**EXERCISE -** 9

**Job Sequencing & Single source shortest path**

**Aim:** Write a java program to implement a JobSequence problem.

**File name:** JobSequence.java

**Program:**

**import java.util.Scanner;**

**public class JobSequence{**

**public static void main(String[] args){**

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

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

**int NO\_OF\_JOBS = sc.nextInt();**

**int Job[][] = new int[NO\_OF\_JOBS][2];**

**int maxAtIndex = 0,maxDeadLine=0,Profit=0;**

**String sequence = new String("");**

**System.out.println("Enter Deadline Profit: ");**

**for (int i = 0; i < NO\_OF\_JOBS; i++) { //Takes: values**

**for (int j = 0; j < 2; j++) {**

**Job[i][j] = sc.nextInt();**

**}**

**}**

**for (int i = 0; i < NO\_OF\_JOBS; i++) { //Otput: Maximum Deadline**

**if (maxDeadLine < Job[i][0]) {**

**maxDeadLine = Job[i][0];**

**}**

**}**

**for (int i = maxDeadLine; i > 0; i--) {**

**String strJob = new String("");**

**int maxProfit = Job[0][1];**

**for (int j = 0; j < NO\_OF\_JOBS; j++) { //Gets: Maximum Profit**

**if (Job[j][0] >= maxDeadLine && maxProfit < Job[j][1]) {**

**strJob = String.valueOf(j);**

**maxProfit = Job[j][1];**

**maxAtIndex = j;**

**}**

**}**

**// System.out.println(strJob);**

**sequence =(Integer.valueOf(strJob)+1) + "-->" + sequence ; //System.out.println(sequence);**

**Profit+=Job[maxAtIndex][1];//Profit gained**

**Job[maxAtIndex][0]=0;//Job executed**

**maxDeadLine--;**

**}**

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

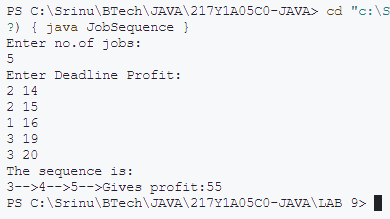
**System.out.println(sequence+"Gives profit:"+Profit);**

**sc.close();**

**}**

**}**

**Output:**

****

**Aim:** Write a java program to implement Single source shortest path problem.

**File name:** SSShortestPath.java

**Program:**

**import java.util.\*;**

**class SSShortestPath {**

**static class Edge { int d, w; Edge(int d, int w) { this.d = d; this.w = w; } }**

**static void dijkstra(int s, List<Edge>[] g, int[] d) {**

**Arrays.fill(d, Integer.MAX\_VALUE);**

**d[s] = 0;**

**PriorityQueue<Edge> q = new PriorityQueue<>(Comparator.comparingInt(a -> a.w));**

**q.offer(new Edge(s, 0));**

**while (!q.isEmpty()) {**

**Edge c = q.poll();**

**for (Edge n : g[c.d]) {**

**int newD = d[c.d] + n.w;**

**if (newD < d[n.d]) { d[n.d] = newD; q.offer(new Edge(n.d, newD)); }**

**}**

**}}**

**public static void main(String[] args) {**

**int n = 5;**

**List<Edge>[] g = new List[n];**

**for (int i = 0; i < n; i++) g[i] = new ArrayList<>();**

**g[0].add(new Edge(1, 2)); g[0].add(new Edge(2, 4));**

**g[1].add(new Edge(2, 1)); g[1].add(new Edge(3, 7));**

**g[2].add(new Edge(3, 1)); g[2].add(new Edge(4, 3));**

**g[3].add(new Edge(4, 5));**

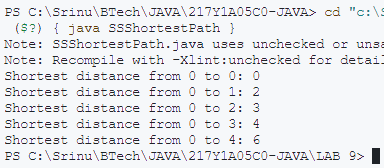
**int[] d = new int[n];**

**dijkstra(0, g, d);**

**for (int i = 0; i < n; i++) System.out.println("Shortest distance from 0 to " + i + ": " + d[i]);**

**}}**

**Output:**

****