539. Minimum Time Difference

Medium Array Math String Sorting

Problem Description

The task is to find the smallest difference in minutes between any pair of given time points where time points are represented in "HH:MM" (24-hour) format. The expectation is to calculate and return the minimum number of minutes that one would need to add or subtract to convert one time into another, considering all possible pairs within the list.

Intuition

To arrive at the intuitive solution, consider that time points are cyclical; 00:00 comes after 23:59 in a day. Hence, the times form a circular sequence. If we have the times as minutes past midnight, then the smallest difference is not necessarily between consecutive times in a 24-hour window but may span across midnight. Here's the step-by-step intuition:

1. **Convert to minutes**: Since we are to find differences in minutes, and times are given in hours and minutes, we convert everything into minutes to simplify calculations.
2. **Sort the times**: Sorting makes consecutive time points the nearest neighbors, saving us from checking each pair of time points. After sorting, the time points form a timeline from the earliest to the latest within a day.
3. **Deal with the 24-hour wraparound**: A crucial step is to handle the end-of-day wraparound when the difference between a late time and an early time spans over midnight (e.g. comparing 23:50 and 00:05). To manage this, we clone the first time point, add 24 hours (in minutes) to it, and append it to the sorted times; this simplifies our loop. The minimum difference can still be found in consecutive times because of our circular timeline.
4. **Find the minimum difference**: Loop through the sorted minutes and find the minimum difference between each adjacent pair, including the end-of-day wraparound we created.

This approach ensures that all comparisons needed to ascertain the minimum time difference are made, with the cyclical nature of time taken into account, using a single loop through the points.

Solution Approach

The solution provided here leverages a few straightforward concepts: sorting, modular arithmetic (to handle the circular nature of time), and greedy algorithms (to find the minimum value). The detailed solution approach is as follows:

1. **Upper Bound Check**: Before we begin any processing, we check if the list has more than `24 * 60` time points. If it does, it means there must be a duplicate time since there are only `24 * 60` minutes in a day. In this case, the minimum difference between any two time-points would be `0`, as two or more time points are the same.
2. **Convert Times to Minutes**: We convert each time-point from the `HH:MM` format to minutes since midnight using a list comprehension. This involves splitting each time string into hours and minutes, converting them to integers, and then to total minutes: `int(t[:2]) * 60 + int(t[3:])`.
3. **Sort Minutes**: Sorting the minute values helps to place the time points on a timeline from the smallest to the largest, making it easier to calculate consecutive differences.
4. **Handle Circular Case**: To address the wraparound at the end of the day (i.e., 00:00 is technically `0` minutes since midnight but follows 23:59 from the previous day), we append the smallest time (plus one full day, `24 * 60` minutes) to the end of our list.
5. **Initialize Result**: We set a variable `res` with a high initial value (using the last element which we artificially inflated in the previous step).
6. **Find Minimum Difference**: Loop through the list starting from index `1`, and calculate the difference between the current element and the previous element (`mins[i] - mins[i - 1]`). If the difference is smaller than the current `res`, update `res`. The process ensures we find the smallest difference.

By looping through only once and having our times sorted, we effectively use a greedy approach to find the minimum time difference without having to compare every single pair, which would be less efficient.

The `Solution` class provided represents an implementation of these steps, using Python's list and sorting capabilities to obtain the result efficiently.

```
class Solution:
    def findMinDifference(self, timePoints: List[str]) -> int:
        if len(timePoints) > 24 * 60:
            return 0
        mins = sorted(int(t[:2]) * 60 + int(t[3:]) for t in timePoints)
        mins.append(mins[0] + 24 * 60)
        res = mins[-1]
        for i in range(1, len(mins)):
            res = min(res, mins[i] - mins[i - 1])
        return res
```

In summary, the algorithm uses an upper bound check, conversion to a single unit (minutes), sorting, circular case handling, and greedy minimum difference calculation to solve the problem effectively.

Example Walkthrough

Let's illustrate the solution approach with a small example. Suppose we have the following list of time points:

```
["23:59", "00:00", "12:34", "22:00"]
```

Now, let's walk through the steps:

1. **Upper Bound Check**: The list has fewer than `24 * 60` time points, so we need to proceed with the rest of the calculation since there is no immediate indication of duplicate times.
2. **Convert Times to Minutes**: We convert each time point to the number of minutes since midnight.
   - "23:59" becomes `23 * 60 + 59 = 1439` minutes.
   - "00:00" becomes `0 * 60 + 0 = 0` minutes.
   - "12:34" becomes `12 * 60 + 34 = 754` minutes.
   - "22:00" becomes `22 * 60 + 0 = 1320` minutes.
3. **Sort Minutes**: We sort the minute values to get `[0, 754, 1320, 1439]`.
4. **Handle Circular Case**: We append the smallest time (plus one full day) to the end. So, `0 + 24 * 60 = 1440` minutes is added to the list, resulting in `[0, 754, 1320, 1439, 1440]`.
5. **Initialize Result**: We set `res` to a high initial value using the last element which was inflated in the previous step, so `res = 1440`.
6. **Find Minimum Difference**: We calculate the differences between consecutive elements.
   - Difference between `754` and `0` is `754 - 0 = 754`.
   - Difference between `1320` and `754` is `1320 - 754 = 566`.
   - Difference between `1439` and `1320` is `1439 - 1320 = 119`.
   - Difference between `1440` and `1439` is `1440 - 1439 = 1`.

The smallest difference is `1`, so we update `res` to `1`.

Following these steps, the solution class's method `findMinDifference` would return `1` as the smallest difference in minutes between any two given time points in this list.

Solution Implementation

Python

```
class Solution:
    def findMinDifference(self, time_points: List[str]) -> int:
        # If there are more time points than the total minutes in a day,
        # it means at least two time points are the same,
        # hence the minimum difference is 0.
        if len(time_points) > 24 * 60:
            return 0

        # Convert the time points to minutes and sort them.
        minutes = sorted(int(time[:2]) * 60 + int(time[3:]) for time in time_points)

        # Add the first element plus 24 hours to the end of the list
        # to calculate the circular difference.
        minutes.append(minutes[0] + 24 * 60)

        # Initialize the result with a maximum value for later comparison.
        min_difference = minutes[-1]

        # Calculate the minimum difference by iterating through the sorted minutes.
        for i in range(1, len(minutes)):
            min_difference = min(min_difference, minutes[i] - minutes[i - 1])

        # Return the minimum difference found.
        return min_difference
```

Java

```
class Solution {

    public int findMinDifference(List<String> timePoints) {
        // If there are more time points than minutes in a day, we have at least two identical times.
        if (timePoints.size() > 24 * 60) {
            return 0;
        }

        // Convert the list of time points into minutes since midnight.
        List<Integer> timePointsInMinutes = new ArrayList<>();
        for (String timePoint : timePoints) {
            String[] time = timePoint.split(":");
            int minutes = Integer.parseInt(time[0]) * 60 + Integer.parseInt(time[1]);
            timePointsInMinutes.add(minutes);
        }

        // Sort the list of time points in minutes.
        Collections.sort(timePointsInMinutes);

        // Add the first time point again to the end of the list converted to the next day,
        // to make it easier to calculate the min difference with the last time point.
        timePointsInMinutes.add(timePointsInMinutes.get(0) + 24 * 60);

        // Initialize the minimum difference to the maximum possible value (24 hours),
        // which will be updated in the following loop.
        int minDifference = 24 * 60;
        for (int i = 1; i < timePointsInMinutes.size(); ++i) {
            // Update the minimum difference if a smaller difference is found.
            minDifference = Math.min(minDifference, timePointsInMinutes.get(i) - timePointsInMinutes.get(i - 1));
        }

        // Return the minimum difference in minutes between any two time points.
        return minDifference;
    }
}
```

C++

```
class Solution {
public:
    int findMinDifference(vector<string>& timePoints) {
        // If there are more time points than minutes in a day, there must be a duplicate.
        // Since duplicate times would have 0 difference, return 0.
        const int totalMinutesInDay = 24 * 60;
        if (timePoints.size() > totalMinutesInDay) {
            return 0;
        }

        // Vector to hold the time points in minutes since midnight.
        vector<int> minutes;

        // Convert time points from string to minutes since midnight.
        for (const auto& time : timePoints) {
            int hour = stoi(time.substr(0, 2));
            int minute = stoi(time.substr(3));
            minutes.push_back(hour * 60 + minute);
        }

        // Sort the converted time points.
        sort(minutes.begin(), minutes.end());

        // Append the first time point plus 24 hours to handle the circular time comparison.
        minutes.push_back(minutes[0] + totalMinutesInDay);

        // Initialize the result with the maximum possible difference.
        int minDifference = totalMinutesInDay;

        // Loop through the sorted time points to find the minimum difference.
        for (int i = 1; i < minutes.size(); ++i) {
            // Compare each pair of adjacent time points.
            minDifference = min(minDifference, minutes[i] - minutes[i - 1]);
        }

        // Return the minimum time difference found.
        return minDifference;
    }
};
```

TypeScript

```
function findMinDifference(timePoints: string[]): number {
    // Convert each time point to minutes since the start of the day and sort the array
    const minutesArray = timePoints
        .map(time => {
            const hours = Number(time.slice(0, 2));
            const minutes = Number(time.slice(3, 5));
            return hours * 60 + minutes;
        })
        .sort((a, b) => a - b);

    const numTimePoints = minutesArray.length;
    let minimumDifference = Infinity;

    // Find the minimum difference between consecutive time points
    for (let i = 0; i < numTimePoints - 1; i++) {
        minimumDifference = Math.min(minimumDifference, minutesArray[i + 1] - minutesArray[i]);
    }

    // To account for the circular nature of the clock,
    // calculate the difference between the last and first time point across midnight
    const differenceAcrossMidnight = minutesArray[0] + 24 * 60 - minutesArray[numTimePoints - 1];
    minimumDifference = Math.min(minimumDifference, differenceAcrossMidnight);

    return minimumDifference;
}
```

```
class Solution:
    def findMinDifference(self, time_points: List[str]) -> int:
        # If there are more time points than the total minutes in a day,
        # it means at least two time points are the same,
        # hence the minimum difference is 0.
        if len(time_points) > 24 * 60:
            return 0

        # Convert the time points to minutes and sort them.
        minutes = sorted(int(time[:2]) * 60 + int(time[3:]) for time in time_points)

        # Add the first element plus 24 hours to the end of the list
        # to calculate the circular difference.
        minutes.append(minutes[0] + 24 * 60)

        # Initialize the result with a maximum value for later comparison.
        min_difference = minutes[-1]

        # Calculate the minimum difference by iterating through the sorted minutes.
        for i in range(1, len(minutes)):
            min_difference = min(min_difference, minutes[i] - minutes[i - 1])

        # Return the minimum difference found.
        return min_difference
```

Time and Space Complexity

Time Complexity

The given code comprises of several operations. Let's analyze them step-by-step:

1. **Checking Length of `timePoints`**: The operation `len(timePoints)` takes $O(n)$ time, where n is the number of elements in `timePoints`.
2. **Conversion and Sorting of `timePoints`**: The conversion takes linear time, $O(n)$ as each element is processed once. Sorting the converted times will take $O(n \log n)$ time where n is the length of the `timePoints`.
3. **Appending an Element**: Takes $O(1)$ time.
4. **Iterating over the Sorted `mins` List**: This loop runs in $O(n)$ time.

The dominant term here is $O(n \log n)$ due to the sorting step. Thus the overall time complexity is $O(n \log n)$.

Space Complexity

The space complexity includes the space required for the inputs and any additional space used by the algorithm to compute the final result:

1. **List `mins`**: Additional space is required to store the converted time points, which is $O(n)$.
2. **Constant Space**: Used by variables for calculating the minimum, such as `res` and the loop index `i`.

Hence, the space complexity of the code is $O(n)$ where n is the number of elements in `timePoints`.