```
from queue import PriorityQueue
def astar_algo(graph,src,des):
    visited_node=PriorityQueue()
    visited_node.put((graph[src]['Heuristic'],src))
    explored_node=set()
    #keep tracking of the path cost from start node to any particular node
    weight={src:0}
    #direct estimated distance (Heuristic) from the node to the destination node
    estimated_distance={src:graph[src]['Heuristic']}
    #print(estimated_distance)
    #Path_track
    path={}
    while not visited_node.empty():
        if src==des:
            return (weight[src],src)
        #From the neighbour of the current node select the neighbor node which has the minimum
          heurestic value
        current_node=visited_node.get()[1]
        #print('Current',current_node)
        if current_node==des:
            #Backtracking path from the destination node to the source node
            backtrack=[des]
            while current_node in path:
                current_node=path[current_node]
                backtrack.append(current_node)
            backtrack.reverse()
            # Return the final cost to reach the goal and the path
            return (estimated_distance[des],backtrack)
        #the node is explored therefore
        explored_node.add(current_node[1])

        #Neighbor info of the current executing node
        for neighbor,info in graph[current_node]["Neighbor"].items():
            #print(f"inf: {info}")
            #print(neighbor)
            path_cost=weight[current_node]+info  #Path cost from starting node to particular node
            #Checking if neighbor is already explored or not
            if neighbor in explored_node:
                continue
            elif neighbor not in weight or path_cost<weight[neighbor]:
                #Updating weight and the f(n) as f(n)=g(n)+h(n)
                weight[neighbor]=path_cost
```

```
            estimated_distance[neighbor]=path_cost+graph[neighbor]['Heuristic']
            #Adding the node and their estiated distance in the visited queue
            visited_node.put((estimated_distance[neighbor],neighbor))
            #upadte parent node
            path[neighbor]=current_node
    return



graph={}
inp_file=open('Input file.txt','r')
lines=inp_file.readlines()
for line in lines:
    info=line.split()
    #print(f'{info}')
    city, heuristic,  neighbor_info= info[0],int(info[1]), {}

    #print(f'{city} {heuristic} {neighbor_info}')
    n_i=2 #neighbor info starts from index 2
    while n_i<len(info):
        neighbor_city=info[n_i]
        distance=float(info[n_i+1])
        neighbor_info[neighbor_city]=distance
        n_i+=2
    #print(f'{city} {heuristic} {neighbor_info}\n')
    graph[city]={'Heuristic':heuristic,'Neighbor':neighbor_info}
#print('\nGraph:\n',graph)
start_node=input('Start Node: ')
destination=input('Destination: ')
result=astar_algo(graph,start_node,destination)
if result==None:
    print('NO PATH FOUND')
else:
    ans='Path: '
    for i in result[1]:
        ans+=str(i)+' -> '
    ans=ans[:-4]+f'\nTotal Distance: {result[0]} km'
    print(ans)
```