UMT Software

-Technical test-

Problem summary

Calendar problem

Given the calendar booked time of two people find all available time they can meet.

Restrictions:

- each calendar will have limits, min and max range (ex: one maybe from 8:00 until 20:00, maybe one starting from 10:00 until 18:00)
- the range of all available (free) time they can meet will have to fit into the meeting required time (a variable input set to minutes)
- you must find all free time between the 2 calendars that is bigger or equal to the given meeting minutes time

Sample input:

- booked calendar1: [['9:00','10:30'], ['12:00','13:00'], ['16:00','18:00']]
- calendar1 range limits: ['9:00','20:00']
- booked calendar2: [['10:00','11:30'], ['12:30','14:30'], ['14:30','15:00], ['16:00','17:00']]
- calendar2 range limits: ['10:00','18:30']
- Meeting Time Minutes: 30

Sample output:

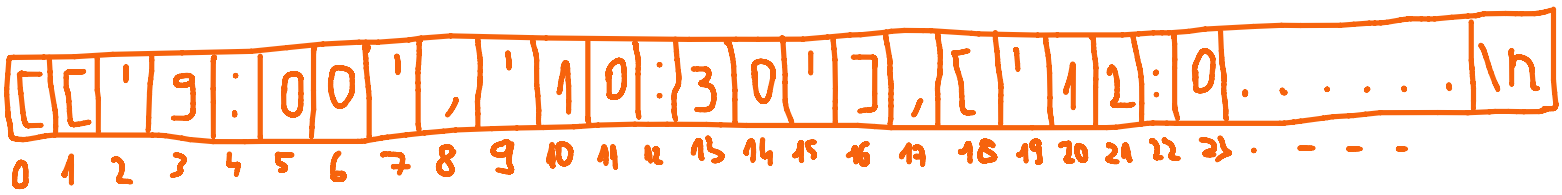
[['11:30','12:00'], ['15:00','16:00'], ['18:00','18:30']]

Solution | Description

My idea is to parse the given input strings and split them into multiple parts, each part representing some information that we need.

We will create a structure, called Intervals, that keeps all the data represented by an interval. (start hour, end hour, start minute and end minute, as well as some boolean variables, set_start and set_end, which denote the fact that we have set the start and the end time).

By instance, let's assume we take this string as input string for booked calendar 1: [['9:00','10:30'], ['12:00','13:00'], ['16:00','18:00]]. Representation:



Now, we will split this string in multiple intervals:

- [['9:00','10:30']
- ['12:00','13:00']
- ['16:00','18:00]

Each one of the above intervals can be separated in 2 parts: start time and end time. For example ['16:00','18:00] => start time = ['16:00'

end time = '18:00]

To achieve this we will use strtok function from stdlib library, our delimiter being "," and we will store all the substrings in a new variable. In fact, the input string was split as well into intervals, using this approach.

After we've done all of these, all the substrings that we generated should look like this:

- ⇨ [['9:00'
- ⇨ '10:30']
- ⇨ ['12:00'
- ⇨ '13:00']
- ⇨ ['16:00'
- ⇨ '18:00]]

We observe 2 patterns:

- ➔ The start times, starting with " [' "
- ➔ The end times, ending with "']"
- ➔ Two by two consecutive times are paired together/correspond.

Now we will use atoi() function to convert all these times into integers. We store them in variables of type Intervals.

These operations are done for both, booked calendar1 and booked calendar2. Somehow, the same applies for range limits, but we store these integers in a 4 dimensional array:

- ➔ Index 1: start_hour
- ➔ Index 2: start_minute
- ➔ Index 3: end_hour
- ➔ Index 4: end_minute

The next important step is to create an array of 60(minutes) * 24(hours) = 1440 elements, each element representing a particular minute within a day. All these elements will be set at the beginning with '0', meaning that these minutes can be used as meeting time. (the 2 persons can meet).

Now we traverse both booked calendars, and for each interval we set the given range time from the array to '1', meaning that at least one person is busy, therefore they can not meet.

At the end, we traverse again the array and check for sufficient big ranges of '0'. Each range must be at least equal to the metting_time to be printed out.

Functions

I used some basic functions like min, max, digits, etc. Besides these, some more complex functions are:

- ➔ int *get_limits: stores start hour, start minute, end hour and end minute in an array of 4 elements and returns it
- ➔ void set_minute_array: sets the array containing all minutes within a day with '0' or '1' depending on the schedules.
- ➔ int *print_meeting_intervals: return an array containing all the meeting intervals in which the persons can meet.
- ➔ void pretty_print_intervals: formats the intervals in a nice way and prints them out.

Testing

In the images below are shown some examples. We have 2 cases. The test case that was already provided to us and a new test case created by me.

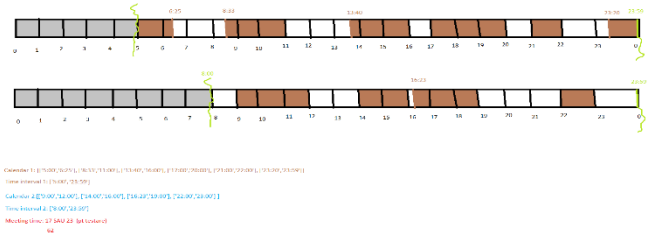


Fig. 0 – Test case created by me.

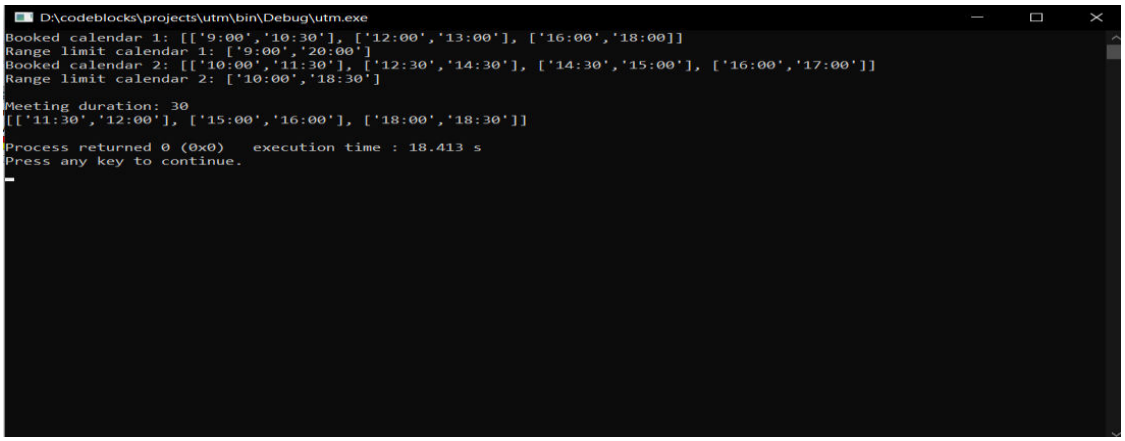


Fig 1. Solution to the provided test case (test case 1)

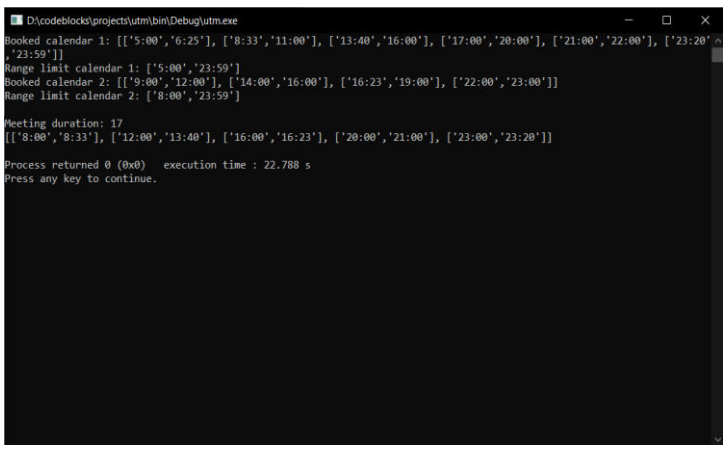


Fig. 2.1. Meeting duration = 17

Fig. 2.2. Meeting duration = 23

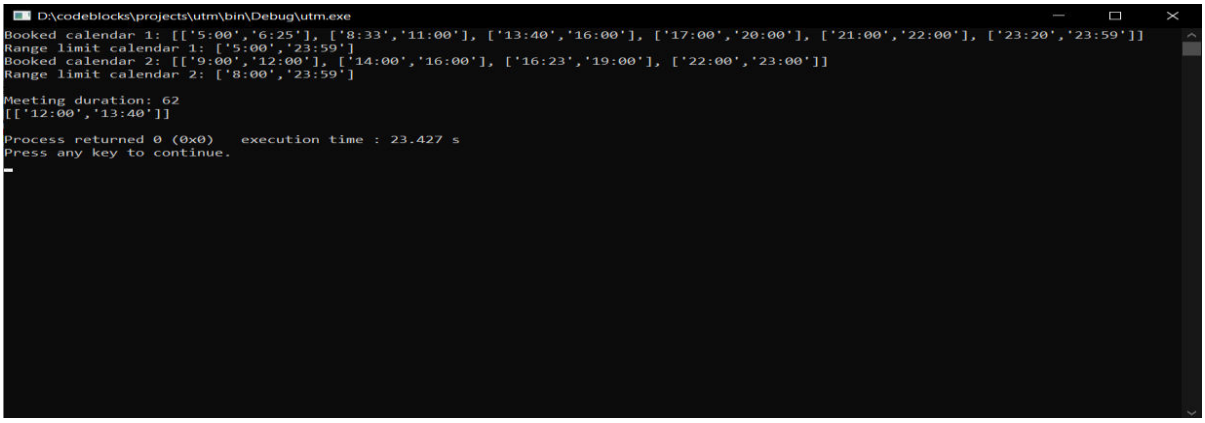


Fig. 2.3. Meeting duration = 62

This is my solution, not the best one nor the prettier, but it's doing it's job right. (at least for the examples that I have tested).