PRAJAKTA DEOKULE

3330

C22019221332

ASSIGNMENT: Best First Search

```
package bestFirst;
import java.util.*;
public class Node {
      int data;
      int dist;
      int cost;
       public Node(int data) {
           this.data = data;
           this.dist = 0;
           this.cost = 0;
     }
}
public class Graph {
         int n; //no. of vertices
         int e; //no. of edges
         int[][] adjMat;
          int[] estimates; //to store heuristic distances
         public Graph() {
                  this.n = 0;
                  this.e = 0;
         }
void create() {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter no. of vertices:");
             this.n = sc.nextInt();
            this.estimates = new int[n+1];
```

```
System.out.println("Enter no. of edges:");
this.e = sc.nextInt();
this.adjMat = new int[n+1][n+1];
System.out.println("Enter connecting vertices for each edge:");

for(int i=1; i<e+1; i++) {
        System.out.println("For edge "+i);
         System.out.println("First vertex:");
          int u = sc.nextInt();
          System.out.println("Second vertex:");
          int v = sc.nextInt();

          System.out.println("Enter cost: ");
          int cost = sc.nextInt();
          this.adjMat[u][v] = cost;
          this.adjMat[v][u] = cost;
    }
}
void display() {
        for(int i=1;i<n+1;i++) {
                    for(int j=1;j<n+1;j++)
                      System.out.print(adjMat[i][j]+"\t");
              System.out.println();
         }
}
//function to calculate heuristic/esti mated distances
void estimate(int goal) {
       estimates[goal] = 0;
        //we shall use breadth first search to traverse all other nodes
Queue<Integer> q = new LinkedList<>();
```

```
int[] visited = new int[n+1];
int dist = 0;
int x;
q.add(goal);
visited[goal] = 1;
while(!q.isEmpty()) {
            int curr = q.poll();
             dist = estimates[curr];
              for(int i=1; i<n+1; i++) {
                     x = 0;
                        if(adjMat[curr][i]!=0 && visited[i]==0) {
                                q.add(i);
                                visited[i] = 1;
                                x = adjMat[curr][i];
                                 estimates[i] = dist+x;
                      }//close if
        }//close for
    }//close while
}//close estimate
//function to implement best first search
void bestFirstSearch(int st, int gl) {
             estimate(gl);
             Node start = new Node(st);
             Node goal = new Node(gl);
             int[] visited = new int[n+1];
             Comparator<Node> nodeComparator = new Comparator<Node>(){
                  @Override
                   public int compare(Node n1, Node n2) {
                        return n1.dist-n2.dist;
                 }
        };
```

```
PriorityQueue<Node> pq = new PriorityQueue<>(nodeComparator);
int cost = 0;
pq.add(start);
visited[st] = 1;
System.out.println("\nBest First Search Path:");

while(!pq.isEmpty()) {
        Node u = pq.poll();
         cost = u.cost;
         System.out.print(u.data+"->");
           if(u.data==gl) {
                  System.out.println("\ntotal cost = "+cost);
                  System.out.println("success");
                    return;
        }
for(int i=1; i<n+1; i++) {
              if(adjMat[u.data][i]!=0 && visited[i]==0) {
                     Node temp = new Node(i);
                      temp.dist = estimates[i];
                      temp.cost = cost + adjMat[u.data][i];
                      pq.add(temp);
                     visited[i] = 1;
             }//close if
}//close for
}
System.out.println("\nfailure");
}
}

public class Main {
```

```
public static void main(String args[]) {
            Scanner sc = new Scanner(System.in);
            Graph g = new Graph();
              System.out.println("Provide a graph:");
              g.create();
              System.out.println("\nThe provided graph is: ");
              g.display();
               System.out.println("\nEnter start node: ");
                int st = sc.nextInt();
               System.out.println("Enter goal node: ");
                 int gl = sc.nextInt();
                 g.bestFirstSearch(st, gl);
          }
}
//OUTPUT:
/*
Provide a graph:
Enter no. of vertices:
7
Enter no. of edges:
8
Enter connecting vertices for each edge:
For edge 1
First vertex:
1
Second vertex:
2
Enter cost:
5
For edge 2
First vertex:
```

```
1
Second vertex:
3
Enter cost:
2
For edge 3
First vertex:
2
Second vertex:
4
Enter cost:
6
For edge 4
First vertex:
2
Second vertex:
5

Enter cost:
3
For edge 5
First vertex:
2
Second vertex:
6
Enter cost:
4
For edge 6
First vertex:
5
Second vertex:
```

```
6
Enter cost:
3
For edge 7
First vertex:
6
Second vertex:
7
Enter cost:
3
For edge 8
First vertex:
3
Second vertex:
7
Enter cost:
1
The provided graph is:
0 5 2 0 0 0 0
5 0 0 6 3 4 0
2 0 0 0 0 0 1
0 6 0 0 0 0 0
0 3 0 0 0 3 0
0 4 0 0 3 0 3
0 0 1 0 0 3 0
Enter start node:
1
Enter goal node:
5
Best First Search Path:
1->2->5->
```

```
total cost = 8

success


*/
```