1536. Minimum Swaps to Arrange a Binary Grid

Leetcode Link

Problem Explanation

Given a square grid with rows and columns of equal `n` length, we need to make all cells above the main diagonal to `0`, or find out if it's not possible. We only have one operation that we can perform: swap two adjacent rows.

A valid grid is where all cells above the main diagonal are `0`. The main diagonal of a square grid starts from (1,1) up to (n,n). We should return the minimum number of adjacent swaps to make the grid valid, or if it's impossible, return `-1`.

For instance, given a grid `[[0,0,1],[1,1,0],[1,0,0]]`, we can make three swaps: Swap row 2 and 3, row 2 and 1, and then again row 2 and 3, such that all cells above the main diagonal become `0`. So, the output for this case is `3`.

Approach

To make the grid valid, each row `i` in the grid must have at least (n-i) zeros as appendages at the end of the row. We can count the number of trailing zeros for each row and store it in `suffixZeros`. Then, for each row `i` from the top to bottom, we find the first row `j` that has no less trailing zeros than (n-i), and move row `j` to position `i` using the adjacent row swap operation. The total number of operations would be our result.

Illustration

Let's take an example for better understanding:

Input: grid = `[[0,0,1],[1,1,0],[1,0,0]]`

SuffixZeros: `[1, 2, 3]`

We get neededZeros = n - i - 1 = 2 - 0 - 1 = 1. We found that the first row `j = 0` (0-indexed) has 1 trailing zero, so we move it to position `i = 0`, which takes `j - i = 0 - 0 = 0` operations.

In the second round, we get neededZeros = n - i - 1 = 2 - 1 - 1 = 0. We found that the second row `j = 1` has 2 trailing zeros, which is not less than 0, so we move it to position `i = 1`, which takes `j - i = 1 - 1 = 0` operations.

Adding up all the operations, we got the final result `0 + 0 = 0`.

C++ Solution

```cpp
cpp
class Solution {
public:
  int minSwaps(vector<vector<int>>& grid) {
    const int n = grid.size();
    int ans = 0;
    vector<int> suffixZeros;

    for (const vector<int>& row : grid) {
      auto itLastOne = find(rbegin(row), rend(row), 1);
      int suffixZeroCount = distance(rbegin(row), itLastOne);
      suffixZeros.push_back(suffixZeroCount);
    }

    for (int i = 0; i < n; ++i) {
      int neededZeros = n - i - 1;
      auto it = find_if(begin(suffixZeros) + i, end(suffixZeros), [&](int count) { return count >= neededZeros; });
      if (it == end(suffixZeros)) return -1;
      int j = distance(begin(suffixZeros), it);
      for (int k = j; k > i; --k) suffixZeros[k] = suffixZeros[k - 1];
      ans += j - i;
    }

    return ans;
  }
};
```

Python Solution

Here is a python solution that follows a similar approach to the C++ solution:

```python
python
class Solution:
    def minSwaps(self, grid):
        n = len(grid)
        suffixZeros = [row[::-1].index(1) if 1 in row else n for row in grid]
        res = 0

        for i in range(n):
            if suffixZeros[i] < (n - i - 1):
                for j in range(i + 1, n):
                    if suffixZeros[j] >= (n - i - 1):
                        suffixZeros.insert(i, suffixZeros.pop(j))
                        res += j - i
                        break
                else:
                    return -1
        return res
```

Here, we also check for each row `i` from the top to bottom, and if the count of trailing zeros in the row is less than `n-d-1`, find a row `j` no earlier than `i` with at least `n-i-1` trailing zeros and swap this row with the current row `i`.

JavaScript Solution

This JavaScript solution also uses a similar logic:

```javascript
javascript
var minSwaps = function(grid) {
    const n = grid.length;
    const suffixZeros = grid.map(row => {
        let count = 0;
        for (let i = n - 1; i >= 0 && row[i] === 0; --i) {
            ++count;
        }
        return count;
    });
    let res = 0;

    for (let i = 0; i < n; ++i) {
        let j = i;
        while (j < n && !(suffixZeros[j] >= n - i - 1)) {
            ++j;
        }
        if (j === n) return -1;
        const num = suffixZeros[j];
        suffixZeros.splice(j, 1);
        suffixZeros.splice(i, 0, num);
        res += j - i;
    }

    return res;
};
```

In this solution, we calculate the number of trailing zeros in each row of the grid in advance, and perform the operations by swapping the rows to make the number of trailing zeros in each row meet the requirements.

Java Solution

Similarly, here is a Java solution:

```java
java
class Solution {
    public int minSwaps(int[][] grid) {
        int n = grid.length;
        Integer[] zeros = new Integer[n];
        for (int i = 0; i < n; i++) {
            int cnt = 0;
            for (int j = n - 1; j >= 0 && grid[i][j] == 0; j--)
                cnt++;
            zeros[i] = cnt;
        }

        int res = 0;
        for (int i = 0; i < n; i++) {
            int cur = i;
            while (cur < n && zeros[cur] < n - i - 1)
                cur++;

            if (cur == n)
                return -1;

            int cnt = zeros[cur];
            for (int j = cur; j > i; j--)
                zeros[j] = zeros[j - 1];

            zeros[i] = cnt;
            res += cur - i;
        }

        return res;
    }
}
```

This solution shares a common approach with the earlier ones. We check for each row with sufficient trailing zeros and then make the appropriate swaps. If we can't make a swap that fulfills the requirement, we return `-1`.