Balaji Sai Charan Jalukuru

CPSC 335-03 18174

Nov 6th, 2024

Project 2

Team Members:
Patrick Hung - twosquare@csu.fullerton.edu
Nhi Danis - ndanis@csu.fullerton.edu
Alexandru Gonzales - alexandrugonzales@csu.fullerton.edu
Johnny Nguyen - johnnynguyenha@csu.fullerton.edu

Github: https://github.com/TheFishEy/335project2

Algorithm:

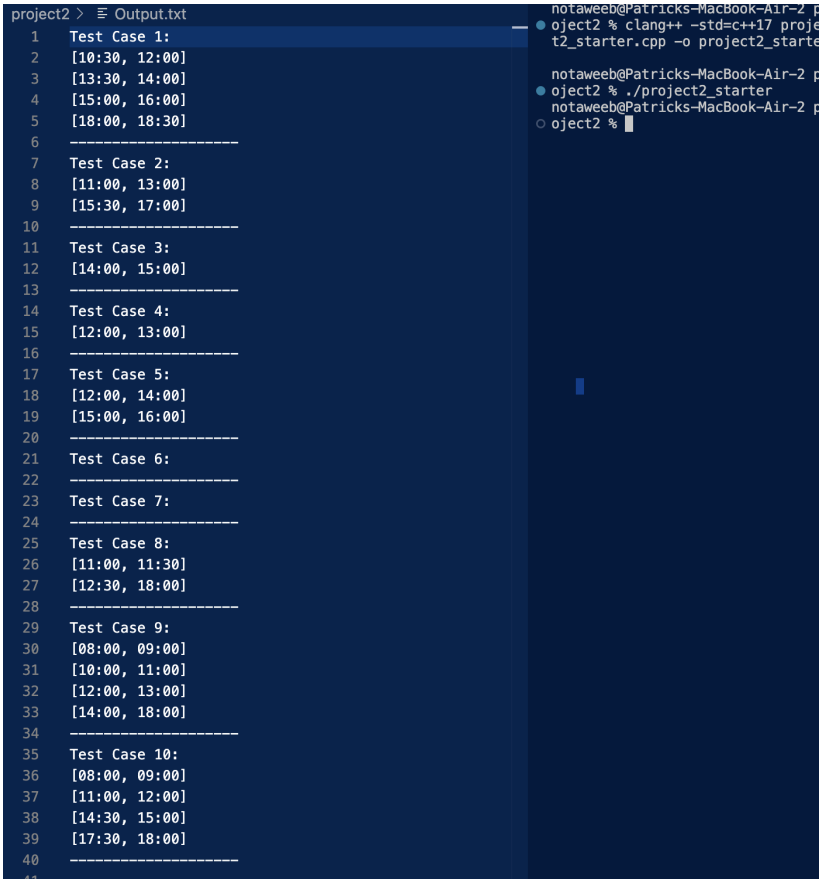
Input:

project2 > Input.txt

```
2
7:00-8:30,12:00-13:00,16:00-18:00
9:00-19:00
9:00-10:30,12:20-13:30,14:00-15:00,16:00-17:00
9:00-18:30
30

2
8:00-10:00,14:00-15:30
9:00-17:00
9:00-11:00,13:00-14:30
9:00-18:00
45

3
9:00-10:00,12:00-13:00,16:00-17:00
9:00-18:00
10:00-11:00,13:00-14:00
9:00-17:30
11:00-12:30,15:00-16:00
9:00-19:00
60

2
8:00-12:00,13:00-18:00
9:00-18:00
9:00-11:00,14:00-16:00
9:00-17:00
30

2
9:00-11:00,14:00-15:00,17:00-18:00
8:00-19:00
10:00-12:00,16:00-17:00
9:00-18:00
20

2
8:00-9:00,10:00-12:00,14:00-15:00
8:00-17:00
9:00-10:30,12:00-13:00,15:30-16:00
8:00-18:00
120

3
9:00-11:00,13:00-14:00
9:00-18:00
10:00-12:00,14:30-16:00
9:00-18:00
11:00-13:00,15:00-16:30
9:00-17:00
45
```

```
53
54  2
55  8:00-9:00,10:00-11:00
56  8:00-18:00
57  9:30-10:30,11:30-12:30
58  8:00-18:00
59  15
60
61  1
62  9:00-10:00,11:00-12:00,13:00-14:00
63  8:00-18:00
64  30
65
66  2
67  9:00-10:30,12:00-13:30,15:00-16:30
68  8:00-18:00
69  9:30-11:00,13:00-14:30,16:00-17:30
70  8:00-18:00
71  30
72
```

Output:



Code:

(in github link)

Pseudocode:

```
# function that converts time format to minutes
int convertTimeToMinutes(time):
   # extract hours and minutes from time string "HH:MM"
   hours = int from first two characters of time
   minutes = int from last two characters of time
   return hours * 60 + minutes

# function that converts minutes into proper format
string convertMinutesToTime(minutes):
   # convert minutes back to "HH:MM"
   hours = minutes // 60
   mins = minutes % 60
   return formatted string as "HH:MM"

# Function to read test cases and store them
Function readTestCase(file, schedules, workingHours, meetingDuration):
   # read until a line without comments or empty lines
   line = next non-comment, non-empty line from file
   if line is empty:
       return False

   try:
       numPeople = integer value of line
   except:
       print("Error: Invalid number format.")
       return False

   # clear previous data
   schedules.clear()
   workingHours.clear()
   # set up schedules and workingHours as lists of lists
   schedules.resize(numPeople)
   workingHours.resize(numPeople)

   for i from 0 to numPeople - 1:
       # read each person's schedule
```

```
        line = next non-empty line from file
        if line is empty:
            return False

        while intervals still in line:
            interval = substring before the first comma
            startTime, endTime = split interval by '-'
            schedules[i].append((convertTimeToMinutes(startTime),
convertTimeToMinutes(endTime)))
            remove the processed interval from line

        # read working hours for the current person
        line = next non-empty line from file
        activeStart, activeEnd = split line by '-'
        workingHours[i] = (convertTimeToMinutes(activeStart),
convertTimeToMinutes(activeEnd))

    # read meeting duration
    line = next non-empty line from file
    try:
        meetingDuration = integer value of line
    except:
        print("Error: Invalid meeting duration format.")
        return False

    return True

#function that finds the times that a free to meet
Function findFreeTimes(schedules, workingHours, meetingDuration):
    # combine all busy intervals
    busyTimes = []
    for personSchedule in schedules:
        add each interval in personSchedule to busyTimes

    # sort intervals by start time
    sort busyTimes by start time

    # merge overlapping intervals
    mergedBusyTimes = []
    for interval in busyTimes:
        if mergedBusyTimes is empty or last interval in mergedBusyTimes does not
overlap with interval:
```

```
            add interval to mergedBusyTimes
        else:
            extend the last interval in mergedBusyTimes to cover interval

    # find common working hours
    groupStart = max start in workingHours
    groupEnd = min end time in workingHours

    if groupStart >= groupEnd:
        return [] # no available common working time

    # find gaps between mergedBusyTimes within group working hours
    availableTimes = []
    start = groupStart
    for busy in mergedBusyTimes:
        if start + meetingDuration <= busy.start and start >= groupStart and busy.start
<= groupEnd:
            availableTimes.append((start, busy.start))
        start = max(start, busy.end)

    # check for time after last busy interval within working hours
    if start + meetingDuration <= groupEnd:
        availableTimes.append((start, groupEnd))

    return availableTimes
# function to write the results to a file
Function writeResults(file, availableTimes, testCaseNum):
    write "Test Case {testCaseNum}:" to file
    for start, end in availableTimes:
        write "[{convertMinutesToTime(start)}, {convertMinutesToTime(end)}]" to file
    write separator line to file
```

Time Complexity Analysis

Task	Complexity
convertTimeToMinutes(): Convert time string to minutes	Purpose: Converts a time string to the total number of minutes since midnight.

	Complexity: O(1) Explanation: This function performs a fixed number of operations regardless of the input size, so its runtime does not depend on the number of people or events.
convertMinutesToTime(): Convert minutes to time string	Purpose: Converts minutes into a formatted time string Complexity: O(1) Explanation: Similarly, this function performs a fixed number of operations, with runtime independent of the input size.
readTestCase(): read test case data from a file	Purpose: Reads the schedule and availability of each person from an input file and stores it in vectors. Complexity: $O(n \cdot m)$ Explanation: ● n is the number of people in the group. ● m is the average number of intervals per person's schedule. ● For each person, we parse their schedule, with each parsing operation

	taking constant time O(1) per interval, resulting in an $O(n \cdot m)$ complexity.
findFreeTimes(): find the free times	Purpose: Finds available time slots when all people are free. Complexity: $O(n \cdot m\log(n \cdot m)+n)$ Explanation: ● Step 1: Flattening busy times: All intervals are added into a single list and then sorted, taking $O(n \cdot m\log(n \cdot m))$ time. ● Step 2: Merging intervals: The sorted list is traversed to merge overlapping intervals, which takes $O(n \cdot m)$. ● Step 3: Finding free intervals: We identify free slots by comparing merged busy intervals, taking O(n) time for all intervals. ● The overall complexity for this function is dominated by the sorting step: $O(n \cdot m\log(n \cdot m))$.

writeResults(): write the results to file	Purpose: Writes available time slots to an output file. Complexity: O(k) Explanation: k is the number of available intervals found. Writing each interval requires a constant-time operation, resulting in a total complexity of O(k), which is typically much smaller than $O(n \cdot m)$.
Total complexity:	The algorithm's complexity is primarily dictated by the findFreeTimes() function due to its $O(n \cdot m\log(n \cdot m))$ sorting step. Thus, the total complexity is **$O(n \cdot m\log(n \cdot m))$.**

Improvement:

The main time-consuming part of this algorithm is sorting the intervals so we can merge them. Here are some ideas to improve it:

1. Optimize Merging Intervals: If we can keep the intervals sorted when we add them (instead of sorting afterward), we could skip the sorting step, making the process faster.

2. Use Better Data Structures: Using data structures like an interval tree or a sweep line method could help manage and merge intervals more efficiently. This could reduce the merging time from needing to sort the intervals first, making it closer to $O(n \cdot m)$ for large inputs.