CPSC 535 Advanced Algorithms

Project 1: Electric Car Traveller

Instructor: Prof. Doina Bein

Submission date: 14th October 2022

Priya Keshri
[885191452]
priyakeshri@csu.fullerton.edu

Shrinivas Pravin Patil
[885212043]
pshrinivas264@csu.fullerton.edu

Summary:

Python3 version 3.10 is used to implement the project. Install the Python compiler and editor as 'PyCharm'. Take input from the user for Cities as edges, distances as vertice and capacity as C.. Created a function min_stops() to compute the list of stops starting with A and ending in H such that the number of stops is minimised, in case the charge station in a stop city is broken, one can make it back to the previous city.

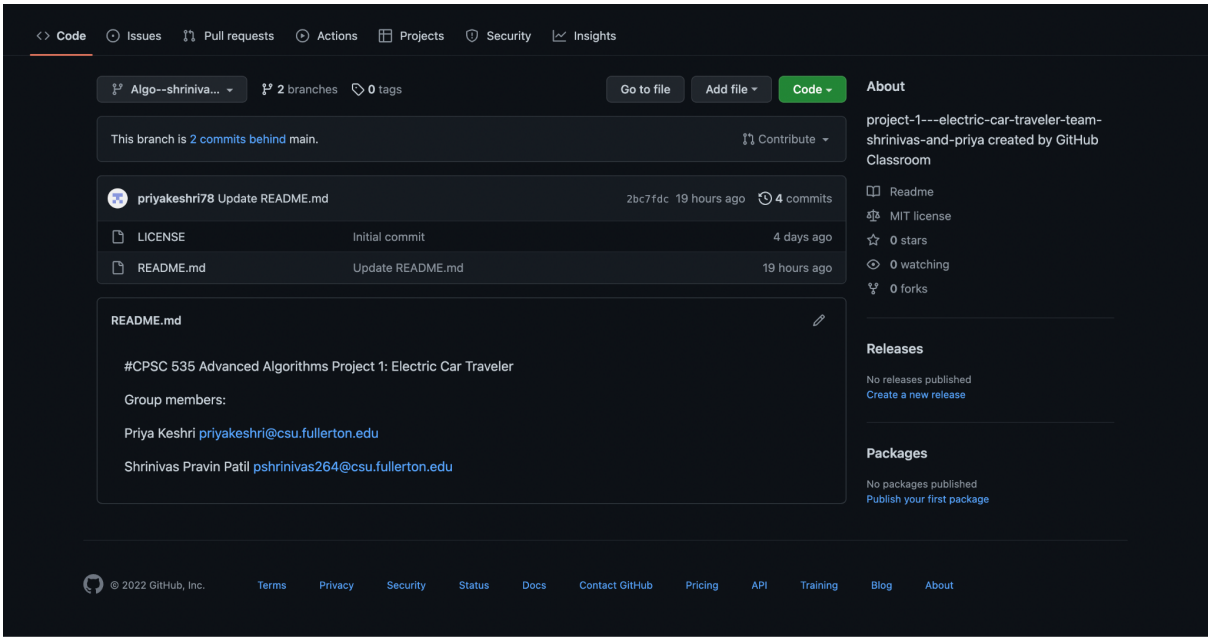
Pseudocode:

1) Take Input from the user for test cases.
2) Take vertices as input from the user and store it in list V.
3) Take edges as input from the user and store it in list E.
4) Add the first vertex in the resulting list of cities.
5) Set the first distance to the dist variable.
6) Initialise i to 1.
7) While i until length of E do,
   If dist+2*E[i]<=C
      dist += E[i]
      i += 1
   Else
      Append V[i] to cities
      dist = 0
8) Add the last vertex in the list of cities.
9) Return the list of cities.

Code:

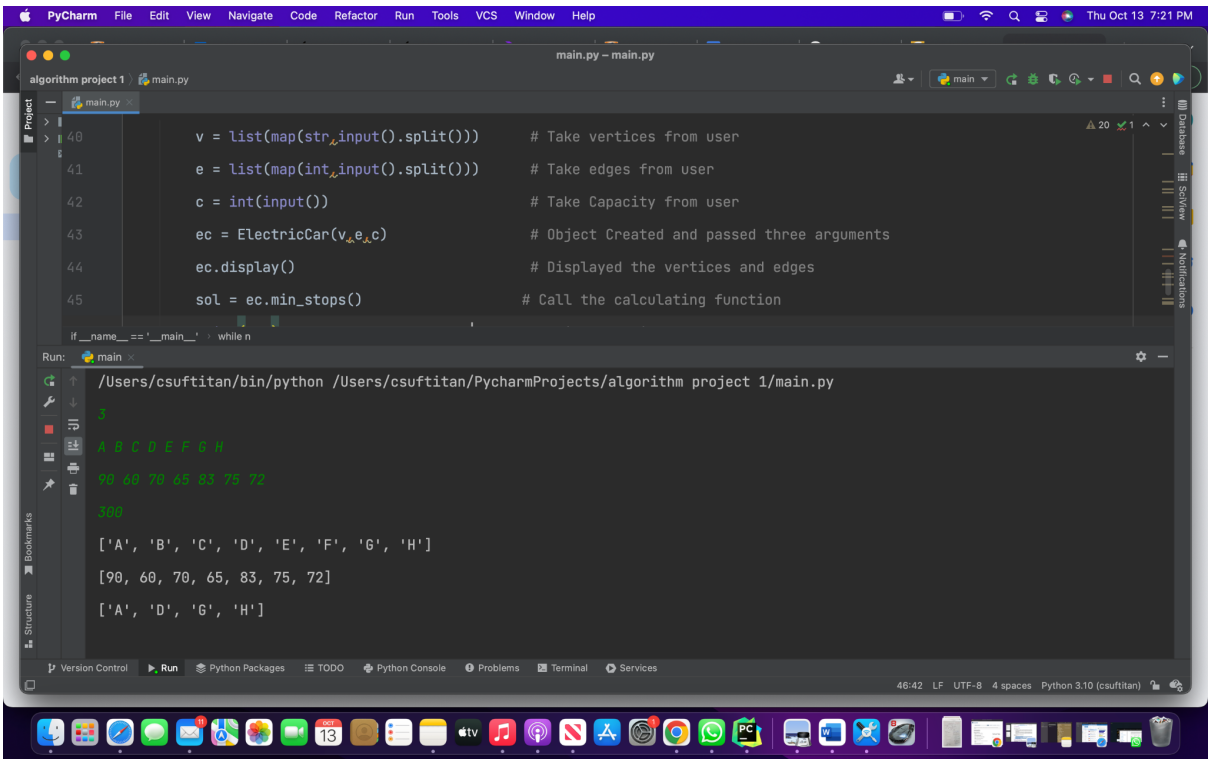
```
class ElectricCar:                                  # Create a class name ElectricCar
   def __init__(self,v,e,c):                        # Initialise input variables using Constructor
      self.V = v
      self.E = e
      self.C = c

   def min_stops(self):                  # Function for calculating to minimise the no.of stops
      cities = []
      for i in range(len(self.E)):
         if self.E[i]==0:
            return "Distance Cannot be Zero"
         elif self.C<250 or self.C>350:
            return "Capacity Should be Considered in between 250 to 300."
         elif self.E[i]>self.C:
            return "Car will break down while going from {} to {}".format(self.V[i],self.V[i+1])
         elif self.E[i]<10 or self.E[i]>self.c//2:
            return "Distance should be between 10 to C/2."

      cities.append(self.V[0])
      dist = self.E[0]
```

```
        i = 1
        while i<len(self.E):
            if self.E[i]<300:                                   # Testing Corner Cases after each refill
                if (dist+2*self.E[i])<=self.C:
                    dist += self.E[i]
                    i+=1
                else:
                    cities.append(self.V[i])
                    dist = 0
                    self.C = c
            else:
                    print("Car will break down while going from {} to {}".format(self.V[i],self.V[i+1]))

        cities.append(self.V[i])
        return cities

    def display(self):                                          # Function to display vertices and edges
        print(self.V)
        print(self.E)

if __name__ == '__main__':
    n = int(input())                                            # Take total number of test cases
    while n:
        v = list(map(str,input().split()))                      # Take vertices from user
        e = list(map(int,input().split()))                      # Take edges from user
        c = int(input())                                        # Take Capacity from user
        ec = ElectricCar(v,e,c)                                 # Object Created and passed three arguments
        ec.display()                                            # Displayed the vertices and edges
        sol = ec.min_stops()                                    # Call the calculating function
        print(sol)                                              # print solution
        n-=1                                                    # Decrement the n
```



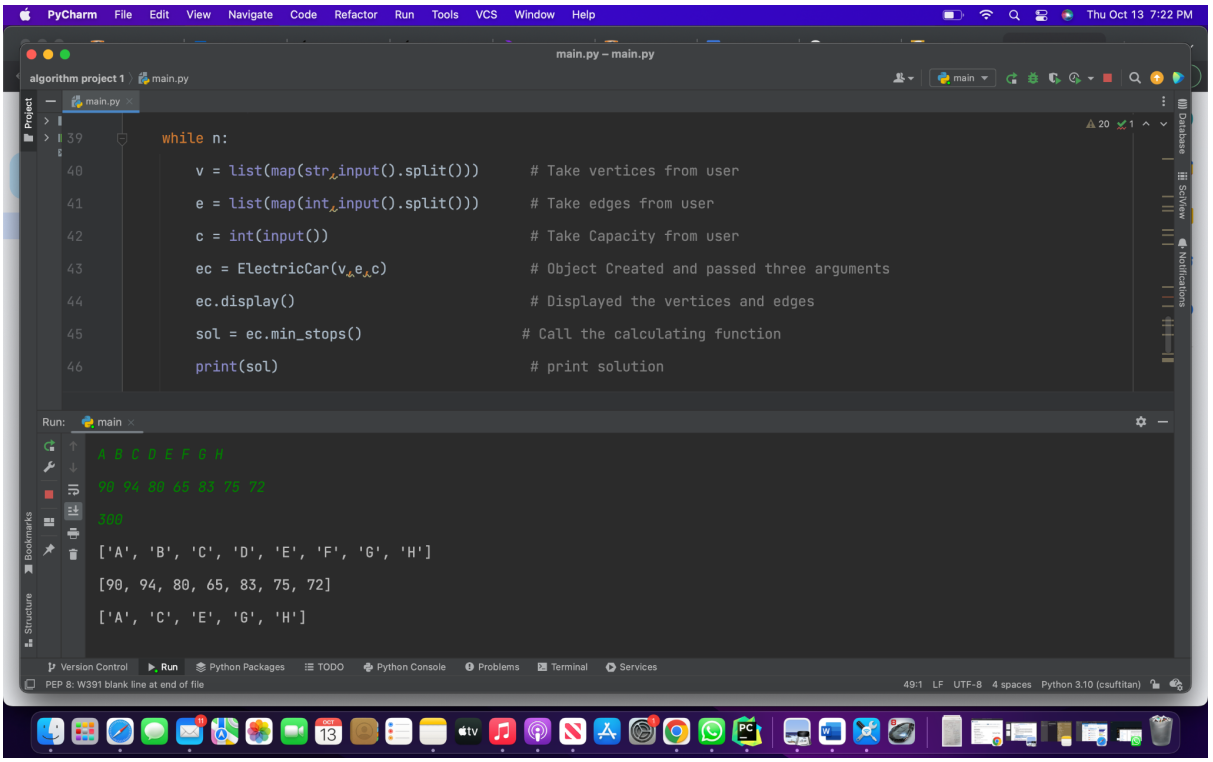
Test Case 1:

Input : A,B,C,D,E,F,G,H
Capacity: 300
Output: A,D,G,H



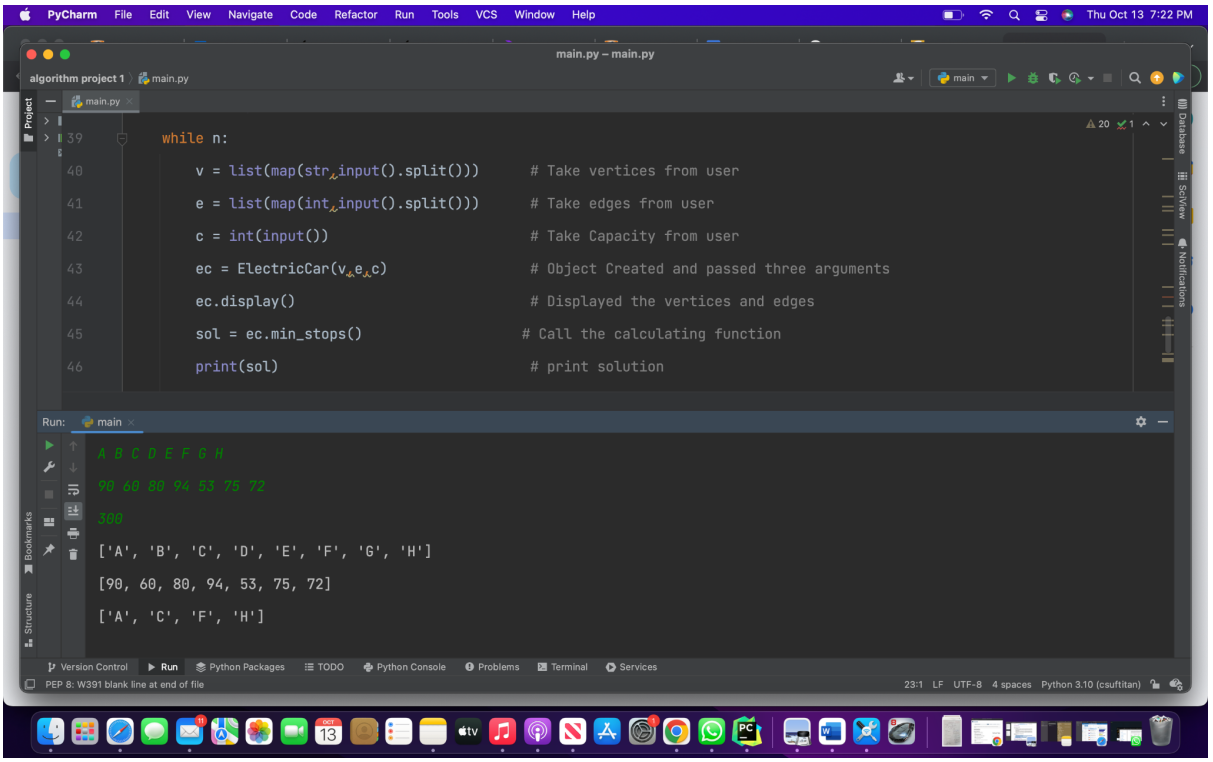
Test Case 2:

Input: A,B,C,D,E,F,G,H
Capacity: 300
Output: A,C,E,G,H



Test Case 3:

Input: A,B,C,D,E,F,G,H
Capacity: 300
Output: A,C,F,H



Time Complexity: O(n)

Reason: In this algorithm, whenever we call the min_stops(), the while loop will run for n-2 times in the worst case where n is the number of cities.

Space Complexity: O(n)

Reason: In this algorithm worst case space complexity will be n because all the cities should be traversed.

Corner Cases:

1) If the user gives input distance as 0 then the program should return distance cannot be zero.
2) If the input distance between city1 and city2 is greater than the actual capacity of the car then it will break down while going from city1 to city2.
3) If the input is not as per constraint then the program will not execute the calculating function.
4) If all the cities are equidistant means distance among all cities is equal.