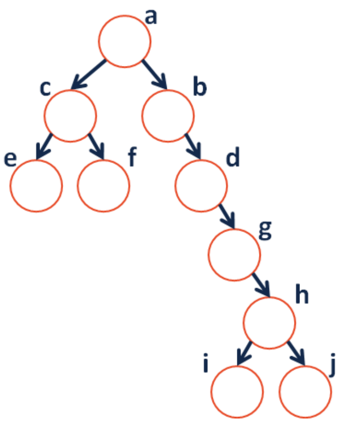
# #9: Trees

Review common tree terminology with the following exercises:

- What's the longest **English word** you can make using the **vertex** labels in the tree (repeats allowed)?
- Find an **edge** that is not on the longest **path** in the tree. Give that edge a reasonable name.
- One of the vertices is called the **root** of the tree. Which one?
- How many parents does each vertex have?
- Which vertex has the fewest **children**?
- Which vertex has the most **ancestors**?
- Which vertex has the most **descendants**?
- List all the vertices is b's left **subtree**.
- List all the **leaves** in the tree.



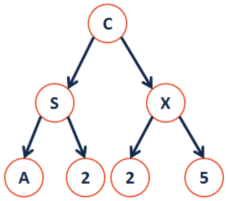
## Definition: Binary Tree

A *binary tree* **T** is either:

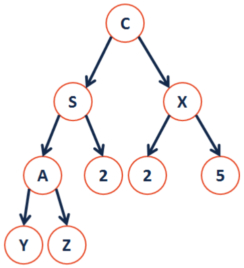
## Tree Property: Tree Height

## Tree Property: Full

## Tree Property: Perfect



## Tree Property: Complete



## Towards a Tree Implementation – Tree ADT:

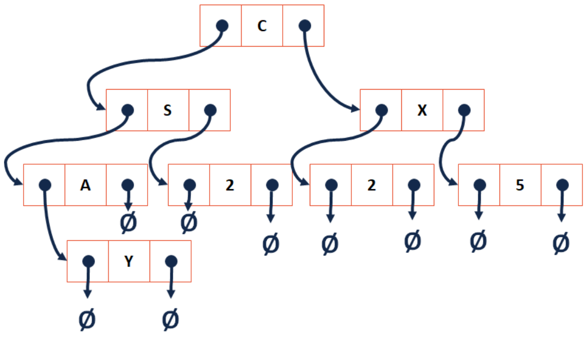
| **ADT Functionality**<br>**(English Description)** | **Function Call** |
|---|---|
| | |
| | |
| | |

## Tree Class

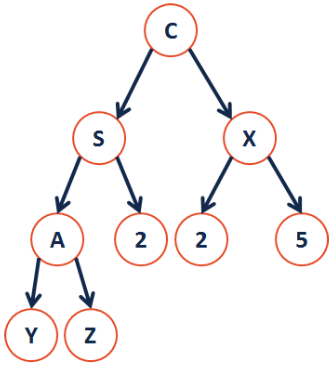
**BinaryTree.h**

```
#pragma once

template <typename T>
class BinaryTree {
  public:
    /* ... */
  private:







};
```

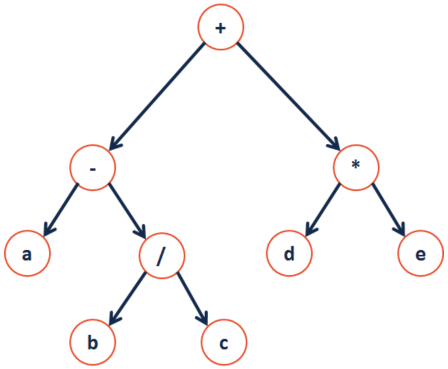
## Trees are nothing new – they're fancy linked lists:



**Theorem:** If there are n data items in our representation of a binary tree, then there are ______________ NULL pointers.



## Traversals:

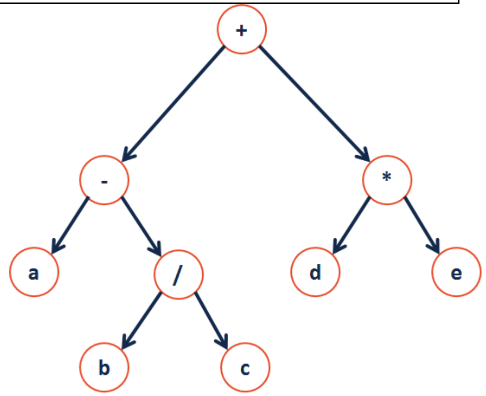


## One Algorithm, Three Traversals:

```
BinaryTree.cpp
void BinaryTree<T>::____Order(TreeNode * cur) {
  if (cur != nullptr) {





  }
}
```

## A Different Type of Traversal

Strategy:



```
BinaryTree.cpp
void BinaryTree<T>::levelOrder(TreeNode * croot) {









}
```