

NOI 2018 Editorial

Nordic Camping Solution

We are given an $N \times M$ grid, where each cell (x, y) is either covered by a rock or empty. We are then given Q queries, each asking us the size of the largest square without any rocks that covers a given cell (x, y) .

Subtask 1, $N, M \leq 50$, $Q \leq 1000$

The grid is small. Since both N, M are bounded by 50, we know that the largest square can have at most a side length of 50.

One way to solve this subtask is to consider each cell as the upper left corner of some square. For each cell we consider, we can slowly increase the size of the square until it contains a rock in it. When we have found the largest square, we can store the size of this square for all cells in this square.

```
for r in range(N):
    for c in range(M):

        # O(NM)
        s = 1
        for ds in range(1,N+1):
            # Check that no rocks are on row (r+ds) between column c and c+ds, otherwise
            break

            # Check that no rocks are on column (c+ds) between row r and r+ds, otherwise
            break

        s = s + 1

        # Now (r,c), (r+s,c), (r+s,c+s), (r,c+s) is the largest square starting at (r,c)
        for i in range(r,r+s+1):
            for j in range(c,c+s+1):
                maxsquare[i][j] = max(maxsquare[i][j], (s+1)*(s+1))
```

This precomputes all answers to all possible queries in $O(N^2M^2)$, which allows to answer each query in $O(1)$.

Subtask 3, $N \leq 10$, $Q \leq 500$

We will use the fact that $N \leq 10$. The largest square is at most 10.

For each query, we then only have to consider squares that start on the 10 columns to the left. This creates a 10×10 area of cells that we want to consider.

```
# Let's say we want to know the largest square containing (x,y)
ans = 0
```

```

for r in range(x-10,x+1): #O(N)
    for c in range(y-10,y+1): #O(N)

        # O(N^2)
        s = 1
        for ds in range(1,11):
            # Check that no rocks are on row (r+ds) between column c and c+ds, otherwise
            break

            # Check that no rocks are on column (c+ds) between row r and r+ds, otherwise
            break

            s = s + 1

        if (the square (r,c), (r+s,c), (r+s,c+s), (r,c+s) contains (x,y)):
            ans = max(ans, s*s)

```

Note that this solution runs in $O(N^4Q)$, which could be a bit too slow.

We can save a factor N by precalculating a 2D-prefix-sum, that can be used to calculate the number of rocks in a given rectangle of the grid. Then we can perform the inner check in just $O(1)$, by checking if the smallest square containing both (r, c) and (x, y) has no rocks in it.

```

# Let's say we want to know the largest square containing (x,y)

ans = 0

for r in range(x-10,x+1): #O(N)
    for c in range(y-10,y+1): #O(N)

        s = max(abs(r-x),abs(c-y))+1

        if (prefixRectangle(r,c,r+s,c+s) has no rocks):
            ans = max(ans,s*s)

```

Now each query can be answered in $O(N^2)$ time with $O(NM)$ precomputation building the 2D-prefix-sum, resulting in an $O(N^2Q + NM)$ solution, which is fast enough for this subtask.

Full Solution

We will precompute all answers, and then answer each query in $O(1)$.

Consider a cell (r, c) , our goal is to calculate the largest square (with no rocks) that contains it.

Before we end up with that, we can first try to figure out how to find the largest square that start on (r, c) (implying that (r, c) is the upper left corner of the square).

One way, is to precalculate for each cell, the closest rock to the right, and closest rock under the cell. Then we can iterate the cells from bottom to up, and right to left, and calculate in the following way

```

for r = N, N-1, ..., 1:
    for c = M, M-1, ..., 1:
        if (r,c) is rock:
            startVal[r][c] = 0
        else:
            startVal[r][c] = min(startVal[r-1][c-1]+1,
                                distToRightRock[r][c],distToUnderRock[r][c])

```

This calculates the largest square starting (r, c) , by taking the minimum value of

- Largest square starting at cell $(r - 1, c - 1) + 1$
- Distance to the rock to the right
- Distance to the rock under the rock.

This is done in $O(NM)$.

Another way to calculate the same thing, is by precalculating a 2D-prefix-sum over the number of rocks in the grid. We can then for each cell binary search the largest square that contains no rocks, starting on (r, c) . This results in a solution running in $O(NM \log(NM))$.

Now $startVal[r][c]$ contains the largest square without rocks, starting on the cell (r, c) . What remains is to have some way to distribute the information in $s = startVal[r][c]$ onto all cells within the square $(r, c), (r + s, c), (r + s, c + s), (r, c + s)$.

Instead of tackling the 2-dimensional problem, we can first consider the 1-dimensional problem. We can consider an array, and we want to distribute the information at index i to the s cells in front of it. An even more generalized version of this is, we are given Q intervals over an array of N positions where each interval has some value s_i assigned to it, and we want to answer for each position, the largest value of any interval that covers the position.

A solution to the general problem is to maintain some array of vectors. Then, for each interval $[a_i, b_i]$, insert the element s_i at index a_i , and element $-s_i$ at index $b_i + 1$. Then we can sweep through the array from index 1 to N while maintaining a data structure like a multiset or priority queue. When we are at an index i of the array, for each element v in the corresponding vector, if v is positive we will insert v into our data structure, and otherwise if v is negative we will remove $-v$ from our data structure. After handling all values, we read what the largest value is in the largest data structure, which is the answer to the problem at index i .

This solution could be applied to our 2-dimensional problem on squares. We can first consider each (r, c) to only distribute it to the s elements to the right, thus we can solve the 1-dimensional problem on each row. Since we are handling squares in the main problem, we know that if a cell (r, c) is covered by some square of side length s from the left, then it will also cover the s cells below the cell (r, c) . Thus, we can solve the same problem for each column again.

This results in each cell containing the information about the largest square containing it, which can be done in $O(NM(\log N + \log M))$.