Nikolas Mavrogeneiadis
gravitorious
August 17, 2024

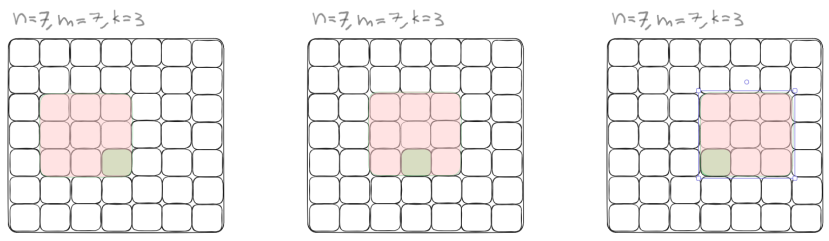
Photoshoot for Gorillas

Problem: *Link*
For the solution, check the next page.

So, we have a $k \times k$ square grid that moves inside the $n \times m$ rectangle. If we knew how many times each cell inside the initial grid getting visited from the $k \times k$ square grid, we would just put greedily the monkey with the highest height to the cell with the most visits, the $2nd$ highest height monkey to the $2nd$ most visited cell, etc. That would be optimal (why?).

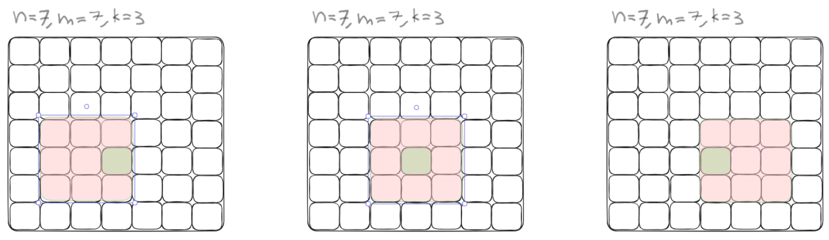
Now, we just need to calculate it for each cell. $n * m$ is bounded above by $2 * 10^5$ and because $t \leqslant 10^3$ we can for each test case to visit all cells. We will try to derive a formula.

Let's assume that $n = 7, m = 7, k = 3$ and we want to draw all the 3×3 squares that cover the cell in $i = 4, j = 3$ (the $4th$ row and $3rd$ column - green cell).



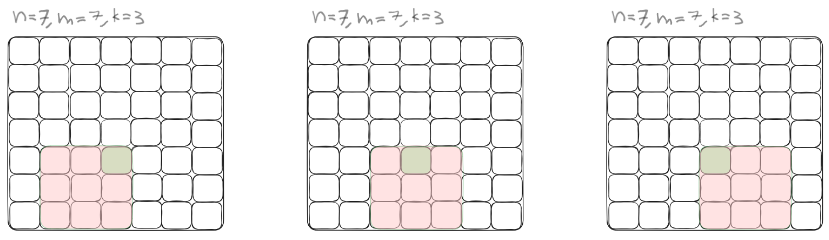
(a) 1st (b) 2nd (c) 3rd

Figure 1: first three visits



(a) 1st (b) 2nd (c) 3rd

Figure 2: second three visits



(a) 1st (b) 2nd (c) 3rd

Figure 3: last three visits

As we can see from the above example, each time the square moves from left to right (meaning that moves along columns - see figure 1), each cell can be visited at most k times. This also holds when it moves along the rows (see the first picture from each figure). Let c be the visits of cell (i, j) when the square grid moves along the columns. We can see that c does not change when the square grid moves down, covers again the (i, j) cell and moves to the right again. From the above example, we can see in the first figure and in the first picture that when the square grid visit (overlaps) the green cell for the first time, it will visit the cell totally k times. This will happen again when the square grid moves down 1 row and visits again the green cell (second figure - second picture).

In the same perspective, we will define the variable r as the times the square grid visits the green cell when it moves along the rows. As previous, this number does not change.

From the above we can say that if we know the numbers c and r, we can say that the number of total visits is $c * r$.

We will show how to calculate only the number c because the other case is similar. So to help ourselves, we can limit it by consider only the number of columns (as $1 \times m$ rectangle) of the rectangle and the number of columns of the square grid as $1 \times k$ rectangle that moves inside the $1 \times m$ rectangle. Check the following figure.

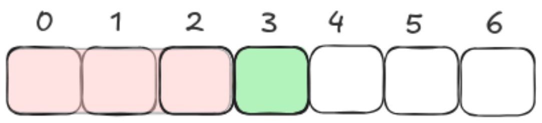


Figure 4: Visualize the process of moving along the columns

Now we have to calculate the times that the 1×3 rectangle will overlap the green cell while moving to the right. Let's define a variable called low that will be the most left $(1, j_1)$ cell from the 1×3 rectangle from the first time that it will overlap the green cell.
On this example it is obvious that $low = j_1 = 1$.

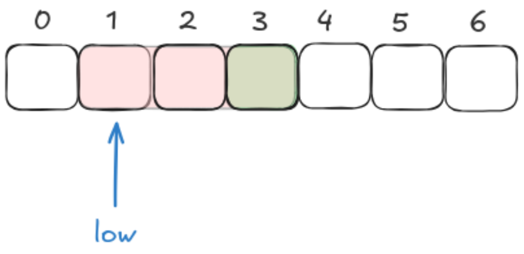


Figure 5: Low variable

Now define the variable high to be again the most left $(1, j_2)$ cell from the 1×3 rectangle, but now from the **last** time that it will overlap the green cell.

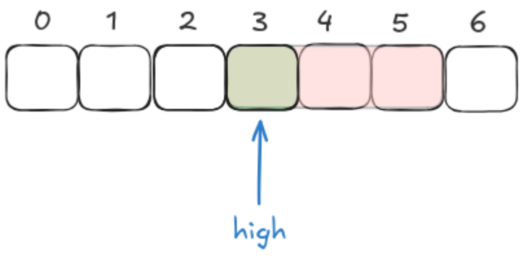


Figure 6: Low variable

Here it is $high = j_2 = 3$.

Now we observe that the number of times that the 1×3 rectangle visits the the green cell is $high - low + 1 = 3 - 1 + 1 = 3$ (actually that's the distance between the two variables; ie., the distance that the rectangle covers). And this is the formula for the general case. We can see that $low = j - k + 1$ ((i, j) are the coordinates of the green cell) and $high = j$. But there are some edge cases we should take care. For example what happens if the green cell is inside the 1×3 rectangle at the beginning of the process.

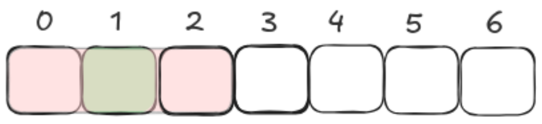


Figure 7: Green cell inside from the start

It is obvious that then, the first cell from the first rectangle is on the zero position. So we have $low = max(0, j - k + 1)$.

Now (on another example) suppose that the green cell is inside the last $k - 1 = 3 - 1 = 2$ cells. That means that the 1×3 rectangle cannot visit the green cell while it is on the first column of it at the end of the process (otherwise it would lead the 1×3 outside of the columns and that's impossible).

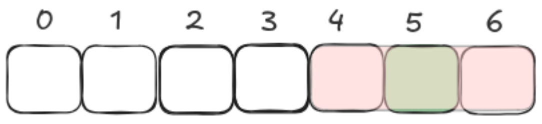


Figure 8: Green cell inside of the last k - 1 cells

On this example, the first cell of the the 1×3 rectangle is on the position $m - k = 7 - 3 = 4$ where m is the number of columns of the initial array. Finally, we have $high = min(j, m - k)$. The code follows.

```cpp
////created by gravitorious
#include <bits/stdc++.h>
using namespace std;

#ifdef XOX
#include "/home/nikos/gravitonlib/mydebug/debug.h"
#else
#define debug(...) 77
#endif

using ll = long long;
using ld = long double;
using uint = unsigned int;
using ull = unsigned long long;

#define getunique(v) {sort(v.begin(), v.end()); v.erase(unique(v.begin(), v
    .end()), v.end());}

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

template <class T>
void chkmax(T &x,T y){
    if(x < y) x = y;
}

template <class T>
void chkmin(T &x,T y){
    if(x > y) x = y;
}

constexpr int popcnt(int x){
    return __builtin_popcount(x);
}

constexpr int ctz(int x){
    return __builtin_ctz(x);
}

constexpr double int_part(double x, double *intpart){
        return modf(x, intpart); //returns the real part
}

ll udiv(ll a, ll b) {
        return a / b + ((a ^ b) > 0 && a % b);
}  // divide a by b rounded up
ll ddiv(ll a, ll b) {
        return a / b - ((a ^ b) < 0 && a % b);
}  // divide a by b rounded down

long long myRand(long long B) {
        //0 to B-1
        return (unsigned long long)rng() % B;
}

int myUniRand(int a, int b){
        //a to b
        uniform_int_distribution<int> distribution(a,b);
        return distribution(rng);
}

void solve(){
        int n, m, k;
        cin >> n >> m >> k;
        int w;
        cin >> w;
        vector<int> h(w);
        for(int i = 0; i < w; i++) cin >> h[i];
        vector<ll> s;
        s.reserve(n * m);
        for(int i = 0; i < n; i++){
                for(int j = 0; j < m; j++){
                        ll cl = max(0, j - k + 1); //low for columns
                        ll ch = min(j, m - k); //high for columns
                        ll c = (ch - cl + 1); //distance
                        ll rl = max(0, i - k + 1); //low for rows
                        ll rh = min(i, n - k); //high for rows
                        ll r = (rh - rl + 1); //distance
                        s.push_back(c * r); //times visited
                }
        }
        sort(h.rbegin(), h.rend());
        sort(s.rbegin(), s.rend());
        ll ans = 0;
        for(int i = 0; i < w; i++){
                ans += h[i] * s[i];
        }
        cout << ans << '\n';
}

int main(){
        ios::sync_with_stdio(false);
        cin.tie(0);
        //freopen("input.txt", "r", stdin);
        //freopen("output.txt", "w", stdout);
        int tc;
        cin >> tc;
        while(tc--) solve();
        return 0;
}
```