Problem A. A Calibration Lab

Problem Overview

- There are N pairs of integers $(A[i], B[i])$.
- Count the number of ordered pairs of two different indices (i, j) such that the four integers $\{A[i], B[i], A[j], B[j]\}$ can be partitioned into two identical multisets.

Solution

- First, ensure $A[i] \leq B[i]$ for each i.
- The required ordered pairs fall into two types:
  1. Both pairs consist of the same integer: (a, a) and (b, b).
  2. The two pairs are equal: (a, b) and (a, b), where $a \neq b$.
- For type 1, maintain a counter for pairs with identical elements.
- For type 2, use a map or similar structure to count occurrences of each distinct pair (a, b) with $a < b$.

Problem B. Box

Problem Overview

- Rearrange a sequence A of N non-negative integers.
- For each index i, add $\left\lfloor \frac{\sum_{j<i} A[j]}{A[i]} \right\rfloor$ to the total cost.
- Find the minimum possible total cost.

Solution

- Sort A in non-decreasing order.
- **Proof (Exchange Argument):** Suppose $A[i] > A[i+1]$. Swapping these two elements affects the cost as follows:
  - For $A[i]$: The numerator increases by $A[i+1]$, so the change in its term is $\leq +1$.
  - For $A[i+1]$: The numerator decreases by $A[i]$, so the change in its term is ≤ -1.

  Thus, the total change in cost is ≤ 0, meaning the swap does not increase the cost.
- Therefore, the optimal arrangement is the sorted order.

Problem C. Chairs and Columns

Problem Overview

- Given an $H \times W$ grid. Some cells are selectable, some are not.
- Cells that are vertically or horizontally adjacent cannot be selected simultaneously.
- Goal: Select as many cells as possible (maximum independent set).
- For each cell, determine if it can be included in some maximum independent set.
- Constraints: $H \leq 400$, $W \leq 400$.

Key Known Fact

A grid graph is bipartite. (Proof: Color like a checkerboard.) Thus, it suffices to solve the following problem: Given a bipartite graph $G = (L, R; E)$, determine for each vertex whether it can be part of some maximum independent set. Let $V = L \cup R$.

Important Fact for Bipartite Graphs

Let M be the size of a maximum matching and I the size of a maximum independent set. Then,

$$M + I = |V|.$$

Brief proof: Show that maximum matching = minimum vertex cover. A minimum vertex cover can be expressed in a "burn/flood-fill" form, which yields the graph for finding a maximum matching.

Observations

For $v \in V$, the following statements are equivalent:

1. There exists a maximum independent set containing v.
2. Removing v and its neighbors $\delta(v)$ from G reduces the size of a maximum independent set by 1.
3. Removing v and its neighbors $\delta(v)$ from G reduces the size of a maximum matching by $|\delta(v)|$.
4. Removing only the neighbors $\delta(v)$ from G reduces the size of a maximum matching by $|\delta(v)|$.

We will consider the last condition.

Matching and Residual Graph

Fix an arbitrary maximum matching M of G. Let $U \subseteq L$ be the set of left vertices unmatched in M. Construct a directed residual graph G' where:

- Edges in M are directed from right to left.
- Other edges are directed from left to right.

Condition for a Left Vertex

For $v \in L$, the following are equivalent:

- Removing v reduces the maximum matching size by 1.
- v is **not** reachable from any vertex in U in G'.

Reasoning: If v is reachable from U, we can swap edges along the alternating path to create a new maximum matching that does not use v, so removing v does not decrease the matching size.

Proof Sketch

Suppose removing v does not decrease the maximum matching size. Show this implies v is reachable from U (for $v \notin U$). Take a new maximum matching M' that does not use v. Consider the symmetric difference of M (red) and M' (blue). Since each vertex has degree at most 2, it consists of simple paths and cycles. By maximality, there is no path starting and ending with blue. Since $|M| = |M'|$, there is also no path starting and ending with red. Because $v \notin U$ and M' does not use v, v is incident only to a red edge. Thus, there exists a simple path ending at v with pattern: blue $\to$ red $\to \cdots \to$ blue $\to$ red. The starting vertex is incident only to a blue edge, so it belongs to U. This path corresponds to a path from U to v in G'.

Implication

If removing $v \in L$ reduces the maximum matching size by 1, then removing v from G' yields a valid residual graph. Removing v does not increase the set of vertices reachable from U.

Conclusion for Right Vertices

For $u \in R$, the following are equivalent:

- There exists a maximum independent set containing u.
- No vertex in $\delta(u)$ (its neighbors) is reachable from U in G'.

Complexity

A maximum bipartite matching can be found in $O(E\sqrt{V})$ time. Performing BFS once gives answers for all right vertices; doing it twice gives answers for all vertices. Total complexity: $O(HW\sqrt{HW})$.

Additional Note (DM Decomposition)

The decomposition of the vertex set obtained by contracting SCCs in the residual graph is called the Dulmage–Mendelsohn (DM) decomposition. It provides various insights about bipartite matchings and can also be used here.

Problem D. Do Count The Inversions

Problem Overview

- Given a string S and an integer K.
- Find the number of inversions in the suffix array of the string formed by repeating S K times.

Solution

1. **Preprocess** S: Ensure S has no period (use Z Algorithm, $O(N)$).
2. **Build Suffix Array for** $S+S$: Construct the suffix array P for the string $S+S$. This can be done with SA-IS in $O(N)$, or using cyclic shifts in $O(N \log N)$.
3. **Compute Inversion Contribution**: Traverse P from left to right while maintaining a Binary Indexed Tree (BIT) to compute contributions to the total inversion count in $O(N \log N)$. For each position i:
   - If $P[i] < N$:

     $$\text{Add } \frac{(K-1)(K-2)}{2} \times \text{BIT.sum}(0, P[i]) \qquad \text{(Between-blocks contribution 1)}$$
     $$\text{Add } \frac{K(K-1)}{2} \times \text{BIT.sum}(P[i]+1, N) \qquad \text{(Between-blocks contribution 2)}$$
     $$\text{Add } (K-1) \times \text{BIT.sum}(N, 2N) \qquad \text{(Block vs } P[i] > N(K-1) \text{ contribution)}$$
     $$\text{Add } \frac{(K-1)(K-2)}{2} \qquad \text{(Within-block contribution)}$$

   - If $P[i] \geq N$:

     $$\text{Add BIT.sum}(P[i]+1, 2N)$$

   Update BIT with $P[i]$ after processing.

Problem E. Earth and Circles

Problem Overview

- There are N non-intersecting circles on a sphere.
- The distance between two points on the sphere is defined as the minimum number of circles crossed when moving between them.
- For a chosen point, consider the maximum distance to all other points.
- Find the minimum possible value of this maximum distance over all points.
- $N \leq 2000$.
- Each circle is given as the intersection of the sphere with a plane defined by its normal vector.

Solution

- Each circle divides the sphere into two regions. Take the smaller region as a "cell".
- The set of these cells (including the whole sphere) forms a laminar family, which induces a tree with $N + 1$ vertices.
- The answer is half the diameter of this tree, rounded up to the nearest integer.

Constructing the Tree

- **Containment relation:** For two cells, check whether a representative point of one lies outside the plane of the other.
- **Parent determination:** For each cell, its parent is the most deeply contained cell among those that contain it.
- If no containing cell exists, its parent is the root (the whole sphere).

Problem F. Find the Minimal Number of Moves

Problem Overview

- A huge binary tree is traversed under the following rules:
- On the i-th move, choose one of:
  - Move to the left child if $S = L$, or to the right child if $S = R$.
  - Move to the parent (only if the parent exists).
- For given nodes u and v, determine if v is reachable from u, and if so, find the minimum number of moves required.
- There are Q queries; u and v can be very large.

Solution

A feasible optimal route follows the path:

$$u \to \mathrm{lca}(u, v) \to v$$

- $u \to \mathrm{lca}(u, v)$: Move upward to the lowest common ancestor as quickly as possible.
- $\mathrm{lca}(u, v) \to v$: Two cases when descending:
  - If the required child direction matches S, move directly.
  - Otherwise, move to the opposite child first, then return upward in the next step.
- Any optimal movement can be transformed into this form without losing optimality.

Implementation

1. **Precomputation:** For each $i = 1, 2, \ldots, N$, compute the first time after i when we can move left/right using a two-pointer method. Note: Only consider positions with the same parity as i.
2. **Query Processing:** Simulate the movement directly. Complexity per query is $O(\log u + \log v)$. Even with naive LCA calculation (repeatedly halving the larger of u and v until they match), the solution passes with careful implementation.

Problem G. Grid And Its Beauty

Problem Overview

- Given a grid with white/black cells.
- Repeatedly perform the following operation:
  - Choose a shortest path from $(1, 1)$ to (H, W) and flip the colors (white $\leftrightarrow$ black) of all cells on that path.
- Goal: Maximize the number of black cells in the final grid.
- Additionally, handle online queries of two types:
  - Flip all cells in a given column.
  - Flip all cells in a given row.
- After each query, output the answer for the current grid.
- Constraints: $H, W, Q \leq 2 \times 10^5$.

Solution

Case: Initially all cells are white

Consider two specific shortest paths:

1. $(1, 1) \to \cdots \to (i, j) \to (i + 1, j) \to (i + 1, j + 1) \to \cdots \to (H, W)$
2. $(1, 1) \to \cdots \to (i, j) \to (i, j + 1) \to (i + 1, j + 1) \to \cdots \to (H, W)$

Applying both flips only cells $(i + 1, j)$ and $(i, j + 1)$ twice, i.e., flips them exactly twice (no net change), while flipping all other cells once. This allows flipping any chosen pair of adjacent diagonal cells.

Greedy construction

By repeatedly applying such paired operations, we can turn the entire $(H-1)\times(W-1)$ top-left rectangular region black. The remaining cells form a path along the last row and last column. The state (color) of each remaining cell is determined by the parity of the number of black cells among cells with the same $i + j$ value.

General case (initial grid may contain black cells)

The effect of initial black cells can be incorporated by flipping the parity condition for remaining cells according to the parity of the total number of initial black cells.

Handling queries

Each row or column flip affects the parity of certain diagonal groups. This reduces to a **range flip + range sum** problem over an array indexed by $i + j$. A lazy segment tree can process each query in $O(\log(H + W))$ time.

Complexity

Total time complexity: $O((H + W + Q)\log(H + W))$.

Problem H. Handle The Queries On Tree

Problem Overview

- Given a tree with N vertices, each vertex has a color.
- Answer Q queries of the form:

  $$(u_1, v_1, u_2, v_2)$$

  Find the largest integer $K \leq N$ such that for every $j = 1, 2, \ldots, K$, the number of vertices of color j on the path $u_1 - v_1$ equals the number of vertices of color j on the path $u_2 - v_2$.

Observations

Special Case: Checking $K = N$

This reduces to checking whether the multiset of colors on the two paths are identical. We can assign a random hash value h_c to each color c. Define the value of a vertex of color c as h_c. Then the two multisets are identical if and only if the sum of vertex values on the two paths are equal (with high probability).

General Case

Since the condition is monotonic in K (if it holds for K, it holds for any smaller K), we can use binary search. To process all queries efficiently, use **parallel binary search**.

Solution Approaches

Method 1: Euler Tour + Segment Tree

- For each color k, maintain a segment tree storing the sum of hash values of vertices of that color.
- Process vertices in DFS order, adding/subtracting hash values appropriately when entering/leaving a vertex.
- Need to handle LCA carefully when adding vertex values.

- Complexity: $O((N+Q)\log^2 N)$.
- A similar approach can be implemented using rolling hash.

Method 2: HLD + Segment Tree

- Maintain a segment tree over the heavy-light decomposition, where leaf k stores the hash value of vertex k (or 0 if not yet active).
- Activate vertices in increasing order of color, updating the segment tree.
- For each query, compare the sum of values on the two paths.
- Complexity: $O((N+Q)\log^3 N)$, but with BIT and careful implementation, it can pass.

Summary

- The solution idea is relatively straightforward, but implementation is heavy.
- It's useful to have well-tested library code for tree path queries (HLD/Euler tour + segment tree).
- Implementation choices affect code length and constant factors significantly.
- Team assignment for implementation may greatly affect solving time.
- Note: An $O((N+Q)\log N)$ solution also exists.

Problem I. Inside Yamanote

Problem Overview 1

- Characterize the feasible distance arrays d.
- Necessary and sufficient conditions:
  1. For all i, $|d[i]-d[i+1]| \le L[i]$.
  2. For some i, $d[i]=0$.
- Necessity follows from triangle inequality / shortest-path properties.
- Sufficiency: If $d[i]=0$ for some i, treat i as the capital and connect each j with an edge of weight $d[j]$.

Problem Overview 2

Let $w[i]$ be the net count (occurrences as X minus occurrences as Y) at vertex i. The problem becomes:

- Find a sequence p such that:
  1. For all i, $|p[i]-p[i+1]| \le L[i]$.
  2. For some i, $p[i]=0$.
- Maximize $\sum p[i]\cdot w[i]$, given that $\sum w[i]=0$.

Problem Overview 3

Since $\sum w[i] = 0$, the objective is invariant under adding a constant to all $p[i]$. Thus, the condition "$p[i] = 0$ for some i" can be dropped. The problem simplifies to:

- Find a sequence p satisfying $|p[i] - p[i+1]| \leq L[i]$ for all i.
- Maximize $\sum p[i] \cdot w[i]$, where $\sum w[i] = 0$.

Problem Overview 4

Take the LP dual. The dual problem is:

- Construct an integer sequence B of length N.
- Constraints: $B[i] - B[i-1] \geq w[i]$ for all i.
- Minimize $\sum L[i] \cdot B[i]$.

Problem Overview 5

Since $\sum w[i] = 0$, the constraints $B[i] - B[i-1] \geq w[i]$ must hold with equality. Define $B'[0] = 0$, $B'[i] = w[i] + B'[i-1]$ for $i \geq 1$. The problem reduces to:

- Given sequences L and B' of length N.
- Choose a real k to minimize $\sum L[i] \cdot (B'[i] + k)$.

Solution

- The optimal k is the weighted median of $B'[i]$ with weights $L[i]$.
- Implementation: Sort indices by $B'[i]$, then compute prefix sums of $L[i]$ and $L[i] \cdot B'[i]$.
- Use binary search on a segment tree to find the median in $O(\log N)$ per query (sublinear overall).

Problem J. Juggle With Tree And Queries

Problem Overview

Construct a weighted tree T satisfying:

- Number of vertices: between 2 and 65.
- Edge weights: between 0 and 10^9.
- Every integer from 1 to 10^9 (inclusive) can be expressed as the sum of at most 5 path-weights in T.

Basic Plan

Since $10^9 < 2^{30}$, we first aim to represent numbers up to 2^{30}. We can use 5 paths, and note that $2^{30} = (2^6)^5 = 64^5$. Thus, try to find a small tree where a single path can generate numbers in $[1, 64)$. Search for such a small tree (e.g., path, star) by brute-force or manual construction.

Small Tree Example

A star graph with 15 vertices and 14 edges, edge weights: $1, 2, \ldots, 7, 8, 16, \ldots, 56$.

Vertex Count Reduction

Using 5 copies of such a star would require 75 vertices. Sharing the central vertex reduces this to 71 vertices.

Further Observation

We can also produce 64 itself. Then, after the first tree, each subsequent tree can have one fewer edge. Total edges: $14 + 13 \times 4 = 66$ edges (67 vertices).

Better Construction

Observation: The left-side tree alone can produce $[1, 69]$. Define:

- Tree A (14 edges): produces $[1, 69]$.
- Tree B (13 edges): produces $[1, 61]$.

Combine them as:

1. A: $[1, 69)$.
2. $A \times 69$: $[69, 69^2)$.
3. $B \times 69^2$: $[69^2, 69^2 \times 61)$.
4. $B \times 69^2 \times 61$: $[69^2 \times 61, 69^2 \times 61^2)$.
5. $B \times 69^2 \times 61^2$: $[69^2 \times 61^2, 69^2 \times 61^3]$.

Since $69^2 \times 61^3 = 1,080,656,541 > 10^9$, the range is covered. Edge count: $14 + 14 + 13 + 13 + 13 = 67$.

Further Reduction

By intentionally not using the maximum of each tree, we can reduce edges in later trees similarly to the 64^5 method. This leads to: $14 + 13 + 12 + 12 + 12 = 63$ edges (64 vertices).

Alternative Geometric Series Method

Choose $r > 1$ and construct a path graph with edge weights r^k. For $r \approx 1.6742$ and $0 \le k < 39$, we can cover up to $1,006,652,149$ (40 vertices). Setting $r = 1.36$ for $0 \le k < 64$ allows fast computer search.

Remarks on Problem-Solving Strategy

- How to use limited computational resources (one PC)?
- Exploration methods are non-trivial.
- How to allocate human resources (three team members)?
- Use standings and perceived difficulty to choose problems.
- The problem looks deceptively approachable; avoid being trapped.

Problem K. Knotwork Machine

Problem Overview

We consider a program with:

- An integer variable x with domain $[0, N)$.
- A stack s.
- A weight multiplier w for transitions.

Initialization:

1. $x = 0$;
2. $w = 1$;
3. $s = \{\}$ (empty stack).

Step Execution

There are N steps. At each step t, choose a parameter y_t:

- If $0 \le y_t$:

$$\begin{aligned} w &:= w \cdot A[t][x][y]; \\ s&.\text{push}(x); \\ x &:= y; \end{aligned}$$

- If $y_t = -1$:

$$\begin{aligned} &\text{assert}(!s.\text{empty}()); \\ w &:= w \cdot A[t][x][s.\text{top}()]; \\ x &:= (x + s.\text{top}()) \bmod N; \\ s&.\text{pop}(); \end{aligned}$$

The value w is the product of transition weights along an execution path.

Goal

Sum the w over all possible valid execution sequences (i.e., over all non-deterministic choices of y_t at each step). Essentially, compute the weighted sum of all feasible transition sequences without omission or duplication.

Solution

Use interval DP over time with complexity $O(N^6)$: Define:

- $f(l, r, x_0, x_1)$: from time l (push) to time $r-1$ (pop of the same element).
- $g(l, r, x_0, x_1)$: from time l (push) to time $r-1$ (pop of an element at the same depth, with no intermediate removal of the element pushed at l).
- $h(l, x_0, x_1)$: the element pushed at time l is never removed in the middle.

Here, x_0 is the value of x before the interval, x_1 after the interval.

Relation to Non-deterministic Turing Machines and Polynomial-time DP

Some classes of languages accepted by non-deterministic Turing machines are subclasses of P, provable via dynamic programming. Here, we have a non-deterministic program with $O(\log N)$ memory and one stack (weighted transitions), which places it in the complexity class LOGCFL.

Memory	Stack	DP	Language class
$O(1)$	–	Ear DP	Regular languages
$O(1)$	1	Interval/Tree DP	CFL
$O(\log N)$	–	Standard DP	NL
$O(\log N)$	1	Tree/Interval DP	LOGCFL

Problem L. Let's Find The Sum

Problem Overview

Given an integer N, compute

$$S(N) = \sum_{i=1}^{N} \sum_{j=1}^{N} \left\lfloor \frac{N}{ij} \right\rfloor .$$

Solution 1

Let:

- $d(n)$: number of positive divisors of n.
- $D(n) = \sum_{k=1}^{n} d(k)$.

Then,

$$S(N) = \sum_{k=1}^{N} d(k) \left\lfloor \frac{N}{k} \right\rfloor = \sum_{k=1}^{N} d(k) \sum_{i=1}^{\lfloor N/k \rfloor} 1 = \sum_{ik \le N} d(k) = \sum_{k=1}^{N} D\left(\left\lfloor \frac{N}{k} \right\rfloor \right).$$

Split the sum by the size of k relative to $N^{1/3}$:

- For small k, compute $\lfloor N/k \rfloor$ directly (each in $O(\sqrt{N/k})$).
- For large k, precompute $D(\cdot)$ and group by the value of $\lfloor N/k \rfloor$.

Total complexity: $O(N^{2/3})$.

Solution 2

Observe that the desired sum equals the number of ordered triples (a, b, c) of positive integers such that $abc \le N$. This is a known problem (e.g., AtCoder ABC227 C). Under the constraint $a \le b \le c$, it can be counted in $O(\sqrt[3]{N})$ or $O(N^{2/3})$ as well.