# ACM ICPC World Finals 2010

## Solution sketches

**Disclaimer** *These are unofficial descriptions of possible ways to solve the problems of the ACM ICPC World Finals 2010. 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`*.*

*Also, note that these sketches are just that—sketches. They are not intended to give a complete solution, but rather to outline some approach that can be used to solve the problem. If some of the terminology or algorithms mentioned below are not familiar to you, your favorite search engine should be able to help.*

*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*

### Mini-analysis

Before the contest, I expected problem B to be easiest followed by C and D and then G and J. When the first solutions started coming in for J then G then D it dawned on me that I was wrong as usual. In the end, I think problems D, G, and J ended up being tied for easiest with 78 teams solving each of them. Problem C was solved by almost the that many teams and problem B by roughly half that. All problems ended up being solved, but problems A, E, and H only by one team each. This was also somewhat surprising to me: I did not believe anyone would solve A, and that H would be solved by several teams. I also believed that Problem I would be solved by more teams, but given that it is a brute-force problem and it is hard to know how much pruning is necessary I guess it is not suprising that many teams were hesitant to spend time on it.

Congratulations to Shanghai Jiaotong University for their third ICPC victory!

### Problem A: APL Lives

This is a somewhat painful exercise in writing a parser. However, the examples given are fairly generous (in particular the case with multiple assignments in the same statement), so there are not really any tricky cases to think of. Being careful is the key.

### Problem B: Barcodes

The main difficulty in this problem appears to have been to categorize the regions as thick or thin. A simple way to do this is to note that the first region is always thin, regardless of whether the code is reversed. Then, everything more than, say, 50% wider than that, must be thick, and the rest of the markings must be thin. Once this classification is done, one can check whether there is a choice of width for the thin regions such that all thin regions are within $\pm 5\%$ of this width and all the thick regions within $\pm 5\%$ of twice that width. Here, it is very important to note that the intended width of a region does not necessarily have to be an integer. An easy way to do the check is as follows: multiply all the thin widths by 2

(to get them on the same scale as the thick widths). Then the widths are OK if the smallest width is at least 95/105 times the largest width.

After this things should be easy. If the second region is thick the code is reversed and we flip direction. Then we simply decode each character (making sure to return `bad code` if we encounter a pattern which does not correspond to one of the valid characters or if the total number of regions is not of the form $6n - 1$), and check whether the $C$ and $K$ checksums are correct.

## Problem C: Tracking Bio-bots

This problem can be solved by the standard trick of coordinate compression. Let us illustrate this using the $8 \times 8$ example grid given in the problem. Note that, with respect to being a stuck square, the $x$-coordinates 4 to 7 are equivalent since no wall starts or ends within this range. Similarly, the $y$-coordinates 0 to 1 are also equivalent. This means that the grid can be compressed to a $7 \times 5$ grid. The compressed grid is equivalent to the old one except that each cell in the compressed grid correspond to potentially many cells in the old grid – the lower right corner in this grid corresponds to the original 8 blocked cells.

In general, when doing this compression, the widths and heights of the compressed grid can be at most $2w + 1 \leq 2001$ which makes it feasible to simply go through the entire grid to find the stuck squares.

## Problem D: Castles

First, note that each castle can be described by two parameters: first the number of soldiers "consumed" when conquering this castle (i.e., the number of soldiers who will die plus the number of soldiers that need to be left there afterwards) and second the number of soldiers needed to attack the castle (i.e., the maximum of the $a$ parameter and the number of soldiers consumed).

Now, let us consider the case when the underlying graph of paths is a complete graph, instead of a tree as in the problem. In this case, one should simply choose the order to conquer the castles greedily, taking the one with largest $(\#\text{attackers} - \#\text{consumed})$ first (with ties broken arbitrarily). This is an intuitively appealing solution and it is not hard to prove that it is optimal.

Finally, when the underlying graph is a tree, as in the problem, the solution is as follows. We try all possible choices of castles as the first castle to conquer, and root the tree at this particular castle. At each castle, we have to decide in which order to conquer the different subtrees rooted at the children. But this we can do using the greedy approach above: we are free to visit the children in any order, and we can think of each subtree as collapsing to a single vertex consuming a certain number of soldiers (simply the sum of the soldiers consumed over all castles in the subtree) and requiring a certain number of attackers (which we can compute recursively).

## Problem E: Channel

This problem has a somewhat standard, but very complicated, dynamic programming solution, making it one of the hardest problems in the set.

The important observation is that, once we have filled in C's in the first $r$ rows, the only thing which matters for how we can complete the channel is the set of C's in row $r$, and *how they are interconnected*.

As an example, consider the following case:

```
.##...
......
#.....
......
```

Now suppose that the first two rows are filled in as follows:

```
C##CCC
CC.C.C
??????
??????
```

We see that the C's in columns 2, 4 and 6 are loose ends, and that the one in column 1 is not. Furthermore, the loose ends in columns 4 and 6 are connected to each other, and the one in column 2 leads to the origin of the channel. In general, the loose ends will always behave like this: there will be one loose end leading to the channel, and the remaining loose ends have to pair up with each other (otherwise there can be no way to complete the picture into a valid channel). Thus, in order to describe the current state, it suffices to describe:

- The set of columns with C's in them.
- Which C is the loose end leading to the origin.
- Which other C's are loose ends and how they pair up.

## Problem F: Contour Mapping

The main difficulty in this problem is to get the contour lines that go along the border of the triangles. Ignoring this part for the time being, we compute the total length of contour lines within each triangle separately and add them up.

To compute the length of contour lines within a triangle where the three vertices have elevations $h_1 \leq h_2 \leq h_3$, a somewhat inefficient way to do it is as follows: for each height $h_1 < z < h_3$ divisible by the parameter $h$, there is a contour line of height $z$ within the triangle. The length of this contour line can be computed using basic trigonometry. However, doing this for all $z$ ends up being too slow, as there can be a million contour lines in each of roughly 10000 triangles. To overcome this, note that the length of a contour line is linear in $z$ in the range $[h_1, h_2]$ and in the range $[h_2, h_3]$, so the total length can be computed by twice using the closed formula for evaluating an arithmetic sum.

To handle the contour lines along the borders of the triangles, one can iterate through all borders and see if there is a contour line along them. This happens if the following two things happen:

- The two heights at the endpoints are equal and a multiple of $h$.

- At least one of the two other points of the two triangles that share the border have a height different from the height of the border (if the border is along the boundary of the triangulation this automatically happens).

## Problem G: The Islands

This problem can be solved with dynamic programming. First, instead of thinking of one tour that goes west and then back east, let us think of it as two separate paths going from west to east at the same time (with every island being visited by exactly one of the two paths). This is of course equivalent but the change of viewpoint makes things a lot easier.

We now let $L(i,j)$ denote the minimum length of two paths starting at islands $i$ and $j$ respectively, together covering all the islands from $\max(i,j)+1$ to $n-1$ and both ending at $n-1$. Letting $k = \max(i,j)+1$ and ignoring the two special islands $b_1$ and $b_2$, a recursion for $L$ is the following:

$$L(i,j) = \begin{cases} \min\left(d(i,k)+L(k,j), d(j,k)+L(i,k)\right) & \text{if } k < n \\ d(i,n-1)+d(j,n-1) & \text{if } k = n \end{cases},$$

where $d(i,k)$ denotes the distance between islands $i$ and $k$. The second line is the base case of the recursion, and the two different options in the first line correspond to the choice of which of the two tours should pass island $k$. With the addition of the two special islands, the only difference is that when $k = b_1$ the value $d(j,k)+L(i,k)$ is not an option since only the first tour can pass island $k$, and similarly when $k = b_2$ the value $d(i,k)+L(k,j)$ is not an option.

To reconstruct the optimal solution, the standard method of backtracking can be used.

## Problem H: Rain

This is the second problem in the set involving a triangulated surface with elevations, but apart from that similarity this problem is quite different.

A conceptually simple approach is the following. First identify all regions (i.e., the triangles and the boundary). Then build a graph where two regions are connected if they share a side. Now, using e.g., Dijkstra's algorithm, compute for each triangle, the smallest possible maximum height along any path to the boundary (where the height of an edge is the minimum of the two endpoints of the shared side corresponding to the edge). Finally, we can reconstruct the lakes as follows: each triangle where the smallest maximum height is larger than the minimum depth of the three points of the triangle must be part of a lake having a depth equal to the smallest maximum height. Doing a DFS or BFS from this triangle finds us all the triangles that are part of this lake. There are a few steps but none of the steps are very difficult.

## Problem I: Robots on Ice

This problem can be solved by generating all paths via exhaustive search, with some pruning to discover when a partial path can never lead to a complete path. One particular set of prunings that is sufficient to make the search fast enough is the following (there are probably even simpler ones than these that are sufficient):

- If the distance to the next check-in point is greater than the number of time-steps left to that check-in time, the current partial path can not be extended to a complete path.
- If the set of unvisited cells form two disconnected regions, there are no solutions, the current partial path can not be extended to a complete path.
- A check-point should never be taken ahead of time.

## Problem J: Sharing Chocolate

This problem can be solved with dynamic programming. A good state is the following: given a subset $S \subseteq [n]$ of friends and a desired width $w$, is it possible to split some chocolate bar of width $w$ among the friends in $S$? Note that the height $h$ of such a chocolate bar must necessarily be the sum of the demands of the friends in $S$ divided by $w$ (and if this is not an integer the answer must be "No").

To compute whether a state $(S, w)$ has answer yes, we iterate over all non-empty proper subsets $0 \neq T \subset S$. For each such $T$, we try cutting the $w \times h$ bar either vertically or horizontally, putting $T$ in one part and $S \setminus T$ in the other part. The position to cut the bar is determined by the set $T$ (and some $T$ can make the cut impossible, e.g., if the sum of demands in $T$ is not a multiple of $w$ then no horizontal cut can be made).

## Problem K: Paperweight

With a basic familiarity with geometry, this problem is relatively straight forward, though there is one notable pitfall. The general idea is to simply try all $\binom{5}{3} = 10$ possible triples of points that can form the base plane of the paper weight, and for each such weight check whether it is valid and then check the distance of the led from that plane.

To check whether a position is valid, one needs to verify that no part of the paperweight lies below the chosen base plane, and that the center of mass, when projected onto the base plane, lies inside the base plane within a margin of 0.2 units. In this part we find what is probably the most tricky case: it can happen that a fourth point of the tetrahedron lies in the base plane, meaning that the projection of the center of mass should lie in the convex hull of the four points that lie in the base plane.