Best Meeting Point

A group of two or more people wants to meet and minimize the total travel distance. You are given a 2D grid of values 0 or 1, where each 1 marks the home of someone in the group. The distance is calculated using Manhattan Distance, where distance(p1, p2) = `|p2.x - p1.x| + |p2.y - p1.y|`.

For example, given three people living at `(0,0)`, `(0,4)`, and `(2,2)`:

```
1 - 0 - 0 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

The point `(0,2)` is an ideal meeting point, as the total travel distance of 2+2+2=6 is minimal. So return 6.

1. Try to solve it in one dimension first. How can this solution apply to the two dimension case?

Solution 1

```
public int minTotalDistance(int[][] grid) {
    int m = grid.length;
    int n = grid[0].length;

    List<Integer> I = new ArrayList<>(m);
    List<Integer> J = new ArrayList<>(n);

    for(int i = 0; i < m; i++){
        for(int j = 0; j < n; j++){
            if(grid[i][j] == 1){
                I.add(i);
                J.add(j);
            }
        }
    }

    return getMin(I) + getMin(J);
}

private int getMin(List<Integer> list){
    int ret = 0;

    Collections.sort(list);

    int i = 0;
    int j = list.size() - 1;
    while(i < j){
        ret += list.get(j--) - list.get(i++);
    }

    return ret;
}
```

written by larrywang2014 original link here

Solution 2

As long as you have different numbers of people on your left and on your right, moving a little to the side with more people decreases the sum of distances. So to minimize it, you must have equally many people on your left and on your right. Same with above/below.

Two O(mn) solutions, both take 2ms.

The neat `total += Z[hi--] - Z[lo++]`-style summing is from larrywang2014's solution.
Originally I used `total += abs(Z[i] - median)`-style.

Solution 1

No need to sort the coordinates if we **collect** them in sorted order.

```
public int minTotalDistance(int[][] grid) {
    int m = grid.length, n = grid[0].length;
    int total = 0, Z[] = new int[m*n];
    for (int dim=0; dim<2; ++dim) {
        int i = 0, j = 0;
        if (dim == 0) {
            for (int x=0; x<n; ++x)
                for (int y=0; y<m; ++y)
                    if (grid[y][x] == 1)
                        Z[j++] = x;
        } else {
            for (int y=0; y<m; ++y)
                for (int g : grid[y])
                    if (g == 1)
                        Z[j++] = y;
        }
        while (i < --j)
            total += Z[j] - Z[i++];
    }
    return total;
}
```

Solution 2

BucketSort-ish. Count how many people live in each row and each column. Only O(m+n) space.

```java
public int minTotalDistance(int[][] grid) {
    int m = grid.length, n = grid[0].length;
    int[] I = new int[m], J = new int[n];
    for (int i=0; i<m; ++i)
        for (int j=0; j<n; ++j)
            if (grid[i][j] == 1) {
                ++I[i];
                ++J[j];
            }
    int total = 0;
    for (int[] K : new int[][]{ I, J }) {
        int i = 0, j = K.length - 1;
        while (i < j) {
            int k = Math.min(K[i], K[j]);
            total += k * (j - i);
            if ((K[i] -= k) == 0) ++i;
            if ((K[j] -= k) == 0) --j;
        }
    }
    return total;
}
```

Not sure Larry's way is actually better here. I'll have to try the other style as well...

written by StefanPochmann original link here

Solution 3

When I first saw this question, intuitively I know shortest meeting point should be found in two separate dimension, however, even if on 1-D, how could I find the shortest meeting point? Then I clicked discuss and found out everybody's solution was using median to get shortest meeting point? WHY?

Actually, there is a famous conclusion in statistics that the median minimizes the sum of absolute deviations.

written by TonyLic original link here