2777. Date Range Generator

Medium

Leetcode Link

Problem Description

The problem requires us to create a range of dates that starts with a given "start" date and ends at an "end" date, including both start and end dates. The range should be generated using a given "step" value, which specifies the interval between consecutive dates in terms of days. We want to yield each date in this range as a string in the format "YYYY-MM-DD".

Intuition

To solve this problem, we use the JavaScript Date object, which enables us to work easily with dates. The procedure includes:

1. Parsing the provided "start" and "end" string dates into JavaScript Date objects, which allows us to manipulate dates with built-in methods.
2. Iterating from the start date to the end date, increasing the date by the step value every iteration. We achieve this by using the `getDate` and `setDate` methods that the Date object provides. The `getDate` method gets the day of the month from a Date object and `setDate` sets the day of the month to a specified number. By adding the step to the current day, we move our date forward by that many days.
3. During each iteration, we convert the current date to an ISO string using `toISOString`. ISO strings are in the format "YYYY-MM-DDTHH:MM:SS.ZZZZ". However, since we only need the date part without the time, we use `slice` to obtain the first 10 characters of the ISO string which represent the date in the required "YYYY-MM-DD" format.
4. Yielding the formatted date string using `yield`, which is part of creating a generator. Generators are special functions in JavaScript that can be exited and re-entered later with their context (variable bindings) being saved between re-entries.
5. Continuing this process until the current date exceeds the end date, at which point the generator finishes.

This solution effectively gives us a custom range of dates, and by using a generator, we can efficiently go through the range one date at a time without needing to precompute the entire range at once.

Solution Approach

The solution to this problem involves a step-by-step approach using a generator function to yield the desired range of dates one by one. Here's how the solution is implemented:

1. **Initialize Dates:** First, we need to convert the input start and end strings into `Date` objects using JavaScript's `new Date()` constructor. This allows us to perform date arithmetic and make use of Date object methods.

2. **Create Generator Function:** We then declare a generator function called `dateRangeGenerator` that takes `start`, `end`, and `step` as arguments. A generator function is defined with `function*` syntax and is capable of yielding values one at a time with the `yield` keyword.

3. **Iterate Through Dates:** Inside the generator function, we initiate a `while` loop that will continue as long as the `currentDate` is less than or equal to the `endDate`.

4. **Yield Current Date:** For each iteration within the loop, the current date (formatted as "YYYY-MM-DD" using `toISOString().slice(0, 10)`) is `yield`ed. This means that every time the generator's `next()` method is called, it will return an object with a value of the current date string and a done status indicating whether the generator has finished iterating.

5. **Increment Date:** To move to the next date in the range, we use `currentDate.getDate()` to get the day of the month for `currentDate`, add the `step` value to it, and update `currentDate` with this new value using `currentDate.setDate()`. This effectively increments `currentDate` by the `step` number of days.

6. **Finish Iteration:** As soon as `currentDate` exceeds `endDate`, the condition in the `while` loop becomes false, and the generator finishes its execution. Any further calls to `next()` after this point will return an object with a done status of true, indicating there are no more values to yield.

By utilizing a generator function and the JavaScript Date object, the solution elegantly traverses a range of dates and provides them on-demand, handling date arithmetic internally without the caller needing to manage the date range state. This allows for an efficient, on-the-fly generation of date strings that can be iterated over using the generator's `next()` method.

Example Walkthrough

Let's consider a small example to illustrate the solution approach. Suppose we want to create a range of dates from "2021-11-01" to "2021-11-05" with a step of 2 days. We will use the outlined solution approach to generate the desired date strings.

1. **Initialize Dates**: We convert the input strings "2021-11-01" and "2021-11-05" to JavaScript Date objects:

```
const startDate = new Date("2021-11-01");
const endDate = new Date("2021-11-05");
```

2. **Create Generator Function**: We create a generator function `dateRangeGenerator` which takes `startDate`, `endDate`, and `step` as parameters:

```
function* dateRangeGenerator(start, end, step) {
  // Generator function body will be implemented here
}
```

3. **Iterate Through Dates**: We initiate a loop inside the generator function, where we will generate dates from `startDate` to `endDate`. We use a while loop for this purpose:

```
let currentDate = new Date(start.getTime()); // avoid modifying the original start date

while (currentDate <= end) {
  // The body of the loop will yield dates and increment `currentDate`
}
```

4. **Yield Current Date**: During each iteration, we format the `currentDate` as a string and yield it:

```
yield currentDate.toISOString().slice(0, 10);
```

5. **Increment Date**: After yielding the date, we increment `currentDate` by the `step` value. In this example, we add 2 days:

```
const nextDay = currentDate.getDate() + step;
currentDate.setDate(nextDay); // This will increment `currentDate` by 2 days.
```

6. **Finish Iteration**: The loop continues until `currentDate` is greater than `endDate`. When that condition is met, the loop terminates, and the generator finishes:

```
// Loop has ended, so the generator is complete.
```

Putting it all together, our generator function will be used as follows:

```
const dateRange = dateRangeGenerator(startDate, endDate, 2);
console.log(dateRange.next().value); // "2021-11-01"
console.log(dateRange.next().value); // "2021-11-03"
console.log(dateRange.next().value); // "2021-11-05"
console.log(dateRange.next().done); // true, as no more dates are left
```

The output of each `console.log` illustrates how the generator yields each date string when `.next()` is called. After the last date, calling `.next()` indicates that the generator is finished by returning `{done: true}`.

Python Solution

```
from datetime import datetime, timedelta

# This generator function yields a range of dates, incrementing by a given step in days.
# Parameters:
# - start_date_str: The start date in ISO format (YYYY-MM-DD).
# - end_date_str: The end date in ISO format (YYYY-MM-DD).
# - step_days: The number of days to increment each time.
# It yields a string representing the current date in ISO format (YYYY-MM-DD) at each iteration.
def date_range_generator(start_date_str, end_date_str, step_days):
    # Convert the start and end date strings into datetime objects.
    start_date = datetime.fromisoformat(start_date_str)
    end_date = datetime.fromisoformat(end_date_str)
    # Initialize the current date to the start date.
    current_date = start_date

    # Continue yielding dates until the current date exceeds the end date.
    while current_date <= end_date:
        # Yield the current date as a string in ISO date format.
        yield current_date.strftime("%Y-%m-%d")
        # Increment the current date by the step value in days.
        current_date += timedelta(days=step_days)

# Usage example:
date_generator = date_range_generator('2023-04-01', '2023-04-04', 1)
for date in date_generator:
    print(date)  # Prints '2023-04-01', '2023-04-02', '2023-04-03', '2023-04-04'
```

Java Solution

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 * This Iterable class provides a range of dates, incrementing by a given step in days.
 * Parameters:
 * - startDate: The start date in ISO format (YYYY-MM-DD).
 * - endDate: The end date in ISO format (YYYY-MM-DD).
 * - step: The number of days to increment each time.
 * It yields a string representing the current date in ISO format (YYYY-MM-DD) on each iteration.
 */
public class DateRange implements Iterable<String> {

    private LocalDate startDate;
    private LocalDate endDate;
    private int step;

    public DateRange(String start, String end, int step) {
        this.startDate = LocalDate.parse(start);
        this.endDate = LocalDate.parse(end);
        this.step = step;
    }

    @Override
    public Iterator<String> iterator() {
        return new Iterator<String>() {

            private LocalDate currentDate = startDate;

            @Override
            public boolean hasNext() {
                // Check if the current date has not passed the end date
                return !currentDate.isAfter(endDate);
            }

            @Override
            public String next() {
                if (!hasNext()) {
                    throw new NoSuchElementException("No more dates to generate.");
                }
                // Store the current date as it should be returned
                LocalDate current = currentDate;
                // Increment the current date by the step value in days
                currentDate = currentDate.plusDays(step);
                // Return the current date as a string in ISO date format
                return current.toString();
            }
        };
    }
}

// Usage example:
public class Main {
    public static void main(String[] args) {
        DateRange dateRange = new DateRange("2023-04-01", "2023-04-04", 1);
        for (String date : dateRange) {
            System.out.println(date); // Outputs each date in the range
        }
    }
}
```

C++ Solution

```
#include <iostream>
#include <ctime>
#include <string>

// This class represents a generator that yields a range of dates, incrementing by a given step in days.
class DateRangeGenerator {
public:
    // Constructor for the generator with start, end dates and step value.
    // - startDate: The start date in ISO format (YYYY-MM-DD).
    // - endDate: The end date in ISO format (YYYY-MM-DD).
    // - stepDays: The number of days to increment each time.
    DateRangeGenerator(const std::string& startDate, const std::string& endDate, int stepDays)
        : endDate(convertToDate(endDate)), step(stepDays) {
        currentDate = convertToDate(startDate);
    }

    // Checks if there are more dates to generate.
    bool hasNext() const {
        return currentDate <= endDate;
    }

    // Returns the next date in the range, moving forward by the step value.
    std::string next() {
        if (!hasNext()) {
            return ""; // End of the range, no more dates to generate.
        }

        std::string currentDateString = convertToString(currentDate);
        incrementDate(currentDate, step); // Move to the next date.
        return currentDateString;
    }

private:
    // Converts ISO date string to tm structure.
    tm convertToDate(const std::string& isoDate) const {
        tm date = {};
        sscanf(isoDate.c_str(), "%d-%d-%d", &date.tm_year, &date.tm_mon, &date.tm_mday);
        date.tm_year -= 1900; // tm_year is years since 1900.
        date.tm_mon -= 1;     // tm_mon is 0-based.
        return date;
    }

    // Converts tm structure to ISO date string.
    std::string convertToString(const tm& date) const {
        char buffer[11]; // Buffer for "YYYY-MM-DD\0"
        // Add 1900 to tm_year to get the full year and 1 to tm_mon as it is 0-based.
        snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d", date.tm_year + 1900, date.tm_mon + 1, date.tm_mday);
        return std::string(buffer);
    }

    // Increments the date by given step value in days.
    void incrementDate(tm& date, int days) const {
        const time_t ONE_DAY = 24 * 60 * 60; // One day in seconds.
        // Convert tm structure to time_t.
        time_t date_time = mktime(&date);
        // Increment the date by the number of days converted to seconds.
        date_time += days * ONE_DAY;
        // Convert back to tm structure.
        date = *localtime(&date_time);
    }

    tm currentDate; // Current date to yield.
    tm endDate;     // End date for the range.
    int step;       // Increment step in days.
};

/*
// Usage example:
int main() {
    // Create a date range generator from April 1st to April 4th with a step of 1 day.
    DateRangeGenerator dateGenerator("2023-04-01", "2023-04-04", 1);

    // Iterate through the generated dates and print them.
    while (dateGenerator.hasNext()) {
        std::cout << dateGenerator.next() << std::endl;
    }

    return 0;
}
*/
```

Typescript Solution

```
// This generator function yields a range of dates, incrementing by a given step in days.
// Parameters:
// - start: The start date in ISO format (YYYY-MM-DD).
// - end: The end date in ISO format (YYYY-MM-DD).
// - step: The number of days to increment each time.
// It yields a string representing the current date in ISO format (YYYY-MM-DD) at each iteration.
function* dateRangeGenerator(start: string, end: string, step: number): Generator<string> {
    // Convert the start and end strings into Date objects.
    const startDate = new Date(start);
    const endDate = new Date(end);
    // Initialize the current date to the start date.
    let currentDate = startDate;

    // Continue yielding dates until the current date exceeds the end date.
    while (currentDate <= endDate) {
        // Yield the current date as a string in ISO date format.
        yield currentDate.toISOString().slice(0, 10);
        // Increment the current date by the step value in days.
        currentDate.setDate(currentDate.getDate() + step);
    }
}

/*
// Usage example:
const dateGenerator = dateRangeGenerator('2023-04-01', '2023-04-04', 1);
console.log(dateGenerator.next().value); // '2023-04-01'
console.log(dateGenerator.next().value); // '2023-04-02'
console.log(dateGenerator.next().value); // '2023-04-03'
console.log(dateGenerator.next().value); // '2023-04-04'
console.log(dateGenerator.next().done); // true (indicates that the generator is finished)
*/
```

Time and Space Complexity

The time complexity of the `dateRangeGenerator` function is O(N) where N is the number of dates generated between the start and end dates with the specified step. This is because the function generates each date in the range one by one in a loop, and the number of iterations of the loop equals the number of dates.

The space complexity of the function is O(1) since it yields the dates one by one and does not store all the dates in memory. The storage used does not grow with the size of the date range; the function only keeps track of the current date and the parameters provided.