Welcome!

WWCode San Francisco - Backend Study Group

January 26, 2023

- We'll start in a moment :)
- We are **RECORDING** tonight's event
- We may plan to take screenshots for social media
- If you are comfortable, turn the video ON. If you want to be anonymous, then turn the video off
- We'll introduce the hosts & make some time for Q&A at the end of the presentation
- Feel free to take notes
- Online event best practices:
    - Don't multitask. Distractions reduce your ability to remember concepts
    - Mute yourself when you aren't talking
    - We want the session to be interactive
    - Use the 'Raise Hand' feature to ask questions
- **By attending our events, you agree to comply with our Code of Conduct**

WOMEN WHO CODE /san-francisco

Introduction & Agenda

- Welcome from WWCode!
- Our mission: Empower diverse women to excel in technology careers
- Our vision: A tech industry where diverse women and historically excluded people thrive at any level
- About Backend Study Group

Harini Rajendran
Presentor
Senior Software Engineer, Confluent
Lead, WWCode SF

Anjali Bajaj
Host
Lead, WWCode SF

- **Data Structures & Algorithms 101**
  - **Why Data Structures and Algorithms??**
  - **Common Types of Data Structures**
    - **Arrays**
    - **Linked Lists**
    - **Stacks**
    - **Queues**
    - **Trees**
    - **Graphs**
  - **DS and Algos Interview Tips**
  - **Q & A**

-Lot of content for 1 hour
-Mainly for beginners trying to enter tech

WOMEN WHO CODE /san-francisco

Why Data Structures & Algorithms??

What is a Data Structure?

- A way of organizing data in a computer
- Used to efficiently store, update and retrieve data as and when needed
- Classification: Linear and Non linear data structures

What is an algorithm?

- A set of well defined instructions to solve a particular problem
- Takes zero or more inputs and produces a deterministic output
- Terminates in a finite time
- Language independent
- There can be multiple ways to solve a problem
    - A problem -> Multiple algorithms (For example, searching and sorting algorithms)

Why are these tested in interviews?

- Almost all programs and software systems that are ever written use them
- Choice of the right data structure greatly determines the performance of the program
- Clearly demonstrates your problem solving skills
- Need not reinvent the wheel again and again

Common Data Structures

Linear Data Structures

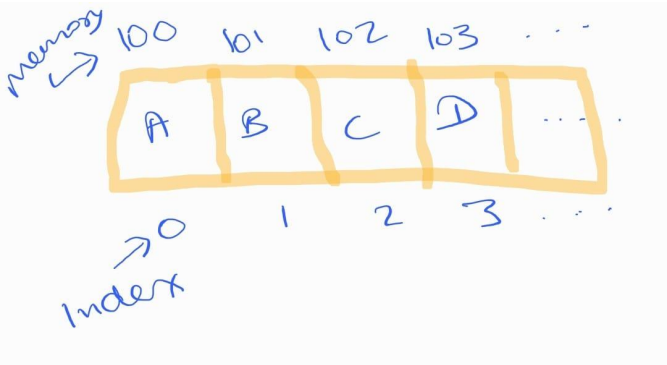
- Arrays
- Linked Lists
- Stacks
- Queues

Non Linear Data Structures

- Trees
- Graphs

Arrays

- Collection of similar data items stored in contiguous memory locations
- Static data structure [Fixed size]
- Basic Operations:
  - Search
  - Insert
  - Delete
- Time Complexity: O(1) [Insert/Delete based on Index]
- O(n): Search
- Auxiliary Space: O(1)




```
class ArrayExample {
    public static void main(String[] args)
    {
        // declares an Array of integers.
        int arr[];

        // allocating memory for 5 integers.
        arr = new int[5];

        // initialize the first elements of the array
        arr[0] = 10;

        // initialize the second elements of the array
        arr[1] = 20;

        // so on...
        arr[2] = 30;
        arr[3] = 40;
        arr[4] = 50;

        // accessing the elements
        for (int i = 0; i < arr.length; i++)
            System.out.println("Element at index " + i
                               + " : " + arr[i]);
    }
}
```

Arrays

Common interview problems

- Finding N or Nth largest element(s)
- Rotating elements in the array
- Sorting elements in the array
- Problem involving counting of various elements in the array
- Elements of sub-array with a given sum (2 sum, 3 sum, etc)
- Peak element in a sorted array
- Binary search in a sorted array
- Majority element in an array
- ....

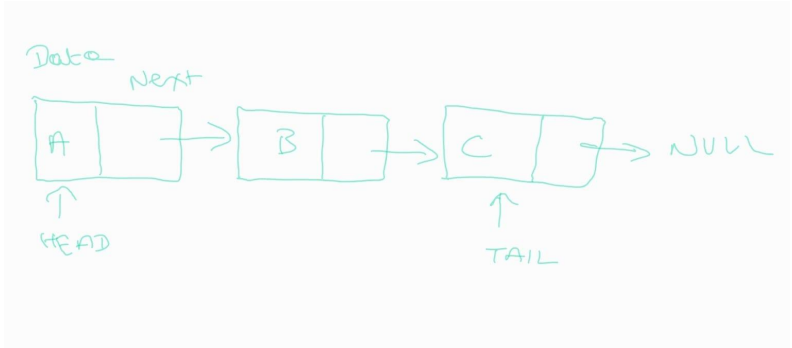
Great resource for Array related problems

https://www.geeksforgeeks.org/array-data-structure/?ref=lbp

Linked Lists

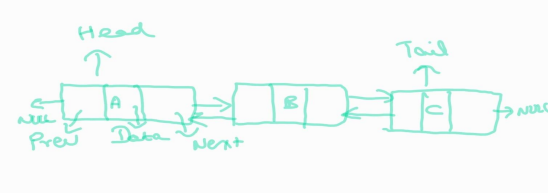
- Collection of similar data items stored in non-contiguous memory locations
- Elements are linked using pointes
- Dynamic data structure [Doesn't have a fixed size]
- Basic Operations:
    - Search
    - Insert
    - Delete
- Random access is not allowed
- One additional space is needed for each element to store the pointer
- Sorting is complicated with the pointers
- Searching is linear
- Time Complexity for Search/Insert/Delete: O(n)
- Auxiliary Space: O(n)

```
class SimpleLinkedList{
  Node head;
  Node tail;

  class Node {
    int data;
    Node next;

    Node(int d) {
      data = d;
      next = null;
    }
  }
}
```



Types of Linked Lists

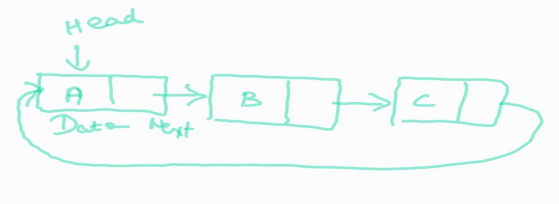
Doubly Linked List



Time Complextity
- Search: O(n)
- Insert/Delete: O(1)

```
class DoublyLinkedList{
  Node head;
  Node tail;

  class Node {
    int data;
    Node next;
    Node previous;

    Node(int d) {
      data = d;
      next = null;
      previous = null;
    }
  }
}
```

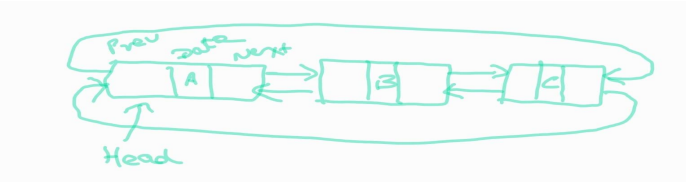
Circular Singly Linked List



Time Complextity
- Search/Insert/Delete: O(n)

Circular Doubly Linked List

Time Complextity
- Search: O(n)
- Insert/Delete: O(1)



Linked Lists

Common interview problems

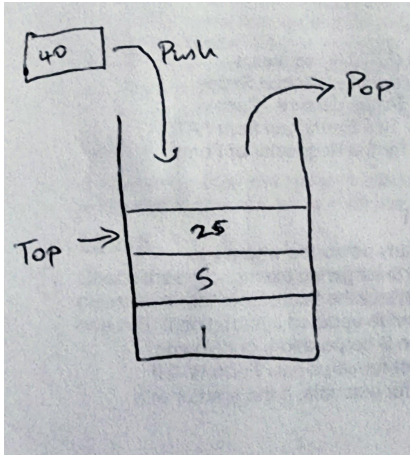
- Reverse a Linked List
- Find an element
- Sort a linked list
- Delete node
- Check if a linked list is circular
- Detect loop in a linked list
- 2 sum, 3 sum
- Peak element in a sorted linked list
- Splitting linked list
- ....

Great resource for Linked List related problems

https://www.geeksforgeeks.org/data-structures/linked-list/?ref=lbp

Stacks

- Linear data structure
- Operations are performed in a fixed order [Last in First Out]
- Stack can be implemented using
    - Arrays [Static stack]
    - Linked List [Dynamic stack]
- Basic Operations:
    - Push
    - Pop
    - Top
    - IsEmpty
- Time Complexity: O(1)
- Auxiliary Space: O(1)




```
class Stack {
    int top;
    int a[];
    int capacity;

    Stack(int cap) {
      top = -1;
      a = new int[cap];
      capacity = cap;
    }

    void push(int num) {
      if (top == capacity -1) {
        throw new Exception("Stack
Overflow");
      } else {
        a[++top] = num;
      }
    }

    int pop() {
      if (top == -1) {
        throw new Exception("Stack
Underflow");
      } else {
        int num = a[top];
        A[top—] = 0;
        return num;
      }
```

```
void top() {
  if (top == -1) {
    throw new Exception("Stack
Underflow");
  } else {
    return a[top];
  }
}

boolean isEmpty() {
  return top == -1;
}
}
```

Stacks

Common interview problems

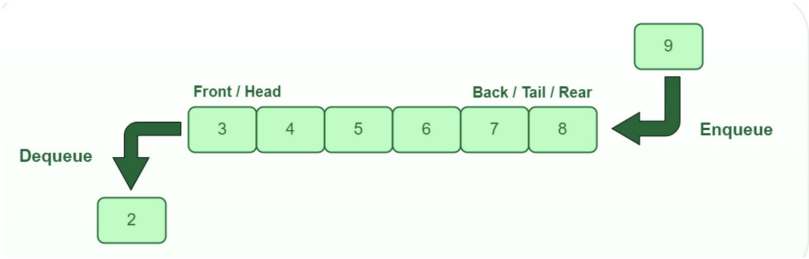
- Implement Queue using Stacks
- MinStack problem
- Many problems related to prefix, infix and postfix conversions
- Reversing a string, reversing words in string, etc
- Next greater element in an array
- Checking mirror in n-ary tree
- Printing ancestors of 2 nodes in a tree
- ....

Great resource for Stack related problems

https://www.geeksforgeeks.org/stack-data-structure/?ref=shm

Queues

- Linear data structure
- Operations are performed in a fixed order [First in First Out]
- Queue can be implemented using
  - Arrays
  - Linked List
- Basic Operations:
  - Enqueue
  - Dequeue
  - Front
  - Rear
  - IsEmpty
- Time Complexity: O(1)
- Auxiliary Space: O(1)



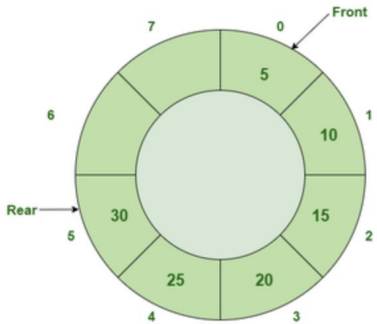

```
class Queue {
 class Node {
  int key;
  Node next;

  Node(int k) {
   key = k;
   next = null;
  }
 }

 Node front, rear;
 Queue() {
  front = null;
  rear = null;
 }

 void enqueue(int n) {
  Node temp = new Node(n);
  if (front == null) {
   front = rear = temp;
  } else {
   rear.next = temp;
   rear = rear.next;
  }
 }

}
```

```
int dequeue() {
 if (front == null) {
  throw new Exception("Can't dequeue
from empty queue");
 }

 int n = front.key;
 front = front.next;
 if (front == null) {
  rear = null;
 }
 return n;
}

int front() {
 if (front == null) {
  throw new Exception("Queue is
empty");
 }
 return front.key;
}

boolean isEmpty() {
  return front == null;
}
}
```

Types of Queues

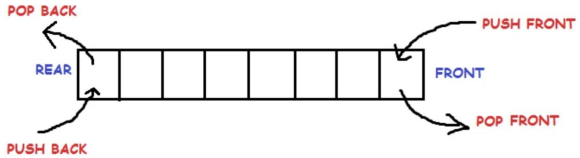
Circular Queue

- Also called ring buffer
- Rear element of the queue is connected to the front
- Basic Operations:
  - Enqueue
  - Dequeue
  - Front
  - Rear
  - IsEmpty
- Time Complexity: O(1)
- Auxiliary Space: O(1)
- Array implementation is always enough.



DeQueue`[Double Ended Queue]`

- Inserts and removals allowed from both the ends
- DeQueue can be implemented using
  - Arrays
  - Linked List
- Basic Operations:
  - PushFront
  - PopFront
  - PushBack
  - PopBack
  - Front
  - Rear
  - IsEmpty
- Time Complexity: O(1)
- Auxiliary Space: O(1)



Queues

Common interview problems

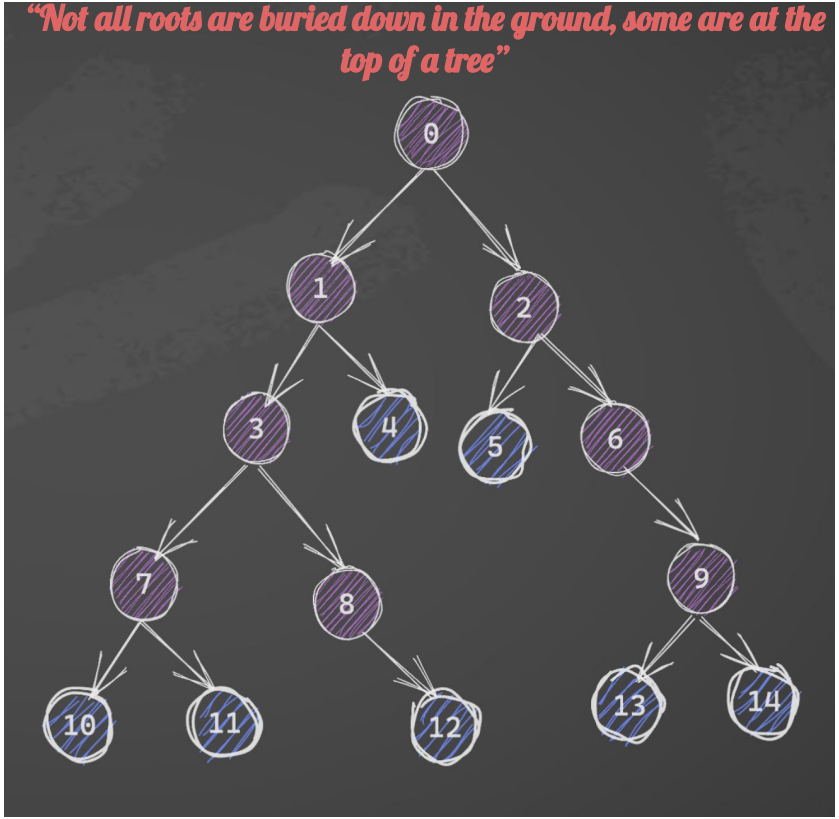
- Breadth First Search of a tree/graph
- Level Order Traversal of a tree
- Detect cycle in a graph
- LRU cache implementation
- Number of islands in a graph, etc
- ....

Great resource for Queues related problems

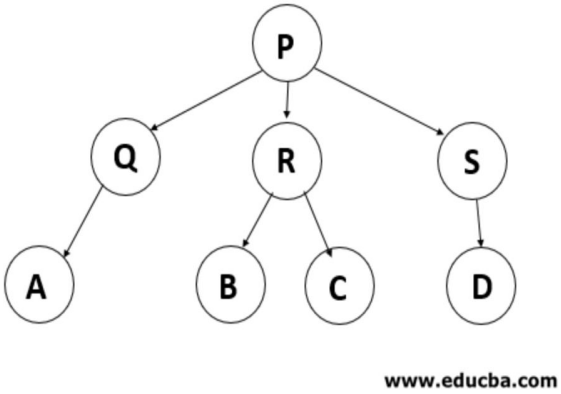
https://www.geeksforgeeks.org/queue-data-structure/?ref=shm

Trees

- Dynamic, hierarchical, non-linear and recursive data structure
- Each node can have any number of children
- Basic Operations:
  - Insert
  - Remove
  - Search
  - Traverse
    - Inorder
    - Preorder
    - Postorder
    - Level Order
- Terminologies
  - Node - Fundamental unit of a tree
  - Edge - Links connecting nodes
  - Root - First node
    Leaf - Node with NO children
  - Internal nodes - All non-leaf nodes
  - Parent - Predecessor of a node
  - Children - Immediate descendent of a node
  - Siblings - Nodes with the same parent

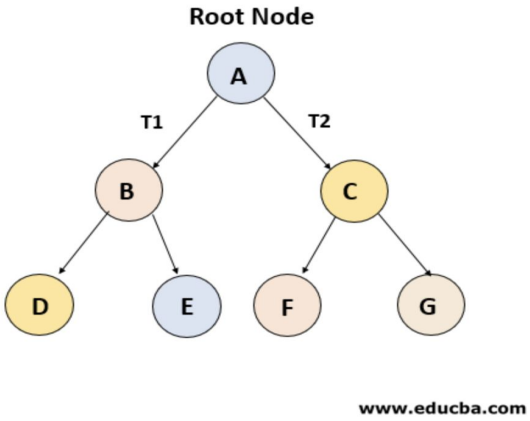


Types of Trees



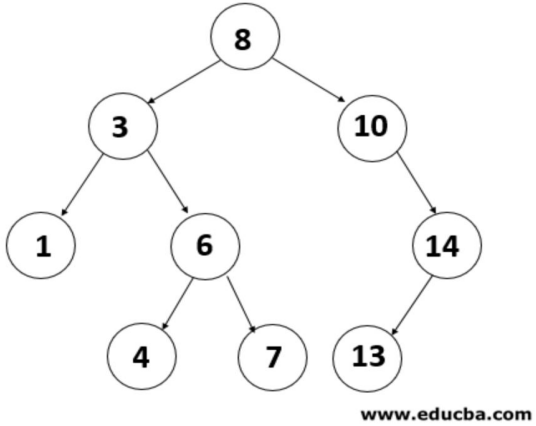
General Trees

- No constraints
- A node can have any # of children



Binary Trees

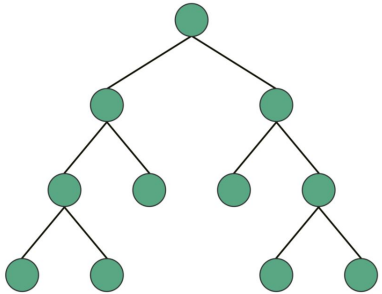
- A node can have utmost 2 children
- Left and right child



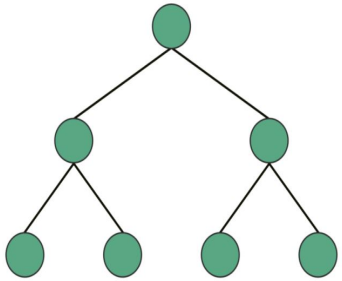
Binary Search Trees

- A node can have utmost 2 children
- Value of left child <= Value of parent
- Value of right child >= Value of parent

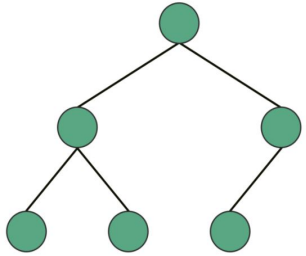
Types of Binary Trees



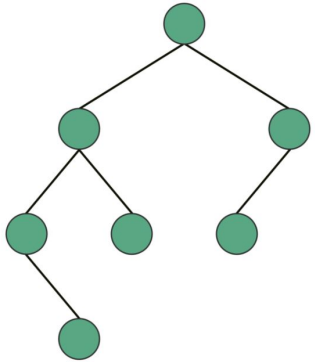
Full BT: Every node has exactly 0 or 2 children



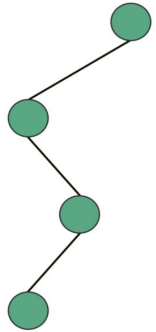
Perfect BT: All internal nodes have 2 children and all leaves are at same level



Complete BT: All levels are completely filled except last level. All nodes in last level are as left as possible



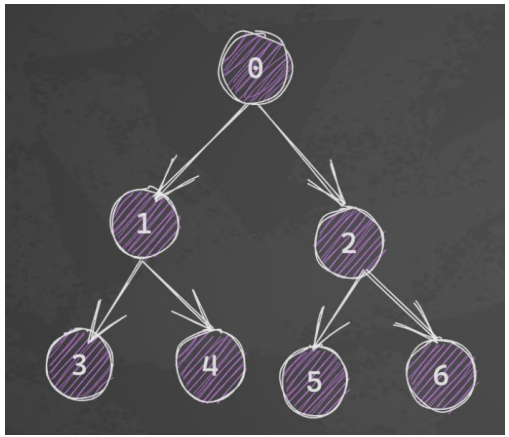
Balanced BT: Height of left and right subtree at every node may differ by at most 1



Degenrate BT: Every parent has only one child

Tree Traversals

```
class BinaryTree {
 class TNode {
  int value;
  TNode left;
  TNode right;

  TNode(int v) {
   value = v;
   left = null;
   Right = null;
  }
 }

 TNode root;
 ….
}
```



Pre-order (Root->Left->Right) : 0 1 3 4 2 5 6
In-order (Left->Root->Right) : 3 1 4 0 5 2 6
Post-order (Left->Right->Root): 3 4 1 5 6 2 0

Breadth First Search (Level Order, One level at a time): 0 1 2 3 4 5 6

Depth First Search (Same as pre-order, go till the end of the path and then backtrack):
0 1 3 4 2 5 6

Left -> Root -> Right (0 - 1 - 3 - 4 - 2 - 5 - 6)

```
public void inOrder(TNode node) {
    if (node != null) {
        inOrder(node.left);
        System.out.print(" " + node.value);
          inOrder(node.right);
    }
}
```

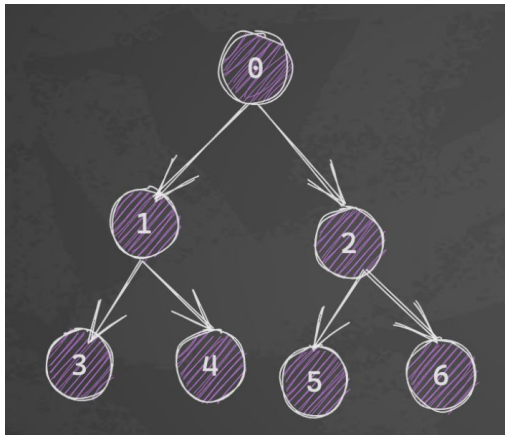
Root -> Left -> Right (0 - 1 - 3 - 4 - 2 - 5 - 6)

```
public void preOrder(TNode node) {
    if (node != null) {
        System.out.print(" " + node.value);
        preOrder(node.left);
        preOrder(node.right);
    }
}
```

Left -> Right -> Root (3 - 4 - 1 - 5 - 6 - 2 - 0)

```
public void postOrder(TNode node) {
    if (node != null) {
        postOrder(node.left);
        postOrder(node.right);
        System.out.print(" " + node.value);
    }
}
```

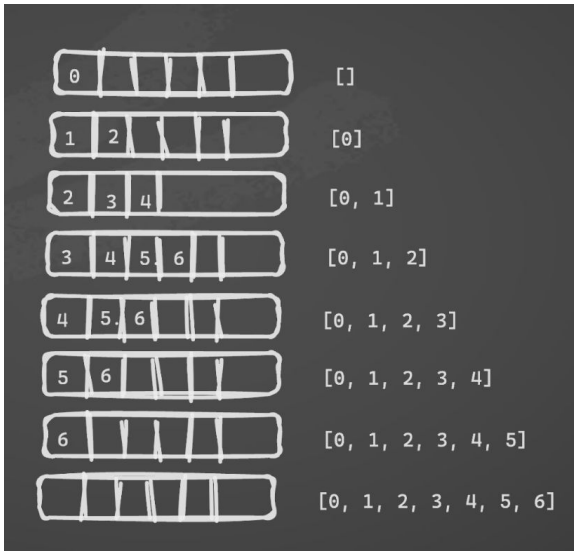
Tree Traversals



Level Order Traversal: 0 1 2 3 4 5 6

```
public void levelOrder(TNode root) {
    if (root == null) {
        return;
    }

    Queue<TNode> nodes = new LinkedList<>();
    nodes.add(root);

    while (!nodes.isEmpty()) {

        TNode node = nodes.remove();

        System.out.print(" " + node.value);

        if (node.left != null) {
            nodes.add(node.left);
        }

        if (node.right != null) {
            nodes.add(node.right);
        }
    }
}
```

```
class BinaryTree {
 class TNode {
  int value;
  TNode left;
  TNode right;

  TNode(int v) {
   value = v;
   left = null;
   Right = null;
  }
 }

 TNode root;
 ….
}
```



Trees

Common interview problems [There are TONS of them!!!]

- Search problems (Smallest element, K-th smallest/largest element)
- 2-sum, 3-sum
- Different types of traversals
- Find lowest common ancestors of 2 nodes in a tree
- Print different views of a tree (Left view, Right view, etc)
- Check if a binary tree is a BST
- Check if a binary tree is balanced
- ....

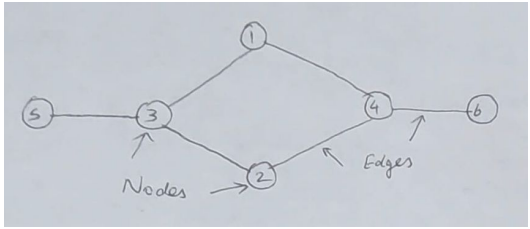
Great resource for Trees related problems

https://www.geeksforgeeks.org/binary-tree-data-structure/?ref=shm
https://www.geeksforgeeks.org/binary-search-tree-data-structure/?ref=shm

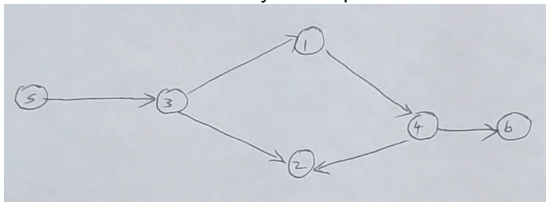
Graph

- Non linear data structure
- Has nodes connected by edges
- Types of graphs
  - Directed graphs
  - Undirected graphs
  - Cyclic graphs
  - Acyclic graphs
  - Disconnected graphs
  - Etc
- Basic Operations on graph
  - Insertion of nodes and edges
  - Deletion of nodes and edges
  - Search in a graph
  - Traversal of a graph
    - Breadth First Search
    - Depth First Search

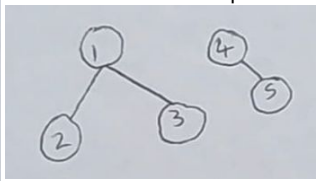
Undirected Graph

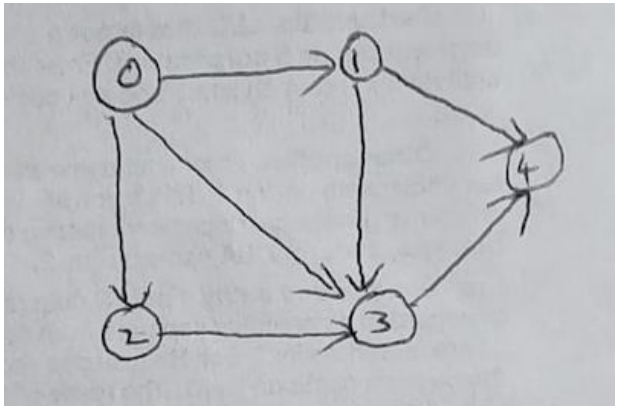


Directed/Acyclic Graph



Disconnected Graph





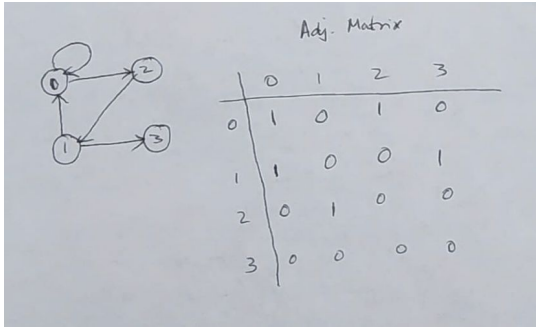
BFS (Similar to Level Order Traversal of a tree)
0 1 2 3 4

DFS
0 1 4 3 2

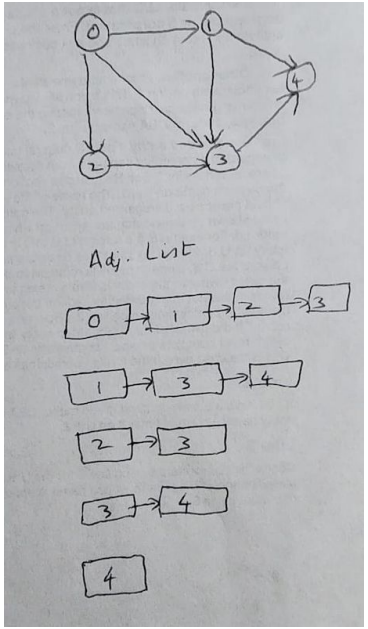
Graph

- Graph Representation
  - Adjacency Matrix
  - Adjacency List

Adjacency Matrix



Adjacency List



- Basic Edge Operations and time complexities

Operations	**Adj. Matrix**	**Adj. List**
Adding an edge	O(1)	O(1)
Removing an edge	O(1)	O(N)
Initializing	O(N*N)	O(N)

To read more about graphs

https://www.geeksforgeeks.org/introduction-to-graphs-data-structure-and-algorithm-tutorials/

DS & Algos Interview Prep Strategy

1. Pick the programming language of choice for interviews. Java and python are most common.
2. Go to the basics. Nail the basics of the programming language.
3. Get a strong understanding of all basic data structures and learn how to code the basic operations on these data structures.
    a. Book that I found useful: Elements of Programming Interviews in Java
    b. Academic book that I have referred to in the past: Introduction to Algorithms
    c. Website that I extensively used: GeeksforGeeks
4. Start solving leetcode style problems. Start with easy problems and progress to difficult ones. Stay consistent and practice.
    a. The more problems you solve, the more you will start seeing patterns
5. Arrays, linked lists, stacks, queues, trees, hash maps and string manipulations are the most commonly asked problems based on my experience

DS & Algos Interview Prep Strategy

6. Attend our WWCode Algorithms and Interview Prep series and refer their github repo for resources
7. Do mock interviews.
8. Solve at least 2 or 3 problems every day for an extended period.
9. Start giving interviews. Apply for companies that you are least interested first, so that you can consider them as practice rounds even if you don't clear them
10. You will get the hang of it all soon!!!

DS & Algos Interview Tips

1. Decide on the programming language of choice and communicate it to the interviewer
2. Listen to the problem keenly and ask follow up clarification questions
3. Think of solutions. You can start with the basic brute force solution which is not at all efficient.
4. Explain your thought process using pseudo code(if needed)
5. Ask if the interviewer is happy with this approach or do they want you to think a more efficient solution.
6. Finalize the approach once the interviewer and you are satisfied with the time and space complexity of the proposed solution

DS & Algos Interview Tips

7. Start coding
    a. Don’t forget to handle edge conditions, null checks, etc
    b. Meaningful variable names
    c. Good class design and abstractions
    d. Clean code
    e. Don’t be scared of hints
    f. Be receptive to feedback and incorporate it into your solution if it makes sense to you
    g. Walkthrough the code with basic example
    h. Add test cases
        i. Think about positive and negative test cases
        ii. Edge cases
    i. Run the code

Backend Study Group

References:

- https://www.geeksforgeeks.org/learn-data-structures-and-algorithms-dsa-tutorial/?ref=shm
- **https://www.freecodecamp.org/news/data-structures-101-graphs-a-visual-introduction-for-beginners-6d88f36ec768/**
- **https://www.indeed.com/career-advice/career-development/how-to-learn-data-structures**
- **WWCode Algos repo: https://github.com/WomenWhoCode/wwcsf-algos**
- **Detailed presentation on Trees**
- Join **#sf-algos** in WWCode slack space

Backend Study Group:

- Presentations on GitHub and session recordings available on WWCode YouTube channel
- Upcoming sessions:
  - February 9th, 2023 - Introduction to different roles in tech
  - Feb 16th, 2023 - Coding Interview 101
  - Feb 23rd, 2023 - System Design Series: Part 2

Women Who Code:

- Technical Tracks and Digital Events for more events
- Join the Digital mailing list for updates about WWCode
- Contacts us at: contact@womenwhocode.com
- Join our Slack workspace and join *#backend-study-group*!

You can unmute and talk or use the chat

WOMEN WHO CODE® /san-francisco