NAME – SHRAVANI BAHULEKAR

ROLL N0 – 32

PRN – 12111224

ASSIGNMENT 2

BATCH – D2

import java.util.\*;  
  
public class InformedSearch {  
 public static class Edge {  
 char source;  
 char destination;  
 int weight;  
  
 public Edge(char s, char d, int w) {  
 this.source = s;  
 this.destination = d;  
 this.weight = w;  
 }  
 }  
  
 public static class Node implements Comparable<Node> {  
 char label;  
 int cost; // g(n) + h(n)  
 int heuristic; // Heuristic value for this node  
  
 public Node(char l, int c, int h) {  
 this.label = l;  
 this.cost = c;  
 this.heuristic = h;  
 }  
  
 public int compareTo(Node other) {  
 return Integer.*compare*(this.cost, other.cost);  
 }  
 }  
  
 public static void AStar(ArrayList<Edge>[] graph, int V, char start, char goal) {  
 PriorityQueue<Node> openSet = new PriorityQueue<>();  
 openSet.add(new Node(start, 0, *heuristic*(start, goal)));  
  
 int[] gScore = new int[V];  
 Arrays.*fill*(gScore, Integer.*MAX\_VALUE*);  
 gScore[*index*(start)] = 0;  
  
 int[] cameFrom = new int[V];  
 Arrays.*fill*(cameFrom, -1);  
  
 while (!openSet.isEmpty()) {  
 Node current = openSet.poll();  
 char currentLabel = current.label;  
  
 if (currentLabel == goal) {  
 // Goal reached, reconstruct and print the path  
 *reconstructPath*(start, goal, cameFrom);  
 return;  
 }  
  
 for (Edge edge : graph[*index*(currentLabel)]) {  
 char neighbor = edge.destination;  
 int tentativeGScore = gScore[*index*(currentLabel)] + edge.weight;  
  
 if (tentativeGScore < gScore[*index*(neighbor)]) {  
 // This is a better path  
 cameFrom[*index*(neighbor)] = *index*(currentLabel);  
 gScore[*index*(neighbor)] = tentativeGScore;  
 openSet.add(new Node(neighbor, gScore[*index*(neighbor)] + *heuristic*(neighbor, goal), *heuristic*(neighbor, goal)));  
 }  
 }  
 }  
  
 System.*out*.println("No path found from " + start + " to " + goal);  
 }  
  
 public static int heuristic(char from, char to) {  
 // Assign heuristic values directly to nodes  
 int[] heuristicValues = {10,8,5,7,3,6,5,3,1,0}; // Corresponding to 'A' to 'J'  
 return heuristicValues[*index*(from)];  
 }  
  
 public static int index(char c) {  
 return c - 'A';  
 }  
  
 public static char label(int index) {  
 return (char) ('A' + index);  
 }  
  
 public static void reconstructPath(char start, char goal, int[] cameFrom) {  
 // Implement this function to reconstruct and print the path  
 // You can use the 'cameFrom' array to backtrack from 'goal' to 'start'  
 List<Character> path = new ArrayList<>();  
 char current = goal;  
 while (current != start) {  
 path.add(current);  
 current = *label*(cameFrom[*index*(current)]);  
 }  
 path.add(start);  
 Collections.*reverse*(path);  
 System.*out*.println("Path from " + start + " to " + goal + ": " + path);  
 }  
  
 public static void createGraph(ArrayList<Edge>[] graph) {  
 for (int i = 0; i < 10; i++) {  
 graph[i] = new ArrayList<Edge>();  
 }  
  
 // Add edges to your graph with weights  
 graph[*index*('A')].add(new Edge('A', 'B', 6));  
 graph[*index*('A')].add(new Edge('A', 'F', 3));  
 graph[*index*('B')].add(new Edge('B', 'A', 6));  
 graph[*index*('B')].add(new Edge('B', 'D', 2));  
 graph[*index*('B')].add(new Edge('B', 'C', 3));  
 graph[*index*('C')].add(new Edge('C', 'B', 3));  
 graph[*index*('C')].add(new Edge('C', 'D', 1));  
 graph[*index*('C')].add(new Edge('C', 'E', 5));  
 graph[*index*('D')].add(new Edge('D', 'B', 2));  
 graph[*index*('D')].add(new Edge('D', 'C', 1));  
 graph[*index*('D')].add(new Edge('D', 'E', 8));  
 graph[*index*('E')].add(new Edge('E', 'C', 5));  
 graph[*index*('E')].add(new Edge('E', 'D', 8));  
 graph[*index*('E')].add(new Edge('E', 'I', 5));  
 graph[*index*('E')].add(new Edge('E', 'J', 5));  
 graph[*index*('J')].add(new Edge('J', 'E', 5));  
 graph[*index*('J')].add(new Edge('J', 'I', 3));  
 graph[*index*('I')].add(new Edge('I', 'E', 5));  
 graph[*index*('I')].add(new Edge('I', 'J', 3));  
 graph[*index*('I')].add(new Edge('I', 'G', 3));  
 graph[*index*('I')].add(new Edge('I', 'H', 2));  
 graph[*index*('G')].add(new Edge('G', 'I', 3));  
 graph[*index*('G')].add(new Edge('G', 'F', 1));  
 graph[*index*('F')].add(new Edge('F', 'A', 3));  
 graph[*index*('F')].add(new Edge('F', 'G', 1));  
 graph[*index*('F')].add(new Edge('F', 'H', 7));  
 graph[*index*('H')].add(new Edge('H', 'I', 2));  
 graph[*index*('H')].add(new Edge('H', 'F', 7));  
 }  
  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int V = 10;  
 ArrayList<Edge>[] graph = new ArrayList[V];  
 *createGraph*(graph);  
  
 char start = 'A';  
 char goal = 'J';  
  
 // Call the A\* algorithm to find the path  
 *AStar*(graph, V, start, goal);  
 }  
}