1 Problem

Bessie is using her supercow strength to push a tractor around for Farmer John, but because she is health-conscious, she wants to measure how much energy she exerts in order to plan out her daily caloric intake. She starts off in a field N ($1 \leq N \leq 1000$) by M ($1 \leq M \leq 1000$) and has to push the tractor along certain paths, at most Q ($1 \leq Q \leq 1000$) which can intersect themselves and are contained in the grid. Given the force she exerts at every point, calculate the total work done over all the paths.
Recall that $W = \vec{F} \cdot \vec{d}$ where W is the work, $\vec{F}$ is the force, and $\vec{d}$ is the displacement.

A path is described by an integer starting point (a, b) where $0 \leq a \leq N$ and $0 \leq b \leq M$, and then followed by at most 10^3 "tokens". Each token specifies one of the 4 cardinal directions and an integer length to move in that direction for. Note that the force can change through a token which changes the work along a token.

INPUT FORMAT:
The first line contains N, M, and Q.
Each line in the next N lines contains M pairs of integers which correspond to the horizontal and vertical components of the force at that point.
The next Q lines contain each path, as described above.

OUTPUT FORMAT:
A single integer representing the total work Bessie does over all the paths.

SAMPLE INPUT:
4 4 3
1 4 -2 3 3 -2 4 -2
-3 2 3 2 1 0 2 3
0 3 -4 2 5 1 -3 2
1 2 2 -3 2 3 -1 5
0 0 1 E 2 S 1 E
0 2 1 S 1 W 1 N 1 E
1 3 1 N 1 S 1 W 1 E

SAMPLE OUTPUT:
-11

```
The force grid looks like this:
(1, 4) (-2, 3) (3, -2) (4, -2)
(-3, 2) (3, 2) (1, 0) (2, 3)
(0, 3) (-4, 2) (5, 1) (-3, 2)
```

On the first path, Bessie starts out at 0, 0, then moves one east (1*1), 2 south (3*1 + 2*1) then one east (-4*1) for a total of 2 work.

On the second path, Bessie starts out at 0, 2, then moves one south (-2*1), one west (1*-1), one north (2*-1), one east (-2*1) for a total of -7 work.

On the third path, Bessie starts out at 1, 3, then moves one north (3*-1), one south (-2*1), one west (2*-1) and one east (1*1) for a total of -6 work.

$$2 - 7 - 6 = -11$$

2 Solutions

2.1 Naive

Simply simulate each query, walking along the path and computing the dot product every time. This runs in $O(QTN)$ in the worst case, where T is the number of tokens, which is too slow for the given bounds.

```
N, M, Q = map(int, input().split())

# force matricies
forcex, forcey = [[0]*M for i in range(N)], [[0]*M for i in range(N)]
for i in range(N):
    line = list(map(int, input().split()))
    for j in range(0, 2*M, 2):
        forcex[i][j//2] = line[j]
        forcey[i][j//2] = line[j + 1]

# direction lookup
dirs = {"N": (-1, 0), "E": (0, 1), "S": (1, 0), "W": (0, -1)}
work = 0

for i in range(Q):
    line = input().split()
    a, b, line = int(line[0]), int(line[1]), line[2:]
    path = 0
    for j in range(0, len(line), 2):
        # vector in magnitude-direction notation
        mag, dir = int(line[j]), dirs[line[j + 1]]
        for k in range(mag):
            # dot product
            path += dir[0]*forcey[a][b] + dir[1]*forcex[a][b]
            # vector addition
            a, b = a + dir[0], b + dir[1]
    work += path

print(work)
```

2.2 Full Credit

The solution relies on the fact that in the dot product, the displacement vector doesn't change and thus can be factored out. Also, the x and y directions can be treated independently. Thus, the answer is just the prefix sum down the particular row or column traversed by the displacement vector.

```
N, M, Q = map(int, input().split())

# force matricies
forcex, forcey = [[0]*M for i in range(N)], [[0]*M for i in range(N)]
for i in range(N):
    line = list(map(int, input().split()))
    for j in range(0, 2*M, 2):
        forcex[i][j//2] = line[j]
        forcey[i][j//2] = line[j + 1]

# prefix sums in x and y directions
def query(prefix, x1, y1, x2, y2):
    return prefix[x2 + 1][y2 + 1] - prefix[x2 + 1][y1] \
         - prefix[x1][y2 + 1] + prefix[x1][y1]

prefixx = [[0]*(M + 1) for i in range(N + 1)]
prefixy = [[0]*(M + 1) for i in range(N + 1)]

for i in range(1, len(prefixx)):
    for j in range(1, len(prefixx[0])):
        prefixx[i][j] += prefixx[i - 1][j] + forcex[i - 1][j - 1]

for i in range(1, len(prefixx)):
    for j in range(1, len(prefixx[0])):
        prefixx[i][j] += prefixx[i][j - 1]

for i in range(1, len(prefixy)):
    for j in range(len(prefixy[0])):
        prefixy[i][j] += prefixy[i - 1][j] + forcey[i - 1][j - 1]

for i in range(1, len(prefixy)):
    for j in range(1, len(prefixy[0])):
        prefixy[i][j] += prefixy[i][j - 1]

# direction lookup
dirs = {"N": (-1, 0), "E": (0, 1), "S": (1, 0), "W": (0, -1)}
work = 0
```

```
for i in range(Q):
    line = input().split()
    a, b, line = int(line[0]), int(line[1]), line[2:]
    path = 0
    for j in range(0, len(line), 2):
        # vector in magnitude-direction notation
        mag, dir = int(line[j]), dirs[line[j + 1]]
        # use prefix sums to compute dot product over the entire vector
        mag -= 1
        path += query(prefixy, a, b, a + mag*dir[0], b + mag*dir[1])*dir[0] + \
                query(prefixx, a, b, a + mag*dir[0], b + mag*dir[1])*dir[1]
        # vector addition
        mag += 1
        a, b = a + mag*dir[0], b + mag*dir[1]
    work += path

print(work)
```

This runs in $O(QT)$, which is sufficient.

2.3 Misc

I orginally planned this problem as a way to build up a "discrete gradient", in the same way that a prefix sum is a discrete integral. Unfortunately, the fundamental theorem of line integrals no longer works on the discrete case, or at least I couldn't think of a way to get it to work.

My code is available here:

```
# precompute the gradient to the vector field, such that
# grad f = F = <x, 2y> (where grad f = <df/dx, df/dy>)
# we know the vector field is conservative so the gradient exists
grad = [[0]*M for i in range(N)]
for y in range(1, N):
    for x in range(1, M):
        # propagate differential in the x direction
        grad[y][x] = x - 1 + grad[y][x - 1]
    if y < N - 1:
        # propagate differential in the y direction
        grad[y + 1][0] = 2*y + grad[y][0]

for i in range(Q):
    line = input().split()
    a, b, line = int(line[0]), int(line[1]), line[2:]
    # endpoint
    c, d = a, b
    for j in range(0, len(line), 2):
        # vector in magnitude-direction notation
        mag, dir = int(line[j]), dirs[line[j + 1]]
        # vector addition
        c, d = c + mag*dir[0], d + mag*dir[1]
    # line integral^b_a of F dr = f(r(b)) - f(r(a))
    print(grad[c][d] - grad[a][b])
    work += grad[c][d] - grad[a][b]
```