Algorithms and Data Structures for Data Science
Review Day

CS 277
Brad Solomon

April 29, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Please fill out ICES evaluations

You can unofficially test a new system — please fill it out twice!

https://illinois.qualtrics.com/jfe/form/SV_6mOBFJa6ch4XKXc?rubric=cs&number=277&netid=bradsol

Review Topics

Recursion

BST and AVL Trees

Traversals

Hashing and Graphs

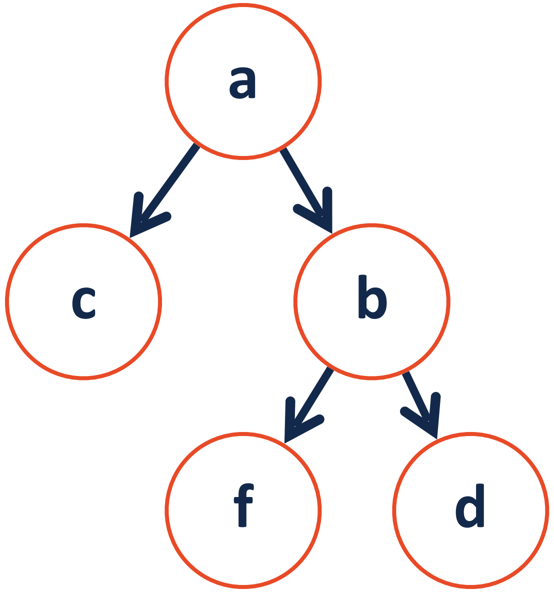
Coding Practice: Identity Matrix

An identity matrix is a 2D square matrix with 1s across the diagonal

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Recursion

Recursion is when a function calls itself directly or indirectly



Programming Toolbox: Recursion

When thinking recursively, break the problem into parts:

Base Case: What is the smallest sub-problem? What is the trivial solution?

Recursive Step: How can I reduce my problem to an easier one?

Combining: How can I build my solution from recursive pieces?

InsertionSort

4	3	6	7	1

1. Assume first value is ‘sorted’

2. Loop through remaining values:

3.Insert value into the ‘sorted’ array

Key trick: Insert by swapping!

Recursive insertionSort (Brainstorm + Code)

0	3	7	5	8	9	2	1	4	6

Base Case:

Recursive Step:

Combining:

Recursive List Partitioning

Using all elements in a list, can we make two lists which have equal sums?

6	5	4	2	7

1	1	1	1	1

2	3	3	3	1

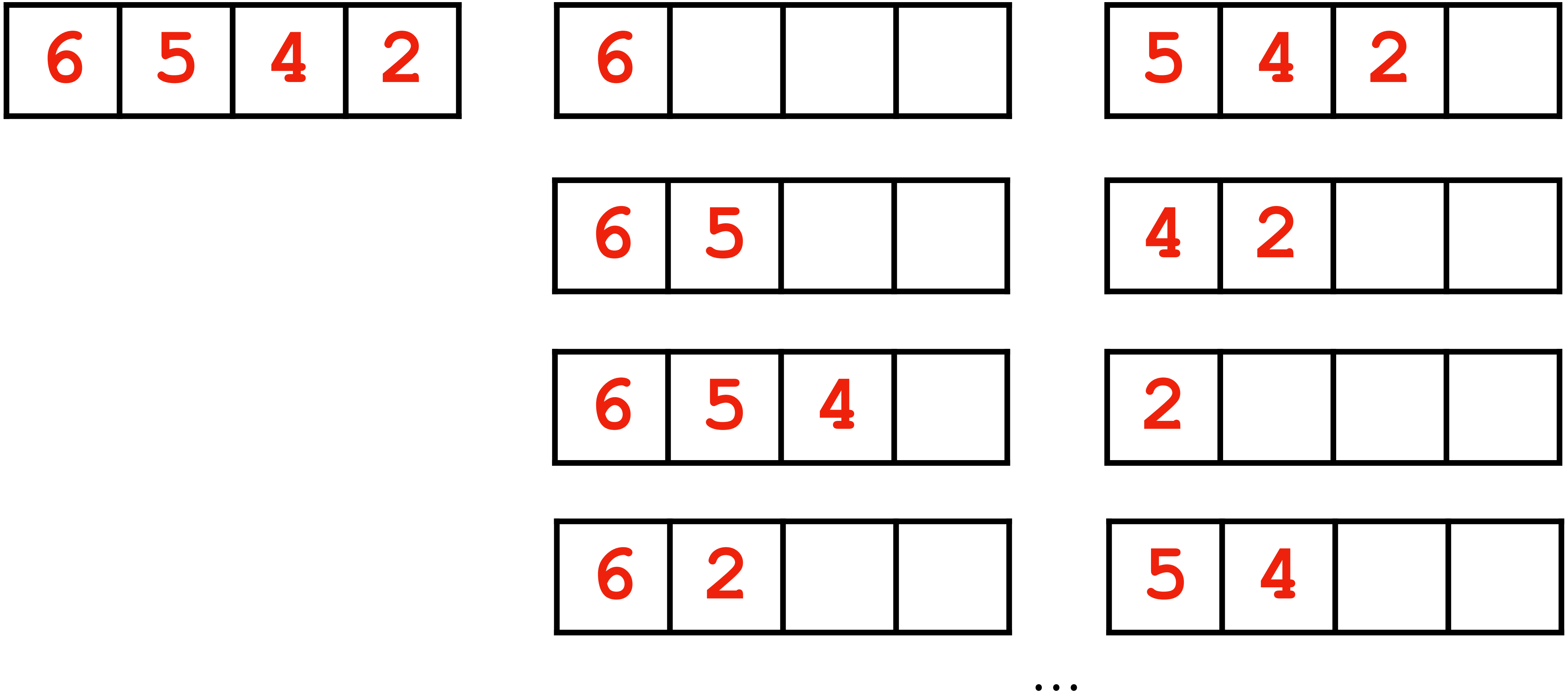
Recursive List Partitioning

How would a computer solve this problem?

6	5	4	2

Recursive List Partitioning

How would a computer solve this problem? **Compute every permutation!**



Recursive List Partitioning (Brainstorm)

2	3	7	4	8

Base Case:

Recursive Step:

Combining:

Recursive List Partitioning

Writing code to attempt every possible permutation is tricky with loops.

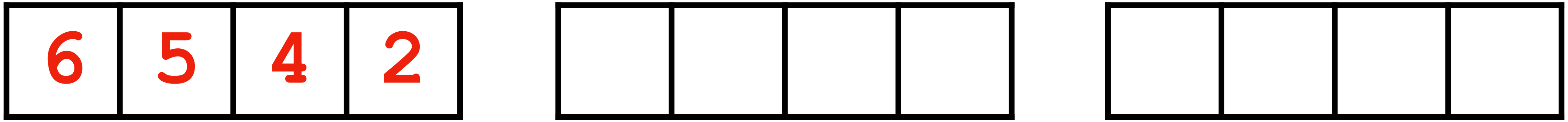
But its a great example of recursion in action!

Recursive Step: Given list L, pop() L[0] to left ***and*** right and recurse on both

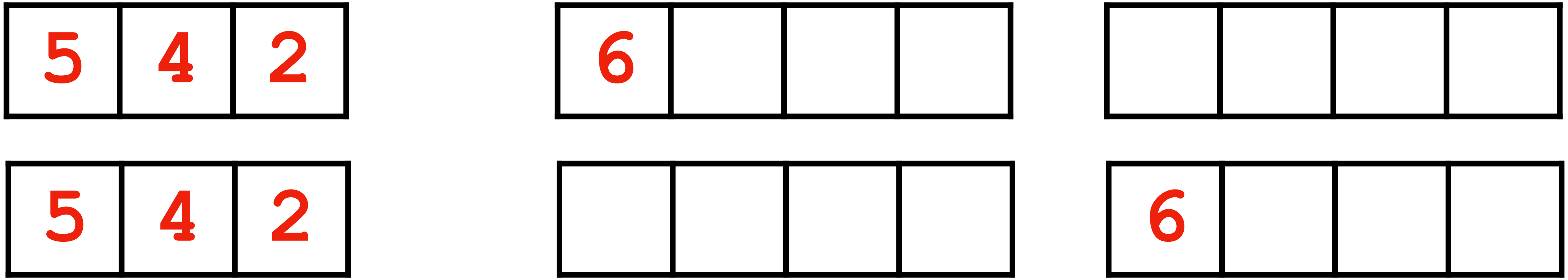
Recursive List Partitioning

Recursive Step: Given list L, pop() L[0] to left ***and*** right and recurse on both

Input: *Left* *Right*



Recursive Calls:



Recursive List Partitioning

Recursive Step: Given list L, pop() L[0] to left ***and*** right and recurse on both

Base Case:

Base Case: When my input list is empty, I have tried every permutation

Recursive Step: Given list L, pop() L[0] to left ***and*** right and recurse on both

[4, 3, 1] ([], [])

[3, 1] ([4], []) ([], [4])

[1] ([3, 4], []) ([4], [3]) ([3], [4]) ([], [3, 4])

[]

([1, 3, 4], []) ([1, 4], [3]) ([1, 3], [4]) ([1], [3, 4])

([3, 4], [1]) ([4], [1, 3]) ([3], [1, 4]) ([], [1, 3, 4])

Recursive List Partitioning (Brainstorm and code)

Base Case: When my input list is empty, I have tried every permutation

Recursive Step: Given list L, pop() L[0] to left ***and*** right and recurse on both

Combination Step:

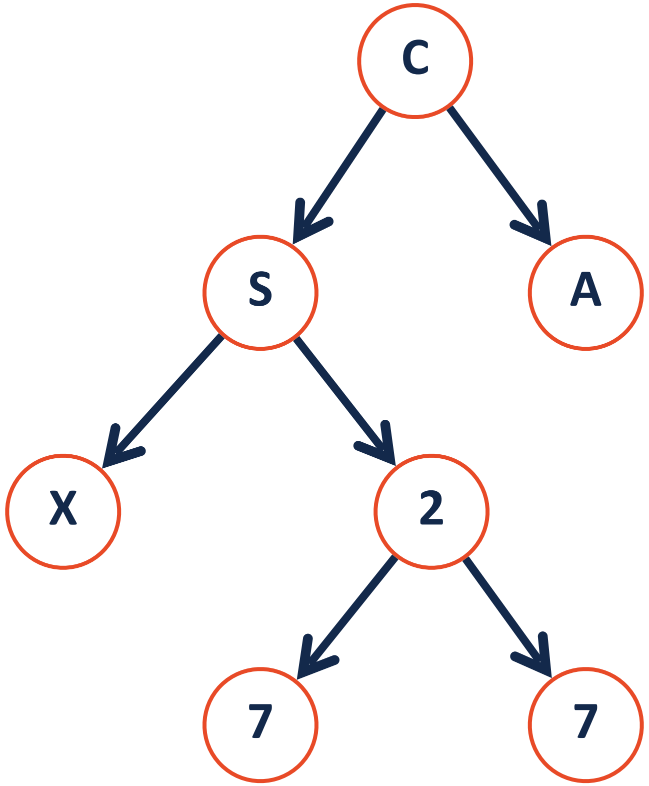
(Binary) Tree Recursion

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

$$T = None$$

or

$$T = treeNode(val, T_L, T_R)$$



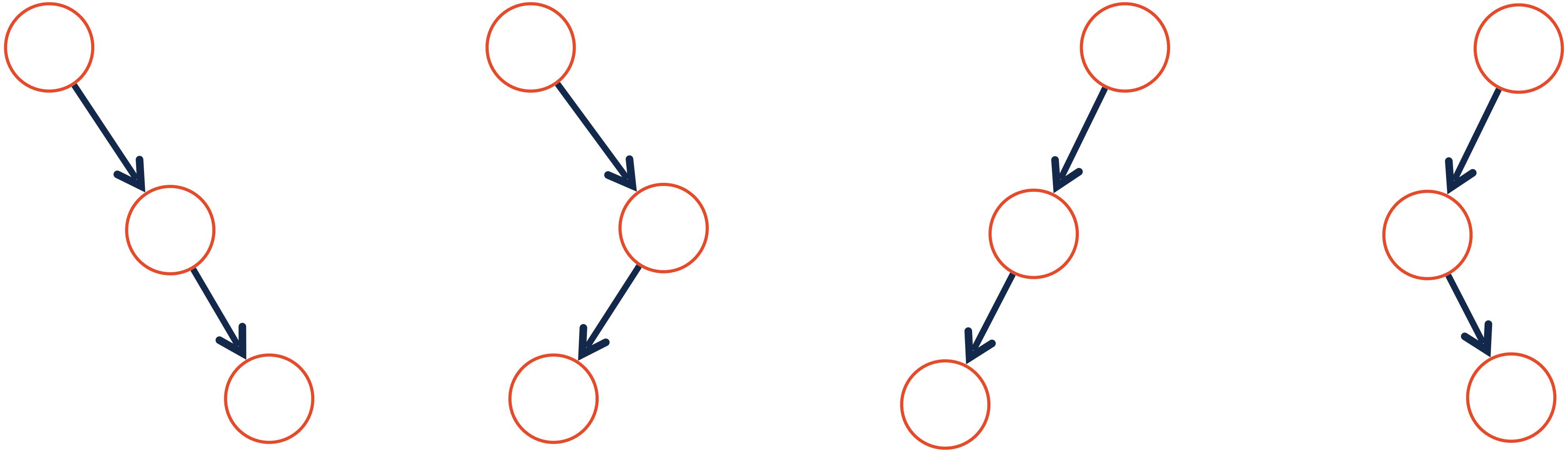

```
class treeNode:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

```
class binaryTree:
    def __init__(self):
        self.root = None
```

AVL Insertion

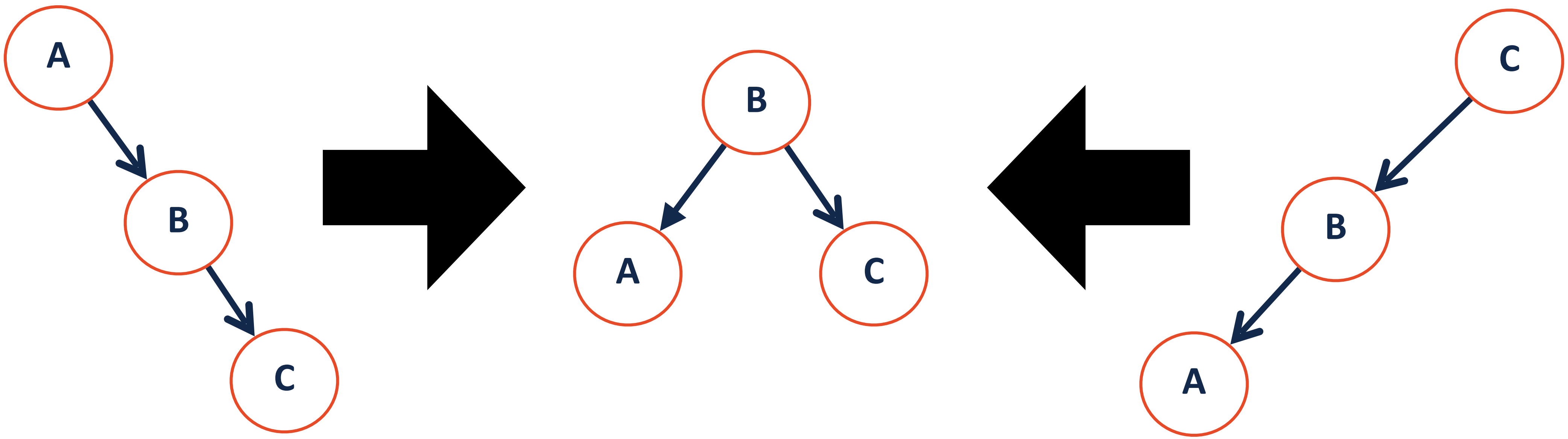
If we know our imbalance direction, we can call the correct rotation.

Left **RightLeft** **Right** **LeftRight**



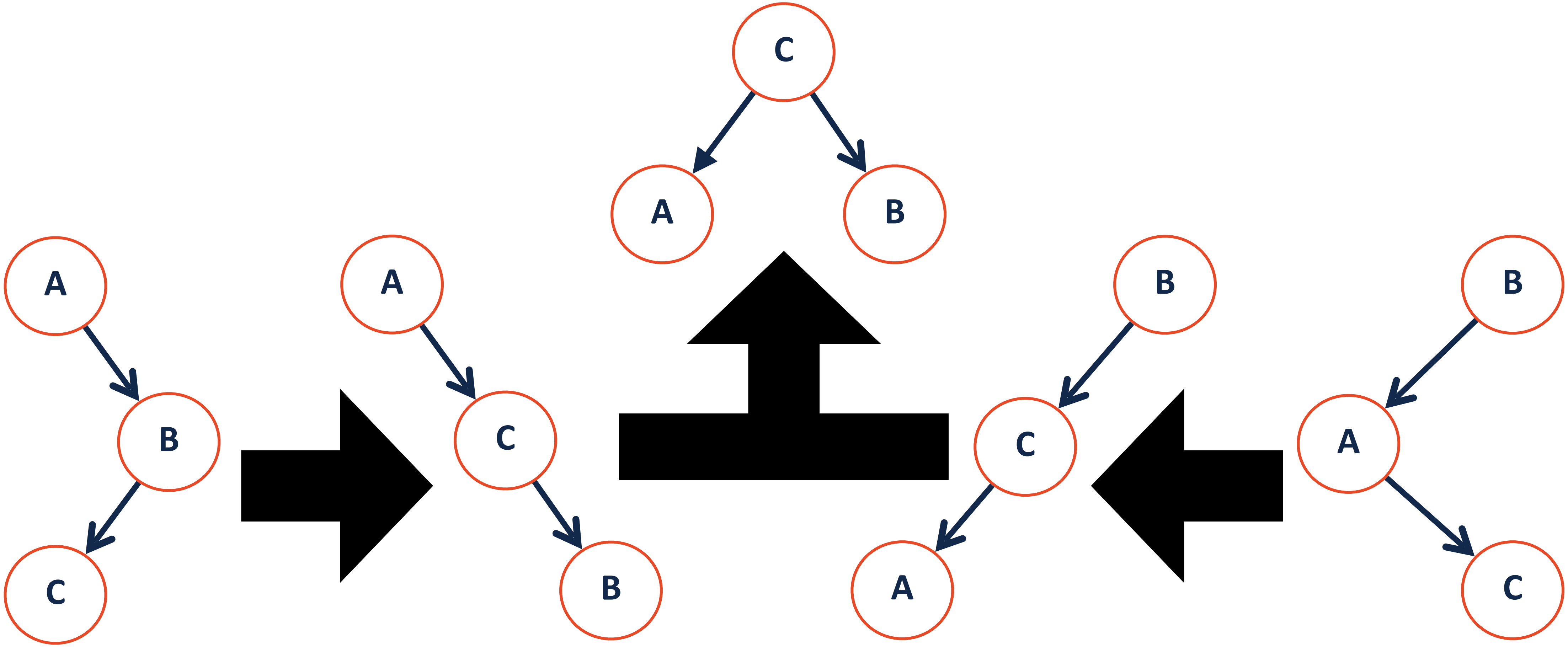
AVL Rotations

Left and right rotation convert **sticks** into **mountains**



AVL Rotations

LeftRight (RightLeft) convert **elbows** into **sticks** into **mountains**



Practice your trees!

Practice exams have randomly generated trees for:

Building a tree from scratch (or inserting one node)

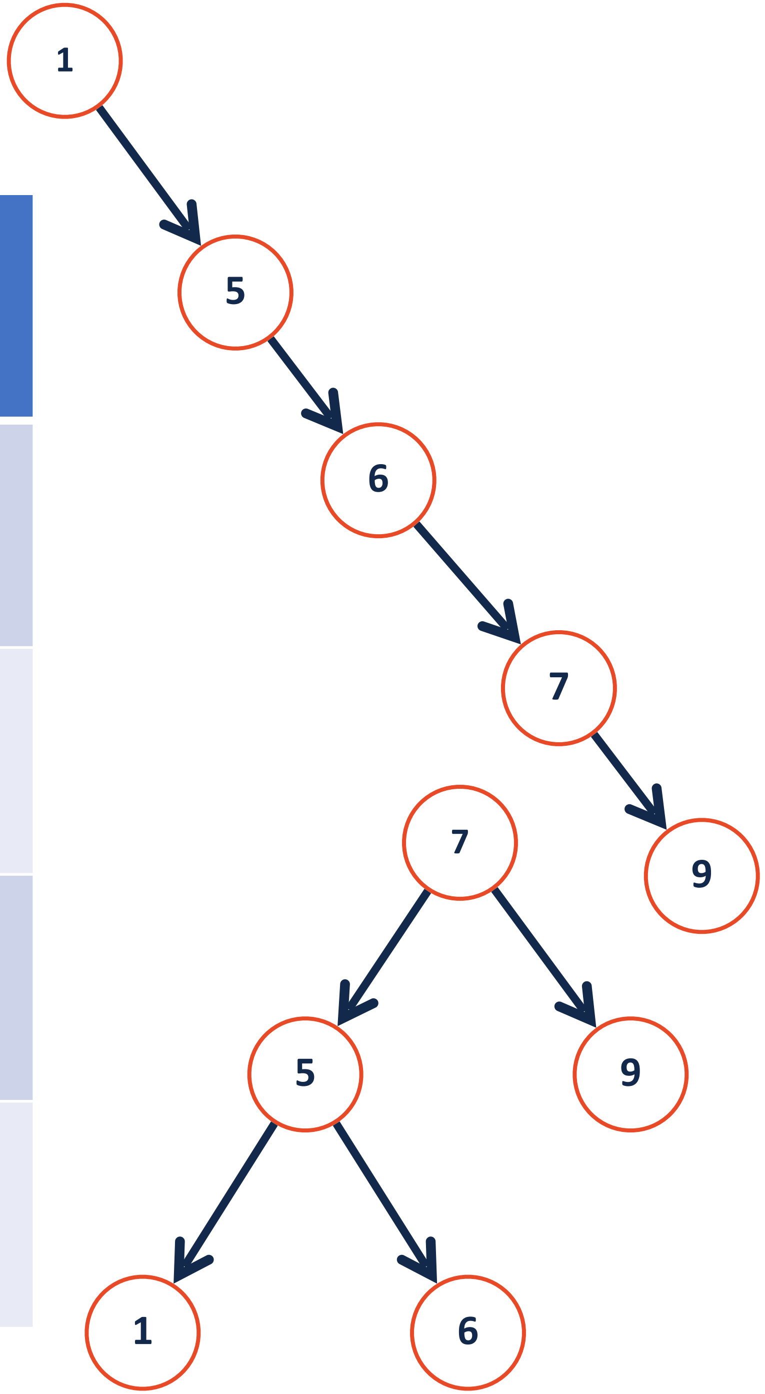
Calculating balance and height

Performing traversals

AVL Tree balance calculations

Tree Efficiency

	BST	AVL Tree
find		
insert		
delete		
traverse		



BST Coding Exercises

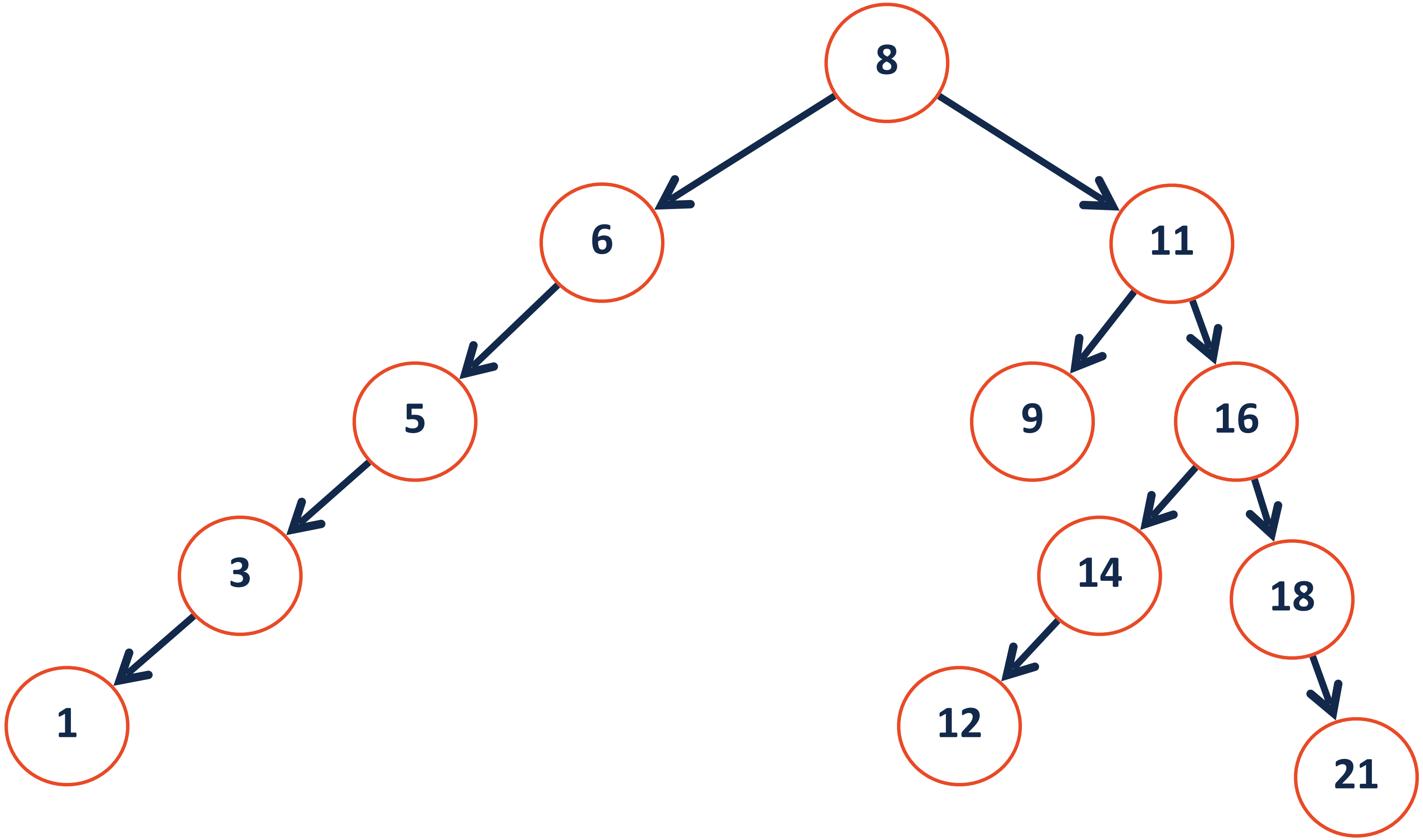
Can you write code to implement:

Find

Insert

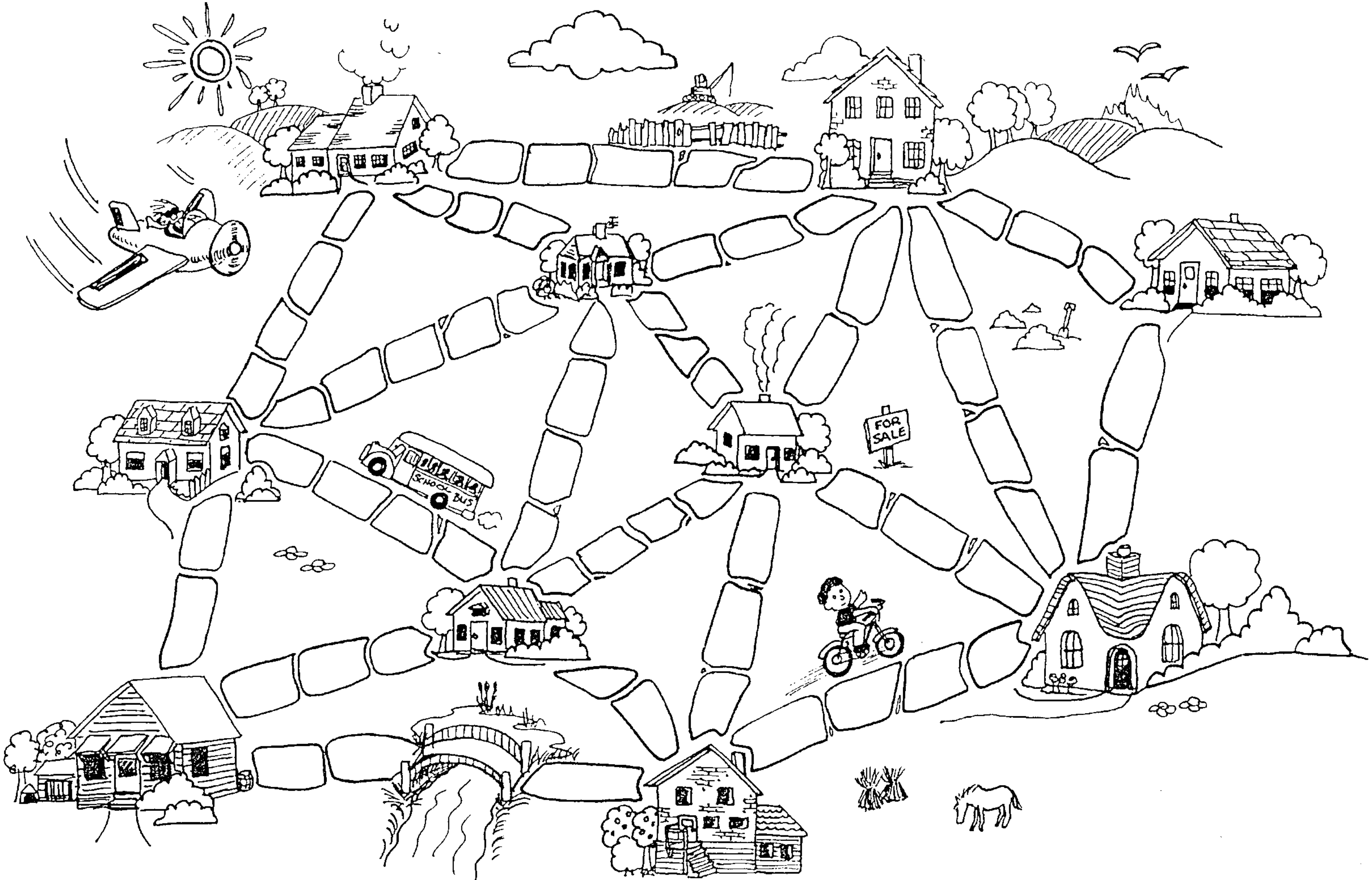
Remove

Traversals


8
6
11
5
9
16
3
14
18
1
12
21

$|V| = n, |E| = m$

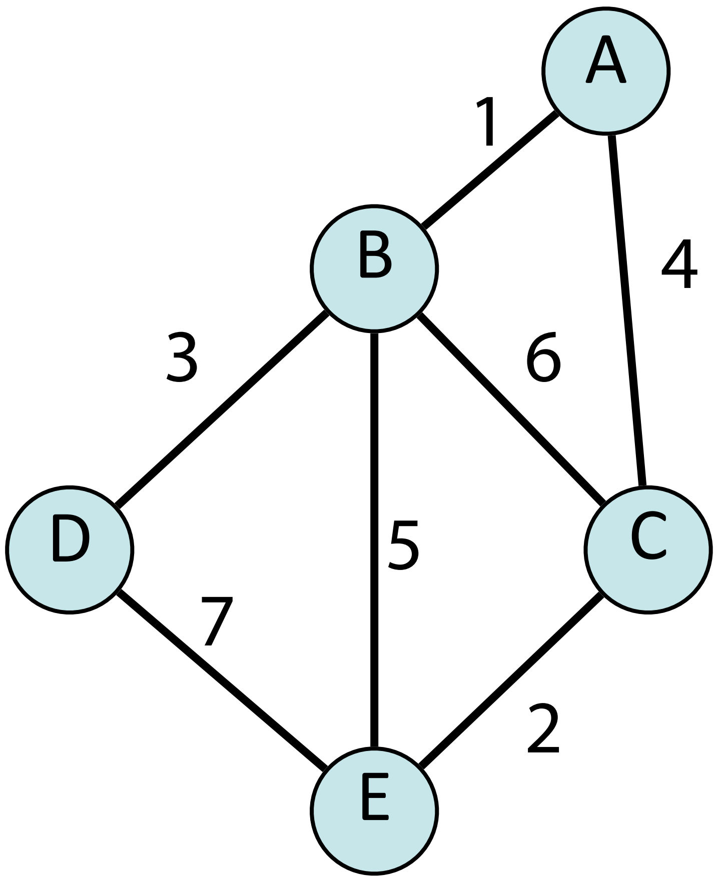
Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	n+m	n^2	n+m
insertVertex(v)	1*	n*	1*
removeVertex(v)	n+m	n*	deg(v)
insertEdge(u, v)	1	1	1*
removeEdge(u, v)	m	1	min(deg(u), deg(v))
getEdges(v)	m	n	deg(v)
areAdjacent(u, v)	m	1	min(deg(u), deg(v))


SCHOOL BUS
FOR SALE

Kruskal's Algorithm

Repeat until |V| - 1 edges found:

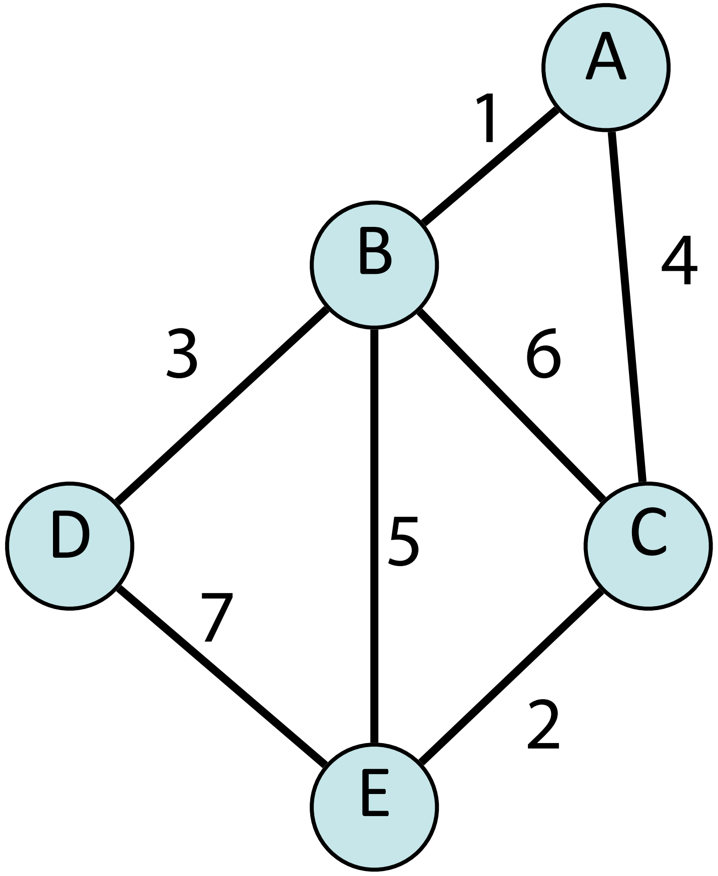
Find the minimum edge connecting two unconnected nodes



Prim's Algorithm

Repeat until |V| - 1 edges found:

Find the minimum edge connecting 'in' and 'out' group



Graph Coding Exercises

What did mp_algorithms ask you to do?

Read and parse input datasets (text / csv files)

Use NetworkX to build graphs with and without attributes

NetworkX Graph ADT Cheatsheet

Find

getVertices() —> list(G.nodes())

getEdges(v) —> G[v]

areAdjacent(u, v) —> G.has_edge(u, v)

Insert

insertVertex(v) —> G.add_node(v)

insertEdge(u, v) —> G.add_edge(u, v)

Remove

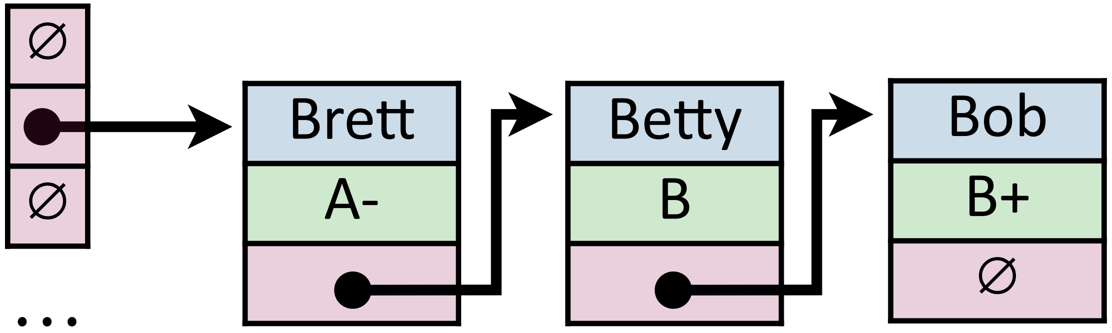
removeVertex(v) —> G.remove_node(v)

removeEdge(u, v) —> G.remove_edge(u, v)

Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

- **Open Hashing:** store *k,v* pairs externally



- **Closed Hashing:** store *k,v* pairs in the hash table

K, V
K, V
K, V

Hash Tables

- **Open Hashing:** store *k,v* pairs externally

  Load Factor ($\alpha = n/m$) can be infinite in size

- **Closed Hashing:** store *k,v* pairs in the hash table

  Load Factor ($\alpha = n/m$) must be between 0 and 1 (not including 1)