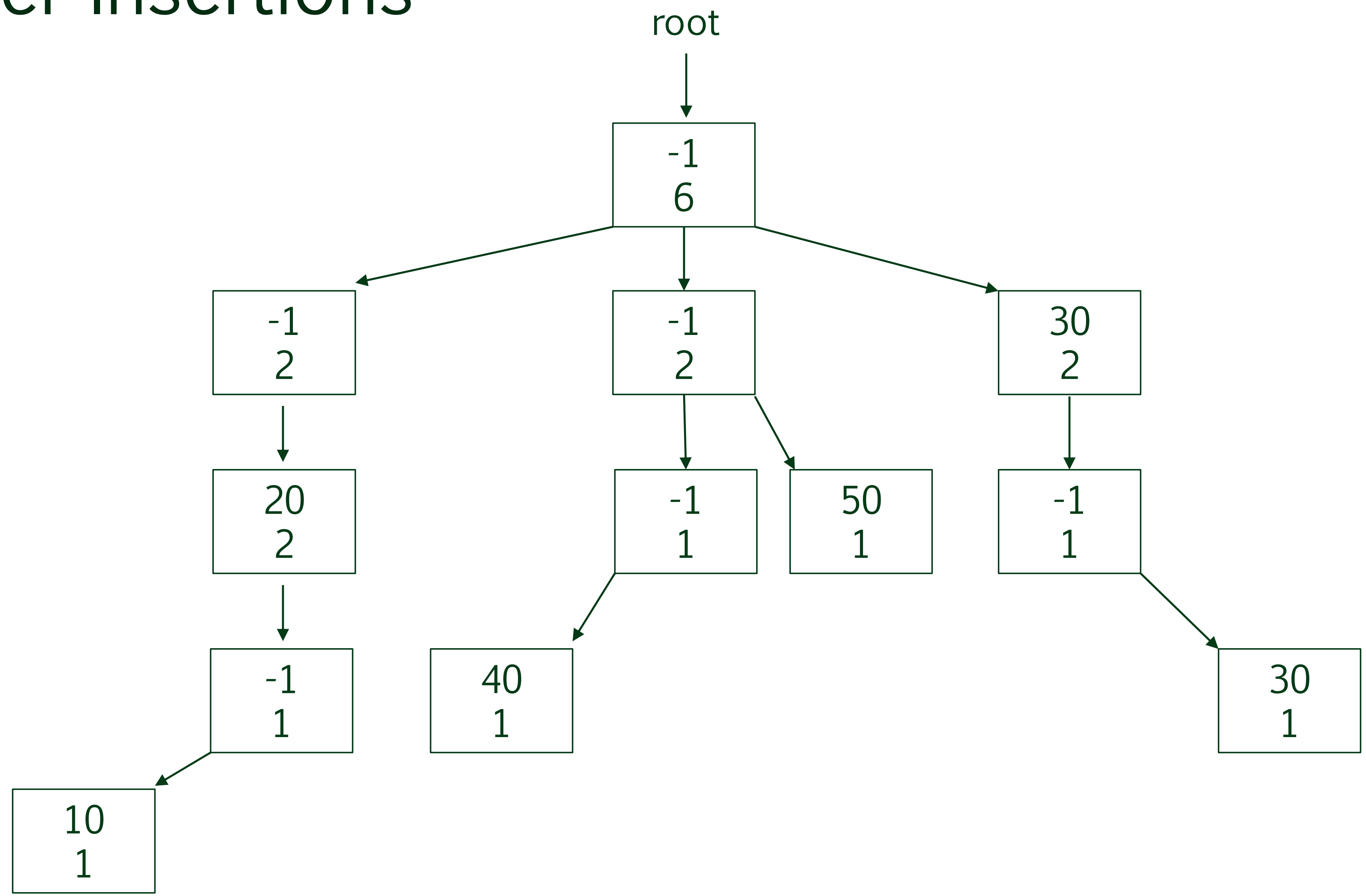
Implementation

```
private void Print(Node p, string key)
        {
            int i;

            if (p != null)
            {
                if (!p.value.Equals(default(T)))
                    Console.WriteLine(key + " " + p.value + " " + p.numValues);
                for (i = 0; i < 26; i++)
                    Print(p.child[i], key+(char)(i+'a'));
            }
        }
public void Print()
        {
            Print(root,"");
        }
```

Keys are constructed along the way

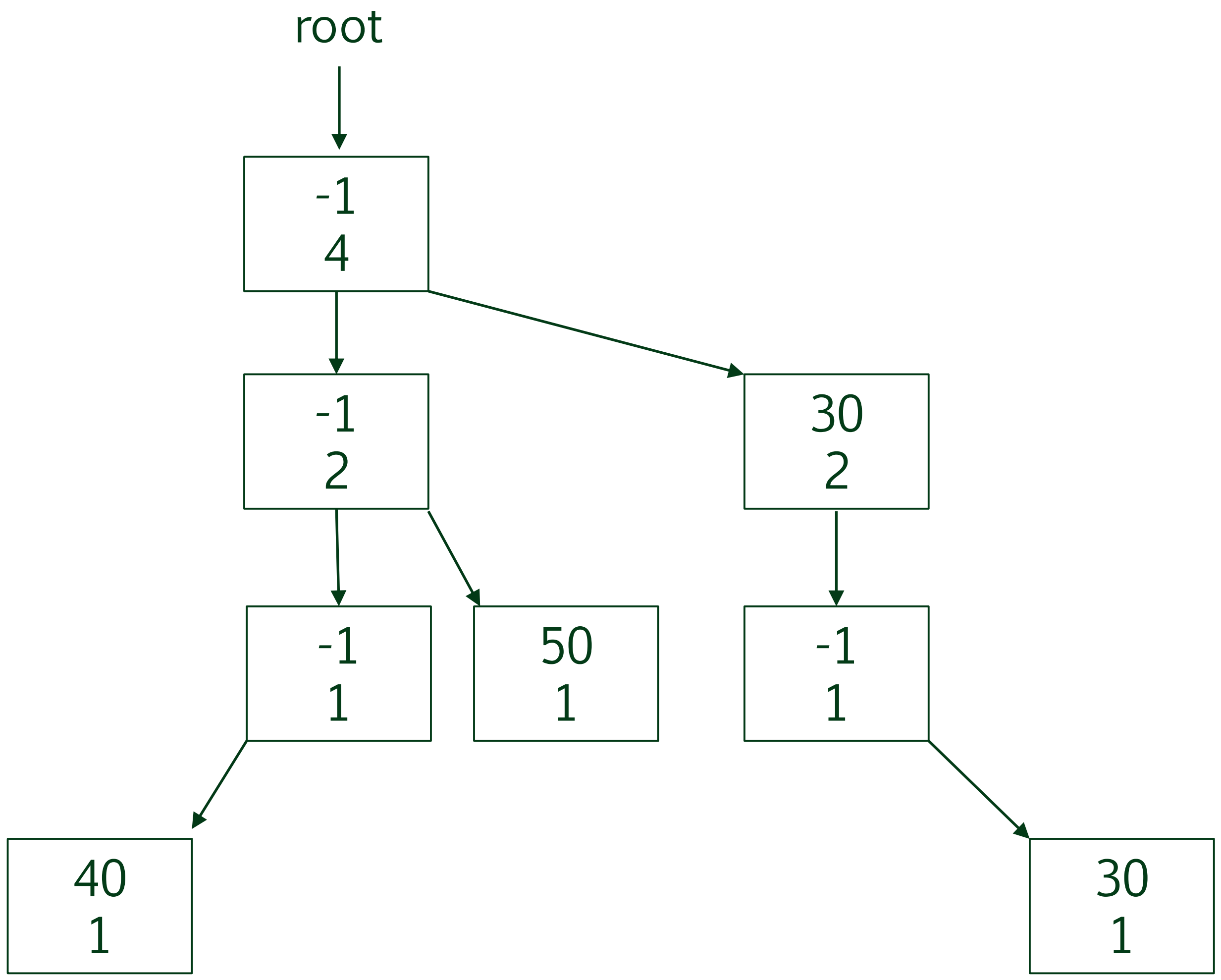
Final Trie After Insertions



Result <key, value, numValues>

ab	20	2
abba	10	1
bba	40	1
bc	50	1
c	30	2
cbc	30	1

Final Trie
After Removals



Result <key, value, numValues>

bba	40	1
bc	50	1
c	30	2
cbc	30	1

Next up …

the ternary tree