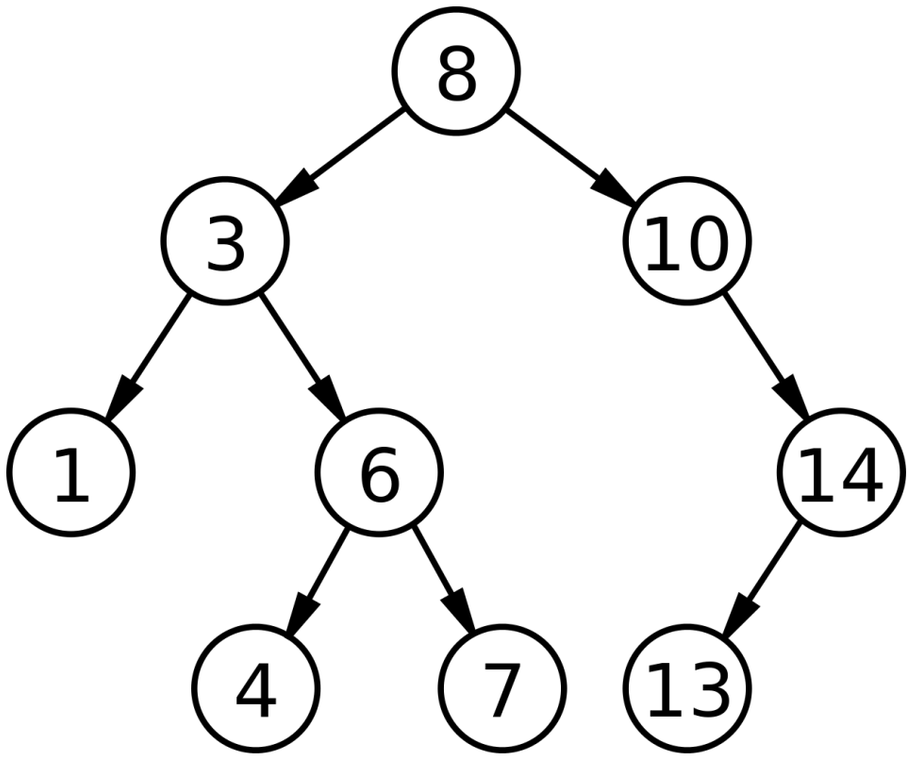
# CS 61BL Lab 11

Ryan Purpura

# Announcements

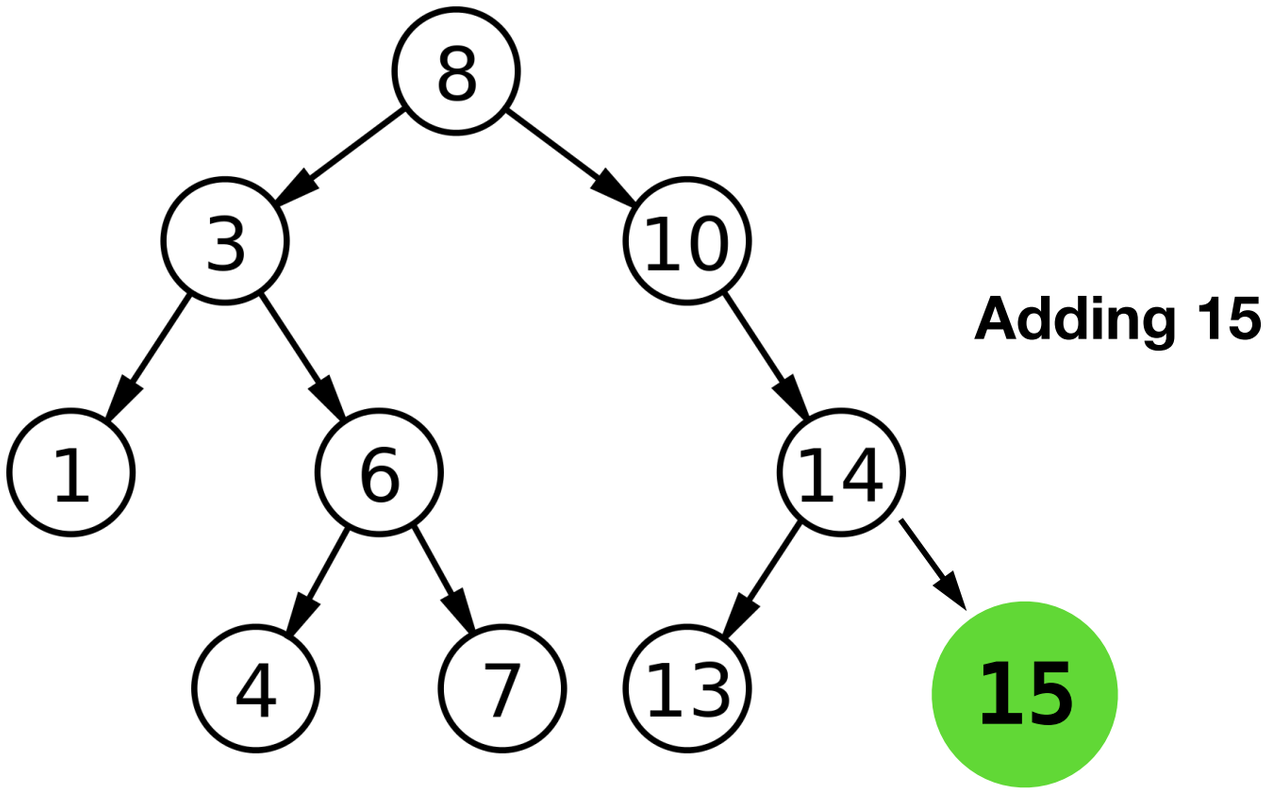
- Midterm 1 scores are out; regrade requests are open and will close Friday at the end of the day.
- Gitlet is due next Friday, July 26 at the end of the day.
- One clarification from Friday's lab: I said that when adding new nodes, you always add the new node to a leaf; this is not always true. For example:




**Adding 15**

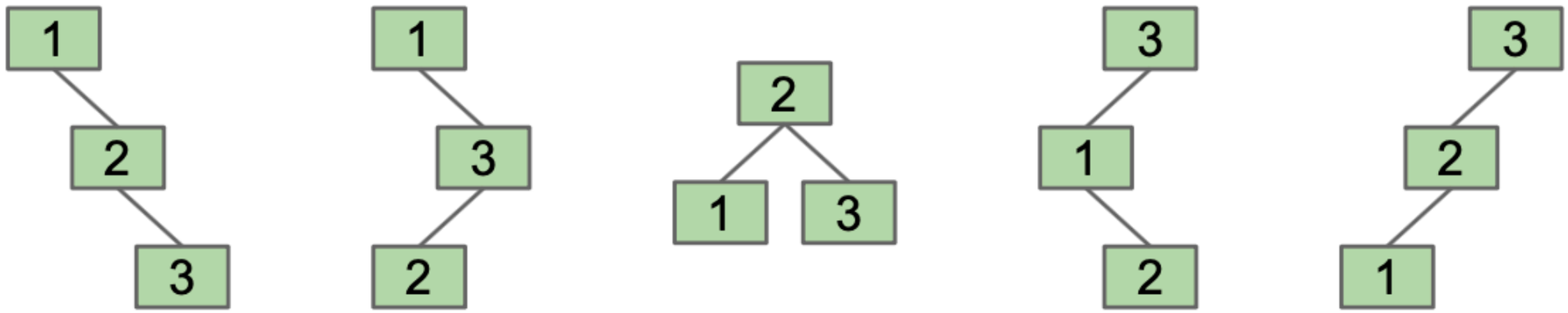
# Announcements

- Midterm 1 scores are out; regrade requests are open and will close Friday at the end of the day.
- Gitlet is due next Friday, July 26 at the end of the day.
- One clarification from Friday's lab: I said that when adding new nodes, you always add the new node to a leaf; this is not always true. For example:



# Tree Rotations

- The same elements can be organized into different binary search trees.

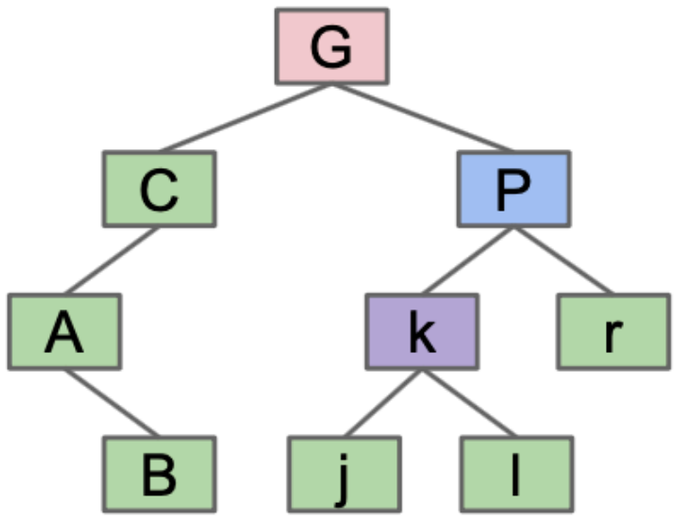


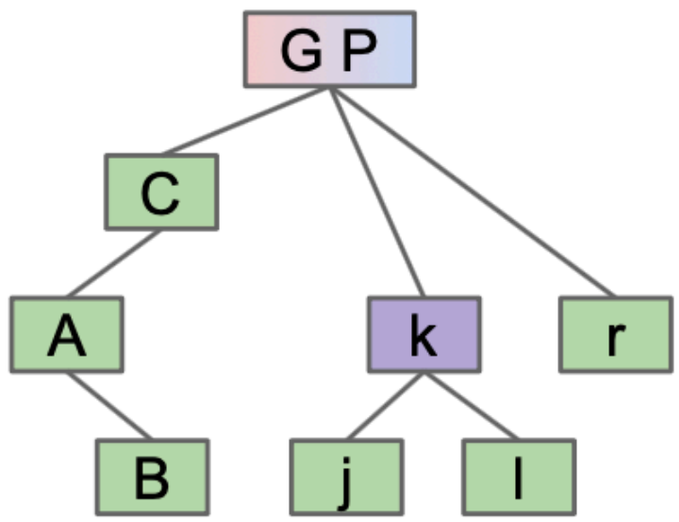

- We can use "tree rotations" to convert between equivalent binary search trees.

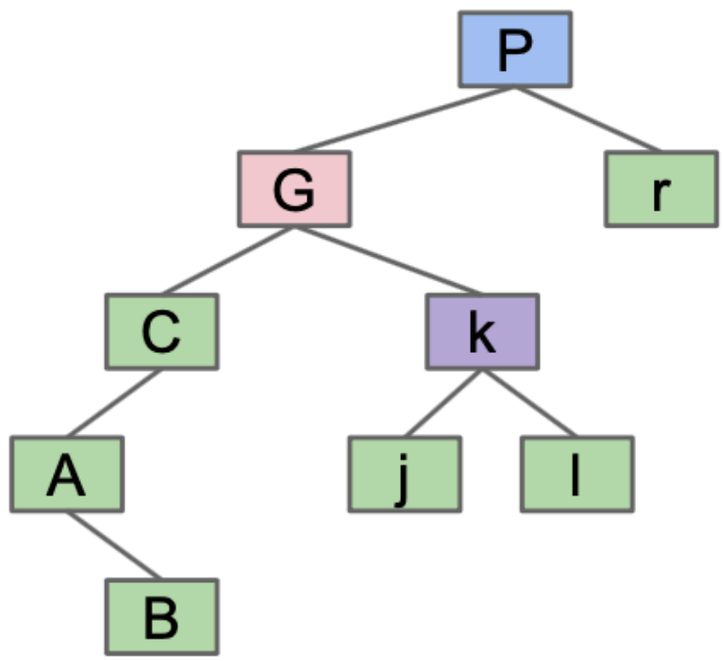
**Credit: Josh Hug**

# Rotate Left

- rotateLeft(G):
  Let x be the right child of G. Make G the new left child of x.
- Can think of it as temporarily merging G and P and then sending G down and to the left. Notice what happens to k!



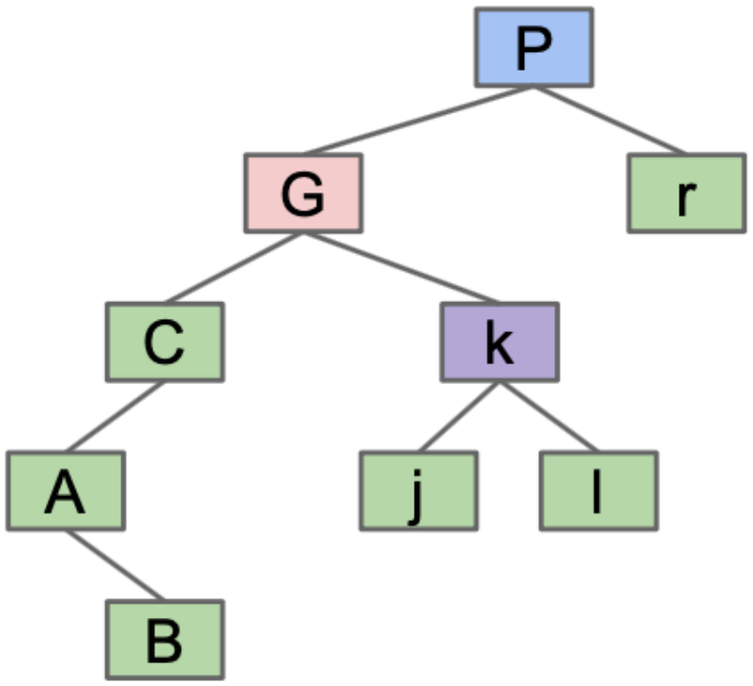



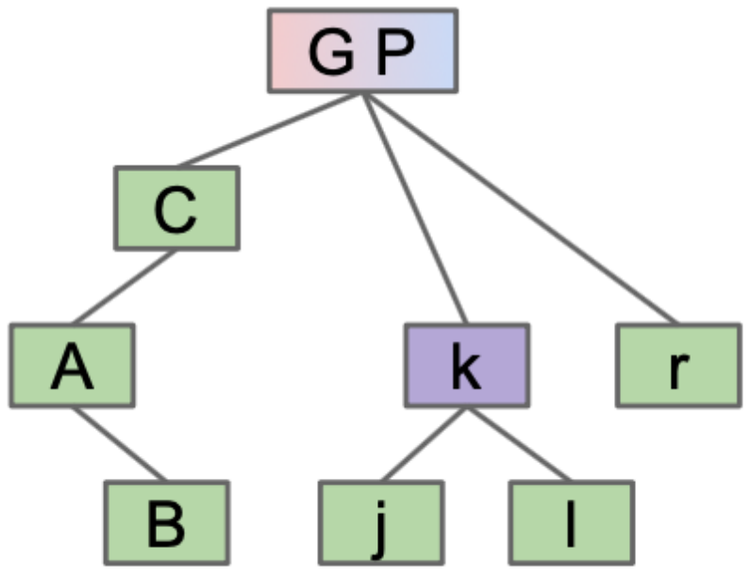



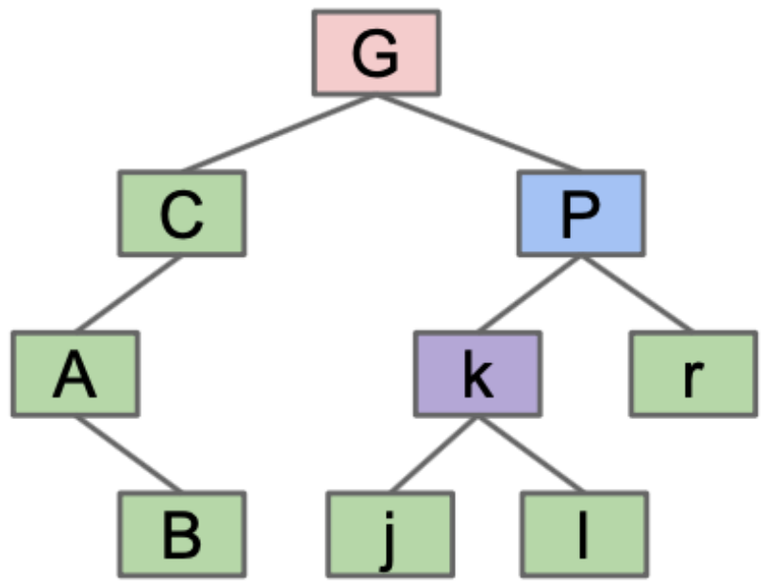

**Credit: Josh Hug**

# Rotate Right

- rotateRight(P):
  Let x be the left child of P. Make P the new right child of x.
- Can think of it as temporarily merging G and P and then sending P down and to the right. Notice what happens to k!



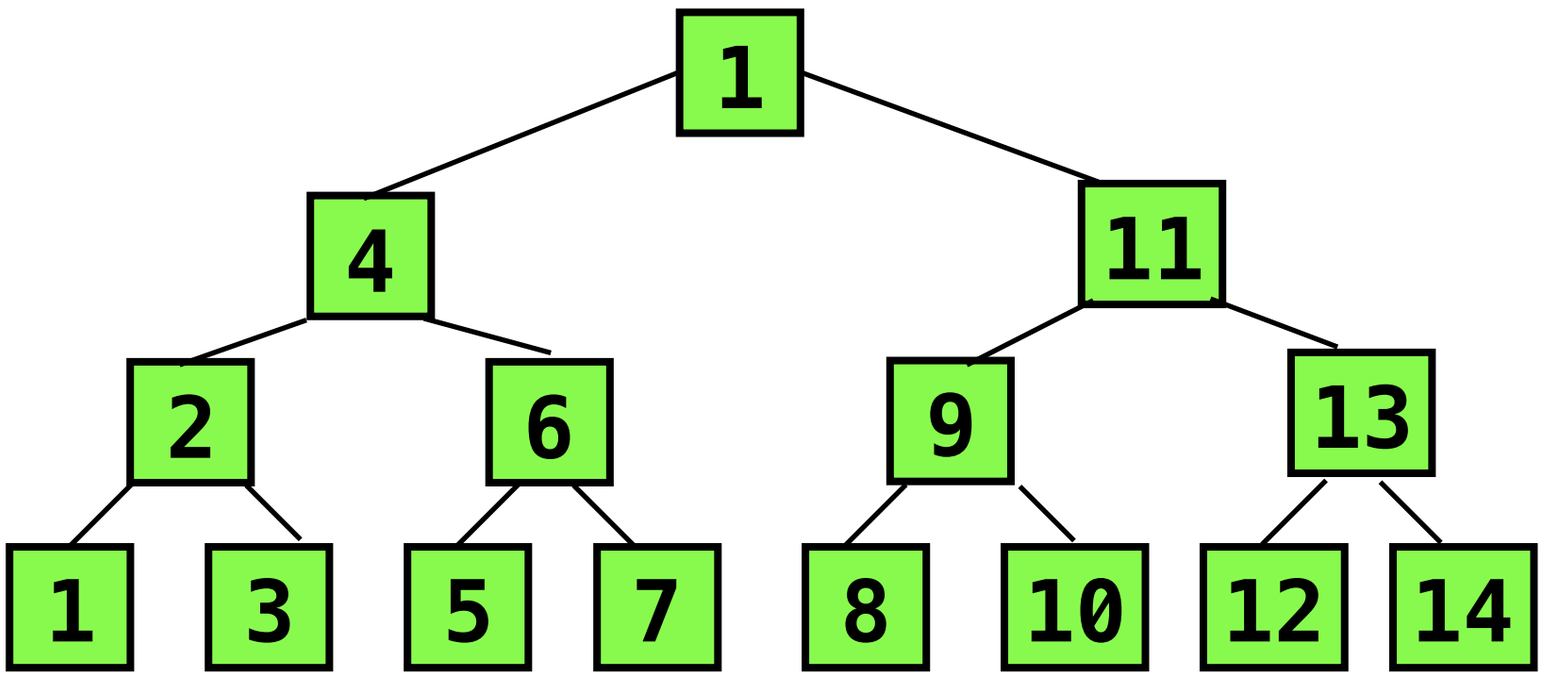



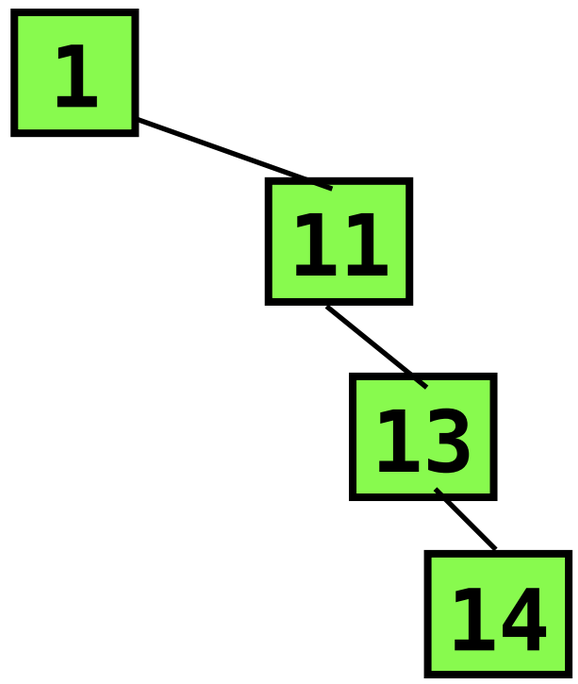

**Credit: Josh Hug**

# Bushy vs. Spindly Trees

- Binary trees come in many shapes and sizes:




**Nice bushy tree.**




**Very spindly tree: basically a linked list!**

# Making Bushy Trees

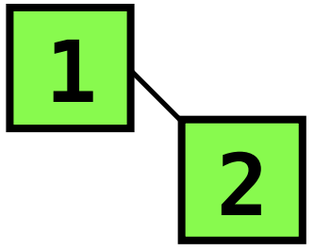
- Idea 1: Smarter insertion order.
- Task: Make a balanced search tree with values 1, 2, 3, 4, 5, 6, 7
- Bad approach: add in order.

# Making Bushy Trees

- Idea 1: Smarter insertion order.
- Task: Make a balanced search tree with values 1, 2, 3, 4, 5, 6, 7
- Bad approach: add in order.

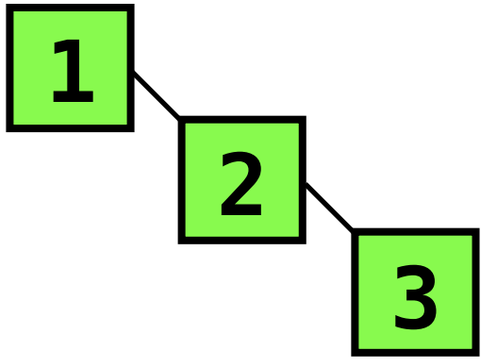
# Making Bushy Trees

- Idea 1: Smarter insertion order.
- Task: Make a balanced search tree with values 1, 2, 3, 4, 5, 6, 7
- Bad approach: add in order.



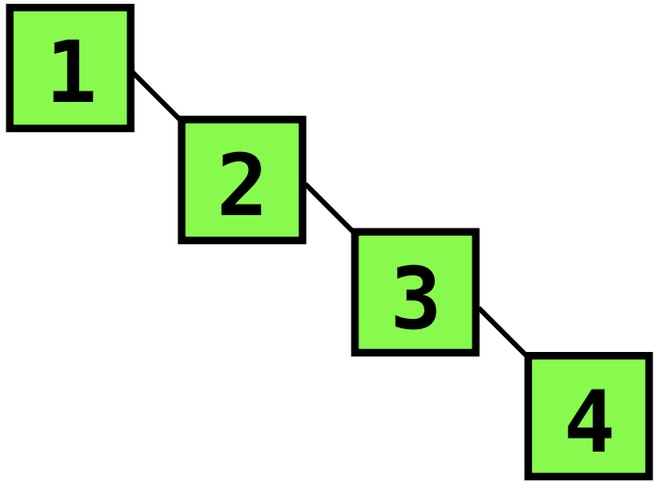
# Making Bushy Trees

- Idea 1: Smarter insertion order.
- Task: Make a balanced search tree with values 1, 2, 3, 4, 5, 6, 7
- Bad approach: add in order.



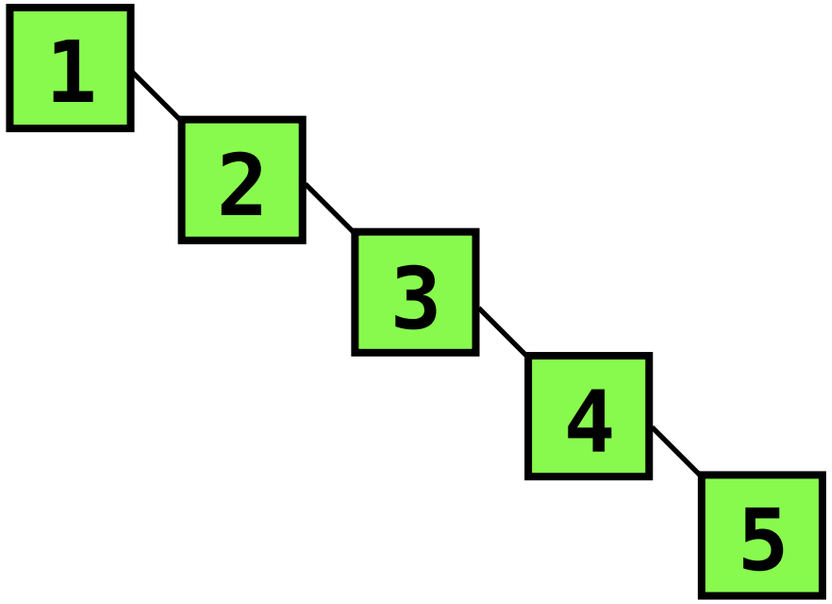
# Making Bushy Trees

- Idea 1: Smarter insertion order.
- Task: Make a balanced search tree with values 1, 2, 3, 4, 5, 6, 7
- Bad approach: add in order.



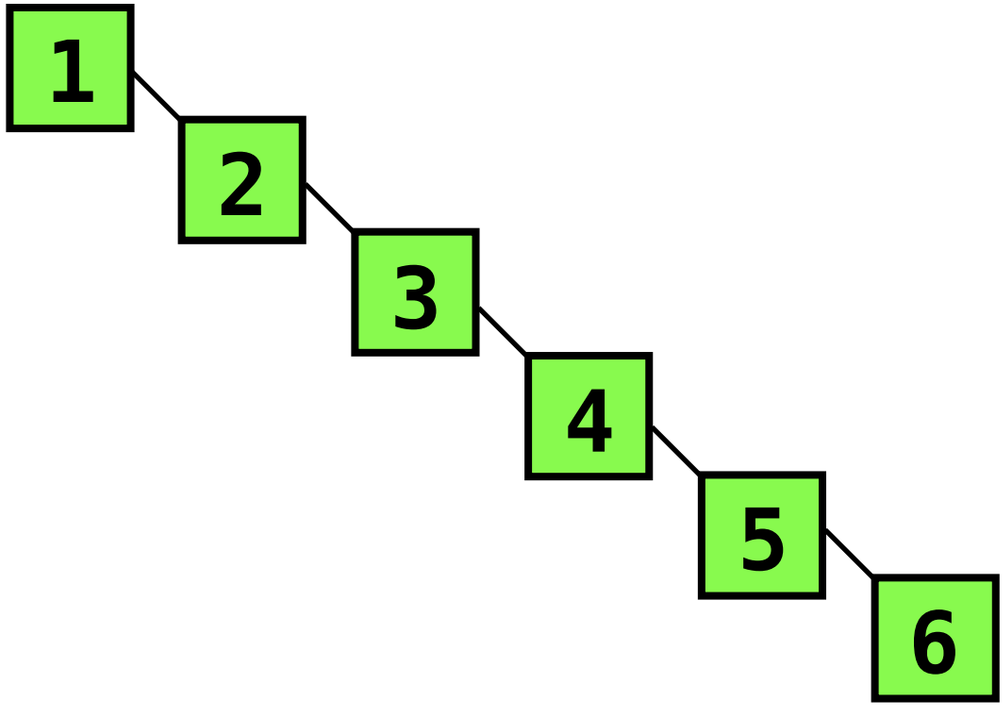
# Making Bushy Trees

- Idea 1: Smarter insertion order.
- Task: Make a balanced search tree with values 1, 2, 3, 4, 5, 6, 7
- Bad approach: add in order.



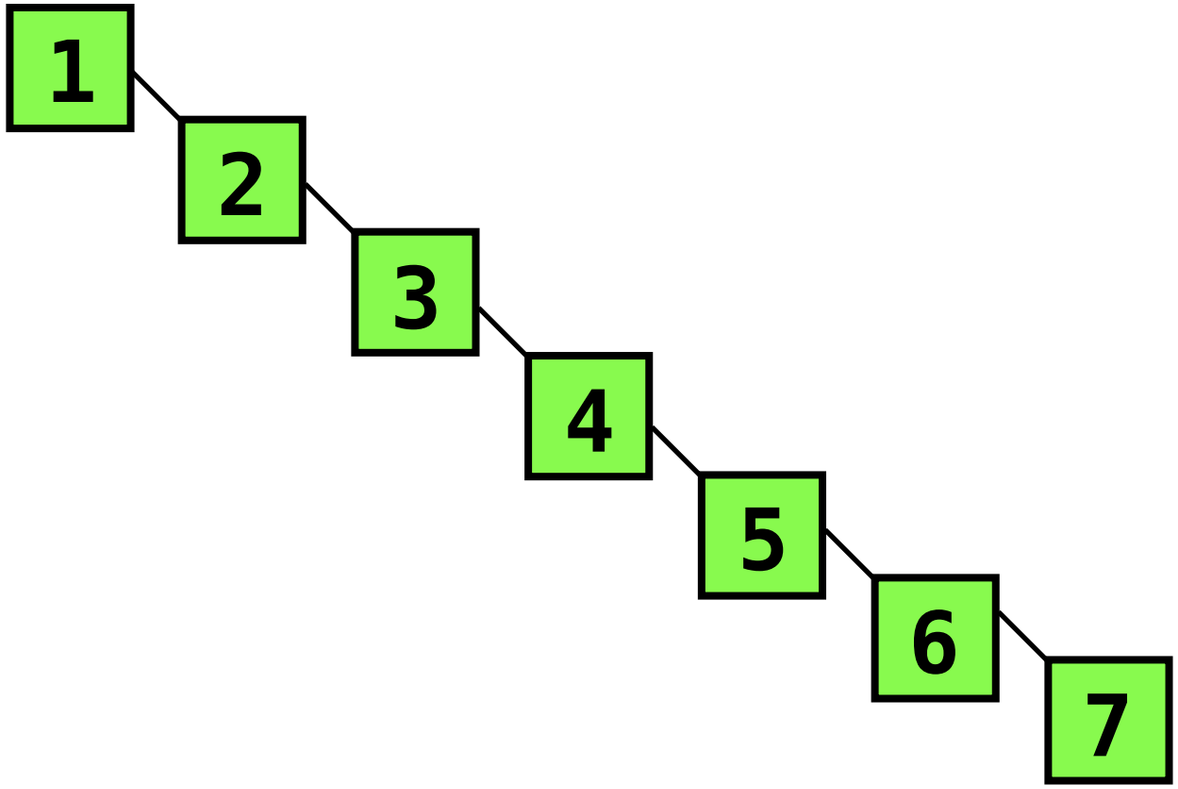
# Making Bushy Trees

- Idea 1: Smarter insertion order.
- Task: Make a balanced search tree with values 1, 2, 3, 4, 5, 6, 7
- Bad approach: add in order.



# Making Bushy Trees

- Idea 1: Smarter insertion order.
- Task: Make a balanced search tree with values 1, 2, 3, 4, 5, 6, 7
- Bad approach: add in order.



# Better Approach

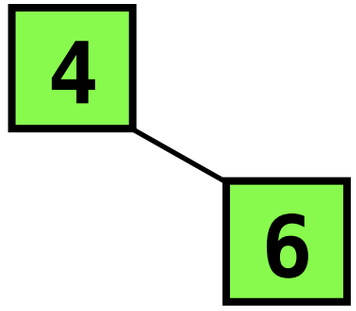
- Start in the middle, so that we make sure each node will have an equal number of elements for both children.
- So start with 4.

# Better Approach

- Start in the middle, so that we make sure each node will have an equal number of elements for both children.
- So start with 4.

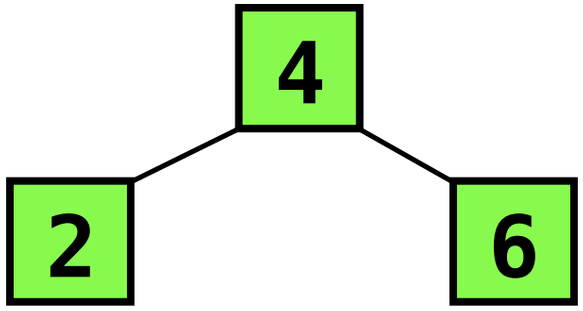
# Better Approach

- Start in the middle, so that we make sure each node will have an equal number of elements for both children.
- So start with 4.



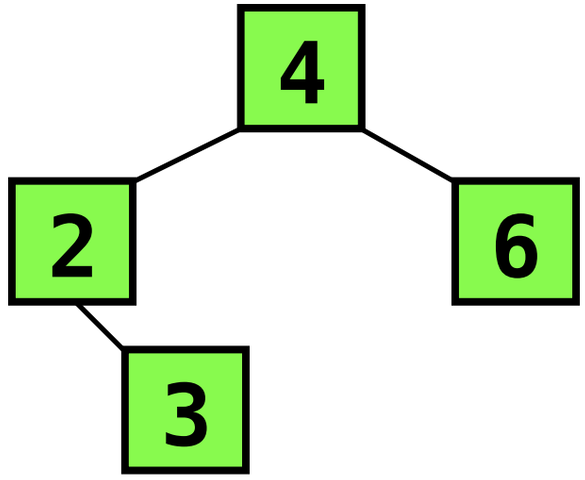
# Better Approach

- Start in the middle, so that we make sure each node will have an equal number of elements for both children.
- So start with 4.



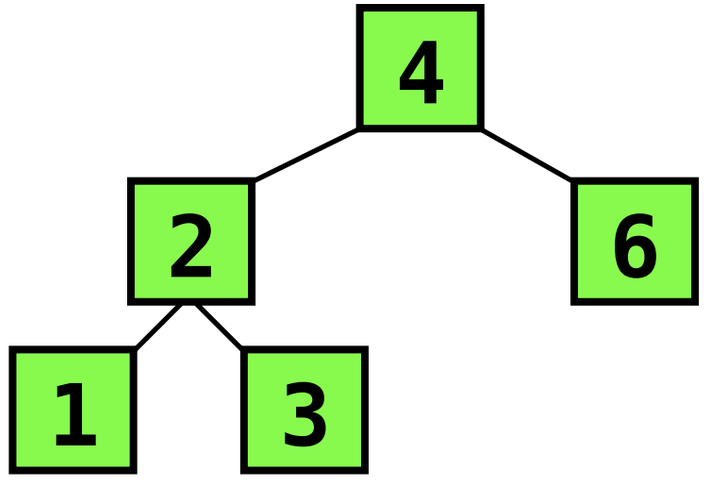
# Better Approach

- Start in the middle, so that we make sure each node will have an equal number of elements for both children.
- So start with 4.



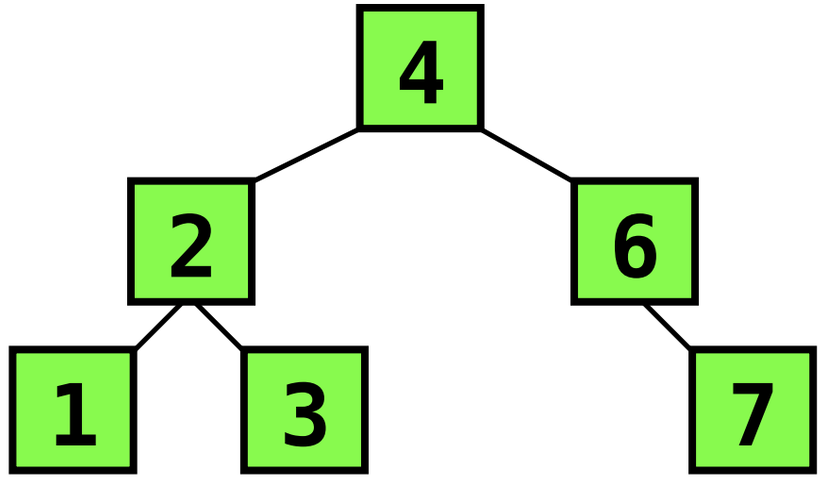
# Better Approach

- Start in the middle, so that we make sure each node will have an equal number of elements for both children.
- So start with 4.



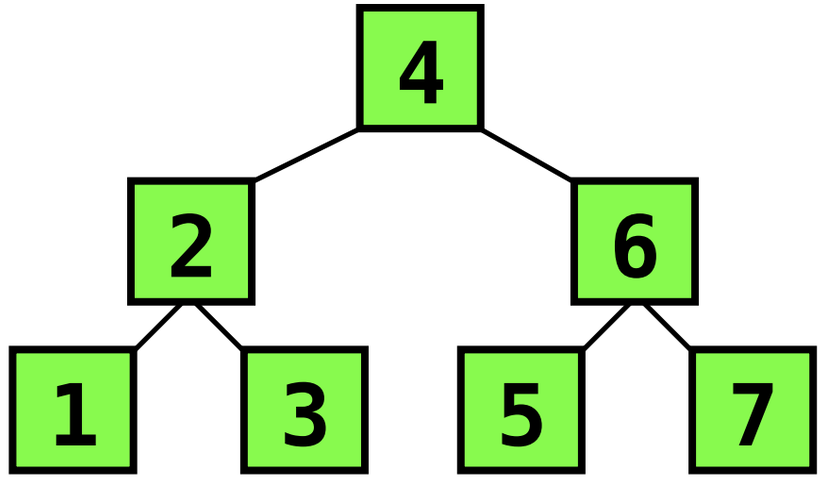
# Better Approach

- Start in the middle, so that we make sure each node will have an equal number of elements for both children.
- So start with 4.



# Better Approach

- Start in the middle, so that we make sure each node will have an equal number of elements for both children.
- So start with 4.

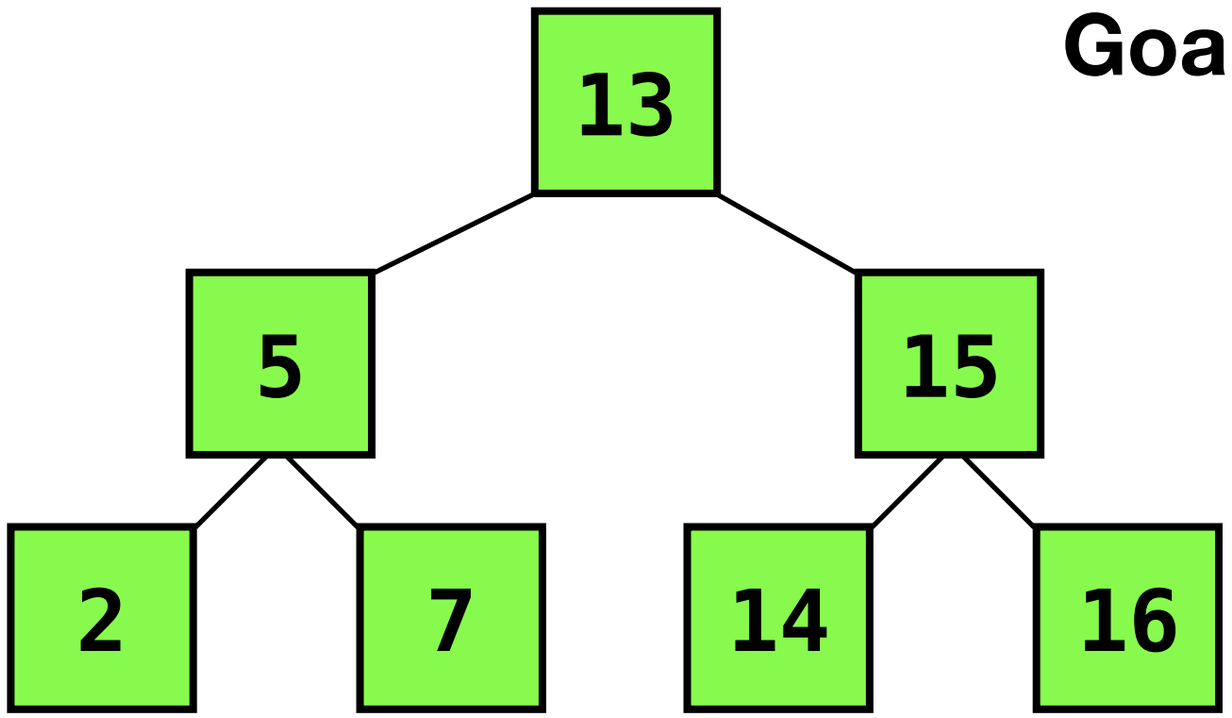


# What's the issue with this?

- We don't always have all the data in advance.

# 🅱️-Trees

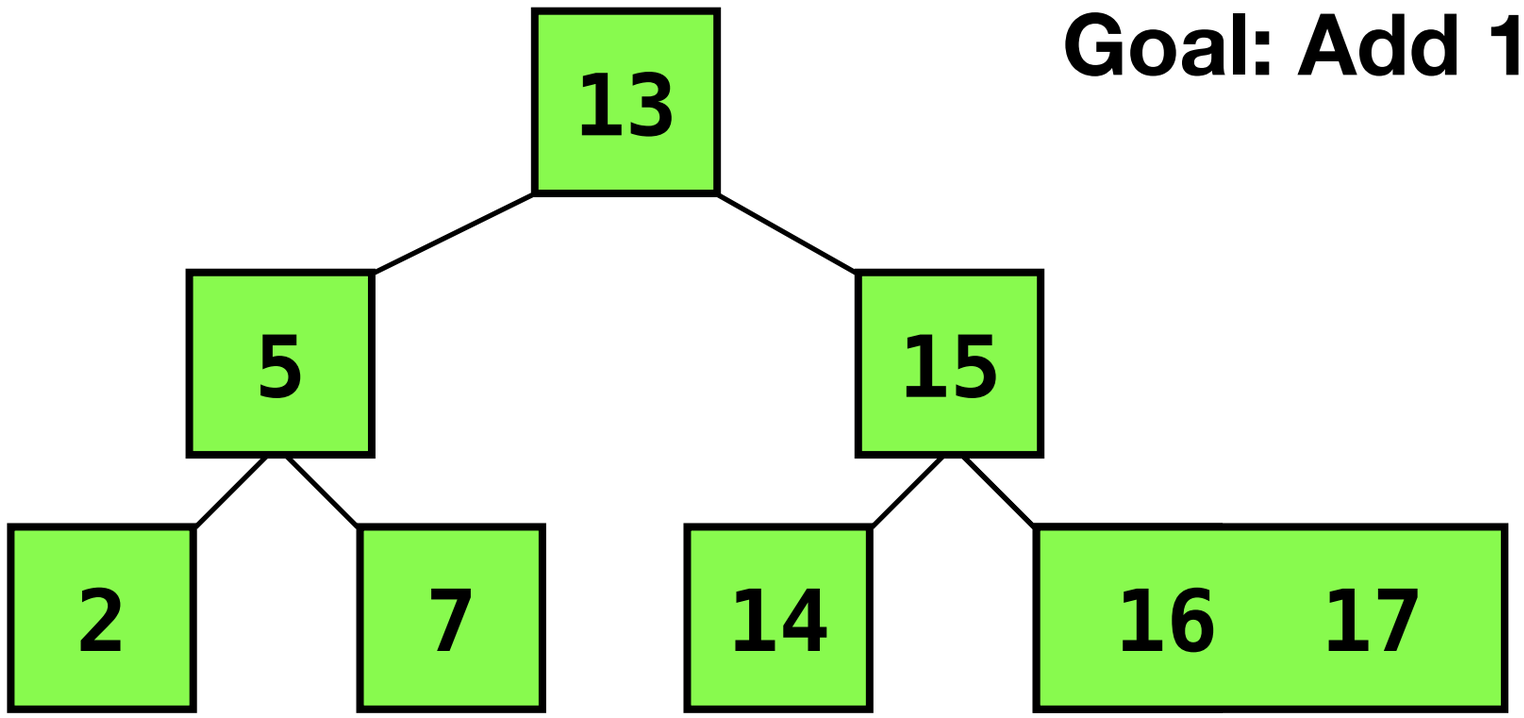
- The problem is adding leaves.
- So let's never add leaves!
- ...what do we do with insertions then?



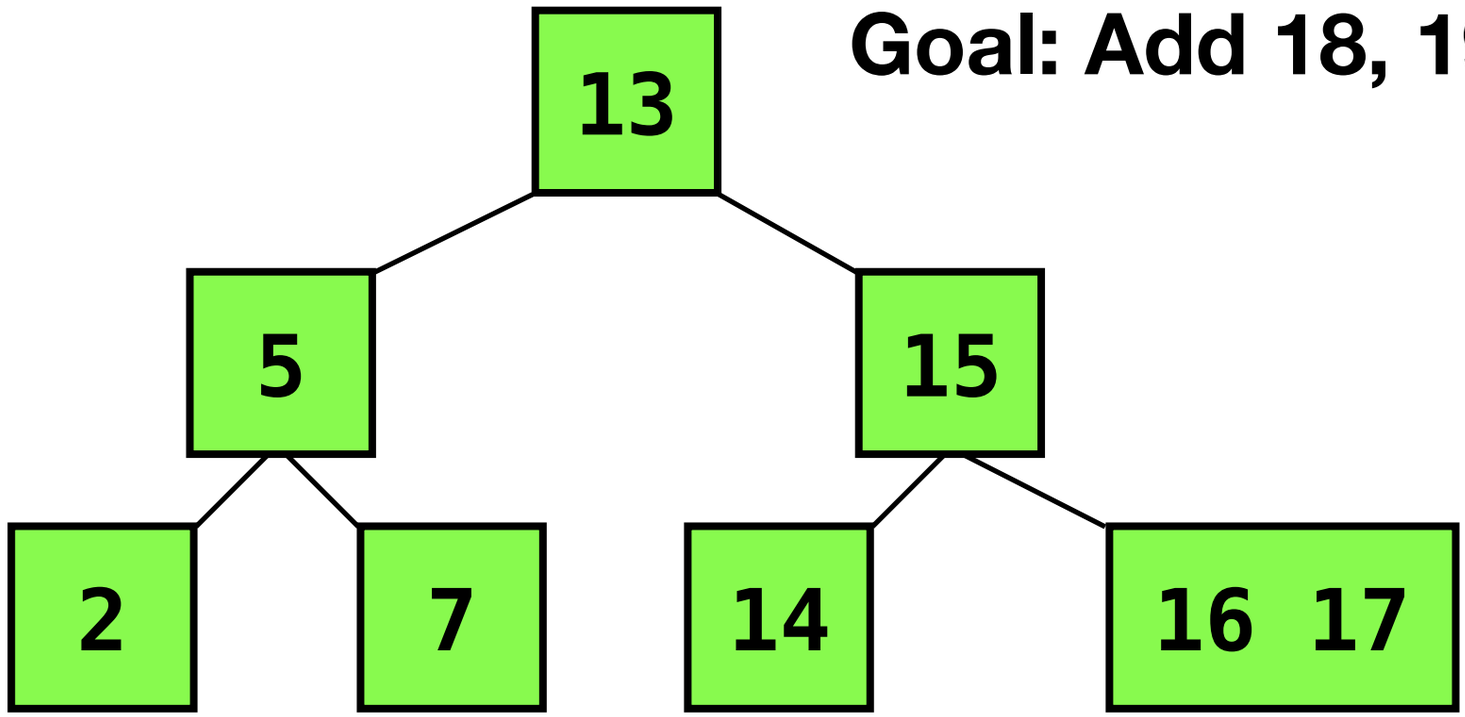

**Goal: Add 17.**

# 🅱️-Trees

- The problem is adding leaves.
- So let's never add leaves!
- ...what do we do with insertions then?

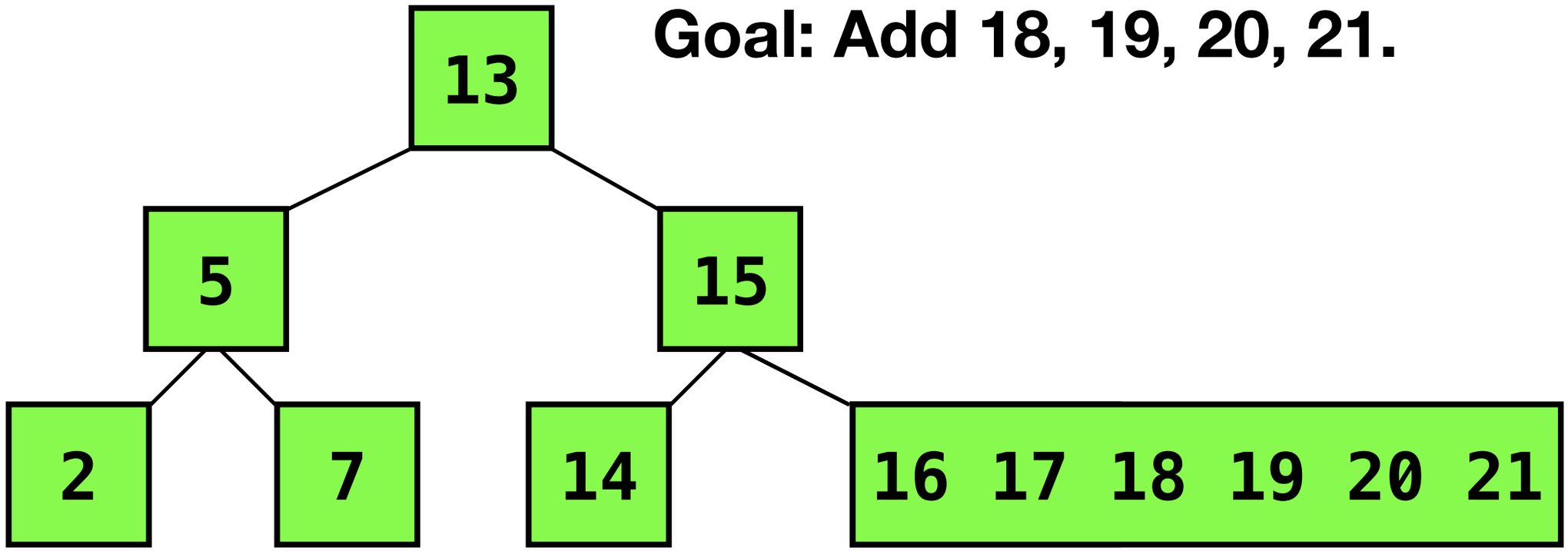


# Let's keep going!



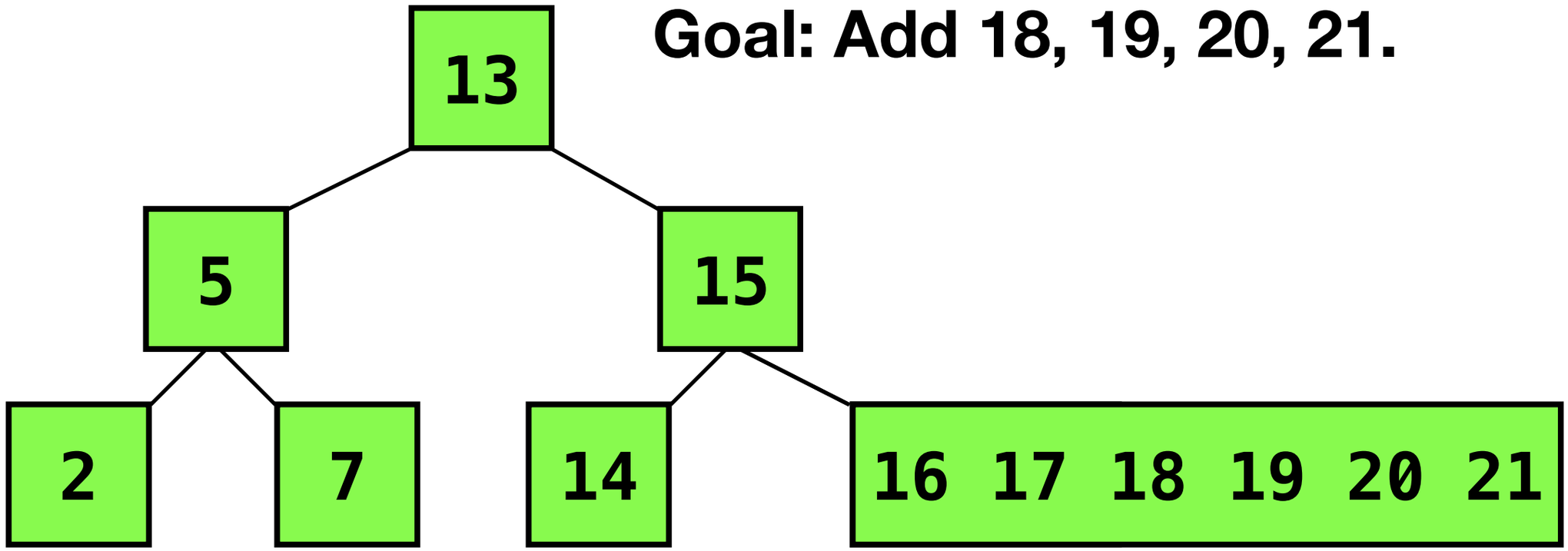

**Goal: Add 18, 19, 20, 21.**

# Let's keep going!




**Goal: Add 18, 19, 20, 21.**

# Let's keep going!

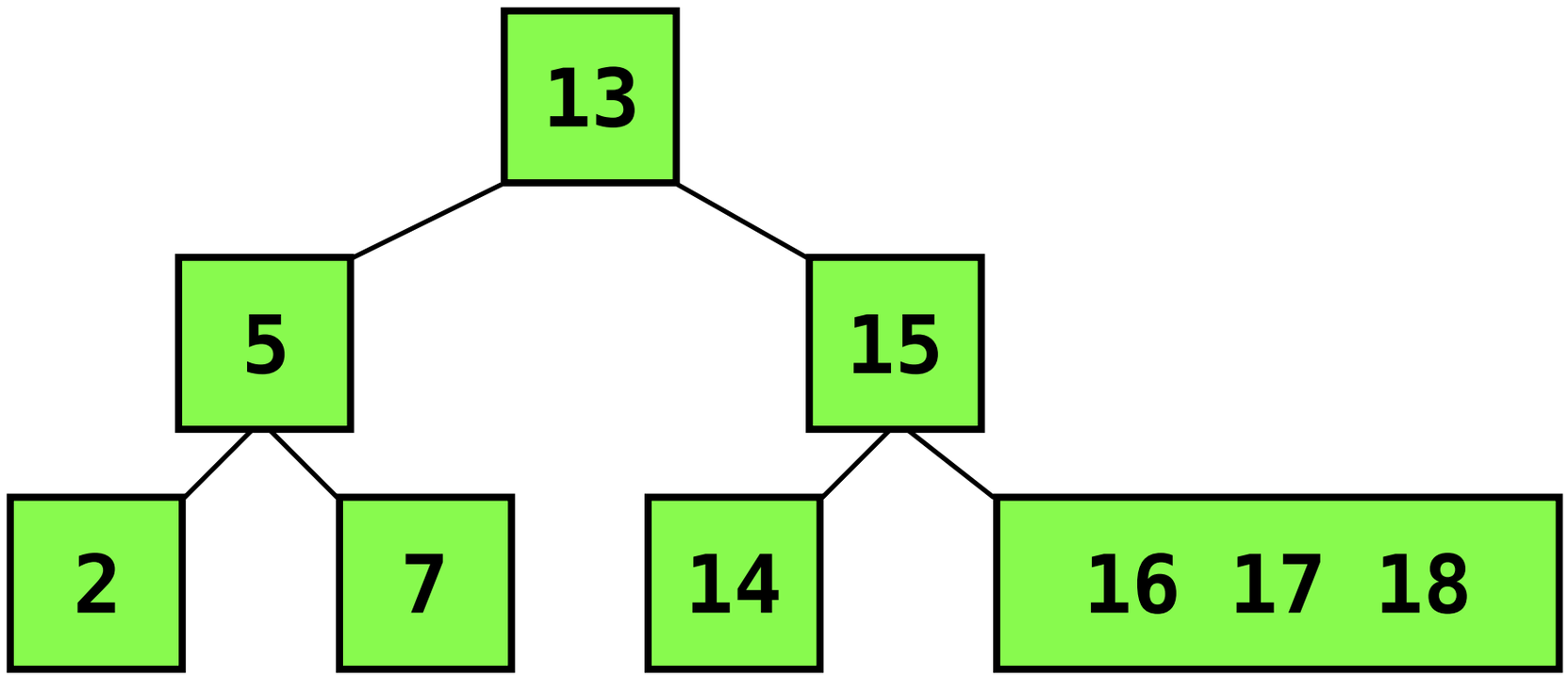



**Goal: Add 18, 19, 20, 21.**

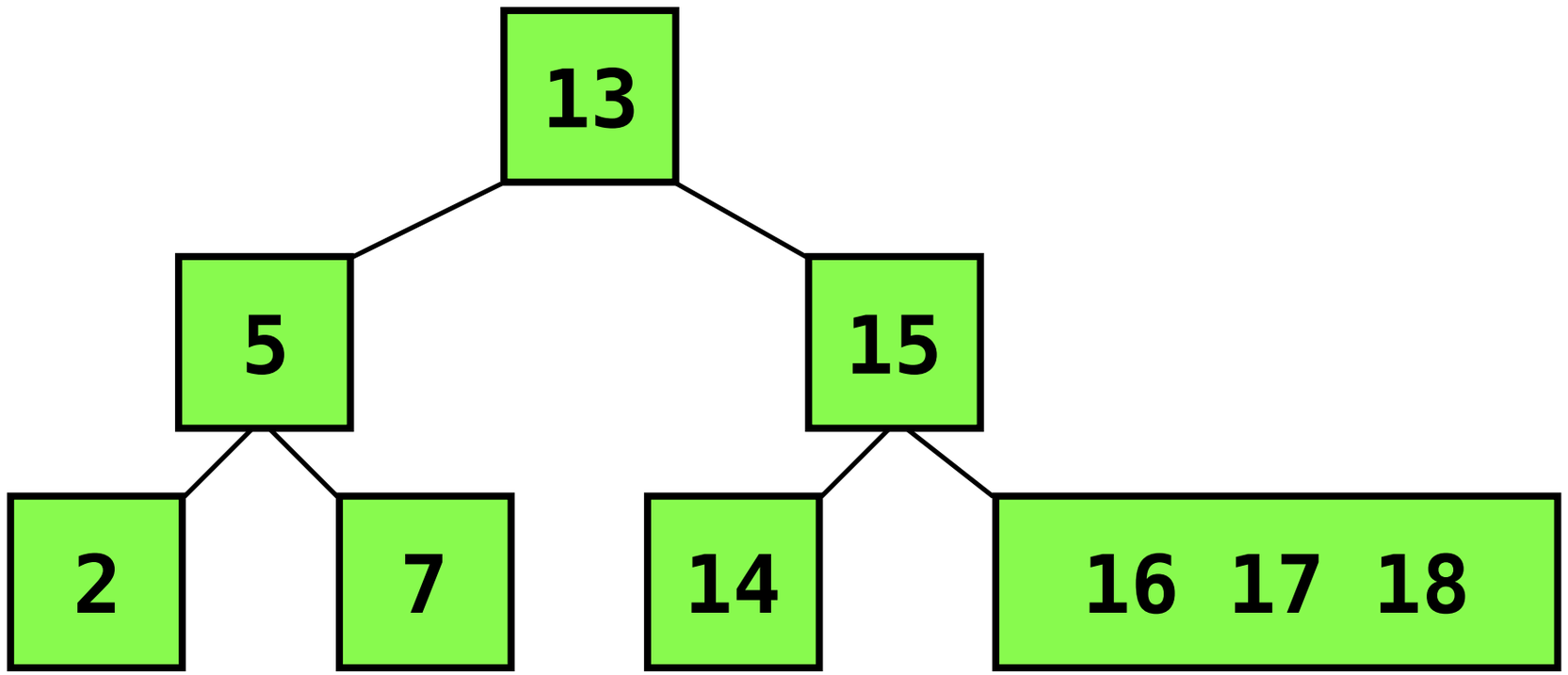
**This is getting unwieldy...**

# Item Limit

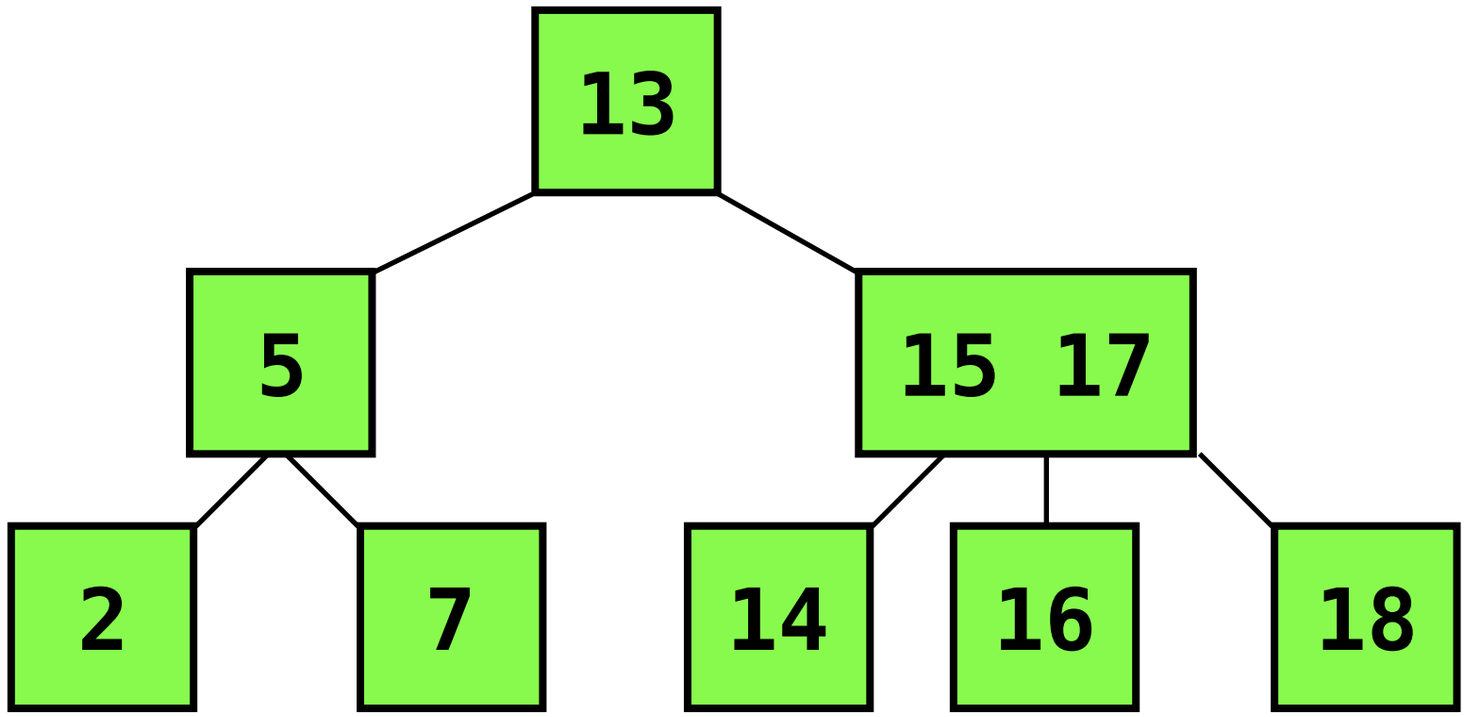
- Idea: add an item limit for each node.
- If the leaf we want to add to is at its item limit, give the middle item to its parent and split, then add.
- Example: Given limit 3, what happens when we insert 19?



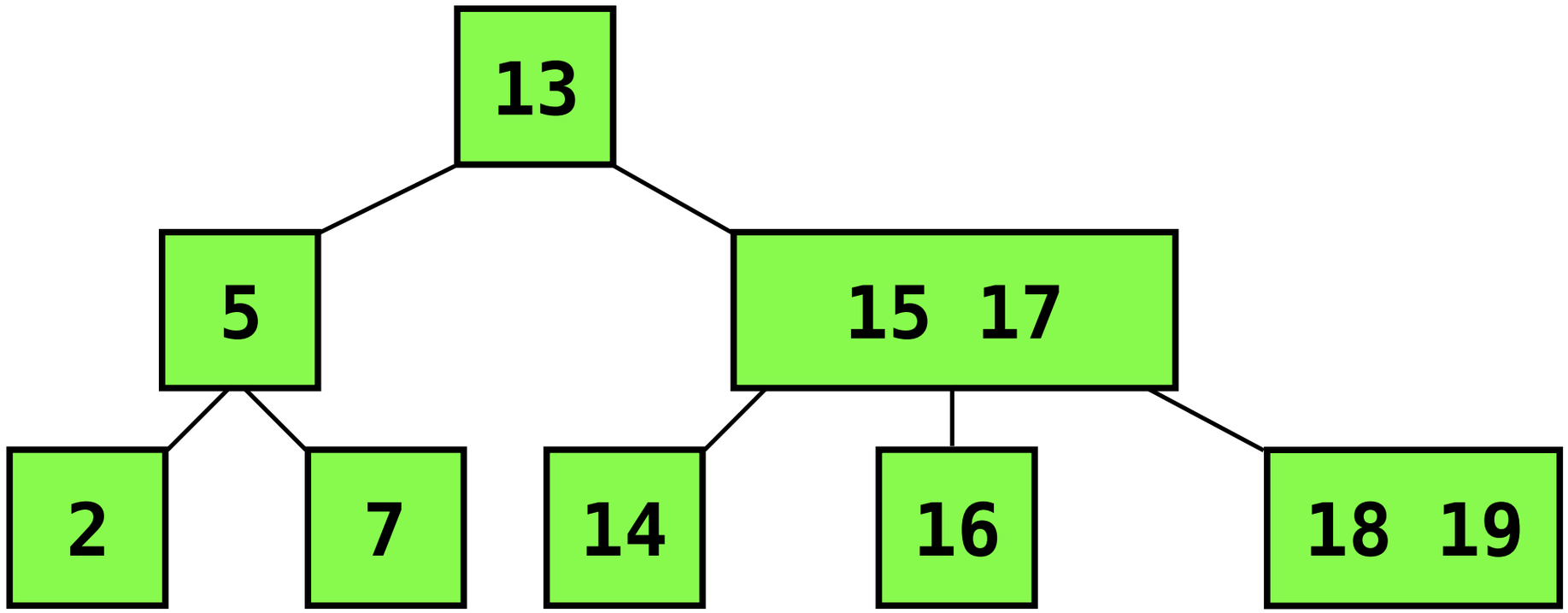
# Adding 19 with Limit 3



# Adding 19 with Limit 3

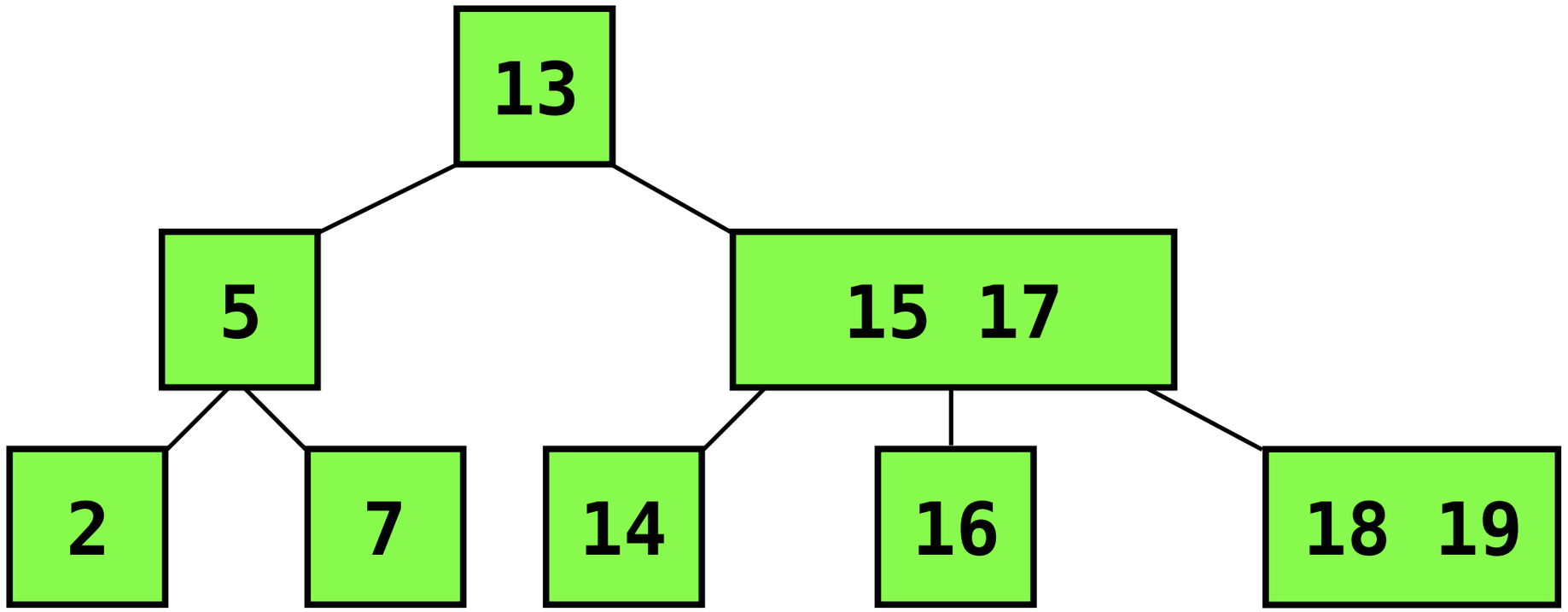


# Adding 19 with Limit 3

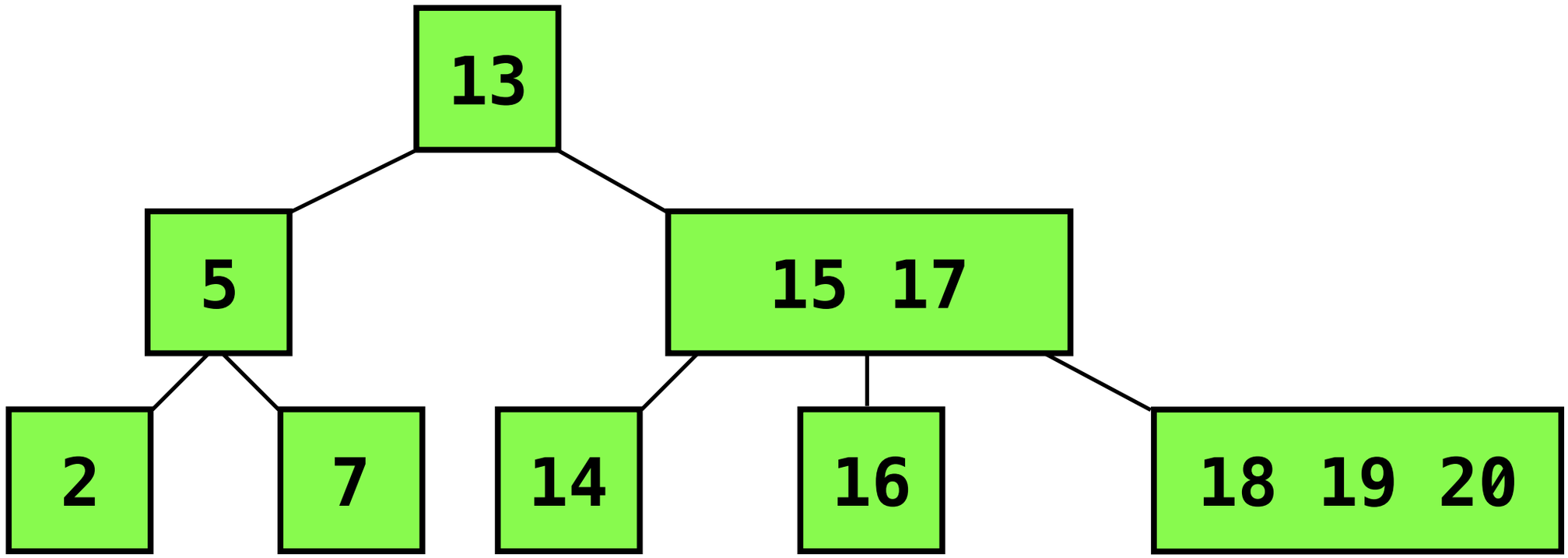



**Now we can easily fit in 19.**

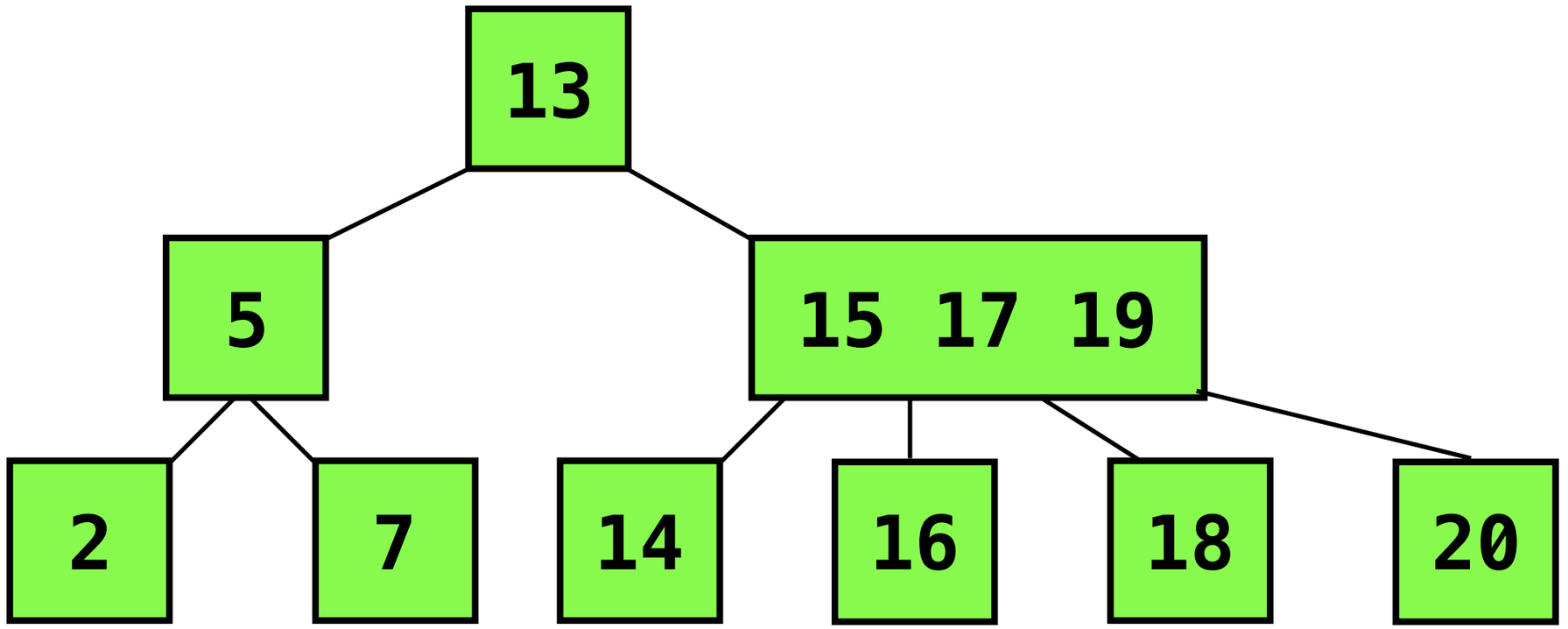
# Add 20



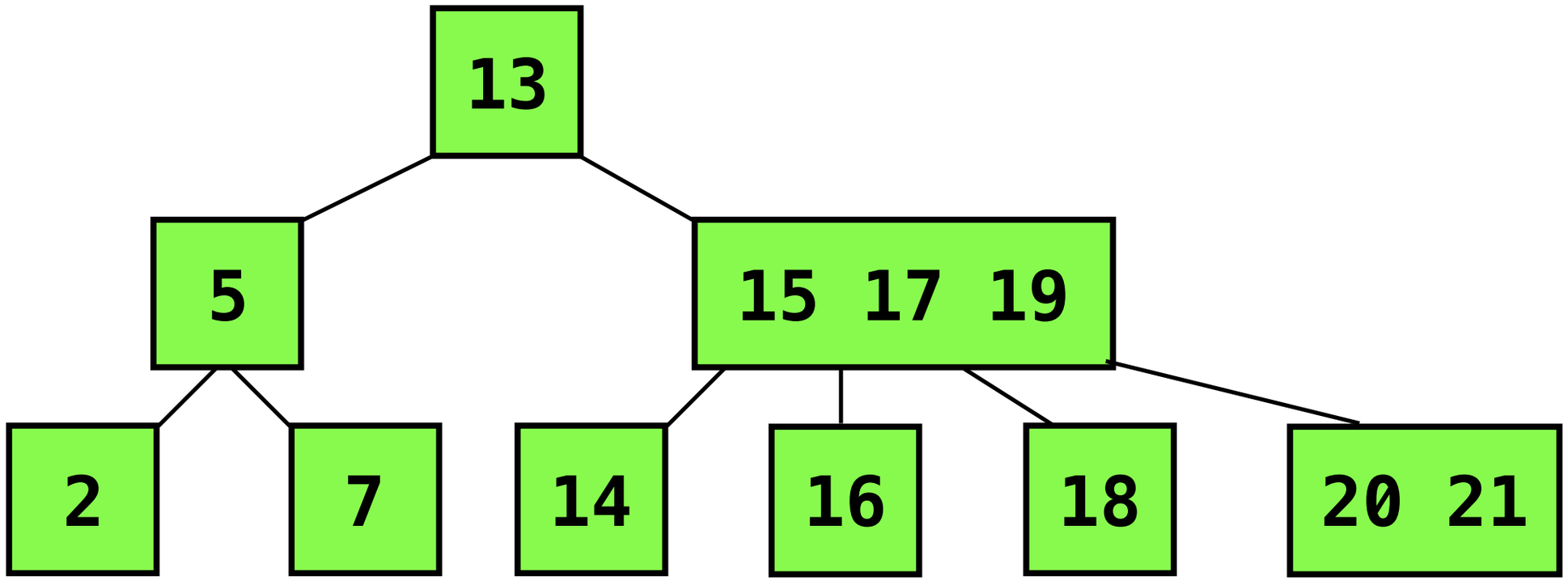
# Add 20



# Add 21

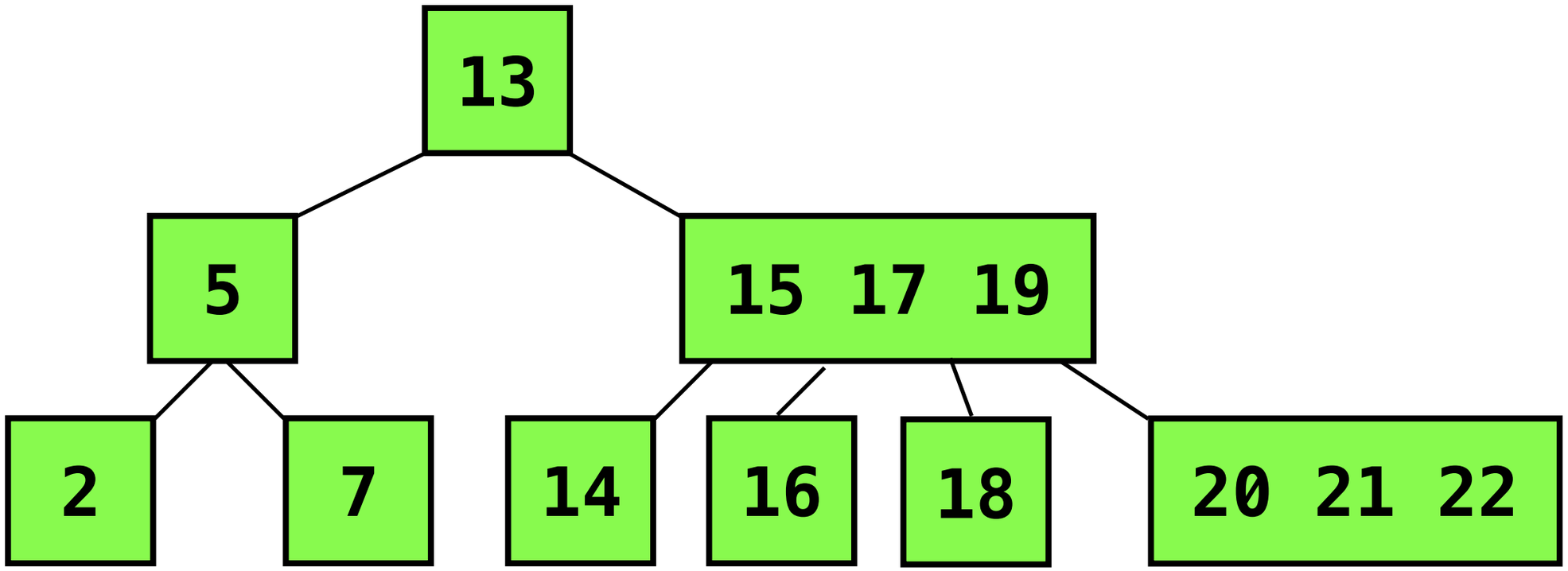


# Add 21



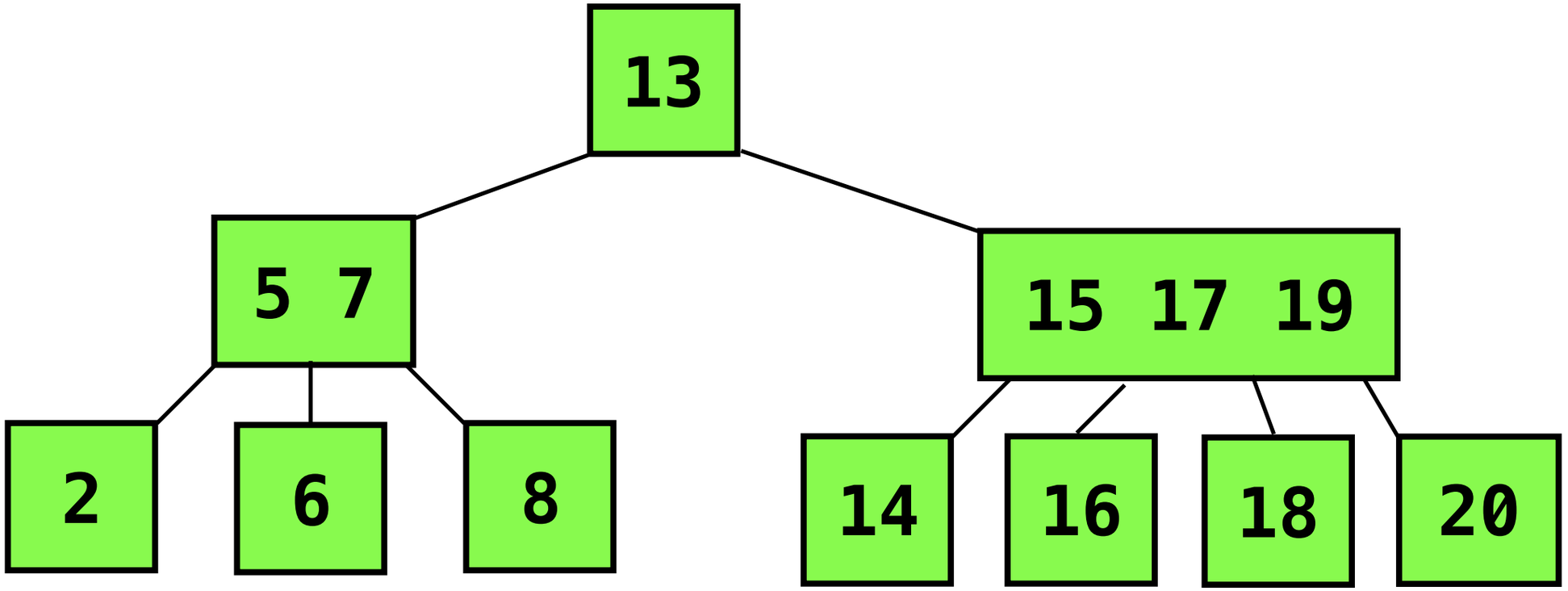
# This is called a B-Tree

- "What, if anything, the *B* stands for has never been established." --Wikipedia
- What we were working with before is a type of B-tree called a 2-3-4 tree, since each node can have two, three, or four children.



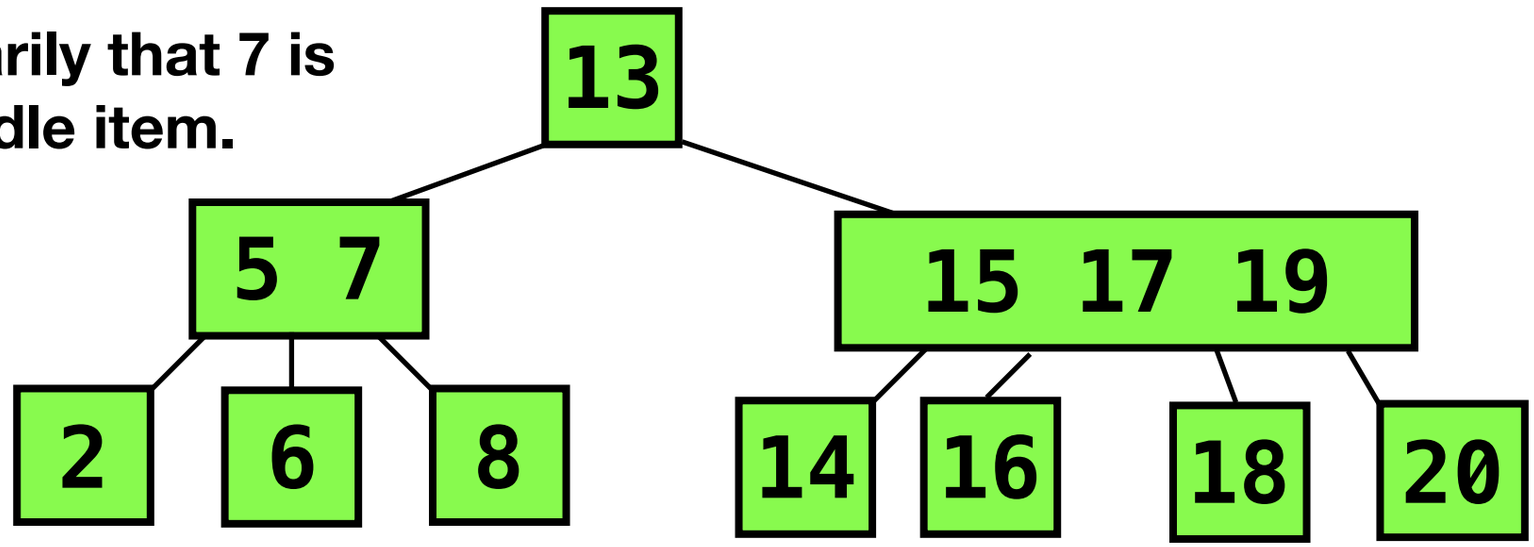
# 🅱️-Tree limitations

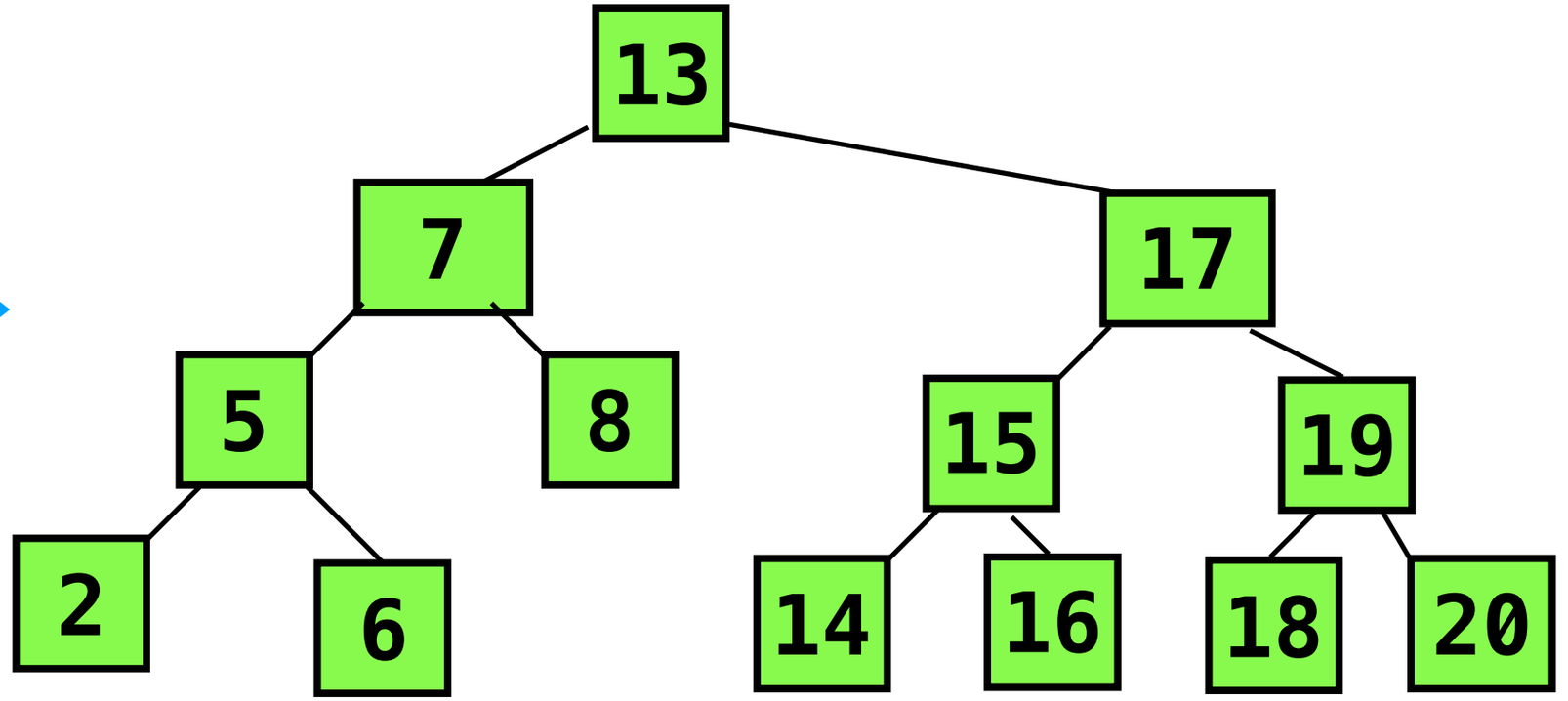
- B-trees are annoying to implement, but give us guaranteed $\log N$ height.
- Can we convert the 2-3-4 trees into binary trees and simplify our logic?



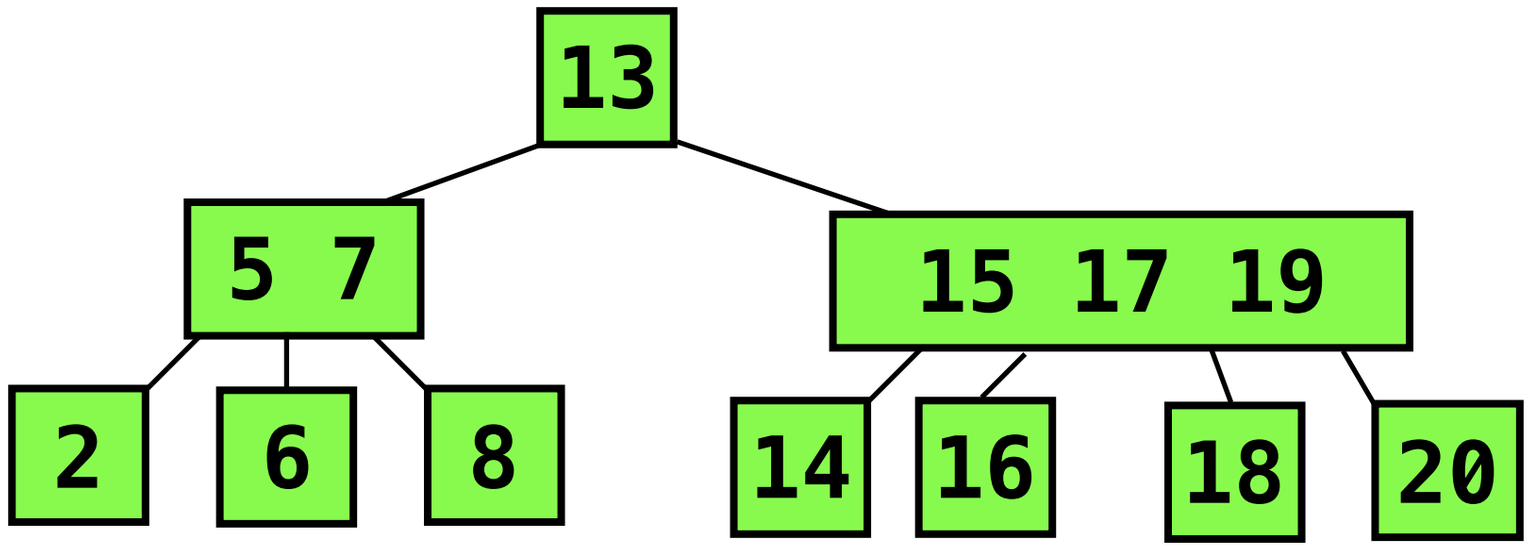
# Idea: "Drop down" the Left and Right Items

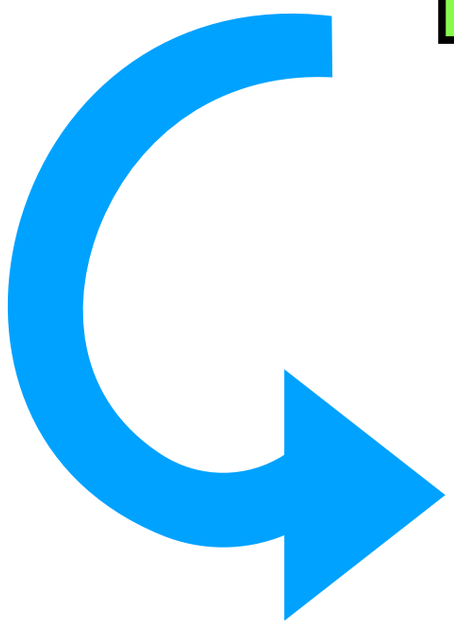
**Pick arbitrarily that 7 is the middle item.**

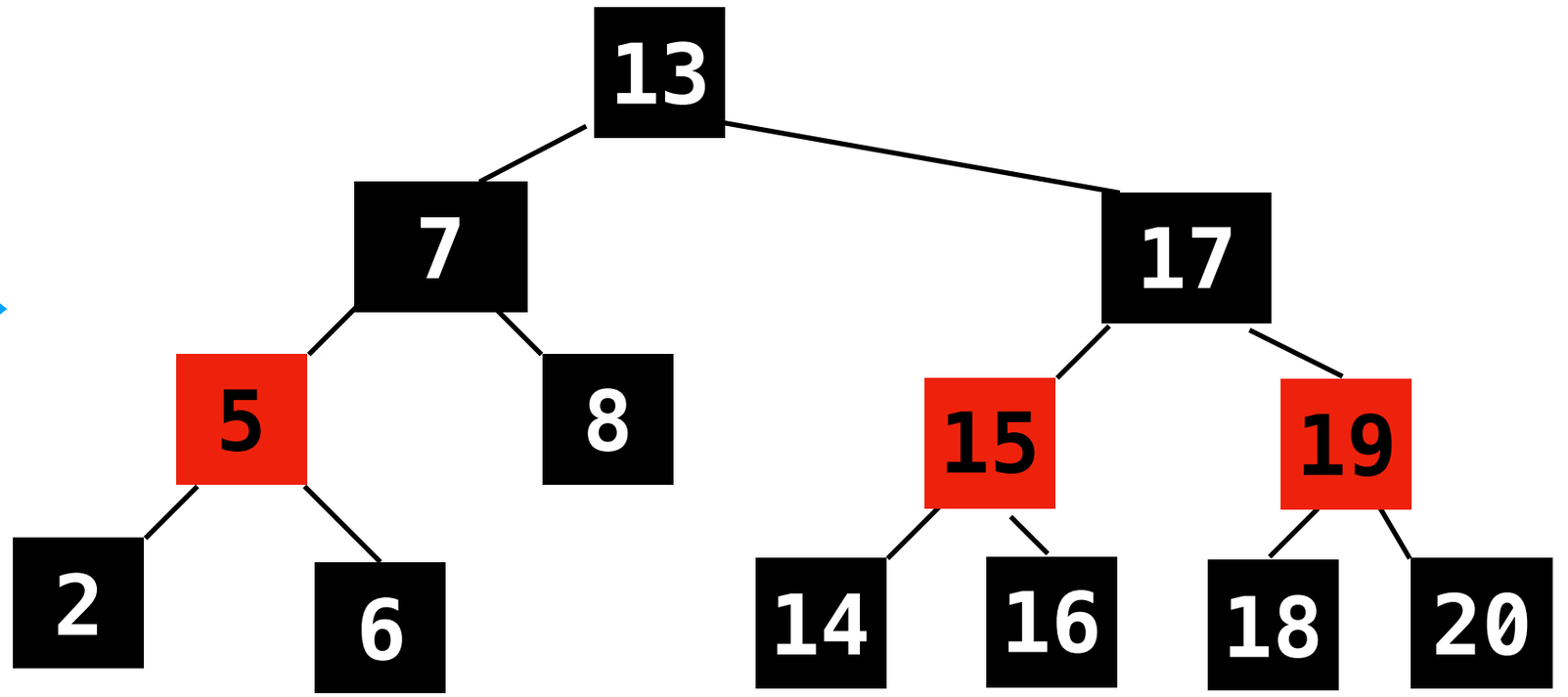


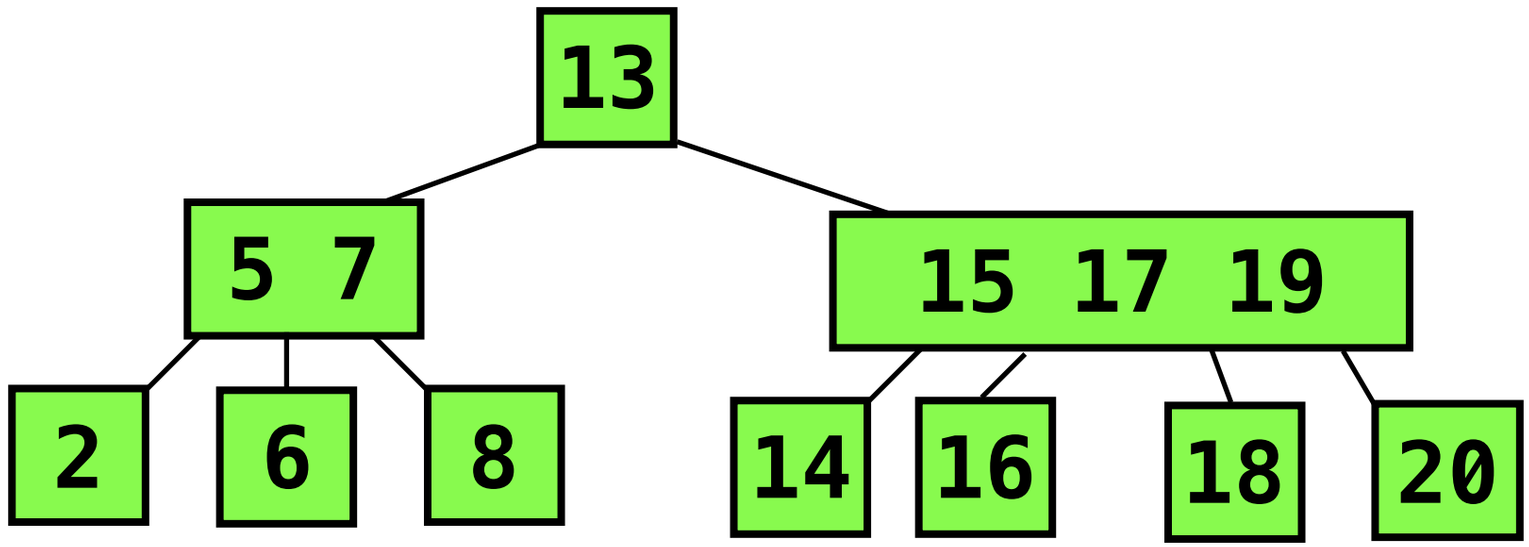
Color nodes that we dropped down red.
Color everything else black.

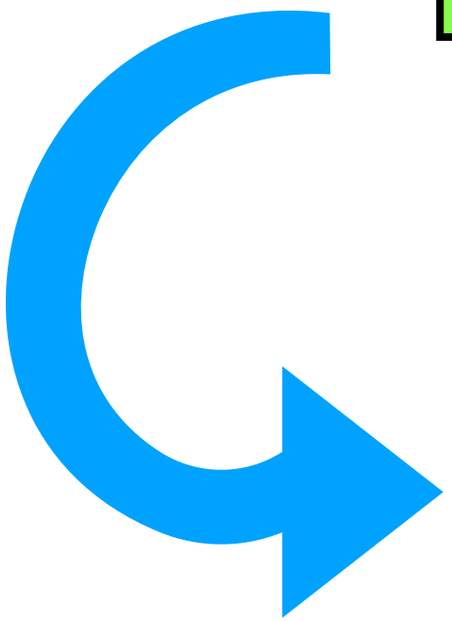


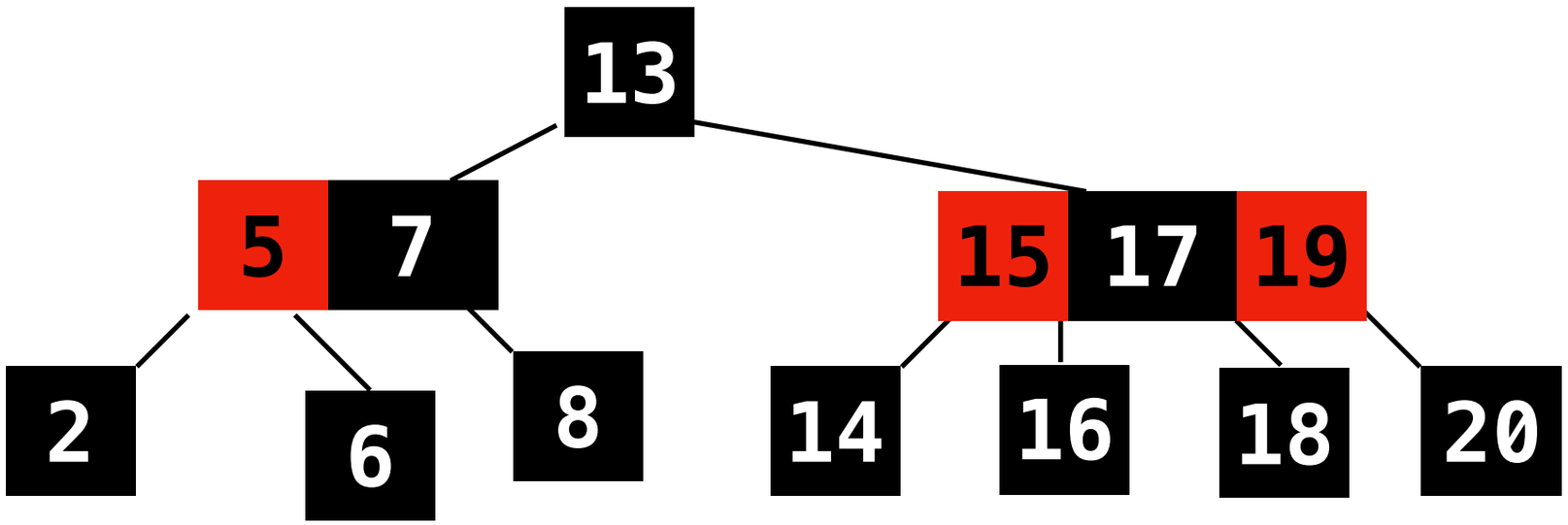





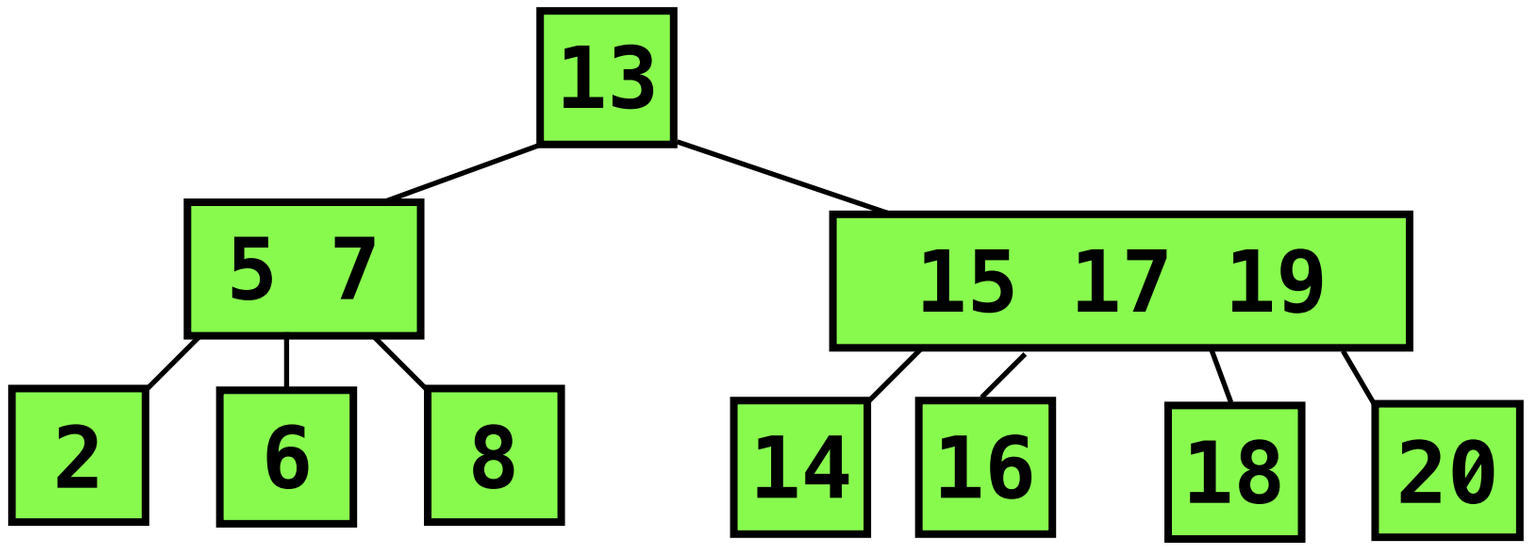
# To convert back, just move all the red elements up.

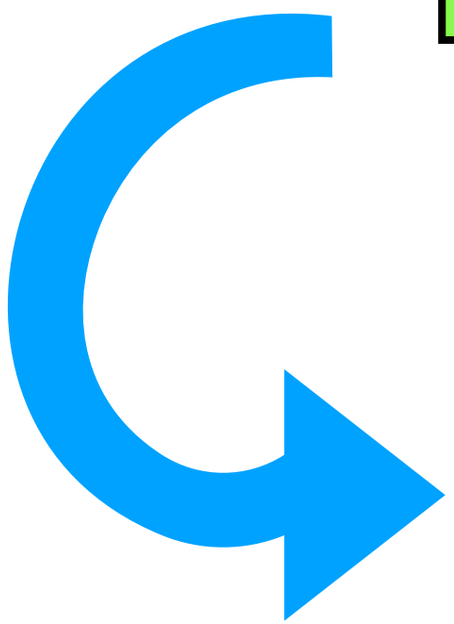


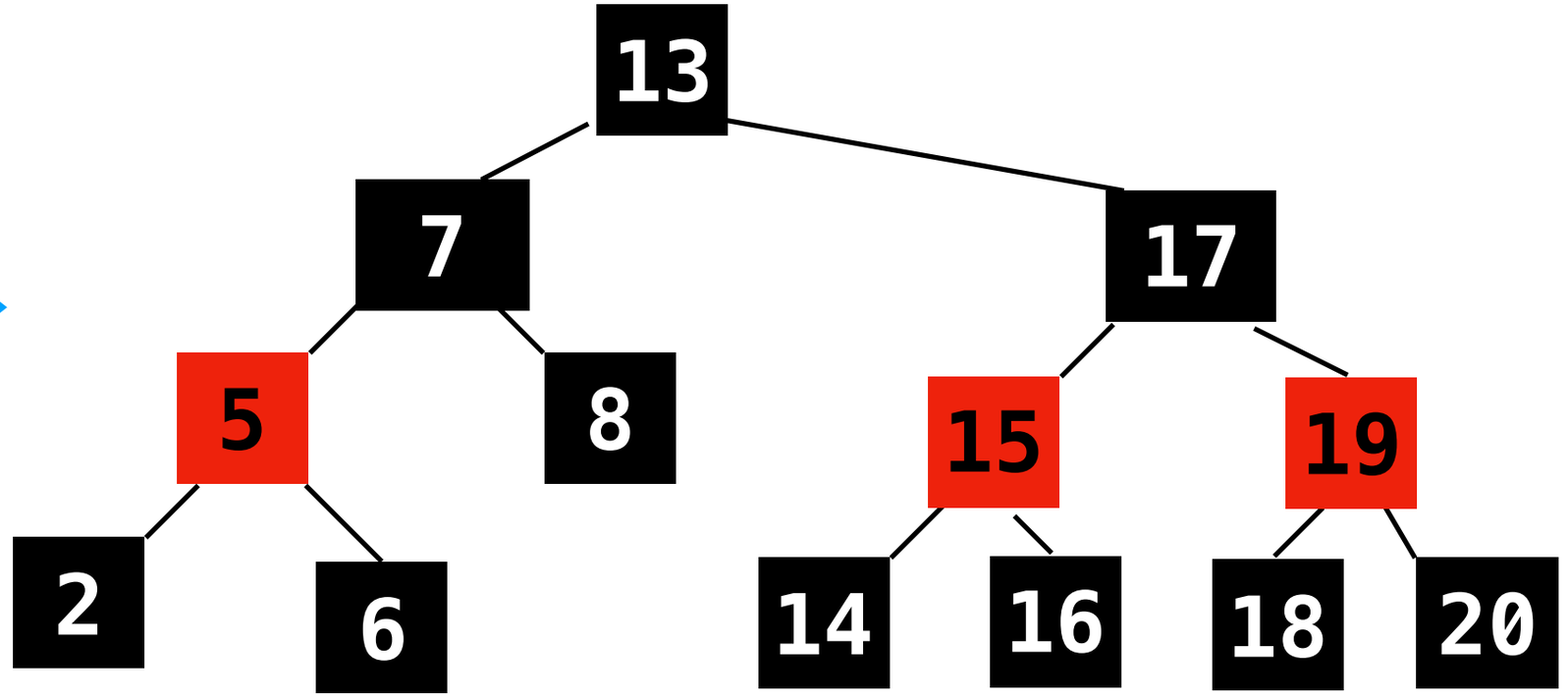





# This is called a "Red-Black Tree"



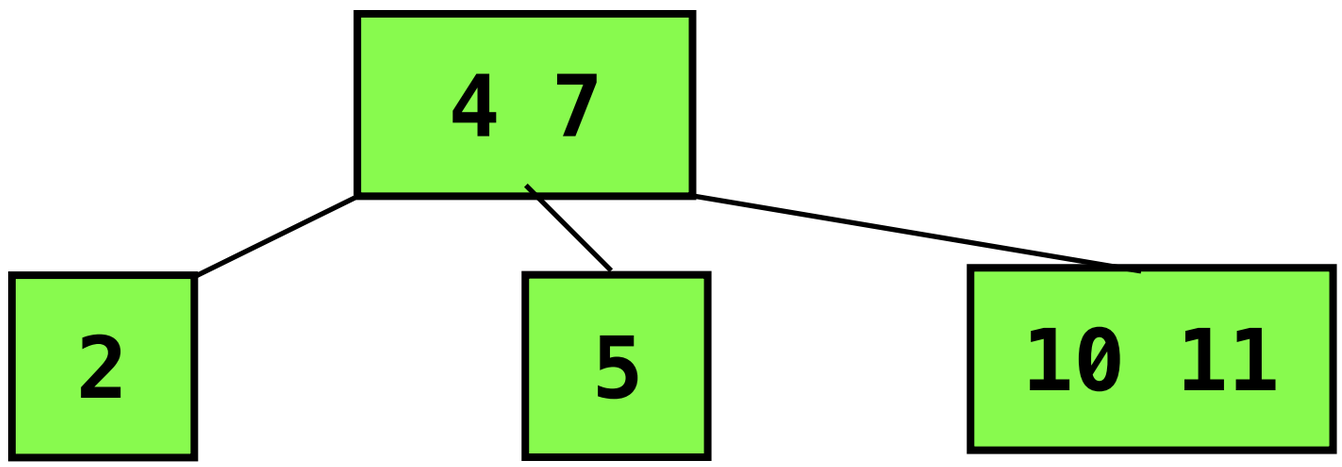





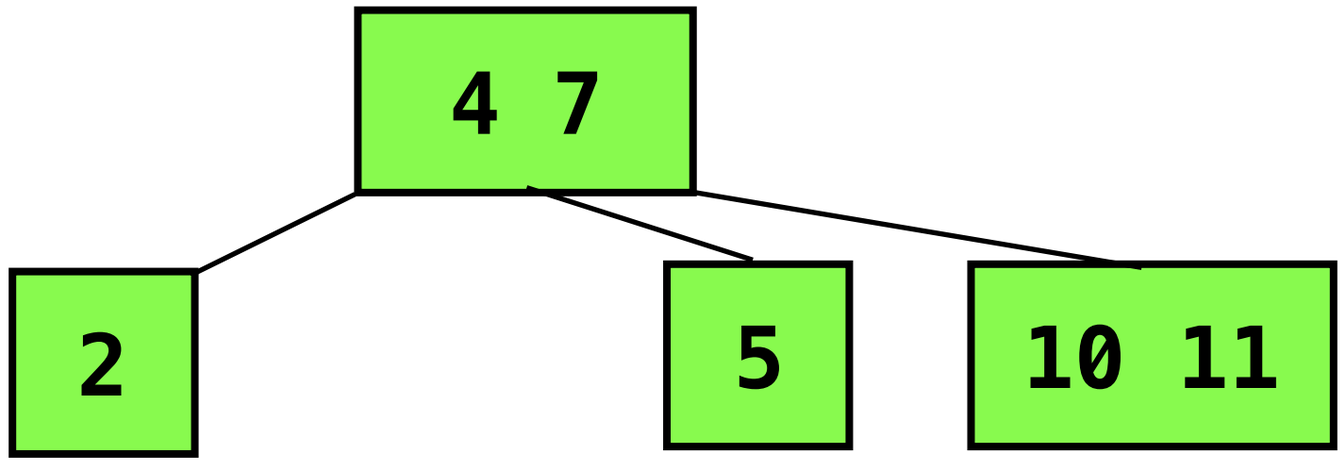
# Now we can have a balanced binary search tree!

- Just use a red-black tree, and do the equivalent operation that a 2-3-4 tree would do.
- ...But this is hard to implement. Let's simplify further by using a 2-3 tree instead.

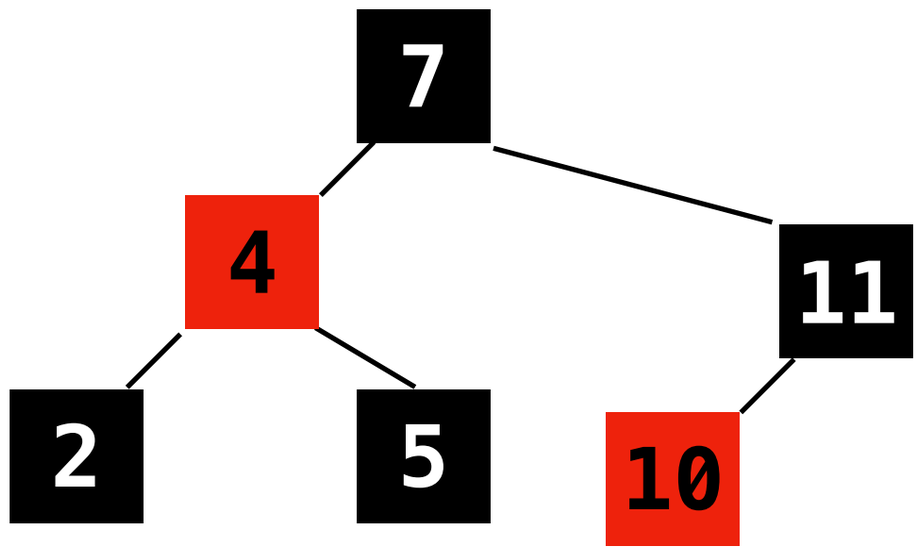
- In a 2-3 tree, a node can have one or two elements
- And each node has either 2 or 3 children.



# Like before, we can make a red-black tree with it.




**Always drop the left element down: this creates a left-leaning red-black tree.**



# Facts for Left-Leaning Red-Black Trees

- By convention, the root is always black.
- Since we always drop the left element down, a red node MUST be the left-child of a black node
- A black node CANNOT have two red children since that would correspond to a node with 3 elements