

Data Structures

Tree Traversal

CS 225

September 15, 2023

Brad Solomon & G Carl Evans



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

State of the class

Exam scores generally match expected trends!

Exam 0: 77%

Exam 1: 83%

Plagiarism is at an 'all-time' low!

Remember — do not share code!

State of assessments

Lab_debug

44% of class found lecture helpful for completing lab

61% of students found attending lab helpful

63% of students found lab improved confidence in CS

Labs better as group assignments

Lab should have set-up instructions

Unclear instructions

Extra Credit Reminder

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

Queue Usage Reminder

Please include meaningful topic

Please include both your name and Discord username

Please be respectful to course staff



Learning Objectives

Discuss the tree ADT

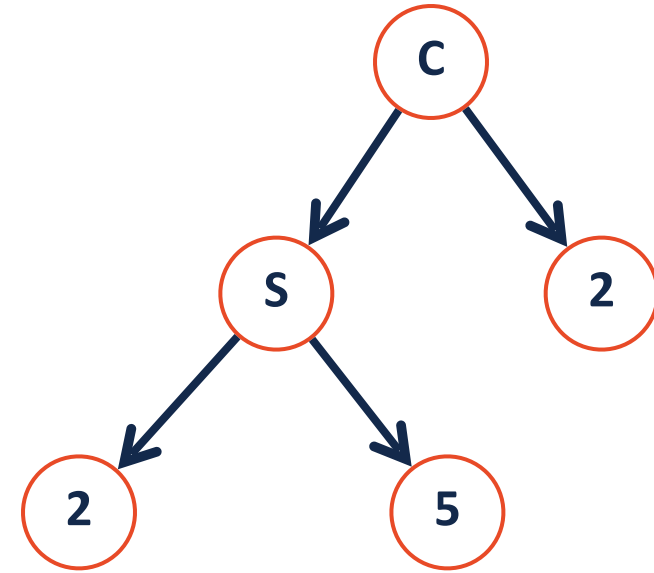
Explore tree implementation details

Binary Tree

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

1. $T = \emptyset$

2. $T = (data, T_L, T_R)$





Tree ADT

Insert

Remove

GetData

Traverse

Constructor

BinaryTree.h

```
1 #pragma once
2
3 template <class T>
4 class BinaryTree {
5     public:
6         /* ... */
7
8     private:
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25 };
```

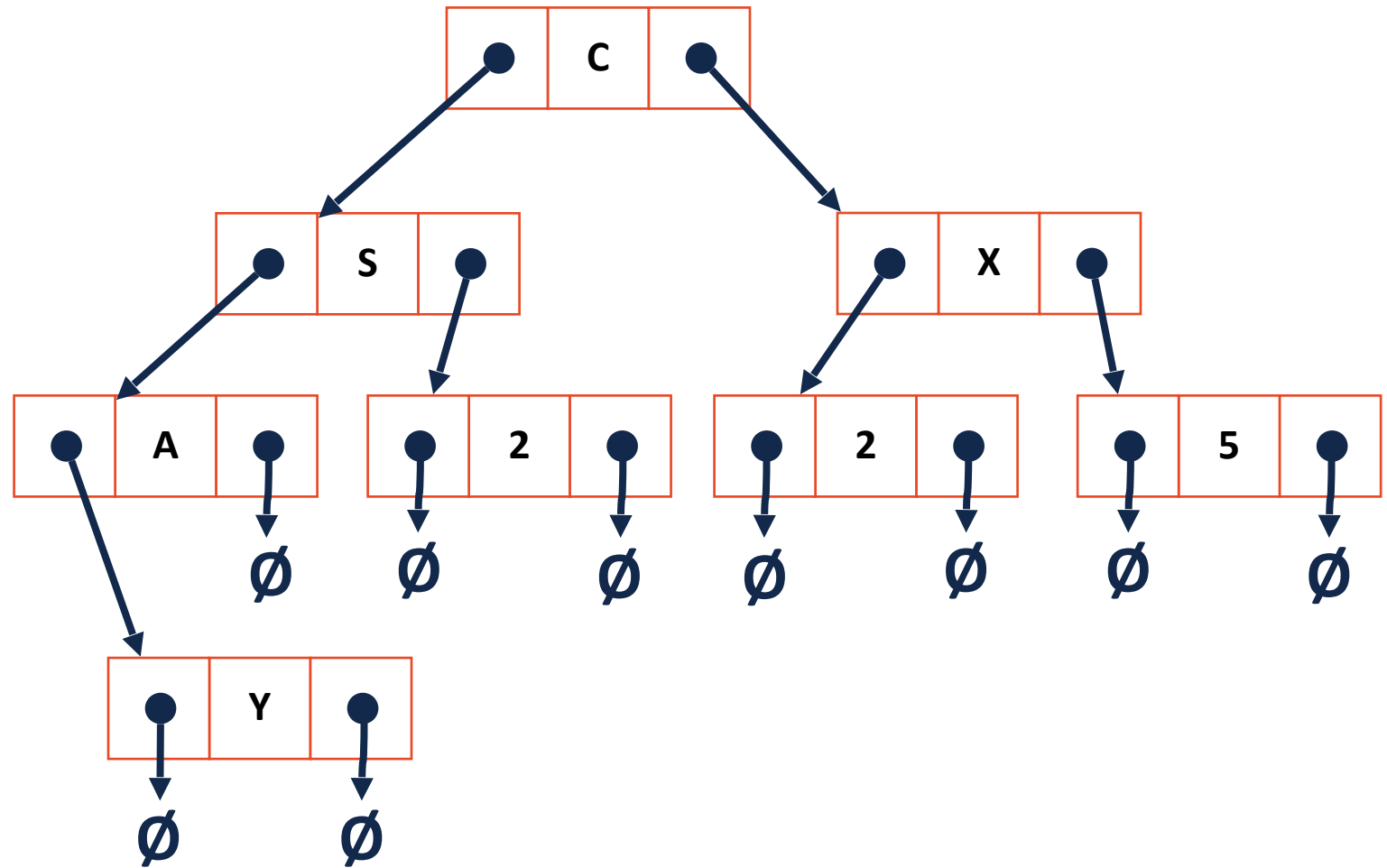
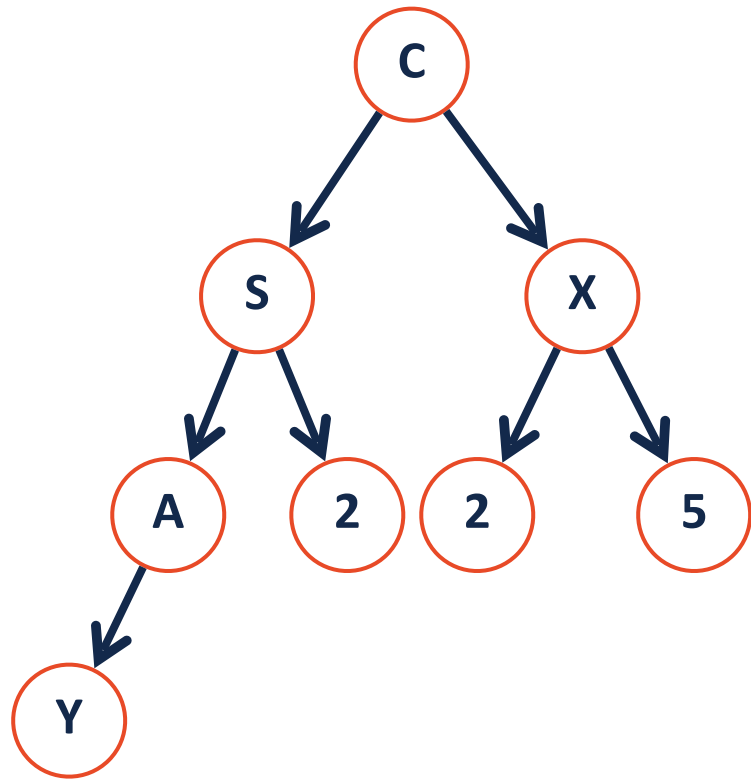
List.h

```
1 #pragma once
2
3 template <typename T>
4 class List {
5     public:
6         /* ... */
7     private:
8         class ListNode {
9             T & data;
10
11             ListNode * next;
12
13
14             ListNode(T & data) :
15                 data(data), next(NULL) { }
16         };
17
18
19         ListNode *head_;
20         /* ... */
21 };
```

Tree.h

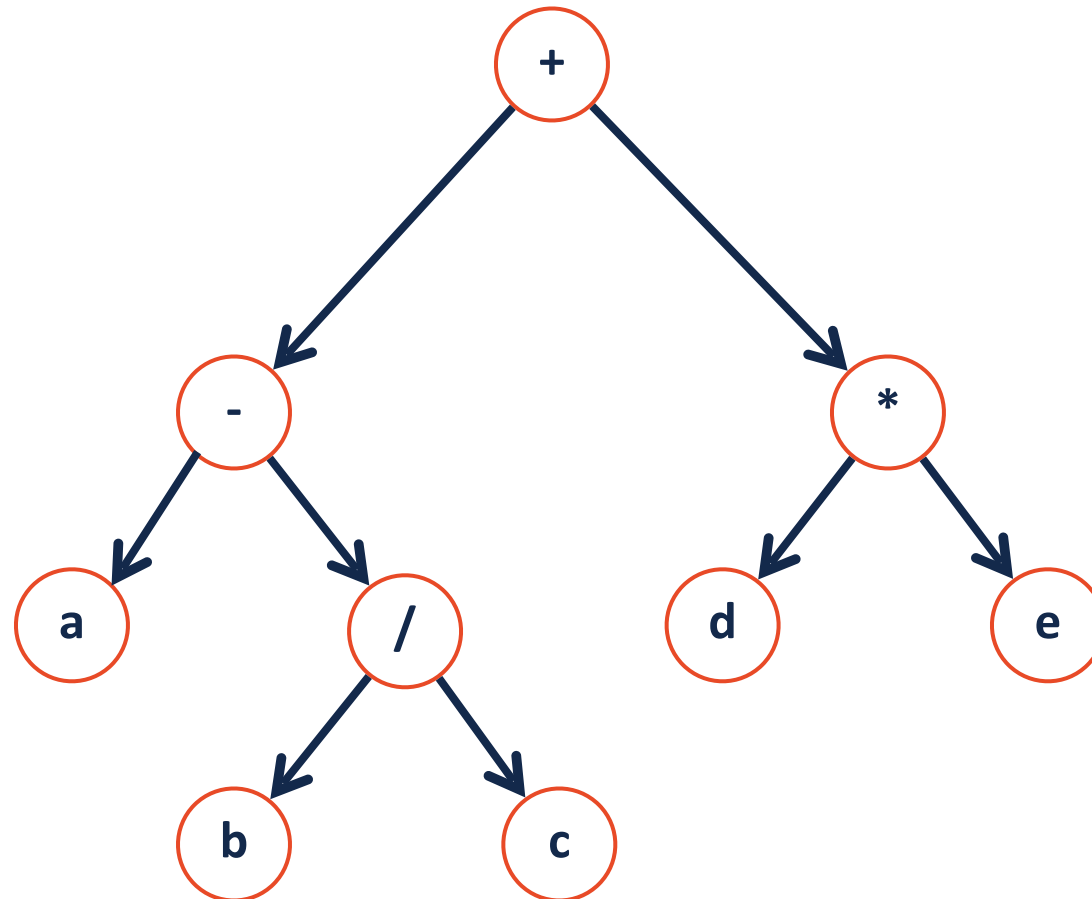
```
1 #pragma once
2
3 template <typename T>
4 class BinaryTree {
5     public:
6         /* ... */
7     private:
8         class TreeNode {
9             T & data;
10
11             TreeNode * left;
12
13             TreeNode * right;
14
15             TreeNode(T & data) :
16                 data(data), left(NULL),
17                 right(NULL) { }
18         };
19
20         TreeNode *root_;
21         /* ... */
22 };
```

Visualizing trees

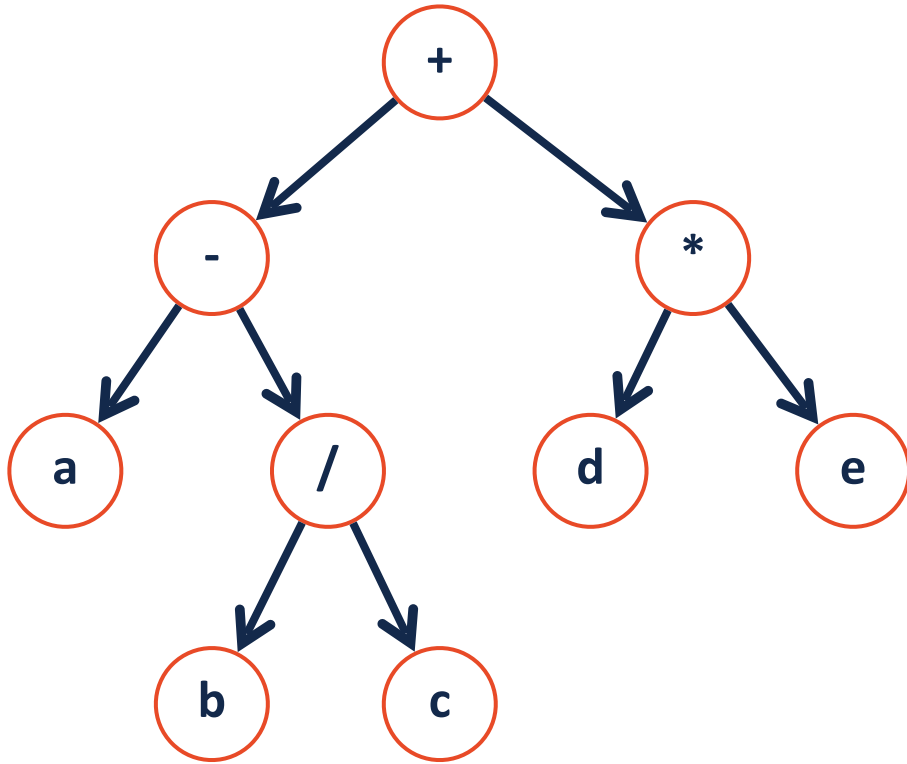


Tree Traversal

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

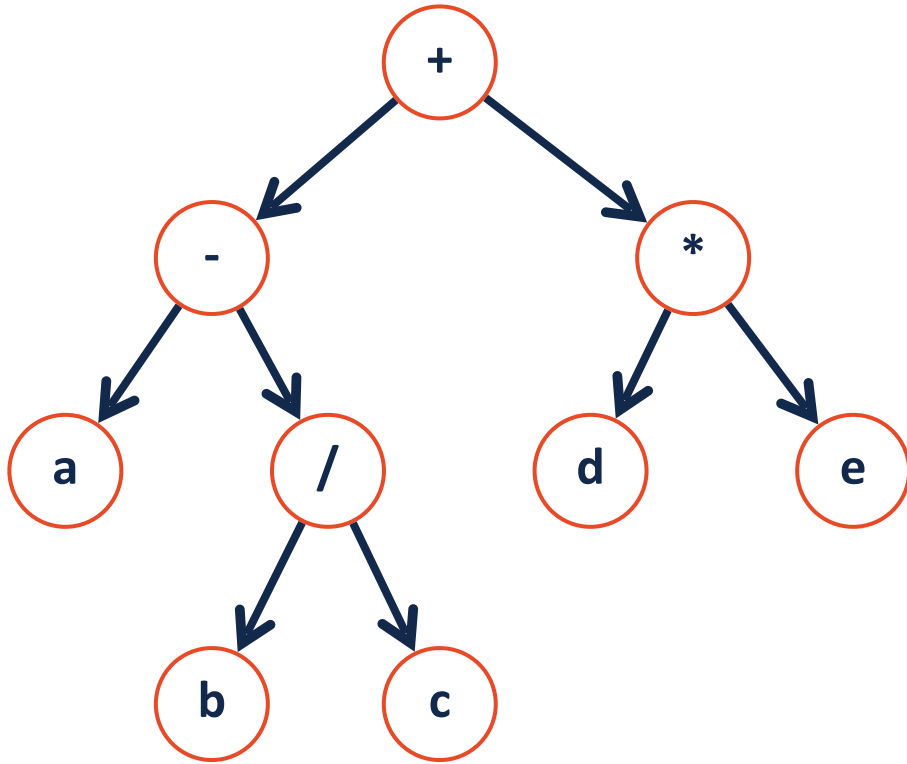


Traversals



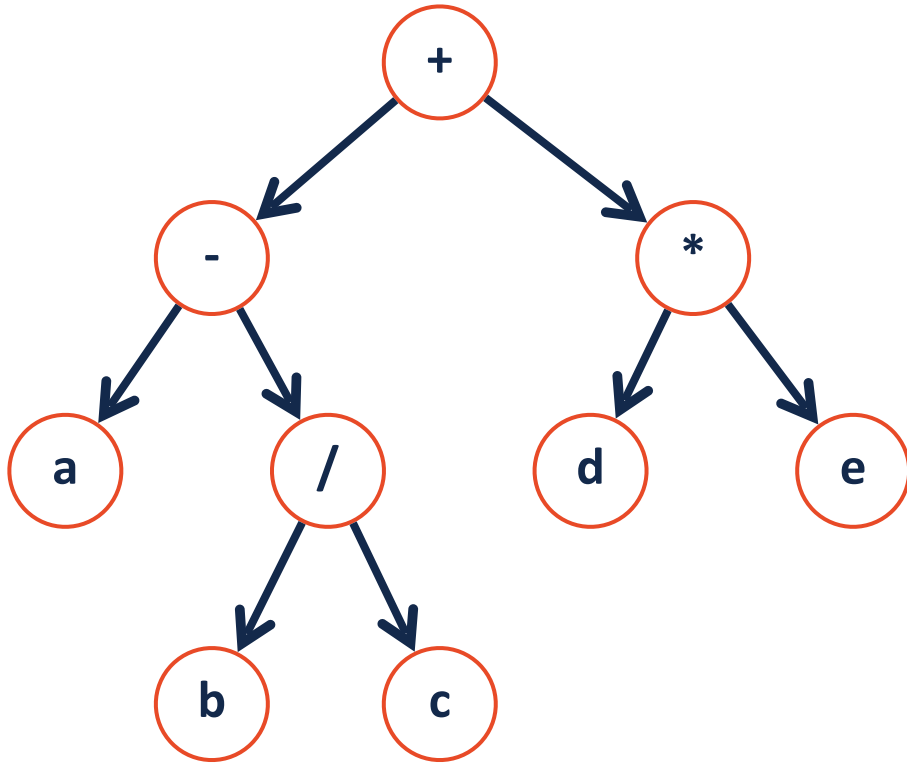
```
1 template<class T>
2 void BinaryTree<T>::_____Order(TreeNode * root)
3 {
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Traversals



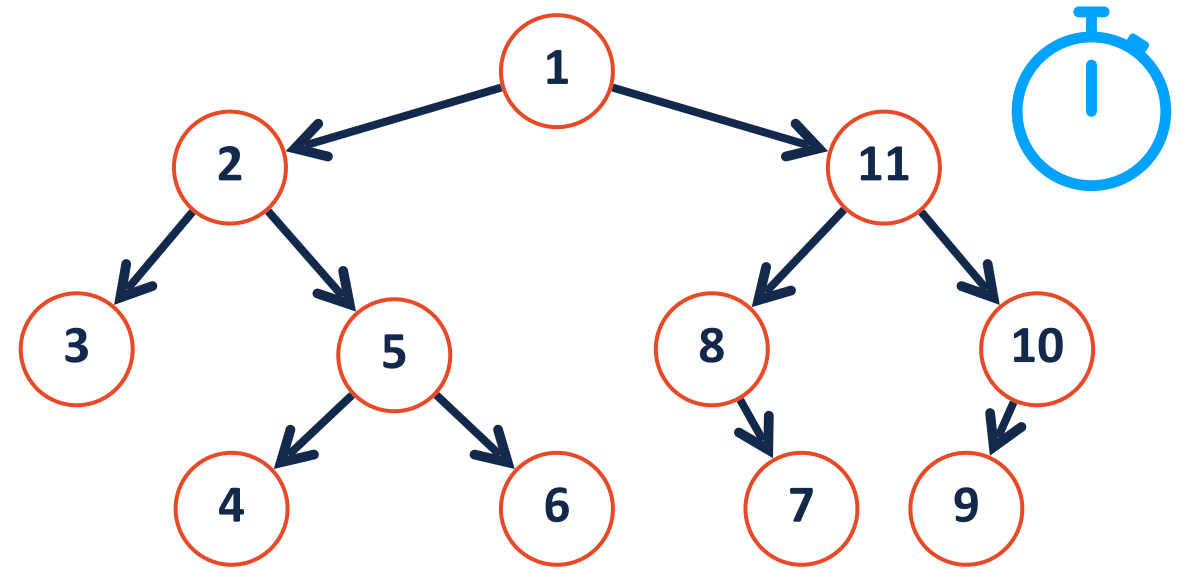
```
1 template<class T>
2 void BinaryTree<T>::_____Order(TreeNode * root)
3 {
4
5     if (root) {
6
7         _____;
8
9         _____Order(root->left);
10
11        _____;
12
13        _____Order(root->right);
14
15        _____;
16
17     }
18
19
20
21 }
```

Traversals



```
1 template<class T>
2 void BinaryTree<T>::_____Order(TreeNode * root)
3 {
4
5     if (root) {
6
7         _____;
8
9         _____Order(root->left);
10
11        _____;
12
13        _____Order(root->right);
14
15        _____;
16
17     }
18
19
20
21 }
```

Tree Traversals



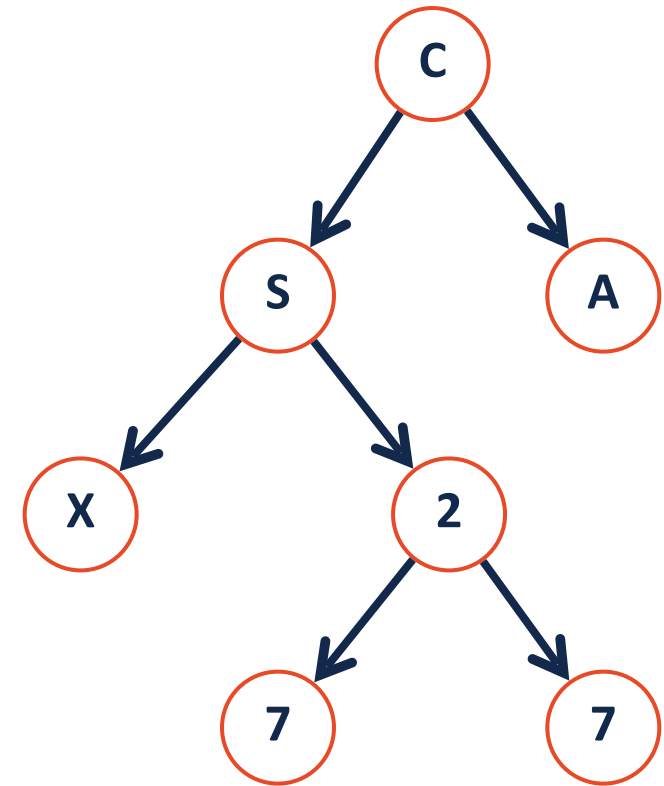
Pre-order:

In-order:

Post-order:

Tree Traversals

Pre-order: Ideal for copying trees

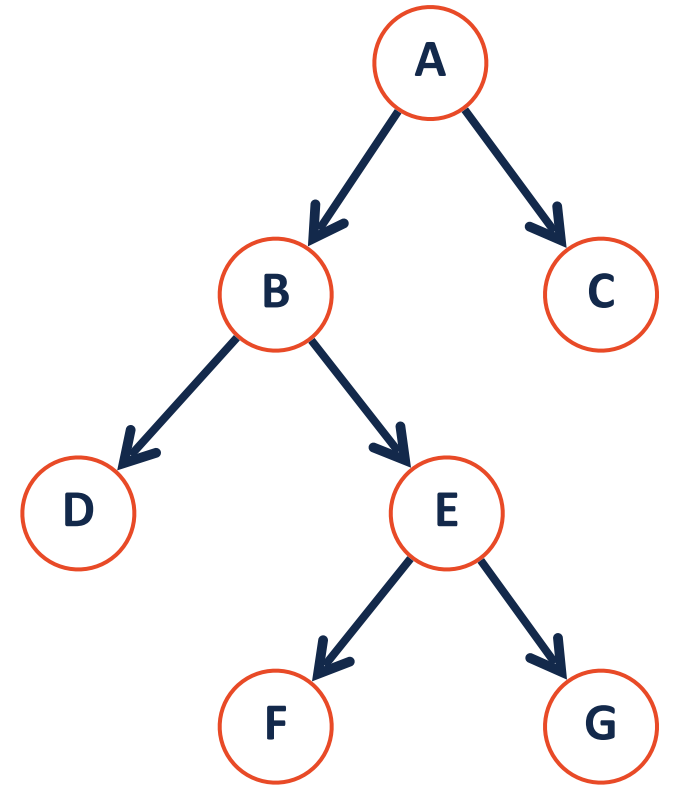


Post-order: Ideal for deleting trees

Traversal vs Search

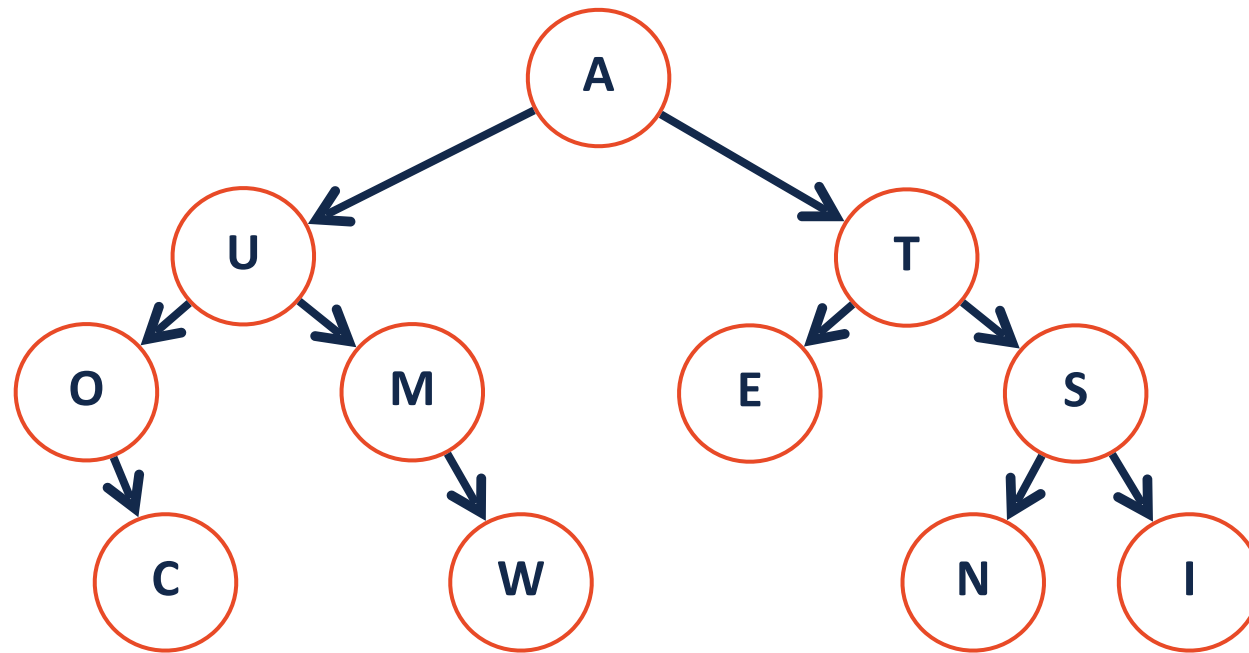
Traversal

Search



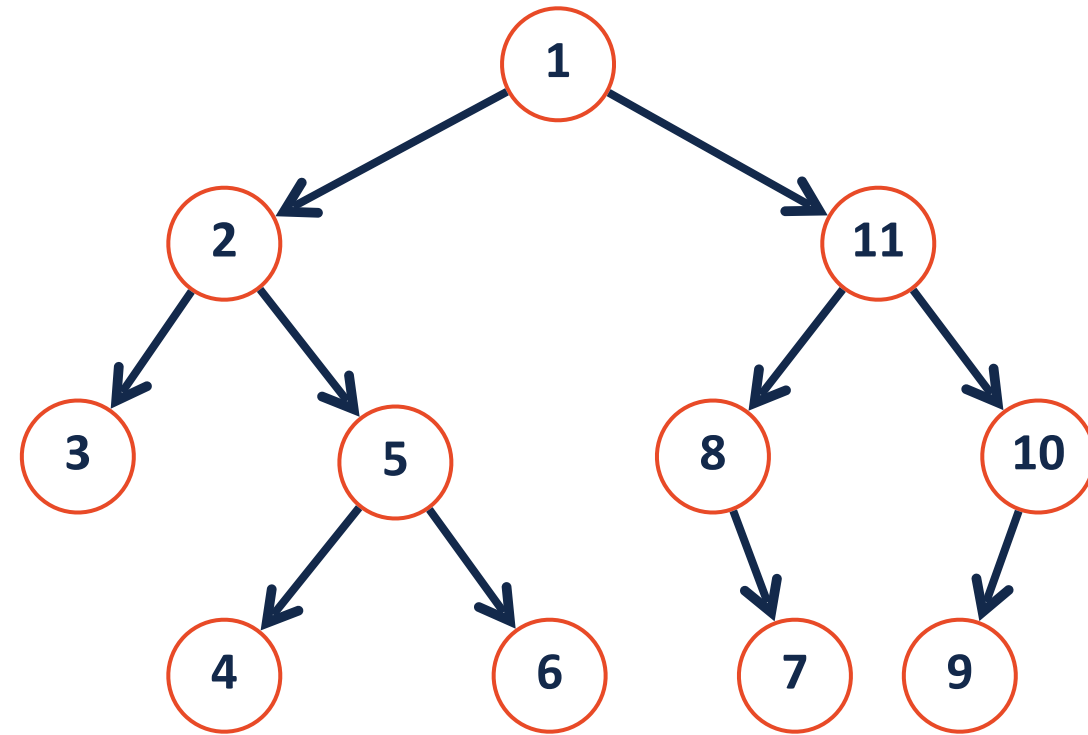
Tree Search

There are two main approaches to searching a binary tree:



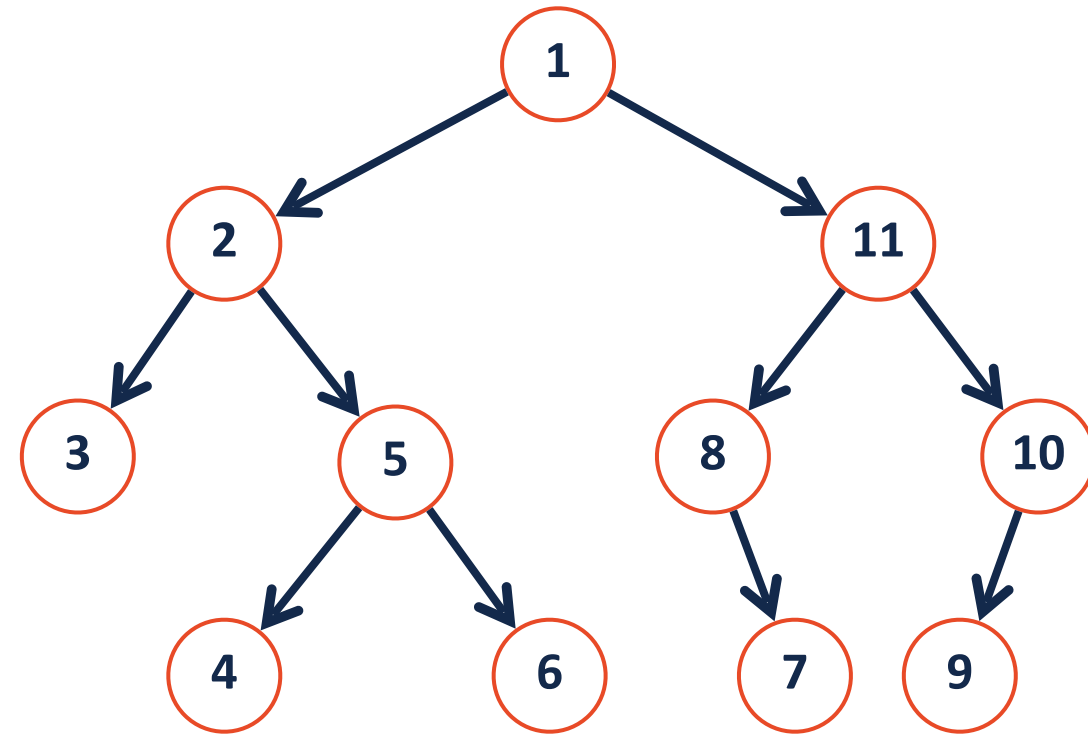
Depth First Search

Explore as far along one path as possible before backtracking

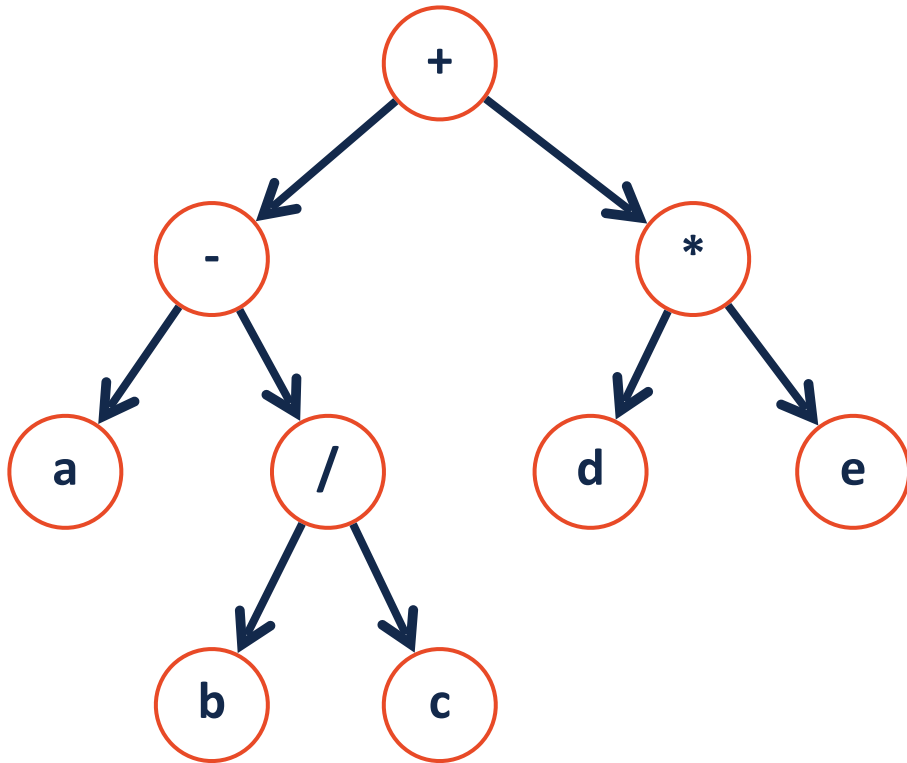


Breadth First Search

Fully explore depth i before exploring depth $i+1$



Level-Order Traversal



```
1 template<class T>
2 void BinaryTree<T>::lOrder(TreeNode * root)
3 {
4
5     Queue<TreeNode*> q;
6     q.enqueue(root);
7
8     while( q.empty() == False){
9
10        TreeNode* temp = q.head();
11        process(temp);
12
13        q.dequeue();
14
15        q.enqueue(temp->left);
16        q.enqueue(temp->right);
17
18    }
19 }
```

Tree Search

How can we improve our ability to search a binary tree?

What do we trade in order to do so?