Sakshi Srivastava
B3 - Batch
IBM18CS090

```
import java.util.*;
class Edge {

int src, dest, w;
public Edge (int src, int dest, int w) {
        this.src = src;
        this.dest = dest;
        this.w = w;
    }
}
class Node {
  int vertex, w;
public Node (int vertex, int w) {
        this.vertex = vertex;
        this.w = w;
    }
}
class Graph {

List<List<Edge>> edgeList = null;
Graph (List<Edge> edges, int N) {
        edgeList = new ArrayList<>();
        for (int i=0; i<N; i++) {

            edgeList.add new ArrayList<>());
    }

    for( Edge edge : edges) {

      edgeList.get (Edge.src).add(edge);
  }
  }
}
```

```
class Dijkstra {
private static void getPath(int[] prev,
int i, List<Integer> route) {
  if (i >= 0) {
    getPath(prev, prev[i], route);
    route.add(i);
  }
}
public static void getShortestPath(Graph
    graph, int src, int N) {
PriorityQueue<Node> minHeap;
minHeap = new PriorityQueue<>
    (Comparator.comparingInt(node
                -> node.w));
minHeap.add(new Node(src, 0));
List<Integer> dist = new ArrayList<>
    (Collections.nCopies(N, Integer.
                MAX_VALUE));
  dist.set(src, 0);
boolean[] done = new boolean[N];
  done[src] = true;
int[] prev = new int[N];
  prev[src] = -1
List<Integer> route = new ArrayList<>();
while(!minHeap.isEmpty()) {
  Node node = minHeap.poll();
  int u = node.vertex;
```

```
for ( Edge edge : graph.edgeDist.get(u)){
    int v = edge.dest;
    int w = edge.w;
if(!done[v] && (dist.get(u)+w) < dist.get(v)){
        dist.set(v, dist.get(u)+w);
            prev[v] = u;
        minHeap.add(new Node(v, dist.get(v)));
    }
  }
    done[u] = true;
}
for(int i=1; i<N; i++){

  if(i!=src && dist.get(i) != Integer.MAX_VALUE)
    {
        getPath(prev, i, route);

  S.O.Println("Route is %d => %d & min
          cost =%d & path is %s\n",
          src, i, dist.get(i), route);

    route.clear();
  }
}
}
```