**Capgemini Technical Interview:**

1. What do you mean by a Data structure?

Ans: In computer science, a **data structure** is a **data** organization, management, and storage format that enables efficient access and modification. More precisely, a **data structure** is a collection of **data** values, the relationships among them, and the functions or operations that **can** be applied to the **data**.

1. What are some of the applications of DS?

Ans:

1. Application of Arrays: Arrays are the simplest data structures that stores items of the same data type. A basic application of Arrays can be storing data in tabular format. For example, if we wish to store the contacts on our phone, then the software will simply place all our contacts in an array.
2. Application of Linked Lists: A linked list is a sequence data structure, which connects elements, called nodes, through links.
3. Application of Stack: A stack is a data structure which uses LIFO order.
4. Application of Queue: A queue is a data structure which uses FIFO order.
5. Application of Graph: Graph is a data structure where data is stored in a collection of interconnected vertices (nodes) and edges (paths).
6. Application of Tree: Trees are hierarchical structures having a single root node.
7. Application of Hash Tables: Hash Tables are store data in key-value pairs. It only stores data which has a key associated with it. Inserting and Searching operations are easily manageable while using Hash Tables.
8. Application of Heap: A Heap is a special case of a binary tree where the parent nodes are compared to their children with their values and are arranged accordingly.
9. What are the advantages of a Linked list over an array?

Ans: The principal benefit of a linked list over a conventional array is that the list elements can be easily inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk, while restructuring an array at run-time is a much more.

1. Write the syntax in C to create a node in the singly linked list.

Ans:

#include <stdio.h>

#include <stdlib.h>

//Represent a node of singly linked list.

struct node{

int data;

struct node \*next;

};

//Represent the head and tail of the singly linked list.

1. What is the use of a doubly-linked list when compared to that of a singly linked list?

Ans: Doubly linked list allows element two-way traversal. On other hand doubly linked list can be used to implement stacks as well as heaps and binary trees. Singly linked list is preferred when we need to save memory and searching is not required as pointer of single index is stored.

1. What is the difference between an Array and Stack?

Ans:

|  |  |
| --- | --- |
| Array | Stack |
| In the array the elements belong to indexes, i.e., if you want to get into the fourth element you have to write the variable name with its index or location within the square bracket eg arr[4] | Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. |
| Insertion and deletion in array can be done at any index in the array. | Insertion and deletion in stacks takes place only from one end of the list called the top. |
| Array has a fixed size. | Stack has a dynamic size. |
| Array contains elements of same data type. | Stack can contain elements of different data type. |
| We can do both linear and Binary search | We can do only linear search |
|  |  |

1. What are the minimum number of Queues needed to implement the priority queue?

Ans: 2 queues. one is used for storing data... another is used for priorities. Priority queues r applied using 2-D array where it has two rows one for element and second for priority, so minimum numbers of queues are needed to implement are two.

1. What are the different types of traversal techniques in a tree?

Ans:

Inorder Traversal. Inorder Traversal is the one the most used variant of DFS (Depth First Search) Traversal of the tree.

Preorder Traversal. Preorder Traversal is another variant of DFS.

Postorder Traversal.

Level Order Traversal.

1. Why it is said that searching a node in a binary search tree is efficient than that of a simple binary tree?

Ans: It supports three main operations: searching of elements, insertion of elements, and deletion of elements. Binary Search Tree allows for fast retrieval of elements stored in the tree as each node key is thoroughly compared with the root node, which discards half of the tree.

1. What are the applications of Graph DS?

Ans: A graph is a non-linear data structure, which consists of vertices (or nodes) connected by edges (or arcs) where edges may be directed or undirected. In Computer science graphs are used to represent the flow of computation.

**Computer Science:** In computer science, graph is used to represent networks of communication, data organization, computational devices etc.

**Physics and Chemistry:** Graph theory is also used to study molecules in chemistry and physics.

**Social Science:** Graph theory is also widely used in sociology.

**Mathematics:** In this, graphs are useful in geometry and certain parts of topology such as knot theory.

**Biology:** Graph theory is useful in biology and conservation efforts.

Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the DSA Self-Paced Course at a student-friendly price and become industry ready.

1. Can we apply Binary search algorithm to a sorted Linked list?

Ans: Yes, Binary search is possible on the linked list if the list is ordered and you know the count of elements in list. But While sorting the list, you can access a single element at a time through a pointer to that node i.e. either a previous node or next node.

1. When can you tell that a Memory Leak will occur?

Ans: In computer science, a memory leak is a type of resource leak that occurs when a computer program incorrectly manages memory allocations in a way that memory which is no longer needed is not released. A memory leak may also happen when an object is stored in memory but cannot be accessed by the running code.

1. How will you check if a given Binary Tree is a Binary Search Tree or not?

Ans:

1. If a node is a left child, then its key and the keys of the nodes in its right subtree are less than its parent’s key.
2. If a node is a right child, then its key and the keys of the nodes in its left subtree are greater than its parent’s key.
3. Which data structure is ideal to perform recursion operation and why?

Ans: Stack. Because of its LIFO (Last in First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls. Every recursive function has its equivalent iterative (non-recursive) function.

1. What are some of the most important applications of a Stack?

Ans:

**Expression Handling –**

Infix to Postfix or Infix to Prefix Conversion –

The stack can be used to convert some infix expression into its postfix equivalent, or prefix equivalent. These postfix or prefix notations are used in computers to express some expressions. These expressions are not so much familiar to the infix expression, but they have some great advantages also. We do not need to maintain operator ordering, and parenthesis.

Postfix or Prefix Evaluation –

After converting into prefix or postfix notations, we have to evaluate the expression to get the result. For that purpose, also we need the help of stack data structure.

**Backtracking Procedure –**

**Ba**cktracking is one of the algorithms designing technique. For that purpose, we dive into some way, if that way is not efficient, we come back to the previous state and go into some other paths. To get back from current state, we need to store the previous state. For that purpose, we need stack. Some examples of backtracking are finding the solution for Knight Tour problem or N-Queen Problem etc.

1. Convert the below given expression to its equivalent Prefix and Postfix notations.

Ans:

**Algorithm for Prefix to Postfix:**

1. Read the Prefix expression in reverse order (from right to left)
2. If the symbol is an operand, then push it onto the Stack.
3. If the symbol is an operator, then pop two operands from the Stack.
4. Repeat the above steps until end of Prefix expression.

e.g.: Input: Prefix: \*+AB-CD

Output: Postfix: AB+CD-\*

Explanation: Prefix to Infix: (A+B) \* (C-D)

Infix to Postfix: AB+CD-\*

1. Sorting a stack using a temporary stack?

Ans:

**We follow this algorithm-**

1. Create a temporary stack say tmpStack.
2. While input stack is NOT empty do this:
3. Pop an element from input stack call it temp
4. while temporary stack is NOT empty and top of temporary stack is greater than temp,

pop from temporary stack and push it to the input stack

1. push temp in temporary stack
2. The sorted numbers are in tmpStack
3. Program to reverse a queue?

Ans:

**Algorithm:**

1. enqueue(x): Add an item x to rear of queue.
2. dequeue (): Remove an item from front of queue.
3. empty (): Checks if a queue is empty or not.

**Approach:** For reversing the queue one approach could be to store the elements of the queue in a temporary data structure in a manner such that if we re-insert the elements in the queue they would get inserted in reverse order. According to the approach, the data-structure should have the property of ‘LIFO’ as the last element to be inserted in the data structure should actually be the first element of the reversed queue. The stack could help in approaching this problem. This will be a two-step process:

1. Pop the elements from the queue and insert into the stack. (Topmost element of the stack is the last element of the queue)
2. Pop the elements of the stack to insert back into the queue. (The last element is the first one to be inserted into the queue)

// Java program to reverse a Queue

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

// Java program to reverse a queue

public class Queue\_reverse {

static Queue<Integer> queue;

// Utility function to print the queue

static void Print()

{

while (!queue.isEmpty()) {

System.out.print( queue.peek() + ", ");

queue.remove();

}

}

// Function to reverse the queue

static void reversequeue()

{

Stack<Integer> stack = new Stack<>();

while (!queue.isEmpty()) {

stack.add(queue.peek());

queue.remove();

}

while (!stack.isEmpty()) {

queue.add(stack.peek());

stack.pop();

}

}

// Driver code

public static void main(String args[])

{

queue = new LinkedList<Integer>();

queue.add(10);

queue.add(20);

queue.add(30);

queue.add(40);

queue.add(50);

queue.add(60);

queue.add(70);

queue.add(80);

queue.add(90);

queue.add(100);

reversequeue();

Print();

}

}

1. Program to reverse first k elements of a queue?

Ans:

standard operations are allowed on queue:

1. enqueue(x): Add an item x to rear of queue
2. dequeue (): Remove an item from front of queue
3. size (): Returns number of elements in queue.
4. front (): Finds front item.

Examples:

Input: Q = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

k = 5

Output: Q = [50, 40, 30, 20, 10, 60, 70, 80, 90, 100]

// Java program to reverse first k elements

// of a queue.

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

public class Reverse\_k\_element\_queue {

static Queue<Integer> queue;

// Function to reverse the first

// K elements of the Queue

static void reverseQueueFirstKElements(int k)

{

if (queue.isEmpty() == true

|| k > queue.size())

return;

if (k <= 0)

return;

Stack<Integer> stack = new Stack<Integer>();

// Push the first K elements into a Stack

for (int i = 0; i < k; i++) {

stack.push(queue.peek());

queue.remove();

}

// Enqueue the contents of stack

// at the back of the queue

while (!stack.empty()) {

queue.add(stack.peek());

stack.pop();

}

// Remove the remaining elements and enqueue

// them at the end of the Queue

for (int i = 0; i < queue.size() - k; i++) {

queue.add(queue.peek());

queue.remove();

}

}

// Utility Function to print the Queue

static void Print()

{

while (!queue.isEmpty()) {

System.out.print(queue.peek() + " ");

queue.remove();

}

}

// Driver code

public static void main(String args[])

{

queue = new LinkedList<Integer>();

queue.add(10);

queue.add(20);

queue.add(30);

queue.add(40);

queue.add(50);

queue.add(60);

queue.add(70);

queue.add(80);

queue.add(90);

queue.add(100);

int k = 5;

reverseQueueFirstKElements(k);

Print();

}

}

1. Program to return the nth node from the end in a linked list?

Ans:

**Method 1 (Use length of linked list)**

1. Calculate the length of Linked List. Let the length be len.
2. Print the (len – n + 1)th node from the beginning of the Linked List.

**Double pointer concept:**

First pointer is used to store the address of the variable and second pointer used to store the address of the first pointer. If we wish to change the value of a variable by a function, we pass pointer to it. And if we wish to change value of a pointer (i. e., it should start pointing to something else), we pass pointer to a pointer.

// Simple Java program to find n'th node from end of linked list

class LinkedList {

Node head; // head of the list

/\* Linked List node \*/

class Node {

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

/\* Function to get the nth node from the last of a

linked list \*/

void printNthFromLast(int n)

{

int len = 0;

Node temp = head;

// 1) count the number of nodes in Linked List

while (temp != null) {

temp = temp.next;

len++;

}

// check if value of n is not more than length of

// the linked list

if (len < n)

return;

temp = head;

// 2) get the (len-n+1)th node from the beginning

for (int i = 1; i < len - n + 1; i++)

temp = temp.next;

System.out.println(temp.data);

}

/\* Inserts a new Node at front of the list. \*/

public void push(int new\_data)

{

/\* 1 & 2: Allocate the Node &

Put in the data\*/

Node new\_node = new Node(new\_data);

/\* 3. Make next of new Node as head \*/

new\_node.next = head;

/\* 4. Move the head to point to new Node \*/

head = new\_node;

}

/\*Driver program to test above methods \*/

public static void main(String[] args)

{

LinkedList llist = new LinkedList();

llist.push(20);

llist.push(4);

llist.push(15);

llist.push(35);

llist.printNthFromLast(4);

}

}

1. Reverse a linked list

Ans:

Given pointer to the head node of a linked list, the task is to reverse the linked list. We need to reverse the list by changing links between nodes.

Examples:

Input: Head of following linked list

1->2->3->4->NULL

Output: Linked list should be changed to,

4->3->2->1->NULL

**Iterative Method**

1. Initialize three pointers prev as NULL, curr as head and next as NULL.
2. Iterate trough the linked list. In loop, do following.

// Before changing next of current,

// store next node

next = curr->next

// Now change next of current

// This is where actual reversing happens

curr->next = prev

// Move prev and curr one step forward

prev = curr

curr = next

**Program:**

// Java program for reversing the linked list

class LinkedList {

static Node head;

static class Node {

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

/\* Function to reverse the linked list \*/

Node reverse(Node node)

{

Node prev = null;

Node current = node;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

node = prev;

return node;

}

// prints content of double linked list

void printList(Node node)

{

while (node != null) {

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

node = node.next;

}

}

public static void main(String[] args)

{

LinkedList list = new LinkedList();

list.head = new Node(85);

list.head.next = new Node(15);

list.head.next.next = new Node(4);

list.head.next.next.next = new Node(20);

System.out.println("Given Linked list");

list.printList(head);

head = list.reverse(head);

System.out.println("");

System.out.println("Reversed linked list ");

list.printList(head);

}

}

1. Replace each element of the array by its rank in the array

Ans:

The rank of an element is defined as the distance between the element with the first element of the array when the array is arranged in ascending order. If two or more are same in the array then their rank is also the same as the rank of the first occurrence of the element.

For Example: Let the given array arr[] = {2, 2, 1, 6}, then rank of elements is given by:

sorted array is:

arr[] = {1, 2, 2, 6}

Rank(1) = 1 (at index 0)

Rank(2) = 2 (at index 1)

Rank(2) = 2 (at index 2)

Rank(6) = 4 (at index 3)

Example:

Input: arr[] = [100, 5, 70, 2]

Output: [4, 2, 3, 1]

Explanation:

Rank of 2 is 1, 5 is 2, 70 is 3 and 100 is 4.

Programme:

// Java program for the above approach

import java.util.\*;

class GFG {

// Function to assign rank to

// array eleements

static void changeArr(int[] input)

{

// Copy input array into newArray

int newArray[]

= Arrays

.copyOfRange(input,

0,

input.length);

// Sort newArray[] in ascending order

Arrays.sort(newArray);

int i;

// Map to store the rank of

// the array element

Map<Integer, Integer> ranks

= new HashMap<>();

int rank = 1;

for (int index = 0;

index < newArray.length;

index++) {

int element = newArray[index];

// Update rank of element

if (ranks.get(element) == null) {

ranks.put(element, rank);

rank++;

}

}

// Assign ranks to elements

for (int index = 0;

index < input.length;

index++) {

int element = input[index];

input[index]

= ranks.get(input[index]);

}

}

// Driver Code

public static void main(String[] args)

{

// Given array arr[]

int[] arr = { 100, 2, 70, 2 };

// Function Call

changeArr(arr);

// Print the array elements

System

.out

.println(Arrays

.toString(arr));

}

}

1. Check if a given graph is a tree or not

Ans: We can simply find it by checking the criteria of a tree. A tree will not contain a cycle, so if there is any cycle in the graph, it is not a tree. We can check it using another approach, if the graph is connected and it has V-1 edges, it could be a tree. Here V is the number of vertices in the graph.

1. Find out the Kth smallest element in an unsorted array

Ans:

Method 1:

A simple solution is to sort the given array using a O(N log N) sorting algorithm like Merge Sort, Heap Sort, etc and return the element at index k-1 in the sorted array.

Programme:

// Java code for kth smallest element

// in an array

import java.util.Arrays;

import java.util.Collections;

class GFG {

// Function to return k'th smallest

// element in a given array

public static int kthSmallest(Integer[] arr,

int k)

{

// Sort the given array

Arrays.sort(arr);

// Return k'th element in

// the sorted array

return arr[k - 1];

}

// driver program

public static void main(String[] args)

{

Integer arr[] = new Integer[] { 12, 3, 5, 7, 19 };

int k = 2;

System.out.print("K'th smallest element is " + kthSmallest(arr, k));

}

}

**Method 2:**

We can find k’th smallest element in time complexity better than O(N Log N). A simple optomization is to create a Min Heap of the given n elements and call extractMin() k times.

1. How to find the shortest path between two vertices? (sir you are not mentioned directed graph, undirected or unweighted graph)

Ans:

1. Input the graph.
2. Input the source and destination nodes.
3. Find the paths between the source and the destination nodes.
4. Find the number of edges in all the paths and return the path having the minimum number of edges.