오답 이유 추정 : 왕복(2\*(가로-1) or 2\*(세로-1))을 제외한 상어의 움직임을 간단히 계산하기 위해서 나머지 계산으로 차후 위치를 계산했는데, 해당 계산이 틀린 부분이 있었던 것 같다. 개념은 같은데 while을 통해 왕복을 제외하고 한 칸씩 상어를 움직였던 다른 사람 코드는 정답이었다. 차후 해당 코드를 다시 작성할 예정

풀이 과정

낚시꾼이 한 칸씩 가로로 움직일 때, 낚시꾼과 같은 열에 위치한 가장 가까운 상어가 한마리씩 잡히고 나머지 상어들은 자신의 속도만큼 상하좌우로 움직이는 구조이다.

낚시꾼과 같은 열에 위치한 가장 가까운 상어는 a열이라고 가정하면 arr[a][1~R]순서로 검사하면서 제일 먼저 있는 것이 확인되는 상어(1행에 제일 가까운 상어)가 가장 가깝다고 할 수 있다. 이렇게 잡히는 상어들의 무게를 합산해서 결과값으로 출력하면 정답이 나온다.

상어의 이동의 경우 벽에 부딪힐 경우 벽을 통과하는 것이 아닌 벽과 충돌해 반대방향으로 이동하게 된다. 이 때, 과도하게 큰 속도를 갖는 상어의 움직임을 한  칸씩 반복문으로 확인하기에는 시간이 많이 걸리기 때문에 왕복을 할 수 있는 경우(동일한 위치, 동일한 방향) 계산하는 의미가 없으므로 왕복하는 거리는 제외하고 계산을 진행하는 것이 좋다. 이 때, 좌측에서 우측(또는 상단에서 하단)까지 가는 게 필요한 거리는 (가로-1 or 세로-1)이기 때문에, 왕복을 위해서 필요한 거리는 2\*(가로-1 or 세로-1)이다.

왕복을 할 수 있는 경우 위의 값을 빼준 뒤 나머지, 할 수 없는 경우 본래 속도 그대로 모든 살아있는 상어들을 이동시킨다. 이후 위치가 겹쳐있는 경우(정답 코드의 경우 arr[][]에 상어의 무게값을 넣어서 상어의 다음 위치에 다른 상어가 있는지 여부를 체크했다.)는 무게 비교를 하여 작은 상어를 제외하고 큰 상어를 해당 위치에 넣으면 된다.

 #include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int R, C, M;

int dx[] = { 0,-1,1,0,0 };

int dy[] = { 0,0,0,1,-1 };

int map[101][101];

struct Shark {

int r;

int c;

int s;

int d;

int z;

//살아있는지 여부

bool isAlive;

};

vector<Shark> vt;

int getShark(int c) {

int getsize = 0;

//낚시왕이 서있는 열의 행들만 검사한다.

for (int i = 1; i <= R; i++) {

//상어가 있으면

if (map[i][c] != 0) {

//크기를 받아오고

getsize = vt[map[i][c]].z;

//잡았으므로 죽인다.

vt[map[i][c]].isAlive = false;

//해당위치에서 지워준다.

map[i][c] = 0;

//가장 가까운 한마리만 잡음로 break

break;

}

}

//크기를 리턴

return getsize;

}

void sharkMove() {

int r, c, s, d, z;

for (int i = 1; i <= M; i++) {

//죽은 상어는 skip

if (vt[i].isAlive == false) continue;

r = vt[i].r;

c = vt[i].c;

s = vt[i].s;

d = vt[i].d;

z = vt[i].z;

//다른 곳으로 이동할 것이므로 원래 있던 자리를 0으로 넣어준다.

//이전 index의 상어가 현재 상어의 자리에 와 있지 않은 경우에만 0으로 초기화

if (map[r][c] == i) {

map[r][c] = 0;

}

//s를 줄여준다.

if (d <= 2) {

s = s % ((R-1) \* 2);

}

else {

s = s % ((C-1) \* 2);

}

//스피드(s)만큼 d방향으로 이동

for (int j = 0; j < s; j++) {

r += dx[d];

c += dy[d];

//범위를 넘어가는 경우

if (r < 1 || c < 1 || r > R || c > C) {

if (d == 1 || d == 3) {

d += 1;

}

else {

d -= 1;

}

//범위 안으로 하나 들어오고

r += dx[d];

c += dy[d];

//j는 다시 빼준다.(한 번 잘못 움직인 것 이므로)

j--;

}

}

//변경된 좌표와 방향을 상어 정보에 새로 저장

vt[i].r = r;

vt[i].c = c;

vt[i].d = d;

//이동한 곳에 다른 상어가 없다면 바로 저장

if (map[r][c] == 0) {

map[r][c] = i;

}

else if (map[r][c] < i) {

// 이전 인덱스의 상어가 있는 경우에는 크기가 더 큰 상어가 더 작은 상어를 잡아먹는다.

int tmp = vt[map[r][c]].z;

if (z > tmp) {

//현재 인덱스의 상어가 더 큰 경우 원래 있던 상어를 잡아먹는다.

vt[map[r][c]].isAlive = false;

map[r][c] = i;

}

else {

//현재 인덱스의 상어가 더 작은 경우 잡아먹힌다.

vt[i].isAlive = false;

}

}

else {

//현재 인덱스보다 뒷순서의 상어라면 어차피 이동할 것이므로 그냥 덮어쓴다.

map[r][c] = i;

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(0);

cin >> R >> C >> M;

//상어의 수가 0인 경우는 바로 0을 출력하고 종료

if (M == 0) {

cout << 0 << '\n';

return 0;

}

int r, c, s, d, z;

//index를 1부터 시작해주기 위해서 0번째에 아무거나 넣어놓는다.

vt.push\_back({ 0,0,0,0,0 });

for (int i = 1; i <= M; i++) {

cin >> r >> c >> s >> d >> z;

//상어 정보를 벡터에 넣는다.

vt.push\_back({ r,c,s,d,z,true });

//배열에 위치한 상어의 index를 저장

map[r][c] = i;

}

int ans = 0;

//C만큼 실행

for (int i = 1; i <= C; i++) {

ans += getShark(i);

sharkMove();

}

cout << ans << "\n";

return

 0;

}