Professor Balaji Sai Charan Jalukuru

CPSC 335-03 18174

Sep 22nd, 2024

Project 2

- **Team members:**

  - Patrick Chau - pchau10@csu.fullerton.edu
  - Le Do – doleminhtriet@csu.fullerton.edu
  - Andrew Ho - andrewhostudent@csu.fullerton.edu
  - Alexander Peras - alopp5570@csu.fullerton.edu

Github: https://github.com/doleminhtriet/cpsc335_project2

- **Algorithm:**

  - **Input**

person1_Schedule = [['09:00', '10:30'], ['12:15', '13:30'], ['15:00', '16:00']] person1_DailyAct = ['08:00', '18:00'] person2_Schedule = [['10:00', '11:30'], ['12:45', '13:15'], ['16:30', '17:30']] person2_DailyAct = ['09:00', '17:00'] duration_of_meeting = 45

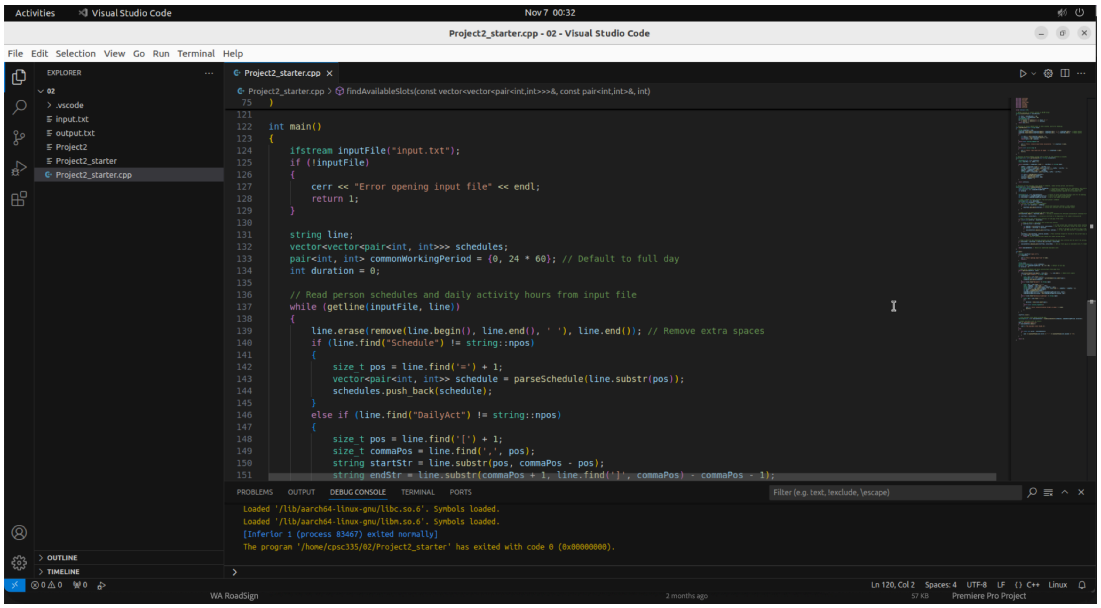
  - **Output**

Available Slots:

11:30 - 12:15

13:30 - 15:00

  - **Project Screenshot**



➢ **Algorithm**

```cpp
// Function to find available slots based on schedules, common working period, and duration
vector<pair<int, int>> findAvailableSlots(
    const vector<vector<pair<int, int>>> &schedules, // Collection of schedules for all individuals, each with a vector of busy intervals
    const pair<int, int> &commonWorkingPeriod,       // Common working hours interval (start and end time)
    int duration                                     // Minimum duration required for an available slot
)
{
    vector<pair<int, int>> availableSlots;        // Vector to store resulting available slots for the meeting
    int overallStart = commonWorkingPeriod.first; // Start of the common working period
    int overallEnd = commonWorkingPeriod.second;  // End of the common working period

    // Step 1: Gather all busy intervals from each person's schedule
    vector<pair<int, int>> busyTimes;
    for (const auto &schedule : schedules)
    { // Iterate over each individual's schedule
        for (const auto &interval : schedule)
        {                                 // Iterate over each busy interval in the schedule
            busyTimes.push_back(interval); // Collect all intervals into the busyTimes vector
        }
    }

    // Step 2: Sort all busy intervals by their start times
    sort(busyTimes.begin(), busyTimes.end()); // Sorting is necessary for efficient processing of intervals to find gaps

    int startTime = overallStart; // Initialize startTime to the beginning of the common working period
```

```cpp

    // Step 3: Iterate over sorted busy intervals to find gaps (free slots)
    for (const auto &interval : busyTimes)
    {
        // Check if there's a gap before the current busy interval
        if (interval.first > startTime)
        {                                                // If the current busy interval starts after startTime, there's a gap
            int endTime = min(interval.first, overallEnd); // End time for this gap is either the start of busy interval or overallEnd
            if (endTime - startTime >= duration)
            {                                                    // Check if the gap is at least as long as the required duration
                availableSlots.emplace_back(startTime, endTime); // If yes, add this interval to available slots
            }
        }
        startTime = max(startTime, interval.second); // Move startTime forward to the end of the current busy interval
        if (startTime >= overallEnd)
            break; // Stop if startTime goes beyond the common working period
    }

    // Step 4: Check for any remaining available slot between the last busy interval and the end of the working period
    if (overallEnd - startTime >= duration && startTime < overallEnd)
    {
        availableSlots.emplace_back(startTime, overallEnd); // Add any final gap as an available slot if it meets the duration requirement
    }

    return availableSlots; // Return all identified available slots
}
```

❖ **Pseudocode**

```
def findAvailableSlots(schedules, workingPeriods, duration):
    # Initialize an empty list to store available slots
    availableSlots = []
    # Step 1: Determine the overall working hours based on the latest login and earliest logout
    overallStart = latest start time among all working periods
    overallEnd = earliest end time among all working periods
    # If overall working hours are less than the meeting duration, return an empty list
    if overallEnd - overallStart < duration:
```

```
    return availableSlots
# Step 2: Gather all busy intervals from individual schedules
busyTimes = []
for schedule in schedules:
    for interval in schedule:
        busyTimes.append(interval)  # Add each busy interval to busyTimes
# Step 3: Sort all busy intervals by their start time
busyTimes.sort()
# Step 4: Find gaps between busy intervals that fit within the working hours and required duration
startTime = overallStart  # Start checking from the beginning of the working hours
for interval in busyTimes:
    # Check if there's a gap between the current interval and the start time
    if interval.start > startTime and interval.start <= overallEnd:
        endTime = min(interval.start, overallEnd)
        # If the gap is large enough for the required duration, add it as an available slot
        if endTime - startTime >= duration:
            availableSlots.append((startTime, endTime))
    # Move the start time forward to the end of the current interval
    startTime = max(startTime, interval.end)
    # Stop checking if we've reached the end of the working hours
    if startTime >= overallEnd:
        break
# Step 5: If there's a gap at the end of the busy intervals, add it as the final available slot
if overallEnd - startTime >= duration and startTime < overallEnd:
    availableSlots.append((startTime, overallEnd))
return availableSlots
```

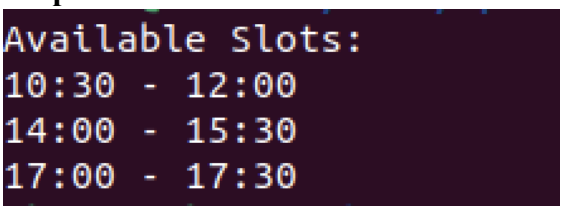
❖ **Complexity calculation**

Task	Complexity
Sorting busyTimes	O(n log n)
Loop through busyTimes intervals	O(n)
Final slot check	O(1)
Total complexity:	**O(n log n)+O(n)+O(1)=O(n log n)**

❖ **10 test cases**

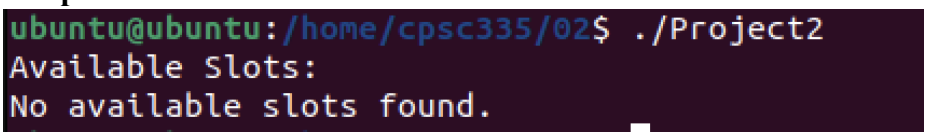
1. **Test Case 1: Basic Overlapping Times**

person1_Schedule = [['09:00', '10:30'], ['12:00', '13:30'], ['16:00', '17:00']]
person1_DailyAct = ['09:00', '18:00']
person2_Schedule = [['08:30', '09:30'], ['13:00', '14:00'], ['15:30', '16:30']]
person2_DailyAct = ['09:00', '17:30']
duration_of_meeting = 30
Output:



2. **Test Case 2: Full-Day Availability (Edge Case)**

Input:
person1_Schedule = [['00:00', '24:00']]
person1_DailyAct = ['00:00', '24:00']
person2_Schedule = [['00:00', '24:00']]
person2_DailyAct = ['00:00', '24:00']
duration_of_meeting = 15
Output:



3. **Test Case 3: No Meetings Scheduled, Only Working Hours**

Input:
person1_Schedule = []
person1_DailyAct = ['08:00', '17:00']
person2_Schedule = []
person2_DailyAct = ['09:00', '18:00']

duration_of_meeting = 60
Output:

```
ubuntu@ubuntu:/home/cpsc335/02$ ./Project2
Available Slots:
09:00 - 17:00
```

4. **Test Case 4: Multiple Overlapping Meetings with Different Start/End Times**

Input:
person1_Schedule = [['09:00', '10:30'], ['12:15', '13:30'], ['15:00', '16:00']]
person1_DailyAct = ['08:00', '18:00']
person2_Schedule = [['10:00', '11:30'], ['12:45', '13:15'], ['16:30', '17:30']]
person2_DailyAct = ['09:00', '17:00']
duration_of_meeting = 45
Output:

5. **Test Case 5: High Meeting Duration Requirement (Edge Case)**

Input:
person1_Schedule = [['09:00', '09:15'], ['10:00', '10:15'], ['11:00', '11:15']]
person1_DailyAct = ['08:00', '12:00']
person2_Schedule = [['08:30', '08:45'], ['09:30', '09:45'], ['10:30', '10:45']]
person2_DailyAct = ['08:00', '12:00']
duration_of_meeting = 120
Output:

```
ubuntu@ubuntu:/home/cpsc335/02$ ./Project2
Available Slots:
No available slots found.
```

6. **Test Case 6: Early and Late Working Hours with Gaps**

Input:
person1_Schedule = [['07:30', '08:00'], ['12:00', '12:30'], ['17:00', '17:30']]
person1_DailyAct = ['07:00', '18:00']
person2_Schedule = [['08:30', '09:00'], ['13:00', '13:30'], ['16:00', '16:30']]
person2_DailyAct = ['07:00', '18:00']
duration_of_meeting = 30
Output:

```
ubuntu@ubuntu:/home/cpsc335/02$ ./Project2
Available Slots:
07:00 - 07:30
08:00 - 08:30
09:00 - 12:00
12:30 - 13:00
13:30 - 16:00
16:30 - 17:00
17:30 - 18:00
```

7. **Test Case 7: Continuous Work Hours with Small Breaks**

   Input:
   person1_Schedule = [['08:00', '09:00'], ['10:30', '11:30'], ['13:30', '14:30']]
   person1_DailyAct = ['07:00', '18:00']
   person2_Schedule = [['08:30', '09:30'], ['11:00', '12:00'], ['14:00', '15:00']]
   person2_DailyAct = ['07:00', '18:00']
   duration_of_meeting = 30
   Output:

```
ubuntu@ubuntu:/home/cpsc335/02$ ./Project2
Available Slots:
07:00 - 08:00
09:30 - 10:30
12:00 - 13:30
15:00 - 18:00
```

8. **Test Case 8: Minimal Work Hours**

   Input:
   person1_Schedule = [['09:00', '10:00']]
   person1_DailyAct = ['09:00', '10:00']
   person2_Schedule = [['09:30', '10:30']]
   person2_DailyAct = ['09:30', '10:30']
   duration_of_meeting = 15
   Output:

```
ubuntu@ubuntu:/home/cpsc335/02$ ./Project2
Available Slots:
No available slots found.
```

9. **Test Case 9: Schedules with Large Breaks**

Input:
person1_Schedule = [['08:00', '09:00'], ['12:00', '13:00'], ['16:00', '17:00']]
person1_DailyAct = ['08:00', '18:00']
person2_Schedule = [['09:30', '10:30'], ['13:30', '14:30'], ['17:30', '18:30']]
person2_DailyAct = ['08:00', '18:00']
duration_of_meeting = 45
Output

```
ubuntu@ubuntu:/home/cpsc335/02$ ./Project2
Available Slots:
10:30 - 12:00
14:30 - 16:00
```

10. **Test Case 10: Tight Schedule with Small Meeting Windows**

Input:
person1_Schedule = [['09:00', '09:15'], ['10:00', '10:15'], ['11:00', '11:15']]
person1_DailyAct = ['08:00', '12:00']
person2_Schedule = [['08:30', '08:45'], ['09:30', '09:45'], ['10:30', '10:45']]
person2_DailyAct = ['08:00', '12:00']
duration_of_meeting = 10
Output:

```
ubuntu@ubuntu:/home/cpsc335/02$ ./Project2
Available Slots:
08:00 - 08:30
08:45 - 09:00
09:15 - 09:30
09:45 - 10:00
10:15 - 10:30
10:45 - 11:00
11:15 - 12:00
```