

# FINAL WRAP UP!

---

Problem Solving with Computers-II

C++

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook!";
    return 0;
}
```

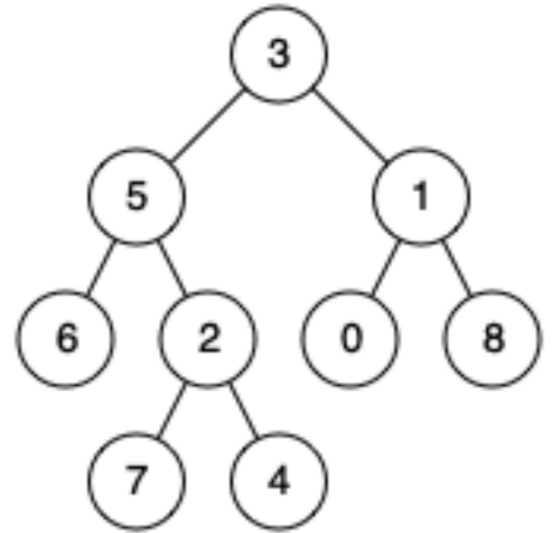
# Preparing for the final exam...

I can deal with pressure, and deadlines.



**Problem:**

**Find the lowest common ancestor of nodes  $(u, v)$  in a binary tree**



# Approach 1: Turn definitions into an algorithm

LCA( $r$ : root of tree,  $u$ ,  $v$ ):

Find the path from root( $r$ ) to  $u$

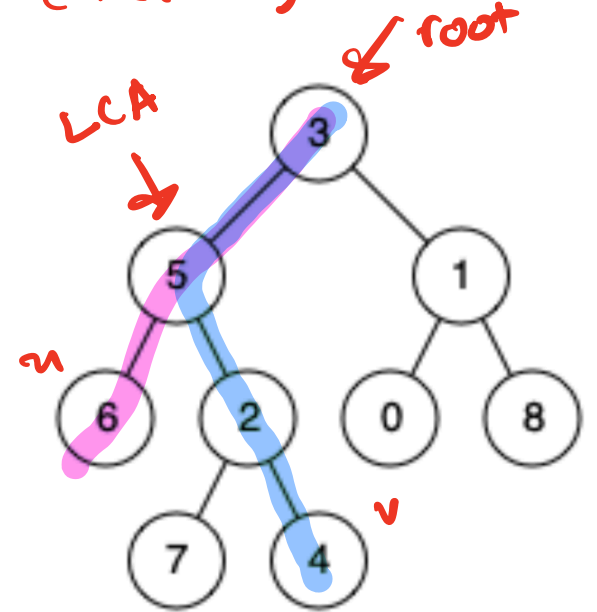
Find the path from root( $r$ ) to  $v$

Return common node on both paths

farthest from the root( $r$ )

Ancestor( $u$ ):

any node on a path ending in  $u$   
(including  $u$ )



path 1: (3), (5), 6  
path 2: (3), (5), 2, 4  
↑ 5 is the LCA

# Explore – Depth First on a Graph

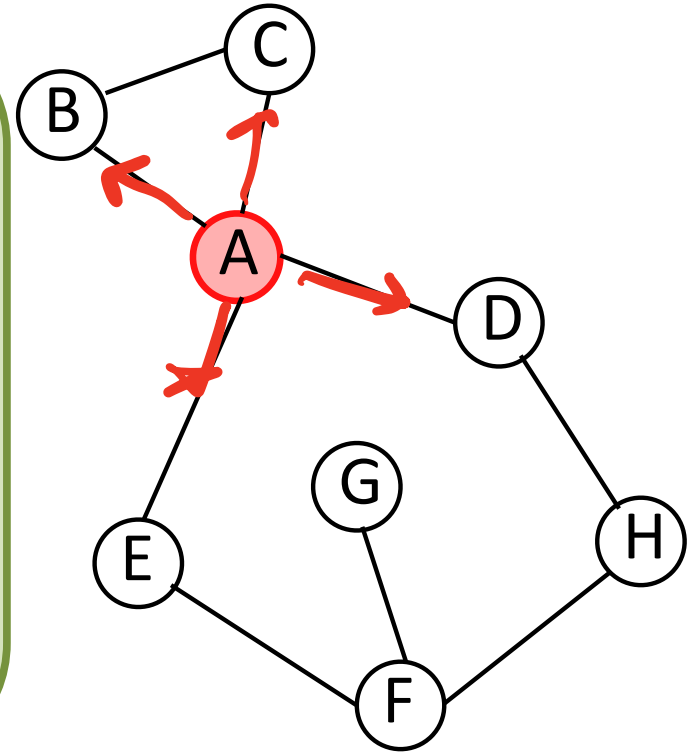
```
exploreDFS(v)
```

```
v.visited ← true
```

```
For each edge (v,w)
```

```
  If not w.visited
```

```
    exploreDFS(w)
```



# Modify exploreDFS to find paths

```
exploreDFS(v)
```

```
v.visited ← true
```

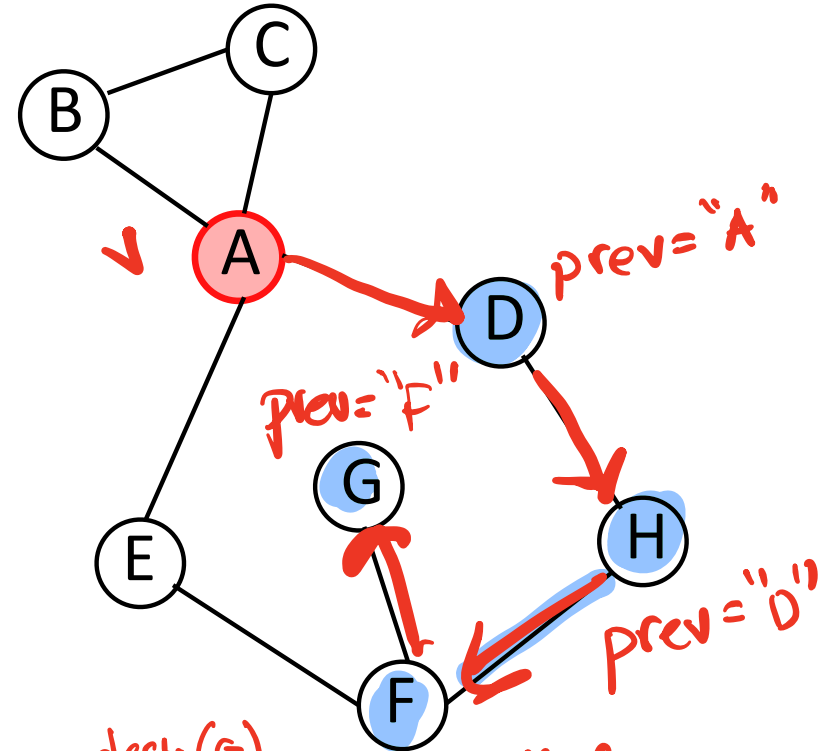
```
For each edge (v,w)
```

```
  If not w.visited
```

```
    w.prev ← v
```

```
    exploreDFS(w)
```

To construct the path, follow prev values starting with the dest. (G), present in reverse order  
Path from source = "A" to destination = "G"  
= A, D, H, F, G



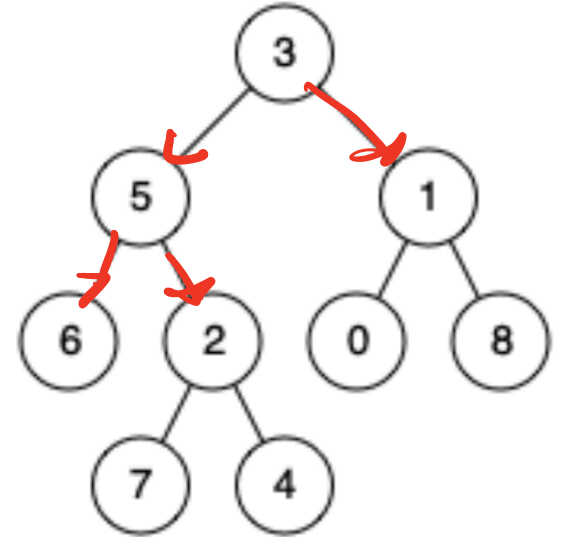
# Explore – Depth First on a Tree

```
exploreDFS(r: root node):
```

```
Print r.val
```

```
if(r.left)      exploreDFS(r.left)
```

```
if(r.right)     exploreDFS(r.right)
```



preorder traversal is an instance of DFS

# Modify DFS to find node u

*root* ↓  
*key to search for* ↓  
*bool flag* ←

```
DFS_FindNode(r, u, found):
```

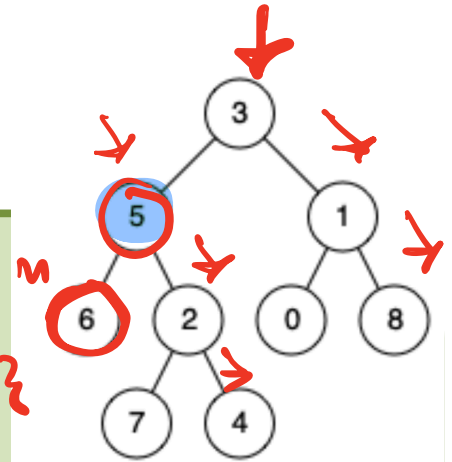
*if (!r) return*

*if (r.val == u.val) { found = true; return; }*

---

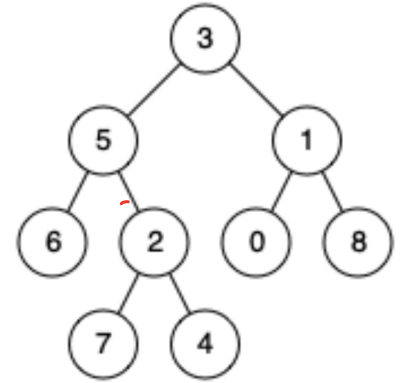
*if (r.left && !found) DFS\_FindNode(r.left, u, found)*

*if (r.right && !found) DFS\_FindNode(r.right, u, found)*





# Modify DFS to find the path to node u



```
DFS_FindPath(r, u, found, path):
```

```
if (!r) return  
path.push_back(r);
```

---

```
if (r.val == u.val) { found ← true; return; }
```

```
if (r.left && !found) DFS_FindPath(r.left, u, found, path)
```

```
if (r.right && !found) DFS_FindPath(r.right, u, found, path)
```

```
if (!found) path.pop()
```

---

path to 6: 3, 5, 6 ✓

path to 2: 3, 5, 6, 2 ✗

↑  
extra entry  
need to  
fix this

## Approach 2: Divide and Conquer

*High-level!*

```
LCA(r: root of tree, u, v):
```

```
  If(r.left && u,v exist in r.left)
```

```
    Return LCA(r.left, u, v)
```

```
  If(r.right && u,v exist in r.right)
```

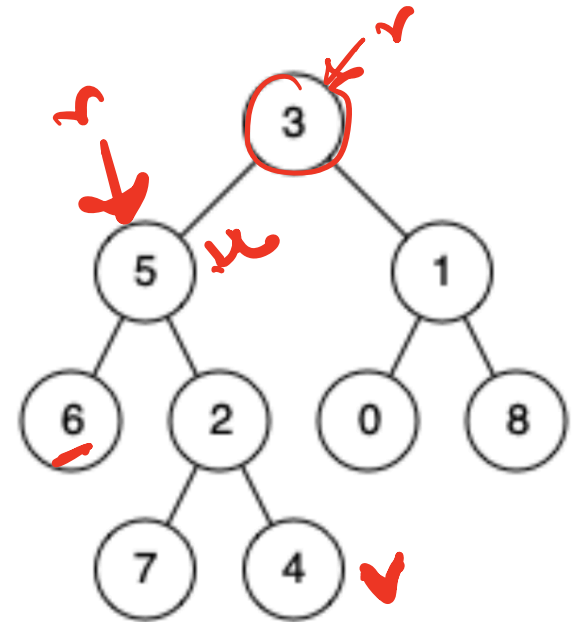
```
    Return LCA(r.right, u, v)
```

```
  If(u or v exists in r.left && u or v exists in r.right):
```

```
    Return root
```

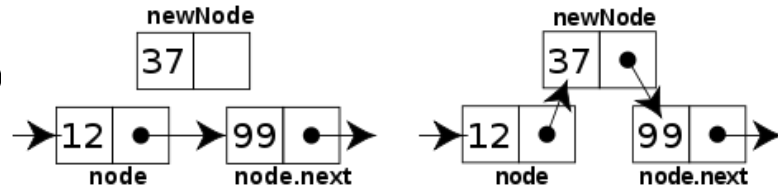
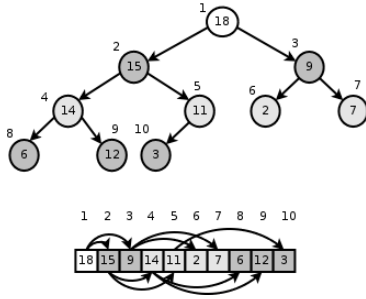
```
  If(u or v exists in r.left || u or v exists in r.right):
```

```
    If(r.val == u.val || r.val == v.val) Return root:
```



# Learning goals

- Design and implement **larger programs** that **run fast**
- Organize **data** in programs using **data structures**
- **Analyze** the **complexity** of your programs
- Understand what goes on **under the hood of programs**



INSERTION-SORT( $A$ )

```

1 for  $j = 2$  to  $A.length$ 
2    $key = A[j]$ 
3   // Insert  $A[j]$  into the sorted
   sequence  $A[1..j-1]$ .
4    $i = j - 1$ 
5   while  $i > 0$  and  $A[i] > key$ 
6      $A[i + 1] = A[i]$ 
7      $i = i - 1$ 
8    $A[i + 1] = key$ 

```

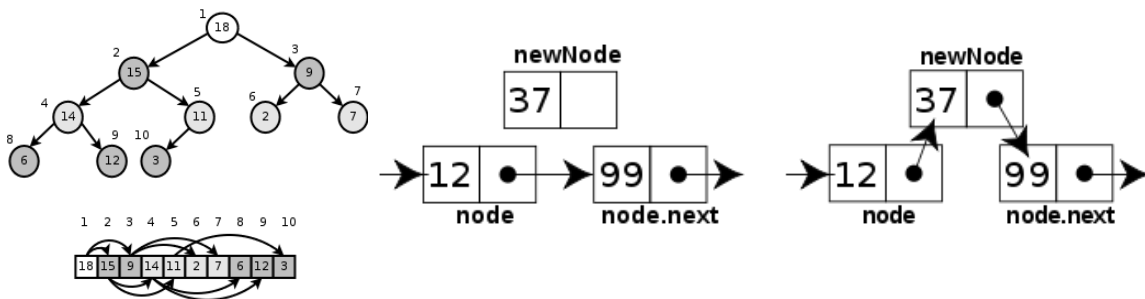
<i>cost</i>	<i>times</i>
$c_1$	$n$
$c_2$	$n - 1$
0	$n - 1$
$c_4$	$n - 1$
$c_5$	$\sum_{j=2}^n t_j$
$c_6$	$\sum_{j=2}^n (t_j - 1)$
$c_7$	$\sum_{j=2}^n (t_j - 1)$
$c_8$	$n - 1$

## Data Structures and C++

## Complexity Analysis

# Resources for the Final Exam

- Office hours will be offered until Wed of Finals Week
- Code from lectures: <https://github.com/ucsb-cs24-w24/cs24-w24-lectures>
- Practice Problems and Labs: <https://ucsb-cs24.github.io/w24/>
- Past Exams: Available on Canvas
- Tool to visualize data structures: <https://visualgo.net/>



	<i>cost</i>	<i>times</i>
INSERTION-SORT( <i>A</i> )		
1 for $j = 2$ to $A.length$	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5 while $i > 0$ and $A[i] > key$	$c_5$	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	$c_8$	$n - 1$

**Data Structures and C++**

**Complexity Analysis**

**Break: Please take a moment to fill the course evaluations!**



**PROBLEM SOLVING II**

Student-FO

<https://go.blueja.io/tJ9I2Cs-j068oUfOzQn2jA>



To access the evaluation, scan this QR code with your mobile phone.

---



**PROBLEM SOLVING II**

Student-FO

<https://go.blueja.io/tJ9I2Cs-j068oUfOzQn2jA>

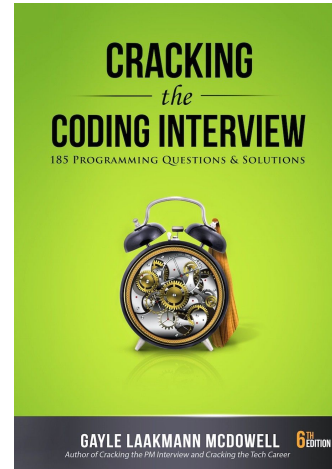


To access the evaluation, scan this QR code with your mobile phone.

---

# Tips for Technical Interviews

1. Listen carefully
2. Draw an example
3. State the brute force or a partially correct solution
  - then work to get at a better solution
4. Optimize:
  - Make time-space tradeoffs to optimize runtime
  - Precompute information: Reorganize the data e.g. by sorting
5. Solidify your understanding of your algo before diving into writing code.
6. Start coding!



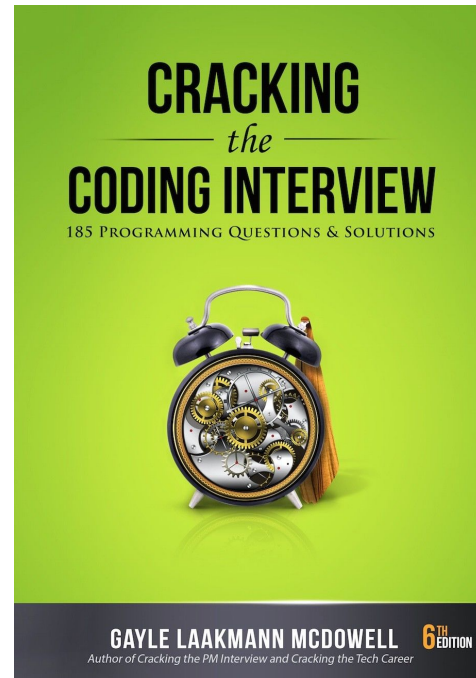
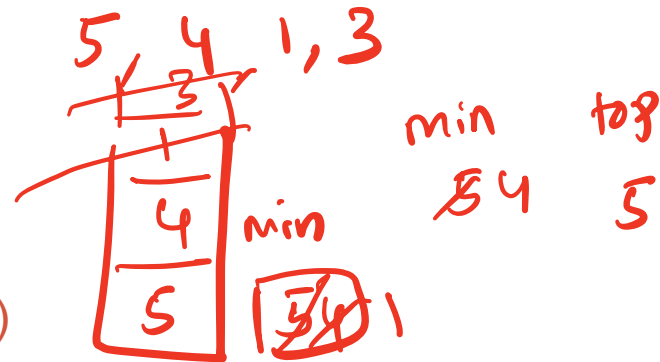
# Interview practice!

Write a ADT called minStack that provides the following methods

- push() // inserts an element to the “top” of the minStack
- pop() // removes the last element that was pushed on the stack
- top () // returns the last element that was pushed on the stack
- min() // returns the minimum value of the elements stored so far

## Practice the interview tips:

- Draw/solve a small example! (2 min)
  - Think of the most straightforward approach (1 min)
  - Evaluate its performance (1 min)
  - Think of another approach and evaluate it (5 min)
    - Can you trade off space/memory for better runtime?
- Pick the most promising approach and start coding! (10 min)



5

4

1

3

-1

<del>1, -1</del>
3, 1
1, 1
4, 4
5, 5

See lecture code on github



Thank you and all the best !

