

Available Meeting Slot Finder

By: Ethan Paransky(eparansky@csu.fullerton.edu), Jing Nie(jingnie@csu.fullerton.edu),
Leonel Noriega-Rojas(leonelnr8@csu.fullerton.edu)

This Python program finds mutually available time slots for a meeting between two individuals based on their respective busy schedules and working hours.

Features

- **Convert Time Formats:** Converts time strings (e.g., "09:30") to minutes and vice versa for easier calculations.
- **Identify Unbusy Slots:** Determines free time slots between meetings for each individual within their specified working hours.
- **Find Common Slots:** Finds overlapping free time between two people that can accommodate a given meeting duration.
- **Filter by Working Periods:** Ensures available slots fall within each person's working hours.

How It Works

1. **Convert Schedules:** All time data (busy schedules and working periods) are converted from "HH" format to minutes.
2. **Calculate Free Slots:** For each person, free time slots are calculated by comparing busy times against their working hours.
3. **Find Common Availability:** Using a two-pointer approach, the code identifies overlapping free time slots between two people.
4. **Filter by Duration:** Ensures the common slots meet the required meeting duration.

Functions

- `time_to_minutes(time_str)`: Converts time from "HH" format to total minutes.
- `minutes_to_time(minutes)`: Converts minutes back to "HH" format.
- `find_unbusy_slots(busy_schedule, daily_act)`: Calculates free time slots for a person based on their busy schedule and daily working hours.
- `find_common_unbusy_slots(unbusy1, unbusy2, duration)`: Finds mutually free slots between two people that meet the required duration.
- `filter_by_working_period(common_slots, working_periods)`: Ensures the

free slots fall within specified working periods.

- **`find_available_slots(busy_schedules, working_periods, duration):`**
Combines all functions to find and return available meeting slots in "HH" format.

Usage

Run the `find_available_slots` function with the following parameters:

- `busy_schedules`: A list of busy schedules for each person (list of lists in "HH" format).
- `working_periods`: Each person's daily working period as a start and end time in "HH" format.
- `duration`: Desired meeting duration in minutes.

Analyze

Efficiency Class:

The primary efficiency gain comes from the two-pointer technique, which avoids nest loops for this case.

`find_unbusy_slots`:

The function loop through the busy schedule, which has a complexity of $O(n)$.

It calculates unbusy slots by processing each interval sequentially.

`find_common_unbusy_slots`:

The function uses a two-pointer technique, which means each element from `unbusy1` and `unbusy2` is processed once, which is $O(n)$.

`filter_by_working_period`:

This function compares each common slot with each working period, which is sequentially and $O(n)$.

find_available_slots:

This function combines the previous steps, so it's $O(n)$;

Pseudocode:

Set busy_schedules and working_periods as inputs

Call find_available_slots with params : busy_schedules, working_periods, duration

Step 1: Find unbusy slots for each person

Convert each time in busy_schedule and daily_act to minutes

Initialize unbusy_slots as an empty list

Set current_start to daily_start

FOR each start, end in busy_schedule:

IF current_start is before start:

Append [current_start, start] to unbusy_slots

Update current_start to max(current_start, end)

IF current_start is before daily_end:

Append [current_start, daily_end] to unbusy_slots

RETURN unbusy_slots

Using it find the unbusy slots for person1 and person2

Step 2: Find common unbusy slots between the first two schedules using
find_common_unbusy_slots

Initialize common_slots as an empty list

Set i, j to 0

WHILE i < length of unbusy1 AND j < length of unbusy2:

Set start1, end1 to unbusy1[i]

Set start2, end2 to unbusy2[j]

Calculate common_start as max(start1, start2)

Calculate common_end as min(end1, end2)

IF common_start is before common_end AND duration fits within this slot:

Append [common_start, common_end] to common_slots

Move pointer with earlier ending slot

RETURN common_slots

Step 3: Filter common slots based on working periods

Initialize filtered_slots as an empty list

FOR each start, end in common_slots:

FOR each work_start, work_end in working_periods:

Calculate common_start as max(start, work_start)

Calculate common_end as min(end, work_end)

IF common_start is before common_end:

Append [common_start, common_end] to filtered_slots

Remove duplicates from filtered_slots

RETURN unique_filtered_slots

Step 4: Convert available slots back to time format

Print the available slots