Team Members: Devansh Sharma
Hetal Tiwari
Meet Turakhia

Problem statement -
Electric cars do not have as large a range as gas cars, so they need periodic recharge. Assume that one needs to travel a large distance, that cannot be done in one charge, so one needs to stop to recharge and continue. But some of the chargers may not function, in case you need to drive back to the previous city and re-charge there. Given a capacity C in miles that represents the maximum number of miles your electric car can drive, n cities and (n-1) distances between two consecutive cities, design an algorithm that outputs the list L of cities where one need to stop and charge the car such that:
L is if minimum length among all possible list of cities
the starting city, which is the first city, is the first element of L
The destination city, which is the last city, is the last element of L
If j and k are two consecutive cities in L, then when the car is in city j, the car is able to drive to city k and back to the city before city k, in case the charge station in city k is broken.
The capacity C in miles is not fixed, but one can assume that is a positive integer between 250 and 350. The number of cities n is not fixed, but you can assume that it is greater than 3 and less than 20. The distance between cities is a positive integer and always less than C/2 and greater than 10.

Solution -
The above problem has been solved using Python language. We first used linked list as the data structure for solving the problem. But upon further analysis we found that the space complexity of the problem can be further improved if we use lists instead of a linked list. Therefore, we again solved the problem using list data structure.

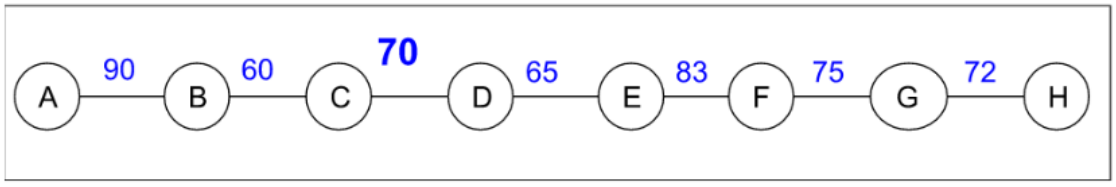
Pseudocode 1:
Data structure - Linked List

```
TravelGraph():
        city = current city/node
        next = next city/node
        previous = previous city/node
        nextCost = cost for travelling to next city
        prevCost = cost for travelling to previous city
carMaxCharge = Total charge capacity of car in miles
cityNo = Total number of cities/nodes
cityList = list of cities
costList = list of vertex costs
current = TravelGraph()
graphStartRef = current
```

```
for i->0 to cityNo:
        cityList.append("Enter the {i+1} city name")
        current.city = cityList[i]
        If i < cityNo -1:
                current.next = TravelGraph()
        If i > 0:
                costList.append("Enter cost of travelling from {cityList[i-1]} to {cityList[i]}: ")
                previous.nextCost = costList[i-1]
                current.previous = previous
                current.prevCost = costList[i-1]
        previous, current = current, current.next
        if i == 0:
                currentMap = f"{cityList[i]}"
        elif i < cityNo:
                currentMap = currentMap + f" <- {costList[i-1]} -> {cityList[i]}"
        print(currentMap)
current = current back to start of linked list
currentCharge = carMaxCharge #creating variable to keep track of current charge
stopsList = list of stops
stopsList.append(cityList[0])

while(current.city is not equal to cityList[-1]):
        If currentCharge >= current.nextCost *2
                currentCharge -= current.nextCost
                current = current.next
        Else:
                currentCharge = carMaxCharge
                stopsList.append(current.city)
                currentCharge -= current.nextCost
                current = current.next
stopsList.append(cityList[-1])
print("Output-> Here are the suggested stops: " + str(stopsList))
```

The time complexity for the above code is linear that is O(n) where n is the number of cities.

Input example 1:

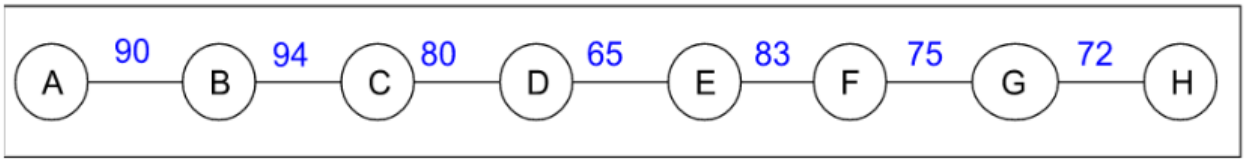


Expected output:
['A', 'D', 'G', 'H']

Achieved Output:

```
Enter the car mileage in one charge: 300
How many cities are there on the map: 8
Enter the 1 city name: A
A
Enter the 2 city name: B
Enter cost of travelling from A to B: 90
A <- 90 -> B
Enter the 3 city name: C
Enter cost of travelling from B to C: 60
A <- 90 -> B <- 60 -> C
Enter the 4 city name: D
Enter cost of travelling from C to D: 70
A <- 90 -> B <- 60 -> C <- 70 -> D
Enter the 5 city name: E
Enter cost of travelling from D to E: 65
A <- 90 -> B <- 60 -> C <- 70 -> D <- 65 -> E
Enter the 6 city name: F
Enter cost of travelling from E to F: 83
A <- 90 -> B <- 60 -> C <- 70 -> D <- 65 -> E <- 83 -> F
Enter the 7 city name: G
Enter cost of travelling from F to G: 75
A <- 90 -> B <- 60 -> C <- 70 -> D <- 65 -> E <- 83 -> F <- 75 -> G
Enter the 8 city name: H
Enter cost of travelling from G to H: 72
A <- 90 -> B <- 60 -> C <- 70 -> D <- 65 -> E <- 83 -> F <- 75 -> G <- 72 -> H
Output-> Here are the suggested stops: ['A', 'D', 'G', 'H']
```

Input example 2:

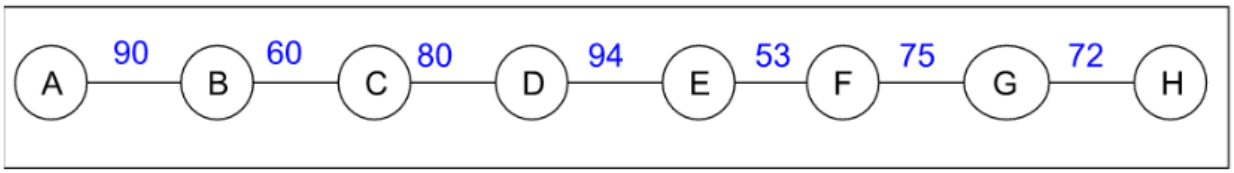


Expected Output:
['A', 'C', 'E', 'G', 'H']

Achieved Output:

```
Enter the car mileage in one charge: 300
How many cities are there on the map: 8
Enter the 1 city name: A
A
Enter the 2 city name: B
Enter cost of travelling from A to B: 90
A <- 90 -> B
Enter the 3 city name: C
Enter cost of travelling from B to C: 94
A <- 90 -> B <- 94 -> C
Enter the 4 city name: D
Enter cost of travelling from C to D: 80
A <- 90 -> B <- 94 -> C <- 80 -> D
Enter the 5 city name: E
Enter cost of travelling from D to E: 65
A <- 90 -> B <- 94 -> C <- 80 -> D <- 65 -> E
Enter the 6 city name: F
Enter cost of travelling from E to F: 83
A <- 90 -> B <- 94 -> C <- 80 -> D <- 65 -> E <- 83 -> F
Enter the 7 city name: G
Enter cost of travelling from F to G: 75
A <- 90 -> B <- 94 -> C <- 80 -> D <- 65 -> E <- 83 -> F <- 75 -> G
Enter the 8 city name: H
Enter cost of travelling from G to H: 72
A <- 90 -> B <- 94 -> C <- 80 -> D <- 65 -> E <- 83 -> F <- 75 -> G <- 72 -> H
Output-> Here are the suggested stops: ['A', 'C', 'E', 'G', 'H']
```

Input example 3:



Expected output:
[‘A’, ‘C’, ‘F’, ‘H’]

Achieved output:

```
Enter the car mileage in one charge: 300
How many cities are there on the map: 8
Enter the 1 city name: A
A
Enter the 2 city name: B
Enter cost of travelling from A to B: 90
A <- 90 -> B
Enter the 3 city name: C
Enter cost of travelling from B to C: 60
A <- 90 -> B <- 60 -> C
Enter the 4 city name: D
Enter cost of travelling from C to D: 80
A <- 90 -> B <- 60 -> C <- 80 -> D
Enter the 5 city name: E
Enter cost of travelling from D to E: 94
A <- 90 -> B <- 60 -> C <- 80 -> D <- 94 -> E
Enter the 6 city name: F
Enter cost of travelling from E to F: 53
A <- 90 -> B <- 60 -> C <- 80 -> D <- 94 -> E <- 53 -> F
Enter the 7 city name: G
Enter cost of travelling from F to G: 75
A <- 90 -> B <- 60 -> C <- 80 -> D <- 94 -> E <- 53 -> F <- 75 -> G
Enter the 8 city name: H
Enter cost of travelling from G to H: 72
A <- 90 -> B <- 60 -> C <- 80 -> D <- 94 -> E <- 53 -> F <- 75 -> G <- 72 -> H
Output-> Here are the suggested stops: ['A', 'C', 'F', 'H']
```

Pseudocode 2:

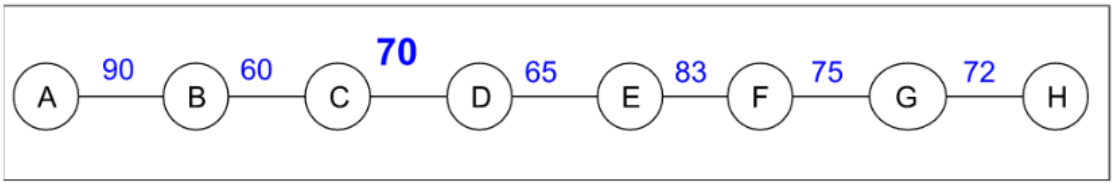
Data Structure - List. Here we have used two lists for the input and one list for the output.

```
maxCharge = max mileage that car gives per charge
cityList = list of cities/nodes
vertexCostList = Enter the list of costs
if len(vertexCostList) != len(cityList) - 1:
    print("Error: The number of vertices/costs should be equal to number of nodes - 1")
else:
    vertexCostList = [int(cost) for cost in vertexCostList]

currentCarCharge = maxCharge # variable to keep track of current car charge as it travels
stopsList + output list with suggested stops
stopsList.append(cityList[0])

for i in range(len(vertexCostList)):
    if currentCarCharge >= vertexCostList[i] * 2:
```

```
        currentCarCharge -= vertexCostList[i]
    Else:
        currentCarCharge = maxCharge
        currentCarCharge -= vertexCostList[i]
        stopsList.append(cityList[i])

stopsList.append(cityList[-1])
print("Output-> Here are the suggested stops: " + str(stopsList))
```

The time complexity for the above code is linear that is O(n) again, where n is the number of cities.
The space complexity is however optimized using list data structure.

Input example 1:

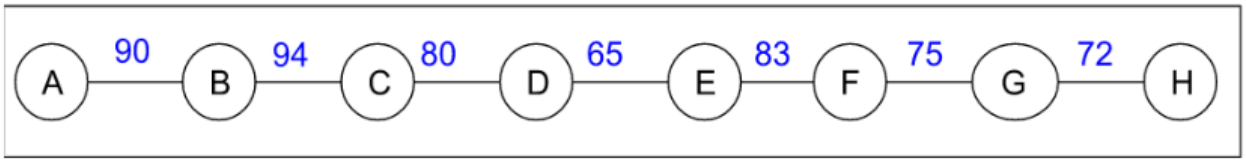


Expected output:
[‘A’, ‘D’, ‘G’, ‘H’]

Achieved output:

```
Maximum car mileage in one charge: 300
Enter the list of all the cities(enter with space in between): A B C D E F G H
Enter the list of costs from left to right respectively(enter with space in between): 90 60 70 65 83 75 72
Output-> Here are the suggested stops: ['A', 'D', 'G', 'H']
```

Input example 2:

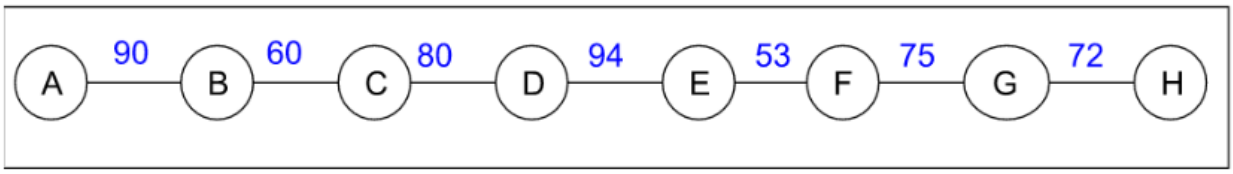


Expected output:
[‘A’, ‘C’, ‘E’, ‘G’, ‘H’]

Achieved output:

```
Maximum car mileage in one charge: 300
Enter the list of all the cities(enter with space in between): A B C D E F G H
Enter the list of costs from left to right respectively(enter with space in between): 90 94 80 65 83 75 72
Output-> Here are the suggested stops: ['A', 'C', 'E', 'G', 'H']
```

Input example 3:



Expected output:
[‘A’, ‘C’, ‘F’, ‘H’]

Achieved output:

```
Maximum car mileage in one charge: 300
Enter the list of all the cities(enter with space in between): A B C D E F G H
Enter the list of costs from left to right respectively(enter with space in between): 90 60 80 94 53 75 72
Output-> Here are the suggested stops: ['A', 'C', 'F', 'H']
```