Nordic Collegiate Programming Contest NCPC 2006

September 30, 2006

Solution Sketches

Problem A

Shoot-out

Problem author: Øyvind Grotmol

The state-space of this problem consists of all subsets of the cowboys with all possible turns, in total $N \cdot 2^{N-1}$ possibilities or 53248 states for $N = 13$. The cowboy in turn should consider each of his remaining enemies as a possible target and calculate what probability he has of surviving if he hits, and on the basis of that information he will decide on his target. The problem can thus be solved recursively, with dynamic programming or memoizing to achieve acceptable run-time. There's one technical issue, however; when descending from a state you'll get back to the same state due to the possibility of all cowboys missing. This loop of dependencies can be corrected for by dividing the total contributions from all the scenarios where some cowboy actually hits, by $1 - p$ where p is the probability that all cowboys will miss. Care must also be taken to handle the situations in which a cowboy has several equally beneficial targets, in which case he will make a random selection between them.

Problem B

Tour Guide

Problem author: `/dev/duff`

This problem is a generalisation of the *Traveling salesman problem*, where the distance between the points changes depending on the visit order. Basically, this must be solved by brute force, trying all possible orders you can collect the tour guests in. This you do by iterating over all permutations of $(1, \ldots, n)$, and choose the permutation that gives the lowest accumulated time.

In addition, there is a little geometry involved in finding the time needed to catch up with tour guest. Let your position and speed be (x, y) and v, and the retired person be at (x_i, y_i), moving with speed v_i in direction a_i. Then $v_{i_x} = v_i \cos a_i$ and $v_{i_y} = v_i \sin a_i$. To find the time t required to catch up, you solve this second order equation:

$$(x_i - x + v_{i_x} t)^2 + (y_i - y + v_{i_y} t)^2 = (vt)^2$$

Since $v > v_i > 0$, this equation always gives one positive and one negative solution. Choose the positive solution t^+ and continue from the new point $(x_i + v_{i_x} t^+, y_i + v_{i_y} t^+)$ to find out where you will catch up with the next person in your current permutation.

You must compute the time it takes before all your tour guests have arrived at the bus.

Problem C

Nasty Hacks

Problem author: Truls Amundsen Bjørklund

If the expected revenue when advertising minus the cost of advertising is bigger than the expected revenue when not advertising, you should advertise. Each test case is thus solved with an if-else statement.

Problem D

Jezzball

Problem author: Jimmy Mårdell

The problem can easily be divided into two equal subproblems: one where we only consider extending a vertical ray, and one where we only consider extending a horizontal ray. The best result of these two is the final answer. Because of precision issues, it's not possible to simulate extending a ray every second (the first interval possible may be stepped over) or every 1/10000th of a second (time out). Another approach is needed, based on calculating when the atoms cross the extension line. In the outline below we consider extending a vertical ray.

For each atom, calculate when its x-coordinate will intersect the vertical line from the ray origin (the problem constraints guarantees that an atom will not travel strictly horizontal or vertical). This will happen two times before the atoms x-coordinate is repeated. These two intersections are then repeated regularly; calculate the intersection times during the first 10000 seconds. For each such intersection, calculate the y-coordinate the atom will have at that time, and based on that calculate the interval at which this atom would hit an extending ray (based on calculating the total time before the vertical ray has finished extending, and the time it takes to reach the intersection point from the ray origin). For each atom and intersection we get a similar interval. Sort them, and do a linear search among the intervals to find the earliest possible start point. Don't forget to also check whether it's possible to extend a ray at time 0.

The calculations can be made easier by mirroring the playing field horizontally and vertically. Instead of having the atoms bouncing, they will then wrap around. Computations can then be done using modulo calculations.

Problem E

Card Trick

Problem author: Henrik Eriksson

This is a quite simple task. One will typically process one card at a time, starting with the smallest. When one should place card i, one just counts upto i free spaces, and inserts card i in the next free space.

Because n is small, one may also pregenerate solutions to all possible test cases.

Problem F

Traveling Salesman

Problem author: Øyvind Grotmol

This problem may be divided into two steps. The first step is to figure out which countries are neighbors, and the second step is to find the minimum number of borders the salesman must cross in order to get from country c_a to c_b.

To figure out which countries are neighbors is actually quite easy even though it is not made obvious by the problem description. Because no point will be on the line between two connected points and no two non-adjacent edges of a country may share a common point, we know that a country is a neighbor of another if and only if they have a common edge. We may thus hash on the (endpoints pair of the) edges. If two edges are exactly equal, we know that the two countries they belong to are neighbors.

When we know what countries are neighbors, we may find the number of borders that need to be crossed by performing a breadth-first search in the graph.

Problem G

Whac-a-Mole

Problem author: Kristoffer Arnsfelt Hansen

Let $p(x, y, t)$ be the maximum amount of moles you may have whacked if you have your hammer placed at position (x, y) after timestep t where $-d \leq x, y < n + d$. Clearly, $p(x, y, 0) = 0$ because no moles have yet appeared at time 0.

We know that we may end up at position (x, y) after timestep t if we were at position $(x - i, y - j)$ at time $t - 1$ for all i, j where $\sqrt{(x-i)^2 + (y-j)^2} \leq d$. We may also whack all moles appearing in holes centered under a straight line from $(x - i, y - j)$ to (x, y) at time t. The amount of moles this allows us to whack can be expressed as:

$$w(x, y, i, j, t) = m(x, y, t) + \sum_{k=0}^{g_{i,j}-1} m\left(x - i + \frac{k \cdot i}{g_{i,j}},\ \ y - j + \frac{k \cdot j}{g_{i,j}}, t\right)$$

where $g_{ij} = gcd(abs(i), abs(j))$ and $m(x, y, t)$ is 1 if there is a mole appearing at position (x, y) at time t and 0 otherwise. We are now able to express the values in $p(x, y, t)$ mathematically:

$$p(x, y, t) = \begin{cases} 0 & \text{if } t = 0 \\ \max\left(p(x - i, y - j, t - 1) + w(x, y, i, j, t)\right) & \text{else} \end{cases}$$

i and j may vary as explained above. We are now able to fill a table using dynamic programming or memoization, and find the maximum value in the table for the last time step. Notice that we may save some memory by discarding information about all timesteps except the previous one at any given time.

What made this problem particularly tricky was the fact that it can sometimes be profitable to move the hammer outside of the actual board.

Problem H

Random Walking

Problem author: Gunnar Kreitz

Let $p_{i,v}$ be the probability that the i:th number output is v. We have that $p_{1,v} = 1/n$ for all nodes v. The probabilities for succesive steps can be computed by

$$p_{i,v} = \sum_{w} \frac{p_{i-1,w}}{d(w)}$$

where w ranges over all neighbours of v (note that we need to include the same neighbour several times if there are multiple edges between v and w) and $d(w)$ is the degree, i.e. the number of edges incident to w (again making sure that multiple edges are counted).

Once the $p_{i,v}$ are computed, we can compute the probability that a specific bit b is set in a specific timestep i by summing $p_{i,v}$ over all v such that the b:th bit of v is set.

Problem I

Honeycomb Walk

Problem author: Henrik Eriksson

This problem can be solved by dynamic programming. Let $W[x, y, s]$ be the number of walks on s steps starting from tbe coordinates x, y ending up in the origin $x = y = 0$. Then $W[x, y, 0]$ is 1 if $x = y = 0$, and 0 otherwise. You can represent the hexagonal grid with a regular matrix in the following manner: For $s > 0$ we have

$$\begin{aligned} W[x, y, s] &= W[x-1, y-1, s-1] + W[x, y-1, s-1] + W[x+1, y, s-1] + \\ &\quad W[x+1, y+1, s-1] + W[x, y+1, s-1] + W[x-1, y, s-1]. \end{aligned}$$

The number of walks with exactly s steps is in the end stored in $W[0, 0, s]$. The best would be to compute all answers up to the maximum s before even reading any input, but programs generating a new array for each input are also accepted.