1878. Get Biggest Three Rhombus Sums in a Grid

Leetcode Link

Problem Explanation

You are given an m x n integer matrix called `grid`. The goal of this problem is to find the three greatest rhombus sums in the grid. A rhombus sum is the sum of the elements that form the border of a regular rhombus shape in `grid`. The rhombus must have the shape of a square rotated 45 degrees with each of the corners centered in a grid cell.

For example, given the following grid:

```
2 3 1 1
4 1 2 1
5 1 1 1
```

The three greatest rhombus sums would be `[2, 6, 8]`. These rhombus sums correspond to the single cells (1,1), (0,0) and beside it, and (1,1) surrounding the whole grid.

Solution Approach

The main idea of the solution is to iterate through each cell of the grid and consider it as a potential top corner of a rhombus. Then, for each such cell, we check what the maximum size of a rhombus can be with its top corner at this cell, and then calculate the rhombus sum for each of its possible sizes. We store these sums in a set data structure, which is sorted by default, and only keep the top three greatest sums found so far. Finally, we return these three greatest sums in a reversed order as a vector.

Let's go step by step through the provided solution, with a small example grid:

```
grid = [
  [1, 1, 1],
  [1, 2, 1],
  [1, 1, 1]
]
```

1. Initialize a variable `sums` to store the top three rhombus sums found so far. We use a set data structure since it stores sorted elements by default.
2. Iterate through each cell (i, j) of the grid.
3. For each cell (i, j), check what the maximum size of a rhombus can be with its top corner at this cell. This is determined by the condition `i + sz < n && 1 - sz >= 0 && j + 2 * sz < n`. For each rhombus size `sz`, calculate the rhombus sum using the `getSum()` function.
4. As we calculate a rhombus sum, insert it into the `sums` set. To maintain the top three sums, remove the smallest element in the set if its size gets larger than 3.
5. Return the sums in the `sums` set, reversed, as a vector.

Let's see how this works for our example grid:

1. Initialize `sums` to an empty set.
2. Start iterating through the grid cells:
   - For cell (0, 0), the rhombus of size 0 has a sum of 1. Insert this into `sums` (sums = {1}).
   - For cell (0, 1), the rhombus of size 0 has a sum of 1. INSERT this into `sums`, do not insert since 1 is already present.
   - For cell (0, 2), the rhombus of size 0 has a sum of 1. INSERT this into `sums`, do not insert since 1 is already present.
   - For cell (1, 0), the rhombus of size 0 and 1 have sums of 1 and 6. Insert these sums into `sums` (sums = {1, 6}).
   - For cell (1, 1), the rhombus of size 0, 1, and 2 have sums of 2, 6, and 8. Insert these sums into `sums` (sums = {2, 6, 8}).
   - For cell (1, 2), the rhombus of size 0 and 1 have sums of 1 and 6. These sums are already in `sums`, so no changes.
   - For cell (2, 0), the rhombus of size 0 has a sum of 1. This sum is already in `sums`, so no changes.
   - For cell (2, 1), the rhombus of size 0 has a sum of 1. This sum is already in `sums`, so no changes.
   - For cell (2, 2), the rhombus of size 0 has a sum of 1. This sum is already in `sums`, so no changes.
3. Return the elements in `sums` as a reversed vector: `[2, 6, 8]`.

C++ Solution

```
cpp
class Solution {
 public:
  vector<int> getBiggestThree(vector<vector<int>>& grid) {
    const int m = grid.size();
    const int n = grid[0].size();
    set<int> sums;

    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        for (int sz = 0; i + sz < m && i - sz >= 0 && j + 2 * sz < n; ++sz) {
          const int sum = sz == 0 ? grid[i][j] : getSum(grid, i, j, sz);
          sums.insert(sum);
          if (sums.size() > 3)
            sums.erase(begin(sums));
        }

    return vector<int>(rbegin(sums), rend(sums));
  }

 private:
  // Returns rhombus sum of grid with top grid (i, j) and edge size `sz`.
  int getSum(const vector<vector<int>>& grid, int i, int j, int sz) {
    int x = i;
    int y = j;
    int sum = 0;

    // Go left down.
    for (int k = 0; k < sz; ++k)
      sum += grid[++x][++y];

    // Go right down.
    for (int k = 0; k < sz; ++k)
      sum += grid[++x][--y];

    // Go right up.
    for (int k = 0; k < sz; ++k)
      sum += grid[--x][--y];

    // Go left up.
    for (int k = 0; k < sz; ++k)
      sum += grid[--x][++y];

    return sum;
  }
};
```

Please note: Solutions in Python, Java, JavaScript, and C# are not provided as the main solution is in C++ and the description requested only one language solution.## Python Solution

```
python
class Solution(object):
    def getBiggestThree(self, grid):
        m, n = len(grid), len(grid[0])
        sums = set()

        for i in range(m):
            for j in range(n):
                for sz in range(n):
                    if i + sz < m and i - sz >= 0 and j + 2 * sz < n:
                        sum = 0 if sz == 0 else self.getSum(grid, i, j, sz)
                        sums.add(sum)
                        if len(sums) > 3:
                            sums.remove(min(sums))

        return sorted(sums, reverse=True)

    # Returns rhombus sum of grid with top grid (i, j) and edge size `sz`.
    def getSum(self, grid, i, j, sz):
        x, y = i, j
        sum = 0

        # Go left down.
        for k in range(sz):
            x -= 1
            y += 1
            sum += grid[x][y]

        # Go right down.
        for k in range(sz):
            x += 1
            y += 1
            sum += grid[x][y]

        # Go right up.
        for k in range(sz):
            x += 1
            y -= 1
            sum += grid[x][y]

        # Go left up.
        for k in range(sz):
            x -= 1
            y -= 1
            sum += grid[x][y]

        return sum
```

Java Solution

```
java
class Solution {
    public int[] getBiggestThree(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;
        TreeSet<Integer> sums = new TreeSet<>();

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                for (int sz = 0; i + sz < m && i - sz >= 0 && j + 2 * sz < n; sz++) {
                    int sum = sz == 0 ? grid[i][j] : getSum(grid, i, j, sz);
                    sums.add(sum);
                    if (sums.size() > 3) {
                        sums.pollFirst();
                    }
                }
            }
        }

        int[] res = new int[sums.size()];
        int index = res.length - 1;
        for (int sum : sums) {
            res[index--] = sum;
        }

        return res;
    }

    // Returns rhombus sum of grid with top grid (i, j) and edge size `sz`.
    private int getSum(int[][] grid, int i, int j, int sz) {
        int x = i;
        int y = j;
        int sum = 0;

        // Go left down.
        for (int k = 0; k < sz; k++) {
            sum += grid[--x][++y];
        }

        // Go right down.
        for (int k = 0; k < sz; k++) {
            sum += grid[++x][++y];
        }

        // Go right up.
        for (int k = 0; k < sz; k++) {
            sum += grid[++x][--y];
        }

        // Go left up.
        for (int k = 0; k < sz; k++) {
            sum += grid[--x][--y];
        }

        return sum;
    }
}
```

JavaScript Solution

```
javascript
var getBiggestThree = function(grid) {
    const m = grid.length;
    const n = grid[0].length;
    const sums = new Set();

    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            for (let sz = 0; i + sz < m && i - sz >= 0 && j + 2 * sz < n; sz++) {
                const sum = (sz === 0) ? grid[i][j] : getSum(grid, i, j, sz);
                sums.add(sum);
                if (sums.size > 3) {
                    sums.delete(Math.min(...sums));
                }
            }
        }
    }

    return [...sums].sort((a, b) => b - a);

    // Returns rhombus sum of grid with top grid (i, j) and edge size `sz`.
    function getSum(grid, i, j, sz) {
        let x = i;
        let y = j;
        let sum = 0;

        // Go left down.
        for (let k = 0; k < sz; k++) {
            sum += grid[--x][++y];
        }

        // Go right down.
        for (let k = 0; k < sz; k++) {
            sum += grid[++x][++y];
        }

        // Go right up.
        for (let k = 0; k < sz; k++) {
            sum += grid[++x][--y];
        }

        // Go left up.
        for (let k = 0; k < sz; k++) {
            sum += grid[--x][--y];
        }

        return sum;
    }
};
```