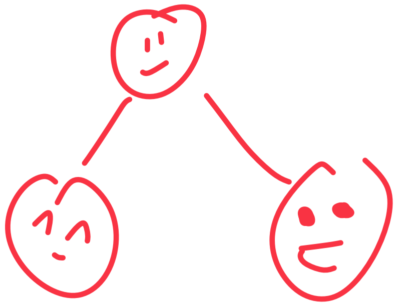
where did the internet go? :(

Data Structures
Tree Definitions

CS 225
Brad Solomon

September 16, 2023



Department of Computer Science

Exam 1 (9/18 — 9/20)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam will be released on PL

Topics covered can be found on website

Registration started August 22

https://courses.engr.illinois.edu/cs225/fa2024/exams/

mp_lists released! (Due September 30th)

MP submission on PL has two separate submissions

The extra credit portion will only test part 1

Completion of the extra credit portion by the following Monday is worth 8 points

Learning Objectives

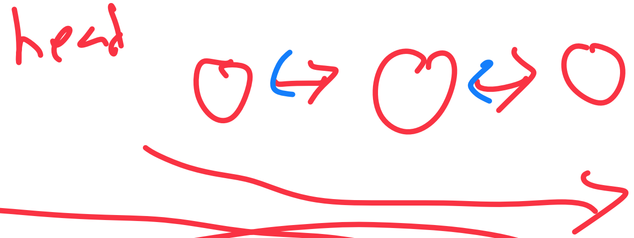
Review trees and binary trees

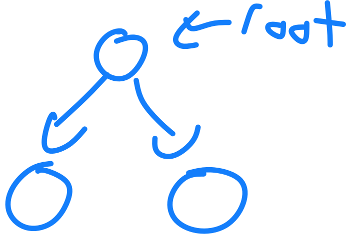
Practice tree theory with recursive definitions and proofs

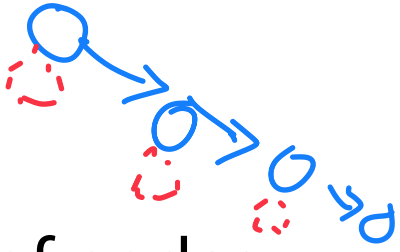
Discuss the tree ADT

Explore tree implementation details ? ? ?

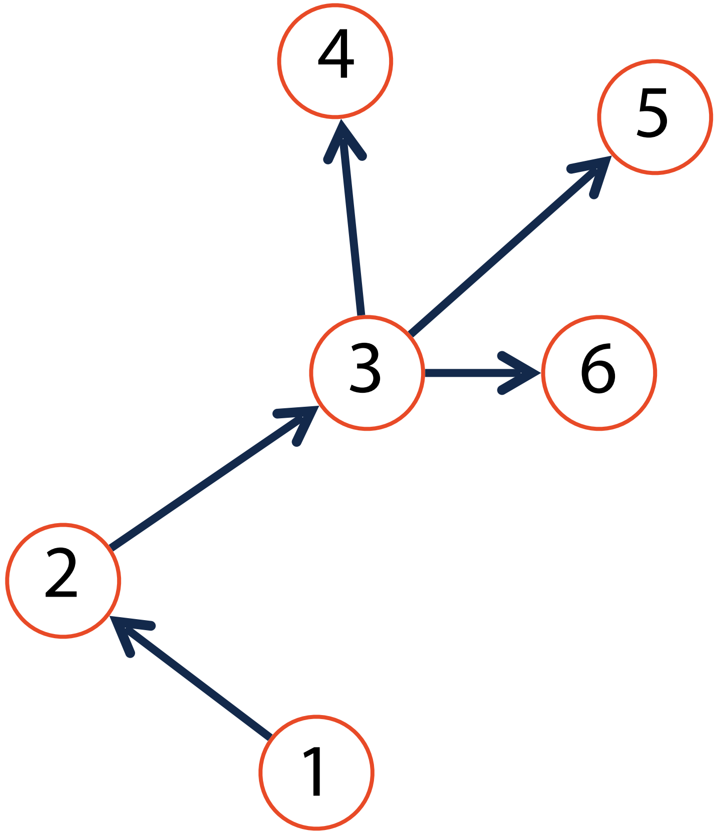
Trees







A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.



[In CS 225] a tree is also:

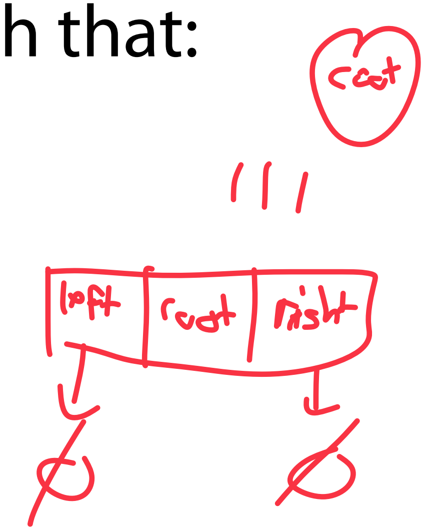
1) Acyclic — No path from node to itself

2) Rooted — A specific node is labeled root

Binary Tree

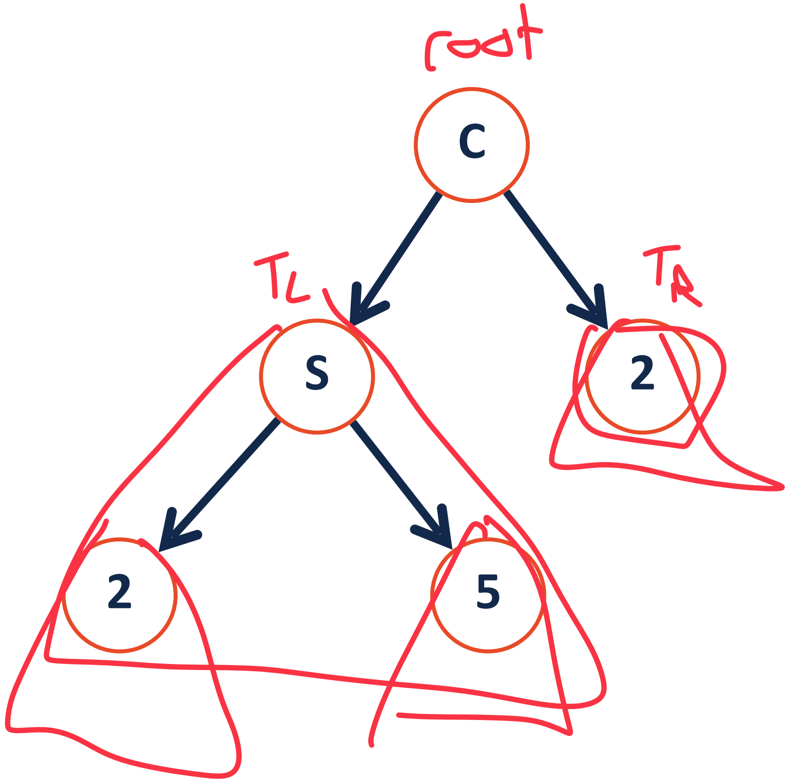
A **binary tree** is a tree T such that:

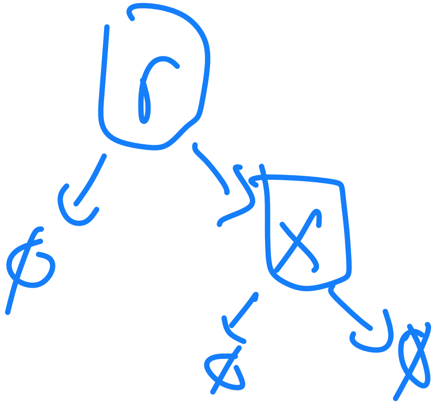
1. $T = \emptyset$



2. $T = (\overset{\text{root}}{\cancel{data}}, T_L, T_R)$

$\hookrightarrow T = (r, \emptyset, (x, \emptyset, \emptyset))$





Binary Tree

Lets define additional terminology for different **types** of binary trees!

1. Full

2. Perfect

3. Complete

Binary Tree: full

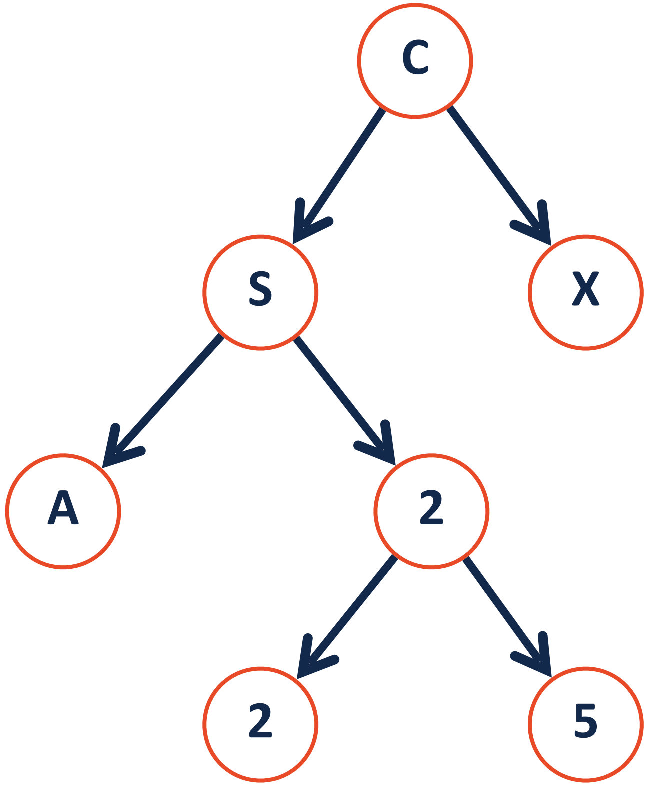
A **full tree** is a binary tree where every node has either 0 or 2 children

A tree **F** is **full** if and only if:

1. $F = \emptyset$ empty tree

2. $F = (root, \emptyset, \emptyset)$ No child

3. $F = (root, F_L \neq \emptyset, F_R \neq \emptyset)$ 2 child



Binary Tree: full

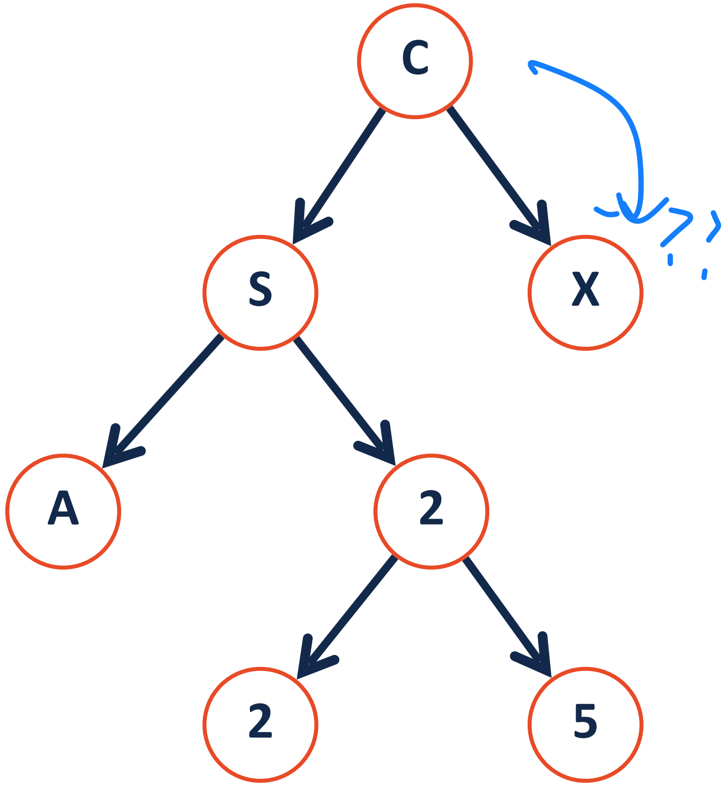
A **full tree** is a binary tree where every node has either 0 or 2 children

A tree **F** is **full** if and only if:

1. $F = \emptyset$

2. $F = (\overset{\text{root}}{\overset{\shortparallel}{data}}, \emptyset, \emptyset)$

3. $F = (data, F_l \neq \emptyset, F_r \neq \emptyset)$



Binary Tree: perfect

$0 \rightarrow 0 \rightarrow \emptyset$

A **perfect tree** is a binary tree where…

Every internal node has 2 children and all leaves are at the same level.

Tree height is future topic!

A tree **P** is **perfect** if and only if:

1. P_h = Perfect tree of height h
   $= (\text{root}, P_{h-1}, P_{h-1})$

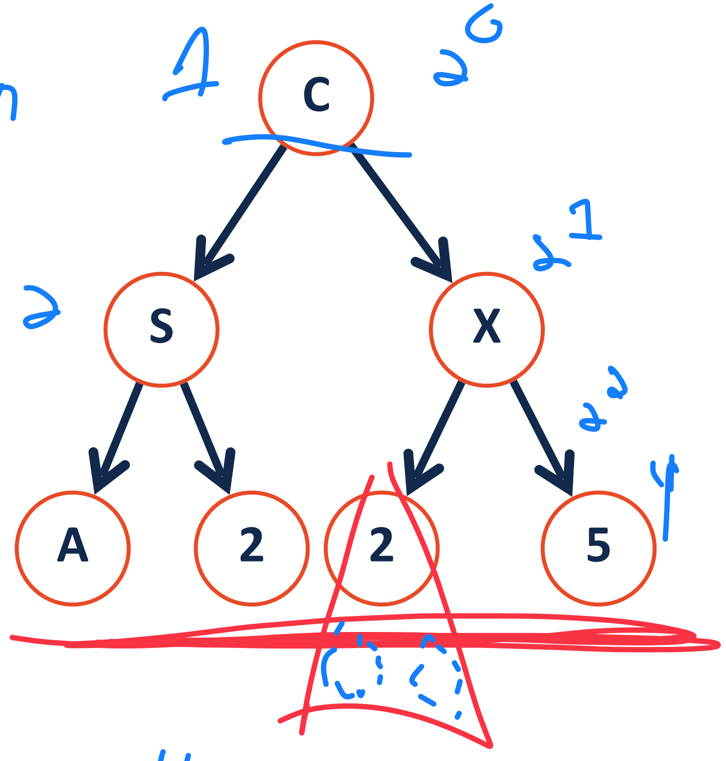
2. $P_0 = (\text{root}, \overset{L}{\emptyset}, \overset{R}{\emptyset})$ check if L & R are both null

   or

   or

   $P_{-1} = \emptyset$ ← stop when hit null

h = 2



depth → level height

$2^{h+1} - 1$ nodes

Binary Tree: perfect

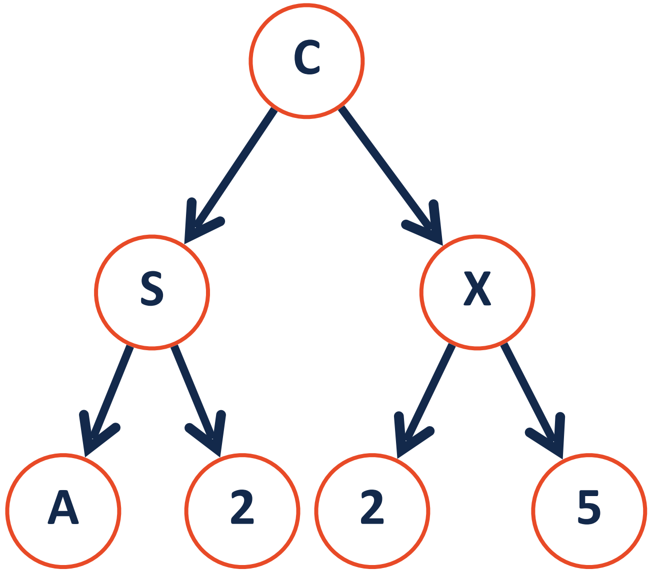
A **perfect tree** is a binary tree where…

Every internal node has 2 children and all leaves are at the same level.

A tree **P** is **perfect** if and only if:

1. $P_h = (data, P_{h-1}, P_{h-1})$

2. $P_0 = (data, \emptyset, \emptyset) \equiv \; P_{-1} = \emptyset$



Binary Tree: complete

A **complete tree** is a B.T. where…

All levels except the last are completely filled.

~~The last level contains at least one node~~ (and is pushed to left)

A tree **C** is **complete** if and only if:

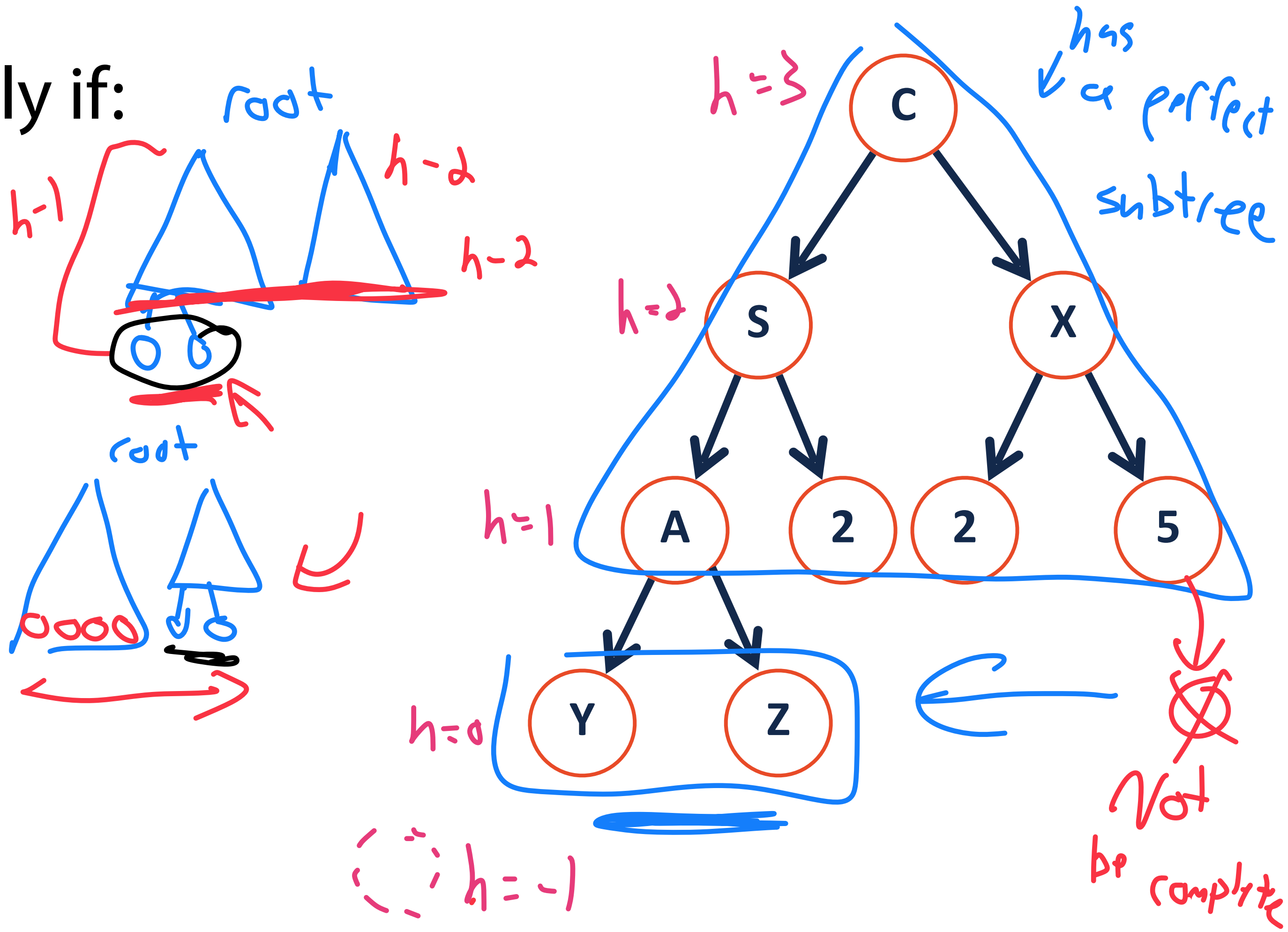
1. $C_h = (root, C_{h-1}, P_{h-2})$

↳ Last level in Subtree Left

2. $C_h = (root, P_{h-1}, C_{h-1})$

↳ Last level is perfect on left & filling in on right

3. $C_{-1} = \emptyset$



Binary Tree: complete

A **complete tree** is a B.T. where…

All levels except the last are completely filled.

The last level contains at least one node (and is pushed to left)

may

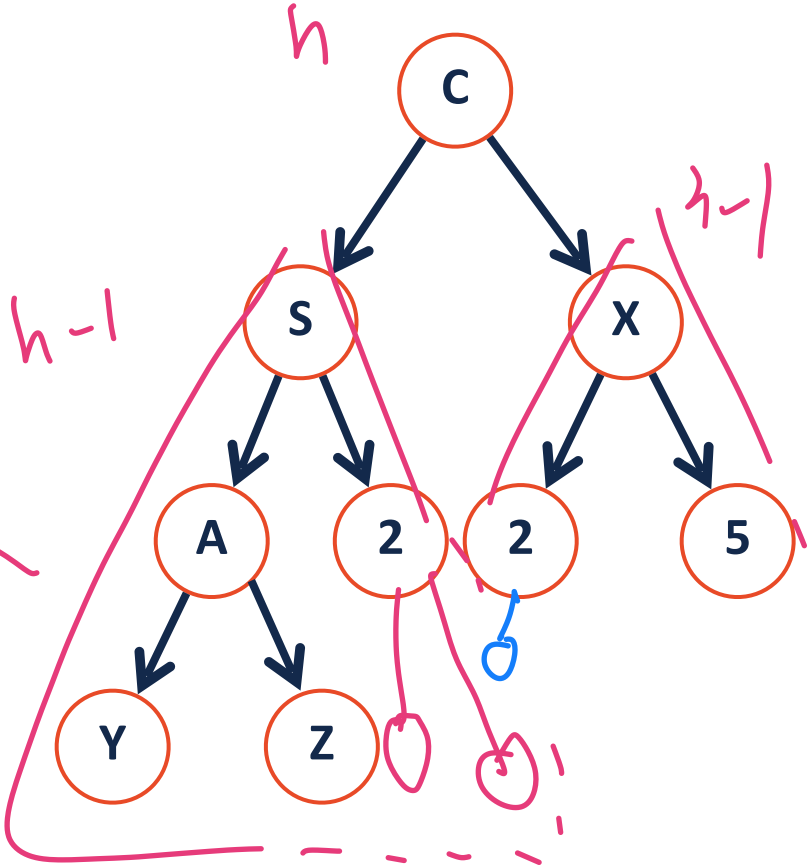
0 or more nodes

A tree **C** is **complete** if and only if:

1. $C_h = (data, C_{h-1}, P_{h-2})$

2. $C_h = (data, P_{h-1}, C_{h-1})$

3. $C_{-1} = \emptyset$



Binary Tree

Why do we care?

1. Terminology instantly defines a particular tree structure

   ↳ Some trees have exactly $2^{h+1}-1$ nodes (perfect tree)

   ↳ English def

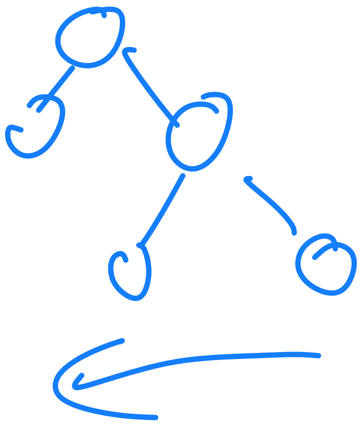
2. Understanding how to think ‘recursively’ is very important.

↳ Practice!

Binary Tree: Thinking with Types

Is every **full** tree **complete**?

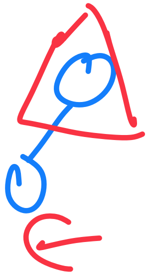
No



Not pushed to left

Is every **complete** tree **full**?

No



Binary Tree: Practicing Proofs

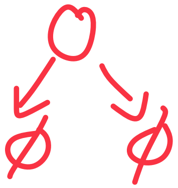
Theorem: If there are **n** objects in our representation of a binary tree, then there are $n + 1$ NULL pointers.

↳ Proof by induction

Base case: Prove this claim true for some fixed value

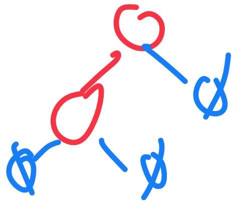
$n = 0$ root → nullptr 1

$n = 1$



2 null ptrs!

$n = 2$



3

Binary Tree: Practicing Proofs

Theorem: If there are **n** objects in our representation of a binary tree, then there are **n+1** NULL pointers.

Base Case:

Binary Tree: Practicing Proofs

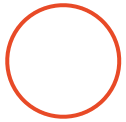
Theorem: If there are **n** objects in our representation of a binary tree, then there are **n+1** NULL pointers.

Base Case:

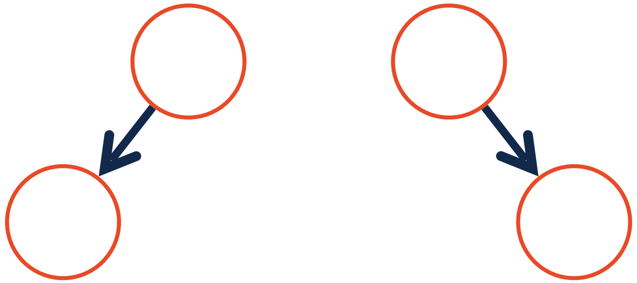
Let F(n) be the max number of NULL pointers in a tree of n nodes

N=0 has one NULL

N=1 has two NULL

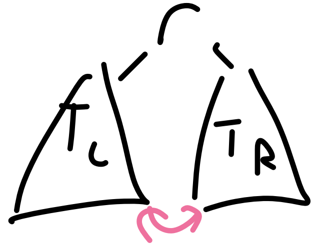


~~N=2 has three NULL~~



Theorem: If there are **n** objects in our representation of a binary tree, then there are **n+1** NULL pointers.

Induction Step: Assume claim is true for $n \leq k-1$ nodes. Prove true for k.



Let $T = \{r, T_L, T_R\}$ & Let q be the # of nodes in T_L

$0 \leq q \leq k-1$

By IH, T_L has $q+1$ nullptrs

T_R is all nodes not root & not T_L $\therefore$ $|T_R| = k - q - 1$

↳ By IH, T_R has $(k-q-1)+1$ nullptrs

Total nulls: $\overset{T_L}{(q+1)} + \overset{T_R}{(k-q)} = k+1$

Theorem: If there are **n** objects in our representation of a binary tree, then there are **n+1** NULL pointers.

IS: Assume claim is true for $|T| \le k - 1$**, prove true for** $|T| = k$

By def, $T = r, T_L, T_R$. Let q be the # of nodes in T_L

Since r exists, $0 \le q \le k - 1$. By IH, T_L has $q + 1$ NULL

All nodes not in r or T_L exist in T_R. So T_R has $k \underline{-} \overset{TL}{\underline{q}} - \overset{\text{root}}{\underline{1}}$ nodes

$k - q - 1$ is also smaller than k so by IH, T_R has $k - q$ NULL

Total number of NULL is the sum of T_L and T_R: $q + 1 + k - q = \boxed{k + 1}$

Tree ADT

Insert

Remove

Traverse

Find

Constructor

BinaryTree.h

```
#pragma once

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

  private:
















};
```

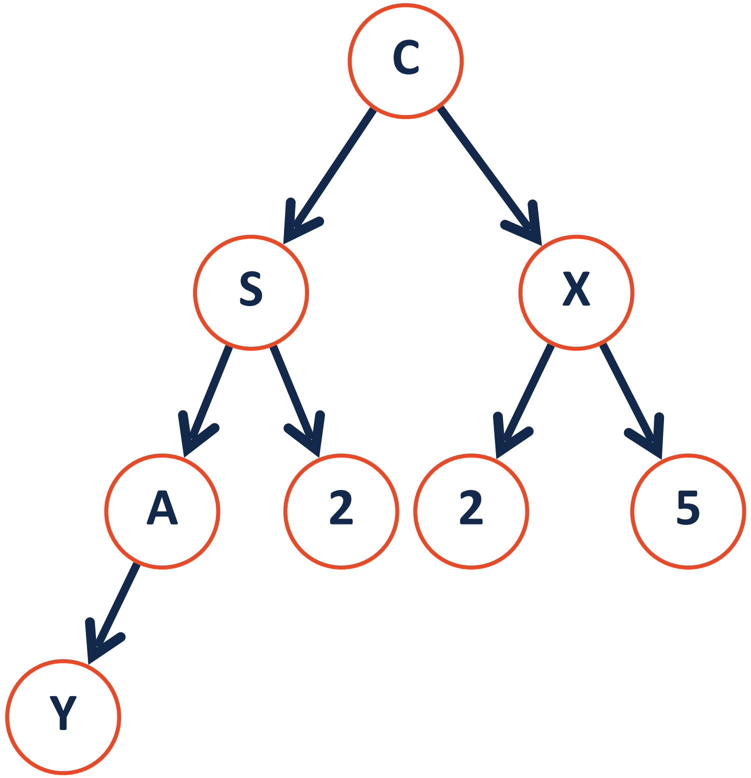
List.h

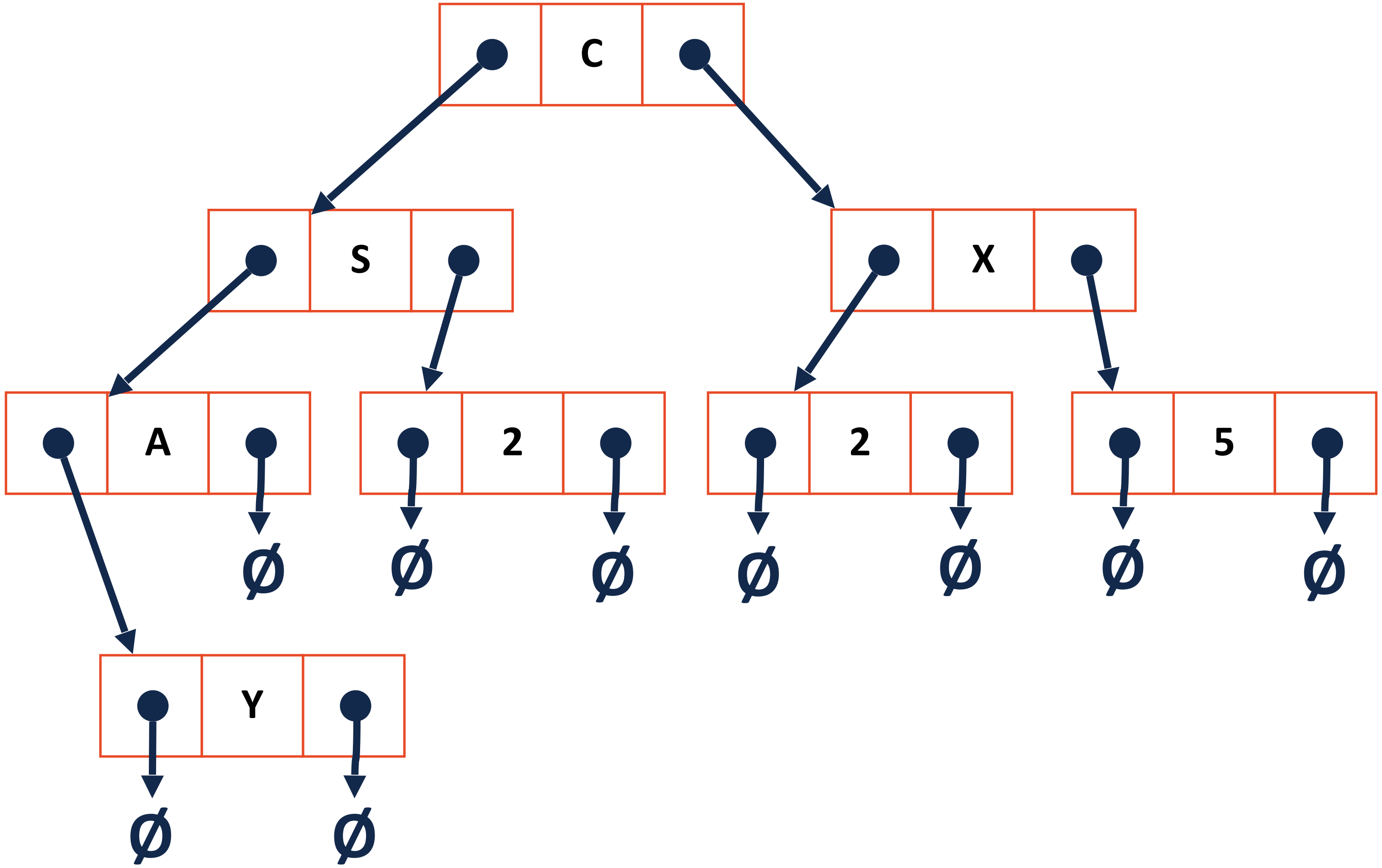
```
#pragma once

template <typename T>
class List {
  public:
    /* ... */
  private:
    class ListNode {
      T & data;

      ListNode * next;



      ListNode(T & data) :
       data(data), next(NULL) { }
    };



    ListNode *head_;
    /* ... */
};
```

Tree.h

```
#pragma once

template <typename T>
class BinaryTree {
  public:
    /* ... */
  private:
    class TreeNode {
      T & data;

      TreeNode * left;

      TreeNode * right;

      TreeNode(T & data) :
       data(data), left(NULL),
right(NULL) { }

    };

    TreeNode *root_;
    /* ... */
};
```

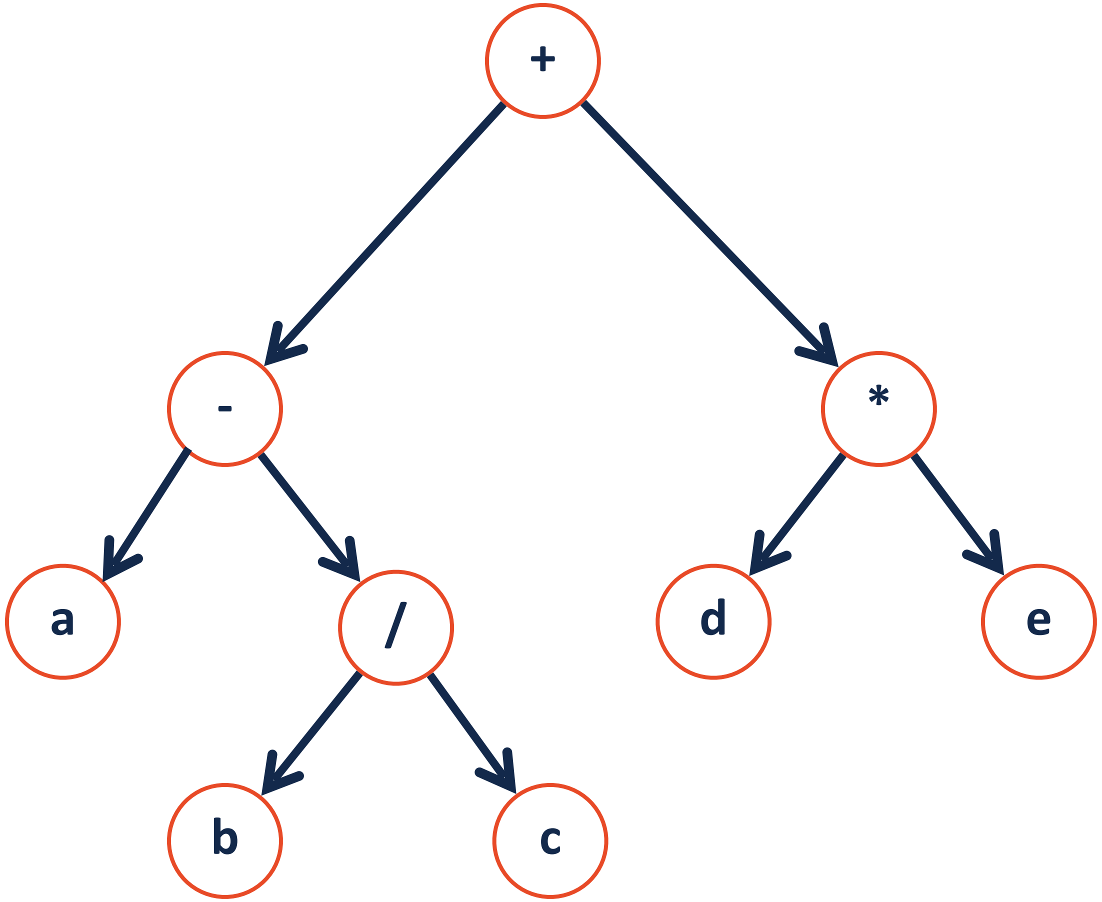
Visualizing trees



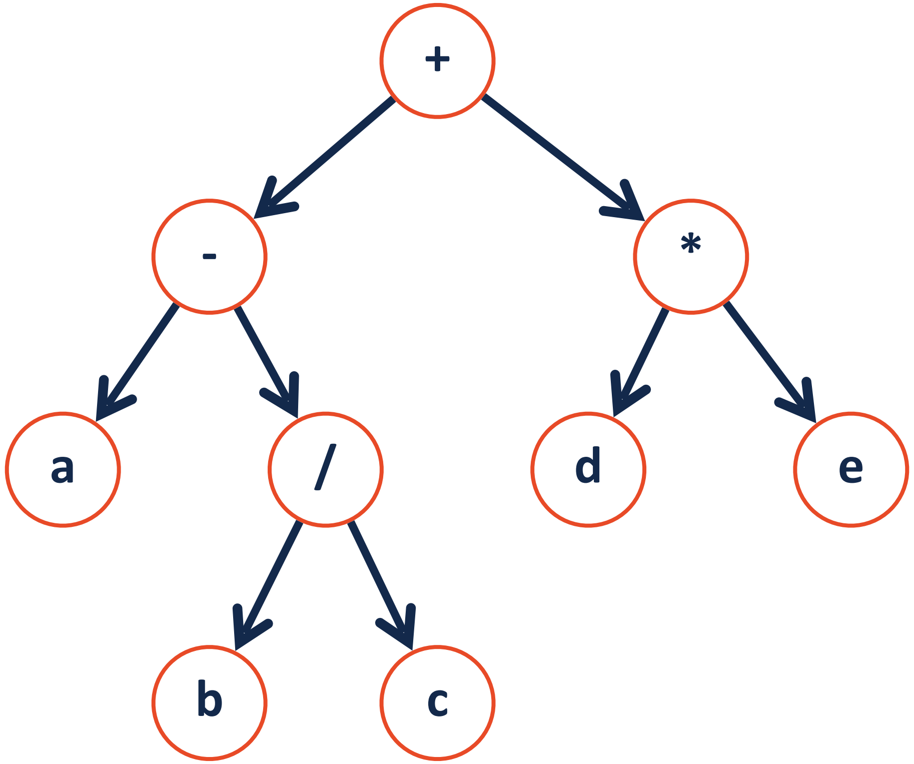


Tree Traversal

A **traversal** of a tree T is an ordered way of visiting every node once.



Traversals



```
template<class T>
void BinaryTree<T>::_____Order(TreeNode * root)
{


















}
```