**Q1. What do you mean by a Data structure?**

**Ans.** Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage.

**Q2. What are some of the applications of DS?**

**Ans.** Some common Data structure and their applications are:

1. Array can be used for sorting elements, can perform matrix operation and can be used in CPU scheduling.

2. Stack is used in Expression evaluation, forward and backward feature in web browsers, syntax parsing. Used in algorithms like tower of Hanoi.

3. Queue is used when a resource is shared among multiple consumers like in cpu scheduling, disk scheduling. It is also used in Palindrome recognition.

4. Tree is used as dictionary.

5. Hash table is used for fast data lookup, database indexing, caches.

6. Graphs are used to represent networks.

**Q3. 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

**Q4. Write the syntax in C to create a node in the singly linked list.**

**Ans.**

Struct node

{

Int data;

Struct node \*next;

};

**Q5. What is the use of a doubly-linked list when compared to that of a singly**

**linked list?**

**Ans.** In singly linked list traversal is possible only in forward direction whereas the doubly linked list can be traversed both in forward and backward direction. It is more efficient in the delete and reverse operation whenever the pointer to node is given in doubly linked list.

**Q6. What is the difference between an Array and Stack?**

**Ans.**

|  |  |
| --- | --- |
| **STACK** | **ARRAY** |
| 1. 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. | 1. 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 |
| 2. Insertion and deletion in stacks take place only from one end of the list called the top. | 2. Insertion and deletion in array can be done at any index in the array. |
| 3. Stack has a dynamic and fixed size. | 3. Array has a fixed size. |
| 4. The stack can contain elements of the different data types. | 4. Array contains elements of same data type. |
| 5. Stack has only one type. | 5. Different types of Arrays are 1D, 2D, etc |

**Q7. What are the minimum number of Queues needed to implement the priority**

**queue?**

**Ans.** Minimum 2 queues are used. One for storing data and another is used for priorities.

**Q8. What are the different types of traversal techniques in a tree?**

**Ans.**

1. Preorder Traversal

2. Postorder Traversal

3. Inorder Traversal

**Q9. Why it is said that searching a node in a binary search tree is efficient than that of a simple binary tree?**

**Ans.** Binary tree is unordered hence slower in process of insertion, deletion and searching. Searching of an element is faster in Binary Search tree than binary tree due to the ordered characteristics. In binary search tree the left subtree has elements less than the nodes elements and the right subtree has elements greater than the nodes element.

**Q10. 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.

**Q11. 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.

**Q12. 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

**Q13. 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 subtree of a particular node will always contain nodes whose keys are less than that node’s key.
2. The right subtree of a particular node will always contain nodes with keys greater than that node’s key.
3. The left and right subtree of a particular node will also, in turn, be binary search trees.

**Q14. 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.

**Q15. What are some of the most important applications of a Stack?**

**Ans.** Applications of stack:

1. Expression handling
2. To check parenthesis matching.
3. Backtracking procedure
4. Memory Management.

**Q17. Sorting a stack using a temporary stack.**

**Ans.**

package assignment;

import java.util.\*;

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(34);

input.add(3);

input.add(31);

input.add(98);

input.add(92);

input.add(23);

Stack<Integer> tmpStack = s(input);

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

while (!tempStack.empty()) {

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

}

}

}

**Q18. Program to reverse a Queue.**

**Ans.**

package assignment;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

public class reverseQueue

{

static Queue<Integer> queue;

static void Print()

{

while (!queue.isEmpty()) {

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

queue.remove();

}

}

static void revqueue()

{

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<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);

revqueue();

Print();

}

}

**Q19. Program to reverse first k elements of a queue**

**Ans.**

package assignment;

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(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();

}

}

**Q20. Program to return the nth node from the end in a linked list.**

**Ans.**

package assignment;

public class linkedListNth {

Node head;

class Node {

int data;

Node next;

Node(int d) {

data = d;

next = null;

}

}

void printNthFromLast(int n) {

int len = 0;

Node temp = head;

while (temp != null) {

temp = temp.next;

len++;

}

if (len < n)

return;

temp = head;

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

linkedListNth lst = new linkedListNth();

lst.push(20);

lst.push(4);

lst.push(15);

lst.push(35);

lst.printNthFromLast(4);

}

}

**Q21. Reverse a linked list.**

**Ans.**

package assignment;

public class reverseLinkedList {

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)

{

reverseLinkedList list = new reverseLinkedList();

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);

}

}

**Q22. Replace each element of the array by its rank in the array.**

**Ans.**

package assignment;

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));

}

}

**Q23. Check if a given graph is a tree or not.**

**Ans.**

package assignment;

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");

}

}

**Q24. Find out the Kth smallest element in an unsorted array**

**Ans.**

package assignment;

import java.util.Arrays;

public class kthSmallest {

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[] { 12, 3, 5, 7, 19 };

int k = 2;

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

}

}

**Q25. How to find the shortest path between two vertices.**

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