Project 1: Implementing Algorithms

Fall 2021 CPSC 335.05 - Algorithm Engineering

Instructor: Dr. Sampson Akwafuo (sakwafuo@fullerton.edu)

Abstract

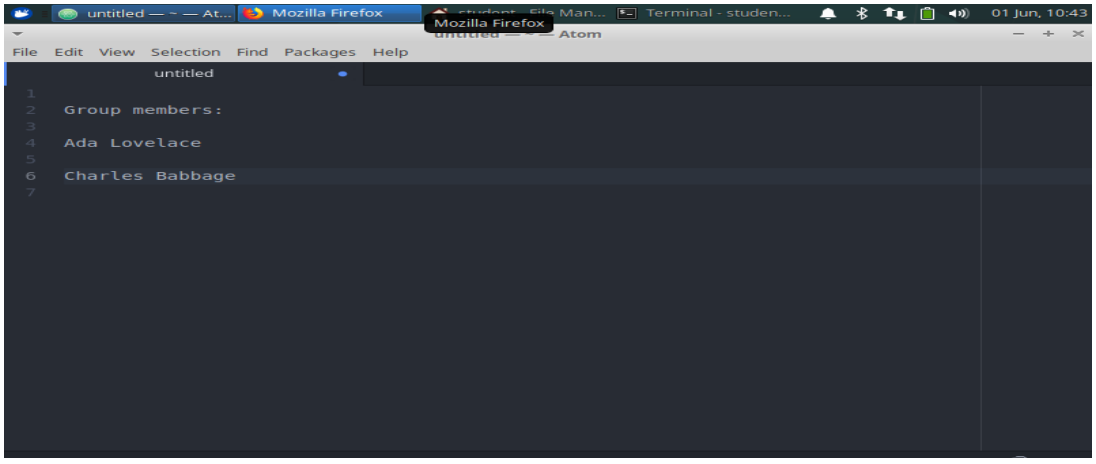
In this project, you will set up a departmental Tuffix environment for implementing algorithms and design an algorithm for solving the giving problem. The first step is for you to either create your own Tuffix Linux install, or arrange to use a Tuffix environment on campus. The second step is for you to develop a pseudocode for an algorithm; analyze your pseudocode mathematically; implement the algorithm in Python or C++; test your implementation; and describe your results.

Installing Tuffix

The first part of this project is installing Tuffix, either using a virtual machine or a bare-metal install. You should follow the instructions in the syllabus and tuffix installation page, and the documents they link to, to build a working Tuffix environment. As described in the syllabus, "Tuffix" is the Computer Science Department's official Linux development environment. Instructions on how to install Tuffix or a Tuffix based VM are online at «http://csufcs.com/tuffixinstall». The Tuffix Titanium Community for Students (here) is the best venue to find help with Tuffix.

Tuffix Deliverables

To prove that you completed the install, you will need to provide a screenshot. After you have Tuffix up and running, run the editor and write down the names of all your group member(s). If you are working alone, put your name only. If the names of your group members are Ada Lovelace and Charles babbage, your screenshot would look like this:



The Problem: 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. It outputs an array containing intervals of time when all members are available for a meeting.

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

Let us assume there are at least two persons in your class project group. You want to schedule a meeting with another group member. The members decide to provide you with (a) a schedule of their 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 military format. For example: 9:30, 22:21. The given times (output) should be sorted in ascending order.

An algorithm for solving this problem involves 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']]

Implementation

You are provided with the following files.

1. Project1_starter.py that defines functions for the algorithm described above. Some functions are incomplete skeletons; you will need to develop and write the functions and make them work properly. It is not mandatory to use these functions. You can implement your algorithm using a different approach. Describe how to run your program in the ReadMe file
2. Input.txt containing the sample input files. Use these sample files to run your program to see whether your algorithm implementations are working correctly.

To Do

1. Create a Readme file and include your name(s). The Readme file should also contain instructions on how to run your program.
2. Study the sample input and output above. Write your own complete and clear pseudocode for an algorithm to solve this problem.
3. Analyze your pseudocode for the algorithm mathematically and prove its efficiency class.
4. Implement your algorithm using either Python or C++.
5. Run your code using different data inputs

Finally, produce a brief written project report ***in PDF format***. Your report should include the following:

1. Your names, CSUF email address(es), and an indication that the submission is for project 1.
2. A Tuffix screenshot showing the atom editor (if atom is your editor, with your group member names inside.
3. A clear pseudocode of your algorithm
4. A brief proof argument for the time complexity of your algorithm.

Mathematical Analysis

Analyze your algorithm mathematically. You should prove a specific big-O efficiency class for the algorithm. The analysis should be routine, similar to the ones we have done in class and in the textbook. The algorithm's efficiency class will be one of $O(n)$, $O(n^2)$, $O(n^3)$, or $O(n^4)$.

Can we do better? What changes do you think can be made to your algorithm to increase its time complexity/efficiency? Will an increase in n change the complexity class? n is the number of persons in the group.

Grading Rubric

The suggested grading rubric is below.

1. Tuffix installation = 20 points
2. Algorithm design and implementation = 40 points, divided as follows:
    a. Clear and complete Pseudocode = 10 points
    b. Complete and clear README.md file = 3 points
    c. Successful compilation = 15 points
    d. Produces accurate result = 12 points

3. Analysis = 40 points, divided as follows
    a. Mathematical analysis and proof =17
    b. Report document presentation = 15 points
    c. Screenshot = 5 points
    d. Comments on possible improvements = 3

Ensure your submissions are your own works. Your submissions will be checked for similarities using a software.

Submitting your code

Submit your files to the Project 1 dropbox on Canvas. It allows for multiple submissions. You can submit your files as a zip folder or submit each file separately.

Deadline

The project deadline is Friday, September 24, 11:59 pm on Canvas. Penalty for late submission is as stated in the syllabus. Projects submitted more than 48 hours after the deadline will not be accepted.