Week-5 Practice Programming Assignment

Problem 1

A courier company **XYZ** provides courier service between `n` cities labeled `0` to `n-1`, where customers can send items from any city to any another city. The company follows the shortest path to deliver items and charges `5 Rs.` per kilometer distance. The company wants to develop an inquiry system where customers can get the information on the cost and route for their courier.

Write a class **XYZ_Courier** that accepts a weighted adjacency list `Route_map` for an undirected and connected graph at the time of object creation in following format:-

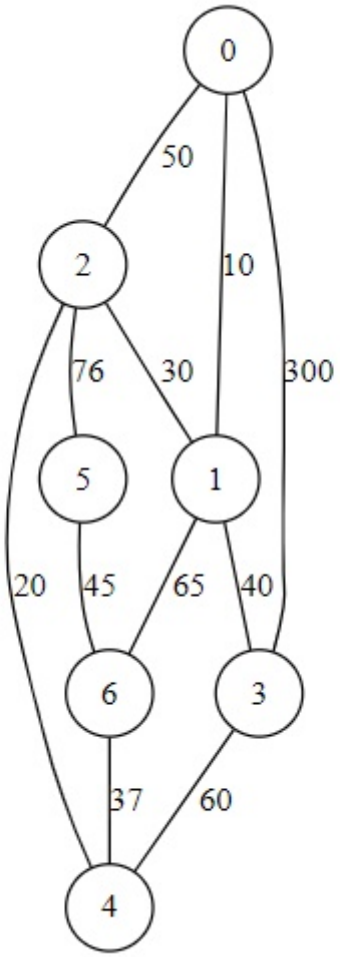
```
Route_map = {
    source_index : [(destination_index,distance),
(destination_index,distance),..],
    ..
    ..
    source_index : [(destination_index,distance),
(destination_index,distance),..]
}
```

The class has following methods:-

- `cost(source, destination)` that accepts source name and destination name and returns minimum cost for delivery.
- `route(source, destination)` that accepts source name and destination name and returns the shortest route for delivery in the following format:

  `[source, place1, place2, ..., destination]`

For the given graph



Sample input 1

```
7 # number of vertices
[(0,1,10),(0,2,50),(0,3,300),(5,6,45),(2,1,30),(6,4,37),(1,6,65),(2,5,76),
(1,3,40),(3,4,60),(2,4,20)] # edges
0 #source
4 #destination
```

Output

```
300 #cost
[0, 1, 2, 4] #route
```

Sample input 2

```
7
[(0,1,10),(0,2,50),(0,3,300),(5,6,45),(2,1,30),(6,4,37),(1,6,65),(2,5,76),
(1,3,40),(3,4,60),(2,4,20)]
2
6
```

Output

```
285
[2, 4, 6]
```

Solution

Solution Code

```
class XYZ_Courier:
    def __init__(self,Route_map):
        self.Route_map = Route_map
    def dijkstra(self,WList,s):
        infinity = 1 + len(WList.keys())*max([d for u in WList.keys()for (v,d) in WList[u]])
        (visited,distance,prev) = ({},{},{})
        for v in WList.keys():
            (visited[v],distance[v],prev[v]) = (False,infinity,None)
        distance[s] = 0
        for u in WList.keys():
            nextd = min([distance[v] for v in WList.keys() if not visited[v]])
            nextvlist = [v for v in WList.keys() if (not visited[v]) and distance[v] == nextd]
            if nextvlist == []:
                break
            nextv = min(nextvlist)
            visited[nextv] = True
            for (v,d) in WList[nextv]:
                if not visited[v]:
                    if distance[v] > distance[nextv]+d:
                        distance[v] = distance[nextv]+d
                        prev[v] = nextv
        return(distance,prev)

    def cost(self,source,destination):
        distance,path = self.dijkstra(self.Route_map, source)
        return 5 * distance[destination]

    def route(self,source,destination):
        distance,path = self.dijkstra(self.Route_map, source)
        Route=[]
        if distance[destination]!=0:
            dest = destination
            while dest != source:
                Route = [dest] + Route
                for i,j in path.items():
                    if dest == i:
                        dest = j
                        break
            Route = [dest] + Route
        return Route
```

Suffix Code(visible)

```
size = int(input())
edges = eval(input())
s=int(input())
d=int(input())
WL = {}
for i in range(size):
    WL[i] = []
for ed in edges: #for create list for undirected graph
    WL[ed[0]].append((ed[1],ed[2]))
    WL[ed[1]].append((ed[0],ed[2]))
C = XYZ_Courier(WL)
print(C.cost(s,d))
print(C.route(s,d))
```

Public Test case

Input 1

```
7
[(0,1,10),(0,2,50),(0,3,300),(5,6,45),(2,1,30),(6,4,37),(1,6,65),(2,5,76),
(1,3,40),(3,4,60),(2,4,20)]
0
4
```

Output 1

```
300
[0, 1, 2, 4]
```

Input 2

```
7
[(0,1,10),(0,2,50),(0,3,300),(5,6,45),(2,1,30),(6,4,37),(1,6,65),(2,5,76),
(1,3,40),(3,4,60),(2,4,20)]
2
6
```

Output 2

```
285
[2, 4, 6]
```

Input 3

```
7
[(0,1,10),(0,2,50),(0,3,300),(5,6,45),(2,1,30),(6,4,37),(1,6,65),(2,5,76),
(1,3,40),(3,4,60),(2,4,20)]
0
6
```

Output 3

```
375
[0, 1, 6]
```

Private Test Case

Input 1

```
7
[(0,1,10),(0,2,50),(0,3,300),(5,6,45),(2,1,30),(6,4,37),(1,6,65),(2,5,76),
(1,3,40),(3,4,60),(2,4,20)]
3
5
```

Output 1

```
710
[3, 4, 6, 5]
```

Input 2

```
7
[(0,1,10),(0,2,20),(0,3,30),(5,6,120),(2,1,5),(6,4,20),(1,6,15),(2,5,70),
(1,3,7),(3,4,100),(2,4,50)]
0
4
```

Output 2

```
225
[0, 1, 6, 4]
```

Input 3

```
7
[(0,1,10),(0,2,20),(0,3,30),(5,6,120),(2,1,5),(6,4,20),(1,6,15),(2,5,70),
(1,3,7),(3,4,100),(2,4,50)]
0
5
```

Output 3

```
425
[0, 1, 2, 5]
```

Input 4

```
6
[(0,1,1),(0,2,6),(1,2,3),(1,3,4),(2,4,4),(2,3,2),(3,4,3),(1,5,2),(2,5,7),
(3,5,1),(4,5,5)]
0
5
```

Output 4

```
15
[0, 1, 5]
```

Problem 2

A taxi driver of an online cab service provider wants to go back to his home after dropping a customer. He wants to reduce the total cost (required for fuel, toll tax, etc.) to reach home by picking some customers. He checks the routes online. So, there are some routes available from his current location to his home location where he can earn money by picking some customers.

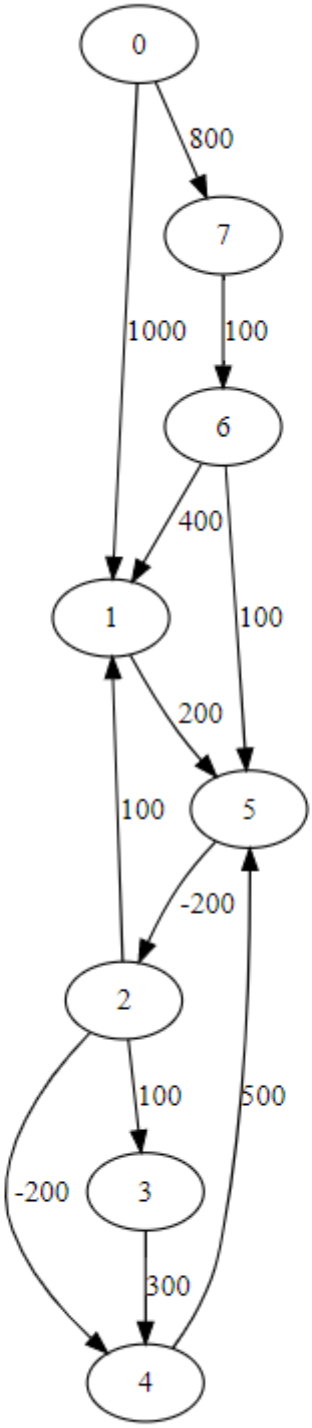
Write a function **min_cost(route_map, source, destination)** that accepts a weighted adjacency list `route_map` for a directed and connected graph of `n` vertices (labeled from 0 to n-1) in the following format:-

```
route_map = {
    source_index : [(destination_index,cost),(destination_index,cost),..],
    ..
    ..
    source_index : [(destination_index,cost),(destination_index,cost),..]
}
```

Note- `cost` between two stops represents `Expenditure (on fuel, toll tax, etc) - Earning` so it may be negative. Assume that no negative weight cycle exists in the graph.

You are also given two integers `source` representing the current location of the taxi driver and `destination` representing the home location of the taxi driver. The function should returns the minimum cost route in the format `(minimum_cost, [source, next_stop, next_stop,.., destination])` from `source` to `destination`.

For the given graph



Sample input 1

```
8 #number of vertices
[(0,1,1000),(0,7,800),(1,5,200),(2,1,100),(2,3,100),(3,4,300),(4,5,500),
(5,2,-200),(2,4,-200),(6,1,400),(6,5,100),(7,6,100)] # edges
0 #source
1 #destination
```

Output

```
(900, [0, 7, 6, 5, 2, 1])
```

Sample input 2

```
8
[(0,1,1000),(0,7,800),(1,5,200),(2,1,100),(2,3,100),(3,4,300),(4,5,500),
(5,2,-200),(2,4,-200),(6,1,400),(6,5,100),(7,6,100)]
0
4
```

Output

```
(600, [0, 7, 6, 5, 2, 4])
```

Solution

Solution Code

```
def bellmanford(WList,s):
    infinity = 1 + len(WList.keys())*max([d for u in WList.keys() for (v,d)
in WList[u]])
    distance = {}
    prev = {}
    for v in WList.keys():
        distance[v] = infinity
        prev[v] = None
    distance[s] = 0
    for i in WList.keys():
        for u in WList.keys():
            for (v,d) in WList[u]:
                if distance[v] > distance[u] + d:
                    distance[v] = distance[u] + d
                    prev[v] = u
    return (distance,prev)


def min_cost(route_map,source,destination):
    distance1,path1 = bellmanford(route_map, source)
    tot_dist = distance1[destination]
    Route_S_D = []
  # shortest route for source to destination
    if distance1[destination] != 0:
        dest = destination
        while dest != source:
            Route_S_D = [dest] + Route_S_D
            for i,j in path1.items():
                if dest == i:
                    dest = j
                    break
        Route_S_D = [dest] + Route_S_D
    return (tot_dist,Route_S_D)
```

Suffix Code(visible)

```
size = int(input())
edges = eval(input())
s = int(input())
d = int(input())
WL = {}
for i in range(size):
    WL[i] = []
for ed in edges: #for create list for directed graph
    WL[ed[0]].append((ed[1],ed[2]))
print(min_cost(WL,s,d))
```

Public Test case

Input 1

```
8
[(0,1,1000),(0,7,800),(1,5,200),(2,1,100),(2,3,100),(3,4,300),(4,5,500),
(5,2,-200),(2,4,-200),(6,1,400),(6,5,100),(7,6,100)]
0
1
```

Output 1

```
(900, [0, 7, 6, 5, 2, 1])
```

Input 2

```
8
[(0,1,1000),(0,7,800),(1,5,200),(2,1,100),(2,3,100),(3,4,300),(4,5,500),
(5,2,-200),(2,4,-200),(6,1,400),(6,5,100),(7,6,100)]
0
4
```

Output 2

```
(600, [0, 7, 6, 5, 2, 4])
```

Input 3

```
8
[(0,1,1000),(0,7,800),(1,5,200),(2,1,100),(2,3,100),(3,4,300),(4,5,500),
(5,2,-200),(2,4,-200),(6,1,400),(6,5,100),(7,6,100)]
6
4
```

Output 3

```
(-300, [6, 5, 2, 4])
```

Private Test Case

Input 1

```
8
[(0,1,10),(0,3,20),(2,0,50),(1,3,-25),(3,2,-5),(4,1,70),(3,4,-30),(2,5,20),
(5,3,-10),(4,6,10),(6,3,40),(5,7,50),(5,6,35),(6,7,15),(7,4,60)]
0
7
```

Output 1

```
(-20, [0, 1, 3, 4, 6, 7])
```

Input 2

```
8
[(0,1,1000),(0,7,800),(1,5,200),(2,1,100),(2,3,100),(3,4,300),(4,5,500),
(5,2,-200),(2,4,-200),(6,1,400),(6,5,100),(7,6,100)]
0
2
```

Output 2

```
(800, [0, 7, 6, 5, 2])
```

Input 3

```
8
[(0,1,10),(0,3,20),(2,0,50),(1,3,-25),(3,2,-5),(4,1,70),(3,4,-30),(2,5,20),
(5,3,-10),(4,6,10),(6,3,40),(5,7,50),(5,6,35),(6,7,15),(7,4,60)]
2
7
```

Output 3

```
(5, [2, 5, 3, 4, 6, 7])
```

Input 4

```
8
[(0,1,10),(0,3,20),(2,0,50),(1,3,-25),(3,2,-5),(4,1,70),(3,4,-30),(2,5,20),
(5,3,-10),(4,6,10),(6,3,40),(5,7,50),(5,6,35),(6,7,15),(7,4,60)]
7
0
```

Output 4

```
(150, [7, 4, 1, 3, 2, 0])
```