USACO MAR14 Problem 'lazy' (Gold) Analysis

by Philip Sun

Notice that for a given patch of grass, the locus of all the points that can access that patch of grass forms a square whose sides have slopes 1 and -1. We first rotate the coordinate system 45 degrees counter-clockwise to make the squares easier to handle. We also modify the squares so that only points on the interior (not the edge) of the square can reach the the grass; this makes the later code simpler. We can move each side of the square "out" .5 units on the plane. However, to keep all coordinates integral, we instead double the coordinates, increasing the fineness of the grid, and then slide the square's edges out by 1. For example, square ABCD with A (2,2) and C(8,8) would first be doubled into A(4,4) and C(16,16). We slide the square's edges out and get A(3,3) C(17,17). Now the problem becomes looking through a group of axes-aligned squares to find which region has the most combined grass.

We solve this by doing a left-to-right sweep. We create a left-to-right sorted list of vertical segments, vert[], from the grass patch squares. Each segment stores its x value, bottom y value, top y value, and amount of grass; x, bot, top, and g respectively. Since sweeping through the right edge of a square is leaving the square and reducing the amount of grass, g is negative for right segments. To keep track of the amount of grass in each y-interval during the sweep, we create a sort of modified segment tree. The ith leaf node of the tree represents the segment from ys[i] to ys[i+1], where ys[] is a sorted array of all the squares' vertices' y values. Each inner node represents the segment formed by joining its two child segments. We can easily store this as a complete binary tree in an array. The root has index 0 and node i has children at 2*i+1 and 2*i+2. During our sweep, a given node i in our tree will tell us the maximum amount of grass Bessie could reach if she is located just right of the sweep line and if her location's y value is within the interval represented by node i. At the start of the sweep, the segment tree is full of zeroes and our answer is zero. In each iteration of a loop, we process the next vertical segment, segment i. In the tree, we must increment everything in the range vert[i].bot .. vert[i].top by vert[i].g and then update the answer by querying the tree's root. Finishing the sweep will give us our final answer.

Note that the only query we ever make is for the root node, the maximum grass over all possible y. This allows us to use lazy propagation to speed up tree updates--if all of the range represented by node i is to be incremented, we don't pass this update to i's children, since we will never query i's children. We store updates to the range in alone[i] and consider grass from child nodes in total[i], with total[i] = alone[i] + max(total[child1], total[child2]). It is well known that at most two nodes in each level will be changed by such a lazy update. For each change, we need to verify that the changed node's ancestors still store the correct data. This leads to O(log^2 N) time per update, so O(N log^2 N) time overall, which is sufficiently fast for N=100000.

Solution code is as follows:

```
#include <fstream>
#include <cstring>
#include <vector>
#include <algorithm>
#define TMAX 540000         // Segtree can't have more than TMAX elements
using namespace std;

struct vert {
    int x, bot, top, g;     // x, y of bottom and top, grass amount
};

ifstream fin("lazy.in");
ofstream fout("lazy.out");
int n, k;

int depth, firstB, numL;     // Root depth is zero
int total[TMAX], alone[TMAX];    // max grass in interval and lazy helper
int startY[TMAX], endY[TMAX];    // the start, end of ith segment


bool vComp(vert a, vert b);
void inc(int ind, int amt, int a, int b);
void fix(int ind);

int main() {
    vector<vert> verts;            // All our vertical segments
    vector<int> ys;                // Stores y to construct segtree with
    fin >> n >> k;
    for (int i = 0; i < n; i++) {
        int gi, xi, yi;
        vert vL, vR;            // The left and right vert segments from this square
        fin >> gi >> xi >> yi;
        int lxi = xi - k, lyi = yi;        // The left point of square
        int rxi = xi + k, ryi = yi;        // The right point of square
        // Lower left after 45 degree spin
        int llxi_r = 2 * (lxi-lyi) - 1, llyi_r = 2 * (lxi+lyi) - 1;
        // Upper right after spin
        int urxi_r = 2 * (rxi-ryi) + 1, uryi_r = 2 * (rxi+ryi) + 1;
        vL.g = gi; vR.g = -gi;            // Add grass on left, remove on right
        vL.x = llxi_r; vR.x = urxi_r;
        vL.bot = llyi_r; vR.bot = llyi_r;
        vL.top = uryi_r; vR.top = uryi_r;
        verts.push_back(vL); verts.push_back(vR);
        ys.push_back(llyi_r); ys.push_back(uryi_r);
    }
    sort(verts.begin(), verts.end(), vComp);
    sort(ys.begin(), ys.end());
    ys.resize(unique(ys.begin(), ys.end()) - ys.begin()); // Remove duplicates and resize

    memset(alone, 0, TMAX * sizeof(int));            // Prepare segtree
    memset(total, 0, TMAX * sizeof(int));
    memset(startY, 0, TMAX * sizeof(int));
    memset(endY, 0, TMAX * sizeof(int));
    numL = ys.size() - 1;                    // Number of leaves
    for (depth = 0; (1 << depth) < numL; depth++) ;
    firstB = (1 << depth) - 1;                // Index of first leaf
    for (int i = 0; i < numL; i++) {            // Get start,end for leaves
        startY[firstB + i] = ys[i];
        endY[firstB + i] = ys[i + 1];
    }
    for (int i = firstB - 1; i >= 0; i--) {            // Get start,end for rest of tree
        startY[i] = min(startY[1 + (i << 1)], startY[2 + (i << 1)]);
        endY[i] = max(endY[1 + (i << 1)], endY[2 + (i << 1)]);
    }

    int ans = 0;
    for (int i = 0; i < n; i++) {
        inc(0, verts[i].g, verts[i].bot, verts[i].top);
        ans = max(ans, total[0]);
    }
    fout << ans << "\n";
    return 0;
}

bool vComp(vert a, vert b) {                // Sort vertical segs so left is first
    return (a.x < b.x);
}

void inc(int ind, int amt, int a, int b) {
    if (startY[ind] >= b || endY[ind] <= a) return;    // a to b is out of range
    if (startY[ind] >= a && endY[ind] <= b) {    // a to b surrounds this seg
        alone[ind] += amt;
        total[ind] += amt;
        fix(ind);                // Make sure parents still have correct data
        return;
    }
    inc(1 + (ind << 1), amt, a, b);            // a to b partially in seg
    inc(2 + (ind << 1), amt, a, b);            // go to children
}

void fix(int ind) {                    // index that was disrupted
    for (int i = (ind - 1) >> 1; i >= 0; i = (i - 1) >> 1)
        total[i] = alone[i] + max(total[1 + (i << 1)], total[2 + (i << 1)]);
}
```