USACO JAN14 Problem skicourse' Analysis

by Nathan Pinsker

An initial insight that helps to solve this problem is that, if we can stamp the course with a square of size B, then we can stamp it with a square of size B-1 as well. This suggests a binary search-like approach; if we can efficiently find whether it is possible to stamp the field with a stamp of size B, then we can solve the problem with logarithmic factor overhead.

Rather than dealing with all the complicated cases that arise from stamping areas on top of each other, we elect to think of the problem backwards in time. What must be true about the course after the last stamp? It must contain a BxB square of one type somewhere. What about after the second-to-last stamp? It must contain a BxB square of one type, *except* when it is covered by our first BxB square. Motivated by this, we stamp squares "backwards": given the desired configuration, we find a BxB square and mark it as stamped. We then consider the course with that stamp removed: the area under it can now be considered either rough or smooth as convenient. We then repeat this procedure until we have either covered the entire course or can find no more BxB squares to stamp. We will fail to stamp the course precisely when we run out of BxB squares to stamp using this method.

This turns out to be fairly inefficient; it takes O((MN)^2 log N) time (with some dynamic programming to speed up checking whether each BxB square is a valid location to stamp). We can improve this slightly by noting that, each time we try to find a BxB square to stamp, we can instead simply find and stamp the largest square remaining in the grid. This gets rid of the logarithmic factor, because we obtain our answer by taking the minimum side length of all the squares we have chosen.

Both of the above solutions actually received full points during the contest. However, we can actually make the O((MN)^2 log N) solution even faster by using a two-dimensional tree structure such as a quadtree, although the details are a little messy.

Here is Bruce Merry's O((MN)^2) solution:

```
#include <fstream>
#include <algorithm>
#include <vector>
#include <string>
#include <cassert>

using namespace std;

static bool isr[101][101];
static bool iss[101][101];

struct rs
{
    int r;
    int s;

    rs() : r(0), s(0) {}
    rs(int r, int s) : r(r), s(s) {}
};

int main()
{
    ifstream in("skicourse.in");
    ofstream out("skicourse.out");
```

```
    int R, C;
    in >> R >> C;
    for (int i = 0; i < R; i++)
    {
        string line;
        in >> line;
        for (int j = 0; j < C; j++)
        {
            if (line[j] == 'R')
                isr[i][j] = true;
            else
                iss[i][j] = true;
        }
    }

    int ans = min(R, C);
    vector<rs> dp, next;
    dp.resize(C + 1);
    next.resize(C + 1);
    while (true)
    {
        int best = -1;
        int bestr = -1, bestc = -1;
        fill(dp.begin(), dp.end(), rs(0, 0));
        for (int r = R - 1; r >= 0; r--)
        {
            for (int c = C - 1; c >= 0; c--)
            {
                rs n;
                n.r = isr[r][c] ? min(min(dp[c].r, dp[c + 1].r), next[c + 1].r)
+ 1 : 0;
                n.s = iss[r][c] ? min(min(dp[c].s, dp[c + 1].s), next[c + 1].s)
+ 1 : 0;
                if (n.r != n.s)
                {
                    int hi = max(n.r, n.s);
                    if (hi > best)
                    {
                        best = hi;
                        bestr = r;
                        bestc = c;
                    }
                }
                next[c] = n;
            }
            next.swap(dp);
        }

        if (best == -1)
            break;

        ans = min(ans, best);
        for (int r = bestr; r < bestr + best; r++)
            for (int c = bestc; c < bestc + best; c++)
            {
                isr[r][c] = true;
                iss[r][c] = true;
            }
    }

    out << ans << '\n';
}
```

And here is his full solution:

```
#include <fstream>
#include <algorithm>
#include <vector>
#include <string>
#include <cassert>
#include <utility>
#include <cstring>
#include <queue>

using namespace std;

typedef pair<int, int> pii;
typedef vector<int> vi;

struct rs
{
    int v[2];
};

static const int bias = 128;
static int tree[2][2 * bias][2 * bias], orig_tree[2][2 * bias][2 * bias];

static vector<int> make_split(int A, int B)
{
    vector<int> ans;
    A += bias;
    B += bias;
    while (A < B)
    {
        if (A & 1)
        {
            ans.push_back(A);
            A++;
        }
        if (B & 1)
        {
            B--;
            ans.push_back(B);
        }
        A /= 2;
        B /= 2;
    }
    return ans;
}

static void findo(int x, int r, int c, vector<pii> &out)
{
    if (tree[x][r][c])
    {
        if (c < bias)
        {
            findo(x, r, 2 * c, out);
            findo(x, r, 2 * c + 1, out);
        }
        else if (r < bias)
        {
            findo(x, 2 * r, c, out);
            findo(x, 2 * r + 1, c, out);
        }
        else
            out.push_back(pii(r, c));
```

```
    }
}

int main()
{
    ifstream in("skicourse.in");
    ofstream out("skicourse.out");

    int R, C;
    in >> R >> C;
    for (int i = 0; i < R; i++)
    {
        string line;
        in >> line;
        for (int j = 0; j < C; j++)
        {
            int idx = (line[j] == 'S');
            tree[idx][i + bias][j + bias] = 1;
        }
    }
    for (int idx = 0; idx < 2; idx++)
        for (int i = 2 * bias - 1; i > 0; i--)
            for (int j = 2 * bias - 1; j > 0; j--)
            {
                if (i < bias)
                    tree[idx][i][j] = tree[idx][2 * i][j] + tree[idx][2 * i +
1][j];
                else if (j < bias)
                    tree[idx][i][j] = tree[idx][i][2 * j] + tree[idx][i][2 * j
+ 1];
            }
    memcpy(orig_tree, tree, sizeof(tree));

    int lo = 1;
    int hi = min(R, C) + 1;
    while (hi - lo > 1)
    {
        memcpy(tree, orig_tree, sizeof(tree));
        int B = (lo + hi) / 2;
        vector<pair<vi, vi> > sites;
        vector<int> rev[2 * bias][2 * bias];
        vector<rs> wait;
        queue<int> active;
        for (int i = 0; i + B <= R; i++)
            for (int j = 0; j + B <= C; j++)
            {
                vi rows = make_split(i, i + B);
                vi cols = make_split(j, j + B);
                rs w = {{0, 0}};
                for (int x = 0; x < 2; x++)
                {
                    for (size_t k = 0; k < rows.size(); k++)
                        for (size_t l = 0; l < cols.size(); l++)
                        {
                            w.v[x] += tree[x][rows[k]][cols[l]];
                            if (x == 0)
                                rev[rows[k]][cols[l]].push_back(sites.size());
                        }
                }
                wait.push_back(w);
                if (w.v[0] == 0 || w.v[1] == 0)
                    active.push(sites.size());
                sites.push_back(make_pair(rows, cols));
```

```
            }

        while (!active.empty())
        {
            int b = active.front();
            active.pop();

            const vi &rows = sites[b].first;
            const vi &cols = sites[b].second;
            for (size_t i = 0; i < rows.size(); i++)
                for (size_t j = 0; j < cols.size(); j++)
                {
                    int r = rows[i];
                    int c = cols[j];
                    for (int x = 0; x < 2; x++)
                    {
                        vector<pii> wipe;
                        findo(x, r, c, wipe);
                        for (size_t k = 0; k < wipe.size(); k++)
                        {
                            for (int p = wipe[k].first; p > 0; p >>= 1)
                                for (int q = wipe[k].second; q > 0; q >>= 1)
                                {
                                    --tree[x][p][q];
                                    for (size_t l = 0; l < rev[p][q].size();
l++)
                                    {
                                        int b2 = rev[p][q][l];
                                        if (--wait[b2].v[x] == 0 &&
wait[b2].v[!x] > 0)
                                            active.push(b2);
                                    }
                                }
                        }
                    }
                }
        }
        if (tree[0][1][1] == 0 && tree[1][1][1] == 0)
            lo = B;
        else
            hi = B;
    }

    out << lo << '\n';
}
```