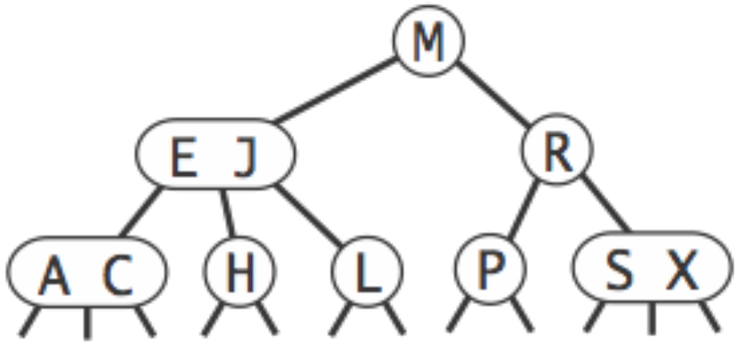
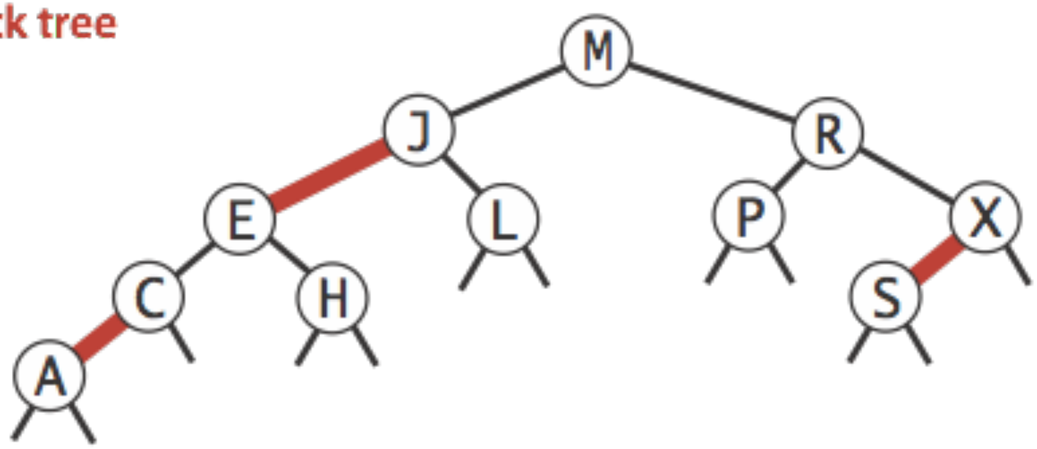
CS62 Class 17: Left Leaning Red Black Trees

Searching

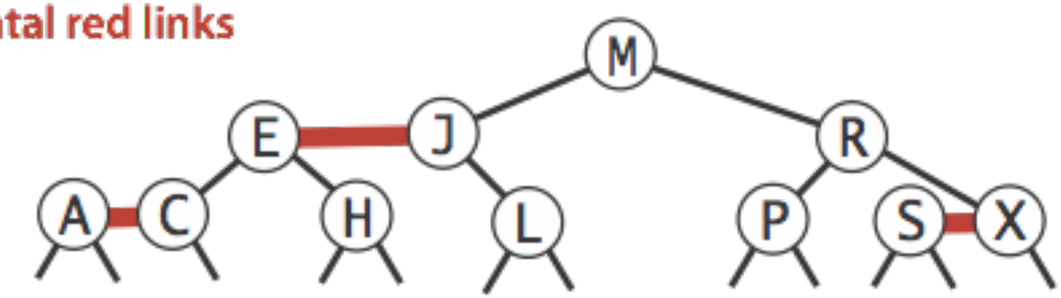
2-3 tree



red–black tree



horizontal red links



An LLRB is if you turned a 2-3 tree into a BST: if a node has multiple items, the smaller item will be the left child with a “red” link.

Agenda

- Tree rotation
- Left-leaning Red Black Trees (LLRBs)
  - 2-3 tree isometry
  - Properties, search, construction
  - Runtime analysis

The Bad News: B-Trees are ugly to implement

B-Trees for small L, e.g. 2-3 trees and 2-3-4 trees, are a real pain to implement, and suffer from performance problems. Issues include:

- Maintaining different node types.
- Interconversion of nodes between 2-nodes and 3-nodes.
- Walking up the tree to split nodes.

fantasy 2-3 code via Kevin Wayne

```
public void put(Key key, Value val) {
   Node x = root;
   while (x.getTheCorrectChildKey(key) != null) {
      x = x.getTheCorrectChildKey();
      if (x.is4Node()) { x.split(); }
   }
   if (x.is2Node()) { x.make3Node(key, val); }
   if (x.is3Node()) { x.make4Node(key, val); }
}
```

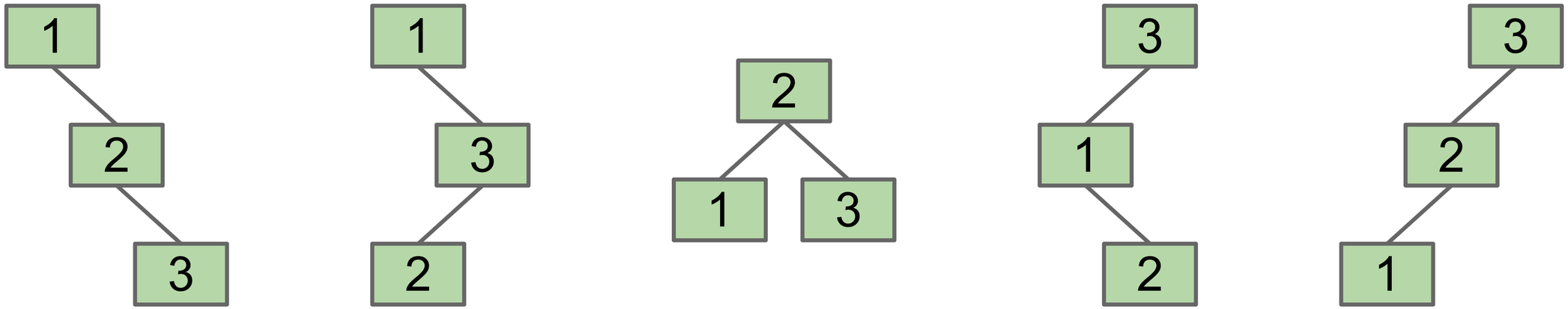
“Beautiful algorithms are, unfortunately, not always the most useful.” - Don Knuth

Tree Rotation

Back to BSTs...

Suppose we have a BST with the numbers 1, 2, 3. There are five possible BSTs.

- The specific BST you get is based on the insertion order.
- More generally, for N items, there are Catalan(N) different BSTs.



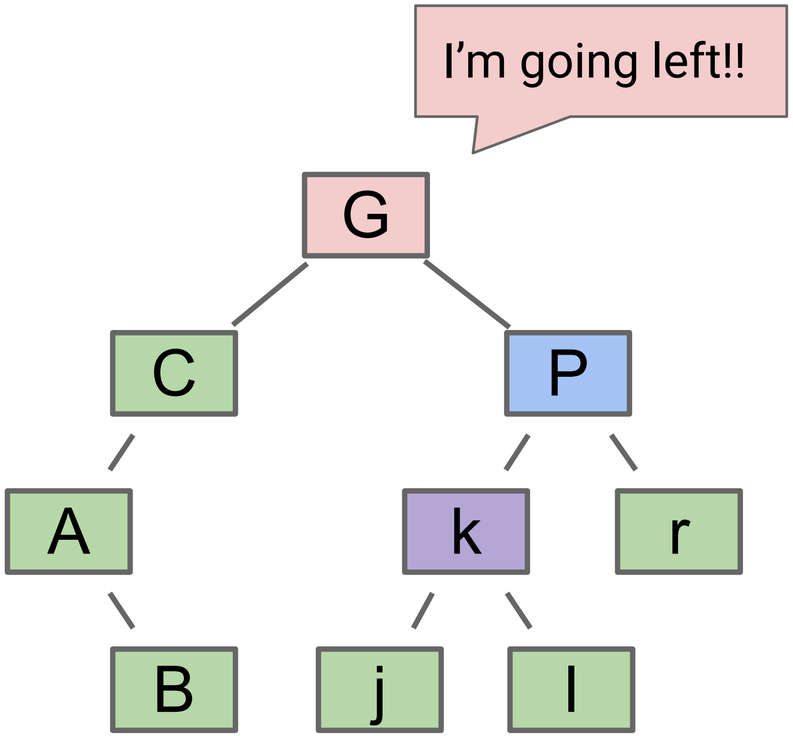
Given any BST, it is possible to move to a different configuration using “rotation”.

- In general, can move from any configuration to any other in 2n - 6 rotations (see Rotation Distance, Triangulations, and Hyperbolic Geometry or Amy Liu).

Tree Rotation Definition (Demo)

rotateLeft(G): Let x be the right child of G. Make G the **new left child** of x.

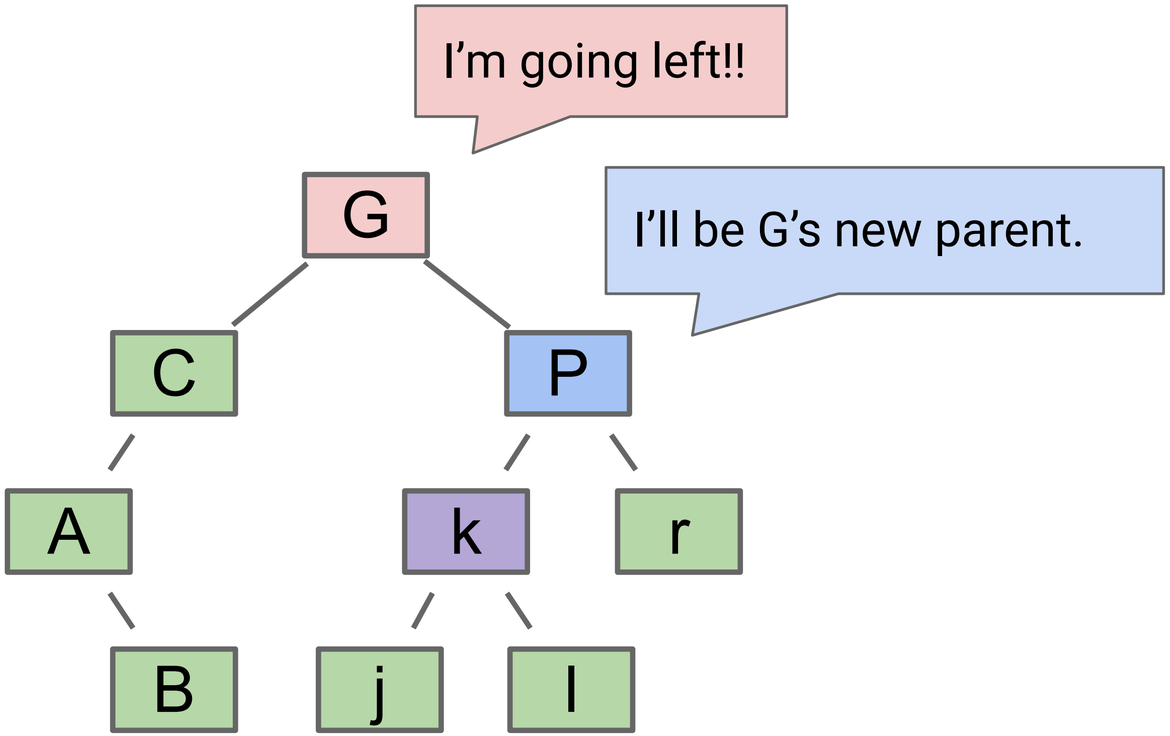
- Preserves search tree property. No change to semantics of tree.



Tree Rotation Definition (Demo)

rotateLeft(G): Let x be the right child of G. Make G the **new left child** of x.

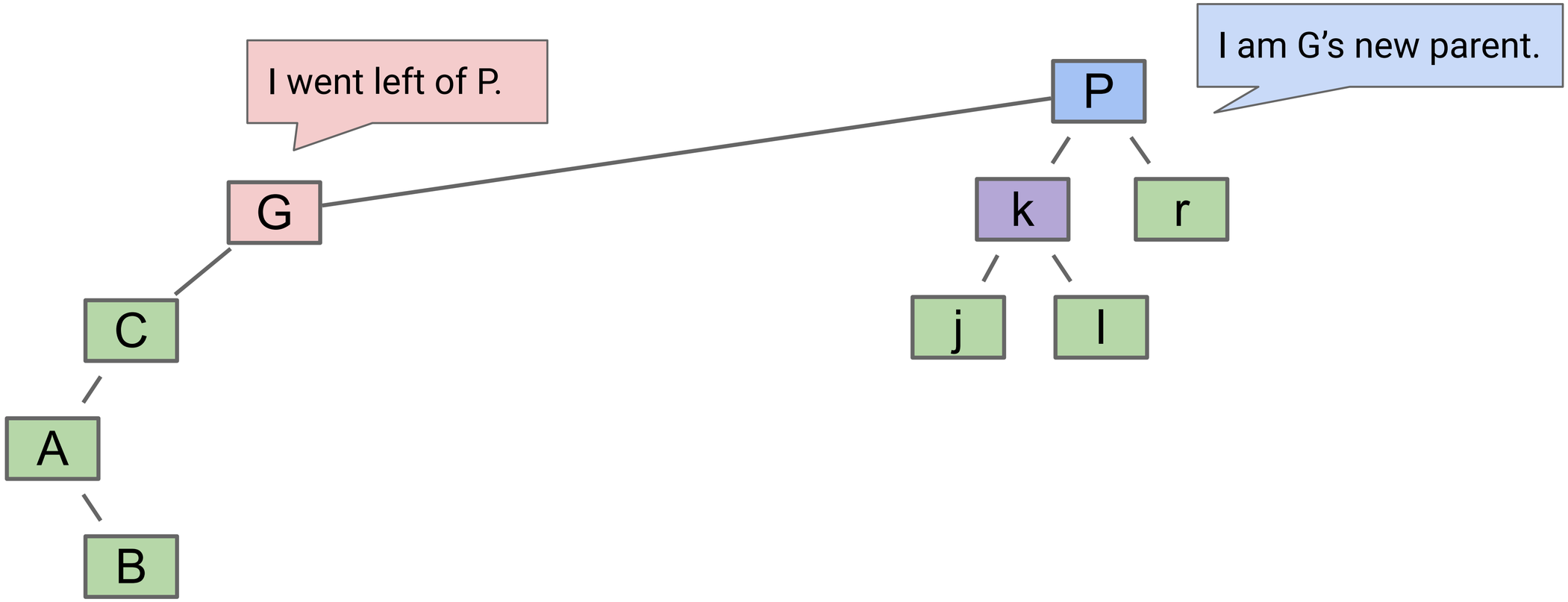
- Preserves search tree property. No change to semantics of tree.



Tree Rotation Definition (Demo)

rotateLeft(G): Let x be the right child of G. Make G the **new left child** of x.

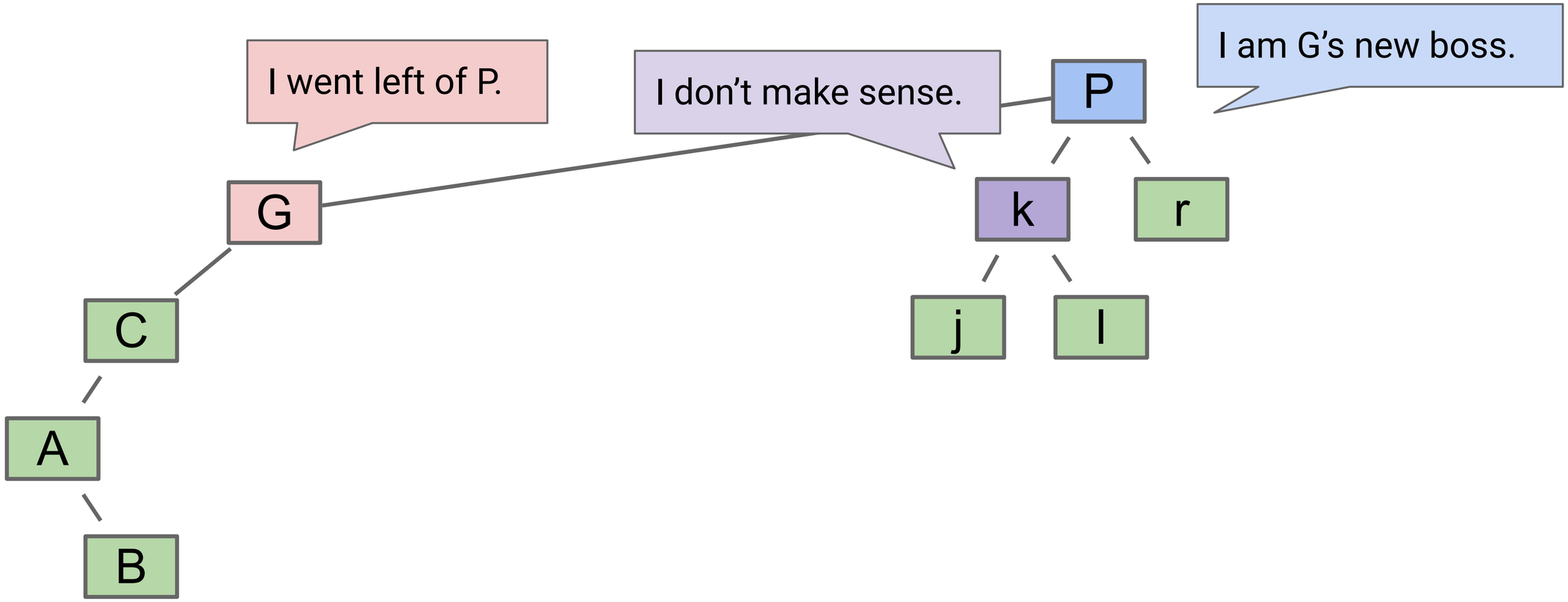
- Preserves search tree property. No change to semantics of tree.



Tree Rotation Definition (Demo)

rotateLeft(G): Let x be the right child of G. Make G the **new left child** of x.

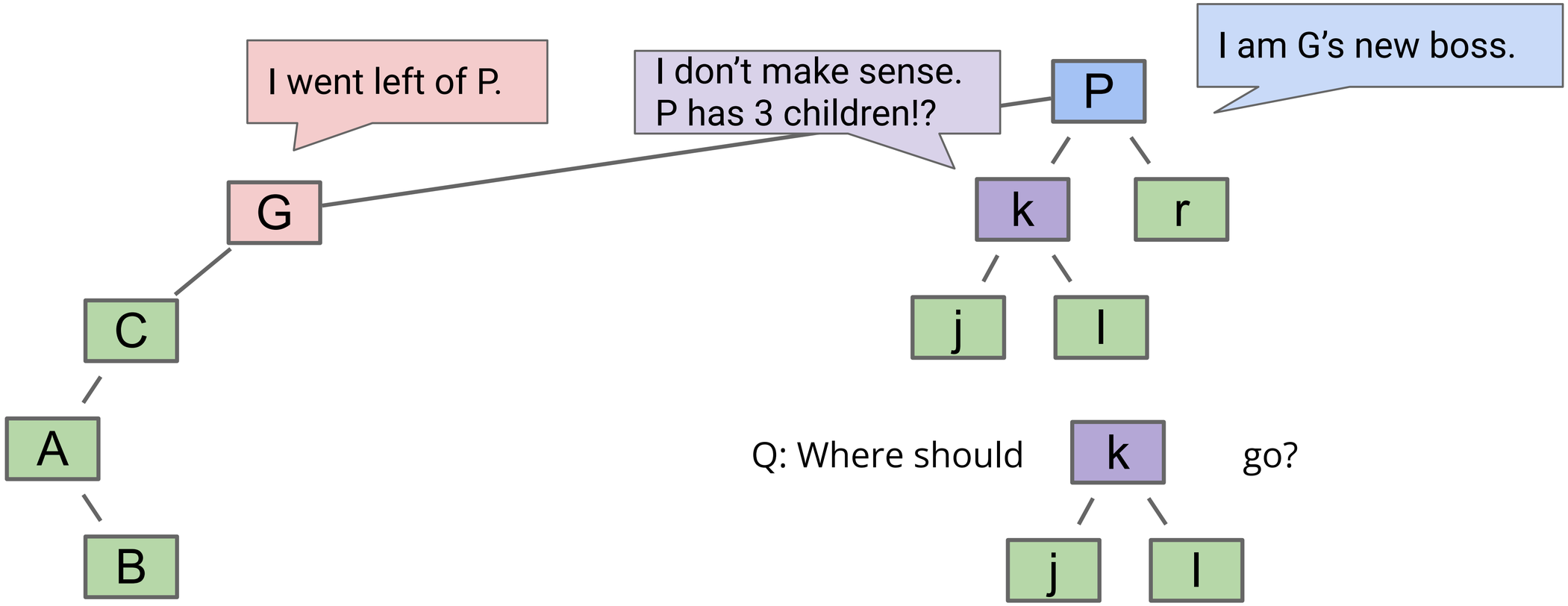
- Preserves search tree property. No change to semantics of tree.



Tree Rotation Definition (Demo)

rotateLeft(G): Let x be the right child of G. Make G the **new left child** of x.

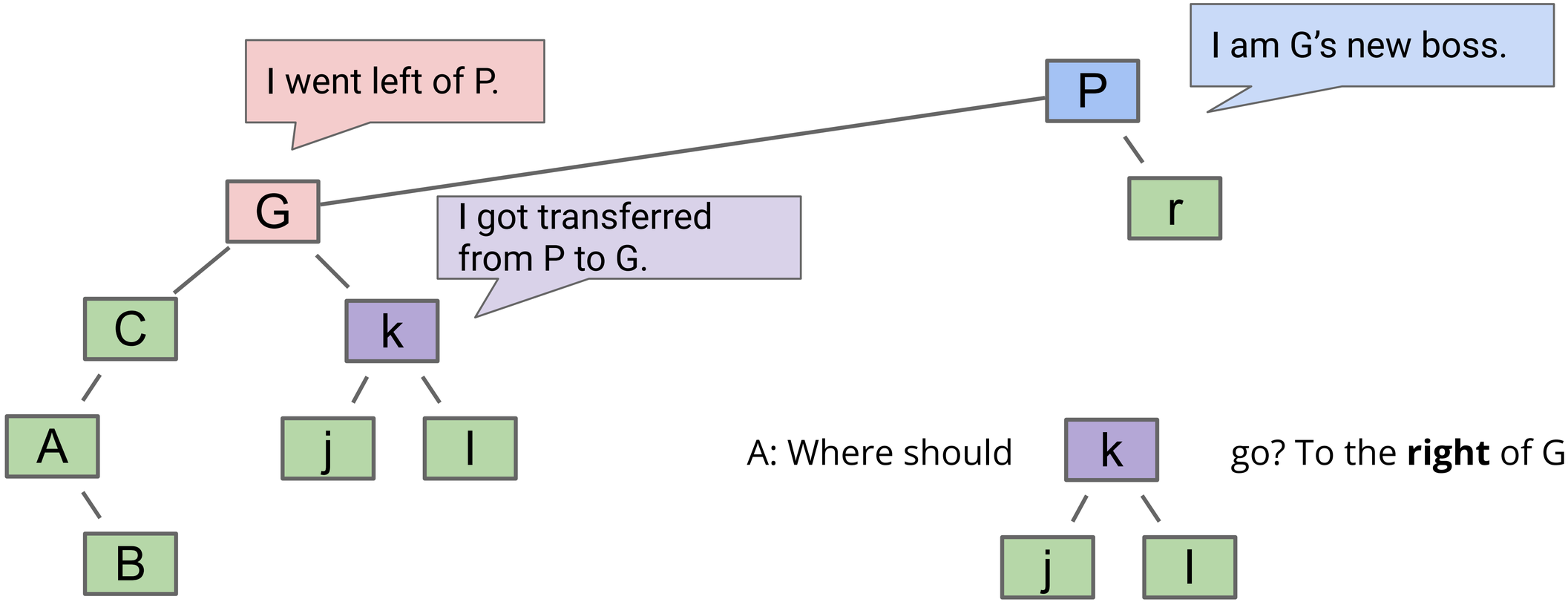
- Preserves search tree property. No change to semantics of tree.



Tree Rotation Definition (Demo)

rotateLeft(G): Let x be the right child of G. Make G the **new left child** of x.

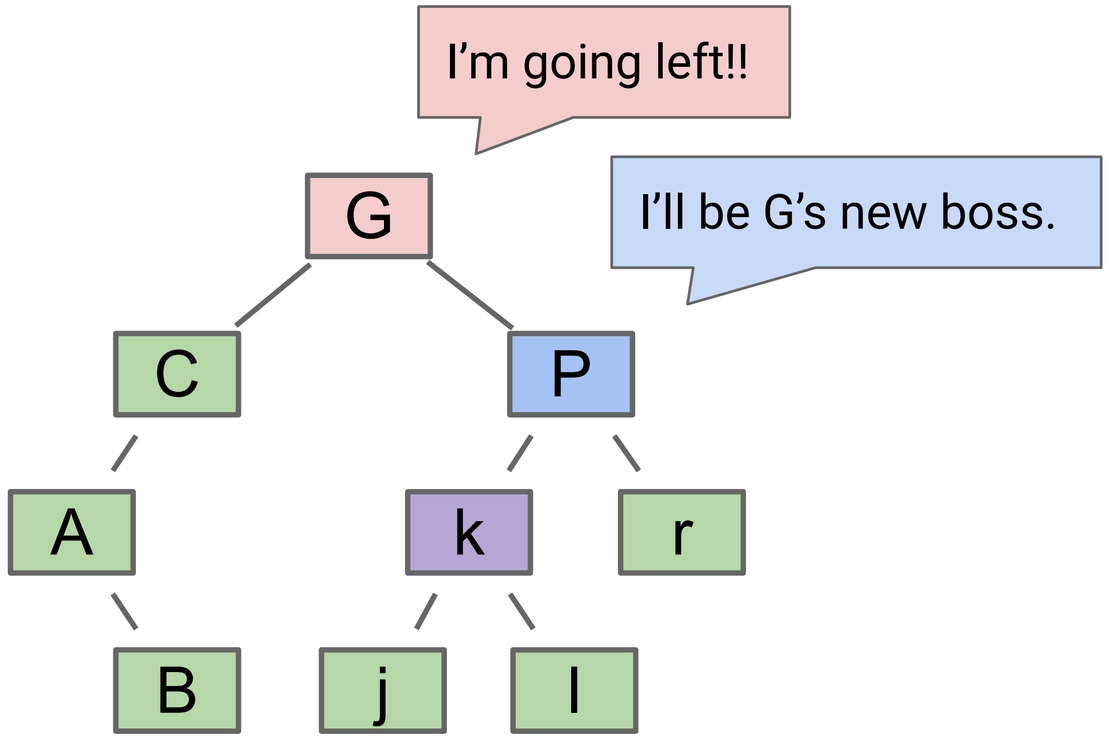
- Preserves search tree property. No change to semantics of tree.

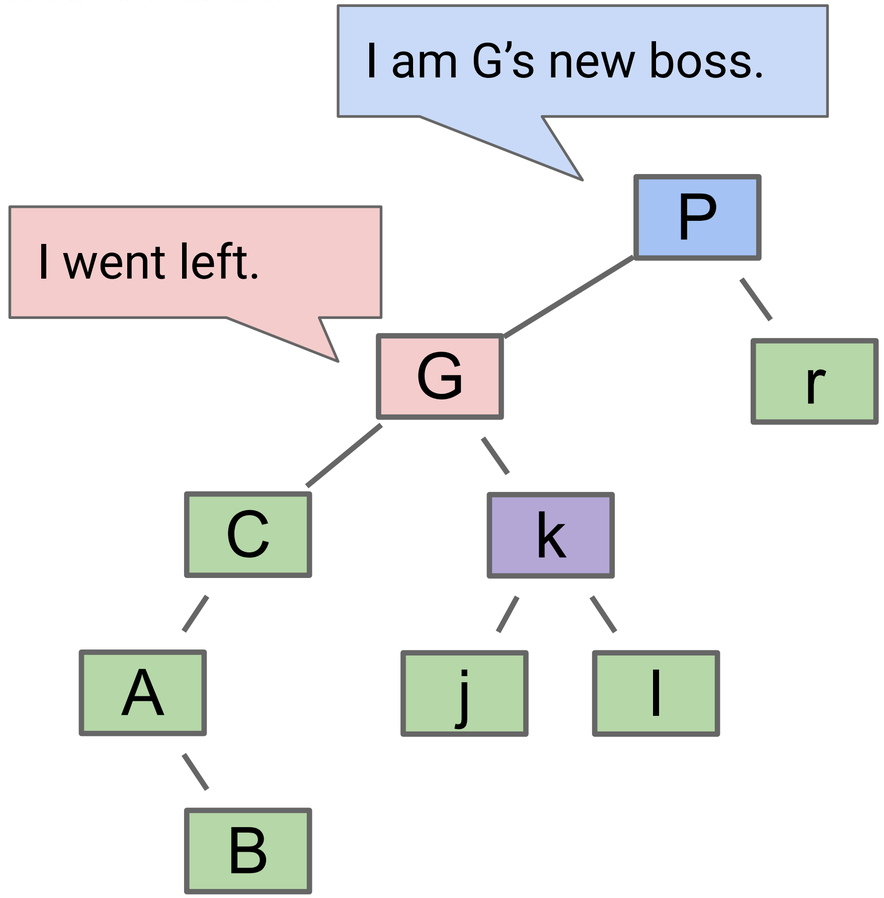


Tree Rotation Definition (All in One Slide)

rotateLeft(G): Let x be the right child of G. Make G the **new left child** of x.

- Preserves search tree property. No change to semantics of tree.





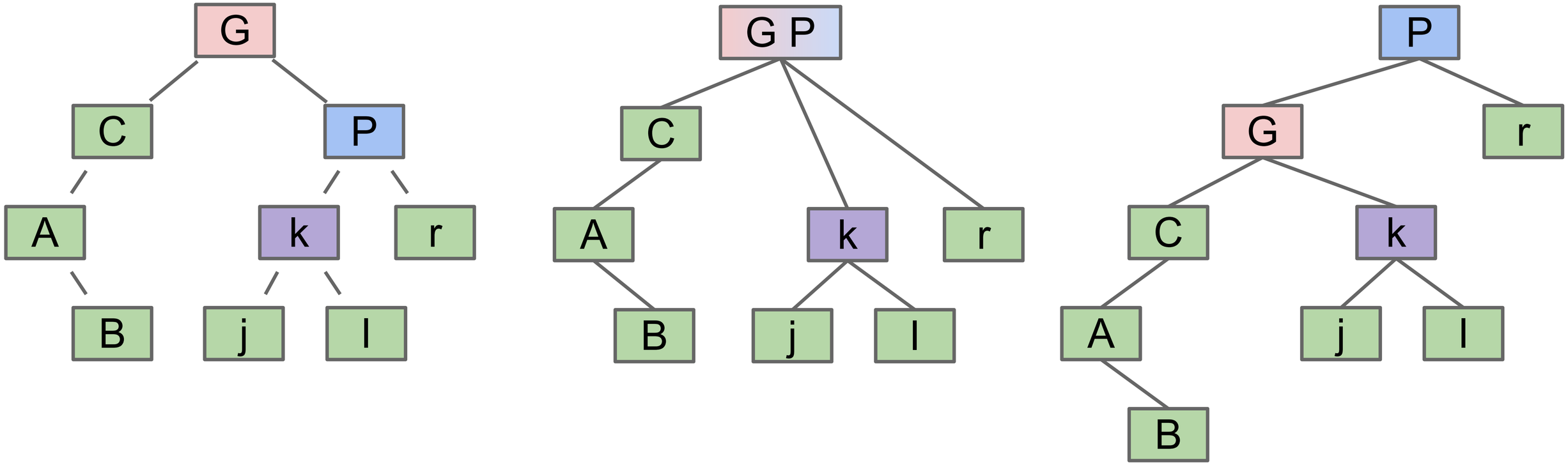
Rotation makes the node to be rotated a **child**

For this example rotateLeft(G) increased height of tree!

Tree Rotation Definition (Alternate Interpretation)

rotateLeft(G): Let x be the right child of G. Make G the **new left child** of x.

- Can think of as temporarily merging G and P, then sending G down and **left**.
- Preserves search tree property. No change to semantics of tree.



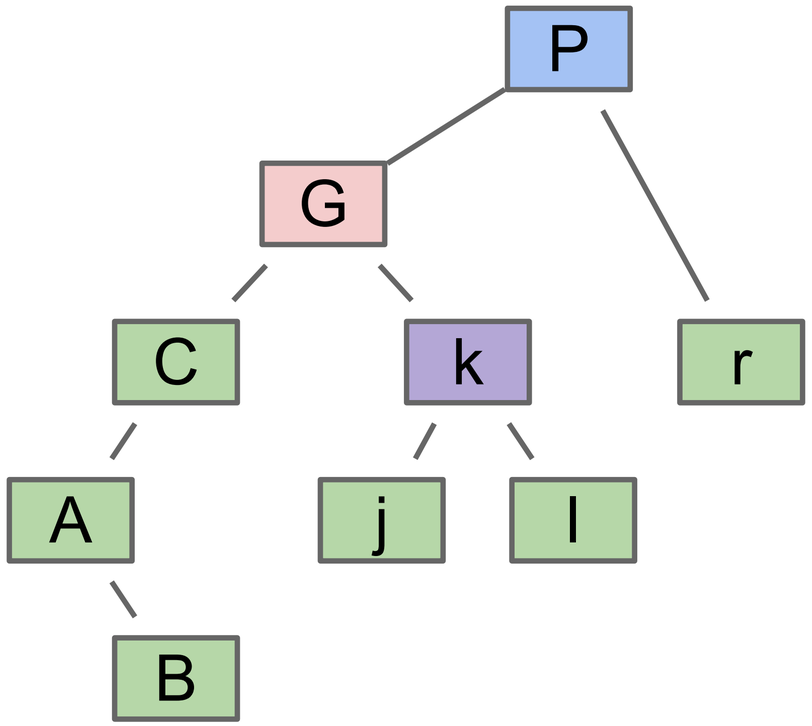
Rotation makes the node to be rotated a **child**

For this example rotateLeft(G) increased height of tree

Worksheet time!

rotateRight(P): Let x be the left child of P. Make P the **new right child** of x.

- Can think of as temporarily merging G and P, then sending P down and **right**.

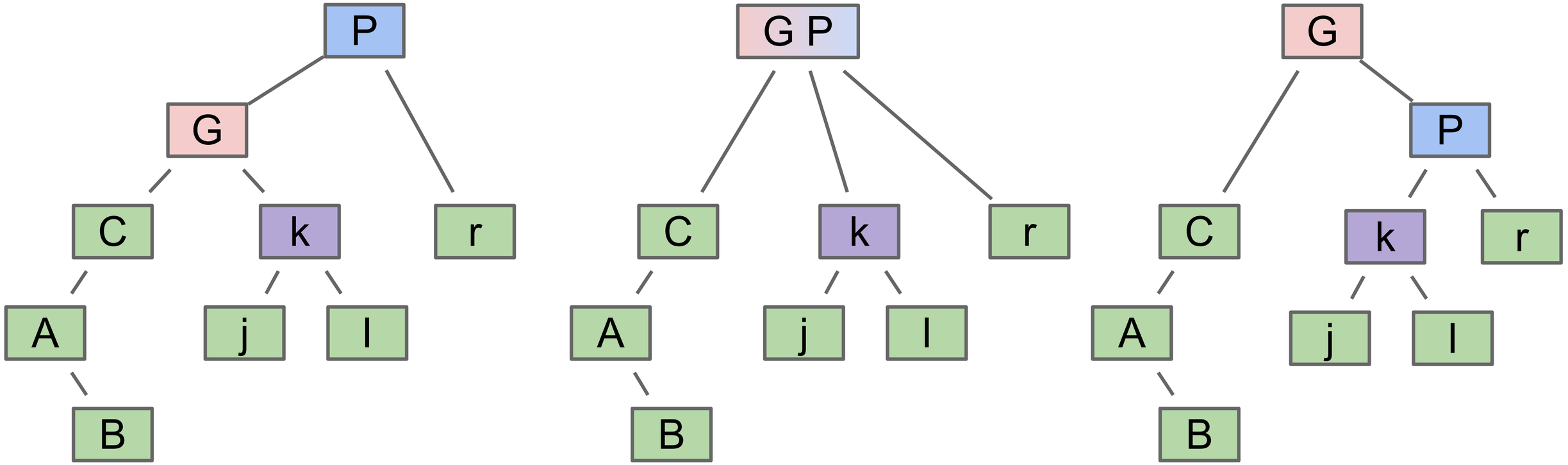


What does the final tree look like after calling rotateRight(P)?

Worksheet answers

rotateRight(P): Let x be the left child of P. Make P the **new right child** of x.

- Can think of as temporarily merging G and P, then sending P down and **right**.
- Note: k was G's right child. Now it is P's left child.



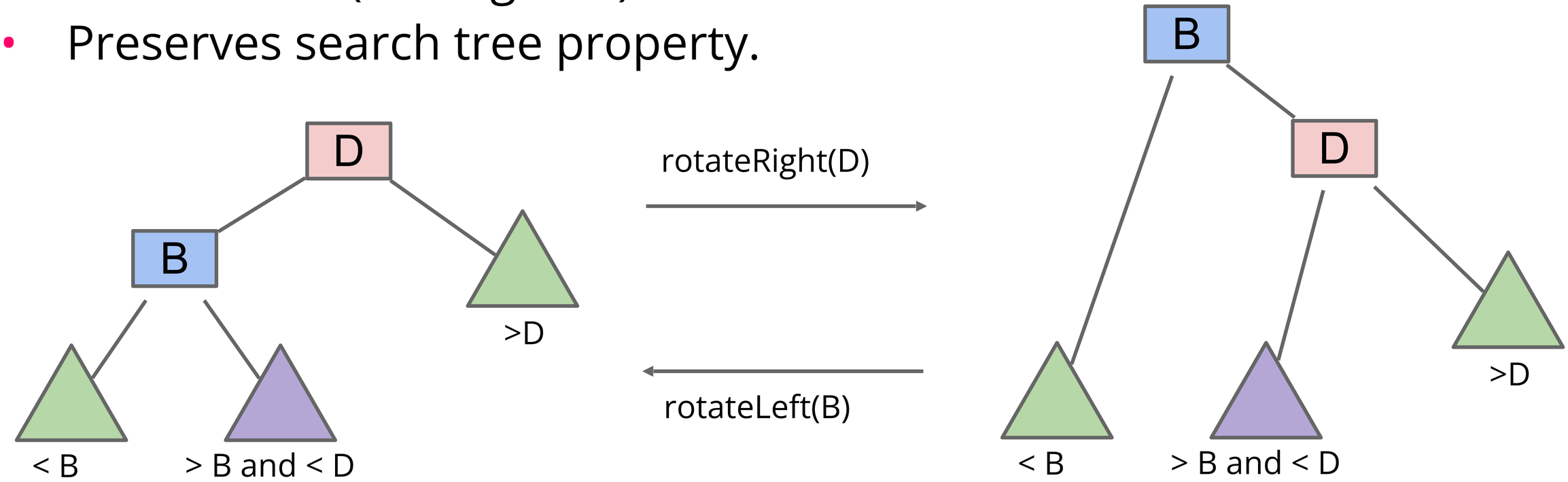
For this example rotateRight(P) decreased height of tree!

Tree Balancing

Rotation for Balance

Rotation:

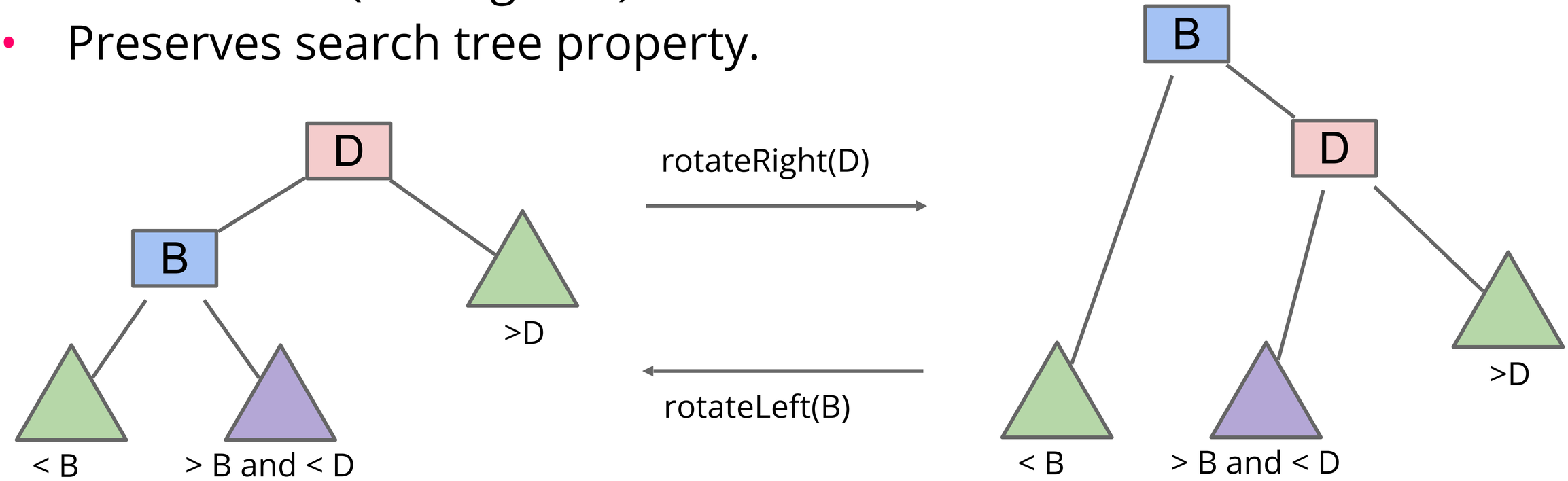
- Can shorten (or lengthen) a tree.
- Preserves search tree property.



Rotation for Balance

Rotation:

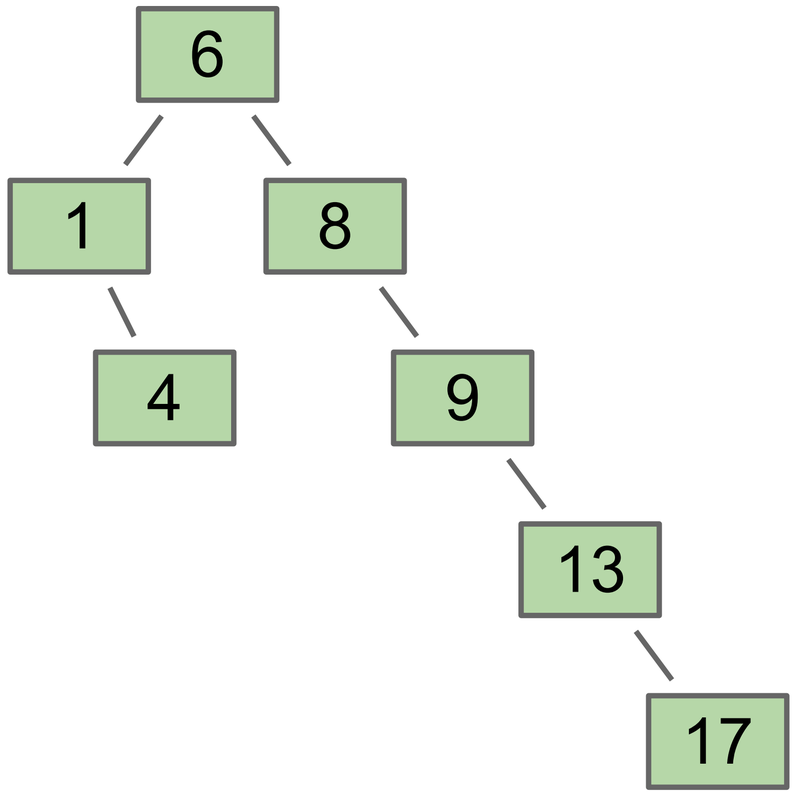
- Can shorten (or lengthen) a tree.
- Preserves search tree property.



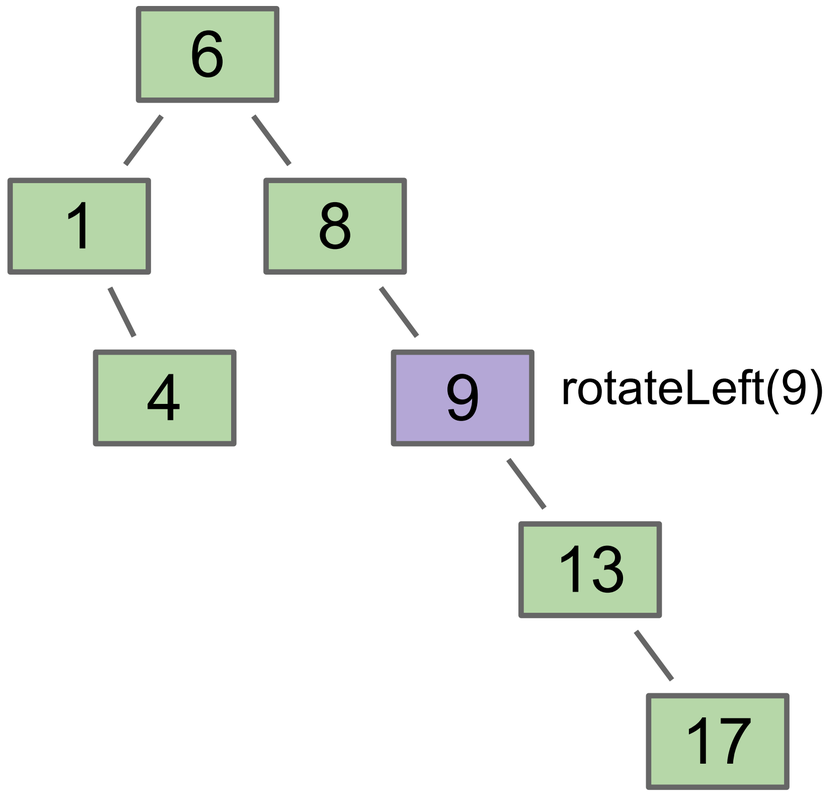
Can use rotation to balance a BST.

- Rotation allows balancing of a BST in O(N) moves.

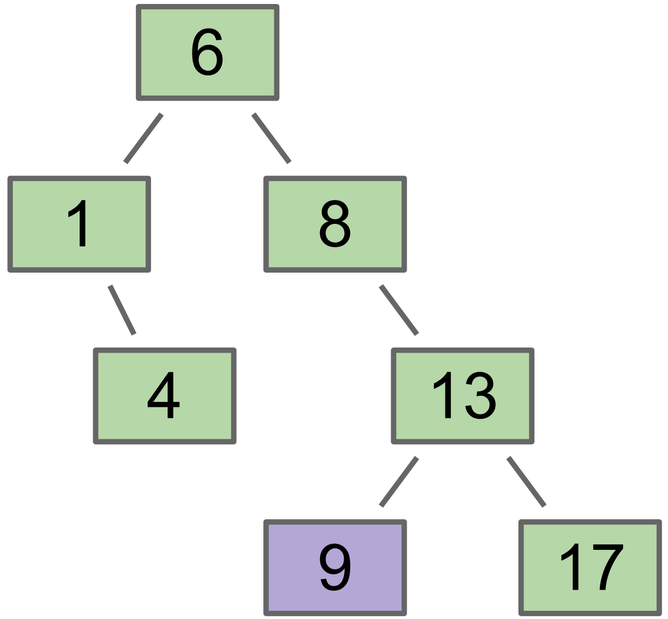
Demo: Balancing with Tree Rotation



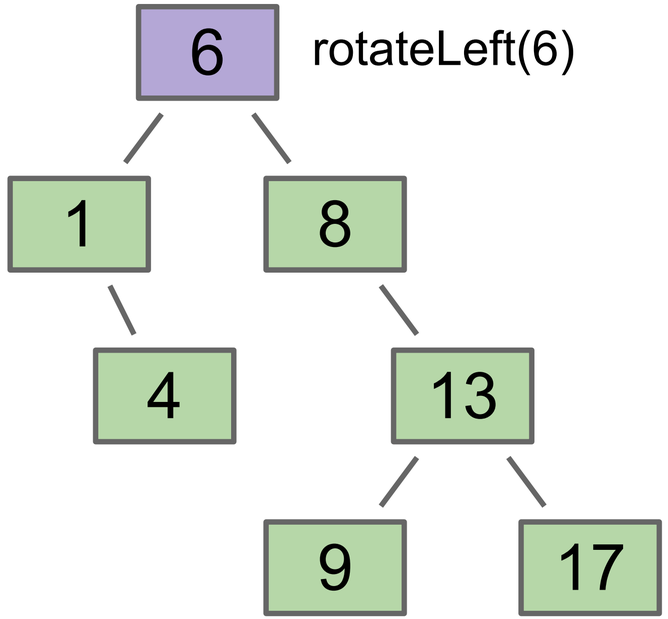
Demo: Balancing with Tree Rotation



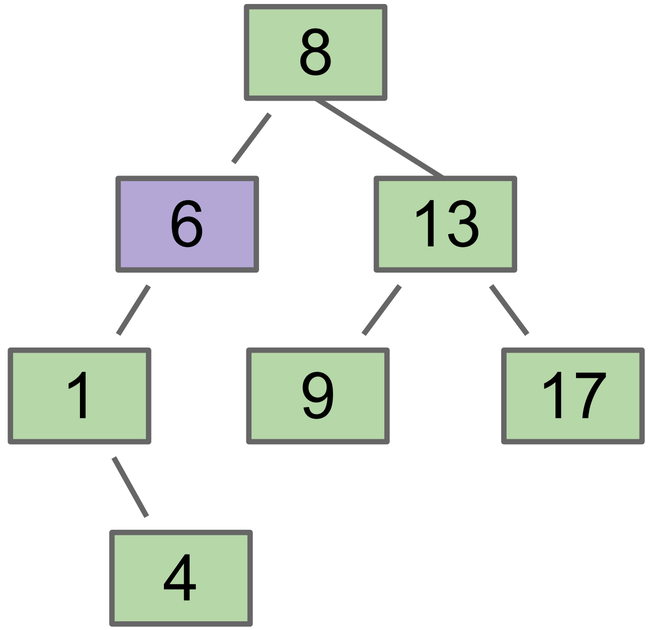
Demo: Balancing with Tree Rotation



Demo: Balancing with Tree Rotation

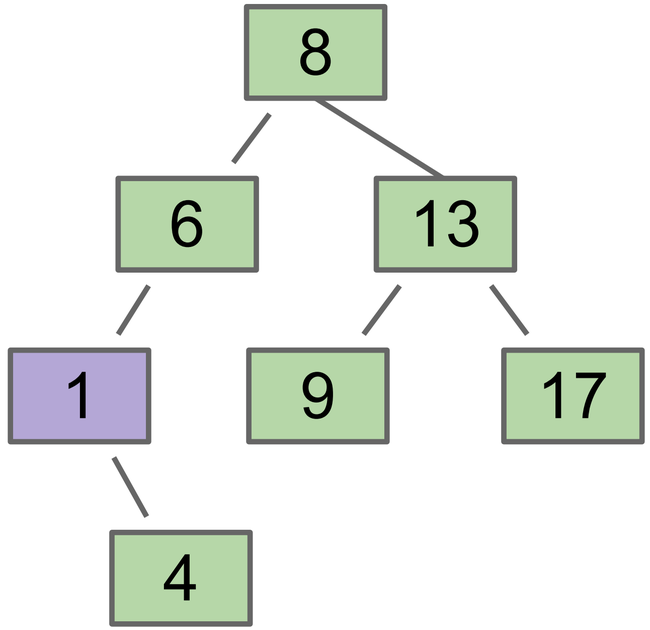


Demo: Balancing with Tree Rotation

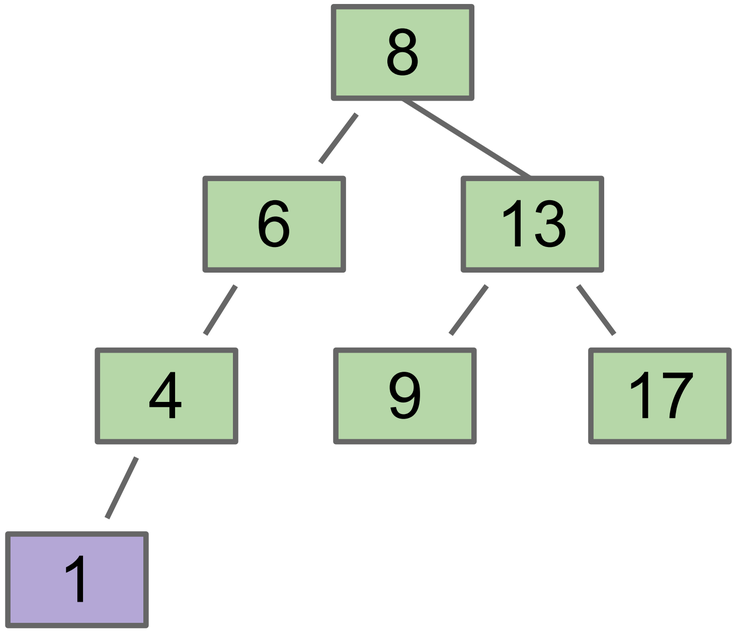


Demo: Balancing with Tree Rotation

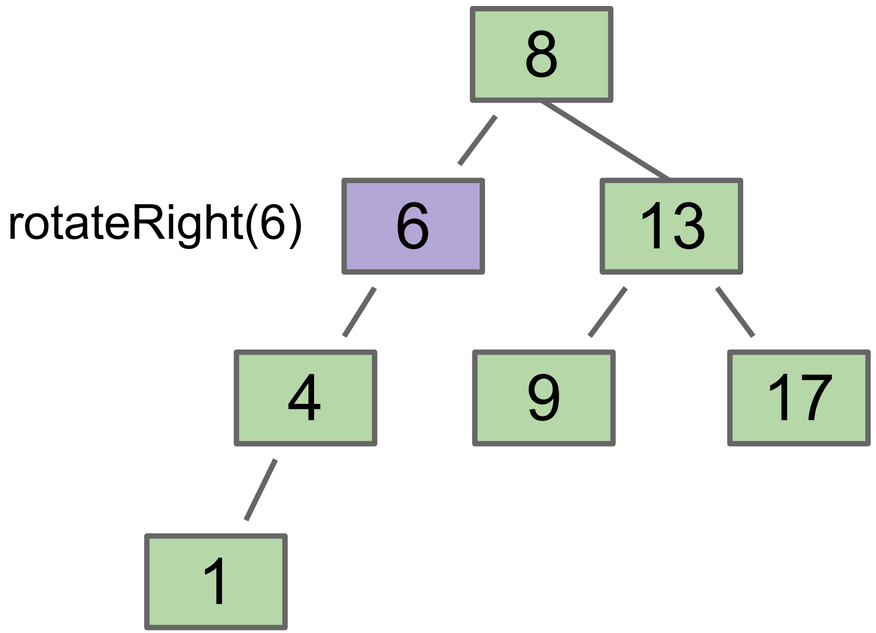
rotateLeft(1)



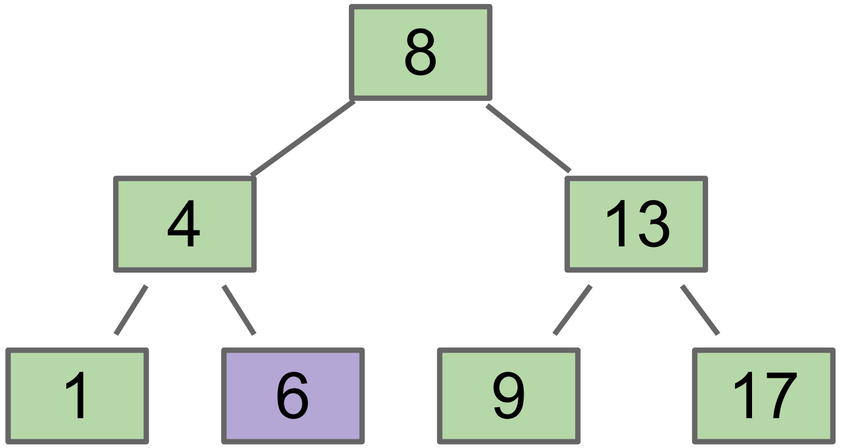
Demo: Balancing with Tree Rotation



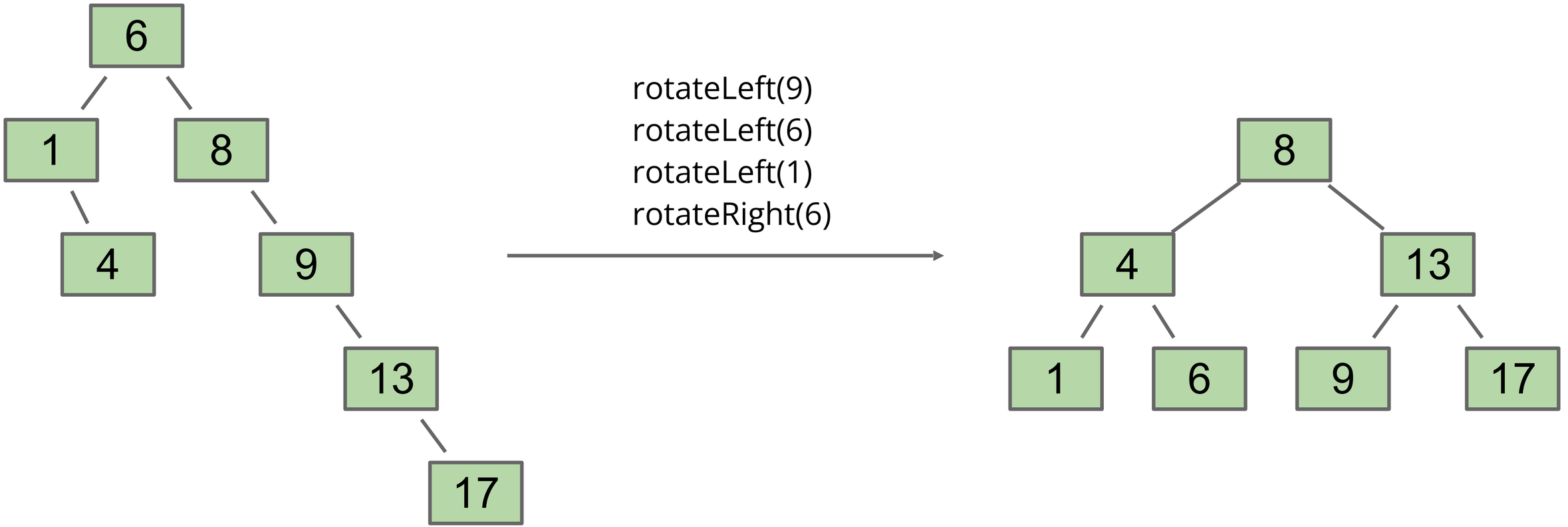
Demo: Balancing with Tree Rotation



Demo: Balancing with Tree Rotation



Demo: Balancing with Tree Rotation



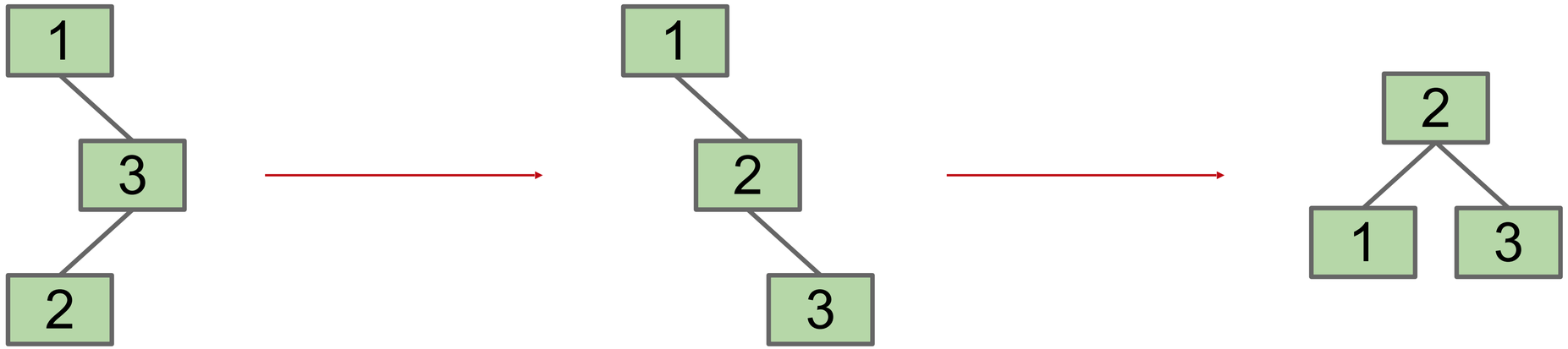
Worksheet time!

Give a sequence of rotation operations that balances the tree on the left.

Worksheet answers

Give a sequence of rotation operations that balances the tree on the left.

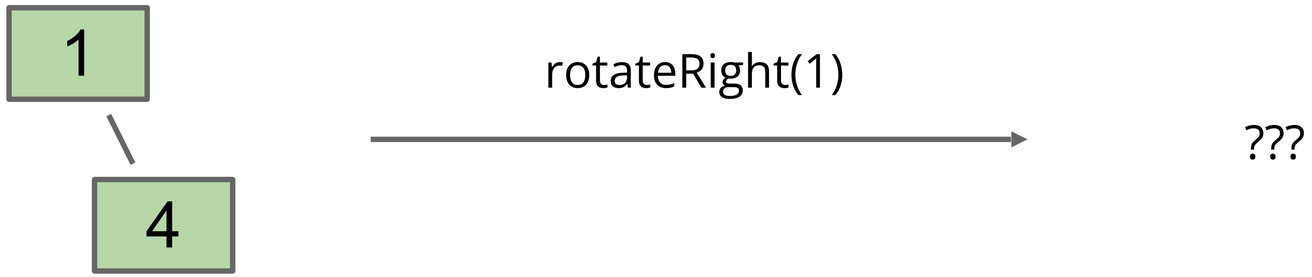
- rotateRight(3)
- rotateLeft(1)



There are other correct answers as well!

Some Rotations are Undefined

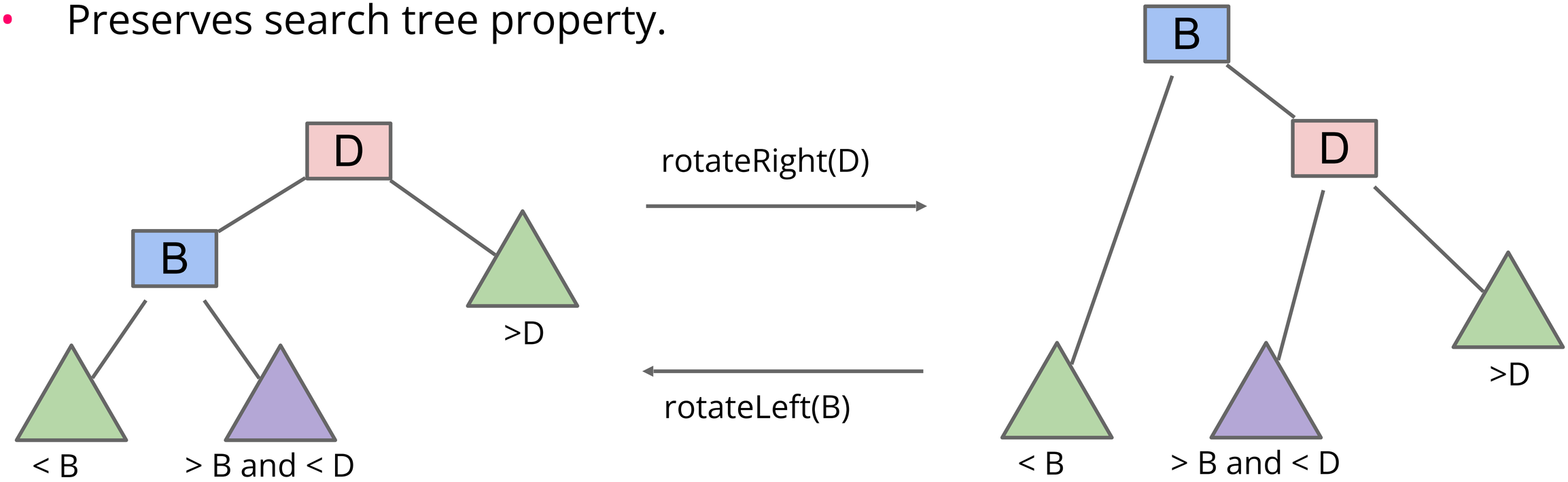
- Rotating a node right is undefined if that node has no left child.
  - We would need to promote that node's left child, but it doesn't exist.
- Rotating a node left is undefined if that node has no right child.
- We won't need to perform any undefined rotations in this lecture, so don't worry about them.



Rotation: An Alternate Approach to Balance

Rotation:

- Can shorten (or lengthen) a tree.
- Preserves search tree property.



Paying O(n) to occasionally balance a BST is not ideal. In this lecture, we'll see a better way to achieve balance through rotation: Left-leaning red black trees (LLRB)s.

LLRBs:
isometry with 2-3 trees

Search Trees

There are many types of search trees:

- **Binary search trees**: Can balance using rotation, but we have no algorithm for doing so (yet).
- **2-3 trees**: Balanced by construction, i.e. no rotations required.

Let’s try something clever, but strange.

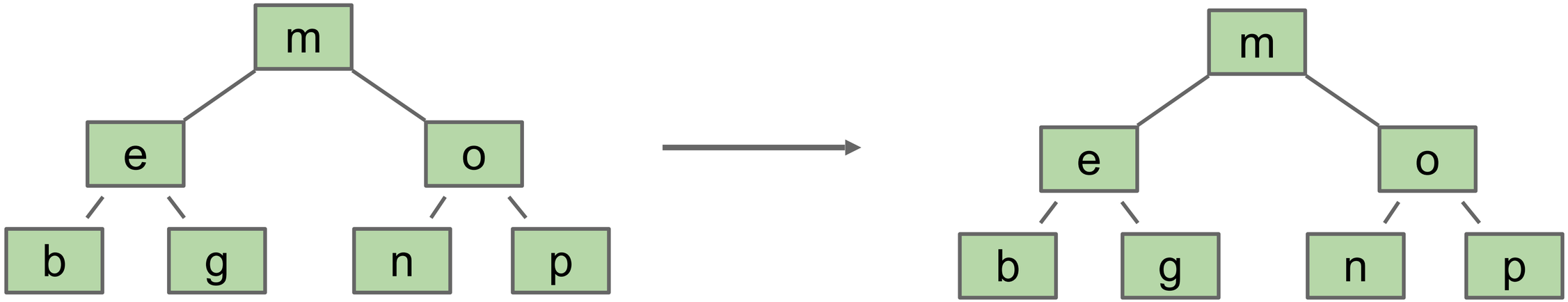
Our goal: Build a BST that is structurally identical to a 2-3 tree.

- Since 2-3 trees are balanced, so will our special BSTs.
- With a BST, it’s easier to implement, since we only have 1 type of node with left/right links.

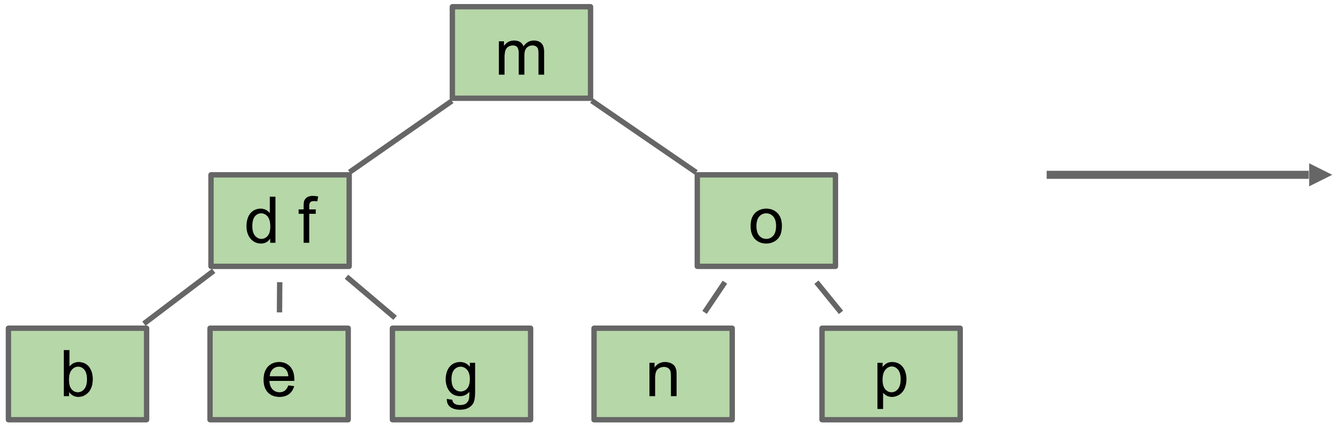
Representing a 2-3 Tree as a BST

A 2-3 tree with only 2-nodes requires no special work.

- BST is exactly the same!



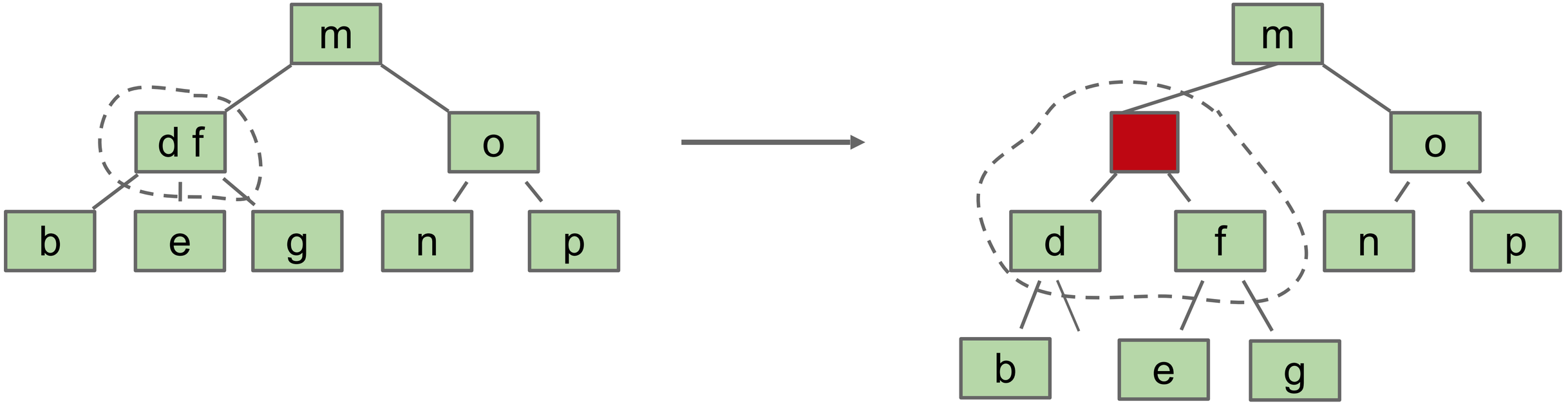
What do we do about 3-nodes?



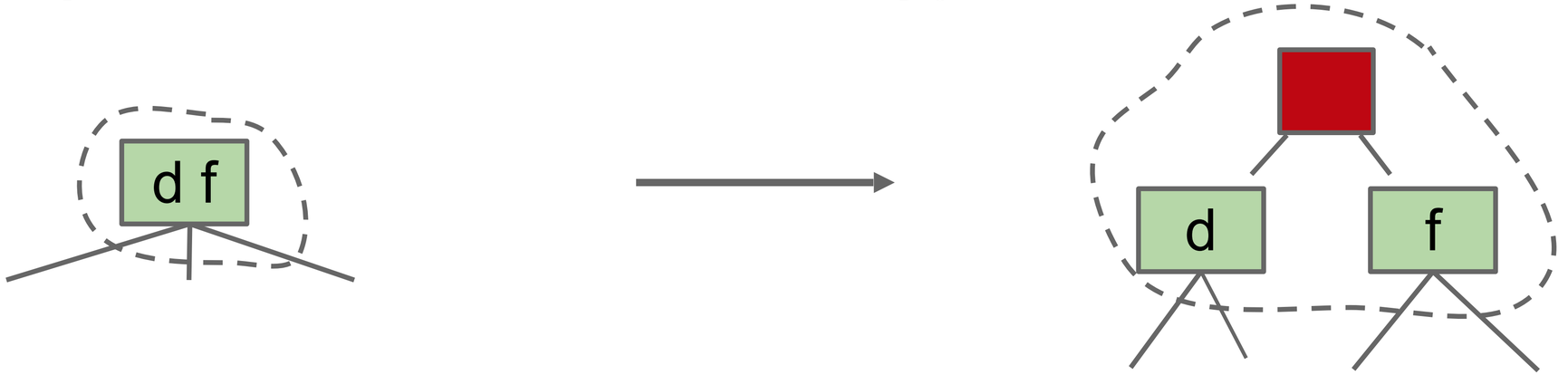
????

Representing a 2-3 Tree as a BST: Dealing with 3-Nodes

Possibility 1: Create dummy “glue” nodes.

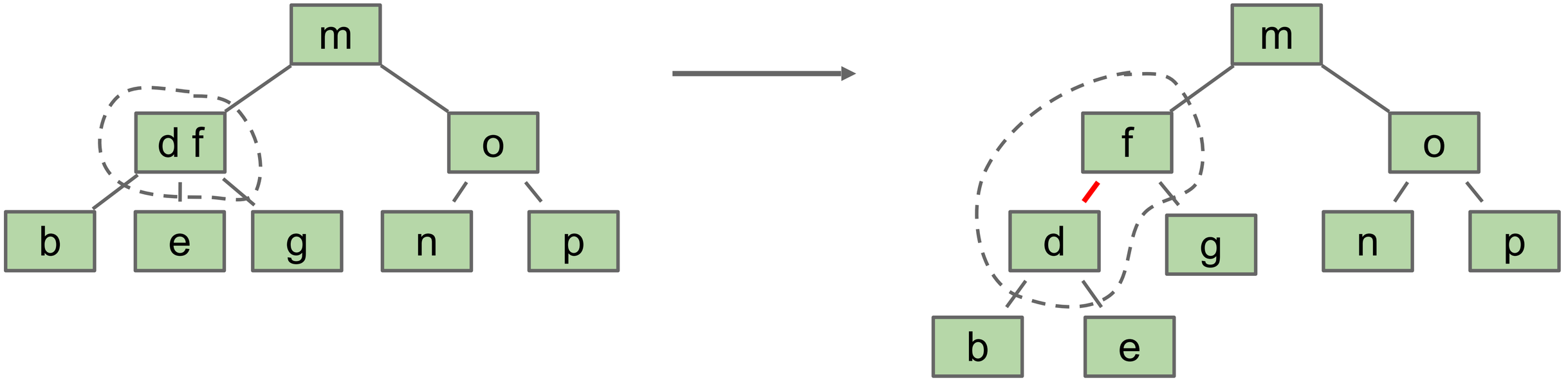


Result is inelegant. Wasted link. Code will be ugly.

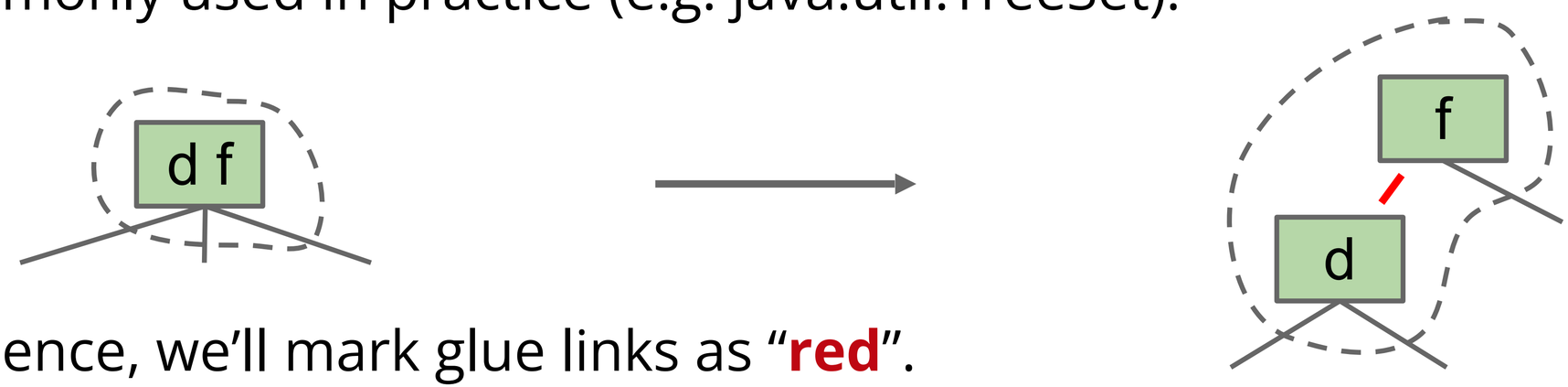


Representing a 2-3 Tree as a BST: Dealing with 3-Nodes

Possibility 2: Create “glue” **links** with the smaller item **off to the left**.



Idea is commonly used in practice (e.g. java.util.TreeSet).

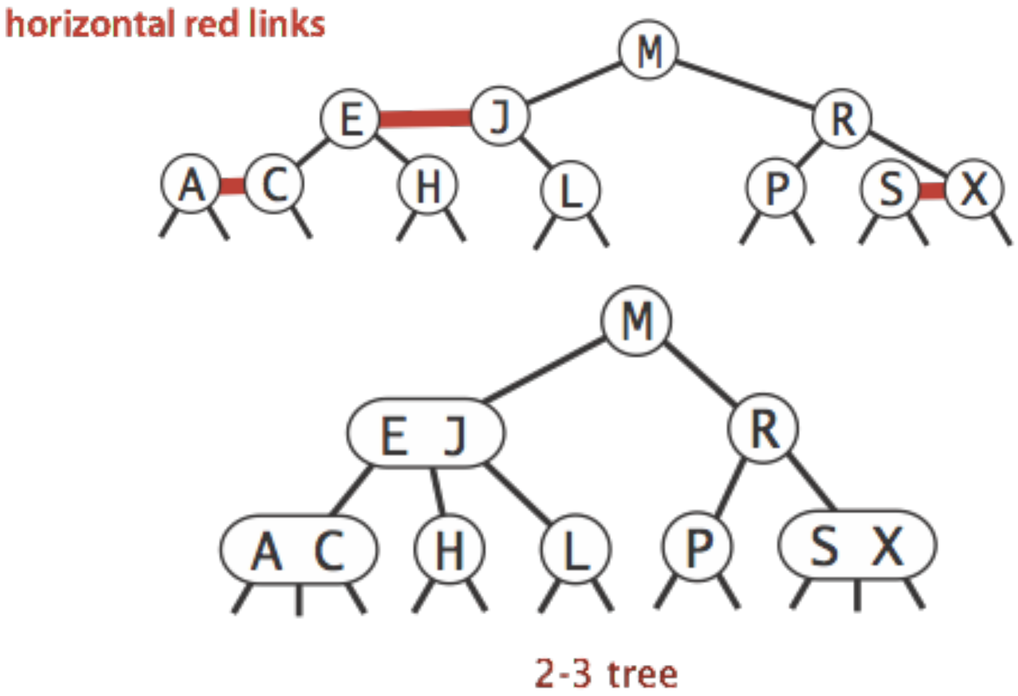


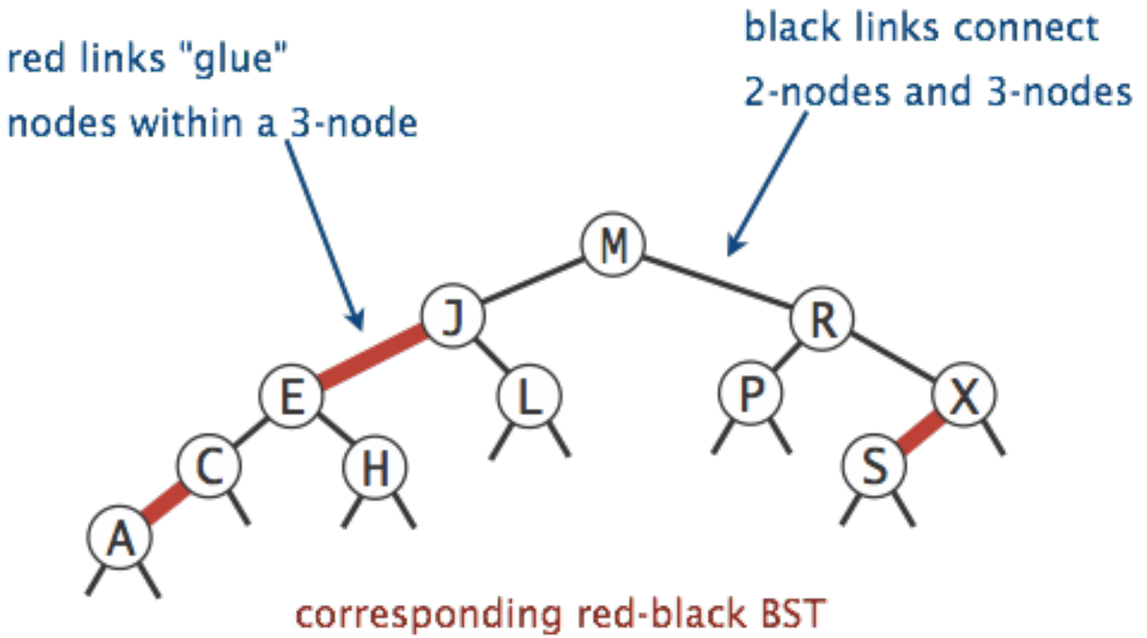
For convenience, we’ll mark glue links as “**red**”.

Left-Leaning Red Black Binary Search Tree (LLRB)

A BST with left glue links that represents a 2-3 tree is often called a “Left Leaning Red Black Binary Search Tree” or LLRB.

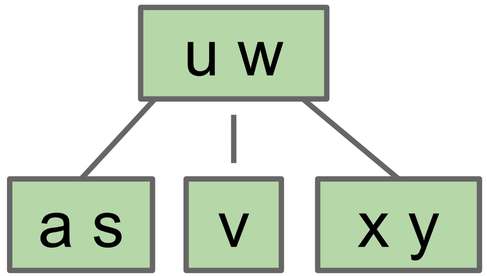
- LLRBs are normal BSTs!
- There is a 1-1 correspondence between an LLRB and an equivalent 2-3 tree.
- The red is just a convenient fiction. Red links don’t “do” anything special.





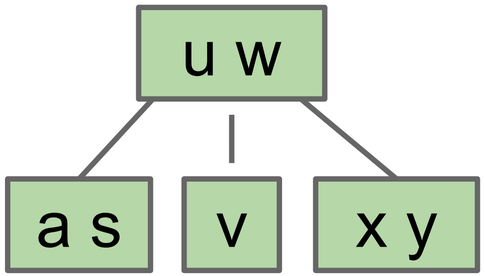
Worksheet time!

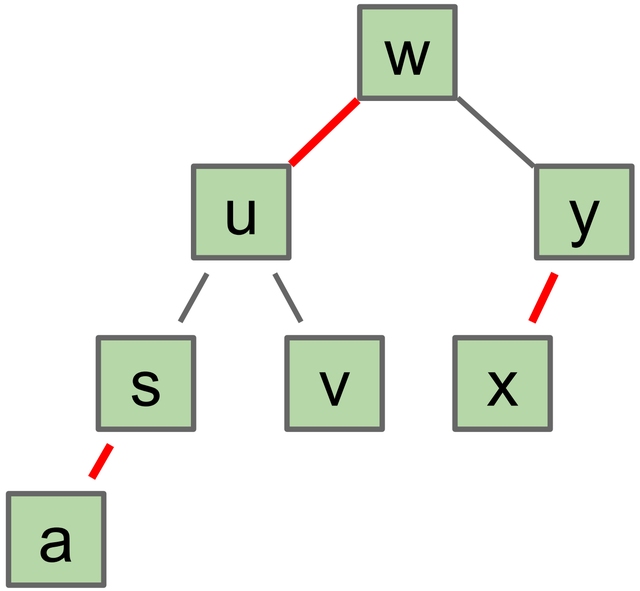
Draw the LLRB corresponding to the 2-3 tree shown below.



Worksheet answers

Draw the LLRB corresponding to the 2-3 tree shown below.

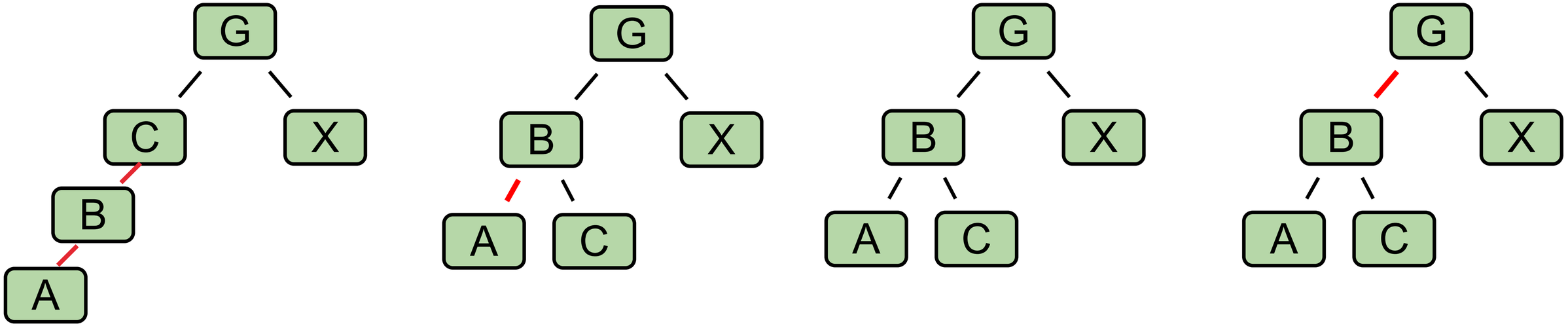




LLRBs: Properties

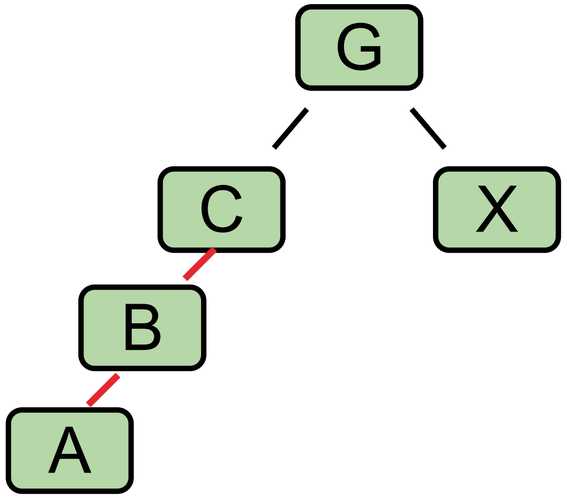
Valid LLRBs?

How many of these are valid LLRBs, i.e. have a 1-1 correspondence with a valid 2-3 tree? Talk with your neighbor. (Hint: draw them as 2-3 trees)

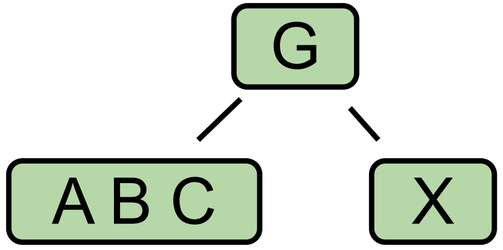


Valid LLRBs?

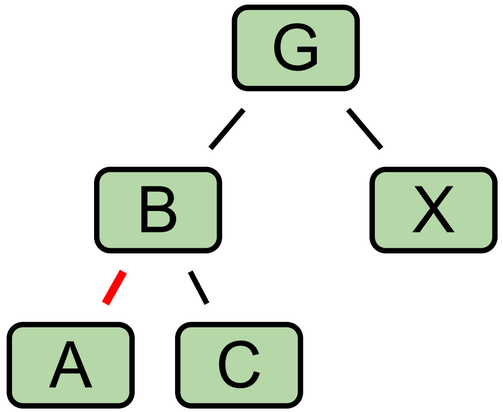
How many of these are valid LLRBs, i.e. have a 1-1 correspondence with a valid 2-3 tree? Talk with your neighbor. (Hint: draw them as 2-3 trees)

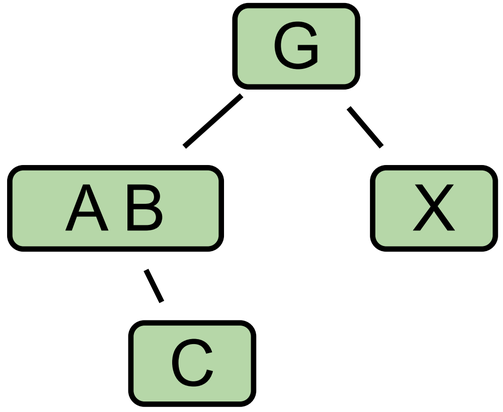


2-3 tree equivalent

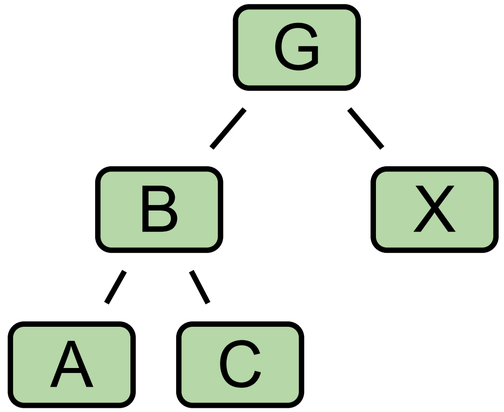


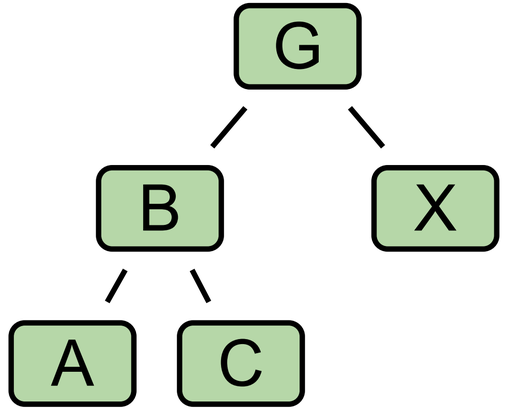
Invalid, has 4 node.



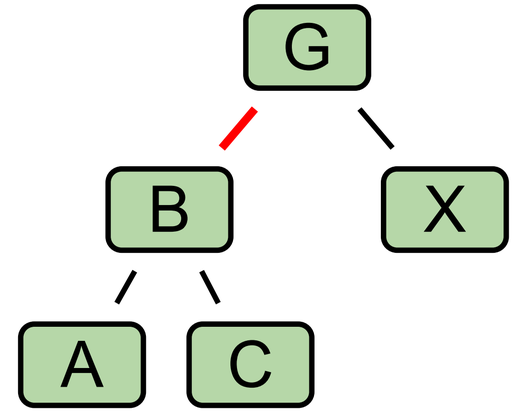


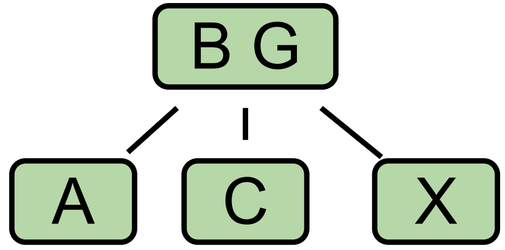
Invalid, not balanced.





Invalid, not balanced.

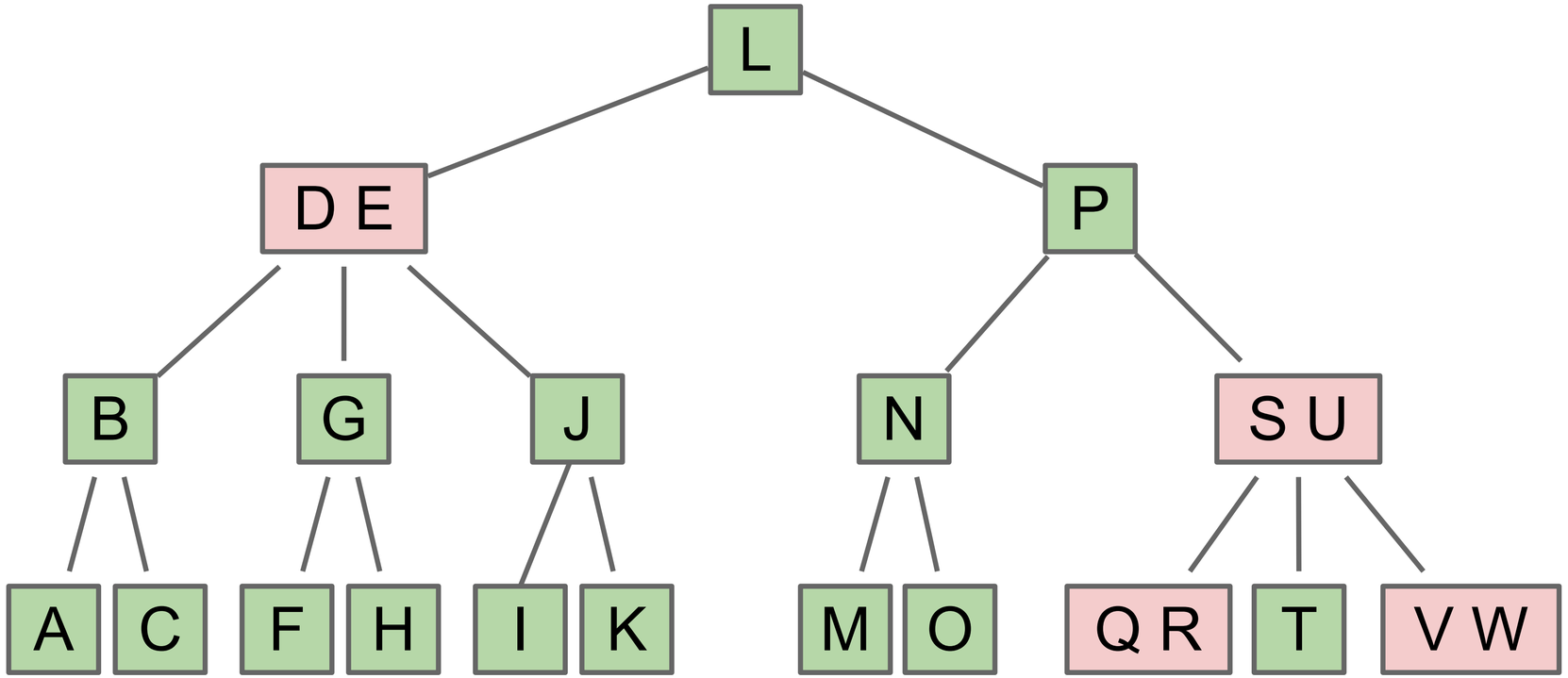




Valid!

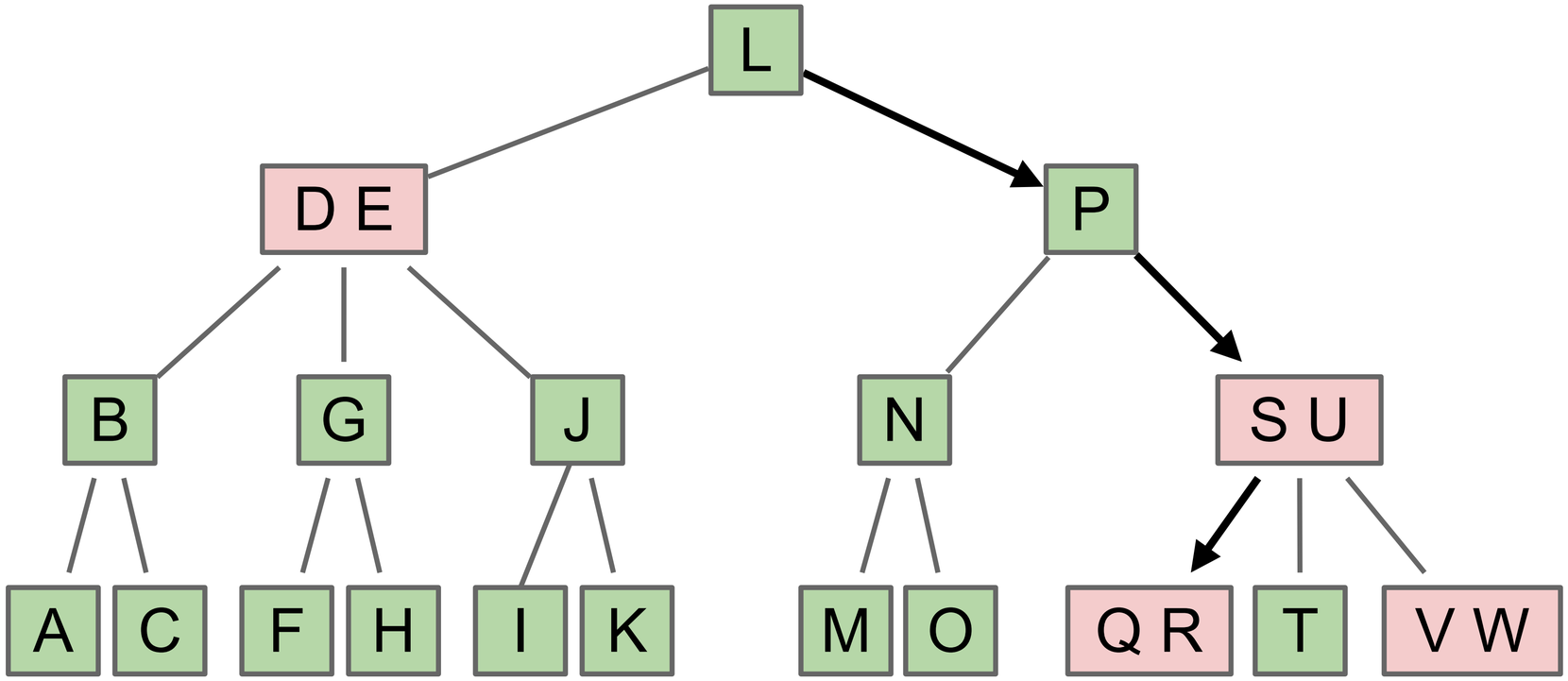
LLRB height

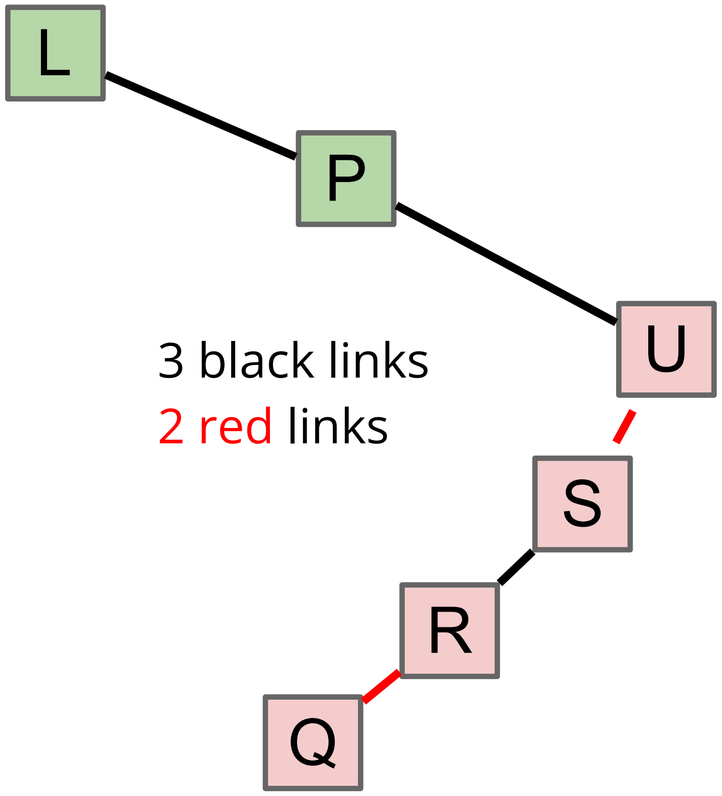
How tall is the corresponding LLRB for the 2-3 tree below? (3 - nodes in pink)



How tall is the corresponding LLRB for the 2-3 tree below? (3 - nodes in pink)

- Each 3-node becomes two nodes in the LLRB.
- Total height is 3 (black) + 2 (red) = 5.
- **More generally, an LLRB has no more than ~2x the height of its 2-3 tree.**



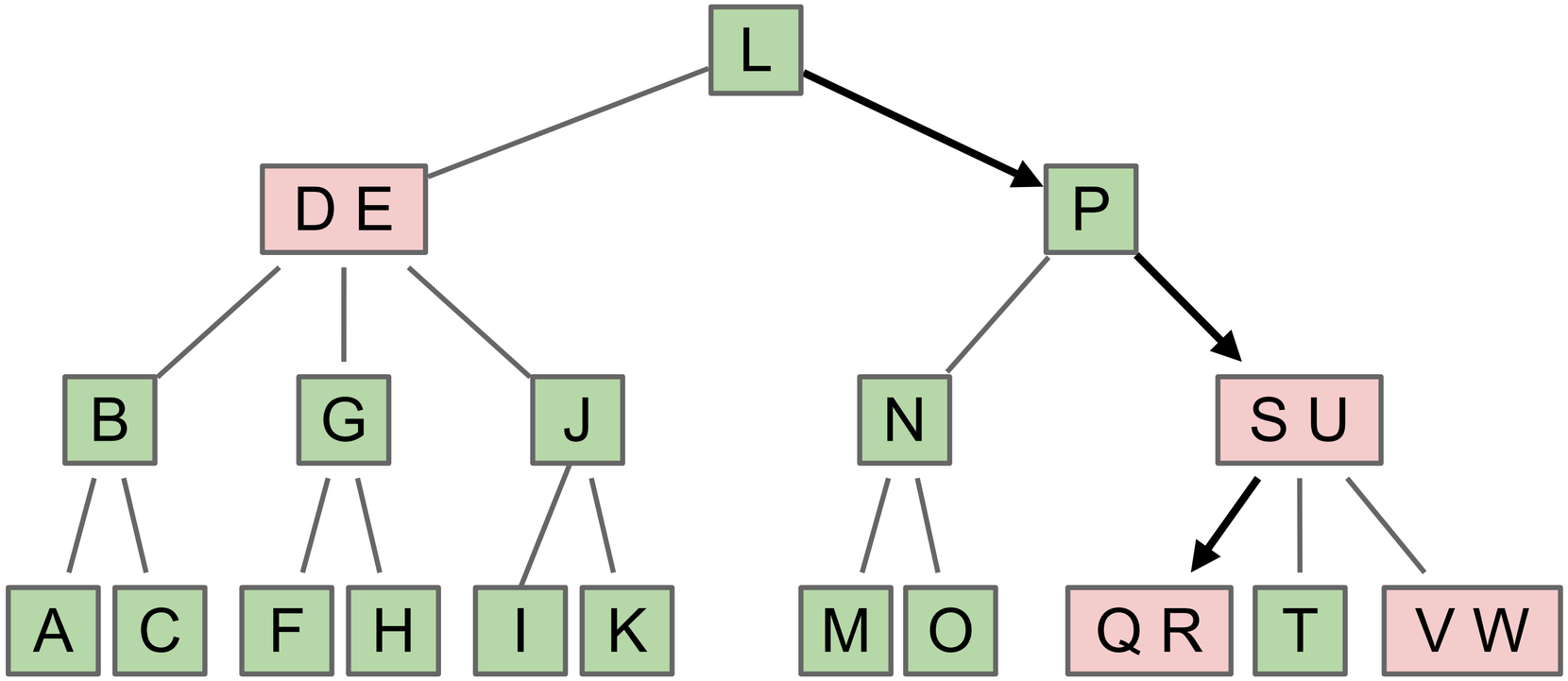


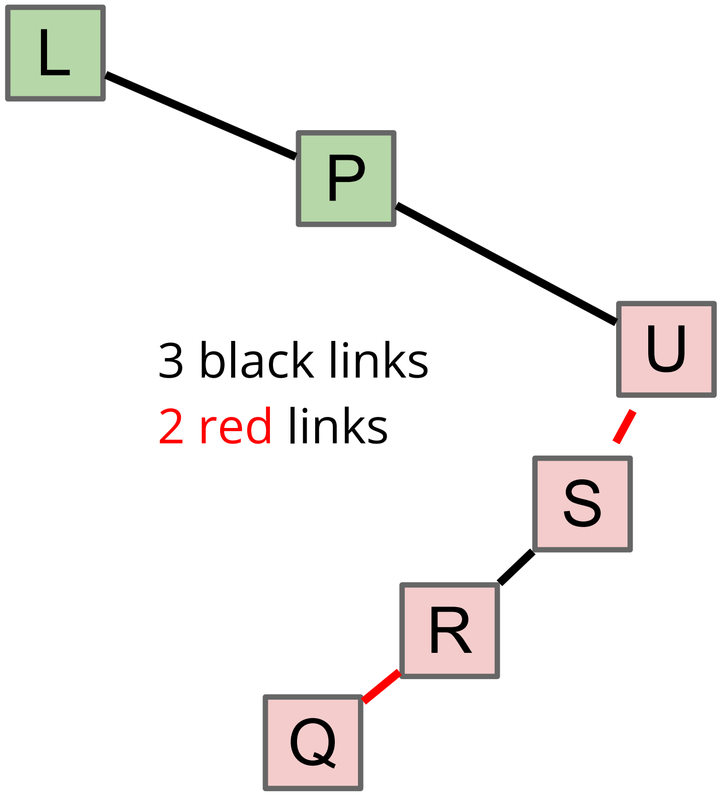
3 black links
2 red links

Dark line shows longest path (3 links).

LLRB Balance

Because 2-3 trees have logarithmic height, and the corresponding LLRB has height that is never more than ~2 times the 2-3 tree height, **LLRBs also have logarithmic height!**



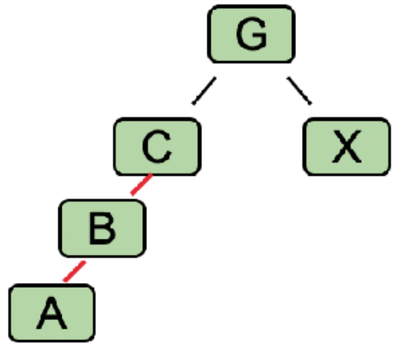


Dark line shows longest path (3 links).

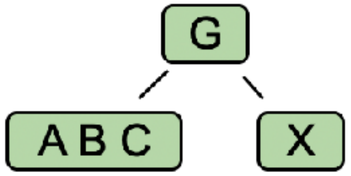
Left-Leaning Red Black Binary Search Tree (LLRB) Properties

Some handy LLRB properties/invariants:

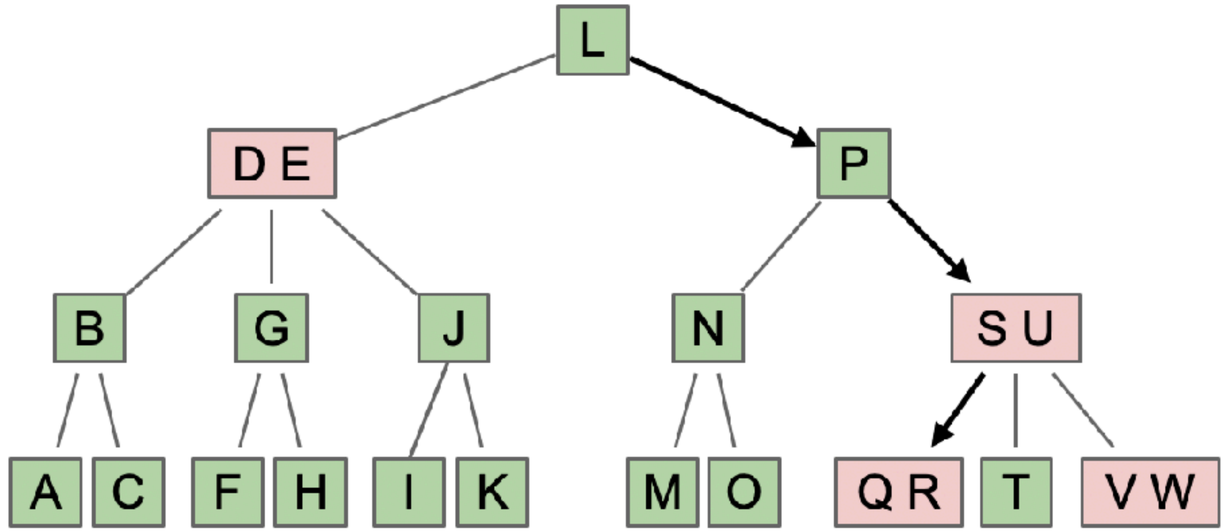
- No node has two red links [otherwise it'd be analogous to a 4 node, which are disallowed in 2-3 trees].
- Every path from root to null has same number of **black links** (because 2-3 trees have the same number of links to every leaf). LLRBs are therefore balanced.



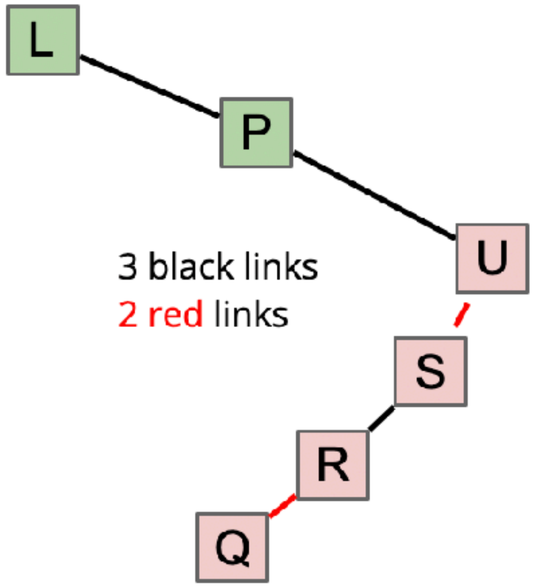
2-3 tree equivalent



Invalid, has 4 node.



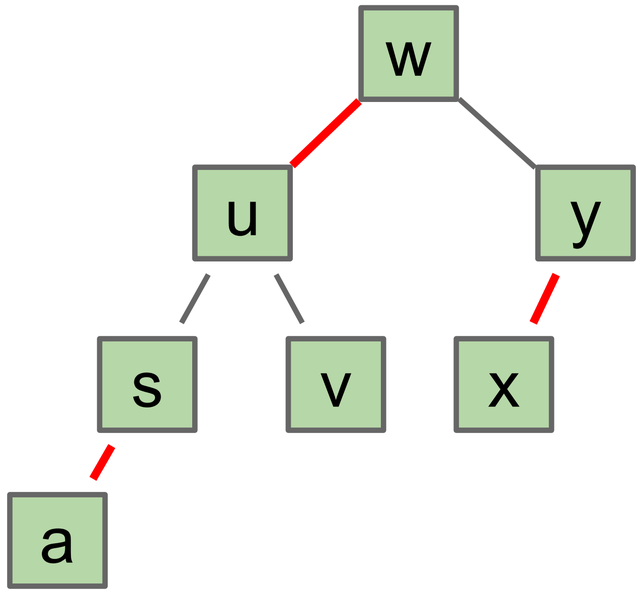
Dark line shows longest path (3 links).



3 black links
2 red links

Searching

Searching for a key in a LLRB is exactly like searching for it in any BST.



contains(b)

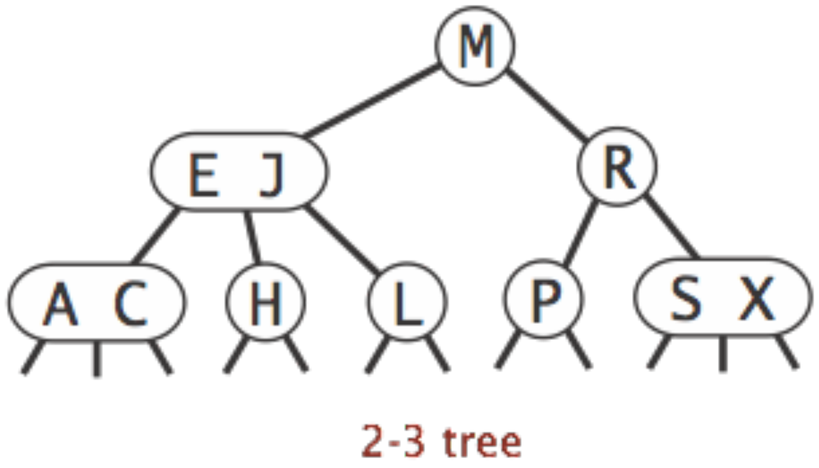
- b < w, go left
- b < u, go left
- b < s, go left
- b > a, go right
  - null node, b does not exist

Runtime? O(logn), because height of the LLRB is log(n).

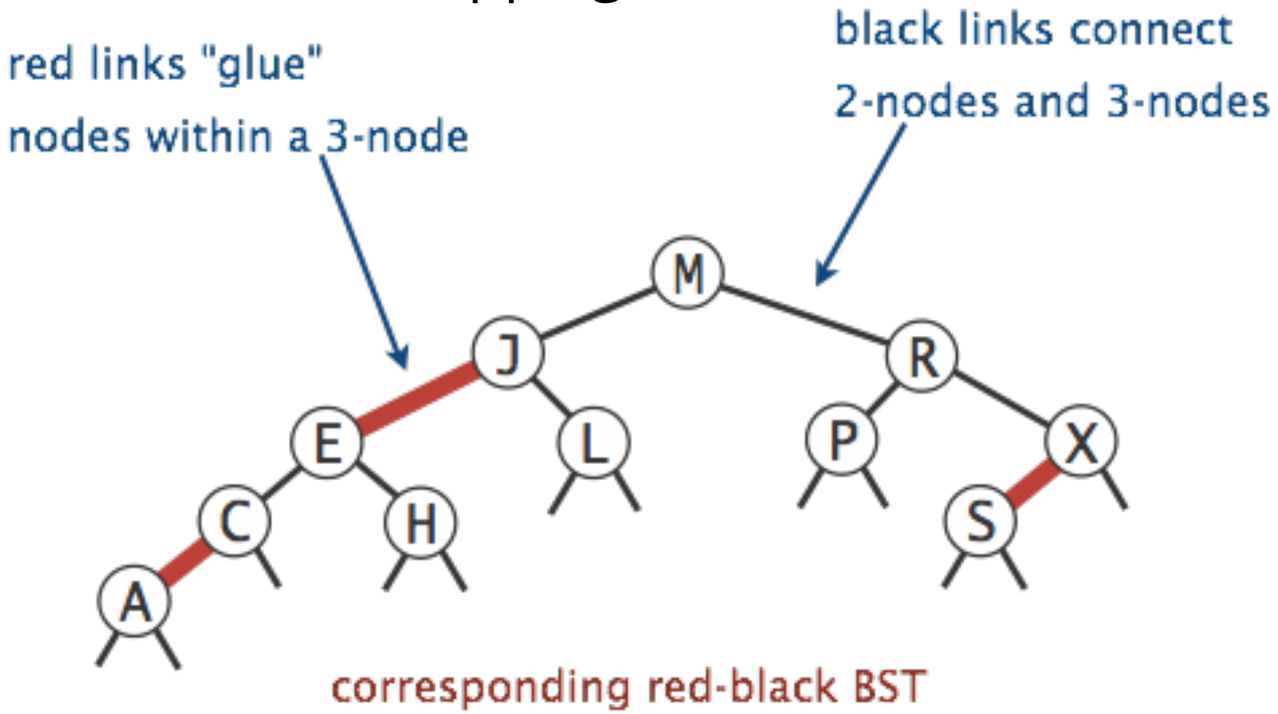
LLRB Construction algorithm

One last important question: Where do LLRBs come from?

- Would not make sense to build a 2-3 tree, then convert. Even more complex.
- Instead, it turns out we implement an LLRB insert as follows:
  - Insert as usual into a BST.
  - Use zero or more rotations to maintain the 1-1 mapping.



2-3 tree



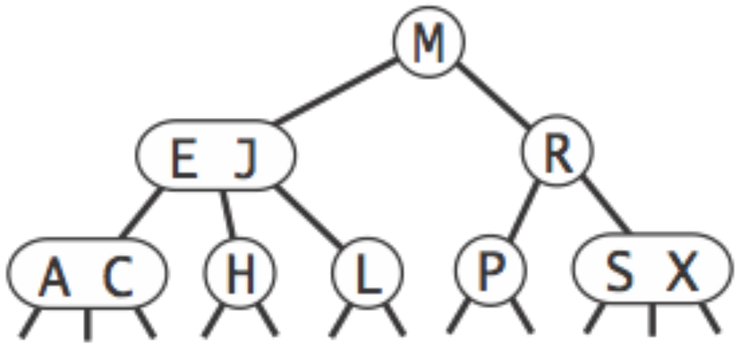
corresponding red-black BST

The 1-1 Mapping

There exists a 1-1 mapping between:

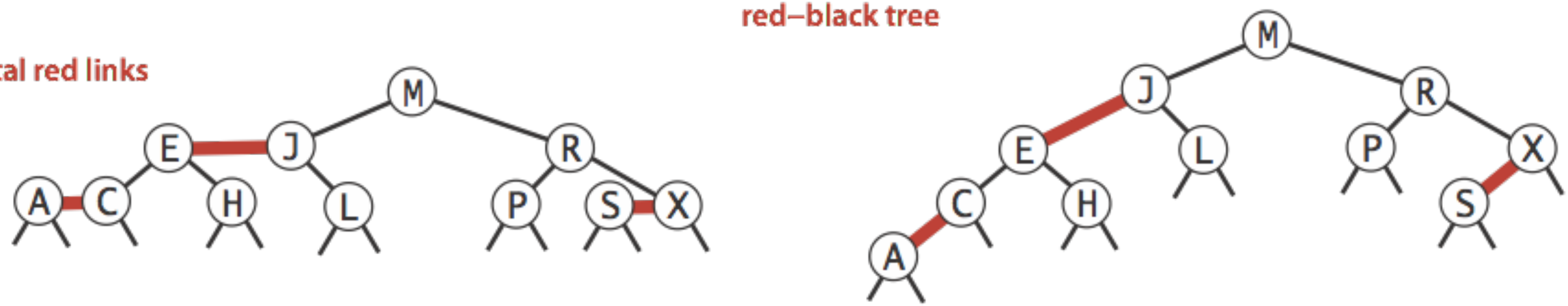
- 2-3 Tree
- LLRB

2-3 tree



red–black tree

horizontal red links



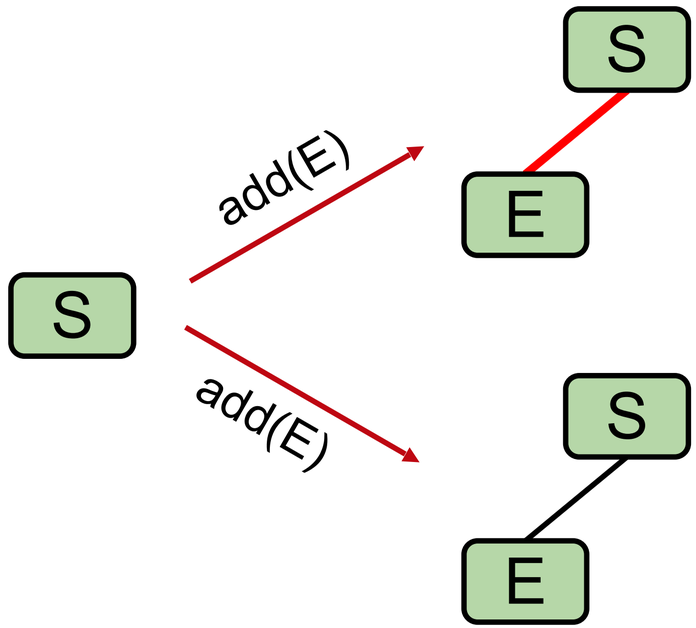
Implementation of an LLRB is based on maintaining this 1-1 correspondence.

- When performing LLRB operations, pretend like you're a 2-3 tree.
- Preservation of the correspondence will involve tree rotations.

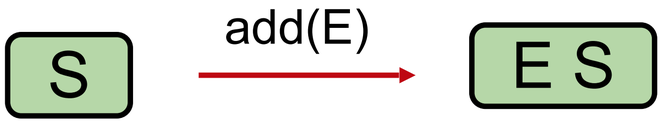
Insertion rules

Design Task #1: Insertion Color

Should we use a red or black link when inserting E to a tree that just contains S?



LLRB World

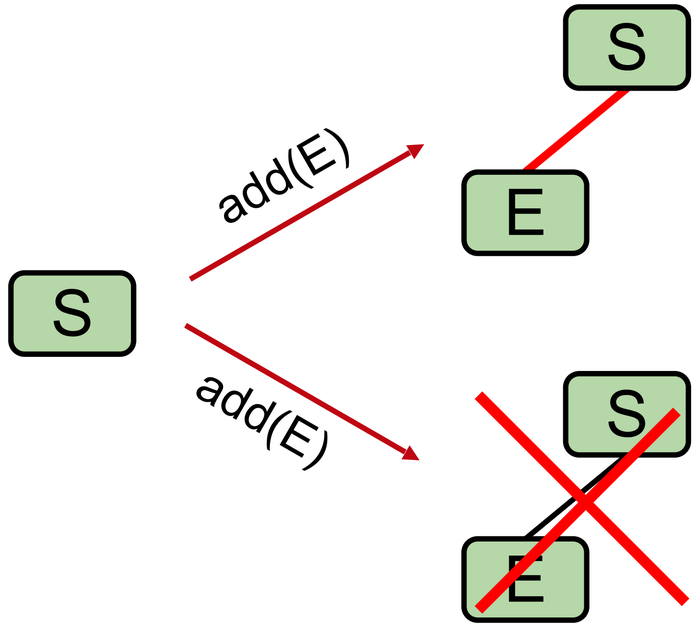


World 2-3

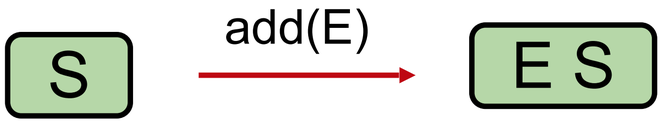
Design Task #1: Insertion Color

Should we use a red or black link when inserting?

- Use red! In 2-3 trees new values are ALWAYS added to a leaf node (at first).



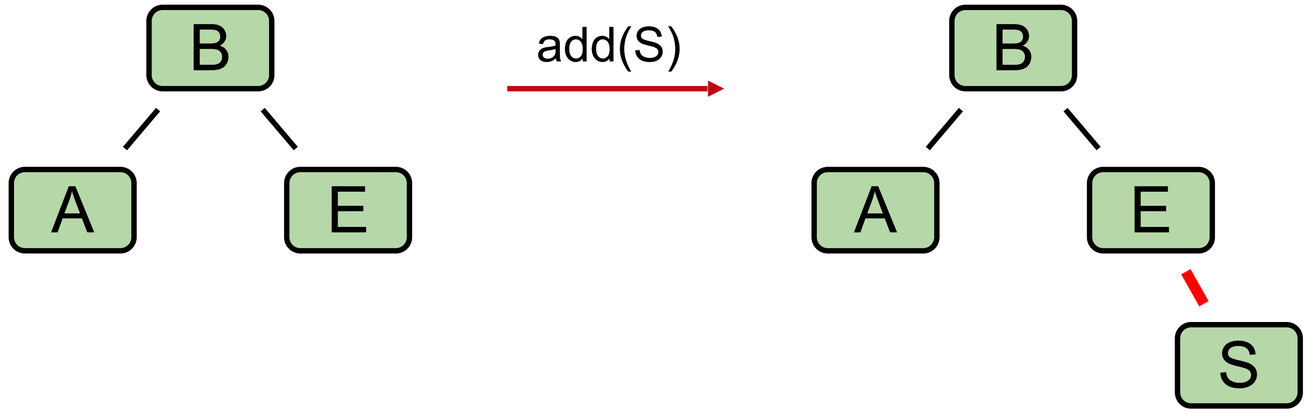
LLRB World



World 2-3

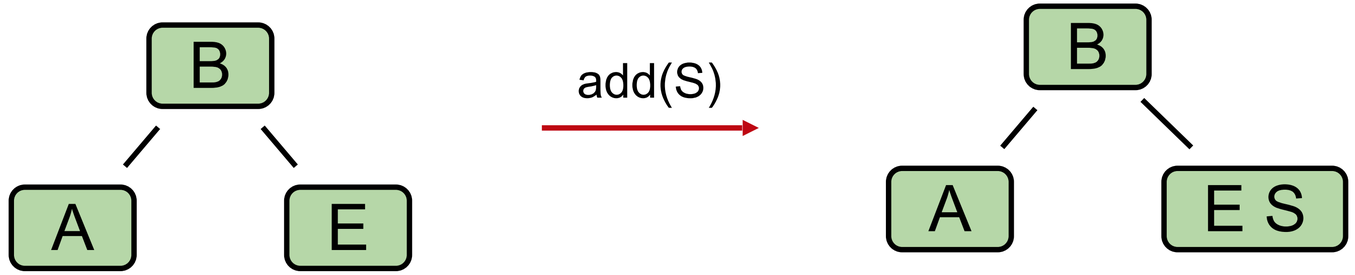
Design Task #2: Insertion on the Right

Suppose we have leaf E, and insert S with a red link. What is the problem below, and what do we do about it?



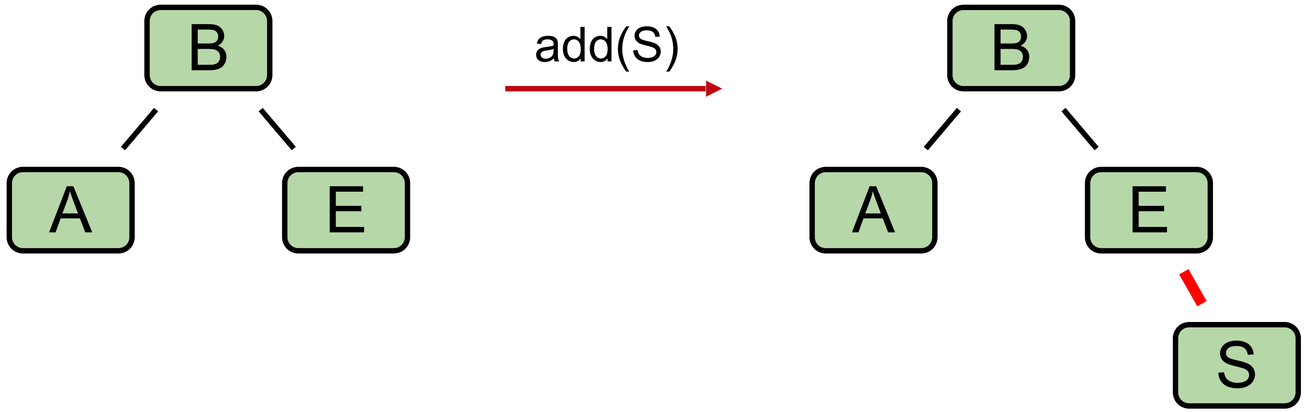
LLRB World

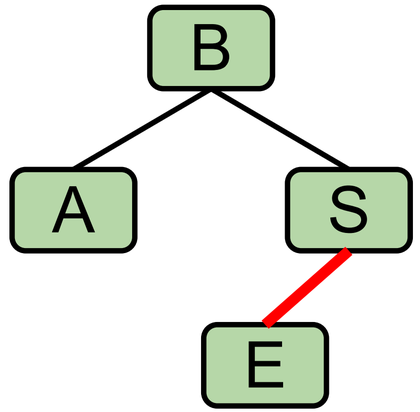
World 2-3



Design Task #2: Insertion on the Right

Suppose we have leaf E, and insert S with a red link. What is the problem below, and what do we do about it: Right links aren't allowed. What rotation fixes this?

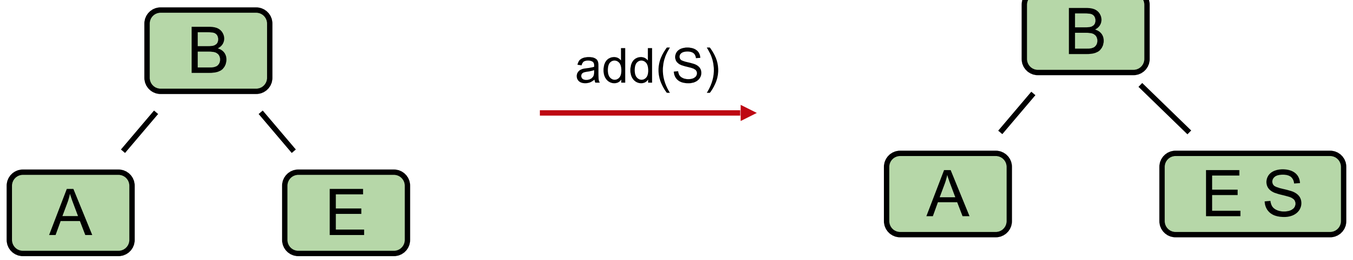




Hint: This is the correct representation of the 2-3 tree.

LLRB World

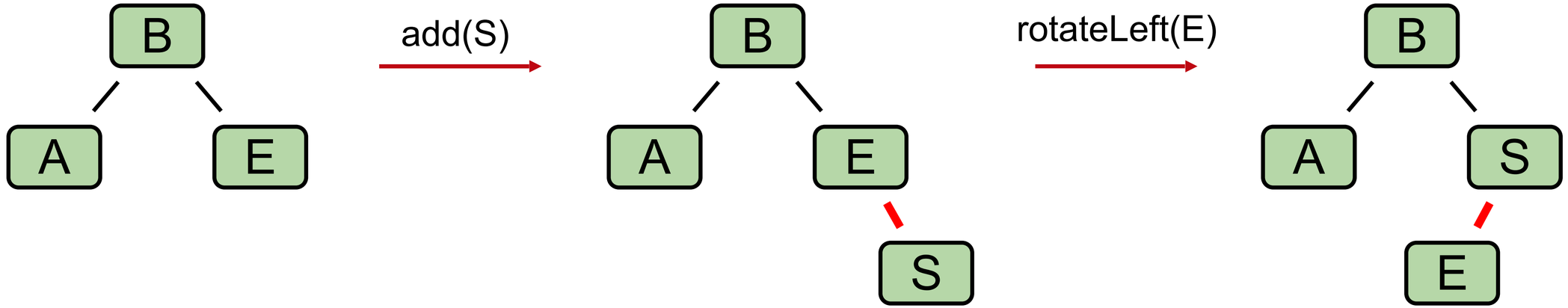
World 2-3



What rotation operation gives us this tree?

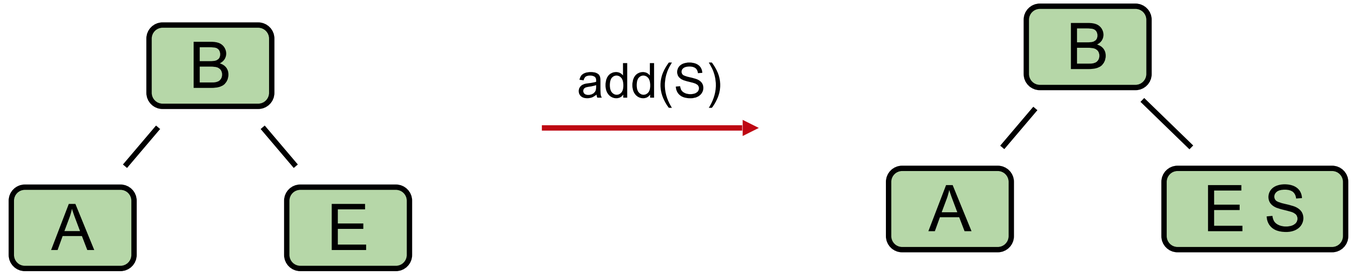
Design Task #2: Insertion on the Right

Suppose we have leaf E, and insert S with a red link. What is the problem below, and what do we do about it: Right links aren't allowed, so rotateLeft(E).



LLRB World

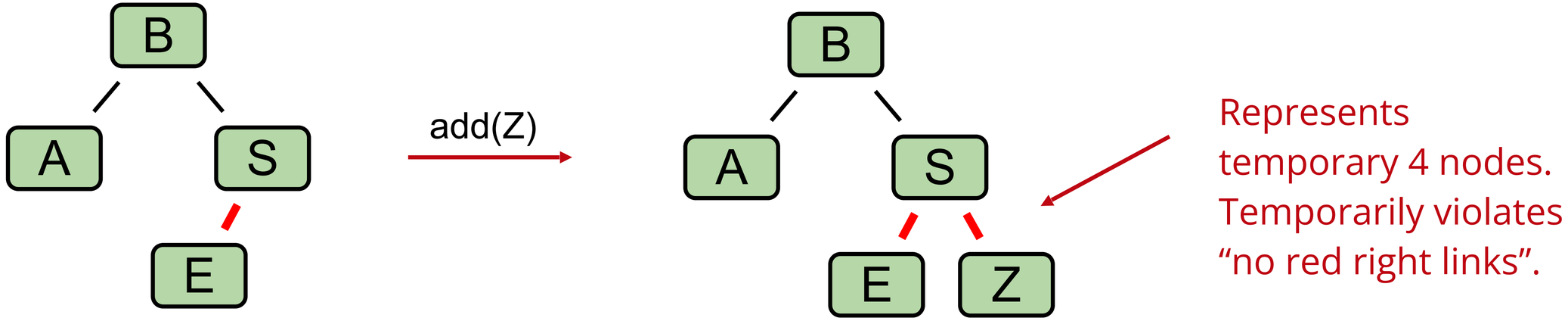
World 2-3



New Rule: Representation of Temporary 4-Nodes

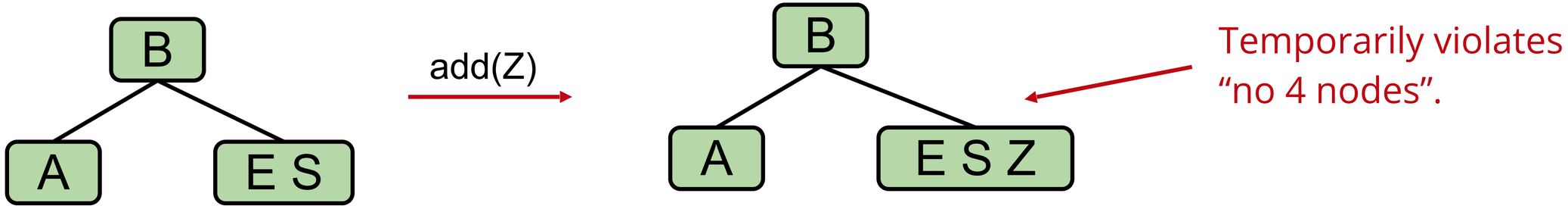
We will represent temporary 4-nodes (before we split and bubble up the middle element) as BST nodes with two red links.

- This state is only temporary, so temporary violation of “left leaning” is OK.



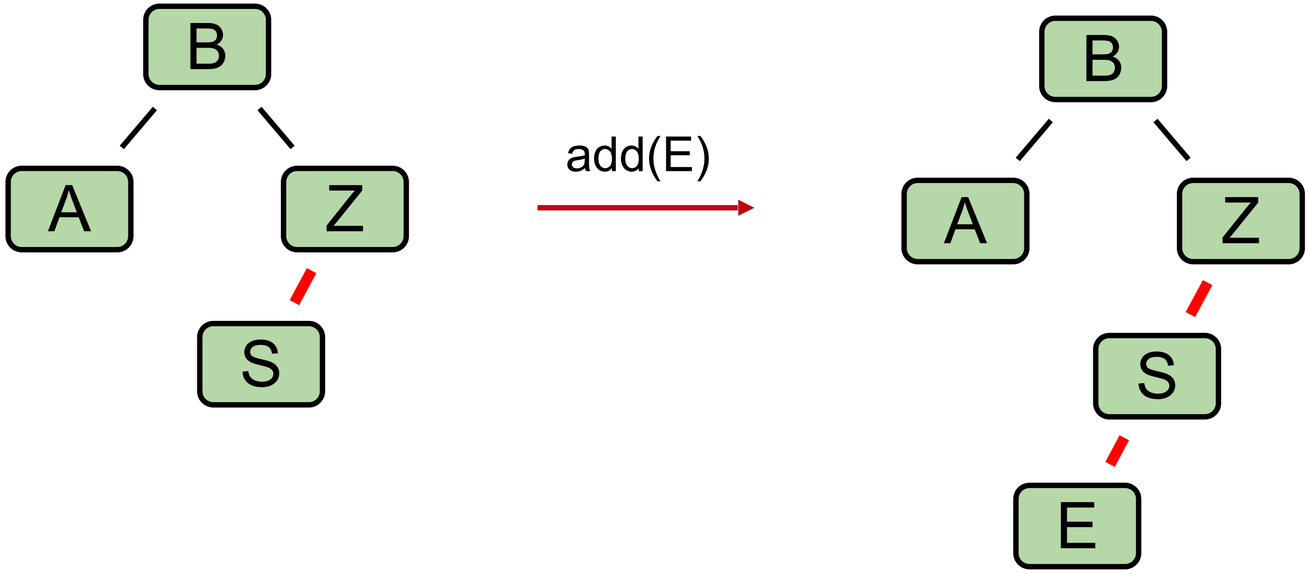
LLRB World

World 2-3



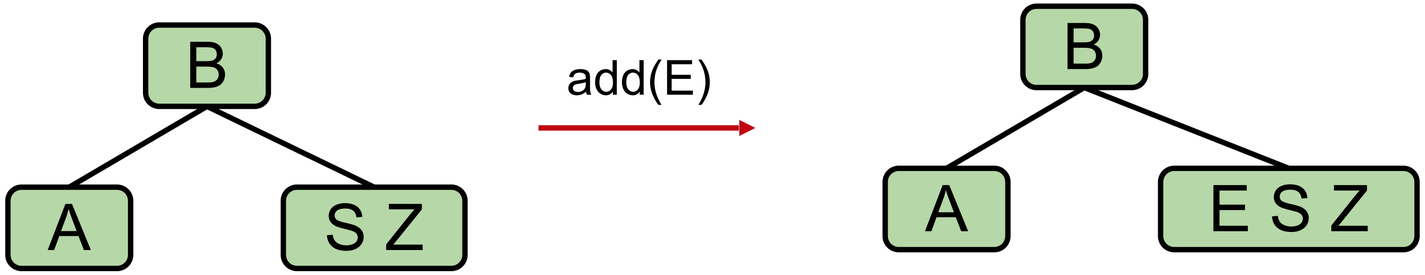
Design Task #3: Double Insertion on the Left

Suppose we have the LLRB below and insert E. We end up with the wrong representation for our temporary 4 node. What rotation should we do so that the temporary 4 node has 2 red children (one left, one right) as expected?



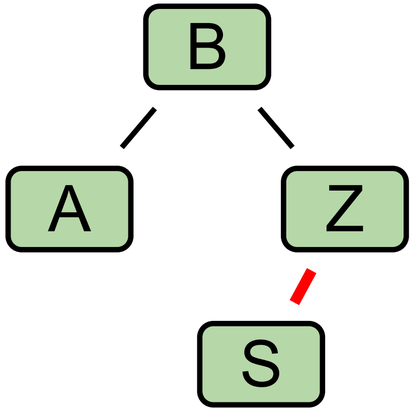
LLRB World

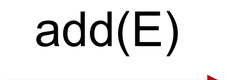
World 2-3

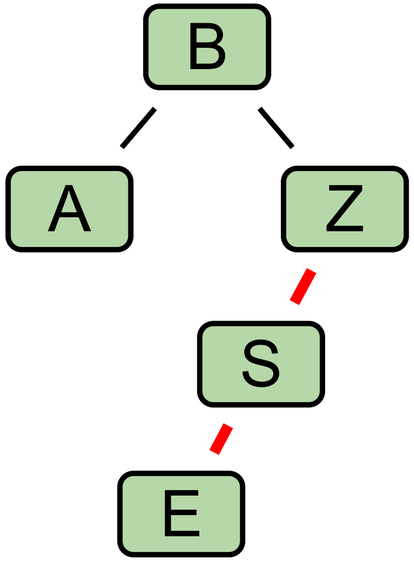


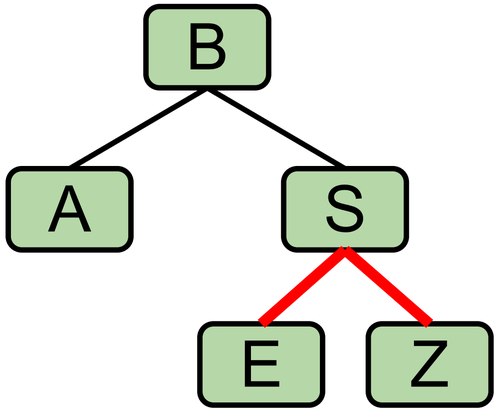
Design Task #3: Double Insertion on the Left

Suppose we have the LLRB below and insert E. We end up with the wrong representation for our temporary 4 node. What rotation should we do so that the temporary 4 node has 2 red children (one left, one right) as expected?



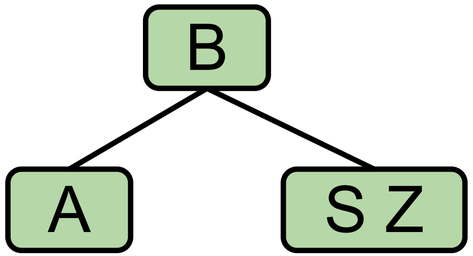


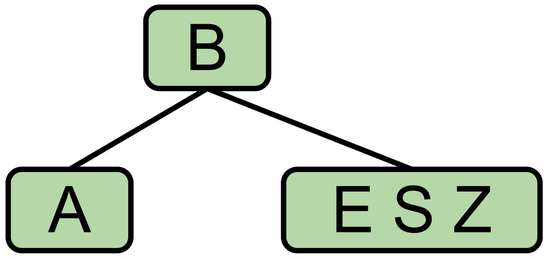




Hint: This is the correct representation of the 2-3 tree.

LLRB World





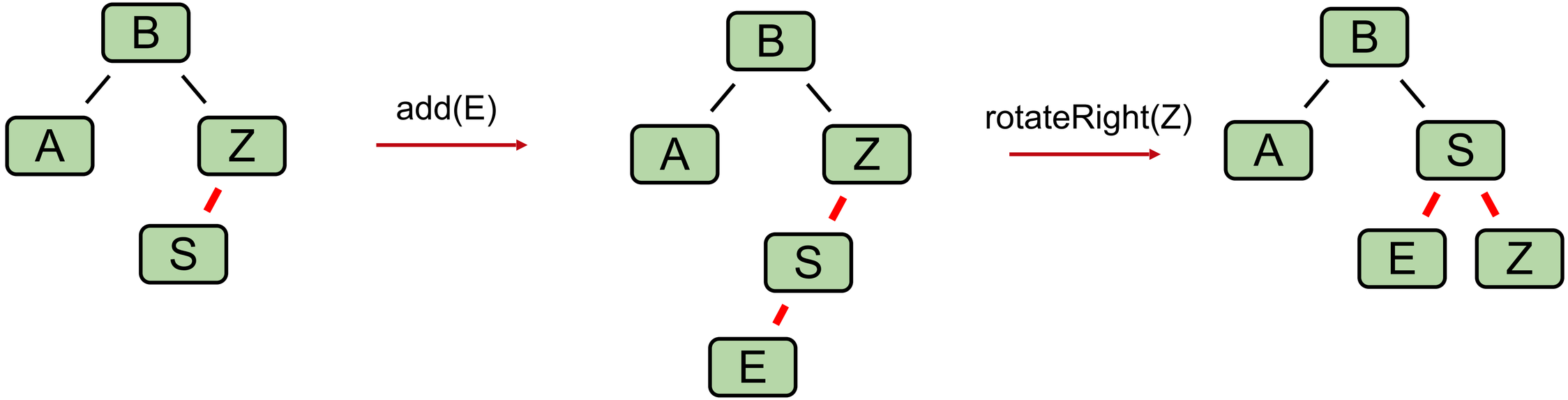
What rotation operation gives us this tree?

World 2-3

Design Task #3: Double Insertion on the Left

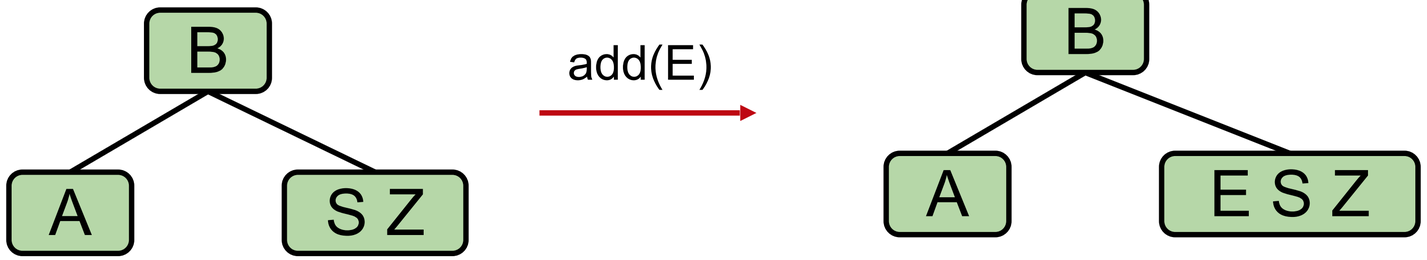
Suppose we have the LLRB below and insert E. We end up with the wrong representation for our temporary 4 node. What should we do?

- Rotate Z right.



LLRB World

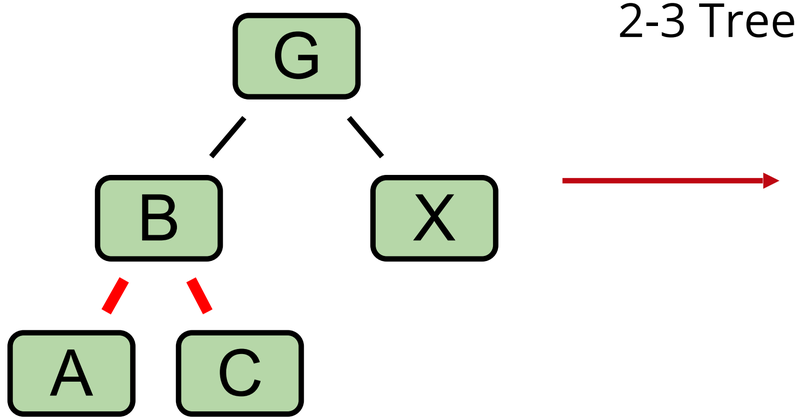
World 2-3



Design Task #4: Splitting Temporary 4-Nodes

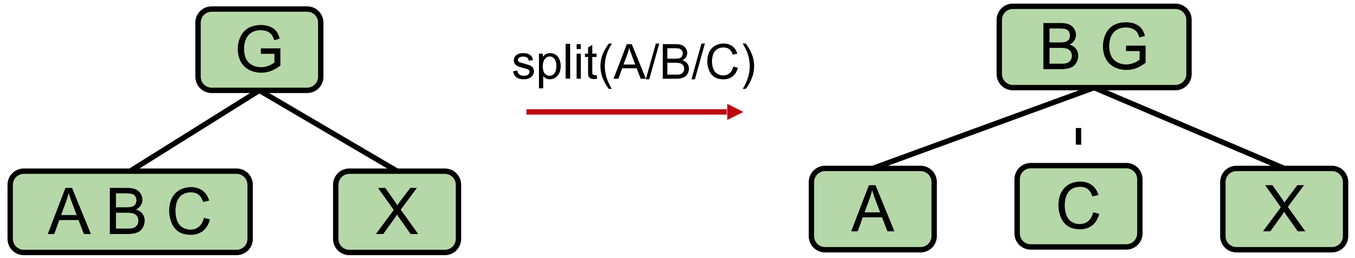
Suppose we have the LLRB below which includes a temporary 4 node. What should we do next?

Hint: Ask yourself “What Would 2-3 Tree Do?” WW23TD?



LLRB World

World 2-3

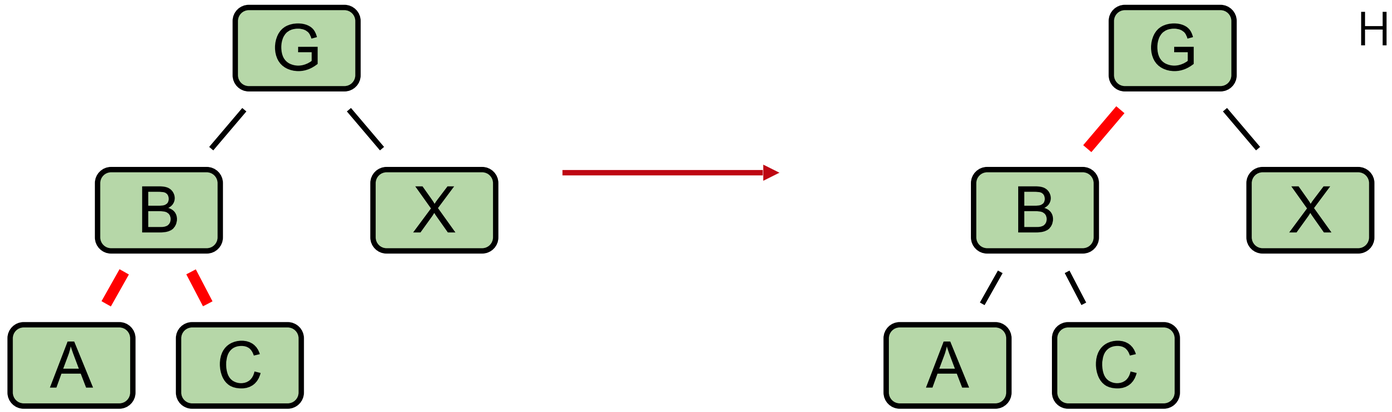


Design Task #4: Splitting Temporary 4-Nodes

Suppose we have the LLRB below which includes a temporary 4 node. What should we do next?

Hint: Ask yourself “What Would 2-3 Tree Do?” WW23TD?

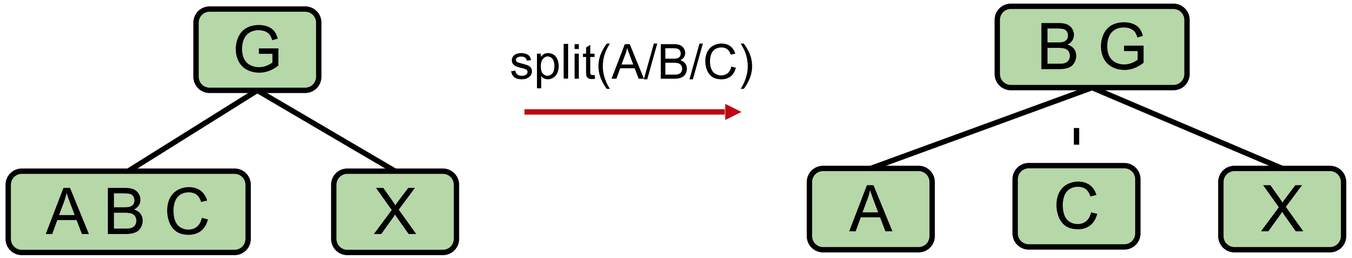
Hint2: This is the correct representation of the 2-3 tree. How do we get this tree?



Hint3: We don’t need rotation.

LLRB World

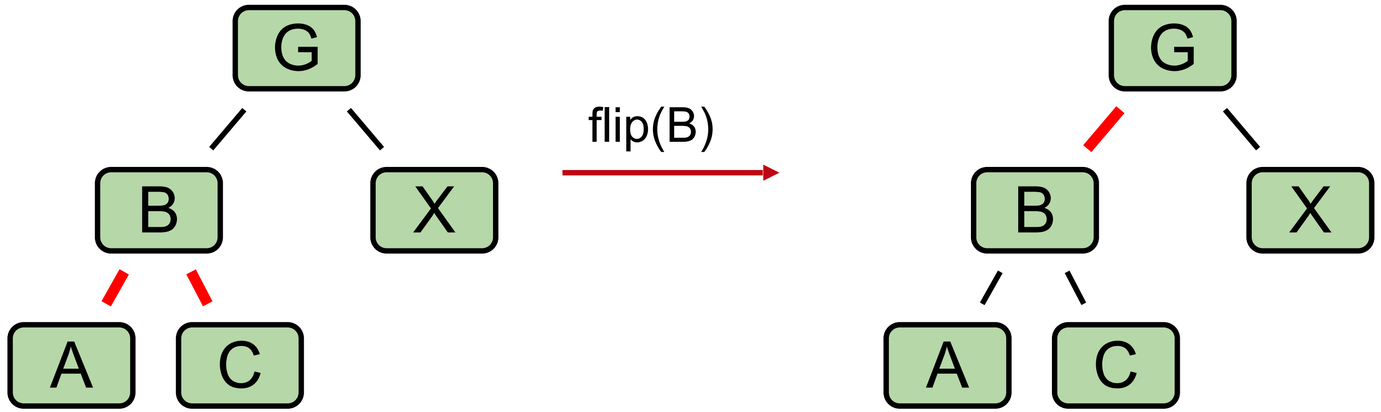
World 2-3



Design Task #4: Splitting Temporary 4-Nodes

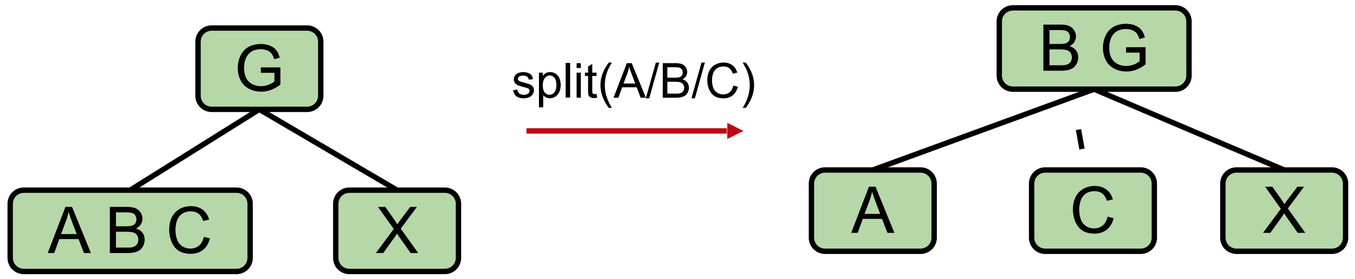
Suppose we have the LLRB below which includes a temporary 4 node. What should we do next?

- **Flip** the colors of all edges touching B.
  - Note: This doesn't change the BST structure/shape.



LLRB World

World 2-3



… and That's It!

Congratulations, you just invented the red-black BST.

- When inserting: Use a red link.
- If there is a *right leaning "3-node"*, we have a **Left Leaning Violation.**
  - **Rotate left** the appropriate node to fix.
- If there are *two consecutive left links*, we have an **Incorrect 4 Node Violation.**
  - **Rotate right** the appropriate node to fix.
- If there are any *nodes with two red children*, we have a **Temporary 4 Node.**
  - **Color flip** the node to emulate the split operation.

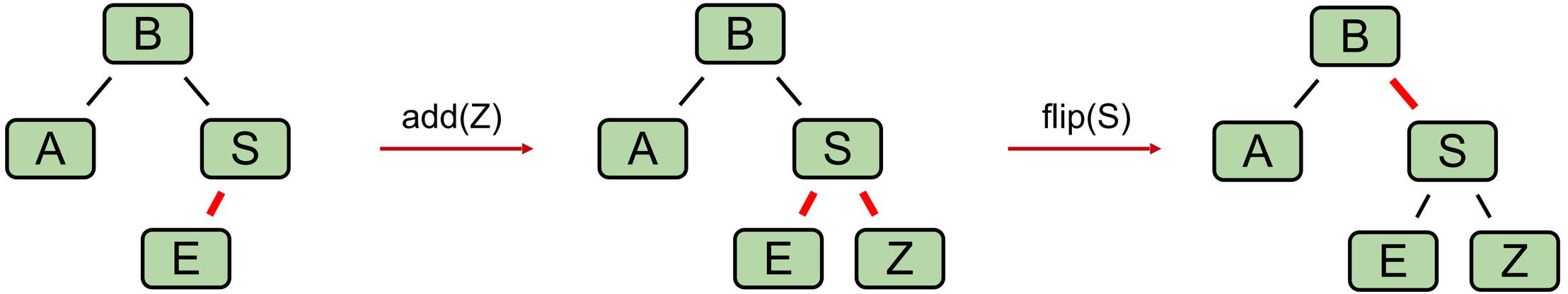
One last detail: Cascading operations.

- It is possible that a rotation or flip operation will cause an additional violation that needs fixing.

Worksheet time!

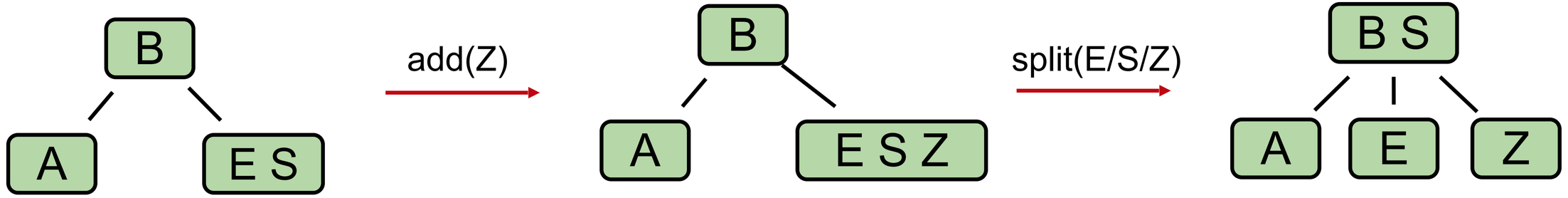
Inserting Z gives us a temporary 4 node.

- To fix it, we color flip. But this yields an invalid tree.
- What is the violation that has occurred? How can we fix it?



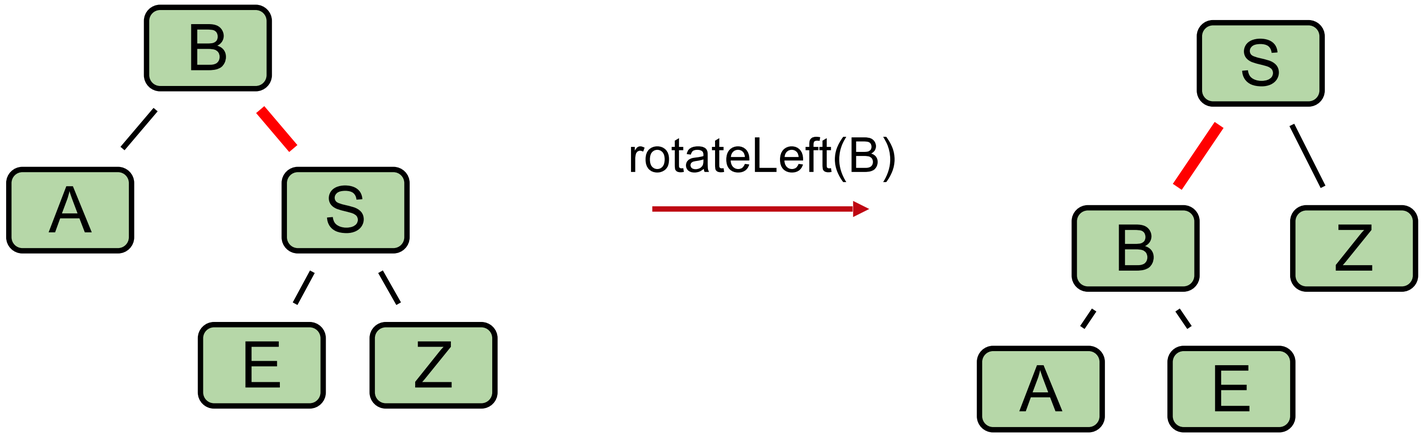
LLRB World

World 2-3

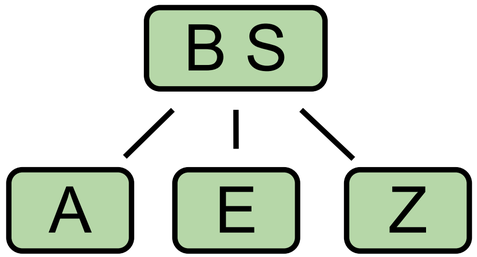


Worksheet answers

- What is the violation that has occurred? -> We have a right-leaning 3 node (B-S).
- How can we fix it? -> RotateLeft(B).



LLRB World



World 2-3

Runtime analysis

LLRB Runtime

The runtime analysis for LLRBs is simple if you trust the 2-3 tree runtime, since they're isometric.

- LLRB tree has height O(log N).
- Contains is trivially O(log N).
- Insert is O(log N).
  - O(log N) to add the new node.
  - O(log N) rotation and color flip operations per insert.
    - Rotation and color flip operations are constant time.

We will not discuss LLRB delete.

Search Trees

In the last 3 lectures, we talked about using search trees to implement dictionaries/maps.

- **Binary search trees** are simple, but they are subject to imbalance.
- **2-3 Trees (B Trees)** are balanced, but painful to implement and relatively slow.
- **LLRBs** insertion is simple to implement (but delete is hard).
  - Works by maintaining mathematical bijection with a 2-3 trees.
- Java's TreeMap (built in dictionaries) is a red-black tree (not left leaning).
  - Maintains correspondence with 2-3-4 tree (is not a 1-1 correspondence).
  - Allows glue links on either side (see Red-Black Tree).
  - More complex implementation, but significantly (?) faster.

Summary for dictionary operations

	Worst case			Average case		
	Search	**Insert**	**Delete**	**Search**	**Insert**	**Delete**
BST	n	n	n	$\log n$	$\log n$	$\sqrt{n}$
B-Trees and Red Black Trees	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$	$\log n$

… and Beyond

There are many other types of search trees out there.

- Other self balancing trees: AVL trees, splay trees, treaps, etc. There are at least hundreds of different such trees.

And there are other efficient ways to implement sets and maps entirely.

- Other linked structures: Skip lists are linked lists with express lanes.
- Other ideas entirely: **Hashing** is the most common alternative. We'll discuss this very important idea in our next lecture.

Lecture 17 wrap-up

- Checkpoint 2 next Monday! You can have a cheat sheet like the first one
- No HW next week!
- Reminder: Declare the CS major!

Resources

- Tree history: https://cs.pomona.edu/classes/cs62/history/trees/
- Reading from textbook: Chapter 3.3 (Pages 424-447); https://algs4.cs.princeton.edu/33balanced/
- LLRB visualization: https://algs4.cs.princeton.edu/GrowingTree/
- Red Black visualization (slightly different than LLRB): https://ds2-iiith.vlabs.ac.in/exp/red-black-tree/red-black-tree-oprations/simulation/redblack.html
- Practice problems behind this slide
- Most of these slides come from UC Berkeley's data structures course

Practice problem 1

Insert 7, 6, 5, 4, 3, 2, 1, into an initially empty LLRB. Make sure to draw the tree out at each iteration.
Hint: You should end up with a perfectly balanced BST!

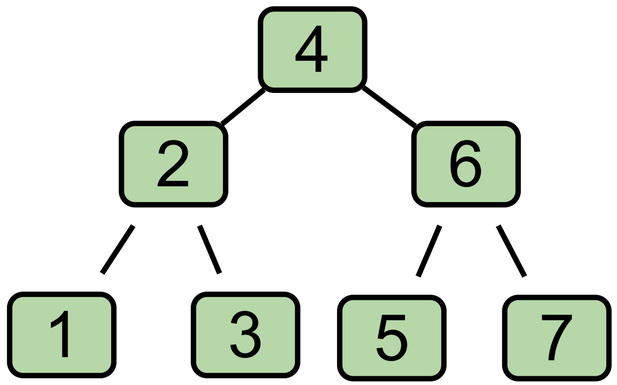
Practice problem 2

Draw the left-leaning red-black BST that results when you insert items with the keys E, A, S, Y, Q, U, E, S, T, I, O, N in that order into an initially empty tree.

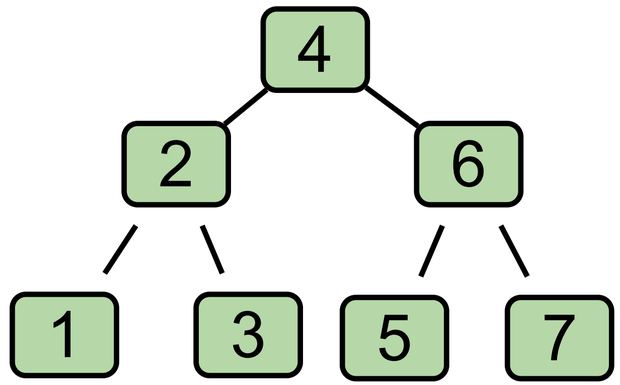
Practice solution 1

To check your work, see this demo (credit to Josh Hug @ UC Berkeley).

- Or see this video walkthrough of solution.



LLRB world



2-3 tree world (same!)

Practice solution 2