Documentation for maximalRectangle Solution

Problem Description

The problem requires finding the largest rectangle containing only '1's in a given binary matrix (composed of '0's and '1's) and returning its area.

Solution Overview

The solution leverages a histogrambased approach to efficiently compute the largest rectangle of '1's for each row in the matrix.

maximalRectangle Method

This method computes the maximal rectangle in a binary matrix.

1. **Input Validation:** If the matrix is empty or the first row is empty, return 0.
2. **Initialize Variables:**
    - rows and cols to store the dimensions of the matrix.
    - heights, a list of zeros with length equal to the number of columns, to store the height of the histogram bars.
    - max_area to store the maximum area of the rectangle found so far.
3. **Iterate Through Each Row:**
    - For each element in the row, update the corresponding height in the heights array.
    - If the element is '1', increment the height by 1; otherwise, reset the height to 0.
4. **Calculate Max Area:** For each updated histogram, calculate the maximum area using the largestRectangleArea helper method.
5. **Return Result:** Return the maximum area found.

largestRectangleArea Method

This method calculates the largest rectangle area in a histogram using a stackbased approach.

1. **Append Zero:** Append a zero to the heights list to ensure the last bar is processed.
2. **Initialize Variables:**
    - stack to keep track of indices of the histogram bars.
    - max_area to store the maximum area of the rectangle found so far.
3. **Iterate Through Heights:**
    - For each bar, maintain a nondecreasing order in the stack.
    - If the current bar is lower than the bar at the index stored at the top of the stack, pop the stack and calculate the area.
    - The width of the rectangle is determined by the difference between the current index and the index of the new top of the stack.
4. **Pop Last Element:** Remove the appended zero to restore the original list.
5. **Return Result:** Return the maximum area found.

Example 1

matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

solution = Solution()

output = solution.maximalRectangle(matrix)

print(output) # Output: 6

Example 2

```
matrix = [["0"]]

solution = Solution()

output = solution.maximalRectangle(matrix)

print(output)  # Output: 0
```

Example 3

```
matrix = [["1"]]

solution = Solution()

output = solution.maximalRectangle(matrix)

print(output)  # Output: 1
```

Constraints

- The number of rows (rows) is between 1 and 200.
- The number of columns (cols) is between 1 and 200.
- Each element in the matrix is either '0' or '1'.

This solution effectively transforms each row into a histogram and uses a stackbased method to find the largest rectangle in each histogram, achieving efficient computation.