1) **What do you mean by a Data structure?**

Ans. Data structure refers to methods of organizing units of data within larger data sets. Achieving and maintaining specific data structures help improve data access and value. Data structures also help programmers implement various programming tasks. For example, a very basic example of a data structure is an array, in which multiple data bits are coordinated into a group sharing a common label. This helps programs call these data bits or perform other work on the data set as a whole.

==================================================================

2) **What are some of the applications of DS?**

Ans. Arrays, Stack, Queue, Graphs, Linked List

==================================================================

3) **What are the advantages of a Linked list over an array?**

Ans. Arrays: 1. Fixed Size

2. Insertion and deletion is difficult.

3. Memory wastage.

Linked List: 1. Dynamic size

2. Ease of Insertion and deletion

3. No memory wastage

=================================================================

4) **Write the syntax in C to create a node in the singly linked list.**

Ans. struct **node**

{

int data;

struct **node** \*next;

};

==================================================================

5) **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 traversal. It can also be used to implement stacks as well as heaps and binary trees. If we need better performance while searching and memory is not a limitation in this case DLL is more preferred.

6) **What is the difference between an Array and Stack?**

Ans.

|  |  |
| --- | --- |
| Stack | Array |
| Stack are based on Last In First Out principle. | 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. |
| Insertion and deletion in stacks takes place only from one end of the list called the top. | 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. |
| Stack has a dynamic size. | Array has a fixed size. |
| Stack can contain elements of different data type. | Array contains elements of same data type. |

==================================================================

7) **What are the minimum number of Queues needed to implement the priority queue?**

Ans. Priority Queue is an extension of queue with following properties.

1) Every item has a priority associated with it.

2) An element with high priority is de-queued before an element with low priority.

3) If two elements have the same priority, they are served according to their order in the queue.

The minimum number of queues needed in this case is two. One queue is intended for sorting priorities while the other queue is intended for actual storage of data.

==================================================================

8) **What are the different types of traversal techniques in a tree?**

Ans. There can be three types of traversal:-

* **In-order traversal-** First, visit all the nodes in the left sub-tree, then the root node,visit all the nodes in the right sub-tree
* **Pre-order traversal-** Visit root node, visit all the nodes in the left sub-tree, visit all the nodes in the right sub-tree
* **Post-order traversal-** Visit all the nodes in the left sub-tree, visit all the nodes in the right sub-tree, visit the root node

==================================================================

9) **Why it is said that searching a node in a binary search tree is efficient than that of a simple binary tree?**

Ans. While Binary search tree is data structure which uses the concept of binary search. Binary search tree has a special quality that every node has at most 2 nodes and left child of every node has less value than node and right child has more value than node. By this method it become easy to search a node in a tree.

10) **What are the applications of Graph DS?**

Ans.

1. In Computer science graphs are used to represent the flow of computation.
2. In Facebook, users are considered to be the vertices and if they are friends then there is an edge running between them. Facebook’s Friend suggestion algorithm uses graph theory. Facebook is an example of undirected graph.
3. In World Wide Web, web pages are considered to be the vertices.
4. In Operating System, we come across the Resource Allocation Graph where each process and resources are considered to be vertices. Edges are drawn from resources to the allocated process, or from requesting process to the requested resource. If this leads to any formation of a cycle then a deadlock will occur.

11) **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.

==================================================================

12) **When can you tell that a Memory Leak will occur?**

Ans. Memory leak occurs when we create a memory in the heap and forget to delete it after using it. It may also occur 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.

13) **How will you check if a given Binary Tree is a Binary Search Tree or not?**

Ans. Binary search trees have following properties:

1. The left sub-tree of a particular node will always contain nodes whose keys are less than that node’s key.
2. The right sub-tree of a particular node will always contain nodes with keys greater than that node’s key.
3. The left and right sub-tree of a particular node will also, in turn, be binary search trees.

14) **Which data structure is ideal to perform recursion operation and why?**

Ans. Stack has the 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.

15) **What are some of the most important applications of a Stack?**

Ans. The Stack is last in First out (LIFO) data structure. This data structure has some important applications in different aspect. These are like below –

1. Stacks can be used for expression evaluation.
2. Stacks can be used to check parenthesis matching in an expression.
3. Stacks can be used for Conversion from one form of expression to another.
4. Stacks can be used for Memory Management.
5. Stack data structures are used in backtracking problems.

16) **Convert the below given expression to its equivalent Prefix and Postfix notations?**

Ans. Expression not given

The steps for conversation from Prefix to postfix are:

1. Read the Prefix expression in reverse order
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

String = operand1 + operand2 + operator

1. Push the resultant string back to Stack
2. Repeat the above steps until end of Prefix expression.

17) **Sorting a stack using a temporary stack**

Ans.

package assignment;

import java.util.\*;

public class stackSort {

public static Stack<Integer> s(Stack<Integer> input) {

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

while(!input.isEmpty()){

int temp = input.pop();

while(!tempStack.isEmpty() && tempStack.peek() > temp) {

input.push(tempStack.pop());

}

tempStack.push(temp);

}

return tempStack;

}

public static void main(String args[]) {

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

input.add(24);

input.add(3);

input.add(51);

input.add(88);

input.add(92);

input.add(63);

Stack<Integer> tempStack = s(input);

System.out.println("Sorted numbers are:");

while (!tempStack.empty()) {

System.out.print(tempStack.pop()+" ");

}

}

}

==================================================================

18) **Program to reverse a queue**

Ans. import java.util.LinkedList;  
import java.util.Queue;  
import java.util.Stack;  
class QueueReverse {  
 static Queue<Integer> queue;  
 static void Print()  
 {  
 while (!queue.isEmpty()) {  
 System.out.print( queue.peek() + ", ");  
 queue.remove();  
 }  
 }  
 static void reverseq()  
 {  
 Stack<Integer> stack = new Stack<>();  
 while (!queue.isEmpty()) {  
 stack.add(queue.peek());  
 queue.remove();  
 }  
 while (!stack.isEmpty()) {  
 queue.add(stack.peek());  
 stack.pop();  
 }  
 }  
 public static void main(String[] args)  
 {  
 queue = new LinkedList<>();  
 queue.add(11);  
 queue.add(22);  
 queue.add(33);  
 queue.add(44);  
 queue.add(55);  
 queue.add(66);  
 queue.add(77);  
 queue.add(88);  
 queue.add(99);  
 queue.add(100);  
 reversequeue();  
 Print();  
 }  
}

19) **Program to reverse first k elements of a queue**

Ans. import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

public class KreverseQueue

{

static Queue<Integer> queue;

static void reverseQueueFirstKElements(int k)

{

if (queue.isEmpty() == true

|| k > queue.size())

return;

if (k <= 0)

return;

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

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

stack.push(queue.peek());

queue.remove();

}

while (!stack.empty()) {

queue.add(stack.peek());

stack.pop();

}

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

queue.add(queue.peek());

queue.remove();

}

}

static void Print()

{

while (!queue.isEmpty()) {

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

queue.remove();

}

}

public static void main(String args[])

{

queue = new LinkedList<Integer>();

queue.add(1);

queue.add(2);

queue.add(3);

queue.add(4);

queue.add(5);

queue.add(6);

queue.add(7);

queue.add(8);

queue.add(9);

queue.add(10);

int k = 5;

reverseQueueFirstKElements(k);

Print();

}

}

20) **Program to return the nth node from the end in a linked list.**

class LinkedList {  
 Node head;  
 static class Node {  
 int data;  
 Node next;  
 Node(int d)  
 {  
 data = d;  
 next = null;  
 }  
 }  
 void printNthFromLast()  
 {  
 int len = 0;  
 Node temp = head;  
 while (temp != null) {  
 temp = temp.next;  
 len++;  
 }  
 if (len < 4)  
 return;  
 temp = head;  
 for (int i = 1; i < len - 4 + 1; i++)  
 temp = temp.next;  
  
 System.out.println(temp.data);  
 }  
 public void push(int new\_data)  
 {  
 Node new\_node = new Node(new\_data);  
 new\_node.next = head;  
 head = new\_node;  
 }  
 public static void main(String[] args)  
 {  
 LinkedList llist = new LinkedList();  
 llist.push(10);  
 llist.push(20);  
 llist.push(30);  
 llist.push(40);  
 llist.printNthFromLast();  
 }  
}

21) **Reverse a linked list?**

Ans. class LinkedList {

    static Node head;

    static class Node {

        int data;

        Node next;

        Node(int d) {

            data = d;

            next = null;

        }

    }

    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;

    }

    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(" Linked list");

        list.printList(head);

        head = list.reverse(head);

        System.out.println("");

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

        list.printList(head);

    }

}

==================================================================\

22) **Replace each element of the array by its rank in the array**

Ans. import java.util.Arrays;

import java.util.Map;

import java.util.TreeMap;

public class arrayRank

{

public static void transform(int[] arr)

{

Map<Integer, Integer> map = new TreeMap<>();

for (int i = 0; i < arr.length; i++) {

map.put(arr[i], i);

}

int rank = 1;

for (var val : map.values()) {

arr[val] = rank++;

}

}

public static void main(String[] args)

{

int[] A = { 10, 8, 15, 12, 6, 20, 1 };

transform(A);

System.out.println(Arrays.toString(A));

}

}

==================================================================

23) **Check if a given graph is a tree or not**

Ans. import java.io.\*;

import java.util.\*;

class GraphTree

{

private int V;

private LinkedList<Integer> adj[];

Graph (int v)

{

V = v;

adj = new LinkedList[v];

for (int i=0; i<v; ++i)

adj[i] = new LinkedList ();

}

void addEdge(int v,int w)

{

adj[v].add(w);

adj[w].add(v);

}

Boolean isCyclicUtil(int v, Boolean visited[], int parent)

{

visited[v] = true;

Integer i;

Iterator<Integer> it = adj[v].iterator();

while (it.hasNext())

{

i = it.next();

if (!visited[i])

{

if (isCyclicUtil(i, visited, v))

return true;

}

else if (i != parent)

return true;

}

return false;

}

Boolean isTree()

{

Boolean visited[] = new Boolean[V];

for (int i = 0; i < V; i++)

visited[i] = false;

if (isCyclicUtil(0, visited, -1))

return false;

for (int u = 0; u < V; u++)

if (!visited[u])

return false;

return true;

}

public static void main(String args[])

{

GraphTree g1 = new GraphTree(4);

g1.addEdge(0,1);

g1.addEdge(1, 2);

g1.addEdge(2, 3);

g1.addEdge(3, 4);

if (g1.isTree())

System.out.println("Graph is Tree");

else

System.out.println("Graph is not Tree");

GraphTree g2 = new GraphTree(5);

g2.addEdge(0,1);

g2.addEdge(1,2);

g2.addEdge(2,3);

g2.addEdge(3,4);

g2.addEdge(4,5 );

if (g2.isTree())

System.out.println("Graph is Tree");

else

System.out.println("Graph is not Tree");

}

}

==================================================================

24) **Find out the Kth smallest element in an unsorted array?**

Ans. import java.util.Arrays;

import java.util.Collections;

class small {

public static int kthSmallest(Integer[] arr, int k)

{

Arrays.sort(arr);

return arr[k - 1];

}

public static void main(String[] args)

{

Integer arr[] = new Integer[] { 19, 2, 6, 17, 12 };

int k = 2;

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

}

}

25) **How to find the shortest path between two vertices.**

1. Ans. 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.
5. In this way we will get the shortest path between the two vertices.