ACM ICPC World Finals 2008

Solution sketches

Disclaimer *These are unofficial descriptions of possible ways to solve the problems of the ACM ICPC World Finals 2008. Any error in this text is my error. Should you find such an error, I would be happy to hear about it at* `austrin@kth.se`.

Finally, I want to stress that while I'm the one who has written this document, I do not take credit for the ideas behind these solutions – they come from many different people.

– Per Austrin

Problem A: Air Conditioning Machinery

This problem can be solved using an exhaustive search. At each step, there are at most 8 possible ways of adding a new elbow. Since there are only 6 elbows available, this means that there are at most $\sum_{i=1}^{6} 8^i = 299592$ possible paths in total.

Problem B: Always an Integer

One of the easiest ways of solving this problem is based on the following fact: $P(x)$ is divisible by D for every integer $x \geq 1$ if and only if $P(x)$ is divisible by D for every integer $1 \leq x \leq \deg(P) + 1$.

Here's a sketch of the proof of this fact: another way of saying that D divides $P(x)$ for every $x \geq 1$ is to say that D divides $P(1)$ and that D divides $P(x+1) - P(x)$ for all $x \geq 1$. But $Q(x) := P(x+1) - P(x)$ is a polynomial of strictly smaller degree than P, so by induction, D divides $Q(x)$ for every $x \geq 1$ if and only if D divides $Q(x)$ for every $1 \leq x \leq \deg(Q) + 1 \leq \deg(P)$, so $P(x)$ is divisible by D for every $x \geq 1$ if and only if D divides $P(1)$ and D divides $P(x+1) - P(x)$ for every $1 \leq x \leq \deg(P)$, which is another way of formulating the statement that we are trying to prove.

When checking if $P(x)$ is divisible by D, we only need to compute $P(x) \bmod D$, which means that there is no need to use Big Integers.

Problem C: Conveyor Belt

This problem had two challenging parts, computing the line segments connecting the shafts, and then finding the actual shortest path.

The first part is "just" a pen and paper exercise in geometry (it can also be done by binary search for those who are so inclined).

The second part can be done by brute force, because of the small number of shafts (it may look as though there are 20! possible paths, but because of the non-crossing condition of the path, the actual number of possibilities is a lot smaller). I do not believe that there is a polynomial time solution for this part, but I would love to be proven wrong.

Problem D: The Hare and the Hounds

The hound has two different modes of operation. In its standard mode, it is simply trying to get to the endpoint by following the main road rule. Whenever it gets to a choice point, it switches to the "confirmation mode", in which it is only searching for a confirmation marker (by trying different exits from the choice point intersection, each time following the main road rule until it has become apparent that the wrong exit from the choice point intersection was taken). Once it finds the confirmation marker, it switches back to the standard mode and continues as before.

Problem E: Huffman Codes

This problem can be solved by brute force with some pruning. Essentially, we just try to perform the Huffman tree creation backwards: we start with the entire tree defined by the input, for which we know that the frequency is 100 and then try all 50 possible ways of distributing this frequency into its two subtrees (i.e., $(1, 99)$, $(2, 98)$, and so on, up to $(50, 50)$). We then do the same things for these subtrees, and so on, until all the frequency has been distributed all the way down to the leaves of the tree.

There are two ways to prune this which will make the solution very fast.

The first observation is that, when choosing how to distribute (or split, as I will henceforth call it) the frequencies among the two subtrees of a tree, *both* of the two "subfrequencies" must be smaller (or equal) than the minimum frequency of any previous split. For instance, if the frequencies of the subtrees of the root were chosen as $(40, 60)$, the right subtree could not have $(10, 50)$ as the frequencies of its subtrees. So in this example, there would only be 11 possible ways to split the 60-tree, rather than 30.

The second observation is that, because of the first observation, we should always split the currently active subtree with the largest frequency. In other words, we do not have to try different orders of splitting the subtrees, there will always be a uniquely determined subtree to do the next split on.

Problem F: Glenbow Museum

The problem can be reformulated as follows: count the number of strings consisting of $L/2 + 2$ "R":s and $L/2 - 2$ "O":s such that there are no two adjacent "O":s (where the first and the last positions are considered adjacent). This can be computed in $O(L^2)$ time and memory using dynamic programming.

There is also an $O(1)$ solution to this problem.

Problem G: Net Loss

Given the y coordinate of the intersection of the two line segments, the area to the left of c is simply a quadratic polynomial in a_1, and the area to the right of c is a quadratic polynomial in b_1. Thus, the optimal values of a_1 and b_1 can be easily computed given the y coordinate at $x = c$. Trying "all" possible values for this y coordinate (or using a more sophisticated numeric minimization method) gives the optimum values asked for in the problem.

Problem H: Painter

This was one of the most challenging problems. The problem has two components, both of which can be solved using a sweepline algorithm.

The first phase, to detect errors, can be done by finding intersections between all line segments defining the triangles (where you have to be careful not to count two segments belonging to the same triangle as an error). There is a standard, albeit somewhat complicated, algorithm for this.

The second phase, constructing the tree of contained triangles, can be done as follows: keep a list of all currently active triangles, sorted by the y coordinate of their bottommost edge. Note that this y coordinate changes as the sweepline moves from left to right, but the ordering of the currently active triangles determined by the y coordinate is invariant. The main observation is that, when adding a triangle, the parent of this triangle will be some ancestor of the triangle T immediately below it (where "below" is in the sense of the ordering described above). This ancestor can be found in $O(\log n)$ time.

Problem I: Password Suspects

One possible solution is by dynamic programming: given that we are at position pos in the password, have already matched the subset S of substrings, and the most recent x characters were the first x characters of pattern P, how many ways are there to fill in the rest of the password? Here, it is important that x is as large as possible, i.e., that there is no $y > x$ and pattern Q such that the y most recent letters match the y first letters of Q.

When implemented right, this gives a time complexity of $O(\Sigma \cdot L \cdot N \cdot M \cdot 2^M)$, where L is the maximum length of a word, and Σ is the alphabet size. The optimization needed to achieve this complexity is to precompute what happens when a new character c is added, for all combinations of x, P and c.

Problem J: The Sky is the Limit

This problem can be solved as follows: for each upper edge of a mountain, find all intervals of this edge for which there is an upper edge of another mountain directly above it. What remains after all these intervals have been removed, is the contribution of this particular upper edge to the Banff skyline. Extra care needs to be taken when two upper edges overlap, since you only want to count one of the two overlapping parts as contributing to the skyline.

Problem K: Steam Roller

This problem can be solved using Dijkstra's algorithm. The graph in which we're interested has as vertices fourtuples (r, c, α, d), where

- (r, c) is a position in the city, indicating our position
- α is one of the four possible directions, indicating in which direction we are currently heading
- d is a boolean, indicating whether the cost of the last traversed street was doubled or not (so that we know whether or not to add its cost when turning or stopping)