Sabrat Singh
1BM18CS094.

CN LAB

Dijkstra's Algorithm

```
import java.util.*
class Edge {
    int src, dst, w;
    public Edge(s, d, w){
        this.src = s;
        this.dst = d;
        this.w = w;
}}
class Node {
    int vertex, w;
    public node(int v, int w){
        this.vertex = v;
        this.w = w;
}}
class Graph{
    List<List<Edge>> adjList = null;
    Graph(List<Edge> edges, int n){
    adjList = new ArrayList<>();
    for(int i = 0; i<n; i++){
    adjList.add(new ArrayList<>());
}
for(Edge edge: edges){
    adjList.get(edge.src).add(edge);
}
}}
class Dijkstra{
static void getRoute(int[] prev, int i, List
<Integer> route)
{
    if(i>0){
```

```
getRoute(prev, prev[i], route);
route.add(i);
}
public static void shortestPath(Graph graph, int src,
int n){
PriorityQueue<Node> minHeap;
minHeap=new PriorityQueue<>(Comparator.
comparing Int(node -> node.weight));
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.adjList.get(u)){
int v = edge.dst;
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[v]=true;
for(int i=1; i<n; i++){
if(i!=src && dist.get(i)!=Integer.MAX_VALUE)
```

```
        Set Route (prev, r, Route);
System.out.println("Path(%d->%d): MinCost =
    %d and Route is %s", src, r, dist.get(i).
    Route);
    Route.clear();
  }
}
```