Data Structures

Tree Traversal

CS 225

Brad Solomon

September 18, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

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

Learning Objectives

Discuss the tree ADT

Explore tree implementation details

Tree ADT

Insert

Remove

Traverse

Find

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 };
```

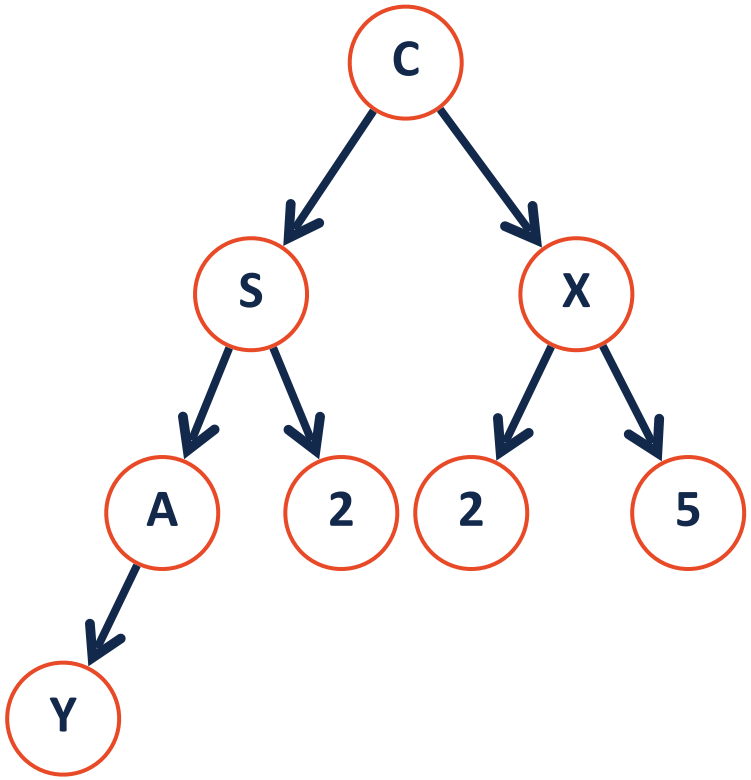
List.h

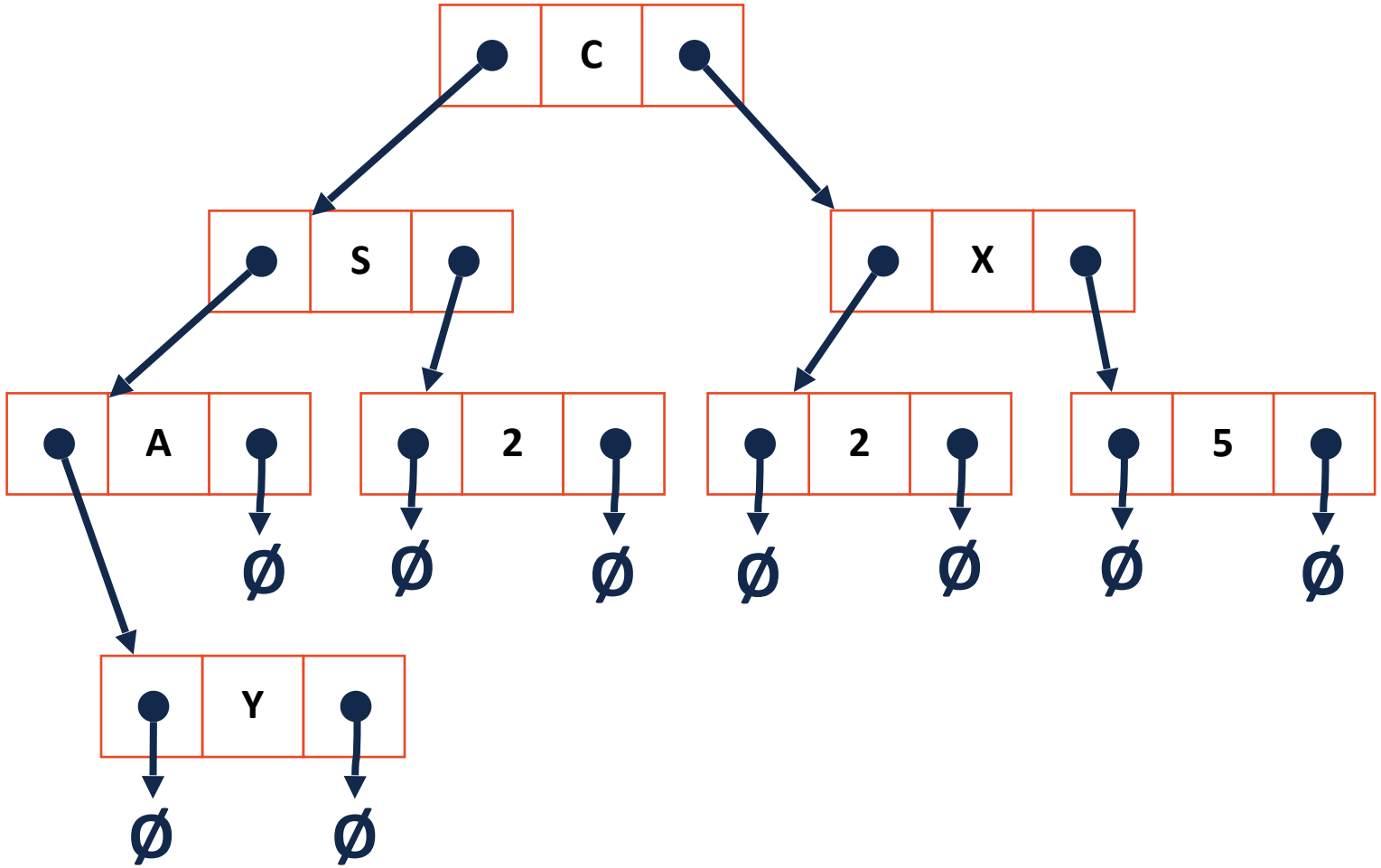
```
1  #pragma once
2
3  template <typename T>
4  class List {
5    public:
6      /* ... */
7    private:
8      class ListNode {
9        T & data;
10
11       ListNode * next;
12
13
14
15       ListNode(T & data) :
16        data(data), next(NULL) { }
17     };
18
19
20
21     ListNode *head_;
22     /* ... */
23 };
```

Tree.h

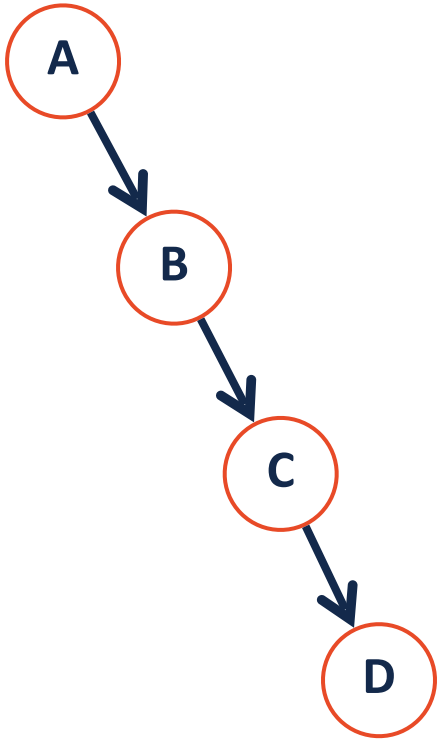
```
1  #pragma once
2
3  template <typename T>
4  class BinaryTree {
5    public:
6      /* ... */
7    private:
8      class TreeNode {
9        T & data;
10
11       TreeNode * left;
12
13       TreeNode * right;
14
15       TreeNode(T & data) :
16        data(data), left(NULL),
17 right(NULL) { }
18
19     };
20
21     TreeNode *root_;
22     /* ... */
23 };
```

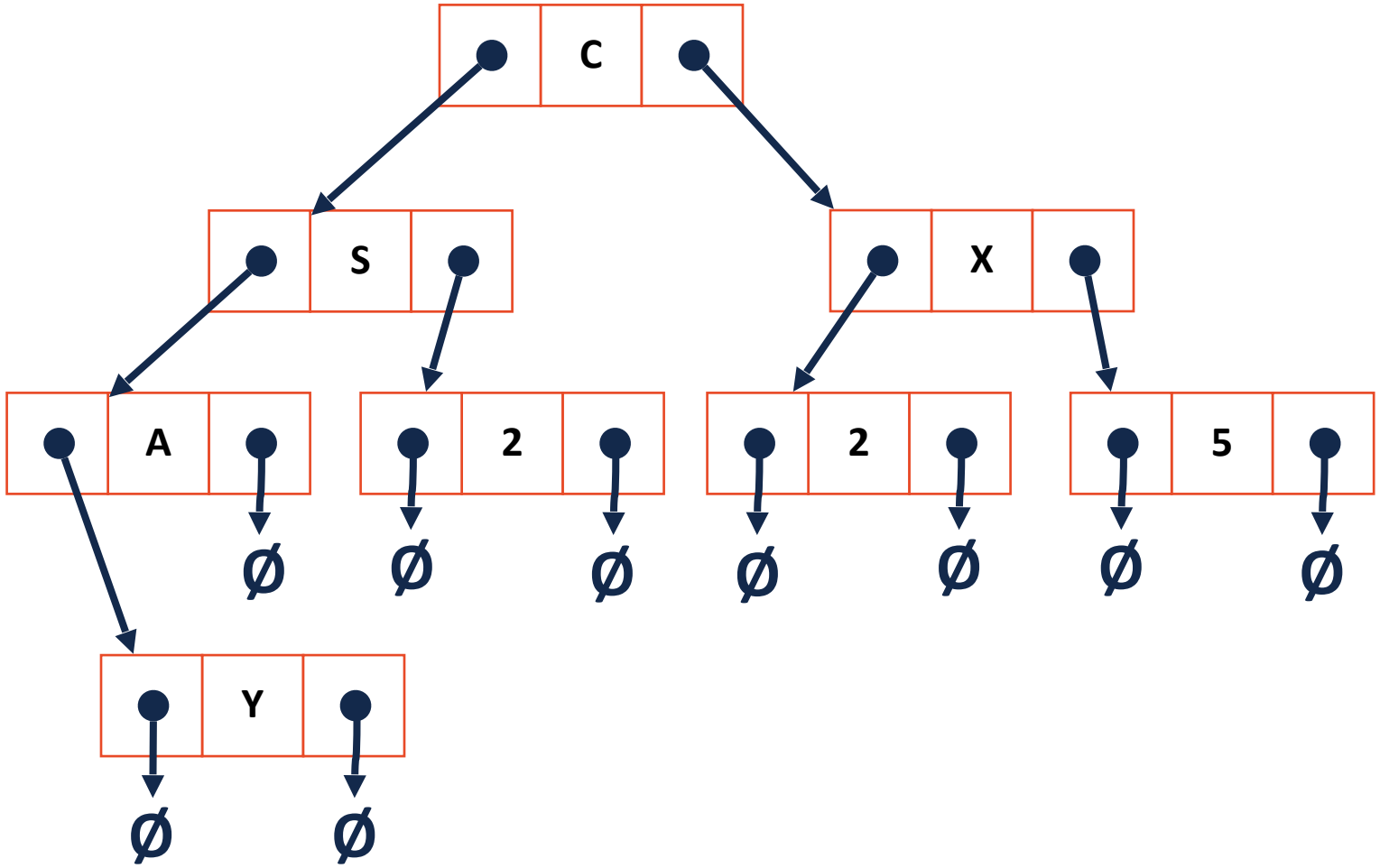
Visualizing trees





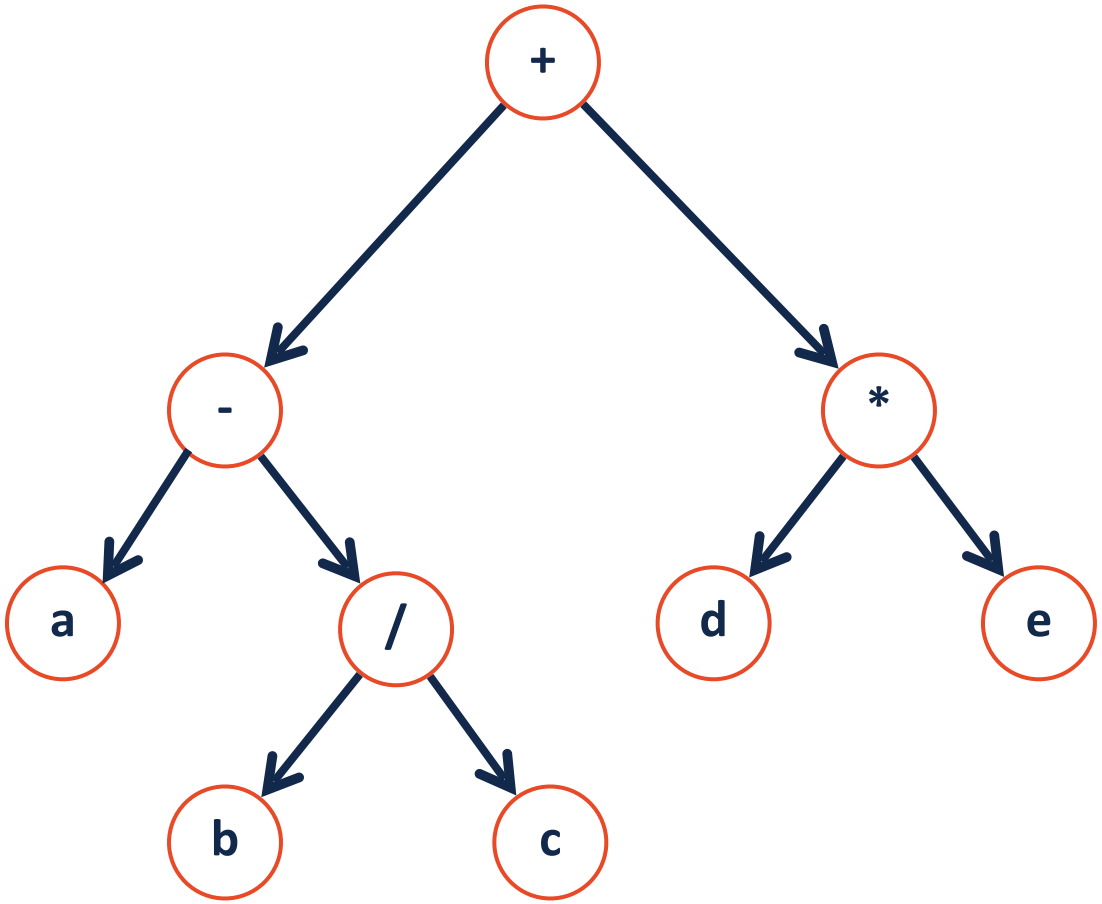
Tree Insert / Remove acts like Linked List



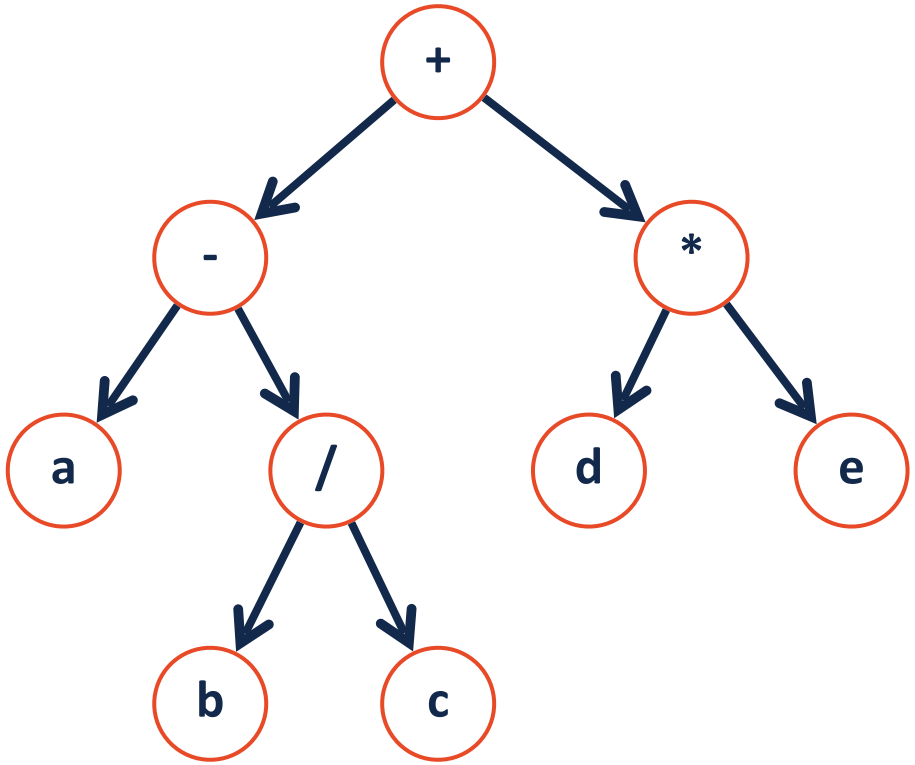


Tree Traversal

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

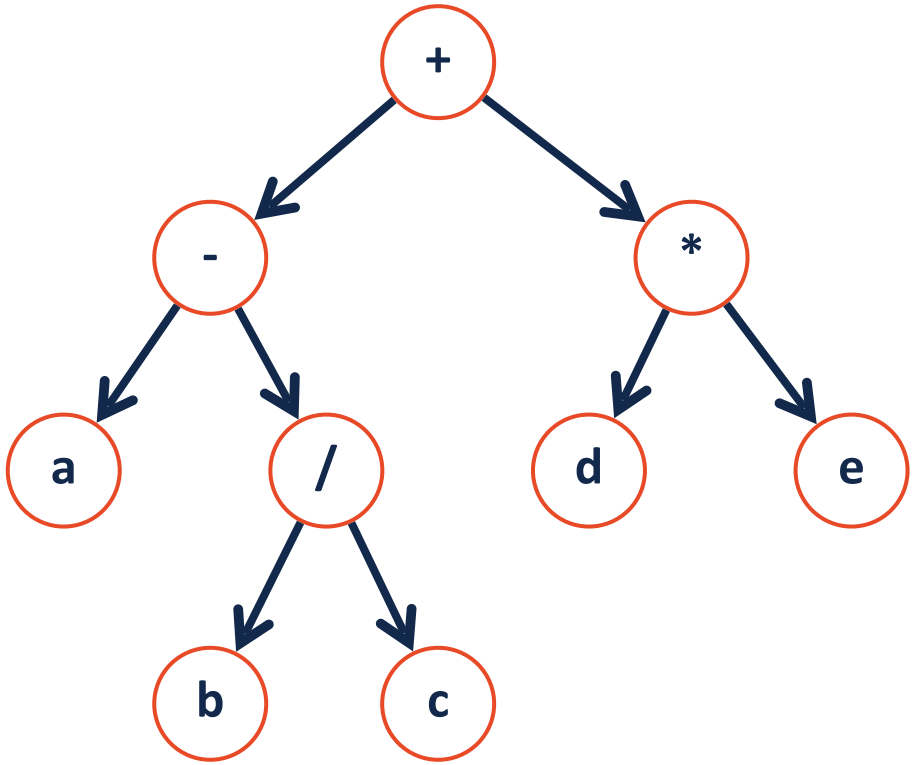


Traversals



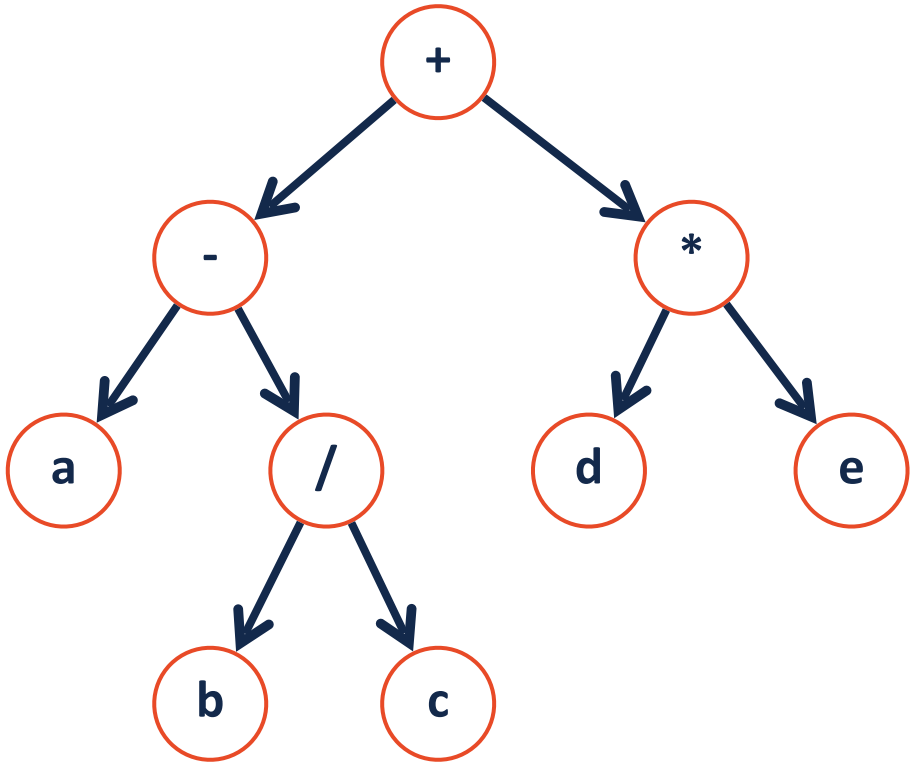

```
1  template<class T>
2  void BinaryTree<T>::_____Order(TreeNode * root)
3  {
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Traversals



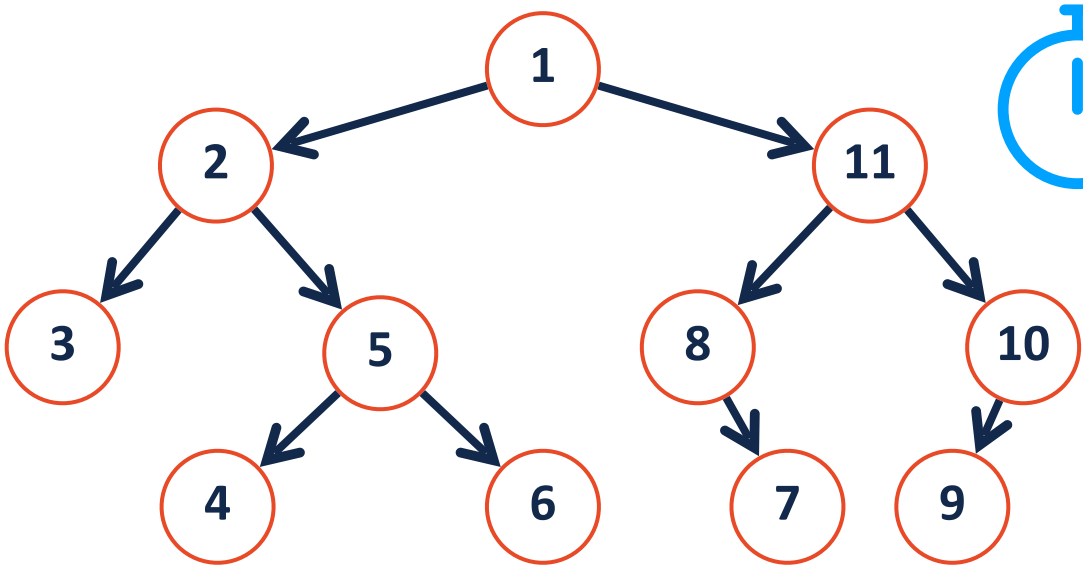

```
1  template<class T>
2  void BinaryTree<T>::_____Order(TreeNode * root)
3  {
4
5    if (root) {
6
7        ____________________________;
8
9        _____Order(root->left);
10
11       ____________________________;
12
13       _____Order(root->right);
14
15       ____________________________;
16
17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

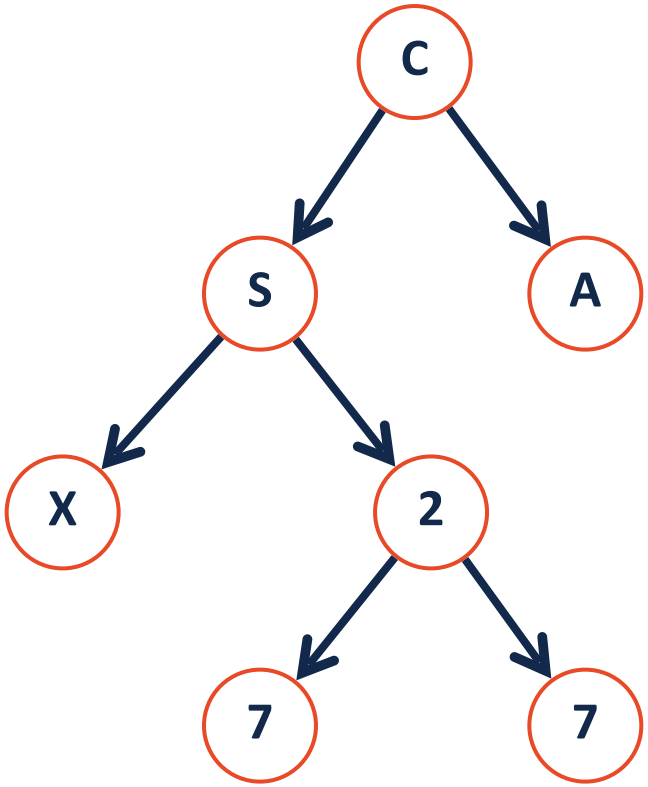
In-order:

Post-order:

Tree Traversals

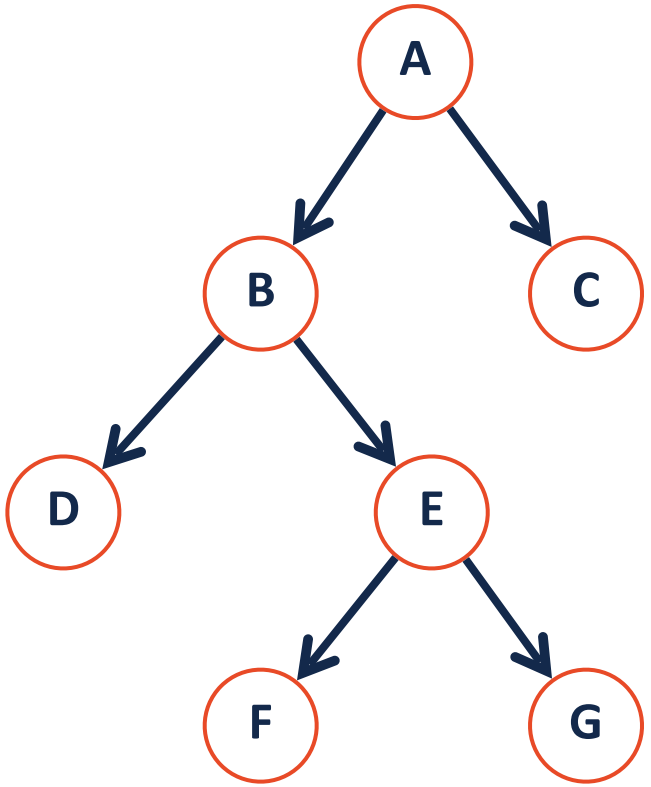
Pre-order: Ideal for copying trees

Post-order: Ideal for deleting trees



Traversal vs Search

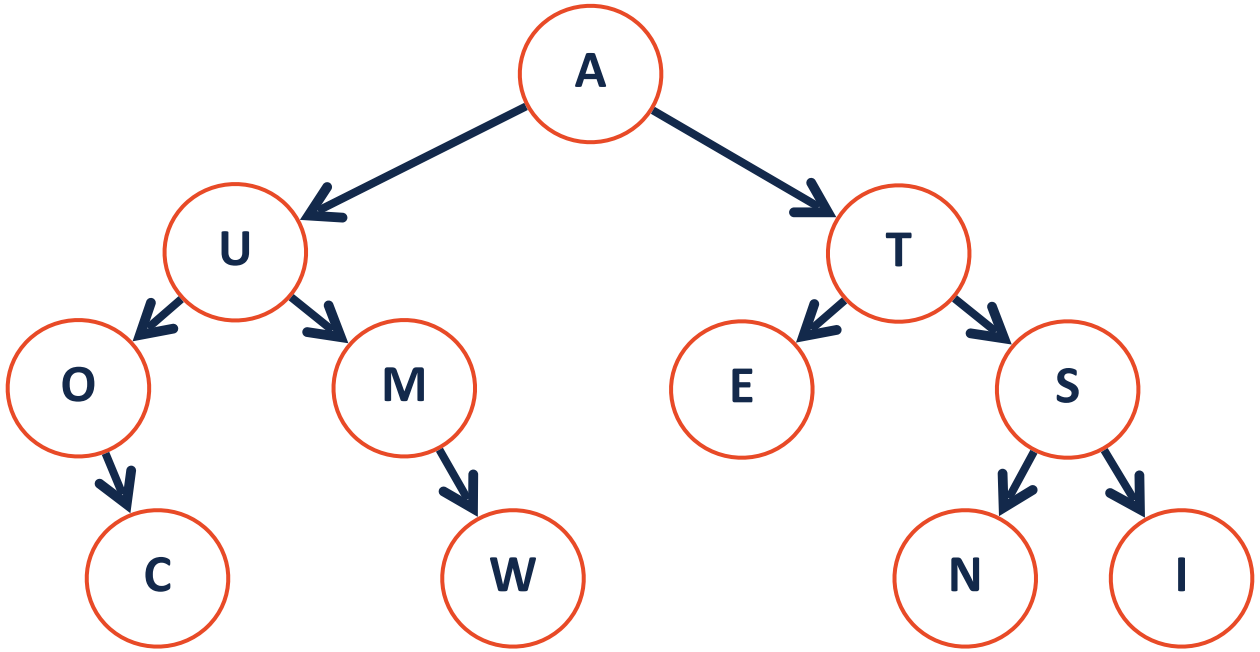
Traversal



Search

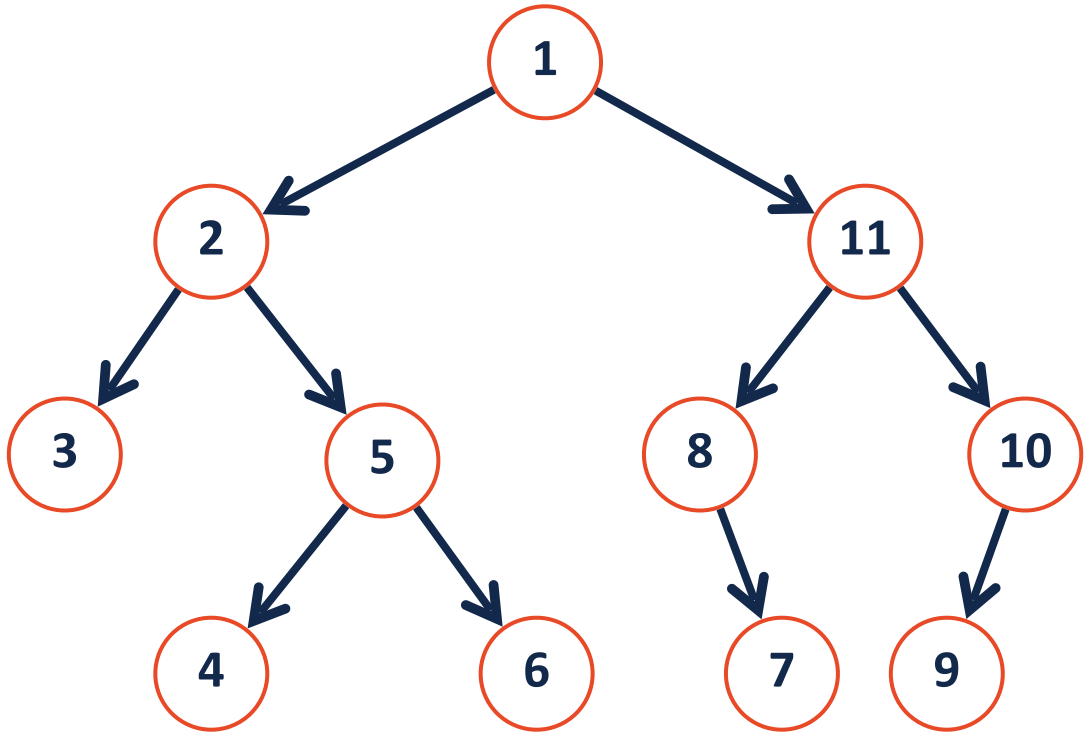
Tree Search

There are two main approaches to searching a binary tree:



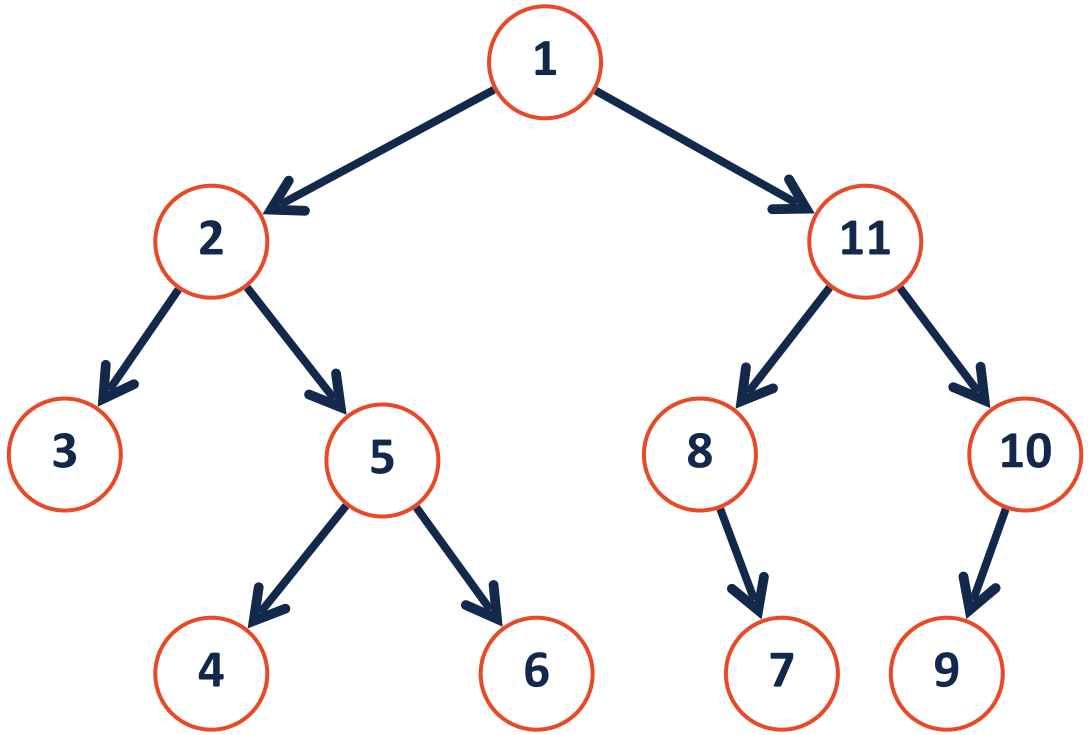
Depth First Search

Explore as far along one path as possible before backtracking

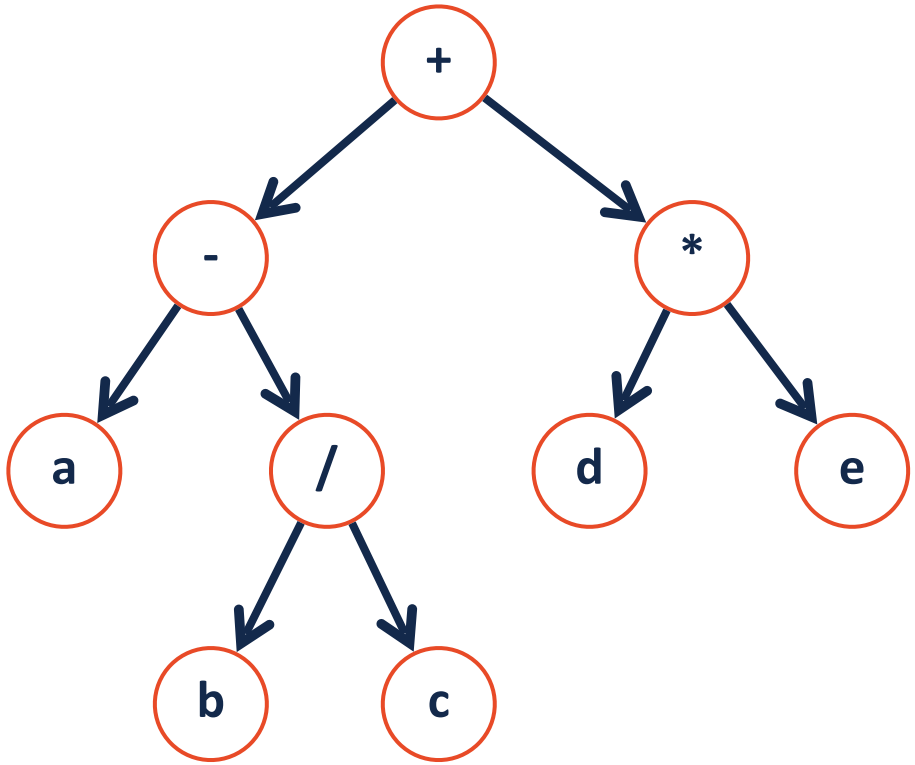


Breadth First Search

Fully explore depth i before exploring depth i+1



Level-Order Traversal




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

    Queue<TreeNode*> q;
    q.enqueue(root);

    while( q.empty() == False){

        TreeNode* temp = q.head();
        process(temp);

        q.dequeue();

        q.enqueue(temp->left);
        q.enqueue(temp->right);

    }
}
```

Tree Search

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

What do we trade in order to do so?