Project 2: Implementing Algorithm

Spring 2024 CPSC 335 - Algorithm Engineering

Abstract

In this project, you will develop pseudocodes for some problems; analyze your pseudocode mathematically; design and implement codes for the algorithms; test your implementation; and describe your results.

Problem 1: Ensuring Convenient Schedules

The group schedule matching takes two or more arrays as input. The arrays represent slots that are already booked and the login/logout time of group members. It outputs an array containing intervals of time when all members are available for a meeting for a minimum duration expected.

Mathematically, the problem can be represented as:

Group Schedule Problem
input: arrays m of related elements comprising the time intervals and an array d, representing the 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

The group schedule matching takes the following inputs:

1. **Busy_Schedule:** An array list that represents the person's existing schedule (they can't plan any other engagement during these hours)
   Hint: Array may be 2D or maybe a list, ArrayList.
2. **Working_period**: Daily working periods of group members. (earliest time, latest time)
3. **Duration of the meeting** It outputs a list of list containing intervals of time when all members are available for a meeting for the minimum duration of the meeting required.

Analogy for the Problem:

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.

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’, ’13:30’], [’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’]]

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.

Problem 2: Arraylist and Subsets

The problem involves finding a contiguous part of an array of numbers that adds up to the maximum possible sum.

largest sum subarray problem
input: a non-empty vector V of n integers **output:** indices b, e such that $0 \leq b < e \leq n$, such the slice $V[b\colon e]$ has maximum sum

Recall that the notation $V[b\colon e]$ means the part of V starting at index $V[b]$ and up to, but not including, $V[e]$. Note that the solution$V[b\colon e]$ may not be empty due to the constraint $b < e$.

Sample input = (-3, -5, 5, -1, -3, 1, 5, -6)

Sample output = (5, -1, -3, 1, 5)

Exhaustive Search Approach

The following straightforward exhaustive search algorithm considers all pairs of indices b, e.

largest sum (V):
    b = 0, e = 1

```
for i from 0 to n-1:
        for j from i+1 to n:
                if sum(V[i:j]) > sum(V[b:e]):
                        b = i, e = j
return (b, e)
```

The Largest Sum Subarray Problem may be solved using any of the patterns we discussed in class. The exhaustive search pattern provides a straightforward approach that considers all pairs of indices b, e

Design an exhaustive search approach for solving the problem. You may read-in the sample input files below as a .txt file **or** ask user to enter a list. You may not hard-code the list into your algorithm.

Sample inputs = (10, 2, -5, 1, 9, 0, -4, 2, -2)
(-7, 1, 8, 2, -3, 1)
(9, 7, 2, 16, -22, 11)
(6,1, 9, -33, 7, 2, 9, 1, -3, 8, -2, 9, 12, -4)

To Do

1. Produce a 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 2.
2. Develop the pseudocode for Problem 1, and implement the **two** algorithms (Pseudocode for Part 2 is already given) in Python or C++
3. Your codes should be saved and submitted in the corresponding executable file.
4. Mathematically analyze each algorithm and state the big *O* efficiency class
5. Using the given sample inputs, print 4 resulting largest sum sub-arrays. Include them in your PDF report

Grading Rubric

The suggested grading rubric is given below:

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

Algorithm 2 (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

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, March 17, 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.