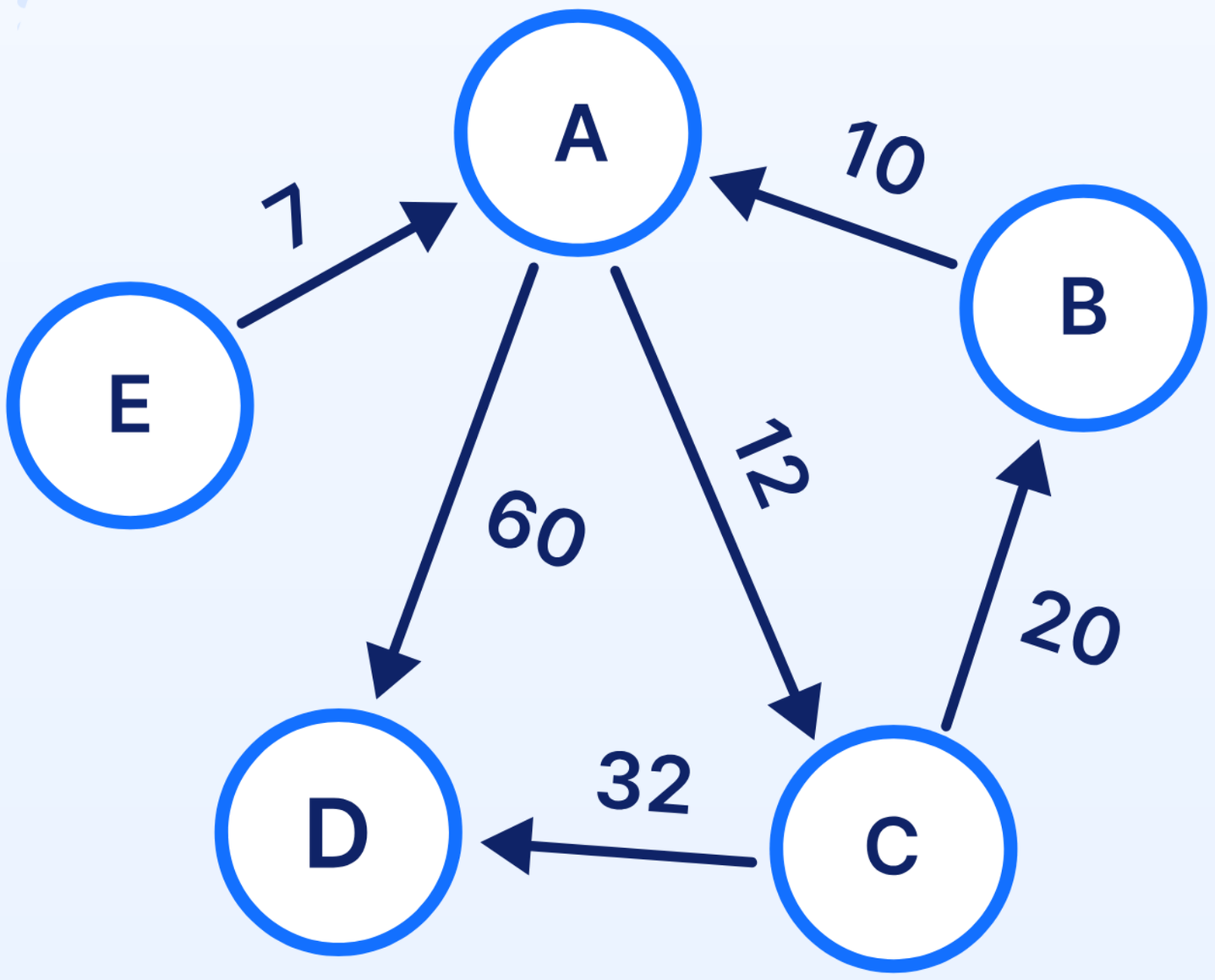
TOP 25

DSA

Questions for MAANG Interviews



For Data Engineering role

Disclaimer

Understanding Data Structures and Algorithms for data engineering can be tough, especially for working professionals.
This document is here to guide you through the key topics you need to cover if you're already in the transition phase.
It will be a valuable resource to help you navigate your journey effectively.

#1

What is a Data Structure?

A data structure is a method of presenting, storing and manipulating data in a computer system in a way that makes it easy to process in order to arrive at the desired output.

Some of them are arrays, linked list, stack, queue, trees and graphs and so on.

#2

What is the difference between a Stack and a Queue?

- **Stack:** Is based on the LIFO – Last in First Out principle. Example: Undo operation in text editors.

- **Queue:** Works on the FIFO (First in First Out) method. Example: Print queue.

#3

What is a Linked List?

In linked list structure each piece of data or node contains information along with a reference pointing to its next data item in the list.

Types:

- Singly Linked List
- Doubly Linked List
- Circular Linked List

#4

What is a Binary Search?

In a sorted array binary search functions by segmenting search intervals in two equal parts during repeated iterations to locate elements.

Steps:

- Compare the middle element with the target.
- The index gets returned if the match occurs.
- To search smaller targets use left subarray regions first and right subarray sections for larger targets.

#5

What is a Hash Table?

It gives users a storage space where key-value pairs are kept using an array index generated by a hash function.

- **Example:** Dictionary in Python.

It establishes a mechanism which provides rapid operations for value insertions and key lookups and value deletions.

#6

Difference between BFS and DFS?

- **BFS (Breadth-First Search):** All neighbouring locations receive exploration before further depth exploration begins. Uses a queue.

- **DFS (Depth-First Search):** An algorithm that explores a network branch to its deepest point before backtracking, using a stack or recursion.

#7

What is a Binary Search?

A tree data structure called a binary tree features nodes that can reach a maximum of two child elements which we identify as left and right children.

#8

What is Dynamic Programming?

Through dynamic programming techniques complex problems become solvable by splitting them into overlapping subparts that store intermediary results to prevent repetitive computations.

#9

What is the difference between a Heap and a Priority Queue?

- **Heap:** A binary tree framework where nodes hold positions higher or lower than their child nodes relative to each other.

- **Priority Queue:** Elements dequeue by priority. Heaps are the common implementation for priority queues.

#10

What is the Time Complexity of Binary Search?

The time complexity of the binary search is measured to be O(log n), since there are only two partitions of the search space at each step.

#11

What are Graphs?

A graph is a set of points called nodes or vertices and a connecting line segment called edges.

Types:

- Directed and Undirected Graphs
- Weighted and Unweighted Graphs

#12

What is a Trie?

Trie is a tree like structure used to store the dynamic collection of strings. It is used in the circumstances such as auto completion, or looking up a word in a dictionary.

#13

What is the difference between Merge Sort and Quick Sort?

- **Merge Sort:** It divides the whole array into equal halves, sorts them, and merging them.
  Time complexity: **O(n log n).**

- **Quick Sort:** Chooses a pivot, partitions elements, and sorts recursively. Avg. time: **O(n log n).**

#14

What are the differences between Array and Linked List?

Feature	Array	Linked List
Storage	Fixed size (contiguous)	Dynamic size (nodes)
Access	Random access	Sequential access
Insertion	Expensive (shifting)	Efficient

#15

What is Recursion?

Recursion is a way to solve a problem by calling a function which solved a subtype of the problem and that function recursively calls itself until reached a base condition.

- **Example:** Calculating factorial:

```
factorial(n) n x factorial(n 1)
```

#16

Reverse a String

- **Problem:** Reverse a given string.

```java
public class ReverseString {
    public static String reverse(String str) {
        StringBuilder reversed = new StringBuilder(str);
        return reversed.reverse().toString();
    }
```

```
    public static void main(String[] args) {
        System.out.println(reverse("hello")); // Output: olleh
    }
}
```

#17

Find the Maximum Element in an Array

- **Problem:** Find the largest element in an array.

```
public class MaxElement {
    public static int findMax(int[] arr) {
        int max = arr[0];
        for (int num : arr) {
            if (num > max) {
                max = num;
            }
        }
        return max;
    }

    public static void main(String[] args) {
        int[] arr = {1, 5, 3, 9, 2};
        System.out.println(findMax(arr)); // Output: 9
    }}
```

#18

Check for Palindrome

- **Problem:** Check if a string is a palindrome.

```
public class Palindrome {
    public static boolean isPalindrome(String str) {
        int left = 0, right = str.length() - 1;
        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    public static void main(String[] args) {
        System.out.println(isPalindrome("racecar")); // Output
        System.out.println(isPalindrome("hello"));   // Output
    }}
```

- **Diagram:**

```
Input: racecar
L       ->    <-    R
Output: true
```

#19

Merge Two Sorted Arrays

- **Problem:** Merge two sorted arrays into one sorted array.

```java
import java.util.*;

public class MergeSortedArrays {
    public static int[] merge(int[] arr1, int[] arr2) {
        int[] result = new int[arr1.length + arr2.length];
        int i = 0, j = 0, k = 0;

        while (i < arr1.length && j < arr2.length) {
            if (arr1[i] < arr2[j]) {
                result[k++] = arr1[i++];
            } else {
                result[k++] = arr2[j++];
            }
        }

        while (i < arr1.length) {
            result[k++] = arr1[i++];
        }

        while (j < arr2.length) {
            result[k++] = arr2[j++];
        }

        return result;
    }
```

```
    }

    public static void main(String[] args) {
        int[] arr1 = {1, 3, 5};
        int[] arr2 = {2, 4, 6};
        System.out.println(Arrays.toString(merge(arr1, arr2))); // Output
    }
```

#20

Two Sum Problem

- **Problem:** Find indexes of two numbers in an array that add up to a target.

```
import java.util.*;

public class TwoSum {
    public static int[] findTwoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[]{map.get(complement), i};
            }
            map.put(nums[i], i);
        }
        return new int[]{}; // No solution
    }
```

```
    public static void main(String[] args) {
        int[] nums = {2, 7, 11, 15};
        int target = 9;
        System.out.println(Arrays.toString(findTwoSum(nums, target))); // Output
    }
}
```

- **Diagram:**

```
Nums: [2, 7, 11, 15]
Target: 9
Output: [0, 1]
```

#21

Find the Missing Number

- **Problem:** Find the missing number in an array of size n containing numbers from 0 to n.

```
public class MissingNumber {
    public static int findMissingNumber(int[] nums) {
        int n = nums.length;
        int totalSum = n * (n + 1) / 2;
        int actualSum = 0;

        for (int num : nums) {
            actualSum += num;
        }
```

```
        return totalSum - actualSum;
    }

    public static void main(String[] args) {
        int[] nums = {0, 1, 3};
        System.out.println(findMissingNumber(nums));
    }
}
```

#22

Move Zeroes to End

- **Problem:** Move Zeroes to EndMove all zeroes in an array to the end while maintaining the order of other elements.

```
import java.util.*;

public class MoveZeroes {
    public static void moveZeroes(int[] nums) {
        int index = 0;

        for (int num : nums) {
            if (num != 0) {
                nums[index++] = num;
            }
        }

        while (index < nums.length) {
            nums[index++] = 0;
        }
    }
```

```
    public static void main(String[] args) {
        int[] nums = {0, 1, 0, 3, 12};
        moveZeroes(nums);
        System.out.println(Arrays.toString(nums)); // Output: [1, 3, 12, 0, 0]
    }
}
```

- **Diagram:**

```
Input: [0, 1, 0, 3, 12]
output: [1, 3, 12, 0, 0]
```

#23

Rotate Array

- **Problem:** Rotate an array to the right by k steps.

```
import java.util.*;

public class RotateArray {
    public static void rotate(int[] nums, int k) {
        k %= nums.length;
        reverse(nums, 0, nums.length - 1);
        reverse(nums, 0, k - 1);
        reverse(nums, k, nums.length - 1);
    }
```

```
    private static void reverse(int[] nums, int start, int end) {
        while (start < end) {
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4, 5, 6, 7};
        rotate(nums, 3);
        System.out.println(Arrays.toString(nums)); // Output: [5, 6, 7, 1, 2, 3, 4]
    }
}
```

#24

Find First Unique Character in a String

- **Problem:** Return the index of the first non-repeating character in a string.

```
import java.util.*;

public class FirstUniqueCharacter {
    public static int firstUniqChar(String s) {
        Map<Character, Integer> charCount = new HashMap<>();
        for (char c : s.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
```

```
        }

        for (int i = 0; i < s.length(); i++) {
            if (charCount.get(s.charAt(i)) == 1) {
                return i;
            }
        }

        return -1; // No unique character found
    }

    public static void main(String[] args) {
        System.out.println(firstUniqChar("leetcode")); // Output: 0
        System.out.println(firstUniqChar("loveleetcode")); // Output: 2
    }
}
```

#25

Valid Parentheses

- **Problem:** Check if a string containing parentheses is valid.

```
import java.util.Stack;

public class ValidParentheses {
    public static boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
```

```
        for (char c : s.toCharArray()) {
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            } else {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if ((c == ')' && top != '(') ||
                    (c == '}' && top != '{') ||
                    (c == ']' && top != '[')) {
                    return false;
                }
            }
        }

        return stack.isEmpty();
    }

    public static void main(String[] args) {
        String s = "([])"; // Example input
        System.out.println(isValid(s)); // Output: true
    }
}
```

BOSSCODER ACADEMY

WHY BOSSCODER?

2200+ Alumni placed at Top Product-based companies.

More than **136% hike** for every **2 out of 3** working professional.

Average package of **24LPA.**

The syllabus is most **up-to-date** and the list of problems provided covers **all important topics.**

Lavanya
Meta

Course is very well **structured** and **streamlined** to crack any **MAANG** company

Rahul .
Google

EXPLORE MORE