Contest tutorials in English version

skywalkert

The difficulty levels that jury has estimated before contest were:

- Very easy: B, E
- Easy: C, G, H
- Medium: A, F, I, K
- Hard: D, J, L

In terms of number of teams that ended up solving each problem, the numbers were:

Problem	A	B	C	D	E	F	G	H	I	J	K	L
Solved	7	457	2	10	906	3	193	44	2	3	1	0
Submissions	29	4513	32	163	3444	36	1720	234	7	88	10	25

We must confess this is a difficult contest, as you might notice the wide difference between supposition and reality. Anyway, we just wish you could obtain better experience and better awards in the future contests.

A. Always Online

Shortest judge solution: 2670 bytes.

By contradiction, we can prove that for a connected undirected graph, each edge belongs to at most one cycle if and only if there are at most two non-intersect paths between any two vertices. In addition, the maximum flow problem and the minimum cut problem are equivalent, so we only need to consider how to cut some edges with the lowest sum of capacity so that the source vertex s is not connected to the sink vertex t.

Let's take a look at the s-t cut problem. If there are at least two edges in one cycle that are cut, we could figure out by adjusting that at most two of them should be cut, since there are at most two non-intersect paths between s to t. In addition, if there are many cycles where some edges of each of them are cut, it can be known by adjusting and contradiction that only the edges of at most one cycle of them should be cut.

One observation is that if there is at least one edge in some cycle should be cut in the best solution of the minimum cut problem, there must be two edges in that cycle need to be cut, one of which is the edge

with the smallest capacity in that cycle. Hence, we can remove the edge with the smallest capacity in each cycle and add its capacity to the other edges in the cycle, and the maximum flow between any two vertices will not change in value.

The modified graph forms a tree. If we add edges to an empty graph in descending order of its capacity, we can determine $\text{flow}(s,t) = w$ $(s \in S, t \in T)$ for the two sets of vertices S and T that are firstly connected after joining the edge of capacity w. Just maintain these sets of vertices by the disjoint set union, count in each set the number of vertices whose bit in some fixed position are 1, and then it is easy to calculate the answer by enumerating each bit. Time complexity is $\mathcal{O}(m \log n)$.

Note that $\text{flow}(s,t)$ can be $2 \cdot 10^9$, which means the answer may exceed 2^{63}.

B. Beautiful Now

Shortest judge solution: 1584 bytes.

For each permutation of digits, the minimum number of swaps equals to n minus the number of disjoint cycles in the permutation. We can solve the problem by enumerating all the n-permutations. That is, for each permutation, we just check if it can be reached within k swaps (because swapping the same digit is permitted) and then update the answer. Time complexity is expected to be $\mathcal{O}(n!)$ while algorithms in time complexity $\mathcal{O}(n!n)$ may get a TLE verdict (but who knows?). In order to achieve the expected time complexity, we may need to maintain some information about chains and cycles efficiently during the process of searching and backtracking.

C. Call It What You Want

Shortest judge solution: 2019 bytes.

It is easy to prove that the constant term of $\Phi_n(x)$ is always equal to 1 except when $n = 1$. In addition, Taylor expansion shows that $\ln(1 + f(x))$ is a formal power series over the integers if $f(x)$ is a polynomial over the integers with zero constant term. Therefore, a simple Möbius inversion may conclude that

$$\ln(1 - x^n) = \ln\left(\prod_{d|n}\left((-1)^{[n=1]}\Phi_d(x)\right)\right) \Leftrightarrow \ln\left((-1)^{[n=1]}\Phi_n(x)\right) = \ln\left(\prod_{d|n}\left(1 - x^d\right)^{\mu(n/d)}\right).$$

It is not difficult to conclude the degree of $\Phi_d(x)$ is equal to $\varphi(d)$. However, it is more important to observe there are only two non-zero terms in each polynomial $(1-x^d)$. Just implement each convolution in modulo $x^{\varphi(d)+1}$ like dynamic programming of knapsack and the time complexity is $\mathcal{O}(\sum_{d|n} 2^{\omega(d)}\varphi(d)) = \mathcal{O}(2^6 n)$ since $2 \times 3 \times 5 \times 7 \times 11 \times 13 = 30030$, $30030 \times 17 = 510510 \geq n$.

This problem was inspired by improving convolution, however, we paid our efforts to reject most solutions using fast convolution approach and not reject some brute-force-like solutions.

P.S. Be careful with negative coefficients.

D. Daylight

Shortest judge solution: 4518 bytes.

Let the sets of vertices from which the distances to u and v do not exceed w as S_u and S_v respectively. Assuming the distance from u to v is k, the intersection of S_u and S_v is equal to the set of vertices from which the distance to the middle point of the path from u to v does not exceed $w - \frac{k}{2}$. If we regard each midpoint of an edge on the tree as a new vertex, the problem will reduce into some online queries of the number of vertices from which the distance to a fixed vertex does not exceed a fixed value.

We call a vertex centroid if the maximal size of the subtrees of its children is the minimum when it becomes the root of the tree. Let's divide and conquer the tree. Every time we select the centroid as the root, process all the paths passing through the centroid, and then deal with each component in recursion after deleting the centroid, such that there is only $\mathcal{O}(n)$ space in each recursive level required for counting the number of effective vertices from which the distances to the centroid of each component do not exceed some fixed value and the number of distinct levels does not exceed $\lceil \log_2(n+1) \rceil$.

In fact, it is not necessary to add these $(n-1)$ new vertices. It is only necessary to discuss which vertex is passed first from the midpoint of the edge to the centroid of the component when processing the query. Time complexity is $\mathcal{O}(n \log n)$.

E. Everything Has Changed

Shortest judge solution: 709 bytes. Shortest judge Java solution: 771 bytes.

Note that the cutting areas do not intersect and the different cutting areas do not affect each other. Just enumerate every cutting area intersecting the disk and then count the lengths of their circumferences covered by each other.

F. Fireflies

Shortest judge solution: 2222 bytes.

Dilworth's theorem shows the minimum number of chains that cover all the space is equal to the maximum number of locations that cannot be reached by each other. Sperner's theorem shows the maximum size of the subset of the power set of S, where none of the sets is contained in another, is equal to $\binom{|S|}{\lfloor |S|/2 \rfloor}$. A generalization of Sperner's theorem could show the answer to this problem is the number of locations satisfying the sum of each coordinate is equal to $M = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} (p_i + 1) \right\rfloor$.

Therefore, we need to count the number of solutions for $\sum_{i=1}^{n} x_i = M$ where $x_i \in \mathbb{Z}$, $1 \le x_i \le p_i$ $(i = 1, 2, \cdots, n)$. By using the inclusion-exclusion principle, we know the answer equals to $\sum_{I \subseteq J} (-1)^{|I|} \binom{M-1-\sum_{i \in I} p_i}{n-1}$ where $J = \{1, 2, \cdots, n\}$. One observation is that the binomial coefficient is a low-degree polynomial of $\sum_{i \in I} p_i$ so we can count the contribution of $\left(\sum_{i \in I} p_i\right)^e$ $(e = 0, 1, \cdots, n-1)$ individually.

By using meet-in-the-middle approach, we can split J as $X = \{1, 2, \cdots, \lfloor n/2 \rfloor\}$ and $Y = \{\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \cdots, n\}$, prepare some information about $\sum_{i \in X} p_i$ and $\sum_{i \in Y} p_i$ individually, and finally obtain the cohesive contribution through binomial theorem and two-pointer approach. The time complexity is $\mathcal{O}(2^{n/2} n^2)$.

G. Glad You Came

Shortest judge solution: 1139 bytes. Shortest judge Java solution: 1614 bytes.

If there are two operations covering the same interval, we can just keep the maximum one. For each operation (l, r, v), let d be $\lfloor \log_2(r - l + 1) \rfloor$, replace this operation by two operations $(l, l + 2^d - 1, v)$ and $(r - 2^d + 1, r, v)$. After that, the length of the interval that each operation covers is a power of 2, which means the lengths are only $\mathcal{O}(\log n)$ types. The remaining part is just to enumerate operations in length decreasing order and split each operation into two operations of equal length until the length is one. Time complexity is $\mathcal{O}(m + n \log n)$ and space complexity is $\mathcal{O}(n \log n)$.

H. Hills And Valleys

Shortest judge solution: 1664 bytes.

Enumerate $[x, y]$ as the range of values in the flipped interval $[l, r]$ that have contribution to the answer, and then find the longest subsequence of A of the form $0^*1^* \cdots x^* y^* (y-1)^* \cdots x^* y^* (y+1)^* \cdots 9^*$, where k^* represents any non-negative times of repetitions of integer k.

I. Innocence

Shortest judge solution: 2851 bytes.

Let's solve an easier version first. In this version, we are given $N, \{R_i\}, K$ and we have to choose N integers $x_1, x_2, \cdots, x_N$ such that $x_i \in [0, R_i]$ $(i = 1, 2, \cdots, N)$ and the bitwise exclusive-OR of them equals to K. Applying dynamic programming on binary digits, we can classify all the situations by the highest digit satisfying there is at least one integer x_i that is strictly less than R_i on the digit. How to count the ways might be the difficult part. In fact, we already know that the lower part of digits of some integer x_j can be any form, so we can leave it first and determine it by equation after other integers are determined.

Now turn back to the harder version. Let's apply the inclusion-exclusion principle and try all 2^N possible states (i.e. positive case of $x_i \in [0, R]$ and negative case of $x_i \in [0, L-1]$). We can speed up the process of dynamic programming by matrix exponentiation for each state. Furthermore, two states may have the same effection from K if they have the same parity of positive cases (and negative cases), which means we could reduce 2^N possible states into only 2 kinds and optimize our solution. If we prepare some information only concerning N, L, R and then apply dynamic programming for each K, the time complexity for each test case will be $\mathcal{O}((M^3 \log N + Q) \log R)$, where M is the side length of matrices.

Our standard solution sets M as 8 by the way.

J. Just So You Know

Shortest judge solution: 5085 bytes.

The process of guessing essentially constructs a decision tree, which is equivalent to make the Huffman encoding for all substrings B of A.

Let's take a look at the suffix tree of A, a dictionary tree consisting of all suffixes of A (with appending a non-existent character \$). It can be seen that the string formed by the path from each node to the root is distinct to each other, and the number of occurrences of the string in A is equal to the number of leaves in the subtree of that node. Since the tree has only n leaves, the tree has at most $(n-1)$ nodes with at least two children. It is not a bad idea to merge the node that has only one child with its child, and then the tree will remain only $(2n-1)$ nodes that is convenient to count how many substrings B appeared in A some fixed times. This process can achieve linear time complexity by constructing a suffix array and building a virtual tree (i.e. the compressed suffix tree), where building suffix arrays in linear time may take advantage of induced sorting.

Let's try to calculate the cost of building a Huffman tree of them in linear time, in which we need to enumerate all possible B in ascending order of the number of occurrences. Denote c_i as the number of substrings in A that appeared i times ($i = 1, 2, \cdots, n$), and we have $\sum_{i=1}^{n} ic_i = \frac{n(n+1)}{2}$. If the number of occurrences does not exceed n, we can enumerate the nodes appeared i times or the nodes appeared i times and $(i+1)$ times respectively, and then update the corresponding c_{2i} or c_{2i+1}. If the number of occurrences exceeds n, then we know the number of such nodes does not exceed $\frac{n}{2}$, so just using a queue to maintain such nodes, merging the two nodes on the head of the queue, appending new nodes on the tail of the queue can achieve linear complexity.

By the way, although the time complexity of our standard solution is $\mathcal{O}(n)$, we have tested and found that some programs in time complexity $\mathcal{O}(n \log n)$ could pass all the test cases.

K. Kaleidoscope

Shortest judge solution: 1248 bytes.

The plane expansion is just a kidding. We don't need it for solving.

This is a classic problem of Pólya enumeration theorem. By applying that, we can only focus on how many ways to set colors for orbits meeting the conditions about the number of colored faces. For each situation where every d faces have the same color, we could utilize dynamic programming to count $f(i, j)$ as the number of ways such that the first i colors have been selected to color j faces with meeting the conditions individually and we only care about states satisfying $d|j$. In fact, for the rotation of rhombic hexecontahedron, there are only 60 distinct equivalence classes which could be formed into 4 types of symmetries. The time complexity is $\mathcal{O}(4 \cdot 60^2 n)$.

By the way, in order to make the division in modulo p only once, you may calculate the other parts in modulo the divisor multiply p and be careful with 64-bit integer overflow.

L. Lost In The Echo

Shortest judge solution: 5116 bytes.

This problem is a bit complicated, so only a brief description of our solution will be given here. It should be reminded that it is not helpful to find it on the OEIS website since it has not been solved on that; it is also not helpful to find a paper concerning it because there are few papers to introduce it. However, if you are interested in this problem, you may write a paper to discuss.

Let's take a look at algebraic expressions that only contain plus and minus operators or only contain multiply and divide operators, such as $a-b+c$ and $a/b*c$. If we rewrite them as $0+a-b+c$ and $1*a/b*c$, we can see that such expressions involving n variables have (2^n-1) distinct kinds, because expressions like $0-a-b-c$ and $1/a/b/c$ are the only exceptions.

Let's take a further look at algebraic expressions that only contain plus, multiply and divide operators with brackets, such as $a+b/c$ and $a*(b+c)$. If we divide items into several operation priority levels such that each level concerns only the plus operator or only the multiply and divide operators, and every adjacent two levels have no operators with the same type, we will be able to count the number of distinct expressions. Denote $f(n)$ as the number of expressions involving n variables such that the last operation, if exists, is an addition, $g(n)$ as the number of expressions involving n variables such that the last operation, if exists, is a multiplication or a division. It is convenient for us to count the labeled objects by the exponential generating function, so let's define exponential generating functions $F(x)=\sum_{n\geq 1}\frac{f(n)x^n}{n!}$, $G(x)=\sum_{n\geq 1}\frac{g(n)x^n}{n!}$, and we have $F(x)=\sum_{k\geq 2}\frac{G^k(x)}{k!}$, $G(x)=\sum_{k\geq 2}\frac{(2^k-1)F^k(x)}{k!}$. In the first equation, we enumerate the unordered combination of k subexpressions and then add some plus operators among them. In the second equation, after we enumerate the unordered combination of k subexpressions, we have (2^k-1) ways to add some multiply and divide operators among them.

Define that $E_F(x)=\sum_{k\geq 2}\frac{F^k(x)}{n!}$, $E_{2F}(x)=\sum_{k\geq 2}\frac{(2F(x))^k}{k!}$, $E_G(x)=\sum_{k\geq 2}\frac{G^k(x)}{k!}$, and we know $F(x)=E_G(x)$, $G(x)=E_{2F}(x)-E_F(x)$. Considering the derivative of E_F with respect to x, we know $E'_F(x)=\sum_{k\geq 2}\frac{F'(x)F^{k-1}(x)}{(k-1)!}=F'(x)\sum_{k\geq 1}\frac{F^k(x)}{k!}=F'(x)\left(F(x)+E_F(x)\right)$, and thus for $n\geq 2$ we have $[x^n]E_F(x)=\frac{1}{n}\left(\sum_{i=1}^{n-1}i[x^i]F(x)\cdot[x^{n-i}]\left(F(x)+E_F(x)\right)\right)$, which can be calculated by divide and conquer with fast convolution in time complexity $\mathcal{O}(n\log^2 n)$. However, lots of typical problems give the function $G(x)$ and require to calculate the function $F(x)$ with the equation $F'(x)=G'(x)F(x)$, which are quite different with this problem. The unknown functions in this problem appear in both sides of some convolution, so we need to arrange the order of convolutions appropriately for calculation, which is left as a practice for the readers.

Let's take a final look at algebraic expressions that contain plus, minus, multiply and divide operators with brackets, such as $a-b/(c-d)$ and $a+b/(d-c)$. The influence of distribution property is of great significance, for example, these two expressions mentioned above are equivalent. In fact, except the expressions excluding any minus operator, we are always able to construct another expression with

the opposite sign for each expression, for example, $a + b + c/(d - e)$ corresponds to $c/(e - d) - a - b$. Therefore, we can find out the answer by counting the number of expressions ignoring its sign or excluding any minus operator respectively. If the sign is ignored but the minus operator is permitted, we have $F(x) = \sum_{k\geq 2} \frac{2^{k-1}G^k(x)}{k!}$, $G(x) = \sum_{k\geq 2} \frac{(2^k-1)G^k(x)}{k!}$.

比赛中文题解

skywalkert

命题人在赛前预估的难度等级是：

- 非常简单：B, E
- 简单：C, G, H
- 中等：A, F, I, K
- 困难：D, J, L

而正确通过每道题目的队伍数量如下：

题目	A	B	C	D	E	F	G	H	I	J	K	L
通过数量	7	457	2	10	906	3	193	44	2	3	1	0
提交数量	29	4513	32	163	3444	36	1720	234	7	88	10	25

可以看到预测结果与实际结果的差距还是很大的，不得不承认这场比赛有点难了。不论如何，我们祝愿大家能在未来的比赛中获得更好的经验和更好的奖项。

A. Always Online

Shortest judge solution: 2670 bytes.

借助反证法，我们可以证明，对于一个连通无向图，每条边在至多一个环中当且仅当任意两点间的不相交路径至多两条。另外，最大流和最小割是等价问题，我们只需要考虑割掉最少容量的边使得源点 s 与汇点 t 不连通即可。

考虑 s-t 割问题，如果一个环中割掉了至少两条边，则通过调整法可以只割掉至多两条边，因为 s 到 t 至多有两条不相交路径经过这个环。此外，如果多个环都被割掉了一些边，则通过反证法和调整法可以知道只割掉至多一个环的边。

一个观察是，如果在最小割中一个环被割掉了至少一条边，那么一定是割掉了两条边，且其中一条是容量最小的边。因此，我们可以将每个环中容量最小的边移除，给这个环中其他边加上它的容量，使得任意两点间的最大流量在数值上不发生变化。

经过改动的图形成了一棵树。我们只需要向空图中按容量降序加入每条边，便可以知道对于加入容量 w 的边后才连通的两个点集 S 和 T 有 $\text{flow}(s,t)=w$ $(s\in S,t\in T)$。利用并查集维护这些点集，同时统计每个点集里有多少个点的某个二进制位是 1 即可按位计算答案。时间复杂度 $\mathcal{O}(m\log n)$。

注意 $\text{flow}(s,t)$ 可以达到 $2\cdot 10^9$，这意味着答案可能超过 2^{63}。

B. Beautiful Now

Shortest judge solution: 1584 bytes.

考虑下标的置换，可以发现最少的交换次数等于 n 减去置换中循环的数量，所以可以枚举所有的 n 排列解决此题。对于每个置换，只需要检查它是否可以在 k 次交换内实现（因为可以交换相同的位置）并更新答案。期望时间复杂度是 $\mathcal{O}(n!)$，$\mathcal{O}(n!n)$ 的算法有可能会超时（也可能不超时）。为了实现期望复杂度可能需要在搜索和回溯过程中高效维护链和环的信息。

C. Call It What You Want

Shortest judge solution: 2019 bytes.

显然，除了 $n = 1$ 之外，其他情况下 $\Phi_n(x)$ 的常数项总是 1。此外，泰勒展开表明，如果 $f(x)$ 是一个常数项为零的整系数多项式，那么 $\ln(1 + f(x))$ 是一个整系数形式幂级数。因此，简单运用 Möbius 反演可知

$$\ln(1-x^n) = \ln\left(\prod_{d|n}\left((-1)^{[n=1]}\Phi_d(x)\right)\right) \Leftrightarrow \ln\left((-1)^{[n=1]}\Phi_n(x)\right) = \ln\left(\prod_{d|n}\left(1-x^d\right)^{\mu(n/d)}\right)$$

不难发现，$\Phi_d(x)$ 的最高项次数是 $\varphi(d)$。但更重要的是注意到每个多项式 $(1 - x^d)$ 只有两项非零。因此直接在模 $x^{\varphi(d)+1}$ 意义下计算类似背包动态规划形式的卷积即可做到时间复杂度 $\mathcal{O}(\sum_{d|n} 2^{\omega(d)}\varphi(d)) = \mathcal{O}(2^6 n)$，因为 $2 \times 3 \times 5 \times 7 \times 11 \times 13 = 30030$ 而 $30030 \times 17 = 510510 \geq n$。

这个题是思考卷积优化时想出来的，不过，我们尽可能卡掉了大部分用了加速卷积方法的做法，放过了一些类似暴力的做法。

P.S. 可能要小心负系数的处理。

D. Daylight

Shortest judge solution: 4518 bytes.

考虑分别到 u 和 v 距离不超过 w 的点集 S_u 和 S_v，令 u 到 v 的距离是 k，则 S_u 和 S_v 的交集是到 u 到 v 这条路径中点距离超过 $w - \frac{k}{2}$ 的点集。不妨把树上的每条边的中点视为一个新点，则问题转化为在线询问到一个点距离不超过定值的点数。

如果一个点作为树根时它最大的子树节点数达到最少，那么定义这个点是树的重心。尝试将树进行分治，每次选择树的重心作为树根，处理所有必经重心的路径，再递归处理删掉当前重心后的每个连通块，这样每一层至多需要 $\mathcal{O}(n)$ 的空间记录每个连通块中距离相应重心不超过定值的有效点数，而这样的层数不超过 $\lceil \log_2(n+1) \rceil$。

实际上，增加 $(n-1)$ 个新点是不必要的，只需要在处理询问时讨论从边的中点走到连通块的重心时先经过这条边的哪个端点即可。时间复杂度 $\mathcal{O}(n \log n)$。

E. Everything Has Changed

Shortest judge solution: 709 bytes. Shortest judge Java solution: 771 bytes.

注意切割区域两两不相交，则不同切割区域不会相互影响。枚举每个与圆盘相交的切割区域，计算互相被包含的圆周长度即可。

F. Fireflies

Shortest judge solution: 2222 bytes.

Dilworth 定理表明，最小链覆盖数等于最大反链数，也即选出互不可达位置的最多数目。Sperner 定理显示，若要选择 S 的幂集的一个子集使得其中没有一个集合包含在另一个集合中，则这种子集的大小最大为 $\binom{|S|}{\lfloor |S|/2 \rfloor}$。Sperner 定理的一种推广可以给出本问题的答案：选择满足所有坐标之和等于 $M = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} (p_i + 1) \right\rfloor$ 的位置即可达到最大的数量。

因此，我们需要计算 $\sum_{i=1}^{n} x_i = M$ 的解数，其中 $x_i \in \mathbb{Z}, 1 \le x_i \le p_i$ $(i = 1, 2, \cdots, n)$。考虑容斥原理，可知答案为 $\sum_{I \subseteq J} (-1)^{|I|} \binom{M-1-\sum_{i \in I} p_i}{n-1}$，这里 $J = \{1, 2, \cdots, n\}$。注意到这个组合数是关于 $\sum_{i \in I} p_i$ 的低次多项式，所以我们可以分别计算 $\left(\sum_{i \in I} p_i\right)^e$ $(e = 0, 1, \cdots, n-1)$ 的贡献。

考虑中途相遇法，我们可以将 J 成 $X = \{1, 2, \cdots, \lfloor n/2 \rfloor\}$ 和 $Y = \{\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \cdots, n\}$，然后分别预处理关于 $\sum_{i \in X} p_i$ 和 $\sum_{i \in Y} p_i$ 的信息，最后利用二项式定理和双指针的方法计算它们共同产生的贡献。这样做的时间复杂度是 $\mathcal{O}(2^{n/2} n^2)$。

G. Glad You Came

Shortest judge solution: 1139 bytes. Shortest judge Java solution: 1614 bytes.

如果有两个操作覆盖相同的区间，我们可以保留最大的那个。对于每个操作 (l, r, v)，令 d 等于 $\lfloor \log_2(r - l + 1) \rfloor$，我们可以用两个操作 $(l, l + 2^d - 1, v)$ 和 $(r - 2^d + 1, r, v)$ 替换此操作。这样做之后，每个操作所覆盖的区间长度均为 2 的幂，这意味着长度仅有 $\mathcal{O}(\log n)$ 种。剩下的只不过是，按长度递减的顺序枚举操作，将每个操作分成两个相等长度的操作，直到区间长度为一。这样做的时间复杂度为 $\mathcal{O}(m + n \log n)$，空间复杂度为 $\mathcal{O}(n \log n)$。

H. Hills And Valleys

Shortest judge solution: 1664 bytes.

枚举 $[x, y]$ 表示翻转区间 $[l, r]$ 中对答案产生贡献的数字所处值域，然后找出 A 最长的类似 $0^*1^* \cdots x^*y^*(y-1)^* \cdots x^*y^*(y+1)^* \cdots 9^*$ 的子序列，其中 k^* 表示任意非负整数个 k。

I. Innocence

Shortest judge solution: 2851 bytes.

考虑这个问题的简化版：给定 $N, \{R_i\}, K$，统计有多少种选择 N 个整数 $x_1, x_2, \cdots, x_N$ 的方案满足 $x_i \in [0, R_i]$ $(i = 1, 2, \cdots, N)$ 且它们的按位异或值等于 K。考虑数位上的动态规划，我们可以根据最高的出现严格小于的二进制位对所有情况进行分类，这里严格小于是指存在至少一个整数 x_i 在这一位上严格小于 R_i。比较麻烦的可能是统计方案数，不过我们已经知道某个整数 x_j 的更低位部分可以取任何数值，因此我们可以先不管它的低位，在确定其他数的低位后再通过方程确定它的低位。

回到原来的问题，我们可以利用容斥原理将其转化为 2^N 个简化版的问题，即用 $x_i \in [0, R]$ 的情况排除掉 $x_i \in [0, L-1]$ 的情况从而得到答案。对于每个子问题，我们可以用矩阵快速幂加速动态规划的过程，还可以发现 K 对这些问题的影响可以根据满足 $x_i \in [0, R]$ 的情况数量的奇偶性来分类，这意味着我们可以将 2^N 种可能的状态变成 2 类问题，从而优化做法。如果先预处理关于 N, L, R 的动态规划信息再去处理关于 K 的部分，那么每组测试数据的时间复杂度可以做到 $\mathcal{O}((M^3 \log N + Q) \log R)$，这里 M 是指矩阵的边长。顺带一提，标程的 M 是 8。

J. Just So You Know

Shortest judge solution: 5085 bytes.

猜测过程实质上构建了一棵决策树，也相当于对 A 的所有子串 B 进行哈夫曼编码。

考虑 A 的后缀树，即所有 A 的后缀（加上一个不存在的字符 \$）组成的字典树，可知树上每个节点到根的路径形成的字符串两两不同，而这样的字符串在 A 中出现的次数等于这个节点的子树里叶子的个数。由于这棵树只会有 n 个叶子，所以这棵树至多有 $(n-1)$ 次分叉。不妨将只有一个孩子的节点与它的孩子合并，这样整棵树将只剩下 $(2n-1)$ 个节点，将会很方便统计有多少子串 B 相应地在 A 中出现了多少次。这个过程可以通过构建后缀数组和构建虚树（即被压缩的后缀树）做到线性复杂度，其中线性时间构建后缀数组可以使用诱导排序。

现在尝试在线性时间内计算构建哈夫曼树的代价，我们需要按照出现次数升序枚举所有可能的 B。令 c_i 表示在 A 中出现了 i 次的子串数量 $(i = 1, 2, \cdots, n)$，则有 $\sum_{i=1}^{n} i c_i = \frac{n(n+1)}{2}$。如果出现次数不超过 n，我们可以枚举合并出现次数均为 i 的节点，或是出现次数分别为 i 和 $(i+1)$ 的节点，然后更新相应的 c_{2i} 或 c_{2i+1}。如果出现次数超过 n，则这样的节点不会超过 $\frac{n}{2}$ 个，使用一个队列维护这样的节点，每次只需要合并队头的两个节点，向队尾增加新节点，即可做到线性复杂度。

顺带一提，虽然标程的时间复杂度是 $\mathcal{O}(n)$，但是在赛前的测试中，一些时间复杂度 $\mathcal{O}(n \log n)$ 的程序也通过了全部的测试数据。

K. Kaleidoscope

Shortest judge solution: 1248 bytes.

平面展开图是逗你笑的，解决这道题并不需要它。

这是一道 Pólya 定理的经典问题。使用该定理后，我们只需要考虑有多少种方案能给每个轨道选择颜色使得颜色数量满足限制。对于每种情况，若每 d 个面都要选择相同的颜色，我们可以用动态规划计算用前 i 种颜色涂 j 个面并满足颜色数量限制的方案数 $f(i, j)$，并且只考虑 $d|j$ 的情况。事实上，菱形六面体的旋转只有 60 种不同的等价类，它们可以分为 4 类旋转群。时间复杂度可以做到 $\mathcal{O}(4 \cdot 60^2 n)$。

顺带一提，为了最后在模 p 意义下做一次除法，你可以让其他部分在模 p 乘以除数意义下计算，但要注意 64 位整型数可能溢出。

L. Lost In The Echo

Shortest judge solution: 5116 bytes.

这道题目较为复杂，这里将只给出简要做法。需要提醒的是，这不是一道 OEIS 数列题，因为 OEIS 上没有给出这道题的解法；这也不是一道论文题，因为没有论文介绍它，不过感兴趣的同学可以写一篇论文对其进行研究。

考虑只有加减运算符或只有乘除运算符的表达式，例如 $a-b+c$ 和 $a/b*c$。我们可以将其改写为 $0+a-b+c$ 和 $1*a/b*c$，因此可知涉及 n 个变量的这类表达式有 (2^n-1) 种，因为只有 $0-a-b-c$ 和 $1/a/b/c$ 这类表达式是无法得到的。

考虑只有加乘除运算符和括号的表达式，例如 $a+b/c$ 和 $a*(b+c)$。我们可以将其划分层次，每一层要么只有加运算符，要么只有乘除运算符，并且相邻两层没有同类运算符，这使得我们可以对其进行统计。考虑由最外层运算符划分的一系列子表达式，我们需要计算每个子表达式的方案数，而这是相似的问题。令 $f(n)$ 表示涉及 n 个变量且最外层为加运算符的表达式数量，$g(n)$ 表示涉及 n 个变量且最外层为乘除运算符的表达式数量。指数型生成函数可以便于我们对带标号的对象进行统计，故定义指数型生成函数 $F(x)=\sum_{n\geq 1}\frac{f(n)x^n}{n!}$, $G(x)=\sum_{n\geq 1}\frac{g(n)x^n}{n!}$，则有 $F(x)=\sum_{k\geq 2}\frac{G^k(x)}{k!}$, $G(x)=\sum_{k\geq 2}\frac{(2^k-1)F^k(x)}{k!}$。在第一个等式中，我们枚举 k 个子表达式的无序组合，在它们之间加上加运算符。在第二个等式中，我们枚举 k 个子表达式的无序组合后有 (2^k-1) 种方法加上乘除运算符。

不妨定义 $E_F(x)=\sum_{k\geq 2}\frac{F^k(x)}{n!}$, $E_{2F}(x)=\sum_{k\geq 2}\frac{(2F(x))^k}{k!}$, $E_G(x)=\sum_{k\geq 2}\frac{G^k(x)}{k!}$，那么有 $F(x)=E_G(x)$, $G(x)=E_{2F}(x)-E_F(x)$。通过关于 x 求导 E_F，我们知道 $E'_F(x)=\sum_{k\geq 2}\frac{F'(x)F^{k-1}(x)}{(k-1)!}=F'(x)\sum_{k\geq 1}\frac{F^k(x)}{k!}=F'(x)\left(F(x)+E_F(x)\right)$，那么对于 $n\geq 2$ 有 $[x^n]E_F(x)=\frac{1}{n}\left(\sum_{i=1}^{n-1}i[x^i]F(x)\cdot[x^{n-i}]\left(F(x)+E_F(x)\right)\right)$，这可以通过分治过程配合快速卷积在 $\mathcal{O}(n\log^2 n)$ 时间内计算得出。一些经典的问题给出 $G(x)$ 和 $F'(x)=G'(x)F(x)$ 并需要你计算 $F(x)$，这是不同于本题目的。本题目中待求解的信息同时出现在卷积式两边，需要合理安排分治卷积的过程才能计算，这个留给读者作为练习。

最后考虑有加减乘除运算符和括号的表达式，例如 $a-b/(c-d)$ 和 $a+b/(d-c)$。我们需要注意到分配律所带来的影响，比如上述两个例子是等价的。事实上，除了不包含减运算符的表达式之外，对于任意一个表达式，我们能构造一个与之符号相反的表达式，例如 $a+b+c/(d-e)$ 对应 $c/(e-d)-a-b$。如果我们统计忽略符号的表达式的数量，以及不包含减运算符的表达式的数量，那么就能算出答案。如果忽略符号但是允许使用减运算符，则有 $F(x)=\sum_{k\geq 2}\frac{2^{k-1}G^k(x)}{k!}$, $G(x)=\sum_{k\geq 2}\frac{(2^k-1)G^k(x)}{k!}$。