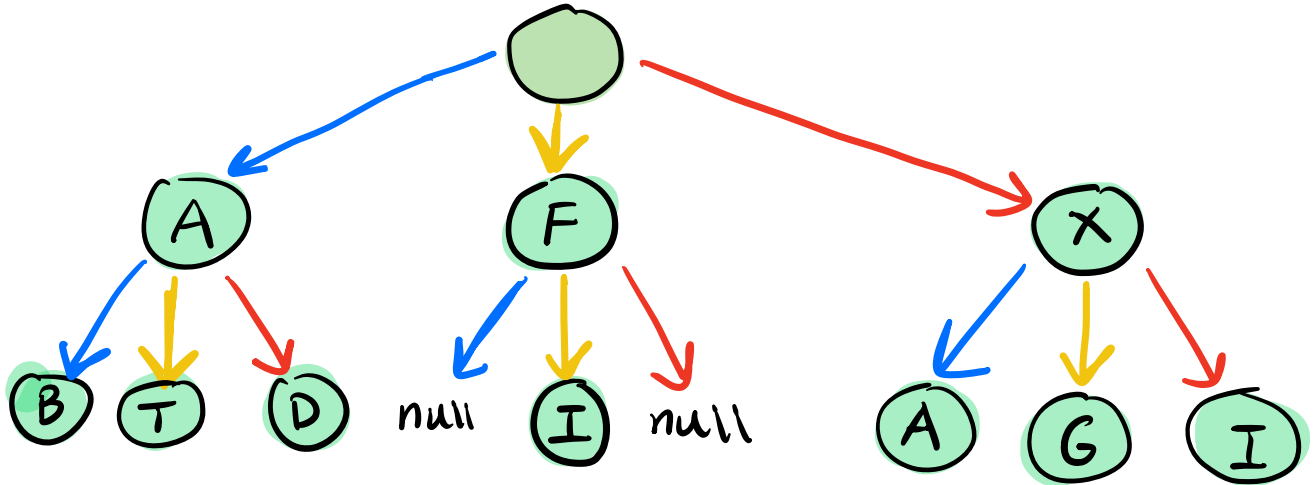




Decode Text (Tree Encoding)

Ben Yan ✨ ✨

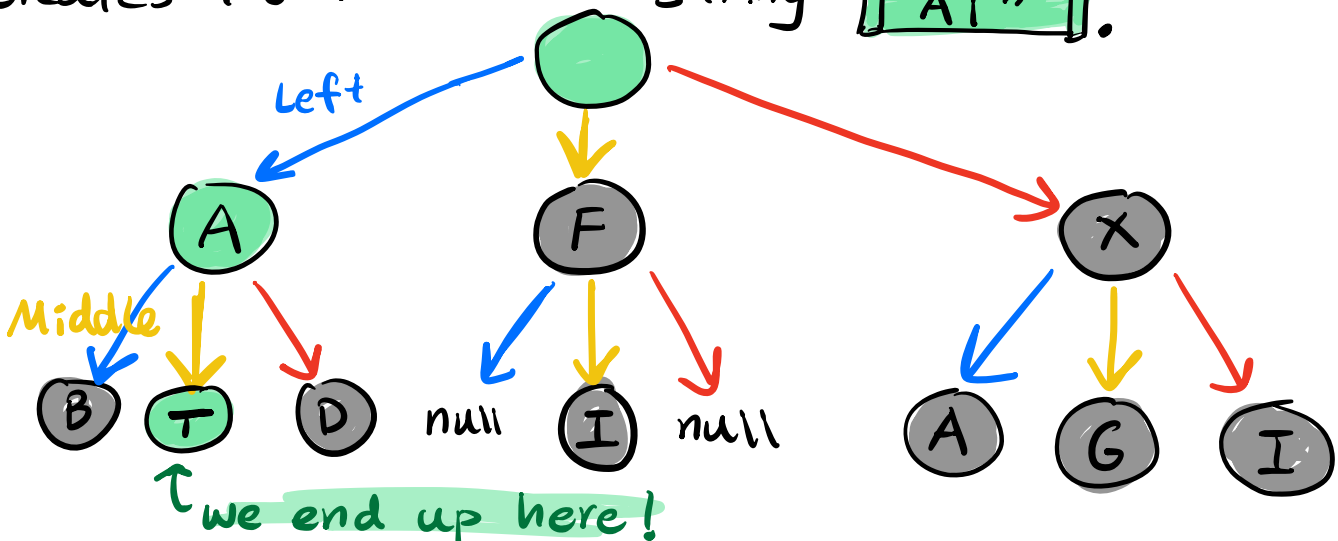
* Use the ternary tree node as defined previously.



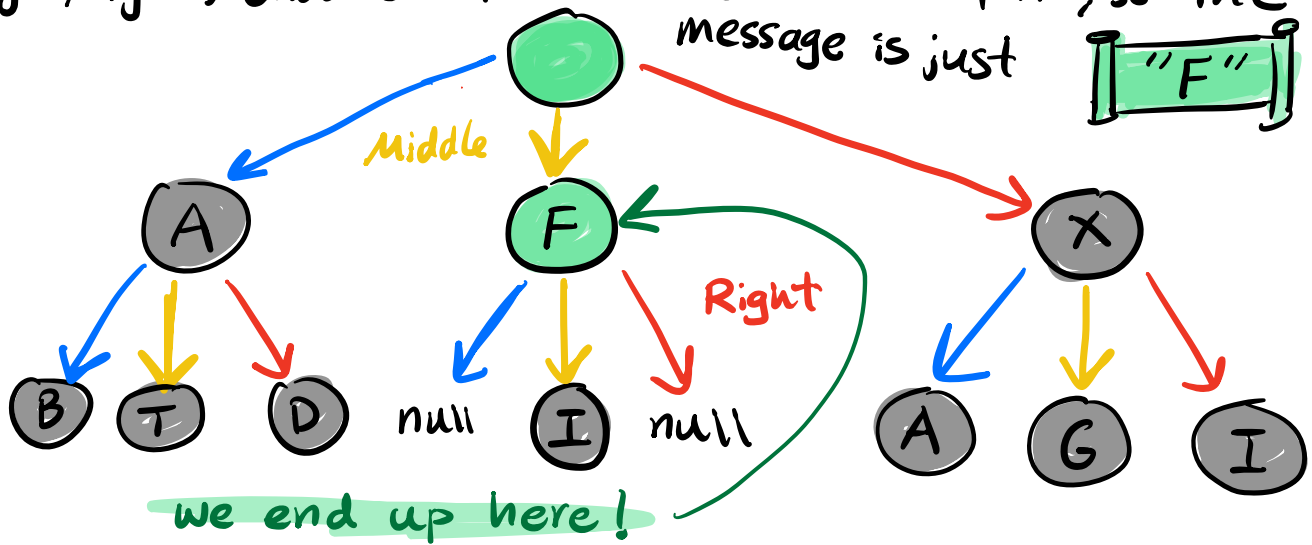
except for the root node!

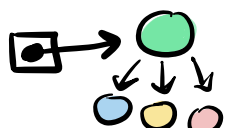
* Note: Each node stores a character, so a path through nodes creates a sequence of characters, or a message!! 🍷 ("message in a bottle" by Taylor Swift)

For example, the path "LM" (left, then middle) creates the message string "AT".



As another example, the path "MRR" (middle, right, right) ends early as we reach a nullptr, so the message is just "F"



* Our task is, given a pointer to the root node , and a path sequence, eg, "LM", return the text message string obtained by the directions in the sequence / the characters in the nodes we visit.

```
string decodeSequence (TernaryTreeNode * root,
                      String pathSequence)
```

* Assume valid directions, and the ternary tree is non-empty.
 ↳ e.g, "L" (left), "M" (middle), "R" (right)

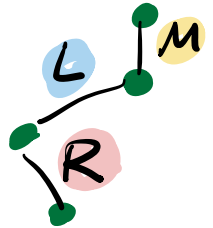
* When reaching a dead end (i.e.  nullptr), stop processing the remaining directions, and return the in-progress string up to that point.

Step 1 First, we'll probably want to keep track of a message string in-progress, and loop through the directions to do some traveling! 🚗

```
string decodeSequence(TernaryTreeNode * root,  
                     string pathSequence) {  
    string message = "";
```

```
    for (char direction : pathSequence) {  
        // keep going from here!
```

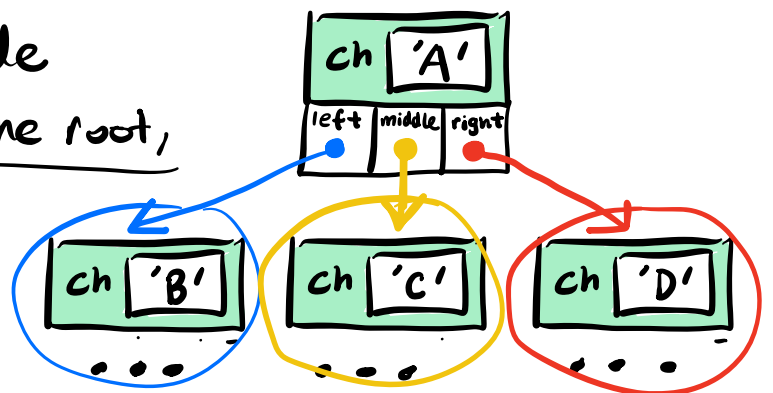
```
    }  
    return message;  
}
```



Step 2 Next, we'll want to consider how we'll handle each direction (L, M, or R)

- * If 'L', we should follow the → left pointer,
- * If 'M', we should follow the → middle pointer,
- * If 'R', we should follow the → right pointer.

Accordingly, we may a node pointer that starts at the root, and descends down the tree.



```
string decodeSequence (TernaryTreeNode * root,  
                      string pathSequence) {
```

```
    string message = "";
```

```
    TernaryTreeNode * messenger = root;  
    for (char direction : pathSequence) {
```

```
        if (direction == 'l') {
```

```
            messenger = messenger → left;
```

```
        } else if (direction == 'm') {
```

```
            messenger = messenger → middle;
```

```
        } else { // direction is 'r'
```

```
            messenger = messenger → right;
```

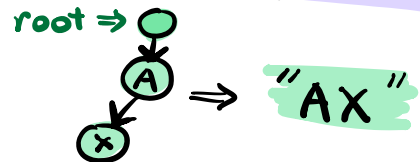
```
        }  
        // keep going with next steps!
```

```
    }  
    return message;
```

messenger
trying to climb
down



Step 3 Next, we'll want to update the message
for each of the nodes we visit. One question
is: should we do this before or after the direction
movement in the for loop?



Answer: This is tricky! we want to skip the root
node (no character), so move first, then add the
next node's character to the message string!

```
string decodeSequence(TernaryTreeNode * root,
                     string pathSequence) {
```

```
    string message = "";
```

```
    TernaryTreeNode * messenger = root;
```

```
    for (char direction : pathSequence) {
```

```
        if (direction == 'l') {
```

messenger
trying to climb
down

```
            messenger = messenger -> left;
```

```
        } else if (direction == 'm') {
```

```
            messenger = messenger -> middle;
```

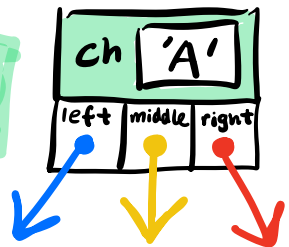
```
        } else { // direction is 'r'
```

```
            messenger = messenger -> right;
```

```
        }
```

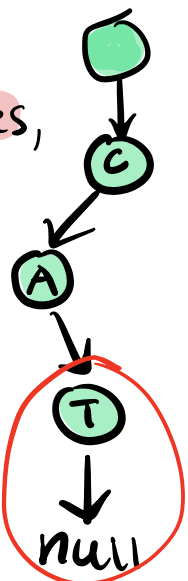
```
        message += messenger -> ch;
```

// almost there!



```
    }
    return message;
}
```

Step 4 Lastly, we'll want to consider edge cases,
i.e. for this problem, after moving left, middle,
or right, we should check if we reached a
dead end / nullptr, and stop iterating if so!



Solution Code ✓

```
string decodeSequence(TernaryTreeNode * root,  
                      string pathSequence) {
```

```
    string message = "";
```

```
    TernaryTreeNode * messenger = root;  
    for (char direction : pathSequence) {
```

```
        if (direction == 'l') {
```

messenger
trying to climb
down

```
            messenger = messenger -> left;
```

```
        } else if (direction == 'm') {
```

```
            messenger = messenger -> middle;
```

```
        } else { // direction is 'r'
```

```
            messenger = messenger -> right;
```

```
        }
```

```
        if (messenger == nullptr) {
```

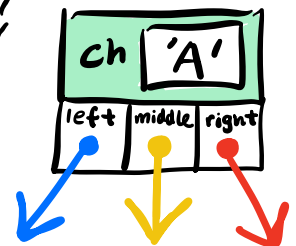
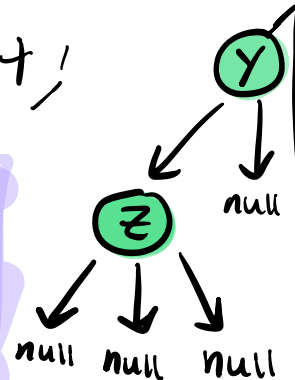
```
            break; // don't kill the messenger  
                  // so stop looping & return message
```

```
        }
```

```
        message += messenger -> ch;
```

```
    }  
    return message;
```

```
}
```



We have a full implementation of decodeText Yay!

