**Algorithm: Uniform_Cost_Search (Applied to Route Finding)**

**Input: Graph *G=(V, E), start, goal*.**
**Every *edge* in *E* in *G* has a non-negative integer weight > 0.**

**Output: A List containing the shortest distance path and route in format [distance, [path]]**

**Procedure:**

```
If start = goal
        return [0, []]
Initialize frontier as empty list
Initialize explored as empty list
Append [0, [start]] to frontier
Loop
        If frontier empty
                Return [0, []]
         Find lowest Cost node in frontier and Pop out of frontier into path
         // Path should have structure [distance, [List of Path]]
        Get city from path  # The last city in the path
        Get pathWeight from path
        Append city to explored
        If city = goal
                Return path
        Initialize cities as connected cities from city  // cities is a list of tuples [city, distance]
        For v in cities
                Get Node c from cities
                Get weight w from cities
                Initialize newPathRoute = path + [c]
                Initialize newPathWeight = pathWeight + w
                Initialize newPath = [newPathWeight, newPathRoute]
                If c not in frontier and c not in explored
                        Append newPath to frontier
```

**Note:** Usually this Algorithm is implemented with the *frontier* as a Priority Queue. Then *frontier* is automatically sorted least to greatest by path weight and access is *O(1).* For above implementation, do a linear search on *frontier* to find the lowest distance path in each Loop. Once you have working, you can experiment with making *frontier* a Priority Queue or by doing a sorting step every time you append a new path to *frontier*.