Lab 7: Dijkstra's Algorithm,

In this lab you will implement dijkstra's shortest path algorithm as described in Fig 4.8. You can use the **HeapDict** priority queue implementation that works like a dictionary, and supports both insert and decrease key operations by simply setting the dictionary key to a given value. It also supports the delete min operation via **popitem**.
Your script should take a input a graph with weights, and a starting vertex v, and it should output the shortest path length from v to all vertices in the graph, and you should also output the short path as a list of vertices (using the **prev** pointers in Fig 4.8).

The input graph will be specified as a pair of edges and weights; the example graph shown in Fig 4.9 is available as:

```
1 2 4
1 4 2
2 3 2
2 4 3
2 5 3
4 2 1
4 3 4
4 5 5
5 3 1
```

For example, the first line 1 2 4 means that the directed edge (1,2) has weight 4.
The output should be as shown in Fig 4.9 when called with 1 as the source node, as follows:

```
1: 0, [ 1 ]
2: 3, [ 1,4,2 ]
3: 5, [ 1,4,2,3 ]
4: 2, [ 1,4 ]
5: 6, [ 1,4,2,5 ]
```

The format per line is vertex id: distance from v, shortest path from v

Test your code on the following dataset: rome99.txt (posted on Piazza). This graph contains a large portion of the directed road network of the city of Rome, Italy, from 1999. The graph contains 3353 vertices and 8870 edges.