Version here does not optimize the Dij Alg with a heap or a priority queue.

```
function dijkstra(DirectedGraph graph) {
   visited <- a set of visited vertices          k0
   unvisited <- a set of unvisited vertices

   copy every vertex into the unvisited   O(V)

   relax[] <- an array with length of V, used for relaxing vertices       k1
   prev[] <- an array keeping the previous vertex to record the route, length = V

   initialize 2 arrays, relax[every index] = infinity, prev[every index] = -1      O(V)

   minRoute(source, target)       O(V^2)  see below
}
```

Conclusion:
O(V + V^2) = O(V^2)

```
function minRoute(int source, int target) {
   boolean found = false;
 int node = source;                  k2
 relax[source] = 0;    // set source to 0, all others infinity by default
 prev[source] = -1;

 while(!found)
 {
    node = findNextNode(node);      O(V) see below

    if(node == target)    // quit when reaches the target       k8
     found = true;

    visited.add(graph.getLabel(node));    // add the current node to Set: visited       k9

    relaxEdge(node);   // relax the edges          O(V) see below

    unvisited.remove(graph.getLabel(node));  // remove the node from Set: unvisited, means it's done
      O(V) in worst based on the set implementation

    if(unvisited.isEmpty()) {          k10
         System.out.println("No route");    // if all nodes are visited and still in the loop, means there is no route
         break;
    }
 }
}
```

In the worst case, every vertex in the graph is visited until it reach the target.
So the loop runs V times, and complexitiy -> O(V * (V + k8 + k9 + V + k10)) = O(V^2)

```
function findNextNode(node) {
   int min = Integer.MAX_VALUE;    // default value       k3
```

```
 for(int i = 0; i < relax.length; i++)          O(V)
    if(relax[i] < min && unvisited.contains(graph.getLabel(i)) && relax[i] > 0)  // get the next smallest relaxation value, and make sure it is in the unvisited Set
     min = relax[i];            k4

 for(int i = 0; i < relax.length; i++)       O(V)
    if(min == relax[i] && min != Integer.MAX_VALUE && unvisited.contains(graph.getLabel(i)))
     node = i;     // when you get it, get the index       k5

 return node;
}
```

O(k3 + V(k4) + V(k5)) = O(V)

Assume E = V - 1 for every vertex in the worst case, which means every vertex has a path to every other vertex, so the edge for the whole graph will reach the maximum = V^2

```
function relaxEdge(current <- currentNode) {
   for(int i = 0; i < relax.length; i++) {         O(V)
    if((graph.getEdge())[current][i] && !visited.contains(graph.getLabel(i))) {         k6
     if(relax[i] == Integer.MAX_VALUE || (relax[current] + graph.getWeight(current, i)) < relax[i]) {        k7
        relax[i] = graph.getWeight(current, i) + relax[current];
        prev[i] = current;    // relax algorithm of Dijkstra
     }
    }
 }
}
```

O(V * (k6 + k7)) = O(V)