Project 1

Spring 2023 CPSC 335 - Algorithm Engineering

Due: 02/26
Points: 100

Algorithm 1: Greedy Approach to Hamilton Problem

You are given a set of cities that are laid out in a circle, connected by a circular road that runs clockwise. Each city has a gas station that provides gallons of fuel, and the distances between these cities are known. You have a car that can drive some number of miles per gallon of fuel, and your goal is to pick a **starting city** such that you can fill up your car in that city (using that city's gas station). You can then drive to the next city, refill up your car with that city's fuel, drive to the next city, and so on and so forth until you return back to the starting city with 0 or more gallons of fuel left. 0This city is called a preferred starting city. In this problem, it is guaranteed that there will always be exactly one valid starting City

The problem set involves series of arrays. The first array is the distance between neighboring cities. Assume that city i is $distances[i]$ away from city $i + 1$. Since the cities are connected via a circular road, the last city is connected to the first city. In other words, the last distance in the distances array is equal to the distance from the last city to the first city. The second array is an array of gas available at each city, where $gas\ [i]$ is equal to the gas available at $city\ i$. The total amount of gas available (from all gas stations) is enough to travel to all cities. Your gas tank always starts out empty, and a positive integer value for the number of miles that your car can travel per gallon of fuel (miles per gallon, or MPG) is also given.

Write a function that returns the index of the preferred starting city.

Sample Input:

city_distances = [5, 25, 15, 10, 15]
fuel = [1, 2, 1, 0, 3]
mpg = 10

The preferred starting city for the sample above is city 4

Algorithm 2: Matching Group Schedules

The group schedule matching problem takes two or more arrays as input. The arrays represent availabilities and daily working periods of group members. The output is an array containing intervals of time when all members are available for a meeting. The group schedule problem can be defined as shown below:

Group Schedule Problem
input: arrays m of related elements, comprising the time intervals, and a HashMap, d, representing a daily active periods of all members. . U is a global set of all arrays. The problem can be represented as: $$U = \sum_{i=1}^{n} m_i$$ **output:** a set of HashMap, r, such that r⊆ U

Assume there are at least two persons in your class project group. You want to schedule a meeting with another group member. You are provided with (a) a schedule of members' daily activities, containing times of planned engagements. They are **not** available to have a meeting you during these periods; (b) the earliest and latest times at which they are available for meetings daily. Your schedule and availabilities are provided too.

Write an algorithm that takes in your schedule, your daily availability (earliest time, latest time) and that of your group member (or members), and the duration of the meeting you want to schedule. Time is given and should be returned in 24hr military format (HH:MM), for example: 9:30, 22:21. The given times (output) should be sorted in ascending order.

An algorithm for solving this problem may involve combing the two sub-arrays into an array containing a set of unavailabilities, with consideration of the daily active periods.

Sample input
person1_Schedule =[['7:00', '8:30'], ['12:00', '13:00'], ['16:00', '18:00']]
person1_DailyAct = ['9:00', '19:00']

person2_Schedule = [['9:00', '10:30'], ['12:20', '14:30'], ['14:00', '15:00'], ['16:00', '17:00']]
person2_DailyAct = ['9:00', '18: 30']
duration_of_meeting =30

Sample output
[['10:30', '12:00'], ['15:00', '16:00'], ['18:00', '18:30']]

To Do:

1. Develop a complete and clear pseudocode for the two algorithms

2. Mathematically analyze your pseudocode and prove the efficiency class, using either induction, limits or step counts. State the method used
3. Produce a readme file (can be in txt), describing how to execute your program.
4. Produce a brief written project report ***in PDF format***. Your report should include:
    a. Your name(s), CSUF-supplied email address(es), and an indication that the submission is for project 1.
5. Implement your algorithms in Python or C++. Your codes should be well commented, including your name(s).
6. Submit all files, including your PDF report and codes, separately.

Grading Rubric

The suggested grading rubric is given below:

Algorithm 1 a (45 points)

a. Clear and complete Pseudocode = 10 points
b. Mathematical analysis and correct Big *O* efficiency class = 5 points
c. Inclusion of a Readme file = 5 points
d. Well commented codes = 5 points
e. Successful compilation of codes= 15 points
f. Produces accurate results = 5 points

Algorithm (55 points each)

a. Clear and complete Pseudocode = 10 points
b. Mathematical analysis and correct Big *O* efficiency class = 5 points
c. Inclusion of a Readme file = 5 points
d. Well commented codes = 5 points
e. Successful compilation of codes= 20 points
f. Produces accurate results = 10 points

Submitting your code

Submit your files to the Project 2 Assignment on Canvas. It allows for multiple submissions. You may submit your files **separately**. Do not zip or use .rar.

Ensure your submissions are your own works. Be advised that your submissions may be checked for plagiarism using automated tools. Do not plagiarize. As stated in the syllabus, a submission

that involves academic dishonesty will receive a 0% score on that assignment. A repeat offense will result in an "F" in the class and the incident will be reported to the Office of Student Conduct.

Deadline

The project deadline is **Sunday, February 26, by 11:59 pm** on Canvas.

Penalty for late submission (within 48 hours) is as stated in the syllabus. Projects submitted more than 48 hours after the deadline will not be accepted.