Solutions

If you had a rejected solution and want to find out where you went wrong, read on and download the official input, output, and solutions!

Download Input / Output / Solution Files

The problems in this round were authored by Jacob Plachta, with additional problem statement writing by Alex Li and Wesley May.

Test data was prepared by Wesley May and Jacob Plachta.

Additional testing was contributed by Ahmed Abdellah, Alberto Alfarano, Yan Soares Couto, Vishwa Gayasen, David Harmeyer, Luis Giro Valdes, and Zichen Zheng

Solutions were written by Jacob Plachta, with additional editing by Wesley May.

A1: Consistency - Chapter 1

There are only 26 possible consistent strings which S might be changed into, with all characters in S ending up equal to some character x ($\texttt{"A"} \le x \le \texttt{"Z"}$). We'll consider each possible value of x and compute the total number of seconds required to change all characters in S into x, with the minimum of those 26 possible totals being our final answer.

For each character x, we'll need to consider each character c in S and compute the number of seconds required to change c into x. If $c = x$, then 0 seconds are required. If c is a vowel while x is a consonant (or vice versa), then 1 second is required. Otherwise (that is, c and x differ but are either both vowels or both consonants), 2 seconds are required.

See David Harmeyer's solution video here.

A2: Consistency - Chapter 2

As in Chapter 1, we'll consider each possible character x ($\texttt{"A"} \le x \le \texttt{"Z"}$) such that all characters in S are to be changed into x. The only difference is in computing the minimum number of seconds it takes to change some character c into another character x (if it's possible at all).

If we imagine an unweighted graph with 26 nodes (one for each letter from "A" to "Z"), such that there's a directed edge from node A_i to node B_i for each type of replacement i, then the minimum number of seconds required to change c into x is equal to the shortest length of any path from node c to node x (if any such path exists).

We can precompute a $26 * 26$ table of these shortest distances between all pairs of nodes by using the Floyd–Warshall algorithm or by performing a BFS (breadth-first search) from each possible starting node, taking care to mark unreachable pairs of nodes as effectively having an infinite distance and being unusable in the final answer. For each character x and for each character c in S, we can then look up the shortest distance from node c to node x in the table.

See David Harmeyer's solution video here.

B: Xs and Os

For any given row or column, if it has no **O**s and has k empty spaces, then you can win by placing k **X**s in it (filling in the empty spaces). This covers all possible optimal ways in which you might win, as you should never place additional **X**s which don't directly contribute to winning in a single row or column. The first part of the answer is therefore the minimum value of k across all such rows and columns, and it's impossible to win if there are none.

If it's possible to win, then the second part of the answer is equal to the number of distinct sets of empty cells (in which **X**s will be placed) in those minimum-k rows/columns. If the minimum k is at least 2, this is simply equal to the number of such rows/columns, as no two of them can correspond to the same set of empty cells. On the other hand, if the minimum k is 1, then the second part of the answer may be smaller than that row/column count, as filling a row and filling a column might both involve placing an **X** in the same cell as one another. Therefore, for this case, we need to compute the set of distinct single empty cells found in all rows/columns which have $k = 1$, with the second part of the answer then being the size of that set.

See David Harmeyer's solution video here.

C1: Gold Mine - Chapter 1

The mine can be thought of as a rooted tree, with one node per cave, the root being node 1, and each tunnel corresponding to an edge.

Assuming $N > 1$, Minerva's route will consist of driving along the path from node 1 to some other node a, drilling a tunnel to some other node b (potentially with $b = 1$), and finally driving along the path from node b to node 1. Any given node pair (a, b) is valid if and only if no undirected edge is traversed multiple times (in other words, the path from node 1 to node a shares no edges in common with the path from node b to node 1). For such a path, the total weight of gold ore collected is equal to the sum of C values across all nodes which are part of at least one of the two paths.

We can observe that a choice of (a, b) is valid if and only if either $b = 1$, or a and b lie within the subtrees of different children of node 1. If they were to both lie within the subtree of the same child c of node 1, then both paths would include the undirected edge between nodes 1 and c, which would be invalid.

We can also observe that the only node which will be part of both paths is node 1, and that the direction of each path is irrelevant (with both paths essentially being interchangeable). If we let $F(i)$ be the sum of C values on the path between node 1 and node i, then the total weight of gold ore collected will be $F(a) + F(b) - C_1$.

Let $G(i)$ be the maximum value of $F(j)$ for any node j in node i's subtree (including i itself). We can recursively compute the F and G values for all N nodes in $O(N)$ time, based on the recurrences $F(i) = F(P_i) + C_i$ (where P_i is node i's parent) and $G(i)$ equalling the maximum of $F(i)$ and of $G(c)$ (for each of node i's children c).

Putting everything together, if node 1 has fewer than two children, the final answer will be $G(1)$, while the answer will otherwise be $G(c_1) + G(c_2) - C_1$, where c_1 and c_2 are the two distinct children of node 1 which yield the largest G values. It's also possible to arrive at a similar result without explicitly computing F and G separately.

See David Harmeyer's solution video here.

C2: Gold Mine - Chapter 2

As in Chapter 1, the mine can be thought of as a rooted tree, with one node per cave, the root being node 1, and each tunnel corresponding to an edge.

If $K = 0$, the answer is C_1. Otherwise, we can think of Minerva's route as consisting of any set of K edge-disjoint paths in the tree (that is, with no edge shared between multiple paths), with at least one of them including node 1. A path including node 1 will implicitly correspond to both the first and last portions of her route: we can imagine her starting at node 1 and driving to one of the path's endpoints before drilling her first tunnel, and then eventually drilling her final tunnel to the path's other endpoint and driving back to node 1. The remaining $K - 1$ paths more directly correspond to the intermediate portions of her route, with her repeatedly drilling to a path's endpoint and then driving along that complete path before drilling another tunnel to the next path. The total value of such a set of paths is equal to the sum of C values across all nodes which are part of at least one path, and we want to find a set of paths which maximizes this total.

We'll take a dynamic programming (DP) approach. Let $F(i, j, k)$ be the maximum total value of paths if only node i's subtree is considered, such that:

- There are j paths contained entirely within that subtree
- There's a path with an endpoint at node i (potentially connecting to i's parent) available to be extended further if and only if $k = 1$ (defined for $0 \le k \le 1$)

We'll additionally enforce that, if $i = 1$, at least one of the paths must include node i. Our final answer will be the maximum of C_1 and of $F(1, j, 0)$ (for $1 \le j \le K$).

Computing $F(i, j, k)$ will require processing i and its children in a secondary level of DP. Let $G(i, f, j, k, c)$ be the maximum total value of paths if only node i and its first f children are considered, such that:

- There are j paths contained entirely within those nodes
- There's a path with an endpoint at node i (potentially connecting to i's parent) available to be extended further if and only if $k = 1$ (defined for $0 \le k \le 1$)
- There's at least one path connecting i to one of those f children if and only if $c = 1$ (defined for $0 \le c \le 1$)

We'll recurse through the tree, computing these values. Once we've computed the F values for all children of a node i, we'll be able to compute node i's G values. After starting with the base case of $G(i, 0, 0, 0, 0) = 0$, we can compute $G(f, ...)$ for each fth child of node i by combining $G(f - 1, ...)$ with that child's F values, while considering the options of either having or not having a path connecting node i and that child. If node i has Z_i children, we can then transform $G(i, Z_i, ...)$ into $F(i, ...)$. Further details around the DP transitions involved can be found in the official solution's source code.

Overall, it takes a total of $O(NK^2)$ time to compute all $O(NK)$ relevant DP values.

See David Harmeyer's solution video here.