USACO OPEN09 Problem 'graze2' Analysis

by David Benjamin

First, we note that the order of the cows does not change; if two cows pass each other, it would be more efficient to have them go to each others' final destinations. Renumber the cows in the input so that the cows are sorted.

We wish to spread the cows out as evenly as possible, so the first cow is at stall 1 and the last at stall N. Letting D be (S-1)/(N-1) and R be the remainder, we must have R of the distances between cows be D+1 and the remaining N-R-1 distances are D.

Because distances can only be D or D+1, cow I (I from 1 to N) can only be located at L(I,J) = D*(I-1) + J, where J < N. (In fact, J < I.) Because there are O(N) places available for each cow, we can solve this problem with dynamic programming.

Let C(I,J) be the minimum time needed to position cows 1 through I, with cow I ending up at L(I,J). Then, cow I-1 can only be at L(I-1,J) or L(I-1,J-1), so we have the recurrence

C(I,J) = min(C(I-1,J), C(I-1,J-1)) + abs(P_I - L(I,J))

The final answer is then C(N,R). We can then write an O(N^2) dynamic programming solution. For memory efficiency, we use a sliding window and only maintain two rows of the dp table at the time.

```
#include <assert.h>
#include <stdio.h>
#include <algorithm>
#include <limits.h>
#include <stdlib.h>

const int MAXN = 1500 + 5;
const int MAXS = 1000000 + 5;
const long long INF = INT_MAX;

int n,s;
int orig_pos[MAXN];
int delta;

//long long cost[MAXN][MAXN]; // cost[i][j] to place ith cow at i*delta+j+1
long long cost1[MAXN];
long long cost2[MAXN];
long long *cost = cost1, *cost_old = cost2;
inline void swap_cost() { std::swap(cost, cost_old); }

int single_cow(int cow, int offset) {
    return abs(orig_pos[cow] - (cow*delta + offset+1));
}

int main() {
```

```
    FILE * fin = fopen("graze2.in", "r");
    FILE * fout = fopen("graze2.out", "w");
    assert(fin != NULL); assert(fout != NULL);

    fscanf(fin, "%d %d", &n, &s);
    for (int i=0;i<n;i++)
        fscanf(fin, "%d", orig_pos + i);
    std::sort(orig_pos, orig_pos+n);

    delta = (s-1) / (n-1);
    int off_tot = (s-1) - (n-1)*delta;

    std::fill(cost, cost+n, INF);

    // endpoints must be right
    cost[0] = single_cow(0,0);

    for (int i=1;i<n;i++) {
        swap_cost();
        std::fill(cost, cost+n, INF);
        cost[0] = cost_old[0] + single_cow(i,0);
        for (int j=1;j<=i;j++) {
            cost[j] = std::min(cost_old[j], cost_old[j-1]);
            cost[j] += single_cow(i,j);
        }
    }

    fprintf(fout, "%lld\n", cost[off_tot]);

    return 0;
}
```