2024 KAIST 14th ICPC Mock Competition Editorial

Hosted By:

Sponsored By:

Problem A. Automata Embedding

Problem Idea: Kijun Shin (sharaelong)
Preparation: Kijun Shin (sharaelong)

Solution

First, one can make the following observation:

- For three integers p, q, r and a failure function f such that $f(p) < f(q) < f(r) < p < q < r$, embedding is impossible.

This fact can be proved easily. Placing those 3 arrows in the plane is impossible!
At this point, it becomes clear that this necessary condition for an embeddable string is quite strong. Assume there exists an index $i \neq 0$ such that $f(i) = 0$. Since f can increase by at most 1 per index, there is no $j > i$ with $f(j) = 2$. It follows that there are only a few cases for embeddable strings.
Assume $n \geq 4$. Let us consider the possible pairs for $(f(2), f(3))$:

1. $(f(2), f(3)) = (0, 0)$. In this case, the first 3 letters look like `abc`. Also, $f(i) \leq 1$ must always hold. It means that whenever the letter `a` appears somewhere else, the next letter cannot be `b`. In summary, in this case, the value of f oscillates between 0 and 1, where the first two letters are different (case 1).

2. $(f(2), f(3)) = (0, 1)$. In this case, the first 3 letters look like `aba`. It is possible for f to become larger when the string is `ababab...`, but once it deviates from this pattern for the first time, f drops immediately to 0 or 1. When f drops to 0, by the above observation, the remaining strings fall into case 1.

   Otherwise, if the pattern breaks at an odd length (e.g., `ababaa`), f drops to 1. After that, f oscillates between 0 and 1, or follows a $1 \to 1 \to \cdots \to 2 \to$ (0/1 oscillation) pattern. (When f reaches 2, it cannot become 3 due to the observation, and since $S[1] \neq S[2]$, $f(j) = 2$ and $f(j+1) = 2$ cannot both hold.) In summary, this is case 2, which includes case 1 as part of its pattern.

3. $(f(2), f(3)) = (1, 0)$. In this case, the first 3 letters look like `aab`. Similar to case 1, f should oscillate between 0 and 1, but here $S[1] = S[2]$. We refer to this as case 3.

4. $(f(2), f(3)) = (1, 1)$. This is impossible for any string.

5. $(f(2), f(3)) = (1, 2)$. It means that the first 3 letters are all the same: `aaa`. Similar to case 2, f can become larger, but once a letter other than `a` appears, f immediately drops to 0. In summary, this is case 4, which includes case 3 as part of its pattern.

These 4 cases can be written as 4 recurrence relations that refer to each other (specifically, they can be expressed as a $\mathbb{Z}_+^4$ vector, where the transition matrix is 4×4 and all entries are constant coefficients). Starting with $n = 3$ as the initial term will provide the answer.

Problem B. Construct a Coin Set

Problem Idea: Eunjae Jin (ai4youej)
Preparation: Eunjae Jin (ai4youej)

Solution

For $N \leq 5$, all possible coin sets cannot obtain a minimal counterexample at N.
There are two main solutions to construct the answer for $N \geq 6$.

Solution 1

If $N = 2k$, $(1, k, 2k-2)$ is the answer. $(k \geq 3)$
If $N = 2k+1$, $(1, k, k+1, 2k-1)$ is the answer. $(k \geq 3)$

Solution 2

$(1, 3, N-3, N-2)$ can be the answer.
Notice that for $N = 6$, you have to print $(1, 3, 4)$.

Note

For those who are interested in the special judge for this problem, refer to this:

1. Pearson, David. "A polynomial-time algorithm for the change-making problem." Operations Research Letters 33.3 (2005): 231-234.

Problem C. Counting Regions

Problem Idea: Geunyoung Jang (azberjibiou)
Preparation: Geunyoung Jang (azberjibiou)

Solution

1. Definition
Let's consider the operations in reverse order. An operation can be viewed as coloring the cells that have not yet been colored in each row/column. From now on, we will consider the operations in reverse order.

For each operation, let the set of cells colored by the operation be referred to as its area.

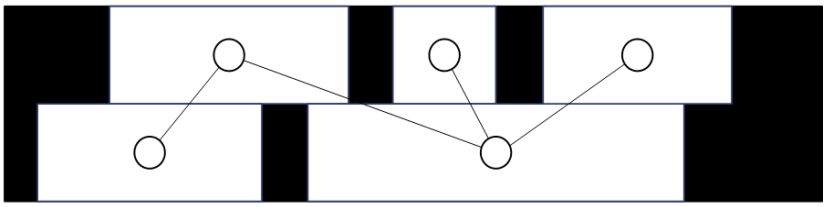
During the execution of an operation, set of uncolored cells can be represented as a rectangle including $(1, 1)$. Therefore, the area of each operation is a shape of a rectangle with one side length of 1 and its top/left side touching the corner of the square. Let $S_{i,1}$ denote the area for the operation on the i-th row, and $S_{i,2}$ denote the area for the operation on the i-th column.

2. Redrawing the Grid
Let's newly define areas $S'_{i,j}$ within a grid of size $2 \times (2N-1)$ while preserving the adjacency of $S_{i,j}$. This means that the adjacency of S_{i_1,j_1} and S_{i_2,j_2} will be the same as that of S'_{i_1,j_1} and S'_{i_2,j_2}.

As described earlier, during the execution of operations, set of uncolored cells appear as a rectangle including (1, 1). The perimeter of this rectangle has 2 adjacent areas. If we have colored up to the x_i-th row and y_i-th column up to the i-th operation, then $S_{x_i,1}$ and $S_{y_i,2}$ are adjacent to the perimeter. The area corresponding to the executing operation is adjacent to these two areas. Thus, it can be observed that for each area, there are at most 2 areas from earlier operations that are adjacent to it.

For $1 \le i \le 2N-2$, let's assign the areas in a $2 \times (2N-1)$ grid so that $(i, 1)$ belongs to $S'_{x_i,1}$ and $(i, 2)$ belongs to $S'_{y_i,1}$. By partitioning the areas this way, we can preserve their adjacency. Finally, since $(1, 1)$ in the original grid is adjacent to $S_{2N-2,1}$ and $S_{2N-2,2}$, we can map it to $(2N-1, 1)$ and $(2N-1, 2)$.

3. Solution
Now, let's observe within the $(2N-1) \times 2$ grid. For each row, group the areas of the same color into one and assign a vertex to them. The color of the vertex is defined by the color of the grouped areas. When edges are drawn between vertices of the same color that share an edge(edges in rectangles), the graph contains no cycles. This is because, for each component, the maximum degree of the vertex corresponding to the area with the leftmost right edge is 1. By sequentially removing these leaf vertices, we can conclude that the components take the form of trees.



The vertex with the leftmost right edge has a maximum degree of 1

Therefore, if we know the number of vertices and edges, we can determine the number of components.
Edges are formed between the same colored $(i, 1)$ and $(i, 2)$. Let $c_{i,j}$ be the color of cell (i, j). Considering the process of creating $S'_{i,j}$, either $S'_{i,1} = S'_{i+1,1}$ or $S'_{i,2} = S'_{i+1,2}$ holds. This implies that it is not possible for both $c_{i,1} \neq c_{i+1,1}$ and $c_{i,2} \neq c_{i+1,2}$ to occur.

Let s_i be defined as 1 if $c_{i,1} = c_{i,2}$, and 0 otherwise. The number of i such that $s_i \neq s_{i+1}$, which is near the twice the number of edges, can be computed as (the count of i where $c_{i,1} \neq c_{i+1,1}$) plus (the count of i where $c_{i,2} \neq c_{i+1,2}$).

(The number of i such that $c_{i,j} \neq c_{i+1,j}$ +1) equals the number of vertices in the j-th row.

The necessary information to compute the number of vertices and edges can be updated in $O(1)$ for each query, leading to a solution of time complexity $O(N + Q)$.

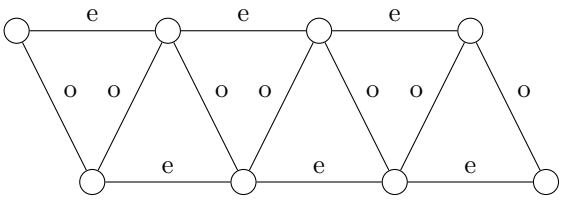
Problem D. Graceful Triangles

Problem Idea: Woosuk Kwak (bubbler)
Preparation: Woosuk Kwak (bubbler)

Solution

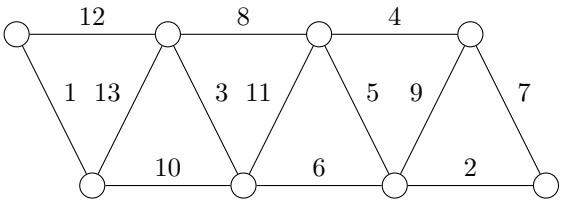
1. Edge parity assignment
First, look at a single triangle. If its three vertices are given the values x, y, and z, its three edges have the values $|x-y|$, $|y-z|$, and $|x-z|$. If the three vertices have the same parity, all three edges' values will be even; if one is different from the other two, one edge's value will be even and the other two will be odd.

There are $n+1$ odd and n even numbers between 1 and $2n+1$ inclusive. In order to satisfy the parity condition of each triangle, one straightforward assignment is to assign even values to the horizontal edges and odd values to the diagonal ones.



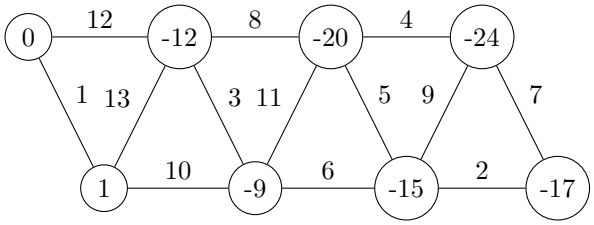
2. Edge value assignment
Again, look at a single triangle. It can be seen that two of the three edge values must sum to the third. To put it another way, if two of the three edge values are given, the third must be either their sum or difference.

Therefore, the task now is to find a suitable ordering of odd numbers so that the even edge values are unique. There exists a rather simple pattern that fits the condition: $1, 2n+1, 3, 2n-1, 5, 2n-3, \cdots$. Then the differences are $2n, 2n-2, 2n-4, \cdots$.



3. Node value assignment
Without loss of generality, we can fix the top leftmost node to an arbitrary value, say 0. We can also fix the bottom leftmost node to 1, which sets the leftmost diagonal edge to the desired value of 1.

Now we will solve each triangle from left to right, filling one node at a time. It can be shown that, when two of the three node values on a triangle are known, the third node's value is unique. Such value can be found in several ways in constant time; one could do case work based on where the largest edge is, or take the intersection of two sets of possible values determined by two incident edges.



Now, this is not a valid answer since the node values are not positive. This can be fixed easily by offsetting all values by a constant. The difference of the largest and smallest node values assigned this way does not exceed n^2, which is good enough to keep all values under the upper limit of 10^{18}.

Problem E. Hexagonal Tiling

Problem Idea: Geunyoung Jang (azberjibiou)
Preparation: Geunyoung Jang (azberjibiou)

Solution

For each rhombus, draw an arrow from the bottom side of a rhombus to the top side of a rhombus. By following the arrows, we can make a path from the bottom of a hexagon to the top of a hexagon. In fact, N non-intersecting paths correspond to a rhombus tiling.

Let's construct a flow graph. Let's represent an edge with capacity c and cost d as (c, d). For each horizontal line segment of length 1, there are two vertices; one for flow coming in, and one for flow coming out. Two vertices are connected with an $(1, 0)$ edge. From a source, there is a $(1, 0)$ edge connecting the bottom line segment. For two horizontal line segments that can form a rhombus, there is a $(1, c_1 - c_2)$ edge, from the bottom segment to the top segment. Here c_1 is the cost of the rhombus formed, and c_2 is the cost of a rhombus that contains the top segment as a diagonal. For each top-line segment, there is a $(1, 0)$ edge connecting the sink.

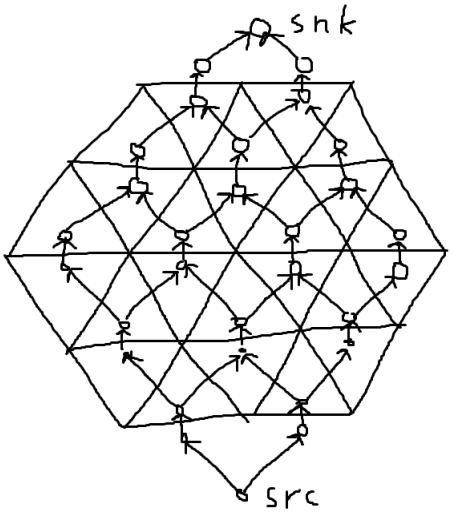


Figure 1: Flow graph

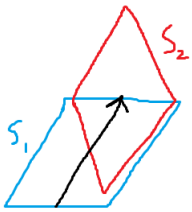


Figure 2: Blue rhombus cost − red rhombus cost is the edge cost

This means that initially, every rhombus has as horizontal segment as its diagonal, and once we add a rhombus that is formed by two horizontal segments, the rhombus containing the top horizontal segment as a diagonal disappears.

The answer would be (Sum of every rhombus having a horizontal segment as diagonal) + (min cost of max flow).
The flow graph has $O(n^2)$ vertices and $O(n^2)$ edges. Also, max flow is n. Using Johnson's potential method, we can solve the problem in $O(n^3 logn)$.

Trivial bipartite matching solutions would be cut off.

Problem F. Jenga Game

Problem Idea: Eunjae Jin (ai4youej)
Preparation: Eunjae Jin (ai4youej)

Solution

For each layer:

- For layers of type 010 and 101, no blocks can be removed.
- For layers of type 110 and 011, exactly one block can be removed. Let us call this type of layer **Type A**.
- For layers of type 111, one or two blocks can be removed. Let us call this type of layer **Type B**.

Let c_1 denote the number of Type A layers, and c_2 denote the number of Type B layers.
If we define $dp[i][j]$ as whether **Yesyes** can win the game, we derive the following equation:

$$dp[i][j] = \sim dp[i-1][j] \vee \sim dp[i+1][j-1] \vee \sim dp[i][j-1],$$

with the base case $dp[0][0] = \text{false}$.
This equation can be understood by considering the following three cases:

- Removing one Type A layer.
- Changing one Type B layer into a Type A layer (111 → 011 or 111 → 110).
- Removing one Type B layer (111 → 101).

Since N is large, it is not feasible to solve the problem by directly filling the DP table. However, it is easy to determine the winning condition: **Nono** wins if and only if both c_1 and c_2 are even.

Problem G. Make RUN Great Again

Problem Idea: Taein Oh (octane)
Preparation: Taein Oh (octane)

Solution

Let the score of RUN s be fixed. For each a_i, the increase in skepticism factor when reducing a_i to s or below is given by $c_i := \max(0, (a_i - s)b_i)$. To determine if s is a valid initial score for RUN, consider the smallest $N - X + 1$ values among the c_i. If their sum is less than K, then s is a valid initial score.
Additionally, we can observe that for the correct answer s_0, s is valid if $s \geq s_0$ and invalid if $s < s_0$. Using this property, we can solve the problem using parametric search.
The time complexity of the solution is as follows:

- For each s, determining its validity requires sorting the c_i, which takes $O(N \log N)$.
- Parametric search requires $O(\log K)$ iterations. Thus, the overall complexity is $O(N \log N \log K)$.

Alternatively, we can improve the efficiency of validity checking for each s:

1. By using the Median of Medians method, find the $(N - X + 1)$-th smallest element t among the c_i in $O(N)$ time.
2. Traverse the c_i values, summing up the smallest $N - X + 1$ values that are small than or equal to t, and check if their sum is less than K.

With this approach, the overall complexity becomes $O(N \log K)$.

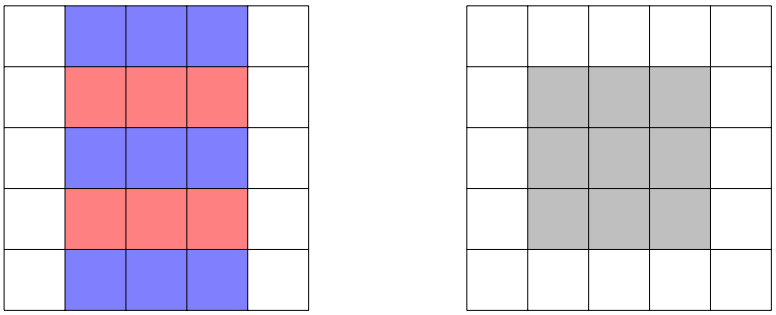
Problem H. Mosaic

Problem Idea: Woosuk Kwak (bubbler)
Preparation: Woosuk Kwak (bubbler)

Solution

1. Easy sizes

Consider a band of three columns as in the left figure below. If we make a new variable per row equal to the number of black cells in the 3-cell region, each given information as in the right figure becomes the sum of three adjacent variables.



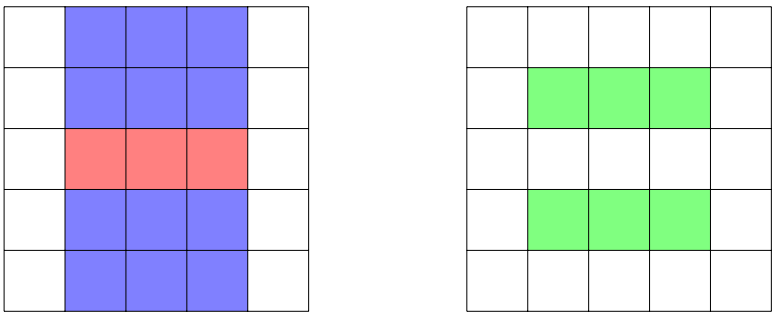
If $R \not\equiv 2 (\mathrm{mod}\, 3)$, such a system of equations is uniquely solvable in linear time. Once all such equations are solved, a similar system of equations appears on each row. If $C \not\equiv 2 (\mathrm{mod}\, 3)$ as well, each row can be solved similarly to reveal whether each cell should be black (1) or white (0). Otherwise, it is possible to fix the first cell and derive all other cells' values from left to right, and see if all values are either 0 or 1. This also takes linear time.

If $R \equiv 2 (\mathrm{mod}\, 3)$ but $C \not\equiv 2 (\mathrm{mod}\, 3)$, the same procedure can be applied to the transpose of the input. This is the official solution to BOI 2010 Problem 6 "Mines".

2. Hard sizes

The main problem now is to solve the case where $R \equiv C \equiv 2 (\mathrm{mod}\, 3)$.

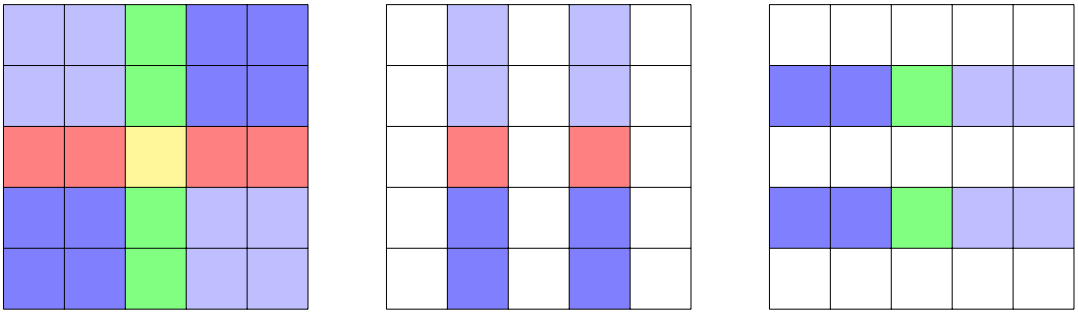
Consider the same set of equations as in the first part of the solution. Simplifying the set of equations, it is possible to reduce the equations to the set of the following:



- the sum of cells on the rows $3k + 1$ and $3k + 2$ (blue groups in the figure above);
- the sum of cells on the rows $3k + 2$ and $3k + 4$ (green group); and

- the sum of cells on row $3k$ (red group).

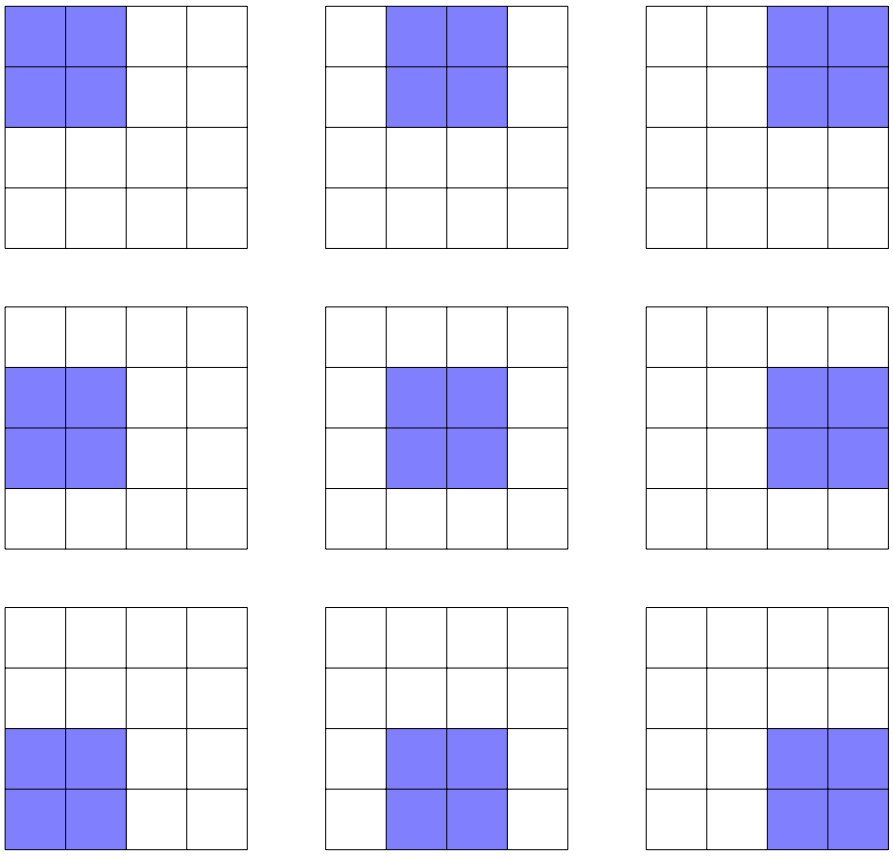
The resulting groups also horizontally form the respective sets of equations, and can be simplified in the same manner, resulting in the following groups below.



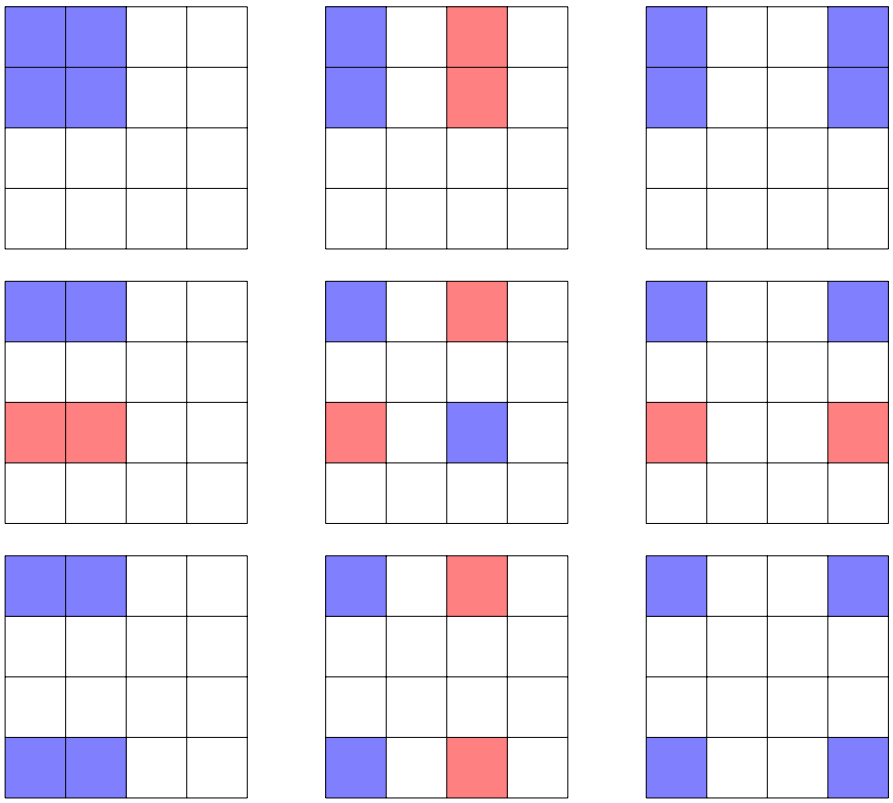
At this point, all groups except the blue ones can be solved:

- Yellow cells are already solved.
- Red groups on a row form a set of $k-1$ equations of k unknowns where the sums of adjacent pairs are given. This can be solved by assuming zero or one for the first variable and solving the rest.
- Green groups on a column can be solved in the same way.

Now, remove all solved cells and stitch the subgrids back together to get the following set of equations:



And perform "reverse Gaussian elimination" so that each equation contains a term corresponding to the top left cell. In the figure below, blue cells indicate the coefficient of 1 and red ones indicate -1. This can be done by performing elimination row-wise and then column-wise.



Now, every equation contains the top leftmost cell, one cell on the leftmost column (excluding the top one), one cell on the topmost row (excluding the leftmost one), and one cell in the remaining region. Moreover, the fourth category of cells appear exactly once. Since each variable must be either 0 or 1, we can construct a 2-SAT instance equivalent to this set of equations as follows:

- Assume the value of the top leftmost cell to be either 0 or 1. Let's call this value w.
- The values of cells on the first column and the first row are 2-SAT variables. Let's call them $x_2, x_3, \cdots$ and $y_2, y_3, \cdots$ respectively.
- The remaining variables act as the constraint that it must be either 0 or 1. Let's call them z_{ij}.

The equations are in the form $w + ax_i + by_j + cz_{ij} = d$, or equivalently $ax_i + by_j = d - w - cz_{ij}$. d is given and w is assumed, so we get the two possible values of $ax_i + by_j$. Since x_i and y_j must be also either 0 or 1, it is possible to deduce "forbidden assignments", e.g. $(x_i, y_j) \neq (0, 1)$. This directly translates to a 2-SAT clause $x_i \vee \neg y_j$. Now we can collect all such clauses and solve the resulting 2-SAT instance using any linear time 2-SAT algorithm.

Overall, the entire solution takes $\mathcal{O}(NM)$ time.

Problem I. Mukjjippa

Problem Idea: Kiyun Park (parkky)
Preparation: Kiyun Park (parkky)

Solution

We may assume that for every i $(1 \le i \le n)$, X_i and Y_i are defined.
For each i $(1 \le i \le n)$, exactly one of the following events occurs:

- $E_{i,1}$: $(X_i, Y_i) \in \{(\mathrm{R},\mathrm{R}), (\mathrm{S},\mathrm{S}), (\mathrm{P},\mathrm{P})\}$
- $E_{i,2}$: $(X_i, Y_i) \in \{(\mathrm{R},\mathrm{S}), (\mathrm{S},\mathrm{P}), (\mathrm{P},\mathrm{R})\}$
- $E_{i,3}$: $(X_i, Y_i) \in \{(\mathrm{R},\mathrm{P}), (\mathrm{S},\mathrm{R}), (\mathrm{P},\mathrm{S})\}$

For each i $(1 \le i \le n)$, by the independence of the choices:

- $\Pr(E_{i,1}) = \dfrac{r_i r'_i + s_i s'_i + p_i p'_i}{(r_i + s_i + p_i)(r'_i + s'_i + p'_i)}$
- $\Pr(E_{i,2}) = \dfrac{r_i s'_i + s_i p'_i + p_i r'_i}{(r_i + s_i + p_i)(r'_i + s'_i + p'_i)}$
- $\Pr(E_{i,3}) = \dfrac{r_i p'_i + s_i r'_i + p_i s'_i}{(r_i + s_i + p_i)(r'_i + s'_i + p'_i)}$

For each i $(1 \le i \le n+1)$, exactly one of the following events occurs:

- $F_{i,1}$: At the beginning of the i-th turn, the game is in progress and there is no attacker.
- $F_{i,2}$: At the beginning of the i-th turn, the game is in progress and A is the attacker.
- $F_{i,3}$: At the beginning of the i-th turn, the game is in progress and B is the attacker.
- $F_{i,4}$: At the beginning of the i-th turn, the game is not in progress and A is the winner.
- $F_{i,5}$: At the beginning of the i-th turn, the game is not in progress and B is the winner.

$F_{1,1}$ occurs.
For each i $(1 \le i \le n)$:

- If $F_{i,1}$ and $E_{i,1}$ occur, then $F_{i+1,1}$ occurs.
- If $F_{i,1}$ and $E_{i,2}$ occur, then $F_{i+1,2}$ occurs.
- If $F_{i,1}$ and $E_{i,3}$ occur, then $F_{i+1,3}$ occurs.
- If $F_{i,2}$ and $E_{i,1}$ occur, then $F_{i+1,4}$ occurs.
- If $F_{i,2}$ and $E_{i,2}$ occur, then $F_{i+1,2}$ occurs.
- If $F_{i,2}$ and $E_{i,3}$ occur, then $F_{i+1,3}$ occurs.
- If $F_{i,3}$ and $E_{i,1}$ occur, then $F_{i+1,5}$ occurs.
- If $F_{i,3}$ and $E_{i,2}$ occur, then $F_{i+1,2}$ occurs.

- If $F_{i,3}$ and $E_{i,3}$ occur, then $F_{i+1,3}$ occurs.
- If $F_{i,4}$ occurs, then $F_{i+1,4}$ occurs.
- If $F_{i,5}$ occurs, then $F_{i+1,5}$ occurs.

For each i, j, j' ($1 \le i \le n$, $1 \le j \le 3$, $1 \le j' \le 5$):

- $E_{i,j}$ depends only on the i-th choices.
- $F_{i,j'}$ depends only on the $1, 2, \ldots, (i-1)$-th choices.
- Since all choices are independent, $E_{i,j}$ and $F_{i,j'}$ are independent.

For each i ($1 \le i \le n$):

- $\Pr(F_{i+1,1}) = \Pr(F_{i,1}) \cdot \Pr(E_{i,1})$
- $\Pr(F_{i+1,2}) = (\Pr(F_{i,1}) + \Pr(F_{i,2}) + \Pr(F_{i,3})) \cdot \Pr(E_{i,2})$
- $\Pr(F_{i+1,3}) = (\Pr(F_{i,1}) + \Pr(F_{i,2}) + \Pr(F_{i,3})) \cdot \Pr(E_{i,3})$
- $\Pr(F_{i+1,4}) = \Pr(F_{i,2}) \cdot \Pr(E_{i,1}) + \Pr(F_{i,4})$
- $\Pr(F_{i+1,5}) = \Pr(F_{i,3}) \cdot \Pr(E_{i,1}) + \Pr(F_{i,5})$

$\Pr(F_{1,1}) = 1$, $\Pr(F_{1,2}) = \Pr(F_{1,3}) = \Pr(F_{1,4}) = \Pr(F_{1,5}) = 0$.
The answer is $\Pr(F_{n+1,4})$.
The answer can be computed in $\mathcal{O}(n)$ time.

Problem J. Running in the Plane

Problem Idea: Kiyun Park (parkky)
Preparation: Kiyun Park (parkky)

Solution

Define $O = \{(0, 0)\}$.
Define $L(\theta) = \{(t\cos(\theta), t\sin(\theta)) : t \in \mathbb{R}_{>0}\}$ for each $\theta \in \mathbb{R}$.
Note that $\{O\} \cup \{L_\theta : \theta \in \mathbb{R}\}$ is a partition of $\mathbb{R}^2$.
Define $Q((x, y)) = (\frac{x}{\gcd(x,y)}, \frac{y}{\gcd(x,y)})$ for each $(x, y) \in \mathbb{Z}^2 \setminus \{(0, 0)\}$.

Claim 0. *S has a good set of size at most 0 if and only if $S \subseteq O$.*

Proof. Trivial. □

Claim 1. *S has a good set of size at most 1 if and only if $S \subseteq O \cup L_\theta$ for some $\theta \in \mathbb{R}$.*

Proof. [$\Longrightarrow$] Let T be a good set of S, where $|T| \le 1$, $T \subseteq \{v\}$, and $v \in O \cup L_\alpha$. Then $S \subseteq O \cup L_\alpha$.
[$\Longleftarrow$] Suppose that $S \subseteq O \cup L_\theta$.
If $S \cap L_\theta = \emptyset$, then it is covered by Claim 0.
Take any $p \in S \cap L_\theta$. Then $\{Q(p)\}$ is a good set of S. □

Claim 2. *S has a good set of size at most 2 if and only if either $S \subseteq O \cup L_\theta \cup L_{\theta+\pi}$ or $S \subseteq O \cup \bigcup_{\theta \le \phi < \theta+\pi} L_\phi$ for some $\theta \in \mathbb{R}$.*

Proof. [$\Longrightarrow$] Let T be a good set of S, where $|T| \le 2$, $T \subseteq \{v, w\}$, $v \in O \cup L_\alpha$, and $w \in O \cup L_\beta$.
If $0 \le (\beta - \alpha) \bmod 2\pi < \pi$, then $S \subseteq O \cup \bigcup_{\alpha \le \phi < \alpha+\pi} L_\phi$.
If $(\beta - \alpha) \bmod 2\pi = \pi$, then $S \subseteq O \cup L_\alpha \cup L_{\alpha+\pi}$.
If $\pi < (\beta - \alpha) \bmod 2\pi < 2\pi$, then $S \subseteq O \cup \bigcup_{\beta \le \phi < \beta+\pi} L_\phi$.
[$\Longleftarrow$] First, suppose that $S \subseteq O \cup L_\theta \cup L_{\theta+\pi}$.
If $S \cap L_\theta = \emptyset$ or $S \cap L_{\theta+\pi} = \emptyset$, then it is covered by Claim 1.
Take any $p \in S \cap L_\theta$ and $q \in S \cap L_{\theta+\pi}$. Then $\{Q(p), Q(q)\}$ is a good set of S.
Second, suppose that $S \subseteq O \cup \bigcup_{\theta \le \phi < \theta+\pi} L_\phi$.
Take the minimum $\alpha \in \mathbb{R}$ such that $\theta \le \alpha < \theta + \pi$ and $S \cap L_\alpha \neq \emptyset$. If such α does not exist, then it is covered by Claim 0.
Take any $p \in S \cap L_\alpha$, and let $u = Q(p) = (a, b)$.
Find $v = (c, d) \in \mathbb{Z}^2$ such that $ad - bc = 1$ and $\max(|c|, |d|) \le 10^8$. Note that this can be found using extended Euclidean algorithm.
Each point in S can be uniquely represented as $xu + yv$ where $(y, x) \in \mathbb{Z}^2$ and $(y, x) \ge (0, 0)$. Here the inequality represents the lexicographic order.
Take a large integer $k = 10^9$. Then using vectors in $\{u, v - ku\}$, we can visit all points in S in the lexicographic order of (y, x).
Therefore, $\{u, v - ku\}$ is a good set of S. □

Claim 3. *S always has a good set of size at most 3.*

Proof. $\{(0, 1), (1, 0), (-1, -1)\}$ is a good set of S. □

Using these claims, we can determine the minimum size of a good set by sorting the given points by polar angles, and then construct a minimum-size good set by following the construction for the corresponding case. Therefore, we can find a minimum-size good set in $\mathcal{O}(n \log n)$ time.

Problem K. Same Segment

Problem Idea: Geunyoung Jang (azberjibiou)
Preparation: Geunyoung Jang (azberjibiou)

Solution

Assume no nested segments exist. Let r_1 be a right end of segment starting from 1. Set $a_1 = K$, $a_i = 0$ for $i \in [2, r_1]$. For each segment $[l, r]$ where $l > 1$, set $l = max(l, r_1 + 1)$. Since no two segments are nested, $r > r_1$ holds. By recursively solving the problem, we can find a sequence a.

If there exists two nested segments $[l_1, r_1]$ and $[l_2, r_2]$, $a_i = 0$ for $i \in [l_1, l_2) \cup (r_1, r_2]$. Remove such indices and repeat the process until no two segments are nested. If there is a segment that disappears (a segment is contained in removed indices), the answer would be no.

We can observe that K does not matter to the existence of a solution. Therefore we can assume $K = 1$.

Define dp_i as whether there exists $a_1, a_2, \ldots, a_i$ where

- $a_i = 1$
- For every segment $[l, r]$ where $l \leq i$, $\sum_{j=l}^{r} a_j = 1$

For a transition from dp_{i_1} to dp_{i_2} to occur, the following should hold.

- No segment should contain both i_1 and i_2.
- No segment should be contained in (i_1, i_2).

For a fixed i_2, i_1 that satisfies the condition forms an interval. This interval can be easily solved in constant time or logarithmic time. Let such interval be $[i_l, i_r]$. By using segment tree, we can find out whethere there exists $i \in [i_l, i_r]$ such that dp_i is true. The problem can be solved in $O(NlogN)$.

Problem L. Simple Tree Decomposition Problem

Problem Idea: Dohoon Kim (dohoon)
Preparation: Dohoon Kim (dohoon)

Solution

To remove ambiguity, let's define $u^{\downarrow}$ as the set of descendants of u, including u itself, when vertex 1 is taken as the root of the tree and the tree is represented in its downward layout.

This problem can be solved using tree DP.

The term of recurrence relation can be intuitively defined as:
$\mathrm{dp}(u, x) :=$ the number of ways to form a subtree of size x that includes u within $u^{\downarrow}$

If we implement this method naively, it would run with a time complexity of $O(n^3)$.

Here, we can optimize the solution by using a dynamic array to ensure that x in each $\mathrm{dp}(u, x)$ does not exceed $|u^{\downarrow}|$. By applying this simple adjustment, the time complexity can be reduced to $O(n^2)$. For more information on this topic, you can search for "Square-order tree DP" or the "검은돌 트릭".

However, this approach is still insufficient to solve the problem efficiently. Therefore, one more optimization can be applied: adding a condition to ensure that x does not exceed B. Since the DP can function without using indices where x exceeds B, this constraint helps reduce unnecessary computations.

By applying these two optimizations, the time complexity is surprisingly reduced to $O(NB)$.
The reason is that each recurrence can be understood as merging two trees, and when we unfold the tree using an Euler tour, we can observe that the total number of segments involved in the merging process is bounded by segments of length $2K$ or less.

The recurrence relation is somewhat simpler compared to the optimization ideas, so I encourage you to derive it yourself. As a reference, note that an $O(NB \log B)$ solution using FFT will not run within the time limit.

Problem M. White-Black-Tree

Problem Idea: Yejun Lim (edward3378)
Preparation: Yejun Lim (edward3378)

Solution

Let $T_w(V, E)$ be white tree, and $T_b(V, E)$ be black tree. In each edge swap operation, focus on changes of $|E(T_w)|$ and $|E(T_b)|$. Let Δw be change of $|E(T_w)|$ and Δb be change of $|E(T_b)|$.

Since at most 1 vertex of T_w(or T_b) move to adjacent vertex in each swap, we can know that $|\Delta w| \leq 1$ and $|\Delta b| \leq 1$. Therefore, $-2 \leq \Delta w + \Delta b \leq 2$. To take advantage of the cost in one swap, $\Delta w + \Delta b + 1 \leq -1$ which means $\Delta w = -1$ and $\Delta b = -1$. In such operation that $\Delta w = -1$ and $\Delta b = -1$, swapped edge should not be included both T_w and T_b. After this swap operation, it is trivial that total cost cannot be reduced more because edge cost is already minimized. Therefore, suppose k is the edge cost of initial tree, minimum possible total cost of a final tree should be k or $k - 1$.

Let's see the condition of the answer is $k-1$. If $k-1$ is available, we should only perform such $\Delta w+\Delta b = -1$ operations and perform $\Delta w+\Delta b = -2$ operation at last. Also, the final tree will be a tree rooted at a specific vertex in which all vertices of a certain subtree have the same color, and the colors of the remaining vertices outside the subtree are also identical. Without loss of generality, assume we want to make T_b of final tree be subtree which vertex s is root. Also, only consider number of black vertices is equal to number of vertices in subtree s. Let t be number of white vertices in initial subtree s. Divide into 3 cases for t :

(1) $t = 0$
Edge cost can't be reduced. The answer is k.

(2) $t = 1$
Let that white vertex be v_w and black vertex that is initially at T_w be v_b.

(a) $\deg(v_w) = 1$ or $deg(v_b) = 1$
When v_w(or v_b) swaps with its parent vertex, $\Delta b = 1$(or $\Delta w = 1$). The answer is k.

(b) $\deg(v_w) \geq 2$ and $deg(v_b) \geq 2$
The answer is $k - 1$.

(3) $t \geq 2$
During the swap operation, there is a moment that vertex s is white and the parent vertex of s is black. Between two vertices should be swapped. In this swap operation, there exists a white vertex at T_b and exists a black vertex at T_w. Therefore $\Delta w \geq 0$ and $\Delta b \geq 0$, the answer is k.

In conclusion, for each s, we can check whether $k-1$ is available for $O(1)$ time complexity by above constraints. Since s can be vertex 1 to N, we can solve this problem with total time complexity of $O(N)$.