***INDEX***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.No | Name of the Program | Page No. | Date | Signature | Remarks |
|  | Basic Implementation of stack using array in java. |  |  |  |  |
|  | Implementation of stack and its methods using array in java. |  |  |  |  |
|  | Program of stack using the built-in collection framework. |  |  |  |  |
|  | Implementation of queue using array in java. |  |  |  |  |
|  | Implementation of linked list in java. |  |  |  |  |
|  | Program to demonstrate Tower of Hanoi problem |  |  |  |  |
|  | Program to convert infix expression to postfix expression using Stack |  |  |  |  |
|  | Program to sort an array using Bubble Sort |  |  |  |  |
|  | Program to sort an array using Insertion Sort |  |  |  |  |
|  | Program to sort an array using Merge Sort |  |  |  |  |
|  | Program to sort an array using Quick Sort |  |  |  |  |
|  | Program to sort an array using Heap Sort |  |  |  |  |
|  | Java program to build a tree and traverse it in Preorder, Inorder & Postorder format. |  |  |  |  |
|  | Program to sort an array using Count Sort |  |  |  |  |
|  | Program to sort an array using Selection Sort |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Java program to find shortest path using Dijkstra's Algorithm (Single Source Shortest Path) |  |  |  |  |
|  | Java Program to find shortest path using Floyed Warshall’s Algorithm (All Pair Shortest Path) |  |  |  |  |
|  | Program to Build and Insert elements in a Red Black Tree |  |  |  |  |
|  | WAP in java to create a B Tree of order 5 for 1st 20 natural number. |  |  |  |  |
|  | Traversing a graph using the DFS (Depth First Search) Algorithm |  |  |  |  |
|  | Traversing a graph using the BFS (Breath First Search) Algorithm. |  |  |  |  |
|  | Java program to return the MST of a graph using Prim's Algorithm |  |  |  |  |
|  | Java program to return the MST of a graph using Kruskal’s algorithm |  |  |  |  |
|  | Java program to return Optimal parameterization using Matrix Chain Multiplication; <5,10,3,12,5,50,1> |  |  |  |  |
|  | Program to find LCS using Recursive Approach |  |  |  |  |
|  | Program to find LCS using Dynamic Programing Approach |  |  |  |  |
|  | Java program to solve the 0/1 Knapsack Problem using recursion. |  |  |  |  |
|  | Java program to solve the 0/1 Knapsack Problem using Dynamic Programing. |  |  |  |  |
|  | Java program to demonstrate Huffman Coding |  |  |  |  |
|  | Program to implement Activity Selection Algorithm |  |  |  |  |
|  | Program to implement string matching algorithm using the Naïve Method |  |  |  |  |
|  | Program to implement string matching algorithm using Rabin-Karp Method |  |  |  |  |
|  | Program to find LPS (Longest Prefix as Suffix) using KMP (Knuth Morris Pratt) Algorithm |  |  |  |  |
|  | Backtracking implementation for finding Hamiltonian Cycle |  |  |  |  |
|  | Java Program to return the Vertex Cover of a given undirected graph (Approximation algorithm) |  |  |  |  |
|  | Java Program to return the Set  Cover of a given undirected graph (Approximation algorithm) |  |  |  |  |
|  | Java program to implement the  Traveling Salesman Problem. |  |  |  |  |

Q1) Implementation and operations of Stack using Array.

public class DSstack{

public static void main(String[] args){

stackop s=new stackop();

s.push(10);

s.push(20);

s.push(30);

s.push(40);

s.push(50);

s.push(60);

s.getsize();

s.display();

s.pop();

s.getsize();

s.display();

s.pop();

s.pop();

s.pop();

s.pop();

s.pop();

s.getsize();

s.display();

}}

class stackop{

int top;

int max=5;

int[] stack=new int[max];

int currentsize=0;

stackop(){

top=-1;

}

//methods

//peek pop display push size

void peek(){

System.out.println(stack[top]+" Peeking the last element ");

}

public void push(int x){

if (top==(max-1)){

System.out.println("Stack is overflow ");

}else{

stack[++top]=x;

System.out.println(stack[top]+" Pushed the element ");

currentsize++;

}}

public void pop(){

if (top<0){

System.out.println("Stack is underflow");

}else{

System.out.println(stack[top]+" Popped the element ");

top--;

currentsize--;

}}

void display(){

if (top<0){

System.out.println("Stack is empty");

}else{

System.out.println("Elements in stack ");

for (int i=top;i>=0;i--){

System.out.print(stack[i]+" ");

}

System.out.println();

}}

void getsize(){

if (currentsize==0){

System.out.println(-1+" Current size of the stack ");

}else{System.out.println(this.currentsize+" Current size of the stack ");

}}}

Q2) Implementation and operations by user input of Stack using Array.

import java.util.Scanner;

public class Autostack{

public static void main(String[] args){

System.out.println("Enter the Size of the Stack ");

Autostackop as=new Autostackop();

int max=0;

Scanner s=new Scanner(System.in);

System.out.println("Automate Stack implement and operations");

while (true){

System.out.println("Select the operations \n 1)PUSH 2)POP 3)DISPLAY 4)PEEK 5)EXIT");

int choice=s.nextInt();

if (choice==1){

System.out.println("How many Values you want to enter ");

int val=s.nextInt();

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

as.push(s.nextInt());

}

}else if(choice==2){

System.out.println("How many Values you want to Pop ");

int val=s.nextInt();

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

as.pop();

}

}else if(choice==3){

as.display();

}else if(choice==4){

as.peek();

}

else{

System.out.println("Bye");

System.exit(0);

}}}}

class Autostackop{

int top;

Scanner ma=new Scanner(System.in);

int max=ma.nextInt();

int[] stack=new int[max];

int currentsize=0;

Autostackop(){

top=-1;

}

//methods

//peek pop display push size

void peek(){

if(top==-1){

System.out.println("There is no element in stack ");

}else{

System.out.println(stack[top]+" Peeking the last element ");

}}

public void push(int x){

if (top==(max-1)){

System.out.println("Stack is overflow ");

}else{

stack[++top]=x;

System.out.println(stack[top]+" Pushed the element ");

currentsize++;

}

}

public void pop(){

if (top<0){

System.out.println("Stack is underflow");

}else{

System.out.println(stack[top]+" Popped the element ");

top--;

currentsize--;

}

}

void display(){

if (top<0){

System.out.println("Stack is empty");

}else{

System.out.println("Elements in stack ");

for (int i=top;i>=0;i--){

System.out.print(stack[i]+" ");

}

System.out.println();

}

}

void getsize(){

if (currentsize==0){

System.out.println(-1+" Current size of the stack ");

}else{

System.out.println(this.currentsize+" Current size of the stack ");

}}}

Q3) Implementation and operations of Stack using Stack class.

import java.util.Stack;

public class StackExample{

public static void main(String[]args){

//creating an instance of Stack class

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

//checking stack is empty or not

boolean result = stk.empty();

System.out.println("IS the stack empty " + result);

stk.push(78);

result = stk.empty();

System.out.println("IS the stack empty " + result);

System.out.println("Element is stack" + stk);

stk.pop();

System.out.println("Element is stack" + stk);

stk.pop();

System.out.println("Element is stack" + stk);

stk.pop();

System.out.println("Element is stack" + stk);

stk.pop();

System.out.println("Element is stack" + stk);

stk.pop();

}}

Q4) Implementation and operations of Queue.

class queueoperation{

int front=0;

int rear=0;

int max=3;

int[] queue=new int[max];

void enqueue(int x)

{

if (rear==max){

System.out.println("Queue is overflow");

}else{

queue[rear]=x;

System.out.println(x+"----------- Insert in queue -----------");

rear++;

}}

void dequeue(){

if (front==rear){

System.out.println("Queue is underflow");

}else{

int v=queue[front];

System.out.println(v+"----------- delete in queue -----------");

front++;

}}

void peek(){

if (front==rear){

System.out.println("Queue is empty");

}else{

System.out.println(queue[front]+" Peeking the first element ");}}

void display(){

if (front==rear){

System.out.println("Queue is empty");

}else{

for(int i=front;i<rear;i++){

System.out.print(queue[i]+" ");

}System.out.println();

}}}

class queue{

public static void main(String args[]){

queueoperation q=new queueoperation();

q.enqueue(30);

q.peek();

q.display();

q.dequeue();

q.dequeue();

q.dequeue();

q.dequeue();

q.display();

}}

Q5) Implementation of linked list in java.

class LL {

Node head;

private int size;

LL() {

size=0;

}

public class Node {

String data;

Node next;

Node(String data) {

this.data=data;

this.next=null;

size++;

} }

public void addFirst(String data) {

Node newNode=new Node(data);

newNode.next=head;

head=newNode;

}

public void addLast(String data) {

Node newNode=new Node(data);

if(head==null) {

head=newNode;

return;

}

Node lastNode=head;

while(lastNode.next!=null) {

lastNode=lastNode.next; }

lastNode.next=newNode;

}

public void removeFirst(){

if(head==null) {

System.out.println("Empty List, nothing to delete.");

return;

}

head=this.head.next;

size=size-1;

}

public void removeLast() {

if(head==null) {

System.out.println("Empty List, nothing to delete.");

return;

}

if(head.next==null) {

head=null;

return;

}

Node currNode=head;

Node lastNode=head.next;

while(lastNode.next!=null) {

currNode=currNode.next;

lastNode=lastNode.next;

}

currNode.next=null;

}

public int getSize() {

return size;

}

public void reverseList() {

if(head==null || head.next==null) {

return;

}

Node prevNode=head;

Node currNode=head;

while(currNode!=null) {

Node nextNode=currNode.next;

currNode.next=prevNode;

prevNode=currNode;

currNode=nextNode;

}

head.next=null;

head=prevNode;

}

public void printList() {

Node currNode=head;

while(currNode!=null) {

System.out.print(currNode.data+" -> ");

currNode=currNode.next;

}

System.out.println("null");

}

public static void main(String args[]) {

LL list = new LL();

list.addLast("are");

list.addLast("a");

list.addLast("student");

list.printList();

list.addFirst("you");

list.printList();

System.out.println(list.getSize());

}

}

Q6) Program to demonstrate Tower of Hanoi problem.

class TowerOfHanoi {

static void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 0) {

return;

}

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

System.out.println("Move disk " + n + " from rod "+ from\_rod + " to rod "+ to\_rod);

towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

public static void main(String args[])

{

int N = 5;

towerOfHanoi(N, 'A', 'C', 'B');

}

}

Q7) Program to convert infix expression to postfix expression using Stack.

import java.util.Stack;

class InfixToPostfixClass {

static int Prec(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

static String infixToPostfix(String exp) {

String result = new String("");

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

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

char c = exp.charAt(i);

if (Character.isLetterOrDigit(c))

result += c;

else if (c == '(')

stack.push(c);

else if (c == ')') {

while (!stack.isEmpty() &&

stack.peek() != '(')

result += stack.pop();

stack.pop();

} else {

while (!stack.isEmpty() && Prec(c) <=

Prec(stack.peek())) {

result += stack.pop();}

stack.push(c);

}}

while (!stack.isEmpty()) {

if (stack.peek() == '(')

return "Invalid Expression";

result += stack.pop(); }

return result;}

public static void main(String[] args) {

String exp = "(p+q)\*(m-n)";

System.out.println("Infix expression: " + exp);

System.out.println("Prefix expression: " + infixToPostfix(exp));

}}

Q8) Program to sort an array using Bubble Sort.

class BubbleSort{

{

int i, j, temp;

boolean swapped;

for (i = 0; i < n - 1; i++) {

swapped = false;

for (j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = true;

}}

if (swapped == false)

break;

}}

static void printArray(int arr[], int size)

{

int i;

for (i = 0; i < size; i++)

System.out.print(arr[i] + " ");

System.out.println();

}

public static void main(String args[]){

int arr[] = { 64, 34, 25, 12, 22, 11, 90 };

int n = arr.length;

bubbleSort(arr, n);

System.out.println("Sorted array: ");

printArray(arr, n);

}

}

Q9) Program to sort an array using Insertion Sort.

import java.util.Arrays;

class InsertionSortP {

void insertionSort(int array[]) {

int size = array.length;

for (int step = 1; step < size; step++) {

int key = array[step];

int j = step - 1;

while (j >= 0 && key < array[j]) {

array[j + 1] = array[j];

--j;}

array[j + 1] = key;

}}

public static void main(String args[]) {

int[] data = { 9, 5, 1, 4, 3 };

InsertionSort is = new InsertionSort();

is.insertionSort(data);

System.out.println("Sorted Array in Ascending Order: ");

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

}

}

Q10) Write a program in java to sort an array using merge sort.

import java.io.\*;

class MergeSort {

void merge(int arr[], int l, int m, int r){

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (int i = 0; i < n1; ++i)

L[i] = arr[l + i];

for (int j = 0; j < n2; ++j)

R[j] = arr[m + 1 + j];

int i = 0, j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}}

void sort(int arr[], int l, int r){

if (l < r) {

// Find the middle point

int m = l + (r - l) / 2;

sort(arr, l, m);

sort(arr, m + 1, r);

merge(arr, l, m, r);

}}

static void printArray(int arr[]){

int n = arr.length;

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

System.out.print(arr[i] + " ");

System.out.println();

}

public static void main(String args[]){

int arr[] = { 12, 11, 13, 5, 6, 7 };

System.out.println("Given array is");

printArray(arr);

MergeSort ob = new MergeSort();

ob.sort(arr, 0, arr.length - 1);

System.out.println("\nSorted array is");

printArray(arr);

}

}

Q11) Program to sort an array using Quick Sort.

public class QuickSort {

public static void quickSort(int[] arr) {

if (arr == null || arr.length == 0) {

return;}

sort(arr, 0, arr.length - 1);}

private static void sort(int[] arr, int low, int high) {

if (low < high) {

int pivotIndex = partition(arr, low, high);

sort(arr, low, pivotIndex - 1);

sort(arr, pivotIndex + 1, high);

}}

private static int partition(int[] arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}}

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;}

public static void main(String[] args) {

int[] arr = {10, 7, 8, 9, 1, 5};

quickSort(arr);

System.out.println("Sorted array:");

for (int num : arr) {

System.out.print(num + " ");

}}}

Q12) Program to sort an array using Heap Sort.

public class HeapSort {

public static void heapSort(int[] arr) {

int n = arr.length;

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i > 0; i--) {

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);}

}

private static void heapify(int[] arr, int n, int i) {

int largest = i; // Initialize largest as root

int left = 2 \* i + 1; // left = 2\*i + 1

int right = 2 \* i + 2; // right = 2\*i + 2

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

int swap = arr[i];

arr[i] = arr[largest];

arr[largest] = swap;

heapify(arr, n, largest);

}}

public static void main(String[] args) {

int[] arr = {12, 11, 13, 5, 6, 7};

heapSort(arr);

System.out.println("Sorted array:");

for (int num : arr) {

System.out.print(num + " ");

}}}

Q13) Java program to build a tree and traverse it in Preorder, Inorder & Postorder format.

import java.util.Scanner;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

public TreeNode(int val) {

this.val = val;

left = null;

right = null;

}}

public class TreeTraversal {

TreeNode root;

public TreeTraversal() {

root = null;}

public void buildTree(Scanner scanner, TreeNode node) {

System.out.print("Enter the value of the node (or -1 to indicate a null node): ");

int val = scanner.nextInt();

if (val == -1) {

return;

}

node.val = val;

System.out.print("Enter the left child of " + val + ": ");

node.left = new TreeNode(0);

buildTree(scanner, node.left);

System.out.print("Enter the right child of " + val + ": ");

node.right = new TreeNode(0);

buildTree(scanner, node.right);

}

public void preorderTraversal(TreeNode node) {

if (node != null) {

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

preorderTraversal(node.left);

preorderTraversal(node.right);

}}

public void inorderTraversal(TreeNode node) {

if (node != null) {

inorderTraversal(node.left);

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

inorderTraversal(node.right);

}}

public void postorderTraversal(TreeNode node) {

if (node != null) {

postorderTraversal(node.left);

postorderTraversal(node.right);

System.out.print(node.val + " ");}}

public static void main(String[] args) {

TreeTraversal tree = new TreeTraversal();

Scanner scanner = new Scanner(System.in);

System.out.println("Building the binary tree:");

tree.root = new TreeNode(0);

tree.buildTree(scanner, tree.root);

System.out.println("\nPreorder traversal:");

tree.preorderTraversal(tree.root);

System.out.println("\nInorder traversal:");

tree.inorderTraversal(tree.root);

System.out.println("\nPostorder traversal:");

tree.postorderTraversal(tree.root);

scanner.close();}}

Q14) Program to sort an array using Count Sort.

public class CountingSort {

public static void countingSort(int[] arr) {

if (arr == null || arr.length == 0) {

return;

}

int max = findMax(arr);

int[] count = new int[max + 1];

for (int num : arr) {

count[num]++;

}

for (int i = 1; i < count.length; i++) {

count[i] += count[i - 1];

}

int[] output = new int[arr.length];

for (int i = arr.length - 1; i >= 0; i--) {

output[count[arr[i]] - 1] = arr[i];

count[arr[i]]--;

}

System.arraycopy(output, 0, arr, 0, arr.length);

}

private static int findMax(int[] arr) {

int max = Integer.MIN\_VALUE;

for (int num : arr) {

if (num > max) {

max = num;

}}

return max;}

public static void main(String[] args) {

int[] arr = {4, 2, 2, 8, 3, 3, 1};

countingSort(arr);

System.out.println("Sorted array:");

for (int num : arr) {

System.out.print(num + " ");

}}}

Q15) Program to sort an array using Selection Sort.

public class SelectionSort {

public static void selectionSort(int[] arr) {

if (arr == null || arr.length == 0) {

return;

}

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

// Find the minimum element in unsorted array

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}}

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;}}

public static void main(String[] args) {

int[] arr = {64, 25, 12, 22, 11};

selectionSort(arr);

System.out.println("Sorted array:");

for (int num : arr) {

System.out.print(num + " ");

}}}

Q16) Java program to find shortest path using Dijkstra's Algorithm (Single Source Shortest Path).

import java.util.\*;

public class DijkstraAlgorithm {

static class Graph {

private final int V;

private final List<List<Node>> adj;

public Graph(int V) {

this.V = V;

adj = new ArrayList<>(V);

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

adj.add(new ArrayList<>());

}}

public void addEdge(int source, int destination, int weight) {

adj.get(source).add(new Node(destination, weight));

}

public int[] dijkstra(int source) {

int[] distance = new int[V];

Arrays.fill(distance, Integer.MAX\_VALUE);

distance[source] = 0;

PriorityQueue<Node> pq = new PriorityQueue<>(V, Comparator.comparingInt(node -> node.weight));

pq.offer(new Node(source, 0));

while (!pq.isEmpty()) {

int u = pq.poll().vertex;

for (Node neighbor : adj.get(u)) {

int v = neighbor.vertex;

int w = neighbor.weight;

if (distance[u] != Integer.MAX\_VALUE && distance[u] + w < distance[v]) {

distance[v] = distance[u] + w;

pq.offer(new Node(v, distance[v]));

}}}

return distance;

}}

static class Node {

int vertex;

int weight;

public Node(int vertex, int weight) {

this.vertex = vertex;

this.weight = weight;

}}

public static void main(String[] args) {

int V = 5; // Number of vertices

int source = 0; // Source vertex

Graph graph = new Graph(V);

graph.addEdge(0, 1, 2);

graph.addEdge(0, 2, 4);

graph.addEdge(1, 2, 1);

graph.addEdge(1, 3, 7);

graph.addEdge(2, 4, 3);

graph.addEdge(3, 4, 1);

int[] distance = graph.dijkstra(source);

System.out.println("Shortest distances from the source vertex " + source + ":");

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

System.out.println("Vertex " + i + ": " + distance[i]); } } }

Q17) Java Program to find shortest path using Floyed Warshall’s Algorithm (All Pair Shortest Path).

public class AllPairShortestPath {

public static void floydWarshall(int graph[][], int V) {

int dist[][] = new int[V][V];

// Initialize distances

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

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

dist[i][j] = graph[i][j];

// Updating the shortest paths

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

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

for (int j = 0; j < V; j++) {

if (dist[i][k] != Integer.MAX\_VALUE && dist[k][j] != Integer.MAX\_VALUE &&

dist[i][k] + dist[k][j] < dist[i][j]) {

dist[i][j] = dist[i][k] + dist[k][j];

}}}}

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

for (int j = 0; j < V; j++) {

if (dist[i][j] == Integer.MAX\_VALUE)

System.out.print("INF ");

else

System.out.print(dist[i][j] + " ");

}

System.out.println();

}}

public static void main(String[] args) {

int graph[][] = {

{0, 5, Integer.MAX\_VALUE, 10},

{Integer.MAX\_VALUE, 0, 3, Integer.MAX\_VALUE},

{Integer.MAX\_VALUE, Integer.MAX\_VALUE, 0, 1},

{Integer.MAX\_VALUE, Integer.MAX\_VALUE, Integer.MAX\_VALUE, 0}

};

int V = graph.length;

floydWarshall(graph, V);

}}

Q18) Program to Build and Insert elements in a Red Black Tree.

import java.util.Scanner;

class Node {

int data;

Node parent;

Node left;

Node right;

boolean isRed; // Indicates whether the node is red or black

public Node(int data) {

this.data = data;

this.parent = null;

this.left = null;

this.right = null;

this.isRed = true; // By default, newly inserted nodes are red}}

public class RedBlackTree {

private Node root;

public RedBlackTree() {

root = null;}

private void insert(int data) {

Node newNode = new Node(data);

root = insertIntoTree(root, newNode);

fixViolation(newNode);}

private Node insertIntoTree(Node root, Node newNode) {

if (root == null)

return newNode;

if (newNode.data < root.data) {

root.left = insertIntoTree(root.left, newNode);

root.left.parent = root;

} else if (newNode.data > root.data) {

root.right = insertIntoTree(root.right, newNode);

root.right.parent = root;

}

return root;}

private void fixViolation(Node node) {

Node parent, grandparent;

while (node != root && node.isRed && node.parent.isRed) {

parent = node.parent;

grandparent = parent.parent;

if (parent == grandparent.left) {

Node uncle = grandparent.right;

if (uncle != null && uncle.isRed) {

grandparent.isRed = true;

parent.isRed = false;

uncle.isRed = false;

node = grandparent;

} else {

if (node == parent.right) {

rotateLeft(parent);

node = parent;

parent = node.parent;

}

rotateRight(grandparent);

parent.isRed = false;

grandparent.isRed = true;

node = parent;

}} else {

Node uncle = grandparent.left;

if (uncle != null && uncle.isRed) {

grandparent.isRed = true;

parent.isRed = false;

uncle.isRed = false;

node = grandparent;

} else {

if (node == parent.left) {

rotateRight(parent);

node = parent;

parent = node.parent;}

rotateLeft(grandparent);

parent.isRed = false;

grandparent.isRed = true;

node = parent;

}}}root.isRed = false;}

private void rotateLeft(Node node) {

Node rightChild = node.right;

node.right = rightChild.left;

if (node.right != null)

node.right.parent = node;

rightChild.parent = node.parent;

if (node.parent == null)

root = rightChild;

else if (node == node.parent.left)

node.parent.left = rightChild;

else{

node.parent.right = rightChild;

rightChild.left = node;

node.parent = rightChild;}

private void rotateRight(Node node) {

Node leftChild = node.left;

node.left = leftChild.right;

if (node.left != null)

node.left.parent = node;

leftChild.parent = node.parent;

if (node.parent == null)

root = leftChild;

else if (node == node.parent.left)

node.parent.left = leftChild;

else{

node.parent.right = leftChild;

leftChild.right = node;

node.parent = leftChild;}

private void inorderTraversal(Node root) {

if (root == null)

return;

inorderTraversal(root.left);

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

inorderTraversal(root.right);}

public static void main(String[] args) {

RedBlackTree rbTree = new RedBlackTree();

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements to insert into the Red-Black Tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements:");

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

int value = scanner.nextInt();

rbTree.insert(value);}

System.out.println("\nInorder Traversal of the Red-Black Tree:");

rbTree.inorderTraversal(rbTree.root);

scanner.close();

}}

Q19) WAP in java to create a B Tree of order 5 for 1st 20 natural number.

import java.util.LinkedList;

import java.util.Queue;

class BTreeNode {

int[] keys;

int t; // Minimum degree (defines the range for number of keys)

BTreeNode[] children; // An array of child pointers

int n; // Current number of keys in the node

boolean leaf; // Is true when node is leaf. Otherwise false.

public BTreeNode(int t, boolean leaf) {

this.t = t;

this.leaf = leaf;

this.keys = new int[2 \* t - 1];

this.children = new BTreeNode[2 \* t];

this.n = 0;

}

// A utility function that returns the index of the first key that is greater

// than or equal to k

int findKey(int k) {

int idx = 0;

while (idx < n && keys[idx] < k)

++idx;

return idx;

}

// A function to remove the key k from the subtree rooted with this node

void remove(int k) {

int idx = findKey(k);

// The key to be removed is present in this node

if (idx < n && keys[idx] == k) {

// If the node is a leaf node

if (leaf){

removeFromLeaf(idx);

}else{

removeFromNonLeaf(idx);

} else {

// If this node is a leaf node, then the key is not present in the tree

if (leaf) {

System.out.println("The key " + k + " does not exist in the tree");

return;

}

// The key to be removed is present in the subtree rooted with this node

boolean flag = (idx == n);

// If the child where the key is supposed to exist has less than t keys, we fill

// that child

if (children[idx].n < t)

fill(idx);

// If the last child has been merged, it must have merged with the previous child

// so we recurse on the (idx-1)th child. Else, we recurse on the (idx)th child

if (flag && idx > n)

children[idx - 1].remove(k);

else{

children[idx].remove(k);

}}

// A function to remove the idx-th key from this node - which is a leaf node

void removeFromLeaf(int idx) {

// Move all the keys after the idx-th pos one place backward

for (int i = idx + 1; i < n; ++i)

keys[i - 1] = keys[i];

// Reduce the count of keys

n--;}

// A function to remove the idx-th key from this node - which is a non-leaf node

void removeFromNonLeaf(int idx) {

int k = keys[idx];

// If the child that precedes k (C[idx]) has at least t keys,

// find the predecessor 'pred' of k in the subtree rooted at

// C[idx]. Replace k by pred. Recursively delete pred

// in C[idx]

if (children[idx].n >= t) {

int pred = getPred(idx);

keys[idx] = pred;

children[idx].remove(pred);

}

// If the child C[idx] has less than t keys, examine C[idx+1].

// If C[idx+1] has at least t keys, find the successor 'succ' of k in

// the subtree rooted at C[idx+1]

// Replace k by succ

// Recursively delete succ in C[idx+1]

else if (children[idx + 1].n >= t) {

int succ = getSucc(idx);

keys[idx] = succ;

children[idx + 1].remove(succ);

}

// If both C[idx] and C[idx+1] have less than t keys,merge k and all of C[idx+1]

// into C[idx]

// Now C[idx] contains 2t-1 keys

// Free C[idx+1] and recursively delete k from C[idx]

else {

merge(idx);

children[idx].remove(k);}}

// A function to get predecessor of keys[idx]

int getPred(int idx) {

// Keep moving to the right most node until we reach a leaf

BTreeNode cur = children[idx];

while (!cur.leaf)

cur = cur.children[cur.n];

// Return the last key of the leaf

return cur.keys[cur.n - 1];}

int getSucc(int idx) {

// Keep moving the left most node starting from C[idx+1] until we reach a leaf

BTreeNode cur = children[idx + 1];

while (!cur.leaf)

cur = cur.children[0];

// Return the first key of the leaf

return cur.keys[0];}

// A function to fill child C[idx] which has less than t-1 keys

void fill(int idx) {

// If the previous child(C[idx-1]) has more than t-1 keys, borrow a key

// from that child

if (idx != 0 && children[idx - 1].n >= t)

borrowFromPrev(idx);

// If the next child(C[idx+1]) has more than t-1 keys, borrow a key

// from that child

else if (idx != n && children[idx + 1].n >= t)

borrowFromNext(idx);

// Merge C[idx] with its sibling

// If C[idx] is the last child, merge it with with its previous sibling

// Otherwise merge it with its next sibling

else {

if (idx != n)

merge(idx);

else{

merge(idx - 1);

}}

// A function to borrow a key from C[idx-1] and insert it into C[idx]

void borrowFromPrev(int idx) {

BTreeNode child = children[idx];

BTreeNode sibling = children[idx - 1];

// The last key from C[idx-1] goes up to the parent and key[idx-1]

// from parent is inserted as the first key in C[idx]. Thus, sibling

// loses one key and child gains one key

// Moving all key in C[idx] one step forward

for (int i = child.n - 1; i >= 0; --i)

child.keys[i + 1] = child.keys[i];

// If C[idx] is not a leaf, move all its child pointers one step forward

if (!child.leaf) {

for (int i = child.n; i >= 0; --i)

child.children[i + 1] = child.children[i];}

// Setting child's first key equal to keys[idx-1] from the current node

child.keys[0] = keys[idx - 1];

// Moving sibling's last child as C[idx]'s first child

if (!leaf)

child.children[0] = sibling.children[sibling.n];

// Moving the key from the sibling to the parent

// This reduces the number of keys in the sibling

keys[idx - 1] = sibling.keys[sibling.n - 1];

child.n += 1;

sibling.n -= 1;}

// A function to borrow a key from the C[idx+1] and place it in C[idx]

void borrowFromNext(int idx) {

BTreeNode child = children[idx];

BTreeNode sibling = children[idx + 1];

// keys[idx] is inserted as the last key in C[idx]

child.keys[child.n] = keys[idx];

// Sibling's first child is inserted as the last child into C[idx]

if (!child.leaf)

child.children[child.n + 1] = sibling.children[0];

// The first key from sibling is inserted into keys[idx]

keys[idx] = sibling.keys[0];

// Moving all keys in sibling one step behind

for (int i = 1; i < sibling.n; ++i)

sibling.keys[i - 1] = sibling.keys[i];

// Moving the child pointers one step behind

if (!sibling.leaf) {

for (int i = 1; i <= sibling.n; ++i)

sibling.children[i - 1] = sibling.children[i];}

// Increasing and decreasing the key count of C[idx] and C[idx+1] respectively

child.n += 1;

sibling.n -= 1;}

// A function to merge C[idx] with C[idx+1]

// C[idx+1] is freed after merging

void merge(int idx) {

BTreeNode child = children[idx];

BTreeNode sibling = children[idx + 1];

// Pulling a key from the current node and inserting it into (t-1)th

// position of C[idx]

child.keys[t - 1] = keys[idx];

// Copying the keys from C[idx+1] to C[idx] at the end

for (int i = 0; i < sibling.n; ++i)

child.keys[i + t] = sibling.keys[i];

// Copying the child pointers from C[idx+1] to C[idx]

if (!child.leaf) {

for (int i = 0; i <= sibling.n; ++i)

child.children[i + t] = sibling.children[i];}

// Moving all keys after idx in the current node one step before -

// to fill the gap created by moving keys[idx] to C[idx]

for (int i = idx + 1; i < n; ++i)

keys[i - 1] = keys[i];

// Moving the child pointers after (idx+1) in the current node one

// step before

for (int i = idx + 2; i <= n; ++i)

children[i - 1] = children[i];

// Updating the key count of child and the current node

child.n += sibling.n + 1;

n--;

// Freeing the memory occupied by sibling

sibling = null;}

// Function to traverse all nodes in a subtree rooted with this node

void traverse() {

// There are n keys and n+1 children, traverse through n keys

// and first n children

int i;

for (i = 0; i < n; i++) {

// If this is not a leaf, then before printing key[i],

// traverse the subtree rooted with child C[i].

if (!leaf)

children[i].traverse();

System.out.print(keys[i] + " ");}

// Print the subtree rooted with last child

if (!leaf)

children[i].traverse();}

void levelOrderTraversal() {

if (this == null)

return;

Queue<BTreeNode> queue = new LinkedList<>();

queue.add(this);

while (!queue.isEmpty()) {

int nodeCount = queue.size();

while (nodeCount > 0) {

BTreeNode node = queue.poll();

node.traverse();

if (!node.leaf) {

for (int i = 0; i <= node.n; i++) {

queue.add(node.children[i]);}}

nodeCount--;}

System.out.println();

}}}

public class BTree {

private BTreeNode root;

private int t; // Minimum degree

public BTree(int t) {

this.root = null;

this.t = t;}

public void insert(int k) {

if (root == null) {

root = new BTreeNode(t, true);

root.keys[0] = k;

root.n = 1; // Update number of keys in root

} else {

if (root.n == 2 \* t - 1) {

BTreeNode s = new BTreeNode(t, false);

s.children[0] = root;

s.splitChild(0, root);

int i = 0;

if (s.keys[0] < k)

i++;

s.children[i].insertNonFull(k);

root = s;

} else

root.insertNonFull(k);}}

public void traverse() {

if (root != null)

root.traverse();}

public void levelOrderTraversal() {

if (root != null)

root.levelOrderTraversal();}

public static void main(String[] args) {

BTree tree = new BTree(5); // B-Tree of order 5

for (int i = 1; i <= 20; i++)

tree.insert(i);

System.out.println("Traversal of the B-Tree:");

tree.traverse();

System.out.println("\n\nLevel order traversal of the B-Tree:");

tree.levelOrderTraversal();

}}

Q20) Traversing a graph using the DFS (Depth First Search) Algorithm.

import java.util.\*;

public class DepthFirstSearch {

static class Graph {

private int V; // No. of vertices

private LinkedList<Integer> adj[]; // Adjacency Lists

// Constructor

Graph(int v) {

V = v;

adj = new LinkedList[v];

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

adj[i] = new LinkedList();}

// Function to add an edge into the graph

void addEdge(int v, int w) {

adj[v].add(w);}

// A function used by DFS

void DFSUtil(int v, boolean visited[]) {

visited[v] = true;

System.out.print(v + " ");

Iterator<Integer> i = adj[v].listIterator();

while (i.hasNext()) {

int n = i.next();

if (!visited[n])

DFSUtil(n, visited);}}

void DFS() {

boolean visited[] = new boolean[V]; // Mark all the vertices as not visited(set as false by default)

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

if (!visited[i])

DFSUtil(i, visited);

}}

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of vertices: ");

int V = scanner.nextInt();

Graph g = new Graph(V);

System.out.println("Enter the edges (vertex pairs, space-separated):");

while (true) {

int v1 = scanner.nextInt();

int v2 = scanner.nextInt();

if (v1 == -1 || v2 == -1)

break;

g.addEdge(v1, v2);}

System.out.println("Depth First Traversal (starting from vertex 0): ");

g.DFS();

scanner.close();

}}

Q21) Traversing a graph using the BFS (Breath First Search) Algorithm.

import java.util.\*;

public class BreadthFirstSearch {

static class Graph {

private int V; // No. of vertices

private LinkedList<Integer>[] adj; // Adjacency Lists

// Constructor

Graph(int v) {

V = v;

adj = new LinkedList[v];

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

adj[i] = new LinkedList<>();

}

// Function to add an edge into the graph

void addEdge(int v, int w) {

adj[v].add(w);}

// Prints BFS traversal from a given source s

void BFS(int s) {

// Mark all the vertices as not visited (by default set as false)

boolean[] visited = new boolean[V];

// Create a queue for BFS

Queue<Integer> queue = new LinkedList<>();

// Mark the current node as visited and enqueue it

visited[s] = true;

queue.add(s);

while (!queue.isEmpty()) {

// Dequeue a vertex from queue and print it

s = queue.poll();

System.out.print(s + " ");

// Get all adjacent vertices of the dequeued vertex s

// If an adjacent vertex has not been visited, then mark it

// visited and enqueue it

for (int i = 0; i < adj[s].size(); i++) {

int n = adj[s].get(i);

if (!visited[n]) {

visited[n] = true;

queue.add(n);

}}}}}

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of vertices: ");

int V = scanner.nextInt();

Graph g = new Graph(V);

System.out.println("Enter the edges (vertex pairs, space-separated):");

while (true) {

int v1 = scanner.nextInt();

int v2 = scanner.nextInt();

if (v1 == -1 || v2 == -1)

break;

g.addEdge(v1, v2);}

System.out.print("Enter the starting vertex for BFS: ");

int startVertex = scanner.nextInt();

System.out.println("Breadth First Traversal starting from vertex " + startVertex + ": ");

g.BFS(startVertex);

scanner.close();

} }

Q22) Java program to return the MST of a graph using Prim's Algorithm.

import java.util.\*;

public class PrimMST {

static class Edge {

int src, dest, weight;

Edge(int src, int dest, int weight) {

this.src = src;

this.dest = dest;

this.weight = weight;}}

static class Graph {

int V;

LinkedList<Edge>[] adjList;

Graph(int V) {

this.V = V;

adjList = new LinkedList[V];

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

adjList[i] = new LinkedList<>();}

void addEdge(int src, int dest, int weight) {

Edge edge = new Edge(src, dest, weight);

adjList[src].add(edge);

edge = new Edge(dest, src, weight);

adjList[dest].add(edge);}

void primMST() {

boolean[] inMST = new boolean[V];

int[] parent = new int[V];

int[] key = new int[V];

Arrays.fill(key, Integer.MAX\_VALUE);

Arrays.fill(parent, -1);

PriorityQueue<Edge> pq = new PriorityQueue<>(V, Comparator.comparingInt(e -> e.weight));

key[0] = 0;

pq.add(new Edge(-1, 0, 0));

while (!pq.isEmpty()) {

int u = pq.poll().dest;

inMST[u] = true;

for (Edge edge : adjList[u]) {

int v = edge.dest;

int weight = edge.weight;

if (!inMST[v] && weight < key[v]) {

key[v] = weight;

parent[v] = u;

pq.add(new Edge(u, v, key[v]));

}}}

printMST(parent);}

void printMST(int[] parent) {

System.out.println("Edge \tWeight");

for (int i = 1; i < V; i++)

System.out.println(parent[i] + " - " + i + "\t" + adjList[i].get(parent[i]).weight);}}

public static void main(String[] args) {

int V = 5; // Number of vertices

Graph graph = new Graph(V);

// Adding edges

graph.addEdge(0, 1, 2);

graph.addEdge(0, 3, 6);

graph.addEdge(1, 2, 3);

graph.addEdge(1, 3, 8);

graph.addEdge(1, 4, 5);

graph.addEdge(2, 4, 7);

graph.addEdge(3, 4, 9);

graph.primMST();

}}

Q23) Java program to return the MST of a graph using Kruskal’s algorithm.

import java.util.\*;

public class KruskalMST {

static class Edge implements Comparable<Edge> {

int src, dest, weight;

Edge(int src, int dest, int weight) {

this.src = src;

this.dest = dest;

this.weight = weight;}

@Override

public int compareTo(Edge compareEdge) {

return this.weight - compareEdge.weight;}}

static class Graph {

int V, E;

Edge[] edges;

Graph(int V, int E) {

this.V = V;

this.E = E;

edges = new Edge[E];

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

edges[i] = new Edge(0, 0, 0);}

int find(int[] parent, int i) {

if (parent[i] == i)

return i;

return find(parent, parent[i]);}

void union(int[] parent, int[] rank, int x, int y) {

int xroot = find(parent, x);

int yroot = find(parent, y);

if (rank[xroot] < rank[yroot])

parent[xroot] = yroot;

else if (rank[xroot] > rank[yroot])

parent[yroot] = xroot;

else {

parent[yroot] = xroot;

rank[xroot]++;}}

void kruskalMST() {

Edge[] result = new Edge[V];

int e = 0;

int i = 0;

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

result[i] = new Edge(0, 0, 0);

Arrays.sort(edges);

int[] parent = new int[V];

int[] rank = new int[V];

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

parent[i] = i;

rank[i] = 0;}

i = 0;

while (e < V - 1) {

Edge nextEdge = edges[i++];

int x = find(parent, nextEdge.src);

int y = find(parent, nextEdge.dest);

if (x != y) {

result[e++] = nextEdge;

union(parent, rank, x, y);}}

System.out.println("Minimum Spanning Tree using Kruskal's Algorithm:");

for (i = 0; i < e; ++i)

System.out.println(result[i].src + " - " + result[i].dest + ": " + result[i].weight);}}

public static void main(String[] args) {

int V = 4; // Number of vertices

int E = 5; // Number of edges

Graph graph = new Graph(V, E);

// Adding edges

graph.edges[0] = new Edge(0, 1, 10);

graph.edges[1] = new Edge(0, 2, 6);

graph.edges[2] = new Edge(0, 3, 5);

graph.edges[3] = new Edge(1, 3, 15);

graph.edges[4] = new Edge(2, 3, 4);

graph.kruskalMST();

}}

Q24) Java program to return Optimal parameterization using Matrix Chain Multiplication; <5,10,3,12,5,50,1>.

public class MatrixChainMultiplication {

static int MatrixChainOrder(int p[], int n) {

int m[][] = new int[n][n];

int i, j, k, L, q;

for (i = 1; i < n; i++)

m[i][i] = 0;

for (L = 2; L < n; L++) {

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

j = i + L - 1;

if (j == n)

continue;

m[i][j] = Integer.MAX\_VALUE;

for (k = i; k <= j - 1; k++) {

q = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];

if (q < m[i][j])

m[i][j] = q;}}}

return m[1][n - 1];}

public static void main(String args[]) {

int arr[] = { 5,10,3,12,5,50,1};

int size = arr.length;

System.out.println("Minimum number of multiplications is " + MatrixChainOrder(arr, size));}}

Q25) Program to find LCS using Recursive Approach.

public class LongestCommonSubsequenceRecursive {

static int lcsRecursive(char[] X, char[] Y, int m, int n) {

if (m == 0 || n == 0)

return 0;

if (X[m - 1] == Y[n - 1]){

return 1 + lcsRecursive(X, Y, m - 1, n - 1);

}else{

return Math.max(lcsRecursive(X, Y, m, n - 1), lcsRecursive(X, Y, m - 1, n));}

public static void main(String[] args) {

String X = "AGGTAB";

String Y = "GXTXAYB";

char[] xArray = X.toCharArray();

char[] yArray = Y.toCharArray();

int m = xArray.length;

int n = yArray.length;

System.out.println("Length of LCS is " + lcsRecursive(xArray, yArray, m, n));

}}

Q26) Program to find LCS using Dynamic Programing Approach.

public class LongestCommonSubsequenceDP {

static int lcsDP(char[] X, char[] Y, int m, int n) {

int[][] dp = new int[m + 1][n + 1];

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

for (int j = 0; j <= n; j++) {

if (i == 0 || j == 0){

dp[i][j] = 0;

}else if (X[i - 1] == Y[j - 1]){

dp[i][j] = dp[i - 1][j - 1] + 1;

}else{

dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);

}}

return dp[m][n];}

public static void main(String[] args) {

String X = "AGGTAB";

String Y = "GXTXAYB";

char[] xArray = X.toCharArray();

char[] yArray = Y.toCharArray();

int m = xArray.length;

int n = yArray.length;

System.out.println("Length of LCS is " + lcsDP(xArray, yArray, m, n));}}

Q27) Java program to solve the 0/1 Knapsack Problem using recursion.

public class KnapsackGreedy {

static class Item {

int weight, value;

Item(int weight, int value) {

this.weight = weight;

this.value = value;}}

static double fractionalKnapsack(int W, Item[] items) {

// Calculate value-to-weight ratios for each item

double[] ratios = new double[items.length];

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

ratios[i] = (double) items[i].value / items[i].weight;}

// Sort items based on their value-to-weight ratios (in non-increasing order)

for (int i = 0; i < items.length - 1; i++) {

for (int j = 0; j < items.length - i - 1; j++) {

if (ratios[j] < ratios[j + 1]) {

// Swap items and ratios

double tempRatio = ratios[j];

ratios[j] = ratios[j + 1];

ratios[j + 1] = tempRatio;

Item tempItem = items[j];

items[j] = items[j + 1];

items[j + 1] = tempItem;

}}}

double totalValue = 0;

int remainingWeight = W;

// Iterate through sorted items and add them to the knapsack

for (Item item : items) {

if (item.weight <= remainingWeight) {

totalValue += item.value;

remainingWeight -= item.weight;

} else {

double fraction = (double) remainingWeight / item.weight;

totalValue += fraction \* item.value;

break;}}

return totalValue;}

public static void main(String[] args) {

int W = 50; // Knapsack capacity

Item[] items = {

new Item(10, 60),

new Item(20, 100),

new Item(30, 120)};

double maxValue = fractionalKnapsack(W, items);

System.out.println("Maximum value that can be obtained: " + maxValue);

}}

Q28) Java program to solve the 0/1 Knapsack Problem using Dynamic Programing.

import java.util.Arrays;

class Item {

int weight;

int value;

double ratio;

public Item(int weight, int value) {

this.weight = weight;

this.value = value;

this.ratio = (double) value / weight;}}

public class FractionalKnapsackDP {

public static double fractionalKnapsack(int[] weights, int[] values, int capacity) {

int n = weights.length;

Item[] items = new Item[n];

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

items[i] = new Item(weights[i], values[i]);}

Arrays.sort(items, (Item item1, Item item2) -> {

if (item1.ratio < item2.ratio){

return 1;

}else if (item1.ratio > item2.ratio){

return -1;

else{

return 0;

});

double[] dp = new double[capacity + 1];

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

for (int j = 0; j < n; j++) {

if (items[j].weight <= i) {

dp[i] = Math.max(dp[i], dp[i - items[j].weight] + items[j].value);

}}}

return dp[capacity];}

public static void main(String[] args) {

int[] weights = { 10, 20, 30 };

int[] values = { 60, 100, 120 };

int capacity = 50;

double maxValue = fractionalKnapsack(weights, values, capacity);

System.out.println("Maximum value that can be obtained: " + maxValue);

}}

Q29) Java program to demonstrate Huffman Coding.

import java.util.HashMap;

import java.util.Map;

class HuffmanNode {

char ch;

int frequency;

HuffmanNode left, right;

public HuffmanNode(char ch, int frequency) {

this.ch = ch;

this.frequency = frequency;

}}

public class HuffmanCoding {

private HuffmanNode buildHuffmanTree(String text) {

Map<Character, Integer> frequencyMap = new HashMap<>();

for (char c : text.toCharArray()) {

frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);}

while (frequencyMap.size() > 1) {

char minChar1 = 0, minChar2 = 0;

int minFreq1 = Integer.MAX\_VALUE, minFreq2 = Integer.MAX\_VALUE;

for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {

if (entry.getValue() < minFreq1) {

minFreq2 = minFreq1;

minChar2 = minChar1;

minFreq1 = entry.getValue();

minChar1 = entry.getKey();

} else if (entry.getValue() < minFreq2) {

minFreq2 = entry.getValue();

minChar2 = entry.getKey();}}

frequencyMap.remove(minChar1);

frequencyMap.remove(minChar2);

HuffmanNode combinedNode = new HuffmanNode('\0', minFreq1 + minFreq2);

combinedNode.left = new HuffmanNode(minChar1, minFreq1);

combinedNode.right = new HuffmanNode(minChar2, minFreq2);

frequencyMap.put(combinedNode.ch, combinedNode.frequency);

}

Map.Entry<Character, Integer> entry = frequencyMap.entrySet().iterator().next();

return new HuffmanNode(entry.getKey(), entry.getValue());

}

public String encode(String text) {

if (text == null || text.isEmpty())

return "";

StringBuilder encodedText = new StringBuilder();

HuffmanNode root = buildHuffmanTree(text);

Map<Character, String> codes = new HashMap<>();

generateCodes(root, "", codes);

for (char c : text.toCharArray()) {

encodedText.append(codes.get(c));

}

return encodedText.toString();}

private void generateCodes(HuffmanNode node, String code, Map<Character, String> codes) {

if (node == null)

return;

if (node.left == null && node.right == null) {

codes.put(node.ch, code);}

generateCodes(node.left, code + "0", codes);

generateCodes(node.right, code + "1", codes);}

public String decode(String encodedText) {

if (encodedText == null || encodedText.isEmpty())

return "";

StringBuilder decodedText = new StringBuilder();

HuffmanNode root = buildHuffmanTree(encodedText);

HuffmanNode current = root;

for (char bit : encodedText.toCharArray()) {

if (bit == '0') {

current = current.left;

} else if (bit == '1') {

current = current.right;}

if (current.left == null && current.right == null) {

decodedText.append(current.ch);

current = root;}}

return decodedText.toString();

}

public static void main(String[] args) {

String text = "Huffman coding is a method used for compressing data";

HuffmanCoding huffman = new HuffmanCoding();

String encodedText = huffman.encode(text);

System.out.println("Encoded text: " + encodedText);

String decodedText = huffman.decode(encodedText);

System.out.println("Decoded text: " + decodedText);

} }

Q30) Program to implement Activity Selection Algorithm.

import java.io.\*;

import java.lang.\*;

import java.util.\*;

class ActivitySelection {

// Prints a maximum set of activities that can be done

// by a single person, one at a time.

public static void printMaxActivities(int s[], int f[],

int n){

int i, j;

System.out.println(

"Following activities are selected");

// The first activity always gets selected

i = 0;

System.out.print(i + " ");

// Consider rest of the activities

for (j = 1; j < n; j++) {

// If this activity has start time greater than

// or equal to the finish time of previously

// selected activity, then select it

if (s[j] >= f[i]) {

System.out.print(j + " ");

i = j;}}}

// Driver code

public static void main(String[] args)

{

int s[] = { 1, 3, 0, 5, 8, 5 };

int f[] = { 2, 4, 6, 7, 9, 9 };

int n = s.length;

// Function call

printMaxActivities(s, f, n);

}

}

Q31) Program to implement string matching algorithm using the Naïve Method.

import java.util.ArrayList;

import java.util.List;

public class NaiveStringMatching {

public static List<Integer> naiveStringMatching(String text, String pattern) {

List<Integer> matches = new ArrayList<>();

int n = text.length();

int m = pattern.length();

for (int i = 0; i <= n - m; i++) { // iterate over possible starting positions

int j;

for (j = 0; j < m; j++) { // compare characters

if (text.charAt(i + j) != pattern.charAt(j))

break;}

if (j == m) { // if all characters match

matches.add(i);

}}

return matches;}

public static void main(String[] args) {

String text = "AABAACAADAABAAABAA";

String pattern = "AABA";

List<Integer> positions = naiveStringMatching(text, pattern);

System.out.println("Pattern found at positions: " + positions);

}}

Q32) Program to implement string matching algorithm using Rabin-Karp Method.

import java.util.ArrayList;

import java.util.List;

public class RabinKarpStringMatching {

private static final int PRIME = 101; // A prime number to calculate hash

public static List<Integer> rabinKarpStringMatching(String text, String pattern) {

List<Integer> matches = new ArrayList<>();

int n = text.length();

int m = pattern.length();

int patternHash = calculateHash(pattern, m); // Calculate hash for the pattern

int textHash = calculateHash(text, m); // Calculate hash for the initial window in the text

for (int i = 0; i <= n - m; i++) {

if (patternHash == textHash && checkEqual(text, i, i + m - 1, pattern, 0, m - 1)) {

// If hash matches and characters also match, add index to matches

matches.add(i);

}

if (i < n - m) {

// Update rolling hash for the next window

textHash = recalculateHash(text, i, i + m, textHash, m);

}

}

return matches;

}

private static int calculateHash(String str, int len) {

int hash = 0;

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

hash += str.charAt(i) \* Math.pow(PRIME, i);

}

return hash;

}

private static int recalculateHash(String str, int oldIndex, int newIndex, int oldHash, int patternLen) {

int newHash = oldHash - str.charAt(oldIndex);

newHash = newHash / PRIME;

newHash += str.charAt(newIndex) \* Math.pow(PRIME, patternLen - 1);

return newHash;

}

private static boolean checkEqual(String str1, int start1, int end1, String str2, int start2, int end2) {

if (end1 - start1 != end2 - start2)

return false;

while (start1 <= end1 && start2 <= end2) {

if (str1.charAt(start1) != str2.charAt(start2))

return false;

start1++;

start2++;

}

return true;

}

public static void main(String[] args) {

String text = "AABAACAADAABAAABAA";

String pattern = "AABA";

List<Integer> positions = rabinKarpStringMatching(text, pattern);

System.out.println("Pattern found at positions: " + positions);

}

}

Q33) Program to find LPS (Longest Prefix as Suffix) using KMP (Knuth Morris Pratt) Algorithm.

import java.util.Arrays;

public class KMPLPS {

public static int[] computeLPS(String pattern) {

int m = pattern.length();

int[] lps = new int[m];

int len = 0;

int i = 1;

lps[0] = 0;

while (i < m) {

if (pattern.charAt(i) == pattern.charAt(len)) {

len++;

lps[i] = len;

i++;

} else {

if (len != 0) {

len = lps[len - 1];

} else {

lps[i] = 0;

i++;}}}

return lps;}

public static void main(String[] args) {

String pattern = "AAACAAAA";

int[] lps = computeLPS(pattern);

System.out.println("LPS array: " + Arrays.toString(lps));

}

}

Q34) Backtracking implementation for finding Hamiltonian Cycle.

import java.util.ArrayList;

public class HamiltonianCycle {

private int V;

private int[][] graph;

public HamiltonianCycle(int[][] graph) {

this.graph = graph;

this.V = graph.length;

}

private boolean isSafe(int v, int pos, int[] path) {

if (graph[path[pos - 1]][v] == 0) {

return false;

}

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

if (path[i] == v) {

return false;}}

return true;}

private boolean hamiltonianCycleUtil(int[] path, int pos) {

if (pos == V) {

if (graph[path[pos - 1]][path[0]] == 1) {

return true;

} else {

return false;}}

for (int v = 1; v < V; v++) {

if (isSafe(v, pos, path)) {

path[pos] = v;

if (hamiltonianCycleUtil(path, pos + 1)) {

return true;}

path[pos] = -1;}}

return false;}

public boolean findHamiltonianCycle() {

int[] path = new int[V];

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

path[i] = -1;}

path[0] = 0; // Start from vertex 0 as the starting point

if (!hamiltonianCycleUtil(path, 1)) {

System.out.println("No Hamiltonian Cycle exists");

return false;}

System.out.println("Hamiltonian Cycle exists: ");

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

System.out.print(path[i] + " ");}

System.out.print(path[0]); // Complete the cycle

return true;}

public static void main(String[] args) {

int[][] graph = {

{0, 1, 0, 1, 0},

{1, 0, 1, 1, 1},

{0, 1, 0, 0, 1},

{1, 1, 0, 0, 1},

{0, 1, 1, 1, 0}

};

HamiltonianCycle hc = new HamiltonianCycle(graph);

hc.findHamiltonianCycle();

}}

Q35) Java Program to return the Vertex Cover of a given undirected graph (Approximation algorithm).

import java.util.\*;

class Graph {

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);}

void vertexCover() {

boolean visited[] = new boolean[V];

Arrays.fill(visited, false);

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

if (!visited[u]) {

for (Integer v : adj[u]) {

if (!visited[v]) {

visited[u] = true;

visited[v] = true;

break;}}}}

System.out.println("Vertex cover using approximation algorithm: ");

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

if (visited[i])

System.out.print(i + " ");

System.out.println();}}

public class VertexCoverApproximation {

public static void main(String args[]) {

int V = 7;

Graph graph = new Graph(V);

graph.addEdge(0, 1);

graph.addEdge(0, 2);

graph.addEdge(1, 3);

graph.addEdge(3, 4);

graph.addEdge(4, 5);

graph.addEdge(5, 6);

graph.vertexCover();}}

Q36) Java Program to return the Set Cover of a given undirected graph (Approximation algorithm).

import java.util.\*;

public class SetCoverApproximation {

// Function to find the set cover using greedy approximation algorithm

public static Set<Integer> setCover(int[][] graph) {

int V = graph.length; // Number of vertices

Set<Integer> uncoveredVertices = new HashSet<>();

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

uncoveredVertices.add(i);}

Set<Integer> setCover = new HashSet<>();

while (!uncoveredVertices.isEmpty()) {

int maxIntersection = 0;

int bestSet = -1;

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

if (!setCover.contains(i)) {

Set<Integer> intersection = new HashSet<>(uncoveredVertices);

intersection.retainAll(Arrays.stream(graph[i]).boxed().toList());

if (intersection.size() > maxIntersection) {

maxIntersection = intersection.size();

bestSet = i;}

}}

if (bestSet == -1) {

// Graph cannot be covered

break;}

setCover.add(bestSet);

uncoveredVertices.removeAll(Arrays.stream(graph[bestSet]).boxed().toList());}

return setCover;}

public static void main(String[] args) {

int[][] graph = {

{1, 2}, // Vertices covered by set 0

{0, 2, 3}, // Vertices covered by set 1

{0, 1}, // Vertices covered by set 2

{1} // Vertices covered by set 3};

Set<Integer> setCover = setCover(graph);

System.out.println("Set Cover: " + setCover);

}}

Q37) Java program to implement the Traveling Salesman Problem.

import java.util.\*;

public class TravelingSalesmanProblem {

static int[][] distanceMatrix = {

{0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}};

static int numCities = distanceMatrix.length;

public static List<Integer> nearestNeighbor() {

List<Integer> tour = new ArrayList<>();

boolean[] visited = new boolean[numCities];

// Start at city 0

tour.add(0);

visited[0] = true;

for (int i = 0; i < numCities - 1; i++) {

int currentCity = tour.get(i);

int nearestNeighbor = findNearestNeighbor(currentCity, visited);

tour.add(nearestNeighbor);

visited[nearestNeighbor] = true;}

// Return to the starting city

tour.add(0);

return tour;}

public static int findNearestNeighbor(int city, boolean[] visited) {

int minDistance = Integer.MAX\_VALUE;

int nearestNeighbor = -1;

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

if (!visited[i] && i != city && distanceMatrix[city][i] < minDistance) {

minDistance = distanceMatrix[city][i];

nearestNeighbor = i;

}}

return nearestNeighbor;}

public static void main(String[] args) {

List<Integer> tour = nearestNeighbor();

System.out.println("Nearest Neighbor Tour: " + tour);

System.out.println("Total Distance: " + calculateTourDistance(tour));}

public static int calculateTourDistance(List<Integer> tour) {

int totalDistance = 0;

for (int i = 0; i < tour.size() - 1; i++) {

int city1 = tour.get(i);

int city2 = tour.get(i + 1);

totalDistance += distanceMatrix[city1][city2];}

return totalDistance;

}}