Previous Next

296. Best Meeting Point

March 5, 2016 | 20.6K views

Average Rating: 4.78 (46 votes)

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|`.

Example:

```
Input:

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

Output: 6

Explanation: Given three people living at (0,0), (0,4), and (2,2):
             The point (0,2)
 is an ideal meeting point, as the total travel distance
             of 2+2+2=6 is minimal. So return 6.
```

Solution

Approach #1 (Breadth-first Search) [Time Limit Exceeded]

A brute force approach is to evaluate all possible meeting points in the grid. We could apply breadth-first search originating from each of the point.

While inserting a point into the queue, we need to record the distance of that point from the meeting point. Also, we need an extra `visited` table to record which point had already been visited to avoid being inserted into the queue again.

```java
public int minTotalDistance(int[][] grid) {
    int minDistance = Integer.MAX_VALUE;
    for (int row = 0; row < grid.length; row++) {
        for (int col = 0; col < grid[0].length; col++) {
            int distance = search(grid, row, col);
            minDistance = Math.min(distance, minDistance);
        }
    }
    return minDistance;
}

private int search(int[][] grid, int row, int col) {
    Queue<Point> q = new LinkedList<>();
    int m = grid.length;
    int n = grid[0].length;
    boolean[][] visited = new boolean[m][n];
    q.add(new Point(row, col, 0));
    int totalDistance = 0;
    while (!q.isEmpty()) {
        Point point = q.poll();
        int r = point.row;
        int c = point.col;
        int d = point.distance;
        if (r < 0 || c < 0 || r >= m || c >= n || visited[r][c]) {
            continue;
        }
        if (grid[r][c] == 1) {
            totalDistance += d;
        }
        visited[r][c] = true;
        q.add(new Point(r + 1, c, d + 1));
        q.add(new Point(r - 1, c, d + 1));
        q.add(new Point(r, c + 1, d + 1));
        q.add(new Point(r, c - 1, d + 1));
    }
    return totalDistance;
}

public class Point {
    int row;
    int col;
    int distance;
    public Point(int row, int col, int distance) {
        this.row = row;
        this.col = col;
        this.distance = distance;
    }
}
```

Complexity analysis

- Time complexity : $O(m^2n^2)$. For each point in the $m \times n$ size grid, the breadth-first search takes at most $m \times n$ steps to reach all points. Therefore the time complexity is $O(m^2n^2)$.
- Space complexity : $O(mn)$. The `visited` table consists of $m \times n$ elements map to each point in the grid. We insert at most $m \times n$ points into the queue.

Approach #2 (Manhattan Distance Formula) [Time Limit Exceeded]

You may notice that breadth-first search is unnecessary. You can just calculate the Manhattan distance using the formula:

$$distance(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|$$

```java
public int minTotalDistance(int[][] grid) {
    List<Point> points = getAllPoints(grid);
    int minDistance = Integer.MAX_VALUE;
    for (int row = 0; row < grid.length; row++) {
        for (int col = 0; col < grid[0].length; col++) {
            int distance = calculateDistance(points, row, col);
            minDistance = Math.min(distance, minDistance);
        }
    }
    return minDistance;
}

private int calculateDistance(List<Point> points, int row, int col) {
    int distance = 0;
    for (Point point : points) {
        distance += Math.abs(point.row - row) + Math.abs(point.col - col);
    }
    return distance;
}

private List<Point> getAllPoints(int[][] grid) {
    List<Point> points = new ArrayList<>();
    for (int row = 0; row < grid.length; row++) {
        for (int col = 0; col < grid[0].length; col++) {
            if (grid[row][col] == 1) {
                points.add(new Point(row, col));
            }
        }
    }
    return points;
}

public class Point {
    int row;
    int col;
    public Point(int row, int col) {
        this.row = row;
        this.col = col;
    }
}
```

Complexity analysis

- Time complexity : $O(m^2n^2)$. Assume that k is the total number of houses. For each point in the $m \times n$ size grid, we calculate the manhattan distance in $O(k)$. Therefore the time complexity is $O(mnk)$. But do note that there could be up to $m \times n$ houses, making the worst case time complexity to be $O(m^2n^2)$.
- Space complexity : $O(mn)$.

Approach #3 (Sorting) [Accepted]

Finding the best meeting point in a 2D grid seems difficult. Let us take a step back and solve the 1D case which is much simpler. Notice that the Manhattan distance is the sum of two independent variables. Therefore, once we solve the 1D case, we can solve the 2D case as two independent 1D problems.

Let us look at some 1D examples below:

```
Case #1: 1-0-0-0-1

Case #2: 0-1-0-1-0
```

We know the best meeting point must locate somewhere between the left-most and right-most point. For the above two cases, we would select the center point at $x = 2$ as the best meeting point. How about choosing the mean of all points as the meeting point?

Consider this case:

```
Case #3: 1-0-0-0-0-0-0-1-1
```

Using the mean gives us $\bar{x} = \frac{0+7+8}{3} = 5$ as the meeting point. The total distance is 10.

But the best meeting point should be at $x = 7$ and the total distance is 8.

You may argue that the mean is *close* to the optimal point. But imagine a larger case with many 1's congregating on the right side and just a single 1 on the left-most side. Using the mean as the meeting point would be far from optimal.

Besides mean, what is a better way to represent the distribution of points? Would median be a better representation? Indeed. In fact, the median *must* be the optimal meeting point.

```
Case #4: 1-1-0-0-1
```

To see why this is so, let us look at case #4 above and choose the median $x = 1$ as our initial meeting point. Assume that the total distance traveled is *d*. Note that we have equal number of points distributed to its left and to its right. Now let us move one step to its right where $x = 2$ and notice how the distance changes accordingly.

Since there are two points to the left of $x = 2$, we add $2 * (+1)$ to *d*. And *d* is offset by –1 since there is one point to the right. This means the distance had overall increased by 1.

Therefore, it is clear that:

> As long as there is equal number of points to the left and right of the meeting point, the total distance is minimized.

```
Case #5: 1-1-0-0-1-1
```

One may think that the optimal meeting point must fall on one of the 1's. This is true for cases with odd number of 1's, but not necessarily true when there are even number of 1's, just like case #5 does. You can choose any of the $x = 1$ to $x = 4$ points and the total distance is minimized. Why?

The implementation is direct. First we collect both the row and column coordinates, sort them and select their middle elements. Then we calculate the total distance as the sum of two independent 1D problems.

```java
public int minTotalDistance(int[][] grid) {
    List<Integer> rows = new ArrayList<>();
    List<Integer> cols = new ArrayList<>();
    for (int row = 0; row < grid.length; row++) {
        for (int col = 0; col < grid[0].length; col++) {
            if (grid[row][col] == 1) {
                rows.add(row);
                cols.add(col);
            }
        }
    }
    int row = rows.get(rows.size() / 2);
    Collections.sort(cols);
    int col = cols.get(cols.size() / 2);
    return minDistance1D(rows, row) + minDistance1D(cols, col);
}

private int minDistance1D(List<Integer> points, int origin) {
    int distance = 0;
    for (int point : points) {
        distance += Math.abs(point - origin);
    }
    return distance;
}
```

Note that in the code above we do not need to sort *rows*, why?

Complexity analysis

- Time complexity : $O(mn \log mn)$. Since there could be at most $m \times n$ points, therefore the time complexity is $O(mn \log mn)$ due to sorting.
- Space complexity : $O(mn)$.

Approach #4 (Collect Coordinates in Sorted Order) [Accepted]

We could use the Selection algorithm to select the median in $O(mn)$ time, but there is an easier way. Notice that we can collect both the row and column coordinates in sorted order.

```java
public int minTotalDistance(int[][] grid) {
    List<Integer> rows = collectRows(grid);
    List<Integer> cols = collectCols(grid);
    int row = rows.get(rows.size() / 2);
    int col = cols.get(cols.size() / 2);
    return minDistance1D(rows, row) + minDistance1D(cols, col);
}

private int minDistance1D(List<Integer> points, int origin) {
    int distance = 0;
    for (int point : points) {
        distance += Math.abs(point - origin);
    }
    return distance;
}

private List<Integer> collectRows(int[][] grid) {
    List<Integer> rows = new ArrayList<>();
    for (int row = 0; row < grid.length; row++) {
        for (int col = 0; col < grid[0].length; col++) {
            if (grid[row][col] == 1) {
                rows.add(row);
            }
        }
    }
    return rows;
}

private List<Integer> collectCols(int[][] grid) {
    List<Integer> cols = new ArrayList<>();
    for (int col = 0; col < grid[0].length; col++) {
        for (int row = 0; row < grid.length; row++) {
            if (grid[row][col] == 1) {
                cols.add(col);
            }
        }
    }
    return cols;
}
```

You can calculate the distance without knowing the median using a two pointer approach. This neat approach is inspired by @larrywang2014's solution.

```java
public int minTotalDistance(int[][] grid) {
    List<Integer> rows = collectRows(grid);
    List<Integer> cols = collectCols(grid);
    return minDistance1D(rows) + minDistance1D(cols);
}

private int minDistance1D(List<Integer> points) {
    int distance = 0;
    int i = 0;
    int j = points.size() - 1;
    while (i < j) {
        distance += points.get(j) - points.get(i);
        i++;
        j--;
    }
    return distance;
}
```

Complexity analysis

- Time complexity : $O(mn)$.
- Space complexity : $O(mn)$.

Rate this article: ★ ★ ★ ★ ★

Previous Next

Comments: 17 Sort By

Type comment here... (Markdown is supported)

Preview Post

hhbagels ★90 July 23, 2019 6:32 AM

Why is the BFS solution made to timeout ? It's brutal for an interviewer to expect someone to write the Math part of this solution! If someone writes the Math answer doesn't mean that they have already memorized the solution ? Do you guys want such candidates ? what's the point of interviewing ? Test memory skills ?

8 Share Reply

SHOW 11 REPLIES

tianchi-seattle ★325 May 11, 2019 1:06 PM

This problem is asking for central of gravity in discrete domain. In natural world, It's a point where matters cancel each other's magnetic force. It's a balance point, a point of dynamic equilibrium. In math, we call this concept the median.

Particles are not aware of this point. They only know how to cancel each other's force to achieve

Read More

8 Share Reply

SHOW 1 REPLY

twilightgod ★13 October 3, 2018 9:50 AM

I think #3 and #4 space complexity is O(m + n)

3 Share Reply

SHOW 2 REPLIES

mayank43 ★81 September 8, 2019 2:49 PM

last solution is awesome

1 Share Reply

cja ★53 August 5, 2019 8:49 AM

Solution with `O(mn)` time, `O(m+n)` space. Idea: first compute row/column sums, then use those and walk down the rows/columns to compute the row/column costs that contribute to the total Manhattan distance, and finally chose the best row and column (with lowest costs).

Read More

1 Share Reply

j_mm ★1 June 14, 2019 9:12 PM

Would be more interesting if the question adds the condition that the number of people living in each building is different. Then, the solution would be to find the weighted median for X and Y?

1 Share Reply

SHOW 1 REPLY

a-b-c ★547 April 12, 2019 1:05 PM

Is the time complexity correct in the 3rd approach? I think it should be `O(mn + logmn)` rather than `O(mnlogmn)`.

My analysis:

Read More

0 Share Reply

SHOW 1 REPLY

Liziyang1110 ★54 March 20, 2019 8:19 PM

I think the space complexity of #4 is O(Math.max(m,n));

0 Share Reply

ilyasaber14 ★216 December 12, 2018 7:59 AM

Funny, my breadth first search was accepted. I originally went with mean approach, but couldn't get the answer right (as explained above).

0 Share Reply

516364598chang ★97 September 30, 2018 3:34 AM

Why median is the correct point? The above explaination only gives and example. How to prove it is always right?

0 Share Reply

SHOW 1 REPLY