CCC 2023 Solutions

J5

Many approaches that work.

I chose to enumerate all the turning points.

Important that you don’t write something like 16 if statements. Have an array for the possible directions instead.

S1

First, count the number of black triangles and multiply it by 3.

Some edges will be counted that are not on the perimeter, so find the number of pairs of black triangles that share an edge and subtract 2 from the answer for each.

There are 2 cases for when two triangles share a side:

- They are beside each other
- They are above each other at an even index (assuming you 0-index)

Read the input into two arrays and count the above with a few for loops.

S2

For each length i from 1 to N, we want to find the subarray with length i that has the smallest asymmetric value.

The asymmetric value of a subarray is the sum of absolute differences between elements that are equidistant from the center of the subarray.

Note that N <= 5000, so we can enumerate all O(N^2) subarrays, we just need to be able to calculate the value of each subarray in O(1) time.

Fixing the left end point of the subarray and iterating the right doesn't work because the positions that get paired up changes every iteration (so complexity would be O(N^3)).

Instead, we fix the center of the subarray and iterate outwards.

Even and odd length subarrays might have to be handled slightly separately.

S3

Given integers N, M, R, and C, construct an N by M grid of letters such that exactly R rows and C columns are palindromes.

First 2 subtasks are trivial.

For full solution, try doing something very simple and see why it doesn't work.

Let's start by saying all our palindromes will consist of one letter repeated, e.g A.

We can set first R rows to all As and first C columns to all As. Now need to make sure rest isn't palindrome.

Cleanest is to set an (N-1) by (M-1) grid to all As, the corner to C, and the first element in each row/col to A/B to exactly control what is a palindrome.

Remaining cases are if R = N or C = M. If both true then trivial. If C = M, swap N with M and R with C. Only case now is R = N.

Use the last row to control which columns are palindromes, some cases are impossible.

S4

First subtask is template MST.

For second subtask, only consider the best edge between a pair of cities. That is, compare length first then tie-break by cost. Might not actually have to implement, but good to know for reasoning that we don't have to account for multi-edges after this point.

Also note that N, M <= 2000.

If there is an edge of length d from a to b, and removing the edge increases dis(a, b), the edge def needs to be kept.

If it does not, we can remove it (proof not trivial).

Full solution is to first process zero weight edges as you do with Kruskal's and merge nodes reachable from each other with zero weight edges. Then find the best edge between each node in the compressed graph. Then check for each edge whether removing it increases dist(a, b) (use Dijkstra's), if it does, add its cost to the answer.

S4 (alternative solution)

Sort all edges by distance, tie-breaking by cost. Start with an empty graph, and for each edge, assuming it has length d and connects nodes a and b, add it to the graph iff dist(a, b) > d.

Similar to Kruskal’s algorithm.

Cleaner to implement since it doesn’t have to do anything special to deal with zero weight edges.

S5

First subtask, solve recursively.

Second subtask, notice that a number is in the Cantor set iff when you write it in base 3, any 1 digit it contains is followed by only 0s.

Check this for all N numbers by performing long division and breaking if a 1 digit is found and there is still a remainder, or when the same remainder is encountered twice (use a bitset to store which remainders have been checked).

Third subtask, use the recursive approach from subtask 1 on a power of 3 to filter the first few layers (e.g first 18 layers). Then use the approach from subtask to check each remaining number.

Optimal score distributions

I predict CCO cutoff will be 51 or 52 based on the unofficial scoreboard.

Ideal strategy for getting CCO would have been to get 15/15/15/3/3 then try to get anything else in the remaining time.

15/15/8/3/3 is also somewhat locally optimal.

I feel this CCC was reasonably speed based. Easiest way to solve the problems (at least last 3) is to go subtask by subtask. This is because subtasks are very useful for the full solution in this contest specifically. Most individual subtasks weren’t too hard.

Solution source codes

https://github.com/bci-csclub/comp-prog-sols