Week 5 Live Coding Solutions

Week-5 Live coding problem 1

An Airline company wants to make airport to connect `n` cities labeled `0` to `n-1` all across the country . Write a function **Airport(distance_map)** that accepts a weighted adjacency list `distance_map` in the following format:-

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

The function returns the minimum distance of airport network to connect all `n` cities.

Sample input

```
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)
```

Output

```
182
```

Solution

Solution Code

```python
def kruskal(WList):
    (edges,component,TE)=([],{},[])
    for u in WList.keys():
        edges.extend([(d,u,v) for (v,d) in WList[u]])
        component[u] = u
    edges.sort()
    for (d,u,v) in edges:
        if component[u] != component[v]:
            TE.append((u,v))
            c = component[u]
        for w in WList.keys():
            if component[w] == c:
                component[w] = component[v]
    return(TE)

def Airport(distance_map):
    R = kruskal(distance_map)
    S = 0
    for e in R:
        for ed in distance_map[e[0]]:
            if ed[0]==e[1]:
                S += ed[1]
    return S
```

Suffix code(visible)

```
size = int(input())
edges = eval(input())
WL = {}
for i in range(size):
    WL[i] = []
for ed in edges:
    WL[ed[0]].append((ed[1],ed[2]))
    WL[ed[1]].append((ed[0],ed[2]))
print(Airport(WL))
```

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)]
```

Output

```
182
```

Input 2

```
6
[(0,1,16),(0,3,2),(1,2,4),(3,4,10),(0,4,9),(3,5,15),(1,5,7),(2,5,6)]
```

Output

```
36
```

Input 3

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

Output

```
6
```

Private Test case

Input 1

```
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)]
```

Output

```
9
```

Input 2

```
7
[(0,1,10),(1,2,50),(2,3,60),(3,0,75),(3,1,80),(6,4,90),(1,6,100),(2,5,110),(3,6,150),(3,4,180),(0,4,200)]
```

Output

```
420
```

Input 3

```
6
[(0,1,1),(1,2,3),(1,3,4),(1,4,5),(0,4,7),(0,5,10),(2,3,12),(3,4,13),(1,5,15),(2,5,17),(3,5,21),(4,5,25)]
```

Output

```
23
```

Week-5 Live coding problem 2

You are given a network of `n` nodes, labelled from `0` to `n-1`. You are also given `travel_times`, a list of signal travel times in as directed edges `travel_times[i] = (ui, vi, wi)`, where `ui` is the source node, `vi` is the target node, and `wi` is the time it takes for a signal to travel from source to target.

Write a function **min_transmission_time(n, travel_times, s)** that accept number of nodes `n`, a list `travel_times` and a source node `s` to send the signal . The function returns the **minimum** time required for the signal sent by the source node `s` to be received by all the remaining `n-1` nodes. If it is impossible to obtain a signal for all `n-1` nodes, return `-1`.

Sample Input 1

```
4 #n
[(2,1,1),(2,3,1),(3,4,1)] #travel_times
2 #s
```

Output

```
2
```

Sample Input 2

```
4
[(2,1,1),(2,3,1),(4,3,1)]
2
```

Output

```
-1
```

Sample Input 3

```
7
[(0,1,10),(0,2,80),(1,2,6),(1,4,20),(2,3,70),(4,5,50),(4,6,5),(5,6,10)]
0
```

Output

```
86
```

Solution

Solution

```python
def dijkstralist(WList,s):
    infinity = 1 + len(WList.keys())*max([d for u in WList.keys() for (v,d) in WList[u]])
    (visited,distance) = ({},{})
    for v in WList.keys():
        (visited[v],distance[v]) = (False,infinity)

    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]
        nextv = min(nextvlist)
        visited[nextv] = True
        for (v,d) in WList[nextv]:
            if not visited[v]:
                distance[v] = min(distance[v],distance[nextv]+d)
    return(distance,infinity)
def min_transmission_time(n, travel_times, s):
    AList={}
    for i in range(n):
        AList[i]=[]
    for u,v,d in travel_times:
        AList[u].append((v,d))
    dist,inf = dijkstralist(AList,s)
    maxtime = 0
    for node,distance in dist.items():
        if distance >= maxtime:
            maxtime = distance
    if maxtime >= inf:
        return -1
    else:
        return maxtime
```

Suffix Code

```python
n = int(input())
edges = eval(input())
s = int(input())
print(min_transmission_time(n, edges, s))
```

Public Test Case

Input 1

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

Output

```
2
```

Input 2

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

Output

```
-1
```

Input 3

```
7
[(0,1,10),(0,2,80),(1,2,6),(1,4,20),(2,3,70),(4,5,50),(4,6,5),(5,6,10)]
0
```

Output

```
86
```

Private Test Case

Input 1

```
5
[(0,1,1000),(0,2,500),(0,3,3000),(2,0,2000),(2,1,3000),(1,3,300),(3,4,600),(2,4,2000)]
0
```

Output

```
1900
```

Input 2

```
5
[(0,1,1000),(0,2,500),(0,3,3000),(2,0,2000),(2,1,3000),(1,3,300),(4,3,600),(4,2,2000)]
0
```

Output

```
-1
```

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)]
0
```

Output

```
1400
```

Input 4

```
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
```

Output

```
-1
```

Input 4

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

Output

```
1000
```

Week-5 Live coding problem 3

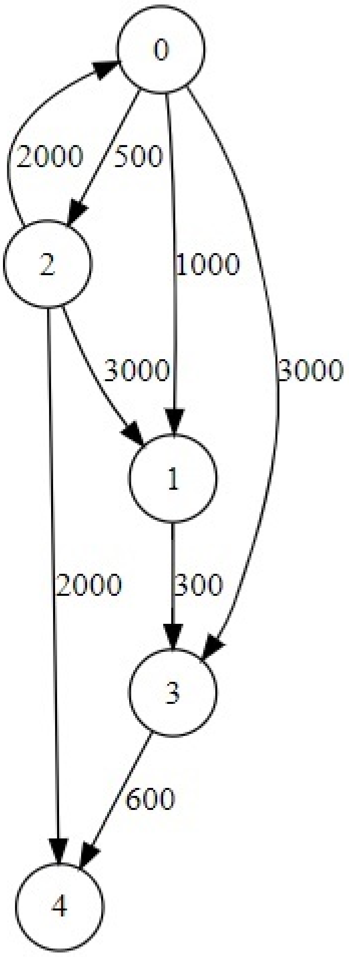
An airlines company has flights operational in `n` cities labeled `0` to `n-1`. Write a function **best_fare(flight_route, source, destination, k)** in which you are given a weighted adjacency list `flight_route` in the following format:-

```
flight_route = {
    source_index : [(destination_index,price),(destination_index,price),..],
    ..
    ..
    source_index : [(destination_index,price),(destination_index,price),..]

}
```

You are also given three integers `source`, `destination` and `k(positive integer)`, function returns minimum cost and flight route in the format `(minimum_cost, [source, next_stop, next_stop,.., destination])` from `source` *to* `destination` with at most `k` stops in between (source and destination are not included). If there is no such route, return string `Not found`.

For the given graph



Sample Input-1

```
5 # number of vertices
[(0,1,1000),(0,2,500),(0,3,3000),(2,0,2000),(2,1,3000),(1,3,300),(3,4,600),(2,4,2000)] #edges
0 # source
4 # destination
1 # k (Maximum stops allowed in route)
```

Output

```
(2500, [0, 2, 4])
```

Sample Input-2

```
5
[(0,1,1000),(0,2,500),(0,3,3000),(2,0,2000),(2,1,3000),(1,3,300),(3,4,600),(2,4,2000)]
0
4
0
```

Output

```
Not found
```

Solution

Prefix code(visible)

```
def addallpath(WList,u, d, visited, path,allpath):
    visited[u]= True
    path.append(u)
    if u == d:
         L = path.copy()
         allpath.append(L)
    else:
         for i in WList[u]:
             if visited[i[0]]== False:
                 addallpath(WList, i[0], d, visited, path, allpath)
    path.pop()
    visited[u]= False
# Following function returns a list of all paths from s to d
# Format of returned list:- [[s,...,d],[s,...,d],...]
def findallpath(WList,s,d):
    visited = {}
    allpath = []
    for v in WList.keys():
        visited[v] = False
    path = []
    addallpath(WList,s, d, visited, path,allpath)
    return(allpath)
```

Solution Code

```
def best_fare(flight_route,source,destination,k):
    L = findallpath(flight_route,source,destination)
    if L != []:
        cost = 1 + len(flight_route.keys())*max([d for u in flight_route.keys() for (v,d) in flight_route[u]])
        route = []
        for pth in L:
            if len(pth) < k+3:
                s = 0
                for i in range(0,len(pth)-1):
                    for j in flight_route[pth[i]]:
                        if pth[i+1] == j[0]:
                            s += j[1]
                if s < cost:
                    cost = s
                    route = pth
        if route != []:
            return (cost,route)
        else:
            return 'Not found'
    else:
        return 'Not found'
```

Prefix code(visible)

```
size = int(input())
edges = eval(input())
s = int(input())
d = int(input())
k = int(input())
WL = {}
for i in range(size):
    WL[i] = []
for ed in edges:
    WL[ed[0]].append((ed[1],ed[2]))
print(best_fare(WL,s,d,k))
```

Public Test case

Input 1

```
5
[(0,1,1000),(0,2,500),(0,3,3000),(2,0,2000),(2,1,3000),(1,3,300),(3,4,600),(2,4,2000)]
0
4
1
```

Output

```
(2500, [0, 2, 4])
```

Input 2

```
5
[(0,1,1000),(0,2,500),(0,3,3000),(2,0,2000),(2,1,3000),(1,3,300),(3,4,600),(2,4,2000)]
0
4
0
```

Output

```
Not found
```

Private Test case

Input 1

```
5
[(0,1,1000),(0,2,500),(0,3,3000),(2,0,2000),(2,1,3000),(1,3,300),(3,4,600),(2,4,2000)]
2
4
2
```

Output

```
(2000, [2, 4])
```

Input 2

```
5
[(0,1,1000),(0,2,500),(0,3,3000),(2,0,2000),(2,1,3000),(1,3,300),(3,4,600),(2,4,2000)]
0
4
2
```

Output

```
(1900, [0, 1, 3, 4])
```