Problem Analysis

The 2024 ICPC Asia Pacific Championship

This is an analysis of some possible ways to solve the problems of The 2024 ICPC Asia Pacific Championship. Since the purpose of this analysis is mainly to give the general idea to solve each problem, we left several (implementation) details in the discussion for reader's exercise. If some of the terminology or algorithms mentioned below are not familiar to you, your favorite search engine should be able to help. If you find an error, please send an e-mail to icpc-apac-judges@googlegroups.com about it.

Problem Title		**Problem Author**	**Analysis Author**
A	Antiparticle Antiphysics	Kevin Charles Atienza	Kevin Charles Atienza
B	Attraction Score	Jonathan Irvin Gunawan	Jonathan Irvin Gunawan
C	Bit Counting Sequence	Mitsuru Kusumoto	Mitsuru Kusumoto
D	Bánh Bò	Kevin Charles Atienza	Kevin Charles Atienza, Mitsuru Kusumoto
E	Duplicates	Mitsuru Kusumoto	Mitsuru Kusumoto
F	Forming Groups	Jonathan Irvin Gunawan	Jonathan Irvin Gunawan
G	Personality Test	Meng-Tsung Tsai	Mitsuru Kusumoto
H	Pho Restaurant	Ammar Fathin Sabili	Ammar Fathin Sabili
I	Symmetric Boundary	Mitsuru Kusumoto	Mitsuru Kusumoto
J	There and Back Again	Peter Rossmanith	Jonathan Irvin Gunawan
K	Tree Quiz	William Gan	Jonathan Irvin Gunawan
L	XOR Operations	Prabowo Djonatan, Muhammad Ayaz Dzulfikar	Mitsuru Kusumoto
M	Zig-zag	Kevin Charles Atienza	Kevin Charles Atienza

The contest judges would also like to thank the following for their valuable feedback to the tasks: Ashar Fuadi, Budiman Arbenda, Edbert Geraldy Cangdinata, Hocky Yudhiono, Lim Li, Masaki Nishimoto, Prof. Takashi Chikayama, R. Fausta Anugrah Dianparama, Riku Kawasaki, William Gan, Yehezkiel Raymundo Theodoroes, and Yui Hosaka.

A Antiparticle Antiphysics

Using the notation X^n to denote the string X repeated n times and ε for the empty string, we can state our rules as P $\to$ APA, A $\to$ PAP, A$^a \to \varepsilon$, and P$^p \to \varepsilon$. Here are more operations we can do:

- APA $\to$ P, because APA $\to$ AAPAA $\to$ AAAPAAA $\to \ldots \to$ A^{a}PA$^a \to$ P. Similarly, PAP $\to$ A.
- AA $\to$ PP, because AA $\to$ APAP $\to$ PP. Similarly, PP $\to$ AA.
- PA $\to$ APPP, because PA $\to$ PPAP $\to$ AAAP $\to$ APPP. Similarly, AP $\to$ PAAA.
- P $\to$ PAAAA, because P $\to$ APA $\to$ PAAAA. Similarly, A $\to$ APPPP, P $\to$ AAAAP, and A $\to$ PPPPA.
- AAAA $\to$ PPPP, because AAAA $\to$ PPAA $\to$ PPPP. Similarly, PPPP $\to$ AAAA.

We can summarize the last few operations as $\varepsilon \to$ A^4 and $\varepsilon \to$ P^4. By repeatedly using these along with A$^a \to \varepsilon$ and P$^p \to \varepsilon$, we get $\varepsilon \leftrightarrow$ A$^{\gcd(a,4)}$ and $\varepsilon \leftrightarrow$ P$^{\gcd(p,4)}$, where we begin writing "$\leftrightarrow$" because we notice that all operations are now reversible. Also, without loss of generality, we can replace a and p by $\gcd(a, 4)$ and $\gcd(p, 4)$ respectively and then swap them if necessary to assume that $a \mid p \mid 4$.

We can then show using the above operations that any string is equivalent to one of A, AA, AAA, AAAA, P, AP, AAP, and AAAP. Let's call these the eight *special strings*. Some of these can be reduced to each other further in case a and/or p are smaller than 4. But since $a \mid p \mid 4$, there are only a few possibilities, and we can just check one by one. The most substantial case is when $a = p = 4$, in which one can show that the eight special strings are distinct because they form a structure isomorphic to the quaternion group Q_8, the eight-element subset $\{1, i, j, k, -1, -i, -j, -k\}$ of the quaternions under multiplication. The other cases are just quotients of Q_8 by some of its subgroups. All in all, we get four distinct groups: the trivial group if $a = p = 1$, $\mathbb{Z}/2$ if $a = 1$ and $p > 1$, $(\mathbb{Z}/2)^2$ if $a = 2$, and Q_8 if $a = 4$.

We have now shown that a string is convertible into another string iff their corresponding group elements are the same, and the above already shows one way to do so. But there are many other ways, some taking fewer steps than others. Here's my procedure to reduce a string X to its corresponding special string:

```
while X is not special or len(X) > gcd(a, 4)
    if len(X) > a and A**a is in X
        remove the A**a
    else if PA is in X
        convert the rightmost PA to APAA
    else if PP is in X
        convert the leftmost PP to AA
    else
        insert AAAA in front of X
```

After this procedure, X will be a special string. To convert between equivalent special strings, we can just hardcode the conversion procedures—their overall contribution to the number of steps will be negligible anyway. There's also a similar procedure to convert back to a given string from its special string. These procedures seem very efficient and seem to require less than $10\,000$ operations even for strings up to length 50.

Explicit bounds can also be obtained by carefully counting the number of steps of each intermediate operation above. For example, setting $n = 50$ and $k = 20$, we see that APA $\rightarrow$ P takes $k + O(1)$ steps, PAAA $\rightarrow$ AP takes $3k + O(1)$, etc. By working through the steps like this, one can show that a slightly altered version of the algorithm needs at most $8nk + 24k^2 + O(n + k)$ steps. This is $< 20\,000$ in the worst case, so it passes comfortably.

B Attraction Score

Because the graph is planar, any subset of v nodes (for $v > 2$) nodes can only contain at most $3v - 6$ edges between them.

Observation B.1. *It is sufficient to check only clique subgraphs or cliques with one missing edge. We define this as our search space.*

Proof. For $v > 6$, the score of any subgraph is not positive since $(3v - 6) - (\binom{v}{2} - (3v - 6))^2 < 0$.

For $v = 6$, the maximum number of edges possible is 12, which means there are at least 3 missing edges (there are 15 edges in K_6). Let x be a node with at least one missing edge, so its degree is at most 4. Removing x decreases the number of edges by at most 4 but reduces the number of missing edges to 2 or less, hence saving a penalty of at least $(3^2 - 2^2) \times 10^6 = 5 \times 10^6$, which is more than the score of 4 edges, so it's better to remove node x. Thus, we can focus on $v \leq 5$.

For $v = 5$, with two missing edges, the score is not better than if the node of minimum degree is removed. This reduced graph (without the node of minimum degree) is in our search space. With more than two missing edges, the score is negative.

For $v < 5$, the score is not positive if there is more than one missing edge. □

We can enumerate all $O(n)$ clique subgraphs of a planar graph in $O(n \log n)$ time. To do that, we look at the node with the minimum degree (call this node u). Let the set of neighbours of node u be S_u. Since the graph is planar, it is guaranteed that $|S_u| \leq 5$, so we can obtain all cliques containing node u in $O(1)$. We then continue the algorithm by enumerating cliques without node u. We can do so by removing node u from the graph and recursively doing the algorithm on the rest of the graph, which is still planar.

Thus, we can compute the maximum score among all clique subgraphs. We now focus on computing the maximum score among all subgraphs with 5 nodes and one missing edge. A similar argument can be applied to subgraphs with fewer nodes.

Consider node u in the recursive algorithm above. The hard case is when the missing edge is **not** incident to u, because then one node in the subgraph is not in S_u. Let nodes a, b, and c be the nodes in the subgraph that are in S_u. To find the missing node d that maximizes the score, we can note that nodes a, b, c, and d form a clique. We can also assume that there is no edge connecting nodes u and d, so the optimal choice for d is independent of u. Therefore, we can find this node d in $O(1)$ by precomputing all $O(n)$ cliques.

This solution runs in $O(n \log n)$ time.

C Bit Counting Sequence

Observation C.1. *For a non-negative integer y, if $p(y) - p(y+1) = t \geq 0$, the rightmost $(t+1)$ bits in the binary representation of y are all 1. Additionally, the $(t+2)$-th bit of y from the right is 0.*

For example, consider $y = 87 = (1010111)_2$. Here, $t = p(y) - p(y+1) = 5 - 3 = 2$. As the observation states, the rightmost three bits of y are 1, and the fourth bit of y from the right is 0.

Let k be the smallest index such that $a_k = \max(a_1, \ldots, a_n)$. Also, let x denote the solution to this problem.

When $k < n$, let $t = a_k - a_{k+1}$. It's worth noting that $t \geq 0$ according to the definition of k. Based on the previous observation, the binary representation of $x + k - 1$ should be like

$$x + k - 1 = (\ldots 0\overbrace{1 \ldots 1}^{t+1})_2.$$

The bits starting from the $(t+3)$-th position from the right of $x + k - 1$ cannot be determined solely from this observation. However, considering that a_k is the largest value among $a_1, \ldots, a_n$, for $s = a_k - (t+1)$,

$$x + k - 1 = (\overbrace{1 \ldots 1}^{s}0\overbrace{1 \ldots 1}^{t+1})_2.$$

is the smallest possible value for $x + k - 1$. Consequently, we can compute the smallest possible value of x. If any of the following conditions hold: $s < 0$, $x < 0$, or $p(x + i - 1) \neq a_i$ for some i, then the solution does not exist.

Similarly, when $k = n$, the smallest possible value for $x + n - 1$ is $2^{a_n} - 1$. The runtime complexity is $O(n)$.

D Bánh Bò

Let $g_{i,j} \in \{0, 1\}$ be the value at row i and column j. We use zero-indexing ($0 \leq i < r$ and $0 \leq j < c$). Also, for simplicity sake, let $r_0 = 6$ and $c_0 = 7$.

Observation D.1 ("Quadrangle Equation"). *For $r_0 \leq i < r$ and $c_0 \leq j < c$, $g_{i,j} + g_{i-r_0,j-c_0} = g_{i-r_0,j} + g_{i,j-c_0}$.*

Proof. Consider a $(r_0 + 1) \times (c_0 + 1)$ grid. Its four $r_0 \times c_0$ subgrids overlap. Adding the top-left and the bottom-right subgrids and subtracting the bottom-left and the top-right subgrids will yield the equation. □

Observation D.1 has several nice consequences. For instance, in any $(r_0 + 1) \times (c_0 + 1)$ subgrid you can check that the only possibilities for the four corners are:

0	⋯	0	1	⋯	1	0	⋯	1	1	⋯	0	0	⋯	0	1	⋯	1
⋮		⋮	⋮		⋮	⋮		⋮	⋮		⋮	⋮		⋮	⋮		⋮
0	⋯	0	1	⋯	1	0	⋯	1	1	⋯	0	1	⋯	1	0	⋯	0

Since these six "tiles" can only be assembled in limited ways, we eventually observe that:

Observation D.2. *In the lattice* $\{(i + r_0a, j + c_0b) \mid 0 \le a < r/r_0, 0 \le b < c/c_0\}$ *for each* $(i, j) \in \{0, \ldots, r_0 - 1\} \times \{0, \ldots, c_0 - 1\}$*, at least one of the following holds: (1) for each row, all cells in it contain the same bit, or (2) for each column, all cells in it contain the same bit. (Both can be true.)*

Observation D.1 also tells us that after we choose the topmost r_0 rows and leftmost c_0 columns, all the other cells are completely determined. Thus, all that remains is to choose the cells in those rows and columns while satisfying the following necessary conditions, which can also be shown to be sufficient by induction:

- For each (i, j) such that $0 \le i < r_0$ and $0 \le j < c_0$, one (or both) of the following hold: (A) $g_{i,j} = g_{i+r_0,j} = g_{i+2r_0,j} = g_{i+3r_0,j} = \ldots$ and/or (B) $g_{i,j} = g_{i,j+c_0} = g_{i,j+2c_0} = g_{i,j+3c_0} = \ldots$,
- $g_{i,0} + g_{i,1} + \ldots + g_{i,c_0-1} = g_{i+r_0a,0} + g_{i+r_0a,1} + \ldots + g_{i+r_0a,c_0-1}$ for $0 \le i < r_0$ and $0 \le a < r/r_0$, and
- $g_{0,j} + g_{1,j} + \ldots + g_{r_0-1,j} = g_{0,j+c_0b} + g_{1,j+c_0b} + \ldots + g_{r_0-1,j+c_0b}$ for $0 \le j < c_0$ and $0 \le b < c/c_0$.

For further clarity, let's define a pair (i, j) as being *constant* if the values of $g_{i+r_0a,j+c_0b}$ remain consistent for all a and b. If condition (A) is satisfied for (i, j), we denote the pair as being *horizontally striped*. Likewise, if condition (B) is met for (i, j), we denote the pair as being *vertically striped*. Note that these conditions for (i, j)–being constant, being horizontally striped, and being vertically striped–are not mutually exclusive. For example, being horizontally striped can include the case of being constant.

For the top-left $r_0 \times c_0$ cells, let's assume that each cell is fixed as one of constant, horizontally striped, or vertically striped. We will now determine the number of possible value assignments for the topmost r_0 rows and the leftmost c_0 columns. For simplicity, let $p = r/r_0 - 1$ and $q = c/c_0 - 1$. Let's fix the values of $g_{i,j}$ to $x_{i,j} \in \{0, 1\}$ for $0 \le i < r_0$ and $0 \le j < c_0$. Let H denote the set of horizontally striped cells and V denote the set of vertically striped cells.

- The number of ways to fill the remaining cells in the leftmost c_0 columns is $\prod_{i=0}^{r_0-1} \binom{m_i}{n_i}^p$, where $m_i = \#\{j \mid (i, j) \in H\}$ and $n_i = \#\{j \mid (i, j) \in H, x_{i,j} = 1\}$.
- The number of ways to fill the remaining cells in the topmost r_0 rows is $\prod_{j=0}^{c_0-1} \binom{m'_j}{n'_j}^q$, where $m'_j = \#\{i \mid (i, j) \in V\}$ and $n'_j = \#\{i \mid (i, j) \in V, x_{i,j} = 1\}$.

By summing over all possible combinations of values for $x_{i,j}$, the number of ways to fill the topmost r_0 rows and the leftmost c_0 columns is

$$2^{r_0c_0-|H|-|V|} \cdot \left(\prod_{i=0}^{r_0-1} \sum_{k=0}^{m_i} \binom{m_i}{k}^{p+1} \right) \cdot \left(\prod_{j=0}^{c_0-1} \sum_{k'=0}^{m'_i} \binom{m'_i}{k'}^{q+1} \right).$$

Furthermore, if (i, j) is vertically striped and it is not constant, we refer to the pair as *strictly vertically striped*.

When we fix that each of the top-left $r_0 \times c_0$ cells is either strictly vertically striped or horizontally striped, the number of ways to fill the cells can be represented as $\prod_{i=0}^{r_0-1} f(m_i) \prod_{j=0}^{c_0-1} f'(m'_j)$ for some functions f and f', which can be obtained through a calculation of inclusion and exclusion. Here, with H denoting a set of horizontally striped cells, we define $m_i = \#\{j \mid (i, j) \in H\}$ and $m'_j = \#\{i \mid (i, j) \in H\}$. Note that the two conditions, "strictly vertically striped" and "horizontally striped," are mutually exclusive and collectively exhaustive.

From this observation, you can compute the solution of this problem in several ways. One is dynamic programming: There are $2^{r_0 c_0}$ ways to fix "strictly vertically striped" or "horizontally striped" for each cell. Performing a brute-force search is too slow, but you can implement DP computation with some pruning tricks to improve efficiency.

Another approach is using generating functions: suppose that column j corresponds to a variable z_j and that having m'_j horizontally striped cells in column j corresponds to $z_j^{m'_j}$. A polynomial for one row is represented as

$$G = \sum_{x_0, x_1, \ldots, x_{c_0-1} \in \{0,1\}} f(c_0 - x_0 - x_1 - \ldots x_{c_0-1}) z_0^{x_0} \ldots z_{c_0-1}^{x_{c_0-1}}.$$

Computing G^{r_0} using Fast Fourier Transform (FFT) can lead to computing the solution efficiently. The runtime is $O(N \log N)$, where $N = (r_0 + 1)^{c_0}$.

(Acknowledgement: This analysis is partially based on insightful feedback from Yui Hosaka.)

E Duplicates

Let's say a row or a column is *colorful* if the values within it are all distinct. Otherwise, we say it's *non-colorful*. Our objective in this problem is to minimize the number of modifications needed to make all rows and columns non-colorful.

Let m_r denote the number of colorful rows, and m_c denote the number of colorful columns. Since the number of colorful rows and colorful columns decreases by at most one with each modification, we require a minimum of $\max(m_r, m_c)$ modifications to make all rows and columns in the matrix non-colorful. We can show that $\max(m_r, m_c)$ is indeed adequate for achieving this objective as follows.

When both $m_r > 0$ and $m_c > 0$ hold, there exists an entry (i, j) such that row i and column j are colorful. By changing the value at (i, j) to an arbitrary integer other than A_{ij}, we simultaneously decrease the number of colorful rows and columns by one, respectively.

When $m_r = 0$ and $m_c > 0$, there exists a colorful column, let's say column j. In this case, there must be at least one value v such that modifying an entry $(1, j)$ to v makes column j non-colorful while keeping row 1 non-colorful.

A similar logic applies when $m_r > 0$ and $m_c = 0$.

Using this approach, we can accomplish the objective with $\max(m_r, m_c)$ modifications. The computational time is $O(n^3)$ with a sloppy implementation. This can be further improved to $O(n^2)$ time.

F Forming Groups

For a fixed value of k, we can compute the minimum ratio by initially assuming that student 1 is standing on the leftmost position. We then try to shift student 1 one position to the right each time, while keeping track of the sum of skill levels of each group, and maintaining a data structure to keep track of the minimum and maximum sum. Each shift only modifies the sum of skill levels of two groups. We can use a multiset data

structure to support each shift in $O(\log n)$ time. After each shift, we compute the ratio between the maximum and minimum sum.

Trying all factors of n as the value of k is too slow. An optimization is to observe that for a fixed arrangement of students, having k groups is better than having $2k$ groups. This is because by pairing $2k$ groups and combining each pair into k groups, the maximum sum is less than doubled, while the minimum sum is more than doubled, which reduces the ratio. A similar argument can be applied for any multiples of k as well.

Therefore, we can try only prime factors of n as a value of k. For $n \leq 10^6$, there are at most 7 prime factors. Therefore, this solution runs in $7 \times O(n \log n)$ time.

G Personality Test

The problem can be reformulated as follows:

> Suppose there exists a graph with n vertices, numbered from 1 to n. Initially, each pair of vertices is connected by an undirected edge with weight zero. We increment an integer b from 1 to n. In each step, for every $1 \leq i < b$ and $1 \leq j \leq m$ such that $S_{ij} = S_{bj} \neq$ '.', we increment the weight between vertices i and b by one. If we encounter an edge with a weight of k or more, we must report that edge and terminate the program.

A straightforward implementation of this procedure requires $\Theta(n^2 m)$ time. An important observation is that the sum of weights across all edges is bounded by $(k-1)\binom{n}{2}$, unless the step is at the termination of the program. This implies that the increase in edge weights occurs at most $O(kn^2)$ times. With some precomputation, for each j such that $S_{bj} \neq$ '.' in the steps of the above procedure, we can enumerate all indices i satisfying $S_{ij} = S_{bj}$ in $O(\#\{i < b \mid S_{ij} = S_{bj}\})$ time, instead of $\Theta(b)$ time.

With this enumeration method, the runtime becomes $O(nm + m\Sigma + kn^2)$, where Σ is the number of Latin characters (in this problem, $\Sigma = 26$).

H Pho Restaurant

Consider an easier version of this problem where we can add new tables. Clearly, it is optimal if, for the tables that are seated by different orders, we move the *minority*, i.e. the lesser orders between the two types of pho, to a new table.

One can assume that this strategy also works for the original problem. However, it only works if existing tables accommodate the minorities. This will not be an issue if we can have both types of pho to be a non-minority on some tables. But in case non-minorities are just one type (which also means there is just one type of minority), we need to choose a table that is dedicated to accommodating minorities instead. As this table will move the non-minority orders, it is not optimal to choose more than one dedicated table. Thus, a single careful iteration will suffice.

There is also an alternative solution that uses DP-Bitmask where we have four bits to represent: there is a

table where we need to move '0', there is a table where we need to move '1', there is a table dedicated to consist of only '0', and there is a table dedicated to consist of only '1'.

All solutions run in $O(n + \sum_{i=1}^{n} |S_i|)$.

I Symmetric Boundary

Let's denote an input point (x_i, y_i) by P_i. Also, let X represent the center of the point symmetry. According to the definition of point symmetry, a point Q_i satisfying $(P_i + Q_i)/2 = X \iff Q_i = 2X - P_i$ should also be included in the boundary of the convex region we are seeking. The problem is to find a point X such that the $2n$ points $P_1, \ldots, P_n, Q_1, \ldots, Q_n$ form a hull, meaning that the boundary of the convex hull of the $2n$ points contains all of them. We say that a point X satisfying this condition is *valid*.

Somewhat briefly, the solution is as follows: for two indices a and b that are between 1 and n, inclusive, let $L(a, b)$ be a line passing through $(P_a + P_b)/2$ and $(P_{a+1} + P_b)/2$. Here, we say $P_{n+1} = P_1$. Draw n^2 lines of $L(a, b)$ on the plane for $1 \leq a, b \leq n$. There are $O(n^4)$ intersection points between the lines. For each intersection point, verify whether the point is valid or not. If it is valid, calculate the area of the hull. This process requires $O(n \log n)$ time for each intersection point. The minimum area found by this procedure represents the solution. The runtime complexity is $O(n^5 \log n)$.

But why is the above solution correct? Before delving into the rationale, let's explain a few concepts.

- The point Q_i depends on the point X, so from now on, we write Q_i as $Q_i(X)$.
- For three points A, B and C, let $CCW(A, B, C)$ be true if and only if $(B - A) \times (C - A) \geq 0$.
- For a valid point X, let $s(X)$ denote the area of a convex hull formed by the points $P_1, \ldots, P_n$, $Q_1(X), \ldots$, $Q_n(X)$.

When we draw n^2 lines, the plane is divided into $O(n^4)$ connected regions. Let's select one region R. Here, R includes the boundary of the region. We denote the interior points of R by R°. For any two points $X \in R^\circ$ and $X' \in R$, and for any two indices a and b, both of the following hold.

- If $CCW(P_a, P_{a+1}, Q_b(X))$ is true, $CCW(P_a, P_{a+1}, Q_b(X'))$ is also true.
  (Hint: prove that $CCW(P_a, P_{a+1}, Q_b(X))$ is equivalent to $CCW((P_a + P_b)/2, (P_{a+1} + P_b)/2, X)$.)
- If $CCW(P_a, Q_b(X), Q_{b+1}(X))$ is true, $CCW(P_a, Q_b(X'), Q_{b+1}(X'))$ is also true.

This implies that if one point in R° is valid, then any point in R is valid. Also, the order of vertices on the boundary of a convex polygon formed by $P_1, \ldots, P_n, Q_1(X), \ldots, Q_n(X)$ is always the same for $X \in R$.

From this, we can deduce that the function $s : \mathbb{R}^2 \to \mathbb{R}$ is a linear function when the domain is restricted in R, because the area of a polygon is the sum of the cross product of neighboring vertices,

- $P_a \times P_{a+1}$ is constant,
- $P_a \times Q_b(X)$ and $Q_b(X) \times P_a$ vary linearly with respect to X, and

- $Q_b(X) \times Q_{b+1}(X) = (2X - P_b) \times (2X - P_{b+1}) = 2X \times (P_b - P_{b+1}) + P_b \times P_{b+1}$ varies linearly with respect to X.

Therefore, the minimum value of s within the domain of R is always achieved on the boundary of R. Specifically, since R is a convex polygon, the minimum value is attained at the vertices of the region. This justifies the fact that we only need to check the intersection points of n^2 lines.

J There and Back Again

We would like to compute the sum of the route that has the minimum total traversal time (the best route) and the route that has the second minimum total traversal time (the second best route). Finding the best route can be done with Dijkstra algorithm. We can also note the set of roads traversed in this route.

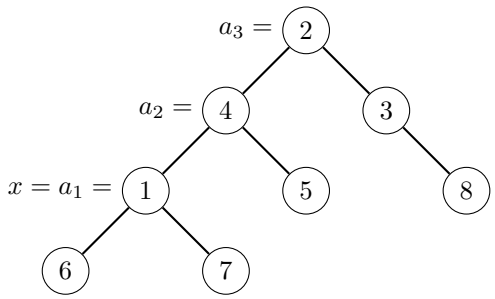
To find the second best route, we run Dijkstra algorithm twice to compute, for each city u, the shortest path to city 1 (denoted as S_u) and the shortest path to city n (denoted as T_u). We then iterate each road that is not traversed in either direction in the best route to try which edge is traversed in the second best route. For each such road (u, v), the minimum total traversal time for the second best route by traversing this edge is $S_u + w(u, v) + T_v$ where $w(u, v)$ is the time it takes to traverse the road. We then take the minimum value among all such roads.

This solution runs in $O(n \log n)$ time.

K Tree Quiz

The integers in L can be represented as tuples $(x, \mathrm{LCA}(x, y), y)$, where the tuples are ordered by x, ties are broken by $\mathrm{LCA}(x, y)$, and further ties are broken by y. Since each i has n tuples with i as its first element, the value of x for the k-th element of L is simply $\lceil \frac{k}{n} \rceil$. Next, we want to find the $((k-1) \bmod n + 1)$-th tuple with x as its first element. For simplicity, let $m = (k-1) \bmod n + 1$.

Let $S(i)$ be the set of nodes in the subtree of node i, and $A = \{a_1, a_2, \ldots\}$ be the ancestors of node x in order from x (i.e., $a_1 = x$ and node a_{i+1} is the parent of node a_i). Let $f(a_i) = |S(a_i) \setminus S(a_{i-1})|$ (i.e., the number of nodes in the subtree of node a_i but not in the subtree of node a_{i-1}), and $f(z) = 0$ for $z \notin A$. This means $f(z)$ is the number of tuples with x and z as the first and second elements respectively.



- $f(a_1) = f(1) = 3$ (nodes 1, 6, and 7)
- $f(a_2) = f(4) = 2$ (nodes 4 and 5)
- $f(a_3) = f(2) = 3$ (nodes 2, 3, and 8)
- $f(3) = f(5) = f(6) = f(7) = 0$

The m-th tuple that we are looking for has l as $\mathrm{LCA}(x, y)$ where $\sum_{i=1}^{l} f(i) \geq m$ and $\sum_{i=1}^{l-1} f(i) < m$. We can find this value using a segment tree that stores the values of $f(i)$. For each edge (u, v), by changing the value of x from u to v, only the values of $f(u)$ and $f(v)$ change. Therefore, we can keep the segment tree for each x using persistent segment tree in $O(n \log n)$ preprocessing time.

Let $m' = m - \sum_{i=1}^{l-1} f(i)$. We want to find the m'-th tuple with x and l as its first and second elements respectively. Let p is the index of A such that $a_p = l$. Thus, we want to find the m'-th node (in increasing order) in $S(a_p) \setminus S(a_{p-1})$.

We can do this by using another segment tree. Let G_i be a segment tree that stores $G_i(j) = 1$ if node i is an ancestor of node j, or $G_i(j) = 0$ otherwise. To compute G_i, we can take G_h as the base, where node h is the child of node i with maximum $|S(h)|$, and change the values of $G_i(j)$ for all nodes $j \in (S(i) \setminus S(j))$ individually. With another persistent segment tree, we can keep all segment trees G in $O(n \log^2 n)$ preprocessing time.

With G_{a_p} and $G_{a_{p-1}}$ which represents the subtree of node a_p and node a_{p-1} respectively, we can find the answer y that we are looking for. It is the unique integer that satisfies the following two inequalities:

$$\left(\sum_{i=1}^{y} G_{a_p}(i)\right) - \left(\sum_{i=1}^{y} G_{a_{p-1}}(i)\right) \geq m'$$

$$\left(\sum_{i=1}^{y-1} G_{a_p}(i)\right) - \left(\sum_{i=1}^{y-1} G_{a_{p-1}}(i)\right) < m'$$

This solution runs in $O(n \log^2(n) + q \log^2(n))$ time for preprocessing the persistent segment trees and answering all questions.

L XOR Operations

As a preprocessing step, let us replace each input integer a_i with $2a_i + 1$. This alteration does not affect the result; however, it proves to be useful for subsequent calculations.

A d-bit integer can be represented by a row vector in $(\mathbb{Z}/2\mathbb{Z})^d$. For an input a_i, let us denote the corresponding row vector in $(\mathbb{Z}/2\mathbb{Z})^d$ by $\vec{a_i}$. The XOR operation $a_i \oplus a_j$ corresponds to the addition $\vec{a_i} + \vec{a_j}$ in $(\mathbb{Z}/2\mathbb{Z})^d$.

Similarly, the sequence B can be represented as n row vectors in $(\mathbb{Z}/2\mathbb{Z})^d$. Now, the operation for indices i and j ($b_i \leftarrow b_i \oplus a_i \oplus a_j$ and $b_j \leftarrow b_j \oplus a_i \oplus a_j$) can be considered as an addition of $(\vec{0} \ldots, \vec{0}, \overset{i}{\vec{a_i} + \vec{a_j}}, \vec{0}, \ldots, \vec{0}, \overset{j}{\vec{a_i} + \vec{a_j}}, \vec{0}, \ldots, \vec{0})$ to B. Let us denote this row vector by $\vec{v_{i,j}}$. Note that the dimension of $\vec{v_{i,j}}$ is nd. There are $\binom{n}{2}$ choices for i and j. The solution to this problem is the number of elements in the space spanned by the vectors $\vec{v_{i,j}}$. Let M be a $\binom{n}{2} \times (nd)$ matrix such that each row corresponds to a vector $\vec{v_{i,j}}$. The solution is equal to $2^{\mathrm{rank}(M)}$. Thus, in summary, we only need to determine the rank of M.

To investigate the rank of M, for $k \geq 2$, let

$$D_k = \begin{bmatrix} \vec{a_1} + \vec{a_k} & & & \\ & \vec{a_2} + \vec{a_k} & & \\ & & \ddots & \\ & & & \vec{a_{k-1}} + \vec{a_k} \end{bmatrix}, F_k = \begin{bmatrix} \vec{a_1} + \vec{a_k} \\ \vec{a_2} + \vec{a_k} \\ \vdots \\ \vec{a_{k-1}} + \vec{a_k} \end{bmatrix},$$

and R_k be an $(k-1) \times (nd)$ matrix $[D_k \mid F_k \mid O]$. Also, let M_k be a matrix formed by stacking matrices $R_2, R_3, \ldots, R_k$ vertically. The matrix M as described above is equivalent to M_n. Without loss of generality, we can assume that s row vectors $\vec{a_1}, \ldots, \vec{a_s}$ are linearly independent, and other vectors can be expressed as the sum of a subset of $\vec{a_1}, \ldots, \vec{a_s}$.

Observation L.1. $\mathrm{rank}(M_s) = \binom{s}{2}$.

Proof. Suppose that $\sum_{(i,j)\in C} \vec{v_{i,j}} = 0$ for some set of indices $C \subseteq \{(i,j) \mid 1 \leq i < j \leq s\}$. Let us examine the first d elements of the vector $\sum_{(i,j)\in C} \vec{v_{i,j}}$. In particular, $\sum_{(1,j)\in C}(\vec{a_1} + \vec{a_j}) = \vec{0}$ must hold. Because $\vec{a_1}, \ldots, \vec{a_s}$ are linearly independent, C cannot contain any element of the form $(1,j)$ for $j = 2, \ldots, s$. With the same arguments applied to other elements, we can deduce that C must be empty. This means that the row vectors in M_s are linearly independent, and therefore $\mathrm{rank}(M_s) = \binom{s}{2}$. □

Observation L.2. *For $k > s$,* $\mathrm{rank}(M_k) = \mathrm{rank}(M_{k-1}) + s - 1$.

From Observations L.1 and L.2, we can conclude that the rank of M is $\binom{s}{2} + (n-s)(s-1)$. Since the computation of s, the number of linearly independent vectors, can be performed in $O(nd)$ time, the runtime is $O(nd)$. (In this problem, $d = 31$.) The remaining part is the outline of the proof for Observation L.2.

Proof. First, we can show that the rank of F_k is $s-1$. Because the rows of F_k are linear combinations of $\vec{a_1}, \ldots, \vec{a_s}$, $\mathrm{rank}(F_k) \leq s$. The rank can be further bounded by $s-1$ because any combination of rows in F_k always contains an even number of $\vec{a_1}, \ldots, \vec{a_s}$. (Here, we utilized the result of the preprocessing, that $\vec{a_{s+1}}, \ldots, \vec{a_k}$ are represented by an odd number of sums of the basis $\vec{a_1}, \ldots, \vec{a_s}$.) Also, without loss of generality, we can assume that $\vec{a_k} = \vec{a_1} + \ldots + \vec{a_t}$, where $t \geq 1$ is an odd integer. Then, the $s-1$ vectors $\vec{a_2} + \vec{a_k}, \vec{a_3} + \vec{a_k}, \ldots, \vec{a_s} + \vec{a_k}$ are linearly independent. From this, we can conclude that $\mathrm{rank}(F_k) = s - 1$.

Next, since the matrix M_k has the form of

$$M_k = \begin{bmatrix} M_{k-1} & O \\ D_k & F_k \end{bmatrix},$$

we can say that $\mathrm{rank}(M_k) \geq \mathrm{rank}(M_{k-1}) + s - 1$. To show that the inequality "$\geq$" is actually an equality, we need to establish that the elements of D_k do not affect the rank of M_k. Suppose that there exists a subset $C \subseteq \{1, \ldots, k-1\}$ such that $\sum_{i\in C}(\vec{a_i} + \vec{a_k}) = \vec{0}$. That is, C represents the indices of rows in F_k whose sum equals $\vec{0}$. For our purposes, it suffices to show that there always exists a subset $C' \subseteq \{(i,j) \mid 1 \leq i < j < k\}$ such that $\sum_{i\in C} \vec{v_{i,k}} = \sum_{(i,j)\in C'} \vec{v_{i,j}}$.

When $|C|$ is odd, because $\vec{a_k} = \sum_{i\in C} \vec{a_i}$, $C' = \{(i,j) \mid i,j \in C, i < j\}$ satisfies the condition. Meanwhile, when $|C|$ is even, because $\sum_{i\in C} \vec{a_i} = \vec{0}$, a set C' constructed from $\{1, \ldots, t\} \times C$ satisfies the condition. This concludes the statement of the observation. □

M Zig-zag

Here, we'll only describe how to count *even-length* partitions of n such that $a_1 < a_2 > a_3 < a_4 > a_5 < \ldots$. There are only a few other kinds of partitions and they can all be handled similarly. Also, let $N := 300\,000$.

Instead of counting partitions satisfying $a_1 < a_2 > a_3 < a_4 > a_5 < \ldots$, it's easier to count partitions satisfying $a_1 < a_2 \leq a_3 < a_4 \leq a_5 < \ldots$. Let's call even-length partitions satisfying the latter *almost increasing*. Once we have computed the latter, the former can be computed by inclusion-exclusion. Specifically, we're counting partitions satisfying $a_1 < a_2 \ ? \ a_3 < a_4 \ ? \ a_5 < \ldots$ where each "?" is either "no condition", or "$\leq$", and in the latter case, the whole term is weighted by -1 (because of inclusion-exclusion).

Formally, let the *weight* of an almost-increasing partition be $(-1)^k$ where k is the number of times "$\leq$" appears, and let $a(n)$ be the sum of the weights of all almost-increasing partitions of n. Then every partition satisfying "$a_1 < a_2 \ ? \ a_3 < a_4 \ ? \ a_5 < \ldots$" is just a conjunction of some number k of almost-increasing partitions. Thus, the number $c(n)$ of partitions of n satisfying $a_1 < a_2 > a_3 < a_4 > a_5 < \ldots$ is

$$c(n) = \sum_{k=0}^{\infty} \sum_{\substack{n_i \geq 0 \\ n_1 + \ldots n_k = n}} a(n_1)a(n_2)\ldots a(n_k),$$

where the inner sum is actually a k-fold sum. The inclusion-exclusion is handled by the weights. Now, notice that this is a convolution and can be stated nicely in terms of generating functions! Let $A(x) := \sum_{n \geq 0} a(n)x^n$ and $C(x) := \sum_{n \geq 0} c(n)x^n$. Then the above translates to

$$C(x) = \sum_{k=0}^{\infty} A(x)^k = \frac{1}{1 - A(x)}.$$

Thus, if we're able to compute $A(x)$, we're also able to compute $C(x)$ with a single generating function inverse in $O(N \log N)$.

The values $a(n)$ for $n \leq N$ can be computed with DP by noticing that for an almost-increasing partition a of n, $a_i \geq i/2$ for all i, and thus its length is $O(\sqrt{n})$. Thus, we can compute them with DP by finding a recurrence for $a(n, \ell)$, the sum of the weights of almost-increasing partitions with sum n and length ℓ. Because $n \leq N$ and $\ell \leq 2\sqrt{N}$, with an $O(1)$ transition this gives an $O(N^{1.5})$-time algorithm.