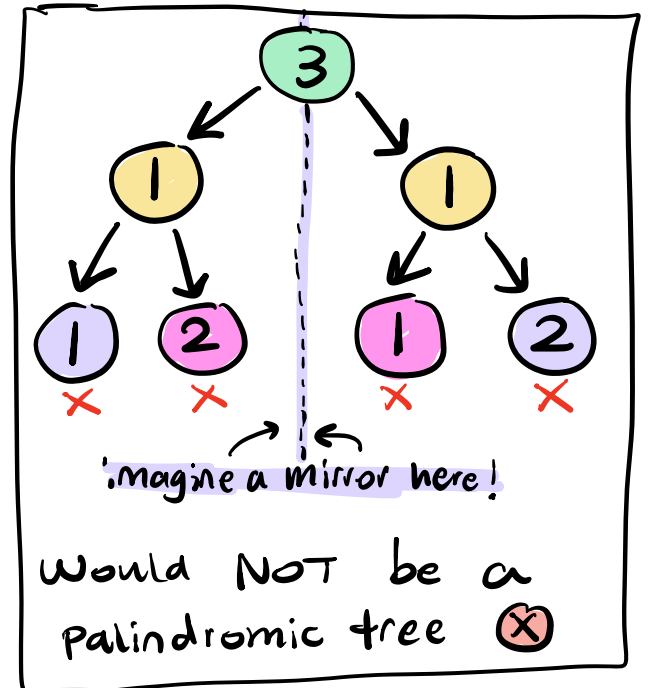
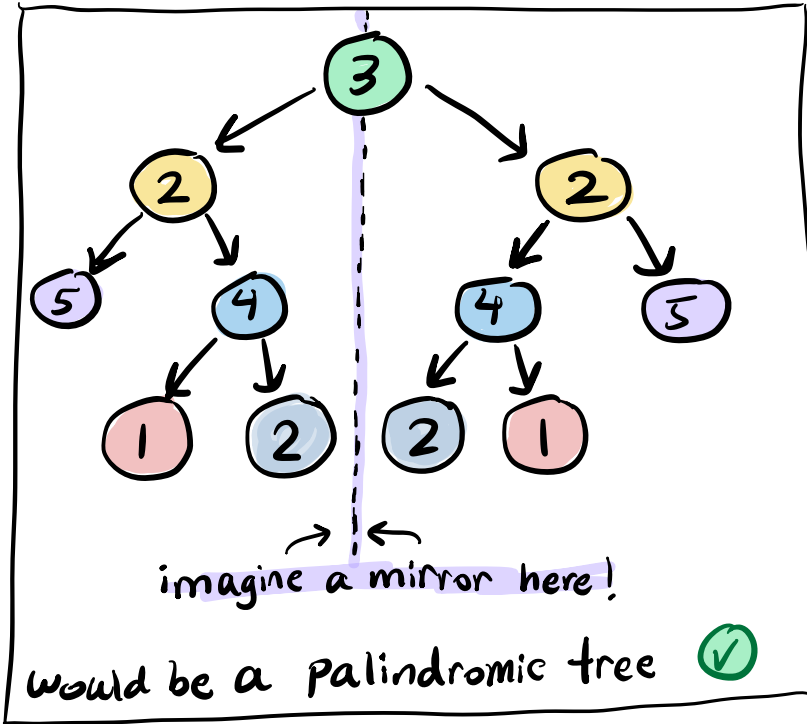


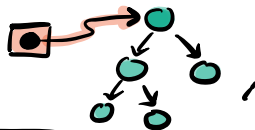


Palindromic Trees

Ben Yan ✨ ✨ ✨

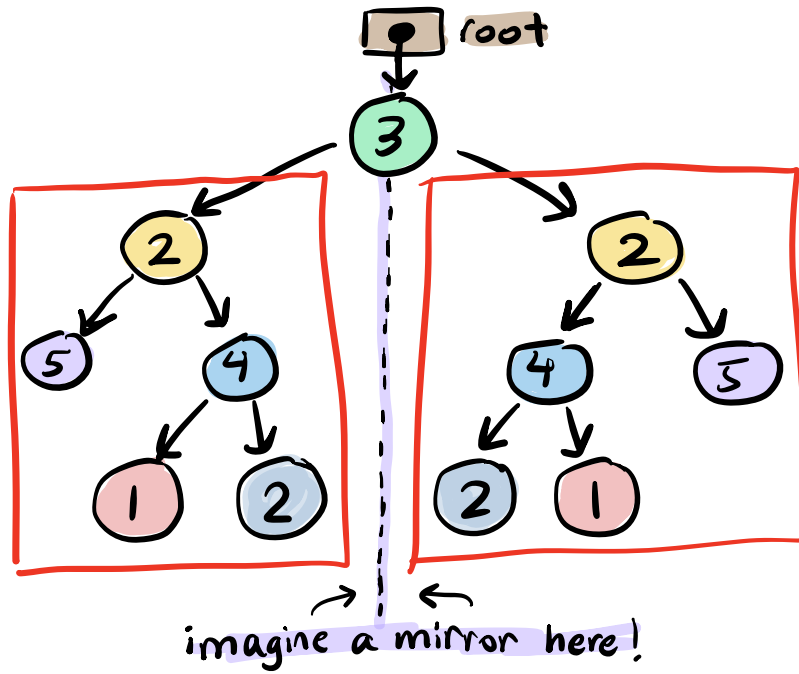
- * Use the binary tree node struct as defined earlier.
- * A word is a palindrome if it reads the same forwards & backwards, eg, "racecar" 🏎️, "tacocat" 🌮 🐱
- * We call a binary tree palindromic if the tree is it's own mirror image. That is,



- * Our task is to write a function, that given a pointer to the root of a binary tree, , returns true if the tree is palindromic, and false otherwise.

```
bool isPalindromic( TreeNode* root )
```

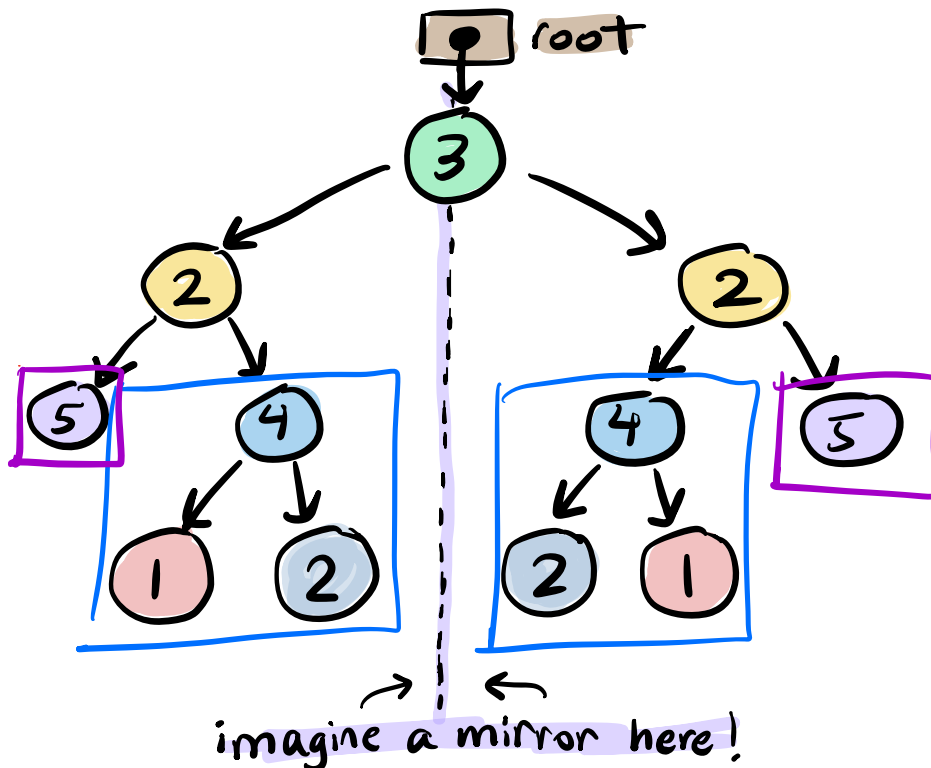
* By definition, root $\rightarrow$ left and root $\rightarrow$ right should be the mirror / reverse images of each other, to be palindromic.



Key observation: This mirroring property applies for the subtrees deeper down too! (recursion déjà vu?)

* root $\rightarrow$ left $\rightarrow$ left and root $\rightarrow$ right $\rightarrow$ right are mirror images!

* root $\rightarrow$ left $\rightarrow$ right and root $\rightarrow$ right $\rightarrow$ left are mirror images!



From here, we can write a helper function

mirror mirror,
on the wall...

```
bool areMirrors (TreeNode* root_one,  
                TreeNode* root_two)
```

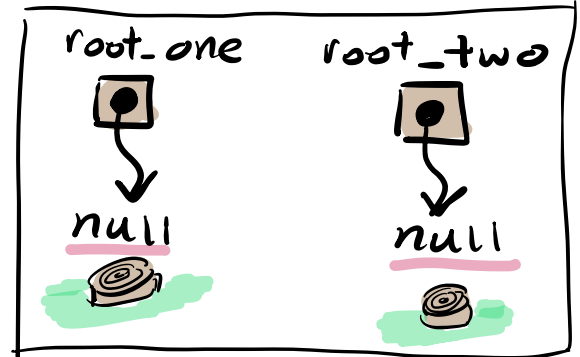


that returns whether the two trees are mirror images of one another.

Base Case(s)

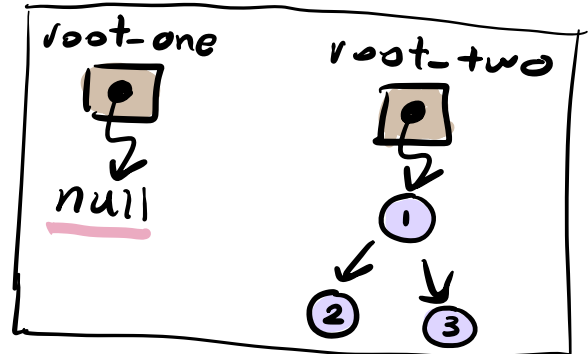
* Both trees are empty

↳ Vacuously, the two trees are mirror images of one another, so **true** ✓.



* One tree is empty, but the other is not.

↳ They can't be mirror images, so **false** ✗.



Recursive Case

* Neither trees are empty.

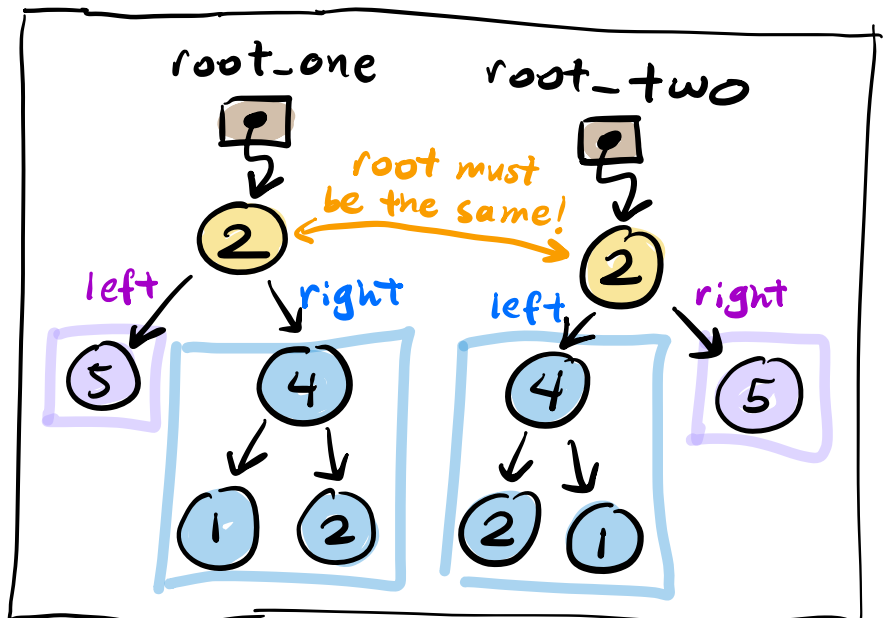
↳ The roots must have the same value.

AND

↳ root_one's left subtree mirrors with root_two's right

AND

↳ root_one's right subtree mirrors with root_two's left.



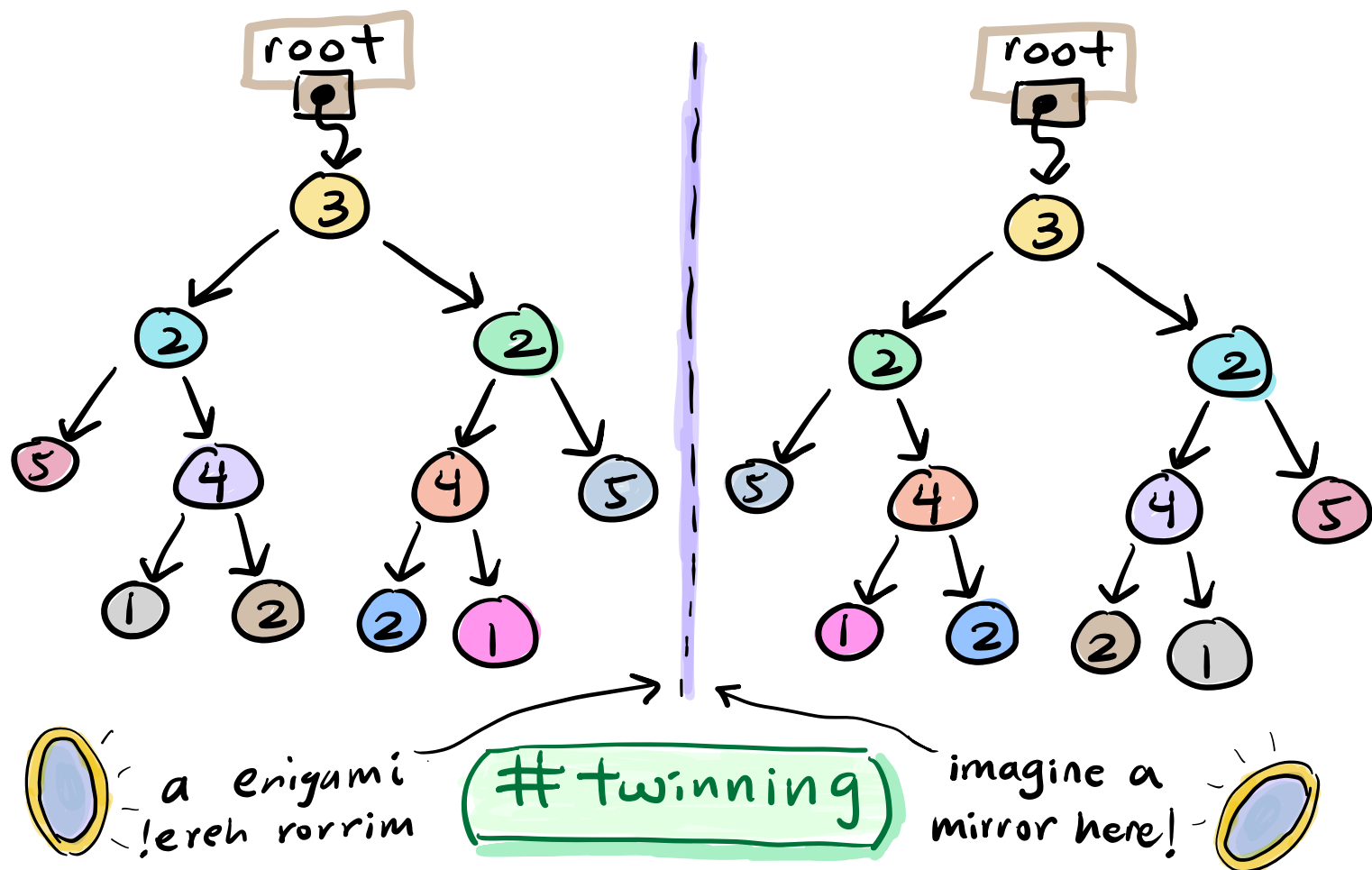
An implementation of this helper function is presented below.

```
bool areMirrors (TreeNode* root_one,  
                TreeNode* root_two) {  
    /* base case (s) */  
    if (root_one == nullptr && root_two == nullptr) {  
        // both trees are empty, so mirrors of each other  
        return true; (✓)  
    }  
    if (root_one == nullptr || root_two == nullptr) {  
        // one tree is empty, but not the other (above case for both)  
        return false; (✗)  
    }  
    /* recursive case */  
    // root nodes must match in data value / number, and the  
    // opposite subtrees must be mirror images  
    return (root_one->data == root_two->data)  
    && areMirrors(root_one->left, root_two->right)  
    && areMirrors(root_one->right, root_two->left);  
}
```

We can use this to implement isPalindrome :D.

```
bool isPalindrome (TreeNode* root) {  
    if (root == nullptr) {  
        return true; (✓) // empty tree  
    }  
    // check for mirror symmetry down the tree  
    return areMirrors(root->left, root->right);  
}
```

Alternatively, we can use the original definition that a palindromic tree is a mirror image of itself! Think drawing an exact duplicate tree next to it 🌲🌲.



That way, a clever, slightly shorter implementation is:

```
bool isPalindrome(TreeNode* root){  
    /* checks if tree is mirror image of itself */  
    return areMirrors(root, root);  
}
```