Java Assignment-2

Q1) What do you mean by a Data structure?

Answer:

Data structures is a way of organizing and storing the data in a computer so that it can be accessed and modified efficiently. The main idea is to reduce the space and time complexities of different tasks.

Q2) What are some of the applications of Data structure?

Answer:

Some applications of Data Structure are arrays, Linked Lists, Stack, Queue, Tree and Graphs.

Q3) What are the advantages of a Linked list over an array?

Answer:

1.An array is the data structure that contains a collection of similar type data elements whereas the Linked list is considered as non-primitive data structure contains a collection of

unordered linked elements known as nodes.

2.Operations like insertion and deletion in arrays consume a lot of time. On the other hand, the performance of these operations in Linked lists is fast.

3. Arrays are of fixed size. In contrast, Linked lists are dynamic and flexible and can expandand contract its size.

4.Elements are stored consecutively in arrays whereas it is stored randomly in Linked lists.

5.In addition memory utilization is inefficient in the array. Conversely,

memory utilization is efficient in the linked list.

Q4) Write the syntax in C to create a node in the singly linked list?

Answer:

#include<stdio.h>

#include<stdlib.h>

Struct node1

{

intdata;

structnode\*next;

};

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

struct node \*head, \*tail = NULL;

void addNode (int data) {

//Create a new node

struct node \*newNode = (struct node\*) malloc (sizeof(struct node));

newNode->data = data;

newNode->next = NULL;

  if (head == NULL) {

//If list is empty, both head and tail will point to new node

head = newNode;

tail = newNode;}

else {

tail->next = newNode;

     tail = newNode;}}

//display () will display all the nodes present in the list

void display () {

    struct node \*current = head;

    if (head == NULL) {

        printf ("List is empty\n");

        return;

    }

    printf ("Nodes of singly linked list: \n");

    while (current! = NULL) {

        printf ("%d ", current->data);

        current = current->next;

 }

}

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

linked list?

Answer:

The use of doubly-linked list when compared to that of a singly linked list is that:

1.In singly linked list the complexity of insertion and deletion at a known position is O(n). Whereas, the complexity of insertion and deletion at a known position in doubly linked list is O(1).

2.Singly linked list allows traversal elements only in one way. whereas, doubly-linked list allows traversal elements in two ways.

3.Singly linked list are generally used for implementation of stacks. whereas, doubly-linked list are used for implementation of stacks as well as heaps and binary trees.

4.Singly linked list is preferred when we need to save memory and searching is not required as pointer of single index is stored. whereas, If we need better performance while searching and memory is not a limitation in this case doubly linked list is more preferred.

Q6) What is the difference between an Array and Stack?

Answer:

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. whereas, In the array the elements belong to indexes, i.e., if you want to get into the sixth element you have to write the variable name with its index or location within the square bracket eg arr[6].

2.Insertion and deletion in stacks takes place only from one end of the list called the top. whereas, Insertion and deletion in array can be done at any index in the array.

3.Stack has a dynamic size. Whereas, Array has a fixed size.

4.We can do only linear search in stack. Whereas, we can do both linear and Binary search in array.

5.Stack can contain elements of different data type. Whereas, Array contains elements of same data type.

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

queue?

Answer:

2 Queues are needed to implement the priority queue.One is used for storing data another is used for priorities.

Priority queues are 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.

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

Answer:

There are three different types of traversal techniques in a tree. They are:

1.Preorder Traversal:

1)Visit root node

2)Visit all the nodes in the left subtree

3)Visit all the nodes in the right subtree

2.Inorder Traversal:

1)First, visit all the nodes in the left subtree

2)Then the root node

3)Visit all the nodes in the right subtree

3.Postorder Traversal:

1)Visit all the nodes in the left subtree

2)Visit all the nodes in the right subtree

3)Visit the root node

Q9) Why it is said that searching a node in a binary search tree is efficient than that of

a simple binary tree?

Answer:

Binary tree can be anything that has two children and one parent. Common operations that can be performed on a binary tree are insertion, deletion, and traversal. Binary search trees are more of sorted binary trees that allows for fast and efficient lookup, insertion, and deletion of items. Unlike binary trees, binary search trees keep their keys sorted, so lookup usually implements binary search for operations. 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.

Q10) What are the applications of Graph DS?

Answer:

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.

Some of the applications of Graphs in Data Structure:

1.Google maps uses graphs for building transportation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.

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. There is an edge from a page u to other page v if there is a link of page v on page u. This is an example of Directed graph. It was the basic idea behind Google Page Ranking Algorithm.

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?

Answer:

No we cannot apply binary search algorithm to a sorted linked list.since, there is no way of indexing the middle element in the list. This is the drawback of the linked list in the data structures.

Q12) When can you tell that a Memory Leak will occur?

Answer:

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.

Q13) How will you check if a given Binary Tree is a Binary Search Tree or not?

Answer:

To check if a given Binary Tress is a Binary Search Tree:

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.

Q14) Which data structure is ideal to perform recursion operation and why?

Answer:

The data structure that is ideal to perform recursion operation is Stack.

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. Even when such equivalent iterative procedures are written, explicit stack is to be used.

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

Answer:

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.

Q16) **Convert the below given expression to its equivalent Prefix and Postfix notations.**

Expression is 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.

Q17) Program for sorting a stack using a temporary stack?

Answer:

import java.util.\*;  
  
class SortStack  
{  
 public static Stack<Integer> sortstack(Stack<Integer> input)  
 {  
 Stack<Integer> tmpStack = new Stack<>();  
 while(!input.isEmpty())  
 {  
 // pop out the first element   
 int tmp = input.pop();  
 while(!tmpStack.isEmpty() && tmpStack.peek() > tmp)  
 {  
 // pop from temporary stack and   
 // push it to the input stack   
 input.push(tmpStack.pop());  
 }  
  
 // push temp in temporary of stack   
 tmpStack.push(tmp);  
 }  
 return tmpStack;  
 }

// Driver Code   
 public static void main(String[] args)  
 {  
 Stack<Integer> input = new Stack<>();  
 input.add(40);  
 input.add(30);  
 input.add(20);  
 input.add(90);  
 input.add(80);  
 input.add(70);  
 // This is the temporary stack   
 Stack<Integer> tmpStack=sortstack(input);  
 System.out.println("Sorted numbers are:");  
 while (!tmpStack.empty())  
 {  
 System.out.print(tmpStack.pop()+" ");  
 }  
 }  
}

**Q18)Program to reverse a queue?**

**Answer:**

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

class Queue\_reverse {

static Queue<Integer> queue;

static void Print()

{

while (!queue.isEmpty()) {

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

queue.remove();

}

}

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<>();

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);

reversequeue();

Print();

}

}

**Q19) Program to reverse first k elements of a queue?**

**Answer:**

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

class Reverse\_k\_element\_queue {

static Queue<Integer> queue;

static void reverseQueueFirstKElements(int k)

{

if (queue.isEmpty()

|| k > queue.size())

return;

if (k <= 0)

return;

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

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

stack.push(queue.peek());

queue.remove();

}

while (!stack.empty()) {

queue.add(stack.peek());

stack.pop();

}

// Remove the remaining elements and enqueue

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();

}

}

// Driver code

public static void main(String[] args)

{

queue = new LinkedList<>();

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 =10;

reverseQueueFirstKElements(k);

Print();

}

}

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

**Answer:**

class LinkedList {

// head

Node head;

//Linked List node

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);

}

// Inserting a new Node at front

public void push(int new\_data)

{

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

new\_node.next = head;

head = new\_node;

}

//Driver code

public static void main(String[] args)

{

LinkedList llist = new LinkedList();

llist.push(1);

llist.push(2);

llist.push(3);

llist.push(4);

llist.printNthFromLast();

}

}

**Q21)Reverse a linked list?**

**Answer:**

class LinkedList {

static Node head;

static class Node {

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

//reverse the linked list

Node reverse(Node node)

{

Node prev = null;

Node current = node;

Node next;

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();

head = new Node(1);

head.next = new Node(2);

head.next.next = new Node(3);

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

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

list.printList(head);

head = list.reverse(head);

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

list.printList(head);

}

}

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

**Answer:**

class GFG {

static void changeArr(int[] input)

{

int[] newArray = Arrays.copyOfRange(input,0,input.length);

Arrays.sort(newArray);

Map<Integer, Integer> ranks = new HashMap<>();

int rank = 1;

for (int element : newArray)

{

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

ranks.put(element, rank);

rank++;

}

}

for (int index = 0; index < input.length; index++)

{

input[index] = ranks.get(input[index]);

}

}

// Driver Code

public static void main(String[] args)

{

int[] arr = { 100,90,80,70};

changeArr(arr);

// Printing

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

}

}

**Q23) Check if a given graph is a tree or not?**

**Answer:**

import java.io.\*;

import java.util.\*;

class Graph

{

//vertices

    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 ();

    }

    // add an edge

    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;

    }

    // Returns true if the graph is a tree, else 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;

    }

    // Driver code

    public static void main(String args[])

    {

        // Create a graph given in the above diagram

        Graph g1 = new Graph(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");

        Graph g2 = new Graph(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?**

**Answer:**

import java.util.Arrays;

import java.util.Collections;

class GFG {

    //returns kth smallest element in a given array

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

    {

        // Sorting

        Arrays.sort(arr);

        // Return kth element in the sorted array

        return arr[k - 1];

    }

    // driver program

    public static void main(String[] args)

    {

        Integer arr[] = new Integer[] {1,2,3,4,5,6,7,8,9,10};

        int k = 2;

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

    }

}

**Q25) How to find the shortest path between two vertices?**

1. Input the nodes of the graph

2. Get the input of the source and destination nodes.

3. Find the paths between the source and destination nodes.

4. Find the number of edges in all the paths and return the path having the minimum number of edges.

5. The shortest path between the two vertices is found.