Problem A. XOR Tree Path

The color of vertex v is determined by the number of leaves selected from the subtree of v. Also, selecting two leaves returns the color of v to its original state, so it is important whether an even or odd number of leaves is selected from the subtree of v.

Therefore, this problem can be solved by the following DP on trees:

$dp[v][i]$: the maximum number of black vertices in the subtree of v when the number of selected leaves in the subtree of v modulo 2 is i.

The time complexity is $O(N)$.

Problem B. Magical Wallet

This problem can be solved by the following DP:

$dp[i][j]$: the maximum number of products that can be bought when the magic wallet contains j yen after visiting the i-th shop.

To simplify the transitions, we assume that the amount of money in the wallet (j) is always the maximum possible. Let $p(j)$ be the set of integers that can be obtained by rearranging the digits of j, and $m(j) := \max p(j)$.

The DP transitions are as follows:

- $dp[0][m(X)] \leftarrow 0$
- $dp[i+1][j] \leftarrow dp[i][j]$
- If j equals $m(j)$:
  - For $k \in p(j)$:
    - If $k \geq A_i$:
      - $dp[i+1][m(k-A_i)] \leftarrow dp[i][j] + 1$

The time complexity is $O(NM)$.

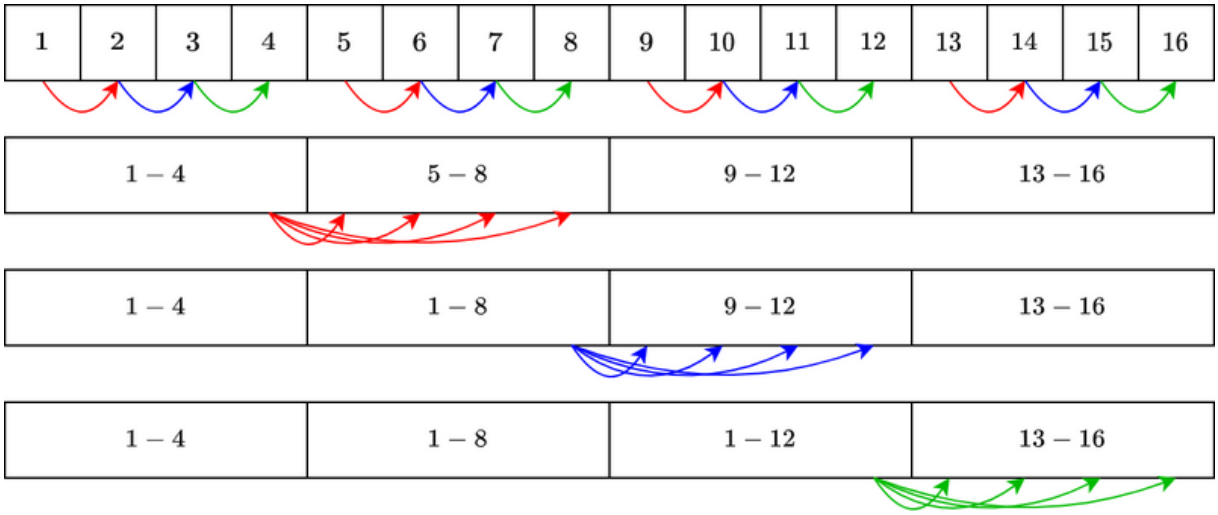
Problem C. Parallel Processing (Easy)

The correspondence between N and L is as follows:

N	2	3–4	5–8	9–11	12–13	14–16
L	1	2	3	4	5	6

Therefore, this problem can be solved by writing a carefully designed brute-force program to search for a solution or manually constructing a solution for $N = 2, 4, 8, 11, 13, 16$.

Here is an example of a solution for $N = 16$:



Problem D. Parallel Processing (Hard)

In conclusion, the minimum number of instructions is: $L = \lceil \max\left(\log_2 N, \frac{2}{5}(N-1)\right) \rceil$.

Lower bound 1: $\log_2 N$

It is clear from the fact that the length of an integer sequence that can be constructed with L instructions is at most 2^L.

Lower bound 2: $\frac{2}{5}(N-1)$

Assuming $N \geq 9$:

- It is pointless to create an integer sequence that is not in the form of $(l, l+1, \ldots, r-1, r)$.
- We will relax the conditions and assume that all previously created sequences can be used without overwriting variables.
- It is pointless to create the same interval multiple times.
- Because each interval is created only once, we know that creating interval $[l, r]$ implies the existence of a unique d $(l \leq d < r)$ such that $[l, d] + [d+1, r]$ creates $[l, r]$.
- Considering a graph of which interval is created from which interval, we see that the structure is like binary trees rooted at each of the intervals $[1, 1], [1, 2], \ldots, [1, N]$.
- Taking only the necessary parts from this graph to construct $[1, N]$, we obtain a binary tree with $2N-1$ vertices and $2N-2$ edges, which contains $(N-1)$ $\oplus$ operations. We denote this binary tree by T.
- If we look at T from the root down, we see that it repeatedly splits a certain interval in two. Therefore, if we take any two vertices from T, the two intervals either do not intersect or one is contained in the other.
- We fix the number of $\oplus$ operations to S, and we have $L \geq \frac{S}{4}$.
- Let A be the number of vertices in T that have the form $[1, i]$. Since these vertices have a dependency on the previous calculation result, we have $L \geq A - 1$.
- If we use the calculation result from more than two steps ago, there exist two vertices in T where two intervals intersect, and neither of them is contained in the other.

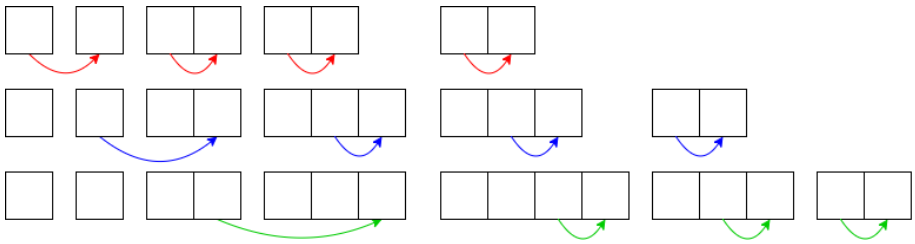
- Since there are $N-1$ operations to calculate T and $N-A$ or more operations to calculate vertices that are not included in T, we have $(N-1)+(N-A) \leq S$.
- From this, we obtain $A \geq 2N-1-S$, and hence $L \geq 2N-2-S$.
- Therefore, we have $L \geq \max\left(\frac{S}{4}, 2N-2-S\right) \geq \frac{2}{5}(N-1)$.

Achieving the lower bound: $L = \left\lceil \frac{2}{5}(N-1) \right\rceil$

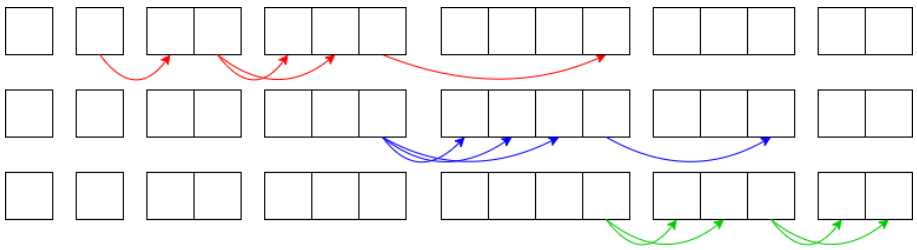
By following the instructions below, it is possible to achieve $L = \left\lceil \frac{2}{5}(N-1) \right\rceil$.

Idea: First, divide the sequence $A_1, A_2, \ldots, A_N$ into $L+1$ blocks. Then, calculate the cumulative sum of each block while also computing the cumulative sum of the entire block. Note that since there are $L+1$ blocks, the computation of the cumulative sum of the entire block needs to be advanced by one operation each time.

Based on this, divide the blocks. While being mindful to advance the cumulative sum of the entire block at each step, try to make each block as long as possible.



Once the length of the blocks is sufficient, calculate the remaining part instead of extending each block.



By doing the above, we can achieve the lower bound of $\left\lceil \frac{2}{5}(N-1) \right\rceil$ instructions.

Problem E. Five Med Sum

To solve the problem, we need to count the number of times each value in the sequence appears as a median.

First, we sort the $5N$ integers with tags indicating which of the five sequences they belong to. Let $F = ((F_1, t_1), (F_2, t_2), \ldots, (F_{5N}, t_{5N}))$ denote the sorted sequence, where t_i is the tag.

Then, for each $k = 1, 2, \ldots, 5N$, we need to count the number of tuples (i, j, l, m) such that $1 \leq i < j < k < l < m \leq 5N$ and $\{t_i, t_j, t_k, t_l, t_m\} = \{A, B, C, D, E\}$. We can solve this problem in $O(1)$ time by brute-forcing all pairs $\{t_i, t_j\}$ and using cumulative sums.

The time complexity is $O(N \log N)$.

Problem F. Forestry

We will rephrase the problem as finding the expected score when each edge is independently cut with a probability of $1/2$.

Choose an arbitrary root vertex, and let T be the rooted tree. For a vertex v, let T' be the subtree of v and let size(v) be the number of vertices in T'. Define the following DP:

dp[v][x]: the probability that the minimum value of a connected component of T' containing v is x, when edges of T' are randomly cut

Calculate this DP and obtain the answer by summing the "contribution to the answer by all connected subgraph of T where v is the shallowest vertex" for each v. The DP table has 10^9 possible values for the second argument x, but only size(v) values are non-zero, so it can be compressed by keeping only non-zero values.

Consider the time complexity of calculating the DP in the compressed state. If v has two children c_1, c_2, we can calculate dp[v][$*$] from dp[c_1][$*$] and dp[c_2][$*$] using cumulative sum in $O(\text{size}(c_1) + \text{size}(c_2))$ time. If v has three or more children, we can repeatedly merge pairs of children to calculate dp[v][$*$] in $O(\text{size}(v) \log(N))$ time.

Method using Merge Technique

The calculation of dp[v][$*$] from dp[c_1][$*$] and dp[c_2][$*$] can be viewed as performing a single-point update or range multiplication on dp[c_1][$*$] using information from dp[c_2][$*$]. Therefore, if we manage them with a lazy propagation segment tree that creates only the necessary elements, we can merge them in $O(\min(\text{size}(c_1), \text{size}(c_2)) \log(A))$ time. The overall time complexity is $O(N \log(N) \log(A))$, but this can be reduced to $O(N \log(A))$ if we handle the segment tree merge efficiently.

Method for accelerating Segment Tree merging: https://codeforces.com/blog/entry/49446

Method using Top Tree

We use the technique explained in problems such as https://atcoder.jp/contests/abc269/tasks/abc269_h. Let C be a connected subgraph of T that is only connected to the outside through u and v. The data that needs to be maintained is as follows:

- For each x, the probability that u and v are connected and the minimum value of the connected component which contains u and v is x when the edges of C are randomly cut.
- For each x, the probability that u and v are not connected and the minimum value of the connected component which contains u is x when the edges of C are randomly cut.
- For each x, the probability that u and v are not connected and the minimum value of the connected component which contains v is x when the edges of C are randomly cut.
- The contribution to the answer from the connected components which do not contain either u or v when the edges of C are randomly cut.

Two data can be merged in linear time with respect to the sum of the number of vertices. Thus, we can solve the problem in $O(N \log(N))$ time overall.

Problem G. Range NEQ

We can solve this problem using the principle of inclusion and exclusion.

Consider a sequence of integers $(k_1, \ldots, k_N)$, and let us consider the cases where k_j indices i satisfy $\left\lfloor \frac{i}{M} \right\rfloor = \left\lfloor \frac{P_i}{M} \right\rfloor = j$, for each $j = 0, 1, \ldots, N-1$.

In this case, the number of ways to choose these k_j indices is $\binom{M}{k_j}$, and the number of ways to assign values to these k_j indices is $\frac{M!}{(M-k_j)!}$.

The number of ways to assign values to the remaining $(NM - \sum_{j=0}^{N-1} k_j)$ indices is $(NM - \sum_{j=0}^{N-1} k_j)!$.

Therefore, the number of permutations corresponding to a given $(k_0, k_1, \ldots, k_{N-1})$ is:

$$(NM - \sum_{j=0}^{N-1} k_j)! \prod_{j=0}^{N-1} \frac{M!}{((M-k_j)!)^2 k_j!}$$

By the principle of inclusion and exclusion, the answer is:

$$\sum_{(k_0,k_1,\ldots,k_{N-1})} (-1)^{\sum_{j=0}^{N-1} k_j} (NM - \sum_{j=0}^{N-1} k_j)! \prod_{j=0}^{N-1} \frac{M!}{((M-k_j)!)^2 k_j!}$$

We can compute it by considering the generating function. Let

$$f(x) := \sum_{k=0}^{M} \frac{M!}{((M-k)!)^2 k!} (-x)^k,\ g(x) := (f(x))^N$$

Then, the answer is:

$$\sum_{s=0}^{NM} (NM - s)! ([x^s] g(x))$$

The time complexity is $O(NM \log(NM))$.

Problem H. Expanded Hull

When K varies, the answer to this problem is a 3rd-degree polynomial in K.[1]

Therefore, we can do the following:

Let $X := \max_i \{|x_i|, |y_i|, |z_i|\}$.

1. Find the planes that constitute the 3D convex hull. A $O(N^4)$ algorithm suffices:
   (a) Fix three non-collinear points and compute the candidate planes.
   (b) If all points lie on one side of the plane, it is valid.
2. For $k = 1, 2, 3$, multiply the coordinates by k, and count the number of lattice points inside (including the boundary) the resulting convex hull.
   - Fix x and y in the range $-X \le x, y \le X$, and compute the intersection with the convex hull.
3. Interpolate the Ehrhart polynomial and evaluate it at K to obtain the answer.

The time complexity is $O(N^4 + NX^2)$.

Instead of counting lattice points inside the convex hull for $k = 1, 2, 3$, we can compute the following three values for $k = 1$ to obtain the Ehrhart polynomial more efficiently:

- Volume of the convex hull
- Number of lattice points inside (including the boundary) the convex hull
- Number of lattice points inside (excluding the boundary) the convex hull

[1] https://en.wikipedia.org/wiki/Ehrhart_polynomial

With further effort, the time complexity can be reduced to $O(N \log N + NX \log X)$. Good luck with the implementation!

Problem I. Peaceful Results

The combinations of their choices that result in a tie are RRR, PPP, SSS, RPS, PSR, SRP, RSP, PRS, and SPR. Let $X_1, X_2, \ldots, X_9$ be the number of times each combination occurs in N games of rock-paper-scissors. Then, there are $\frac{N!}{X_1!X_2!\ldots X_9!}$ possible ways to play the N games that satisfy these constraints.

Let us consider the conditions that $X_1, X_2, \ldots, X_9$ must satisfy. If we consider the total number of times each hand is used, the following equations hold:

$$\begin{cases} X_1 + X_4 + X_7 = A_R \\ X_2 + X_5 + X_8 = A_P \\ X_3 + X_6 + X_9 = A_S \\ X_1 + X_6 + X_8 = B_R \\ X_2 + X_4 + X_9 = B_P \\ X_3 + X_5 + X_7 = B_S \\ X_1 + X_5 + X_9 = C_R \\ X_2 + X_6 + X_7 = C_P \\ X_3 + X_4 + X_8 = C_S \end{cases} \leftrightarrow \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \\ X_9 \end{pmatrix} = \begin{pmatrix} A_R \\ A_P \\ A_S \\ B_R \\ B_P \\ B_S \\ C_R \\ C_P \\ C_S \end{pmatrix}$$

The rank of this 9×9 matrix is 7, so $X_1, \ldots, X_9$ are not uniquely determined. If we define $Y_2 := X_2 - X_1$, $Y_3 := X_3 - X_1$, $Y_5 := X_5 - X_4$, $Y_6 := X_6 - X_4$, $Y_8 := X_8 - X_7$, and $Y_9 := X_9 - X_7$, then

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & -1 & -1 & 1 \\ 0 & 1 & 1 & -1 & -1 & 0 \\ 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 1 & -1 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} Y_2 \\ Y_3 \\ Y_5 \\ Y_6 \\ Y_8 \\ Y_9 \end{pmatrix} = \begin{pmatrix} A_P - A_R \\ A_S - A_R \\ B_P - B_R \\ B_S - B_R \\ C_P - C_R \\ C_S - C_R \end{pmatrix}$$

holds, and since this 6×6 matrix is full rank, $Y_2, Y_3, Y_5, Y_6, Y_8, Y_9$ are uniquely determined. It is necessary that these are integers.

Therefore, $(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9)$ can be expressed as $(x_1, x_1+Y_2, x_1+Y_3, x_4, x_4+Y_5, x_4+Y_6, x_7, x_7+Y_8, x_7+Y_9)$ using three integer variables x_1, x_4, x_7 and six integer constants $Y_2, Y_3, Y_5, Y_6, Y_8, Y_9$. Since all of these must be non-negative and their sum must be N, there exist non-negative integer constants C, l_1, l_4, l_7 such that:

$$\begin{cases} x_1 + x_4 + x_7 = C \\ x_1 \geq l_1 \\ x_4 \geq l_4 \\ x_7 \geq l_7 \end{cases}$$

Under these conditions, x_1, x_4, x_7 can move freely. The sum of the following expression when x_1, x_4, x_7 satisfy the above conditions:

$$\frac{N!}{x_1!(x_1+Y_2)!(x_1+Y_3)!x_4!(x_4+Y_5)!(x_4+Y_6)!x_7!(x_7+Y_8)!(x_7+Y_9)!}$$

should be calculated, which can be done efficiently using FFT. The time complexity is $O(N \log N)$.

Problem J. Make Convex Sequence

Let S be the convex hull of the set of points $\{(i, y) \mid y \geq R_i\}$, and define the sequence $T = (T_1, T_2, \ldots, T_N)$ as $T_i := \min\{y \mid (i, y) \in S\}$. In other words, we take the lower convex hull of R to obtain T.

The T obtained in this way is the "largest" sequence that satisfies all the conditions in the problem statement except for the condition concerning L. More precisely, for any A satisfying the following conditions, we have $A_i \leq T_i$.

- For all $1 \leq i \leq N$, $A_i \leq R_i$ holds.
- For all $2 \leq i \leq N-1$, $A_{i-1} + A_{i+1} \geq 2A_i$ holds.

This can be derived from the definition of a convex hull.

Therefore, the answer is Yes if T satisfies $L_i \leq T_i$, and No otherwise. T can be computed in $O(N)$ time using algorithms such as Andrew's monotone chain.

Problem K. Count Arithmetic Progression

We assume that the indices of L, R, and A are 0-indexed for simplicity. We also define $X := \max_i R_i$.

Fix a common difference d. The possible range for the initial term A_0 can be obtained as follows:

$$\max_i\{-id + L_i\} \leq A_0 \leq \min_i\{-id + R_i\}$$

$-id + L_i$ is a linear function of d for each i. Therefore, the maximum value of these functions, $\max_i\{-id + L_i\}$, forms a piecewise linear curve that is concave downward with respect to d. This curve can be calculated in $O(N)$ time using techniques such as the Convex Hull Trick. Similarly, the conditions for R form a piecewise linear curve that is convex upward with respect to d.

We need to count the number of lattice points within the region enclosed by these two piecewise linear functions. By appropriately dividing the range of d into $O(N)$ intervals, the region enclosed by the curves can be partitioned into trapezoids, and the number of lattice points in each trapezoid can be calculated in $O(1)$ time.

The time complexity is $O(N)$.

Problem L. Many Products

For simplicity, we write $\sum_{(x_1,x_2,\ldots,x_N)\in \boldsymbol{X}}$ as $\sum_{x_1\ldots x_N=M}$.

Expanding $\prod_{i=1}^{N}(x_i + A_i)$ as a sum of 2^N terms $T_1, T_2, \ldots, T_{2^N}$, we have

$$\sum_{x_1\ldots x_N=M}\left(\sum_{s=1}^{2^N} T_s\right) = \sum_{s=1}^{2^N}\left(\sum_{x_1\ldots x_N=M} T_s\right)$$

Therefore, it doesn't matter which $x_1, x_2, \ldots, x_N$ appear in T_s, but rather how many times they appear in total. That is, ignoring the subscripts of $x_1, x_2, \ldots, x_N$, we can write

$$\prod_{i=1}^{N}(x + A_i) = B_0 + B_1 x + \cdots + B_N x^N$$

and the answer can be expressed as

$$\sum_{k=0}^{N} B_k \left(\sum_{x_1 \dots x_N = M} x_1 x_2 \dots x_k \right)$$

Here, $\left(\sum_{x_1 \dots x_N = P} x_1 x_2 \dots x_k \right)$ is multiplicative with respect to P. Therefore, if we factorize M into primes $M = p_1^{e_1} \dots p_c^{e_c}$, we have:

$$\left(\sum_{x_1 \dots x_N = M} x_1 x_2 \dots x_k \right) = \prod_{j=1}^{c} \left(\sum_{x_1 \dots x_N = p_j^{e_j}} x_1 x_2 \dots x_k \right)$$

We can express the sum as a simple formula using binomial coefficients. Specifically, we have:

$$\sum_{x_1 \dots x_N = p_j^{e_j}} x_1 x_2 \dots x_k = \begin{cases} \binom{e_j+N-1}{N-1} & (k=0) \\ \sum_{d=0}^{e_j} p_j^d \binom{d+k-1}{k-1} \binom{e_j-d+N-k-1}{N-k-1} & (0<k<N) \\ p_j^{e_j} \binom{e_j+N-1}{N-1} & (k=N) \end{cases}$$

Computing $B_0, \dots, B_N$ can be done using convolution in $O(N(\log N)^2)$ time. Factoring M into primes takes $O(\sqrt{M})$ time, and other computations can be done in $O(N \log M)$ time.

Problem M. Colorful Graph

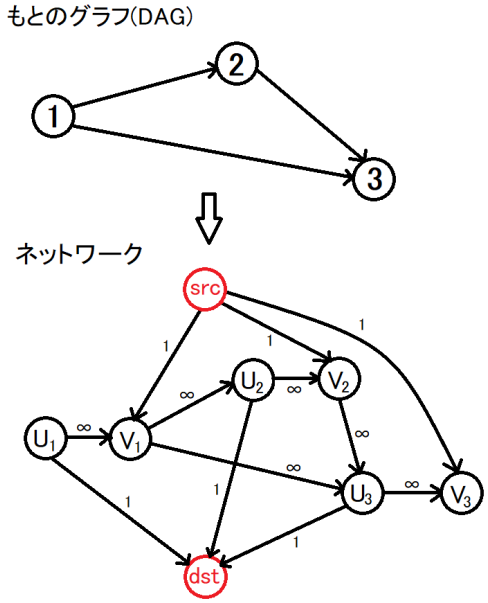
You can paint all vertices in the same color if they belong to the same strongly connected component. Compress the strongly connected components into a single vertex to make the graph a DAG.

When there are edges $i \to j$ and $j \to k$, add an edge $i \to k$. This reduces the problem to the minimum path cover problem in a directed acyclic graph[2], which can be solved by maximum flow. However, the number of edges can reach $\Theta(N^2)$, which will result in insufficient memory.

We can reduce edges as follows:

- Prepare the following vertices:
  - $\mathrm{src}, \mathrm{dst}$
  - $U_1, \cdots, U_N$
  - $V_1, \cdots, V_N$
- Add an edge from src to all V_i with capacity 1.
- Add an edge from all U_i to dst with capacity 1.
- For each edge $i = 1, \dots, M$, add an edge from V_{A_i} to U_{B_i} with capacity ∞.
- For all $i = 1, \dots, N$, add an edge from U_i to V_i with capacity ∞.

[2] https://en.wikipedia.org/wiki/Maximum_flow_problem#Minimum_path_cover_in_directed_acyclic_graph



The maximum flow of this network is N, and the number of edges is at most $3N + M$, so it can be solved by Ford-Fulkerson algorithm with time complexity $O(N(N + M))$ and space complexity $O(N + M)$.

After obtaining the maximum flow, we can color the graph by looking at the flow on each edge with time complexity $O(N^2)$ and space complexity $O(N)$.

Finally, restore the original graph from the compressed graph obtained by the strongly connected component decomposition and output it. The overall time complexity is $O(N(N + M))$, and the space complexity is $O(N + M)$.

Problem N. XOR Reachable

We consider traversing $D = 0, 1, \ldots, 2^{30} - 1$ in order. For each edge j, the set of D such that $(C_j \oplus D) < K$ can be decomposed into at most 30 intervals. Therefore, this problem can be reduced to an offline dynamic connectivity problem where there are $30M$ edge additions and deletions.

This can be solved in $O((Q+M \log N) \log MAX)$ time using undoable Union-Find and divide-and-conquer.

Problem O. Jewel Game

Noting that $K \leq 10$, we want to do the following DP.

dp[set of remaining gems][position of the current player][position of the other player] = (the score of the game)

However, there are loops in the calculation, so we cannot easily determine the values.

One way to analyze games with loops is to use backward induction, which is usually used to determine wins, losses, or draws, but we can use it to determine the score of the game.

If there are loops in the calculation, the set of remaining gems does not change within the loop. Therefore, we can determine the value by performing backward induction in order of increasing size of the set of remaining gems.

In other words, we can do the following:

- Perform backward induction 2^K times, in order of increasing size of the set of remaining gems.
    1. If there is a vertex for which the values of all its outgoing edges are already determined, then we can determine the value of that vertex.

2. If there are still vertices whose values have not been determined, then examine the destination of all outgoing edges from those vertices and choose the highest value, if positive. Then, determine the value of the vertex from which that edge originates.
3. If the value still cannot be determined, then it is 0.

The time complexity is $O(2^K N(N+M)\log N)$.