AlgoExpert   Quad Layout   Go   12px   Sublime   Monokai   00:00:00

Prompt   Scratchpad   Our Solution(s)   Video Explanation   Run Code

Solution 1

```go
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

package main

import (
  "fmt"
  "strconv"
  "strings"
)

type StringMeeting struct {
  Start string
  End   string
}

type Meeting struct {
  Start int
  End   int
}

// O(c1 + c2) time | O(c1 + c2) space - where c1 and c2 are the respective
// numbers of meetings in calendar1 and calendar2.
func CalendarMatching(
  calendar1 []StringMeeting, dailyBounds1 StringMeeting,
  calendar2 []StringMeeting, dailyBounds2 StringMeeting,
  meetingDuration int,
) []StringMeeting {
  updatedCalendar1 := updateCalendar(calendar1, dailyBounds1)
  updatedCalendar2 := updateCalendar(calendar2, dailyBounds2)
  mergedCalendar := mergeCalendars(updatedCalendar1, updatedCalendar2)
  flattenedCalendar := flattenCalendar(mergedCalendar)
  return getMatchingAvailabilities(flattenedCalendar, meetingDuration)
}

func updateCalendar(calendar []StringMeeting, dailyBounds StringMeeting) []Meeting {
  updatedCalendar := append([]StringMeeting{
    {Start: "0:00", End: dailyBounds.Start},
  }, calendar...)
  updatedCalendar = append(updatedCalendar, StringMeeting{
    Start: dailyBounds.End, End: "23:59",
  })

  meetings := []Meeting{}
  for _, i := range updatedCalendar {
    meetings = append(meetings, Meeting{
      Start: timeToMinutes(i.Start),
      End:   timeToMinutes(i.End),
    })
  }
  return meetings
}

func mergeCalendars(calendar1, calendar2 []Meeting) []Meeting {
  merged := []Meeting{}
  i, j := 0, 0
  for i < len(calendar1) && j < len(calendar2) {
    meeting1, meeting2 := calendar1[i], calendar2[j]
    if meeting1.Start < meeting2.Start {
      merged = append(merged, meeting1)
      i++
    } else {
      merged = append(merged, meeting2)
      j++
    }
  }

  for i < len(calendar1) {
    merged = append(merged, calendar1[i])
    i++
  }
  for j < len(calendar2) {
    merged = append(merged, calendar2[j])
    j++
  }
  return merged
}

func flattenCalendar(calendar []Meeting) []Meeting {
  flattened := []Meeting{calendar[0]}
  for i := 1; i < len(calendar); i++ {
    currentMeeting := calendar[i]
    previousMeeting := flattened[len(flattened)-1]
    if previousMeeting.End >= currentMeeting.Start {
      newPreviousMeeting := Meeting{
        Start: previousMeeting.Start,
        End:   max(previousMeeting.End, currentMeeting.End),
      }
      flattened[len(flattened)-1] = newPreviousMeeting
    } else {
      flattened = append(flattened, currentMeeting)
    }
  }
  return flattened
}

func getMatchingAvailabilities(calendar []Meeting, meetingDuration int) []StringMeeting {
  matchingAvailabilities := []StringMeeting{}
  for i := 1; i < len(calendar); i++ {
    start := calendar[i-1].End
    end := calendar[i].Start
    availabilityDuration := end - start
    if availabilityDuration >= meetingDuration {
      matchingAvailabilities = append(matchingAvailabilities, StringMeeting{
        Start: minutesToTime(start),
        End:   minutesToTime(end),
      })
    }
  }
  return matchingAvailabilities
}

func max(a, b int) int {
  if a > b {
    return a
  }
  return b
}

func timeToMinutes(time string) int {
  split := strings.SplitN(time, ":", 2)
  hours, _ := strconv.Atoi(split[0])
  minutes, _ := strconv.Atoi(split[1])
  return hours*60 + minutes
}

func minutesToTime(minutes int) string {
  hours, minutes := minutes/60, minutes%60
  return fmt.Sprintf("%d:%02d", hours, minutes)
}
```