2194. Cells in a Range on an Excel Sheet

Easy String

Leetcode Link

Problem Description

The problem provides a way to represent the positions of cells in an Excel sheet, using a combination of columns denoted by alphabetical letters and rows denoted by integers. Here, we are given a range in the format `<col1><row1>:<col2><row2>` which refers to a rectangle on the spreadsheet. The `<col1>` and `<col2>` are the start and end column letters, and `<row1>` and `<row2>` are the start and end row numbers, respectively. The task is to list all cells within this range inclusively, sorted first by columns and then by rows. Each cell should be represented by its respective column letter followed by its row number.

Intuition

To solve this problem intuitively, consider how you would manually list all cells if given a range on a real Excel sheet; you would likely start from the top left cell and move horizontally until you reach the end of a row, and then move down to the next row and repeat until the entire range is covered.

The intuition behind the solution is derived from this manual process. Here's how we translate the process into the solution approach:

1. Convert the start and end columns from letter to their corresponding ASCII values using `ord()`. This will allow us to iterate over the columns numerically.
2. Convert the start and end row numbers to integers to iterate over them numerically.
3. Use a nested loop to generate all combinations of column letters and row numbers within the specified range. The outer loop iterates over the ASCII values for columns, converting them back to letters using `chr()`, while the inner loop iterates over the row numbers.
4. During each iteration, concatenate the column letter and row number to form the cell representation, such as 'A1', 'A2', etc.
5. The loops are ordered to match the expected sorting of the result, by columns first and then by rows.

By using this approach, we cover all cells within the specified range systematically, and construct the list in the required sorted order.

Solution Approach

The solution implements a simple but effective approach that leverages Python's list comprehension and character manipulation functions. No advanced data structures are needed as the problem only requires generating a range of string identifiers for Excel cells. The Python code provided uses nested loops inside a list comprehension to generate the result directly.

Here's the step-by-step walk-through of the implementation:

1. **Extracting Column and Row Range Boundaries**: The given string `s`, representing the cell range in the format `<col1><row1>:<col2><row2>`, is dissected to get the starting and ending characters for columns (`s[0]` and `s[3]`, respectively) and integers for row numbers (`s[1]` and `s[4]`).
2. **Iterating Over Column Characters (Alphabetical Letters)**:
   - Convert the start column character `s[0]` and end column character `s[3]` to their ASCII values using `ord()`. This provides a numeric range that can be used in the loop.
   - A `for` loop is constructed to iterate over this numeric range using `for i in range(ord(s[0]), ord(s[3]) + 1)`. The `+ 1` is necessary to include the end column in the range.
3. **Iterating Over Row Numbers (Integers)**:
   - Similarly, the row numbers are parsed as integers.
   - A nested `for` loop is used to iterate over the rows for each column, with `for j in range(int(s[1]), int(s[4]) + 1)`, including the end row in the range.
4. **Generating Cell Identifiers**:
   - Inside the nested loop, the code concatenates the current column letter and row number to form the cell identifier. It uses `chr(i)` to convert the ASCII value back to a letter and `str(j)` to convert the row number to string.
   - The result of the concatenation `chr(i) + str(j)` is the cell identifier in the required format (e.g., 'A1').
5. **List Comprehension**:
   - The entire process takes place within a list comprehension, allowing for concise representation and direct output of the full list of strings representing the cell range.

The algorithm's time complexity is O(N*M), where N is the number of columns within the range and M is the number of rows in the range, as it iterates once for each cell in the rectangular area defined by the input string.

The solution can be stated succinctly as: Generate all combinations of columns and rows in the given range using nested iterations, and construct the cell identifier for each combination in the correct format.

Example Walkthrough

Let's use a small example to illustrate the solution approach. Suppose the range provided is `"B2:C3"`. This denotes a rectangle on the spreadsheet that starts at column B, row 2, and ends at column C, row 3.

Here's how the solution approach is applied to this example:

1. **Extracting Column and Row Range Boundaries**: We dissect the string `"B2:C3"` to find the starting and ending characters for columns and integers for row numbers. In this case, `col1` is `'B'`, `row1` is `'2'`, `col2` is `'C'`, and `row2` is `'3'`.
2. **Iterating Over Column Characters (Alphabetical Letters)**:
   - Convert `'B'` and `'C'` to their ASCII values using `ord()`, resulting in 66 for `'B'` and 67 for `'C'`.
   - Construct a `for` loop to iterate over the ASCII range: `for i in range(66, 68)` (68 because we include `'C'`).
3. **Iterating Over Row Numbers (Integers)**:
   - Convert `'2'` and `'3'` to integers, getting 2 and 3.
   - For each column letter in the outer loop, a nested `for` loop iterates over the row numbers: `for j in range(2, 4)` (4 because we include row 3).
4. **Generating Cell Identifiers**:
   - In the nested loop, for every combination of column and row, concatenate the column letter (converted back from ASCII) and row number to form the cell identifier. For example, `chr(66) + str(2)` becomes `'B2'`.
5. **List Comprehension**:
   - All these steps take place within a list comprehension: `[chr(i) + str(j) for i in range(66, 68) for j in range(2, 4)]`.

Executing this list comprehension, we should get the list of cell identifiers: `["B2", "B3", "C2", "C3"]`. These are all the cells included in the Excel range `"B2:C3"`, sorted by columns first and then by rows.

Python Solution

```python
from typing import List

class Solution:
    def cellsInRange(self, s: str) -> List[str]:
        # Initialize an empty list to hold the cell range.
        cell_range = []

        # Iterate over the character part of the range, from the start to the end character.
        for col_char in range(ord(s[0]), ord(s[3]) + 1):
            # Iterate over the numeric part of the range, from the start to the end number.
            for row_num in range(int(s[1]), int(s[4]) + 1):
                # Construct the cell label by combining the character and the number as a string,
                # and append each cell label to the cell range list.
                cell_range.append(chr(col_char) + str(row_num))

        # Return the list containing the range of cell labels.
        return cell_range

```

Java Solution

```java
import java.util.ArrayList;
import java.util.List;

class Solution {
    // Method to return a list of cell labels in the range specified by the input string s
    public List<String> cellsInRange(String s) {
        // List to store the answer
        List<String> cellRangeList = new ArrayList<>();

        // Start column character (e.g., 'A')
        char startColumn = s.charAt(0);
        // Start row character (e.g., '1')
        char startRow = s.charAt(1);
        // End column character (e.g., 'D')
        char endColumn = s.charAt(3);
        // End row character (e.g., '5')
        char endRow = s.charAt(4);

        // Loop from start column to end column
        for (char column = startColumn; column <= endColumn; ++column) {
            // Nested loop from start row to end row
            for (char row = startRow; row <= endRow; ++row) {
                // Add the cell label (e.g., 'A1') to the list
                cellRangeList.add("" + column + row);
            }
        }
        // Return the list containing all cell labels in the range
        return cellRangeList;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <string>

class Solution {
public:
    // Function to generate all cell names in the range specified by the input string
    std::vector<std::string> cellsInRange(const std::string& range) {
        // Initialize a vector to store the result
        std::vector<std::string> result;

        // Loop over each column character from the starting column to the ending column
        for (char col = range[0]; col <= range[3]; ++col) {
            // Loop over each row character from the starting row to the ending row
            for (char row = range[1]; row <= range[4]; ++row) {
                // Combine the current column and row characters to form a cell name
                std::string cellName = {col, row};

                // Add the generated cell name to the result vector
                result.push_back(cellName);
            }
        }

        // Return the generated list of cell names
        return result;
    }
};

```

Typescript Solution

```typescript
// Import statements are not necessary in TypeScript for such a simple snippet of code.

// Function to generate all cell names in the range specified by the input string
function cellsInRange(range: string): string[] {
    // Initialize an array to store the result
    let result: string[] = [];

    // Loop over each column character from the starting to the ending column
    for (let col = range.charCodeAt(0); col <= range.charCodeAt(3); ++col) {
        // Loop over each row character from the starting to the ending row
        for (let row = range.charCodeAt(1); row <= range.charCodeAt(4); ++row) {
            // Convert ASCII codes back to characters and combine them to form a cell name
            let cellName = String.fromCharCode(col) + String.fromCharCode(row);

            // Add the generated cell name to the result array
            result.push(cellName);
        }
    }

    // Return the generated list of cell names
    return result;
}

```

Time and Space Complexity

The given Python code is a one-liner list comprehension that generates a list of cell labels within a given range in a spreadsheet. The range is described by a string `s`, where `s[0]` and `s[1]` represent the column letter and row number of the top-left corner, and `s[-2]` and `s[-1]` represent the column letter and row number of the bottom-right corner, respectively.

The time complexity of the code can be analyzed based on the number of iterations the list comprehension performs. It iterates through all the columns from `s[0]` to `s[-2]` and, for each column, iterates through all the rows from `s[1]` to `s[-1]`. If `C` is the number of columns and `R` is the number of rows in the range, the total iterations will be `C * R`.

- The number of columns is `ord(s[-2]) - ord(s[0]) + 1`.
- The number of rows is `int(s[-1]) - int(s[1]) + 1`.

So, the time complexity is `O(C * R)`.

The space complexity of the code corresponds to the amount of space used to store the output list. Since the list comprehension generates a list with one item for every cell in the range, the space complexity is proportional to the number of cells in the output list, which is also `C * R`.

- The space complexity is also `O(C * R)`.

Hence, both the time complexity and space complexity of the code are `O(C * R)` where `C` is the number of columns spanned by the range and `R` is the number of rows spanned by the range.