Largest Plus Sign

In a 2D `grid` from (0, 0) to (N-1, N-1), every cell contains a `1`, except those cells in the given list `mines` which are `0`. What is the largest axis-aligned plus sign of `1`s contained in the grid? Return the order of the plus sign. If there is none, return 0.

An "*axis-aligned plus sign of `1`s* of order **k**" has some center `grid[x][y] = 1` along with 4 arms of length `k-1` going up, down, left, and right, and made of `1`s. This is demonstrated in the diagrams below. Note that there could be `0`s or `1`s beyond the arms of the plus sign, only the relevant area of the plus sign is checked for 1s.

Examples of Axis-Aligned Plus Signs of Order k:

```
Order 1:
000
010
000

Order 2:
00000
00100
01110
00100
00000

Order 3:
0000000
0001000
0001000
0111110
0001000
0001000
0000000
```

Example 1:

```
Input: N = 5, mines = [[4, 2]]
Output: 2
Explanation:
11111
11111
11111
11111
11011
In the above grid, the largest plus sign can only be order 2.  One of them is marked
in bold.
```

Example 2:

```
Input: N = 2, mines = []
Output: 1
Explanation:
There is no plus sign of order 2, but there is of order 1.
```

Example 3:

```
Input: N = 1, mines = [[0, 0]]
Output: 0
Explanation:
There is no plus sign, so return 0.
```

Note:

1. `N` will be an integer in the range `[1, 500]`.
2. `mines` will have length at most `5000`.
3. `mines[i]` will be length 2 and consist of integers in the range `[0, N-1]`.
4. *(Additionally, programs submitted in C, C++, or C# will be judged with a slightly smaller time limit.)*

Solution 1

Algorithms: For each position `(i, j)` of the `grid` matrix, we try to extend in each of the four directions (left, right, up, down) as long as possible, then take the minimum length of `1`'s out of the four directions as the order of the largest axis-aligned plus sign centered at position `(i, j)`.

Optimizations: Normally we would need a total of five matrices to make the above idea work -- one matrix for the `grid` itself and four more matrices for each of the four directions. However, these five matrices can be combined into one using two simple tricks:

1. For each position `(i, j)`, we are only concerned with the minimum length of `1`'s out of the four directions. This implies we may combine the four matrices into one by only keeping tracking of the minimum length.
2. For each position `(i, j)`, the order of the largest axis-aligned plus sign centered at it will be `0` if and only if `grid[i][j] == 0`. This implies we may further combine the `grid` matrix with the one obtained above.

Implementations:

1. Create an `N-by-N` matrix `grid`, with all elements initialized with value `N`.
2. Reset those elements to `0` whose positions are in the `mines` list.
3. For each position `(i, j)`, find the maximum length of `1`'s in each of the four directions and set `grid[i][j]` to the minimum of these four lengths. Note that there is a simple recurrence relation relating the maximum length of `1`'s at current position with previous position for each of the four directions (labeled as `l`, `r`, `u`, `d`).
4. Loop through the `grid` matrix and choose the maximum element which will be the largest axis-aligned plus sign of `1`'s contained in the grid.

Solutions: Here is a list of solutions for Java/C++/Python based on the above ideas. All solutions run at `O(N^2)` time with `O(N^2)` extra space. Further optimizations are possible such as keeping track of the maximum plus sign currently available and terminating as early as possible if no larger plus sign can be found for current row/column. Note for the Python version, using the builtin `max` or `min` functions results in `TLE`, don't know why :).

1. **Java:**

```java
public int orderOfLargestPlusSign(int N, int[][] mines) {
    int[][] grid = new int[N][N];

    for (int i = 0; i < N; i++) {
        Arrays.fill(grid[i], N);
    }

    for (int[] m : mines) {
        grid[m[0]][m[1]] = 0;
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0, k = N - 1, l = 0, r = 0, u = 0, d = 0; j < N; j++, k--) {
            grid[i][j] = Math.min(grid[i][j], l = (grid[i][j] == 0 ? 0 : l + 1));
            grid[i][k] = Math.min(grid[i][k], r = (grid[i][k] == 0 ? 0 : r + 1));
            grid[j][i] = Math.min(grid[j][i], u = (grid[j][i] == 0 ? 0 : u + 1));
            grid[k][i] = Math.min(grid[k][i], d = (grid[k][i] == 0 ? 0 : d + 1));
        }
    }

    int res = 0;

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            res = Math.max(res, grid[i][j]);
        }
    }

    return res;
}
```

2. **C++:**

```
int orderOfLargestPlusSign(int N, vector<vector<int>>& mines) {
    vector<vector<int>> grid(N, vector<int>(N, N));

    for (auto& m : mines) {
        grid[m[0]][m[1]] = 0;
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0, k = N - 1, l = 0, r = 0, u = 0, d = 0; j < N; j++, k--) {
            grid[i][j] = min(grid[i][j], l = (grid[i][j] == 0 ? 0 : l + 1));
            grid[i][k] = min(grid[i][k], r = (grid[i][k] == 0 ? 0 : r + 1));
            grid[j][i] = min(grid[j][i], u = (grid[j][i] == 0 ? 0 : u + 1));
            grid[k][i] = min(grid[k][i], d = (grid[k][i] == 0 ? 0 : d + 1));
        }
    }

    int res = 0;

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            res = max(res, grid[i][j]);
        }
    }

    return res;
}
```

3. **Python:**

```
def orderOfLargestPlusSign(self, N, mines):
    """
    :type N: int
    :type mines: List[List[int]]
    :rtype: int
    """
    grid = [[N] * N for i in range(N)]

    for m in mines:
        grid[m[0]][m[1]] = 0

    for i in range(N):
        l, r, u, d = 0, 0, 0, 0

        for j, k in zip(range(N), reversed(range(N))):
            l = l + 1 if grid[i][j] != 0 else 0
            if (l < grid[i][j]):
                grid[i][j] = l

            r = r + 1 if grid[i][k] != 0 else 0
            if (r < grid[i][k]):
                grid[i][k] = r

            u = u + 1 if grid[j][i] != 0 else 0
            if (u < grid[j][i]):
                grid[j][i] = u

            d = d + 1 if grid[k][i] != 0 else 0
            if (d < grid[k][i]):
                grid[k][i] = d

    res = 0

    for i in range(N):
        for j in range(N):
            if (res < grid[i][j]):
                res = grid[i][j]

    return res
```

written by fun4LeetCode original link here

Solution 2

For this problem, I first think of the related problem finding largest square, and I tried DP as DP can give O(n^2) time complexity. They way I do it is use a dp[N][N][4] to record each 1s consecutive 1s in 4 directions and they can be abrupt by 0. However, my code fails at last test case because of memory. Obviously, there is a tradeoff in space and time.

In order to pass I take advantage of having limited 0s on last test case and generated the following O(N^3) solution, the general idea is do bfs (I actually update everything on row and col) and update grid every time we meet a 0. There could be total N^2 0s and update board takes additional N so it's N^3.

To make things easier, I initialize the board with rules to take in count of boarder.

```
int orderOfLargestPlusSign(int N, vector<vector<int>>& mines)
{
    vector<vector<int>> grid(N,vector<int>(N,N));

    for(int i =0; i < N; i++)
        for(int j =0; j< N; j++)
        {
            grid[i][j] = min(i-0+1,j-0+1);
            grid[i][j] = min(grid[i][j],N-i);
            grid[i][j] = min(grid[i][j],N-j);
        }

    for(auto v : mines)
    {
        int i = v[0];
        int j = v[1];

        for(int ii =0; ii< N;ii++)
            grid[ii][j] = min(grid[ii][j],abs(ii-i));

        for(int jj =0; jj< N;jj++)
            grid[i][jj] = min(grid[i][jj],abs(jj-j));
    }


    int result =0;

    for(int i =0; i < N; i++)
        for(int j =0; j< N; j++)
            result = max(result,grid[i][j]);

    return result;
}
```

written by zhutianqi original link here

Solution 3

```
from bisect import bisect_right

class Solution(object):
    def orderOfLargestPlusSign(self, N, mines):
        """
        :type N: int
        :type mines: List[List[int]]
        :rtype: int
        """
        rows = [[-1, N] for _ in xrange(N)]
        cols = [[-1, N] for _ in xrange(N)]
        for r, c in mines:
            rows[r].append(c)
            cols[c].append(r)
        for i in xrange(N):
            rows[i].sort()
            cols[i].sort()
        mxp = 0
        for r in xrange(N):
            for i in xrange(len(rows[r])-1):
                left_b = rows[r][i]
                right_b = rows[r][i+1]
                for c in xrange(left_b+mxp+1, right_b-mxp):
                    idx = bisect_right(cols[c], r)-1
                    up_b = cols[c][idx]
                    down_b = cols[c][idx+1]
                    mxp = max(mxp, min(c-left_b, right_b-c, r-up_b, down_b-r))
        return mxp
```

written by pedro original link here