Data Structures

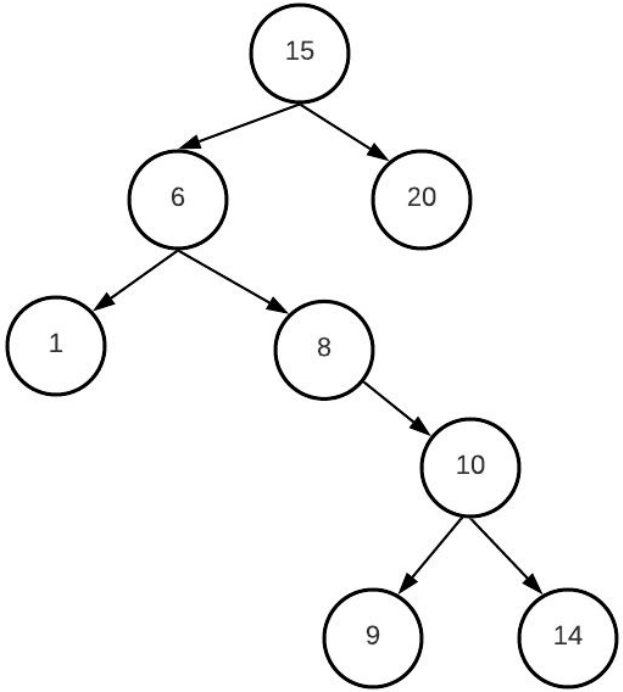
Minimum & Successor 2

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

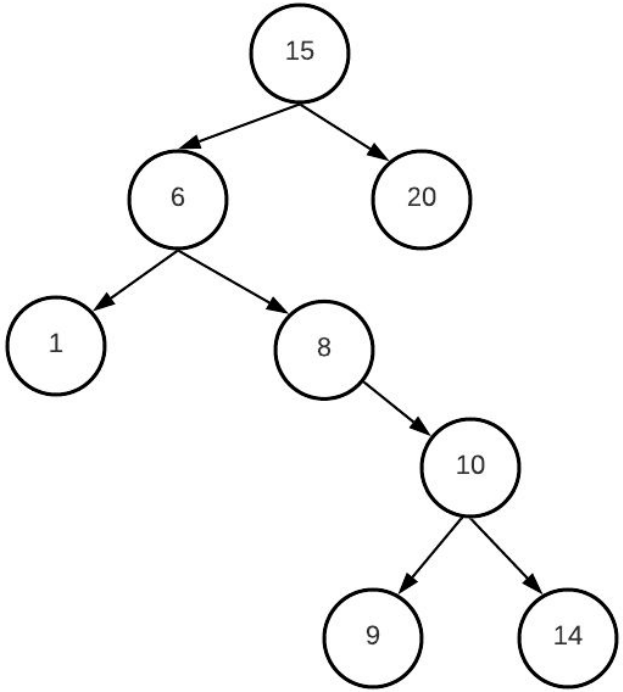
Why correct?

- We proved inorder traversal (LVR) is sorted!
- Given a value, its successor is one after it in traversal
- Actually the 2 cases are driven from inorder traversal behaviour :)
- Recall inorder traversal is
  - Process and print left
  - Print me
  - Process and print right
- Think for 10 minutes about correctness



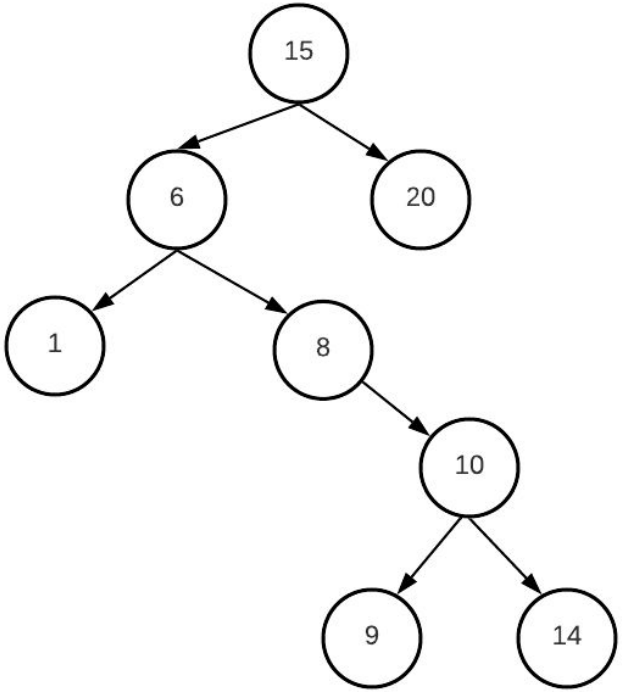
Why correct?

- Consider node (8)
- Case 1: we have a right subtree
- So far inorder printed: [1, 6, **8**]
  - Left till 8 is printed, then 8, then ?
- Now inorder will jump to my right to print it
  - Then my successor MUST be in my right
- Which one? Logically the minimum in them
  - Which also comes from inorder that keeps going left first tell no left, then print this min node



Why correct?

- Consider node (14)
- Case 2: no right subtree
- So far inorder printed: [1, 6, 8, 9, 10, **14**]
  - Left till 14 is printed, then 14, then ?
- As no right subtree, recursion ends and goes up
- It actually chain of rights: [6 ⇒ 8 ⇒ 10 ⇒ 14]
  - So recursion keeps going up and ends recursive calls
- Now 6 is done to, go up
- Left of 15 is done. Now print 15
  - So 15 first value printed after 14 ⇒ Successor



Implementation

- Now we need to keep going up in the tree!
- But we don’t have up parent
- 2 approaches
  - 1) Add parent pointer
    - You have to maintain it in insertion & deletion for nodes
  - 2) Get the ancestors nodes from root to target

```
 9 class BinarySearchTree {
10 private:
11     int data { };
12     BinarySearchTree* left { };
13     BinarySearchTree* right { };
14     BinarySearchTree* parent { };
15
```

Implementation

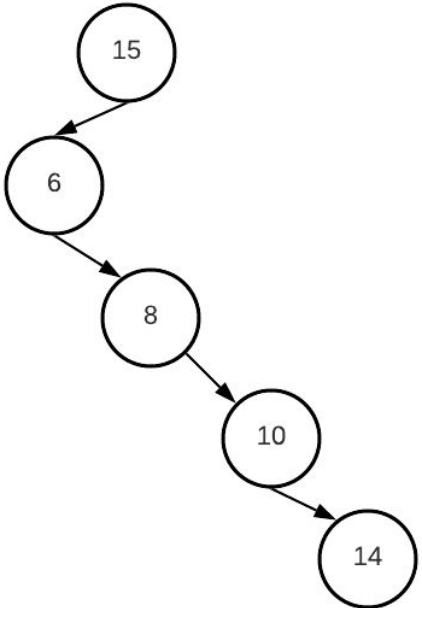
- We can modify the search function to get all nodes in its path to target
  - find_chain(9) = 15, 6, 8, 10, 9 [we include target too]
  - find_chain(14) = 15, 6, 8, 10, 14

```
// Modified search: Return chain of ancestors from node to target
bool find_chain(vector<BinarySearchTree*> &ancestors, int target) {
    ancestors.push_back(this);

    if (target == data)
        return true;

    if (target < data)
        return left && left->find_chain(ancestors, target);

    return right && right->find_chain(ancestors, target);
}
```

Implementation

- Simple utility: Extract next parent (from back) or null if nothing

```
BinarySearchTree* get_next(vector<BinarySearchTree*> &ancestors) {
    if (ancestors.size() == 0)
        return nullptr;
    BinarySearchTree* node = ancestors.back();  // last element
    ancestors.pop_back();
    return node;
}
```




```
pair<bool, int> successor(int target) {
    vector<BinarySearchTree*> ancestors;

    if (!find_chain(ancestors, target)) // not exist
        return make_pair(false, -1);

    BinarySearchTree* child = get_next(ancestors);

    if (child->right)   // must have be in min of right
        return make_pair(true, child->right->min_value());

    BinarySearchTree* parent = get_next(ancestors);

    // Cancel chain of ancestors I am BIGGER than them
    while (parent && parent->right == child)
        child = parent, parent = get_next(ancestors);

    if (parent) //
        return make_pair(true, parent->data);
    return make_pair(false, -1);
}
```

Max and Predecessor

- To get the Max node, we just keep going right!
- Predecessor is opposite of successor
  - Find node y that is the largest y < x (slides typo)
- If the inorder traversal is 10 20 30 40 50
  - Node's 30 successor is 40 (directly after)
  - Node's 30 predecessor is 20 (directly before)

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."