RMI 2021, Day 1, English Editorial

RMI Scientific Committee

Saturday 23rd October, 2021

Problem 1: Gardening

(Proposed by *Tamio-Vesa Nakajima*.)

Observation 1. *Consider a region that is $2 \times (2k+1)$ cells. Suppose that the first row contains only flowers of type t, and the second row contains t on the first and last cell. Call such a region a* $(2k+1)$-cup. *Such a region cannot exist for $k > 0$.*

Proof. This can be shown by induction. For $2k+1 = 3$, this can be checked immediately. For $2k+1 > 3$, note that in the second row of the $(2k+1)$-cup contains an odd sequence, say of length $2k'+1 < 2k+1$, containing one type of flower. If no third row exists, then clearly the situation is impossible. otherwise, the odd length sequence determines a $(2k'+1)$-cup, which cannot exist. □

Corollary 1. *Thus no $N \times M$ garden can exist when N or M are odd.*

Proof. Consider M odd. Create an $(N+2) \times (M+2)$ garden by adding a new type of flower around the old garden. Then the first two rows form a $(M+2)$-cup, which cannot exist as $M+2$ is odd. □

Corollary 2. *Thus for a garden to be possible, we must have N and M even.*

Thus, from here on assume N, M are even.

Observation 2. *Any $N \times M$ garden satisfies $K \leq NM/4$, and $K \neq NM/4 - 1$.*

Proof. Any type of flower must occupy at least 4 cells, and if it occupies more, it must occupy at least 12 cells. This can checked by exhaustion. Thus clearly we must use at most $NM/4$ flower types. Furthermore, if we were to remove flowers, one flower must take up more than 4 cells; it must thus take up at last 12 cells, and we can use at most $NM/4-2$ flower. So $K \neq NM/4-1$. □

Observation 3. *For any row in a garden, there exists a flower type appearing within that row that appears only at that row and below, or at that row and above.*

Proof. Say that a segment on this row is *closed* if it contains all the appearances within that row of all the flower types within that segment. We prove by induction that any closed segment contains a flower type that appears only at this row and above/below it. The base case is a segment of length 2, which is obvious: such a segment corresponds to a 2×2 square with only one flower type. Now consider longer segments. Suppose a flower type appeared only in one continuous subsegment of our closed segment. It then follows that the region bounded by that flower type does not intersect the row – which implies that the flower type only appears at the row and on one side of it, as required. Otherwise consider any flower type that appears as at least two continuous subsegments of our segment. Then th interior of the region bounded by this flower type intersects our segment. Take any segment of the intersection of this interior with our segment – this is also a smaller closed subsegment. Thus we have completed our inductive argument, as this smaller closed subsegment must contain a flower type that appears at this row and only above/below it.

Note also that the entire row is a closed segment. Thus is must contain at least one color that appears only at the row and above/below it. □

Observation 4. *For any $N \times M$ garden, we have $K \geq \max(N/2, M/2)$.*

Proof. Without loss of generality let $M \leq N$. Thus we want to prove that $K = N/2$. Suppose $K < N/2$. Consider the top-most and the bottom-most row where each flower type appears. There are $2K < 2N$ such rows, so for at least one row all flowers that appear have appeared both above and below it. But this is impossible by the previous observation. So $K \geq N/2$. □

Observation 5. *For an $N \times N$ garden, $K \neq N/2 + 1$.*

Proof. Suppose $k = N/2 + 1$. Consider like in the previous observation the rows where each flower type first and last appears. Clearly for at least one row there are two flower types that either first or last appear at that row. Furthermore, because each row contains at least one flower that appears first/last at that row, we can deduce that exactly two rows have exactly two flowers that appear first/last at that row.

We now investigate the structure of the garden more fully. Suppose that for exactly the first R rows we have had exactly one flower type appear. Then no flowers can appear last at row $R+1$ unless $R = N/2$ (since the "flower type in the middle" is the only that can stop appearing, and it can only stop appearing when it has width 2, i.e. at $R = N/2$). Thus at row $R+1$ we must have exactly two flower types appearing. If both of these had width greater than 2, then at the next row we would be forced to have again two flower types appearing, but this would mean that there are too many flowers, so this cannot happen. So one of the flower types has width two, and disappears on the next row. All other rows have exactly one appearance or disappearance; and in fact all the rows from $R+2$ up to $N/2$ have one appearing flower, and all those left after $N/2$ have exactly one disappearing flower. We can thus deduce that $(N-2R) \times (N-2R)$ that is between rows and column $R+1$ and $N-R$ has a 2×2 corner of one type, and otherwise is comprised of concentric flower types. Furthermore, it must have $N/2 + R$ colors. This is impossible. □

Observation 6. *For any even N, M, a garden exists for any K where $\max(N/2, M/2) \leq K \leq NM/4$, $K \neq NM/4 - 1$, and either $N \neq M$ or $K \neq N/2 - 1$.*

Proof. We provide a recursive procedure that achieves this:

1. If $K = NM/4$ then fill the entire garden with 2×2 areas with one flower type each.

2. Suppose $NM/4 - N - M + 1 \leq K < NM/4 - 1$. Then there exists a rectangle boundary of appropriate size, with one corner in one of the corners of the garden, that if planted with one flower type, and the rest of the garden filled with 2×2 areas with one flower type each, gives us a garden with K flower types.

3. Suppose $K = NM/4 - N - M$. Suppose without loss of generality that $N \leq M$. If $N \leq 4$ then $K \leq \max(N/2, M/2)$ which is impossible – so $N, M \geq 6$. First create a $N \times (M-2)$ rectangle filled border with one flower type, with its corner in the corner of the garden; then within that rectangle create a 4×4 rectangle border, filled with one flower type. Fill the rest of the flower with 2×2 regions each with one flower type.

4. Suppose $K < NM/4 - N - M$. Then it can be proved that we can fill the border of the garden with one flower type, and recursively find the answer for $N-2, M-2, K-1$.

With a careful implementation this can be done in $O(NM)$. □

Problem 2: Present

(Proposed by *Tamio-Vesa Nakajima.*)

Say that a set is gcd-closed if it contains the greatest common divisor of all pairs of its elements. Let $\mathcal{G}$ be the set of (finite) gcd-closed sets. The problem asks for the K-th gcd-closed set in *Laikan* order. More usually, *Laikan* order is called colexicographic order; if A is before B in Laikan/colexicographic order, write $A \preceq B$. Furthermore, we define an "opposite" of the gcd-closed sets. Call these gcd-independent sets: a set S is gcd-independent if and only if for all $x, y \in S$ we have $\gcd(x,y) \notin S$. Let $\mathcal{I}$ be the set of (finite) gcd-independent sets.

Note that for any collection of sets $\{S_i\}_{i\in I} \subseteq \mathcal{G}$ we can deduce that $S = \bigcap_{i\in I} S_i \in \mathcal{G}$. This is because if $x, y \in S$ then $x, y \in S_i$ for all $i \in I$, so $\gcd(x,y) \in S_i$ for all $i \in I$, so $\gcd(x,y) \in S$. Thus we can state the following definition.

Definition 1. For any set $S \subseteq \mathbb{N}$, let $\langle S \rangle \in \mathcal{G}$ be the smallest superset of S that is gcd-closed, i.e.

$$\langle S \rangle = \bigcap_{S \subseteq T \in \mathcal{G}} T.$$

We call this the gcd-closed set *generated* by S.

Observation 7. *For any $S \in \mathcal{G}$ there exists a unique $T \in \mathcal{I}$ such that $S = \langle T \rangle$.*

Proof. To find T, start with $T = \emptyset$. While $S \neq \emptyset$, remove $x = \max S$ from S, add x to T, then remove $\gcd(x,y)$ for all $y \in T$ from S. T remains gcd-independent throughout, and $\langle T \rangle = S$. Furthermore, each step is in a sense "forced" (the maximal element must always belong to T for instance), so T is unique. □

Observation 8. *For any $S, T \in \mathcal{I}$, we have $S \preceq T$ iff. $\langle S \rangle \preceq \langle T \rangle$.*

Proof. Left as an exercise; one direction can be shown using the procedure from the previous observation. □

These facts essentially show that $\langle \cdot \rangle$ is an order-preserving bijection between $\mathcal{I}$ and $\mathcal{G}$. Thus finding the K-th gcd-closed set is in fact equivalent to finding the K-th gcd-independent set, and generating a gcd-closed set from that. An interesting subproblem is finding the next gcd-independent set in colexicographic order.

Observation 9. *Given some $S \in \mathcal{I}$, the following procedure can compute the smallest $T \in \mathcal{I}$ that is greater than S (in colexicographic order):*

1. *Let x be the smallest integer* not *in $\langle S \rangle$.*
2. *Add x to S, and remove all elements less than x.*
3. *The resulting set is T.*

In order to make repeatedly computing the next gcd-independent set in colexicographic order efficient, one must also maintain the gcd-closed sets generated by these gcd-independent sets. To do this, maintain the elements of the current gcd-independent sets on a stack together with the elements in the generated set added because of the current elements. Removing small elements from S is then equivalent to popping the stack. To push x to the stack, we must find all the elements that are not currently in the generated set that must be added as a consequence of adding x. This can be done by iterating over the divisors d of x (that are not currently in the generated set) and checking if some element y is in the set for which $\gcd(x,y) = d$. All of this can be made efficient by precomputing $\{y \mid \gcd(x,y) = d\}$ for all pairs (x,d) that can appear in the problem (it can be proved that $x, d \leq 40$). Furthermore, as an implementation note, it is much more efficient to represent sets as bitmasks.

We have now found a procedure that can, for a gcd-independent set, efficiently find the next gcd-independent set in colexicographic order. To solve our problem, precompute the 0-th, 10^7-th, $(2 \cdot 10^7)$-th, etc., gcd-independent sets, storing them in the source code for the problem. Then, on run-time, starting from one of the precomputed gcd-indpendent sets, iterate to the desired one. Recall of course that finding the K-th gcd-independent set is equivalent to finding the K-th gcd-closed set.

Problem 3: Speedrun

(Proposed by *Matei Tinca*.)

Subtask 1. For each node, we can just hold it's line in the adjacency matrix. That is, for node x, we store a string of length N, where the i-th bit is 1 if there is an edge between x and i. To walk through every node, we can just run a dfs algorithm, because for each node we have the list of its neighbours.

Subtask 2. Since the graph is a star, we have a central node which is connected to every other node. We can label every node with the index of the central node written in binary form. For instance, if the central node is 13, we can store 0000001101 in every node (although, if we hold the bits in reverse order, the implementation would be easier). Therefore, we know the structure of the entire tree from every node, so we can run a dfs algorithm to go through every node. Also, since we have 20 bits, we only use the first 10, which are enough, since they can store every number from 0 to $2^{10} - 1 = 1023$.

Subtask 3. Since the graph is a path, for each node we can store its neighbours, or 0 if the node is an endpoint. To go through every node, we go to the left as much as we can, and after hitting the endpoint, we can go to the right until we hit the other endpoint.

Subtask 4. In this subtask we must exploit the fact that we have a lot of possible calls and big labels. For this, we must observe that there are at most $\sqrt{N}$ nodes which have a degree higher than $\sqrt{N}$. So, for this subtask, for the nodes with less than $\sqrt{N}$ neighbours, we can just hold its adjacency list. This will use at most $\sqrt{N} \log N$ bits. For the other nodes, we don't need to store anything, and we can just try every other node. In the end, each node will use at most $\sqrt{N} \log N$, and we do at most $N\sqrt{N}$ wrong calls.

Subtask 5. In this solution, we use labels of size $3 \log N$ and we make no wrong calls. After rooting the tree in one node, for instance node 1, for each node, we can hold its father, one of its sons, and a link to one of it's "sibling". Therefore, for a node we can discover all of its neighbours: the father, one of the sons, and by going from "sibling" to "sibling" from that son, we can discover every son of that node. Keep in mind, that the sons of a node have to be linked in a cycle so that we can discover every node.

Subtask 5. In this solution, we use labels of size $2N$ and we make at most $2N$ wrong calls. We can run a depth-first search (DFS) from the root, and whenever we visit an unvisited node, we push it to a stack. After the DFS ends, the stack will contain the DFS-walk of the graph. For each node, we are going to hold its father and the next node to it in the DFS-walk (we are considering the DFS-walk to be cyclic, after the last element we have the first element of the stack). To walk through the entire graph, when we are in a node, we try to go through the next node in the DFS-walk. If the function fails, we go to its father, and try again. If it doesn't work again, we go to its father and try again and do this until it works.