

**DESIGN AND ANALYSIS OF ALGORITHMS LAB**

**(BCSC-0807)**

**Made by** :- VISHAL DIXIT

**Section** :- B (62)

**University Roll No.** :- 201500792

**Submitted to** :- Miss. Varsha Thakur Mam

**Sortings**

1. **Selection Sort** ➖

package com.programs.DAA\_lab;

import java.util.Scanner;

public class Selection\_sort {

public static void selection\_sort(int arr[]){

int n=arr.length;

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

int min=i;

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

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

min=j;

}

}

int temp=arr[min];

arr[min]=arr[i];

arr[i]=temp;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the no. of elements: ");

int n=sc.nextInt();

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

int arr[]=new int[n];

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

arr[i]=sc.nextInt();

}

selection\_sort(arr);

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

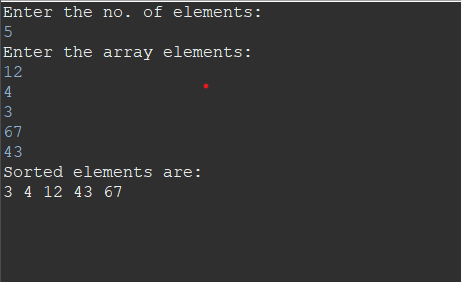
for(int i=0;i<n;i++){

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

}

}

}



1. **Bubble Sort :-**

package com.programs.DAA\_lab;

import java.util.\*;

class Bubble\_sort{

public static void bubble\_sort(int arr[]){

int n=arr.length;

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

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

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

int temp=arr[j];

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

arr[j+1]=temp;

}

}

}

}

public static void main(String args[]){

Scanner sc = new Scanner(System.in);

System.out.println("Enter the no. of elements: ");

int n=sc.nextInt();

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

int arr[]=new int[n];

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

arr[i]=sc.nextInt();

}

bubble\_sort(arr);

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

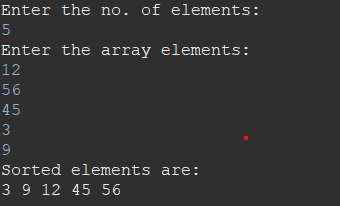
for(int i=0;i<n;i++){

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

}

}

}



**c) Insertion: -**

package com.programs.DAA\_lab;

import java.util.Scanner;

public class Insertion\_sort {

public static void insertion\_sort(int arr[]){

int n=arr.length;

int key, j;

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

key=arr[i];

j=i-1;

while(j>=0 && arr[j]>key)

{

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

j=j-1;

}

arr[j+1]=key;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the no. of elements: ");

int n=sc.nextInt();

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

int arr[]=new int[n];

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

arr[i]=sc.nextInt();

}

insertion\_sort(arr);

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

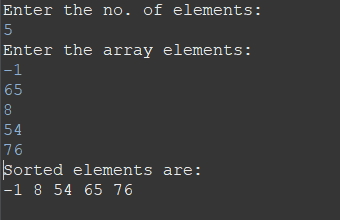
for(int i=0;i<n;i++){

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

}

}

}

****

**d) Quick Sort: -**

package com.programs.DAA\_lab;

import java.util.\*;

public class Quick\_sort{

public static void quicksort(int arr[], int si, int ei){

if(si<ei){

int q=partition(arr,si,ei);

quicksort(arr,si,q-1);

quicksort(arr,q+1,ei);

}

}

public static int partition(int arr[], int si, int ei){

int pivot=arr[ei];

int j=si;

for(int i=si;i<=ei-1;i++){

if(arr[i]<=pivot){

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

j++;

}

}

int temp=arr[j];

arr[j]=arr[ei];

arr[ei]=temp;

return j;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the no. of elements: ");

int n=sc.nextInt();

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

int arr[]=new int[n];

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

arr[i]=sc.nextInt();

}

quicksort(arr,0,n-1);

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

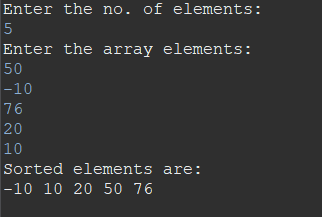
for(int i=0;i<n;i++){

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

}

}

}



**e) Merge Sort: -**

package com.programs.DAA\_lab;

import java.util.\*;

public class Merge\_Sort {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

Merge\_Sort ms = new Merge\_Sort();

ms.sort(arr, 0, n - 1);

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

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

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

}

}

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

if (l < r) {

int m = (l + r) / 2;

sort(arr, l, m);

sort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

public 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, 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];

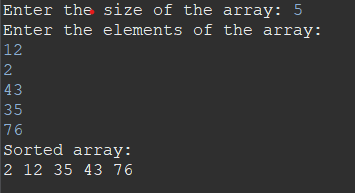
j++;

k++;

}

}

}



**f) Heap Sort:-**

package com.programs.DAA\_lab;

import java.util.Scanner;

public class HeapSort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

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

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

arr[i] = scanner.nextInt();

}

HeapSort hs = new HeapSort();

hs.sort(arr);

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

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

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

}

}

public void sort(int[] arr) {

int n = arr.length;

// Build max heap

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

heapify(arr, n, i);

}

// Heap sort

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

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

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

int largest = i;

int left = 2 \* i + 1;

int 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 temp = arr[i];

arr[i] = arr[largest];

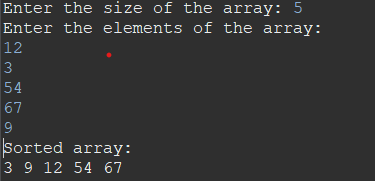
arr[largest] = temp;

heapify(arr, n, largest);

}

}

}



**g) Counting Sort:-**

package com.programs.DAA\_lab;

import java.util.Scanner;

public class CountingSort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array (between 0 and 9):");

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

arr[i] = scanner.nextInt();

}

CountingSort cs = new CountingSort();

cs.sort(arr);

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

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

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

}

}

public void sort(int[] arr) {

int n = arr.length;

int[] count = new int[10];

int[] output = new int[n];

// Count the occurrences of each element

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

count[arr[i]]++;

}

// Modify count to show the cumulative sum

for (int i = 1; i < 10; i++) {

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

}

// Build the output array

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

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

count[arr[i]]--;

}

// Copy the output array to the input array

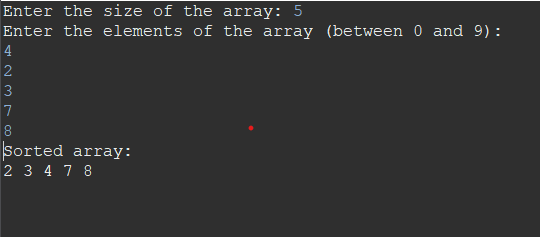
for (int i = 0; i < n; i++) {

arr[i] = output[i];

}

}

}



**Implementation of BFS and DFS**

1. **BFS: -**

package com.programs.DAA\_lab;

import java.util.\*;

public class BFS {

private int V;

private LinkedList<Integer>[] adj;

public BFS(int v) {

V = v;

adj = new LinkedList[V];

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

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

}

}

public void addEdge(int v, int w) {

adj[v].add(w);

}

public void bfs(int s) {

boolean[] visited = new boolean[V];

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

visited[s] = true;

queue.add(s);

while (queue.size() != 0) {

s = queue.poll();

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

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();

BFS g = new BFS(v);

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

int e = scanner.nextInt();

System.out.println("Enter the edges (u v):");

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

int u = scanner.nextInt();

int w = scanner.nextInt();

g.addEdge(u, w);

}

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

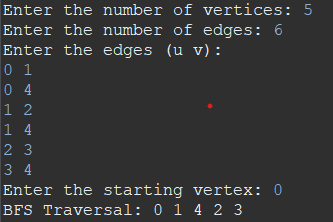
int s = scanner.nextInt();

System.out.print("BFS Traversal: ");

g.bfs(s);

}

}



1. **DFS:-**

package com.programs.DAA\_lab;

import java.util.\*;

public class DFS {

private int V;

private LinkedList<Integer>[] adj;

public DFS(int v) {

V = v;

adj = new LinkedList[V];

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

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

}

}

public void addEdge(int v, int w) {

adj[v].add(w);

}

public void dfs(int s) {

boolean[] visited = new boolean[V];

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

visited[s] = true;

st.push(s);

while (st.size() != 0) {

s = st.pop();

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

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

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

if (!visited[n]) {

visited[n] = true;

st.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();

DFS g = new DFS(v);

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

int e = scanner.nextInt();

System.out.println("Enter the edges (u v):");

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

int u = scanner.nextInt();

int w = scanner.nextInt();

g.addEdge(u, w);

}

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

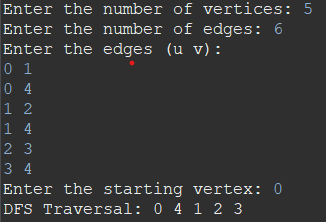
int s = scanner.nextInt();

System.out.print("DFS Traversal: ");

g.dfs(s);

}

}



**Searching**

**1). Linear Searching:-**

package com.programs;

import java.util.\*;

public class Linear\_Search {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the no. of elements: ");

int n = sc.nextInt();

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

int arr[] = new int[n];

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

arr[i]=sc.nextInt();

}

System.out.println("Enter the elements you want to search:");

int item = sc.nextInt();

System.out.println("Search element is present at index :");

System.out.println(search(arr, item));

}

public static int search(int[] arr, int item) {

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

if (arr[i] == item) {

return i;

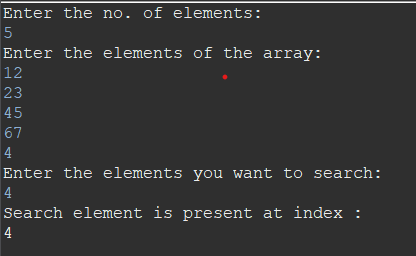
}

}

return -1;

}

}



1. **Binary Search:-**

package com.programs;

import java.util.Scanner;

public class Binary\_search {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the no. of elements: ");

int n=sc.nextInt();

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

int arr[]=new int[n];

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

arr[i]=sc.nextInt();

}

System.out.println("Enter the elements you want to search:");

int item=sc.nextInt();

System.out.println("Search element is present at index :");

System.out.println(Search(arr, item));

}

public static int Search(int[] arr, int item) {

int si = 0;

int ei = arr.length - 1;

while (si <= ei) {

int mid = (si + ei) / 2;

if (arr[mid] == item) {

return mid;

} else if (arr[mid] > item) {

ei = mid - 1;

} else {

si = mid + 1;

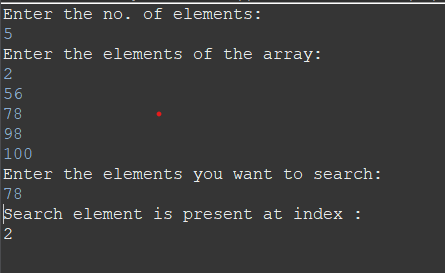
}

}

return -1;

}

}

****

**Minimum Spanning Tree**

1. **Kruskal Algorithm: -**

package com.programs.DAA\_lab;

import java.util.\*;

public class KruskalAlgorithm {

private static class Edge implements Comparable<Edge> {

int src, dest, weight;

public Edge(int s, int d, int w) {

src = s;

dest = d;

weight = w;

}

@Override

public int compareTo(Edge other) {

return weight - other.weight;

}

}

private static int[] parent;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int V = sc.nextInt();

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

int E = sc.nextInt();

Edge[] edges = new Edge[E];

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

System.out.print("Enter the source vertex of edge " + (i+1) + ": ");

int u = sc.nextInt();

System.out.print("Enter the destination vertex of edge " + (i+1) + ": ");

int v = sc.nextInt();

System.out.print("Enter the weight of edge " + (i+1) + ": ");

int w = sc.nextInt();

edges[i] = new Edge(u, v, w);

}

kruskalMST(edges, V);

}

private static int find(int i) {

if (parent[i] == i) {

return i;

}

return find(parent[i]);

}

private static void union(int i, int j) {

int rootI = find(i);

int rootJ = find(j);

parent[rootI] = rootJ;

}

private static void kruskalMST(Edge[] edges, int V) {

Arrays.sort(edges);

parent = new int[V];

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

parent[i] = i;

}

Edge[] result = new Edge[V-1];

int e = 0;

int i = 0;

while (e < V-1) {

Edge nextEdge = edges[i++];

int srcParent = find(nextEdge.src);

int destParent = find(nextEdge.dest);

if (srcParent != destParent) {

result[e++] = nextEdge;

union(srcParent, destParent);

}

}

printMST(result, V);

}

private static void printMST(Edge[] result, int V) {

System.out.println("Edge Weight");

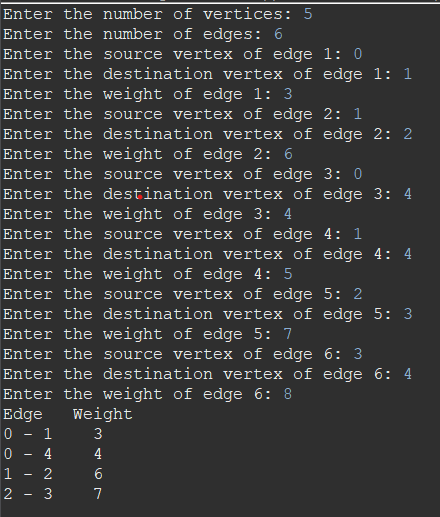
for (int i = 0; i < V-1; i++) {

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

}

}

}



1. **Prims Algorithm**

package com.programs.DAA\_lab;

import java.util.\*;

public class Prims\_Algorithm {

private static int INF = Integer.MAX\_VALUE;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int V = sc.nextInt();

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

int E = sc.nextInt();

int[][] graph = new int[V][V];

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

Arrays.fill(graph[i], INF);

}

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

System.out.print("Enter the source vertex of edge " + (i+1) + ": ");

int u = sc.nextInt();

System.out.print("Enter the destination vertex of edge " + (i+1) + ": ");

int v = sc.nextInt();

System.out.print("Enter the weight of edge " + (i+1) + ": ");

int w = sc.nextInt();

graph[u][v] = w;

graph[v][u] = w;

}

primMST(graph, V);

}

private static void primMST(int[][] graph, int V) {

int[] key = new int[V];

Arrays.fill(key, INF);

boolean[] mstSet = new boolean[V];

int[] parent = new int[V];

Arrays.fill(parent, -1);

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V-1; count++) {

int u = minKey(key, mstSet, V);

mstSet[u] = true;

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

if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

printMST(parent, graph, V);

}

private static int minKey(int[] key, boolean[] mstSet, int V) {

int min = INF, minIndex = -1;

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

if (!mstSet[v] && key[v] < min) {

min = key[v];

minIndex = v;

}

}

return minIndex;

}

private static void printMST(int[] parent, int[][] graph, int V) {

System.out.println("Edge Weight");

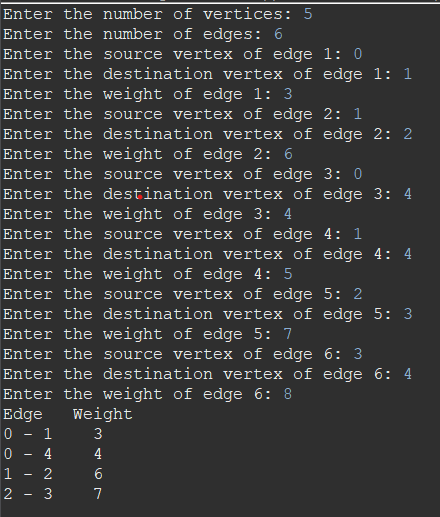
for (int i = 1; i < V; i++) {

System.out.println(parent[i] + " - " + i + " " + graph[i][parent[i]]);

}

}

}



**GREEDY ALGORITHM:-**

**1.** **Fractional KnapSack:**

package com.programs.DAA\_lab;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

public class FractionalKnapsack{

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

Item[] items = new Item[n];

System.out.println("Enter the weight and value of each item:");

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

int weight = scanner.nextInt();

int value = scanner.nextInt();

items[i] = new Item(weight, value);

}

System.out.print("Enter the knapsack capacity: ");

int capacity = scanner.nextInt();

double maxValue = fractionalKnapsack(items, capacity);

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

}

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

Arrays.sort(items, Comparator.comparingDouble(Item::valuePerWeight).reversed());

double maxValue = 0.0;

for (Item item : items) {

if (capacity - item.weight >= 0) {

maxValue += item.value;

capacity -= item.weight;

} else {

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

maxValue += item.value \* fraction;

break;

}

}

return maxValue;

}

static class Item {

int weight;

int value;

public Item(int weight, int value) {

this.weight = weight;

this.value = value;

}

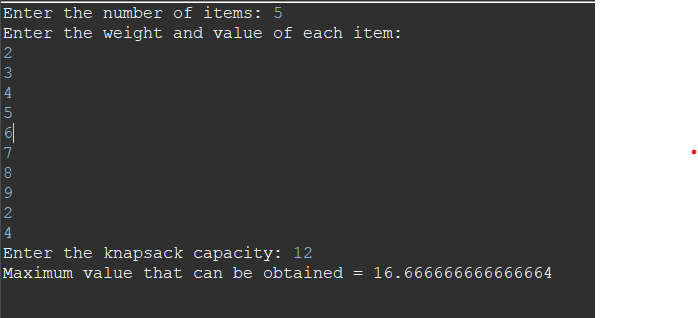
public double valuePerWeight() {

return (double) value / (double) weight;

}

}

}



**Activity Selection**

package com.programs.DAA\_lab;

import java.util.\*;

public class ActivitySelection {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

int n = input.nextInt();

int[] startTimes = new int[n];

int[] endTimes = new int[n];

// Input the start and end times of each activity

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

System.out.print("Enter start time of activity " + (i+1) + ": ");

startTimes[i] = input.nextInt();

System.out.print("Enter end time of activity " + (i+1) + ": ");

endTimes[i] = input.nextInt();

}

// Sort the activities by end time in ascending order

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

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

if (endTimes[i] > endTimes[j]) {

int temp = endTimes[i];

endTimes[i] = endTimes[j];

endTimes[j] = temp;

temp = startTimes[i];

startTimes[i] = startTimes[j];

startTimes[j] = temp;

}

}

}

// Select the activities

int selected = 1;

int lastEnd = endTimes[0];

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

if (startTimes[i] >= lastEnd) {

selected++;

lastEnd = endTimes[i];

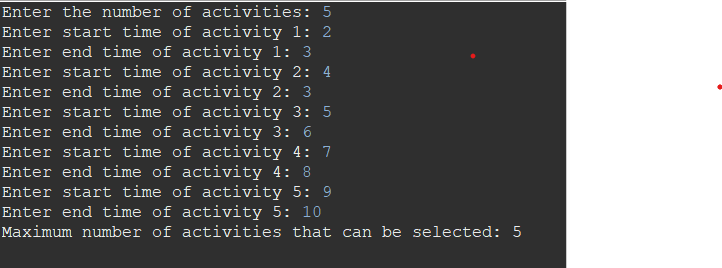
}

}

System.out.println("Maximum number of activities that can be selected: " + selected);

}

}



**Dijkstra Algorithm**

package com.programs.DAA\_lab;

import java.util.\*;

public class DijikstraAlgorithm {

static int INF = Integer.MAX\_VALUE; // infinity value for distances

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int V = sc.nextInt();

int[][] graph = new int[V][V];

System.out.println("Enter adjacency matrix for the graph:");

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

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

graph[i][j] = sc.nextInt();

}

}

System.out.print("Enter source vertex: ");

int source = sc.nextInt();

dijkstra(graph, source);

}

public static void dijkstra(int[][] graph, int source) {

int V = graph.length;

boolean[] visited = new boolean[V];

int[] distance = new int[V];

// initialize all distances to infinity and visited array to false

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

distance[i] = INF;

visited[i] = false;

}

// distance from source vertex to itself is 0

distance[source] = 0;

// find shortest path for all vertices

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

int minDist = INF;

int minIndex = -1;

// find the vertex with minimum distance that has not been visited

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

if (!visited[j] && distance[j] < minDist) {

minDist = distance[j];

minIndex = j;

}

}

// mark the vertex as visited

visited[minIndex] = true;

// update distance of adjacent vertices

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

if (!visited[k] && graph[minIndex][k] != 0 && distance[minIndex] != INF

&& distance[minIndex] + graph[minIndex][k] < distance[k]) {

distance[k] = distance[minIndex] + graph[minIndex][k];

}

}

}

// print the distances

System.out.println("Shortest distances from source vertex " + source + " to all other vertices:");

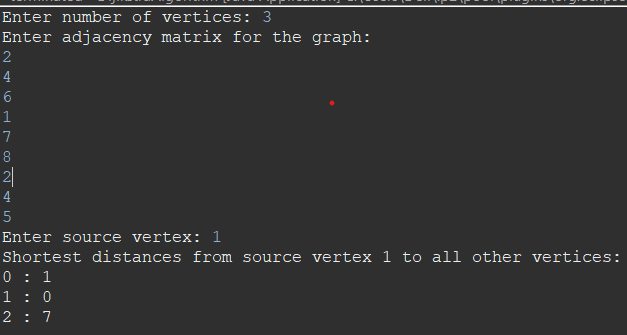
for (int i = 0; i < V; i++) {

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

}

}

}



**BellmanFord Algorithm**

package com.programs.DAA\_lab;

import java.util.\*;

public class BellmanFord {

static int INF = Integer.MAX\_VALUE; // infinity value for distances

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int V = sc.nextInt();

int[][] graph = new int[V][V];

System.out.println("Enter adjacency matrix for the graph:");

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

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

graph[i][j] = sc.nextInt();

}

}

System.out.print("Enter source vertex: ");

int source = sc.nextInt();

bellmanFord(graph, source);

}

public static void bellmanFord(int[][] graph, int source) {

int V = graph.length;

int[] distance = new int[V];

// initialize all distances to infinity except for the source vertex which is 0

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

if (i == source) {

distance[i] = 0;

} else {

distance[i] = INF;

}

}

// relax edges repeatedly

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

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

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

if (graph[j][k] != 0 && distance[j] != INF && distance[j] + graph[j][k] < distance[k]) {

distance[k] = distance[j] + graph[j][k];

}

}

}

}

// check for negative-weight cycles

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

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

if (graph[j][k] != 0 && distance[j] != INF && distance[j] + graph[j][k] < distance[k]) {

System.out.println("Graph contains negative-weight cycle");

return;

}

}

}

// print the distances

System.out.println("Shortest distances from source vertex " + source + " to all other vertices:");

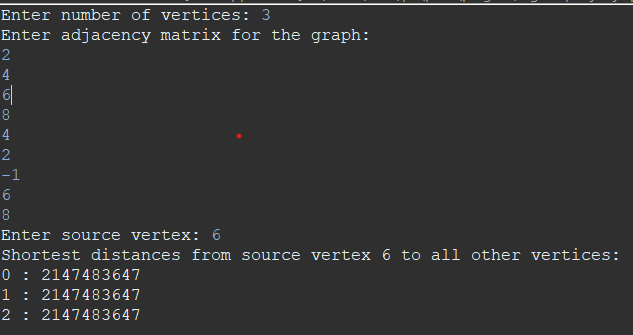
for (int i = 0; i < V; i++) {

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

}

}

}



**Knapsack Problem(DP)**

import java.util.\*;

public class Knap\_sack {

static int max(int a, int b) {

return (a > b) ? a : b;

}

static int knapSack(int W, int wt[], int val[], int n)

{

if (n == 0 || W == 0) {

return 0;

}

if (wt[n - 1] > W) {

return *knapSack*(W, wt, val, n - 1);

}

else {

return *max*(val[n - 1]+ *knapSack*(W - wt[n - 1], wt,val, n - 1),*knapSack*(W, wt, val, n - 1));

}

}

public static void main(String[] args) {

Scanner sc = new Scanner (System.***in***);

System.***out***.println("Enter the no. of elements: ");

int n=sc.nextInt();

System.***out***.println ("Enter the profit: ");

int profit[]=new int[n];

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

profit[i]=sc.nextInt();

}

System.***out***.println("Enter the weight: ");

int weight[]=new int[n];

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

weight[i]=sc.nextInt();

}

System.***out***.println("Enter the capacity: ");

int m=sc.nextInt();

System.***out***.println("The Maximum profit is: ");

System.***out***.println(*knapSack*(m, weight, profit, n));

}

}

