Eid al-Fitr Celebration Contest Editorial

Rezwan Arefin
Mamnoon Siam
Arman Ferdous

June 18, 2018

Contents

1 Editorials

1.1 A. Set Queries Revisited

Setter: Rezwan Arefin.
Tester: Arman Ferdous, Shubhashis Roy Dipta.

Problem Statement: We are given a set, we need to be able to insert/delete an element, and find sum of maximum Y elements.

■ **Solution of Subtask 1:** Here we have Y constant. Using this fact we can solve the problem easily. Let us maintain two sets A and B, in sorted order. A will maintain maximum Y elements, and B will store rest of the elements. Now we query as follows -

1. Add X: Two cases -
   - Insert X into set A.
   - While size of A is larger than Y, remove the smallest element from A, and insert it to B.
2. Remove X: Again two cases -
   - If X is currently in set B, just remove it from B.
   - Else remove X from set A, and transfer the largest element from set B to set A, if it exists.
3. Query - Whenever we insert an element to set A, we need to add the element to a global variable S, and when removeing an element from A subtract it from S. Whenever we have a query of type 3, we can just print the variable S. A corner case is, if this is the first query of this type, you need to remove elements from set A while there are more than Y elements in it.

All these operations can be done in `std::set<int>` of C++, or similar thing in other languages.
Time Complexity: $O(Q \log Q)$.

■ **Full Solution:** Actually getting the idea for full task is much easier than getting the idea for Subtask 1. There are many ways to solve this. Here is two possible approach -

1. **Using BBST:** We store subtree sum in each node of a BBST. When a query comes. We can have two cases, either we don't have more than Y elements in BBST so far, then just print the subtree sum of root node In the other case, we need to find $(|S| - Y)^{th}$ element in the BBST, here $|S|$ = size of current set. It can be done in $O(\log n)$ by using subtree sizes and traversing the tree once. Then we need to find sum of all numbers greater than that element in the set. You can use the subtree sums to find this.
   If you use a Treap, then you can just split the tree into two parts, one containing elements that are greater than the element we just found, and other the remaining. Then print the answer and merge them back.
   Time Complexity: $O(Q \log Q)$.
2. **Using Segment Tree:** Make a two segment trees A and B which can handle these things -
   - `update(i, x)`: Set i^{th} index to x.
   - `query(l, r)`: Find sum of numbers from index l to r.
   - `find(y)`: Find first index i such that sum of range $[i, 10^9]$ is less than or equal to y.

   Now our queries transform into these -

   (a) Add X: `A.update(X, X), B.update(X, 1);`
   (b) Remove X: `A.update(X, 0), B.update(X, 0);`
   (c) Query: Answer is `A.query(B.find(Y), 1e9);`

   Now the problem is, we can't have a segment tree build on 10^9 indexes, as it will take "too much" memory. There are two ways to overcome this. We can compress all X first to be used as indexes, then build segment tree on 10^5 indexes. Or we can use a sparse segment tree, which is build implicitly. That means we create nodes only when we access that. See code samples for implementation details. However, we can actualyl replace segment tree B with ordered_set. That makes the solution, but only for GNU G++ compiler.

Memory Complexity: $O(Q)$ or $O(Q \log X_{max})$.
Time Complexity: $O(Q \log Q)$ or $O(Q \log X_{max})$.

Code Samples: Solution of Subtask 1, Full solution using Treap, Full solution using Sparse Segment Tree.

1.2 B. HrSiam and Jetpack Joyride

Setter: Rezwan Arefin.
Tester: Jubayer Rahman Nirjhor

Problem Statement: How many ways you can reach from point $(0,0)$ to point $(2n,0)$, without going under x-axis, allowed moves are $(+1,+1),(-1,-1),(+2,0)$.

■ **Solution of Subtask 1:** At seems like the problem is related to catalan numbers as it is quite well known that number of paths from $(0,0)$ to $(2n,0)$, with moves $(+1,+1),(-1,-1)$ is n^{th} Catalan Number C_n. Which also solves the problem - Number of paths from $(0,0)$ to (n,n) using moves $(+1,0),(0,+1)$, and not crossing the line $x=y$.

In our problem, we have a $(+2,0)$, move too. If you think the ground as the $x=y$ line, and then rotate the grid 45° then our problem turns into counting number of paths from $(0,0)$ to (n,n) using moves $(+1,0),(0,+1),(+1,+1)$ and not crossing line $x=y$. Now, if you know the Catalan Number recurrence, -

$$C_0=1, C_n=\sum_{i=0}^{n-1} C_i C_{n-i-1}$$

and how to derive it, then you can easiliy arrive it this – If f_n is the number of ways path from $(0,0)$ to (n,n), using the mentioned moves, then –

$$f_0=1, f_n=f_{n-1}+\sum_{i=0}^{n-1} f_i f_{n-i-1}$$

This immidiately gives us an $O(n^2)$ solution. The answer to our main problem is $f_{n/2}$.
These numbers also has a name, they are called Schröder numbers. If you want a proof of the recurrence, here is it - Proof of Schröder numbers recurrence, MathStackExchange

■ **Full Solution:** Main idea is finding a Generating Function for the recurrence. Let us modify the recurrence a bit such that it holds for all non-negative integers, $f_0=0$, and $f_{-n}=0$ for all integer n. It can be done like this –

$$f_n=\begin{cases}0, & \text{if } n\le 0\\ f_{n-1}+\sum_{i=0}^{n-1} f_i f_{n-i-1}+[n=0], & \text{otherwise}\end{cases}$$

Now, let $F(z)$ be the generating function for the sequence $\langle f_n\rangle$, then $F(z)=\sum_{n=0}^{\infty} f_n z^n$. That means, f_n is the coefficient of z^n in $F(z)$.

Now, our task is to find a simpler form of the generating function. Let's start with the recurrence, multiply both side by z^n, and take sum over all n. We get –

$$\begin{aligned}
&f_n z^n = f_{n-1}z^n+\left(\sum_{i=0}^{n-1} f_i f_{n-i-1}\right)z^n+[n=0]z^n\\
&\Rightarrow \sum_{n=0}^{\infty} f_n z^n=\sum_{n=0}^{\infty} f_{n-1}z^n+\sum_{n=0}^{\infty}\left(\sum_{i=0}^{n-1} f_i f_{n-i-1}\right)z^n+\sum_{n=0}^{\infty}[n=0]z^n\\
&\Rightarrow \sum_{n=0}^{\infty} f_n z^n=z\sum_{n=0}^{\infty} f_{n-1}z^{n-1}+z\sum_{n=0}^{\infty}\left(\sum_{i=0}^{n-1} f_i f_{n-i-1}\right)z^{n-1}+1\\
&\Rightarrow F(z)=zF(z)+zF(z)^2+1\\
&\Rightarrow zF(z)^2+(z-1)F(z)+1=0\\
&\Rightarrow F(z)=\frac{-(z-1)-\sqrt{(z-1)^2-4z}}{2z}\\
&\Rightarrow F(z)=\frac{1-z-\sqrt{z^2-6z+1}}{2z}\\
&\Rightarrow F(z)=\frac{\left(1-z-\sqrt{z^2-6z+1}\right)\left(1-z+\sqrt{z^2-6z+1}\right)}{2z\left(1-z+\sqrt{z^2-6z+1}\right)}\\
&\Rightarrow F(z)=\frac{2}{1-z+\sqrt{1-6z+z^2}}
\end{aligned}$$

Notice that, the 3^{rd} line is valid because we defined f_n to be 0 for all non-positive numbers. Also we ignored the positive root for $F(z)$. Because if we took the positive root then $F(0)$ becomes ∞, but by definition $F(0)$ was supposed to be f_0 .

But still there is a problem: How to calculate Square Root of a power series, and inverse of a power series? Fortunately, there is "Newton's Iteration Method" for this.

Lets assume we want to calculate first n terms of the inverse of $P(z)$. That is a power series $H_n(z)$, which satisfies $H_n(z)P(z)^{-1} \equiv 1 \pmod{z^n}$. Trivially $H_0(z) =$ inverse of the constant term of $P(z)$.

Now, we can take the 1 in left side and square it to arrive at –

$$H_{2n}(z) = 2H_n(z) - R_n(z)^2 P(z) \pmod{z^{2n}}$$

By doubling n like this we can get enough terms. The multiplications can be done with Number Theoretic Transform, since the modulo in the problem statement is NTT Friendly.

Similarly we can find first n terms of square root of a power series $P(z)$ too. that is a power series $S_n(z)$ satisfying $S_n(z)^2 \equiv P(z) \pmod{x^n}$. Trivally $S_0(z) = \sqrt{p_0}$, in our problem it is 1. Now, we can take the $P(z)$ left side and square it to get –

$$S_{2n}(z) = \frac{1}{2}\left(S_n(z) + P(z)S_n(z)^{-1}\right) \pmod{z^{2n}}$$

Now we can solve our problem.

Time Complexity: Let $T(n)$ be the complexity needed to compute first n terms of the inverse and the square root. Then we have –

$$\begin{aligned} T(n) &= T(n/2) + O(n \log n) \\ &= O(n \log n) + O(n/2 \log n) + O(n/4 \log n) + \cdots + O(1) \\ \Rightarrow T(n) &= O(2n \log n) = O(n \log n) \end{aligned}$$

However, the constant factor seems to be very high. You need faster implementation of NTT and those Inverse and Square Root. The sample solution runs in $\approx 500ms$. Though we set the TL to 2 seconds.

Code Samples: Solution of Subtask 1, Full Solution.

Note: You can solve a very similar problem here - CF438E - The Child and Binary Tree. Though finding out the generating function is much harder than this problem. If you want to learn more about Generating Functions, you can start reading Generating Function Chapter from "Concrete Mathematics" by Ronald L. Graham, Donald E. Knuth and Oren Patashnik.

Can we still do better? It turns out that we still can! Our recursion has $O(n)$ transions, but it is possible to find a recursion with $O(1)$ transion. Check the problem - CodeChef - JADUGAR and it's editorial for a more general version of this problem. This approach differentiates the generating function and then collects coefficients to make the recursion simpler.

1.3 C. Turn 'em All On

Setter: Mamnoon Siam.
Tester: Rezwan Arefin.

Problem Statement: Given some numbers, we need to change atmost K of them to maximize AND sum.

Solution: If we want i^{th} bit of the AND to be 1, then we must have i^{th} bit on in all numbers. So, if $k \geq n$, what is optimal? Optimal is to change all numbers to $2^{30} - 1$, that is $(\underbrace{11\ldots11}_{30\ 1's})_2$ in binary.

Now what if $k < n$? Lets say you can change k numbers currently, and you want to turn on i^{th} bit of all numbers, of course those who already have that bit on doesn't need changing. So if you have less than or equal to k numbers that have i^{th} bit off, you can change them to $2^{30}-1$. Why $2^{30}-1$? Because it has chances to turn on many other bits too. As we are talking about "Changing numbers", not "Turning on bits" so it is optimal to change to $2^{30} - 1$ if we change any number.

But, how do we decide if we should turn a bit on or not? Lets say we have ability to turn on i^{th} bit in all numbers. But what if, maybe we could turn some more bits later if we didn't turn on this bit? But as i^{th} bit contributes 2^i to total answer, here is the thing –

$$2^i > 2^{i-1} + 2^{i-2} + \cdots + 2^1 + 2^0$$

That means, if you have chance to turn on i^{th} bit, and if you ignore it, then even if you turn all other bits after this, you won't get a big number. So if you have ability to turn on a higher bit you should turn that on and reduce k by number of numbers changed.

So our algorithm is simple, we iterate on bits from 30 down to 0. For each bit we check how many numbers have this bit off. If the count is not more than k, we make all those numbers $2^{30} - 1$ and reduce k.

Time Complexity: $O(n)$, or $O(n \times b)$ where b is number of bits we allow. In this problem it is 30.

Code Samples: Turn 'em All On Solution.

1.4 D. Stories S**k

Setter: Mamnoon Siam.
Tester: Rezwan Arefin, Shubhashis Roy Dipta.

Problem Statement: Given a graph with N nodes and M edges. Find a subset of edges such that in the graph formed with those N nodes, and the selected edges, each node has odd degree.

Solution: Before we start solving the problem, lets find our when a solution is possible or not. For that we need to prove a theorem first –
Theorem: In any graph $G(V, E)$, there are even number of nodes with odd degree.
Proof: In the graph, some node has even degree, and others have odd degree. But sum of degree of all nodes is $2E$, that is an even number. Now, sum of degrees of those even degree nodes is even. If the sum of degrees of odd degree nodes sum up to a even number, then count of those nodes must be even. As –

$$even \times something + odd \times odd \neq even$$

So, we can't have odd number of odd degree nodes in any graph.
But how does this help us? It helps because now we know, if we want every node in the final graph to have odd degree, then the number of nodes in the graph must be even. **So, there is no solution if n is odd.**
Now another question arises. Is it true that if n is even then there is always a solution? The answer is **yes**. Lets prove this too -
Proof: We will prove that for any graph with even number of nodes, there is a solution. The first step is to take any spanning tree of the graph. Actually a solution exists in any spanning tree of the graph. So we will basically prove that any tree with even number of nodes can have a solution.
We will prove this by induction on n.
Base case: For $n = 2$, it is true, as there is only one graph possible.
Inductive case: Lets assume it is true for all even $k \leq n$, we will show that it is true for $k + 2$ too.
Lets root the tree at a fixed node. Then take a node u such that all of its childs are leaf. Now we have two cases -

1. If u has odd number of childs, then the degree of u is -
   - Odd, if u is root. In that case we are done. Because now every u has odd degree, and all the leafs has off degree too.
   - Even, if u is not root. Because now u has a parent. So degree is (number of childs $+1$), that is even. In this case, we can remove the edge from parent of u to u. Now what can we say about the rest of the tree? We can say that the rest of the tree has even number of nodes. Because u + childs of u is even in count. Since $k + 2$ is even, and we are subtracting an even number, so the tree is reduced to another tree with even number of nodes. As we assumed $k \leq n$, k even has solution, so we are done.
2. If u has even number of childs, then also two cases -
   - If u is not root, then degree of u must be odd, if we keep the edge to its parent. Now lets consider the tree excluding the subtree of u, ans including u. That tree must have even number of nodes, and as we assumed, it has a solution. So in total, we again get a solution.
   - What if u is root? The answer is this case is not possible! If u was the root, and still has even number of childs, all of which is leaf, that will mean $k + 2$ is odd, which is not possible.

Now, the solution is simple. We do exactly as we did in the inductive step of the proof. Pseudo code is something like -

```
edge[] solution;
solve(u, parent) {
    for each v adjacent to u:
        add edge (u, v) to solution;
        solve(v, u);
    if u has odd number of childs, and u is not root:
        remove (parent, u) edge from solution;
}
// make a spanning tree of the input graph and call solve(1, 0)
```

Time Complexity: $O(n)$, or $O(n \log n)$ depending on implementation.
Code Samples: Stories S**k Solution.

1.5 E. Hash is Overrated!

Setter: Rezwan Arefin.

Problem Statement: Given some mappings from distinct integers to distinct integers, $x_i \rightarrow y_i$. Construct a sequence by taking numbers from xs, such that some elements occur odd number of times, but xor of corresponding ys becomes 0.

Solution: It can be easily shown that the problem basically requires us to find a subset from y's, such that they xor to 0. Then we need to print corresponding x's.

Now, problem is how to find a subset that xor to 0?

■ **Solution of Subtask 1:** For subtask 1 we can just generate all possible subset an check if their xor is 0. If we can find any then we can print their x's.

■ **Full Solution:** For full solution we will use Gaussian-Elimination. Lets construct a matrix by using y's binary representations. And reduce it to row-echelon form. As all of our elements are 1 or 0, you can see that the divisions in general Gaussian-Elimination has no effect at all. And the substitution of one row into another converts to xor-ing one row with another. So if we run Gaussian Elimination, then each row of the resulting matrix will represent xor of some rows.

Now, if rank of the matrix is $|S|$, then there is no solution. Else there will be some zero-rows in the bottom of the matrix. And if you know how a row was created, i.e. with which rows this was xor-ed then you know a solution. It is worth mentioning in the row-echelon form you can get atmost 64 non-zero rows. So for $|S| > 64$ there is **always** such a test.

So, we basically for each row we keep a set of numbers with whom a row was xor-ed. Then in the row echelon form, if we find a zero-row, we can print find the solution from those numbers.

For example: Lets say the numbers are $[3, 2, 1]$, we will construct matrix –

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

If we reduce it to row-echelon form, we'll get –

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$$

And we can see that xor-ing all numbers were needed to get the last row. So we can print all three numbers, they will xor to zero.
Time Complexity: $O(n^2)$ or $O(n^2 \log n)$ depending on implementation.
Memory Complexity: $O(n^2)$, as each row can be xor-ed with $O(n)$ other rows.
Code Samples: Solution of Subtask 1, Full Solution.

1.6 F. Tight Four-Subsets

Setter: Mamnoon Siam.
Tester: Rezwan Arefin.

Problem Statement: Count number of graphs with N vertices where this property holds – "If you take any four vertices, then at least one of them is connected to other three."

Solution: First lets see when a graph is so called "Tight". If we draw some more graphs with large N, we'll see that such graphs are always nearly complete (each pair of vertices has an edge between them). Precisely this property holds – "There are at least $N-3$ vertices with degree at least $N-1$." We won't prove it here, but it's obvious.

Now, about counting such graphs, the main trick is to count not the main graphs, but the **complement** graphs of main graphs. A complement graph of a graph is another graph where (u,v) edge exists iff (u,v) edge doesn't exist in main graph.

Now we can see, in our case this complement graph can have only 4 shapes -

1. 3 vertices are in a cycle, and all other vertices are not connected to any vertices. Count this type of graphs is simple, just take 3 vertices from N vertices and form a cycle, then we get this type of graph. If we complement it again we get main graphs. So number of ways is –
$$\binom{N}{3} = \frac{N(N-1)(N-2)}{6}$$
.
2. 3 vertices are in a chain, and all other vertices have degree 0. For this kind of graphs, we can take the first vertex for chain in N ways, second vertex in $N-1$ ways and third vertex in $N-2$ ways. But this way we double count pairs like $1-2-3$ and $3-2-1$. So divide by two. Total number of ways is –
$$\frac{N(N-1)(N-2)}{2}$$
3. 2 vertices are in a chain, other vertices has degree 0. This is easy, just take two vertices and connect them. Total number of ways is
$$\binom{N}{2} = \frac{N(N-1)}{2}$$
4. All vertices are of degree 0. There is only one graph of this type. If you take complement of it, you get the complete graph with N nodes.

So, we can say that total number of possible ways is –
$$\frac{N(N-1)(N-2)}{6} + \frac{N(N-1)(N-2)}{2} + \frac{N(N-1)}{2} + 1$$
$$= \frac{1}{6}\left(4N^3 - 9N^2 + 5N + 6\right)$$

■ **Solution of Subtask 1:** We can see that for $N \leq 10^5$ the sum is always less than our mod. So we can just find and print the sum.

■ **Full Solution:** For full solution, there is a problem. We can't just multiply two numbers when mod is 998989898989898989. Because that will overflow 64-bit integer type. But we can write something similar to BigMod for multiplying two numbers. Notice that if $f(a,b) = a \times b$ then this holds –
$$f(a,b) = \begin{cases} 2 \times f(a, \frac{b}{2}), & \text{if } b \text{ is even} \\ a + f(a, b-1), & \text{if } b \text{ is odd} \\ 0, & \text{if } b = 0 \end{cases}$$

So we can multiply a and b in $O(\log \min(a,b))$ complexity. And for that $\frac{1}{6}$, we can multiply by inverse of 6 mod P, or that is $6^{P-2} \mod P$, where $P = 998989898989898989$. It is 832491582491582491.
Time Complexity: $O(T \log N)$
Code Samples: Solution of Subtask 1, Full Solution.

1.7 G. Change in GCD

Setter: Arman Ferdous.
Tester: Rezwan Arefin.

Problem Statement: Given an array a and you need to solve queries – Count minimum number of elements to change such that $gcd(a_l, \dots a_r)$ becomes X, for given l, r and X.

Solution: The solution is quite simple. In the segment $a_l, \dots, a_r$, some number might be divisilbe by X, and some might not.

If there exist some numbers that are not divisible by X, then we can change all of them to X, or we can change one of them to X and change others to some multiple of X. Basically if there are some numbers that are not divisible by X, we **need** to change them all. And there is a way to change them all such that the gcd becomes X.

If all of those numbers are divisilbe by X, then there are two cases –

- If X exists in the segment, then gcd is already X. We don't need to change anything.
- If X doesn't exist in the segment, then we can take any number from the segment and make it X. So we need to change only 1 element in this case.

Now our problem is to calculate these two things fast –

1. Count how many numbers in the segment $a_l, \dots, a_r$ are divisible by X.
2. Check if X appreas in the segment $a_l, \dots, a_r$.

■ **Solution of Subtask 1:** We can calculate both of above in $O(N)$ time by just looping through the segment. Resulting in $O(QN)$ time complexity.

■ **Full Solution:** Calculating number of occurance in segment is easy. For each number of the array, create a list of positions where that numbers occurs in the array. Now if we want to check if X appears in $a_l, \dots, a_r$, we can just check if any numbers in X's list in in range $[l, r]$ – which can easily be done with Binary Search.

Now for the first one, counting how many numbers are divisible by X in segment – we can see that this is equilivant of calculating – (#-of numbers divisible by X in $a_1, \dots, a_r$) minus (#-of numbers divisible by X in $a_1, \dots, a_{l-1}$).

So, we can solve the problem offline. We will divide each query into two prefix queries – one for the prefix $[1, r]$, another for the prefix $[1, l-1]$. We should add the solution for prefix $[1, r]$, and subtract the solution for prefix $[1, l-1]$.

We will solve all queries in one sweep in the array. Lets keep an array cnt. When we are at index i, we will add 1 to cnt_d, where d is a divisor of a_i. Now we will process all queries on prefix $[1, i]$. And what's interesting is, now the #-of numbers divisible by X in the prefix $[1, i]$ is just cnt_X.

Pseudo Code can be something like this –

```
Q[N] = Array of prefix queries that we will solve offline.
    Queries in the form (X, query_id, direction).
    direction is --
        * +1 if it was [1, r] query
        * -1 if it was [1, l-1] query.

ans[] will store answer of queries.
cnt[] will store count of some numbers.

When you read i-th query (l, r, X), do this -
    Add query (X, id, +1) to Q[r]
    Add query (X, id, -1) to Q[l - 1]

for each index i:
    for each divisor d of a[i]:
        increment cnt[d];
    for each query q in Q[i]:
        ans[q.id] += q.direction * cnt[q.X];
```

We can precalculate list of divisors of all numbers before hand in by kind of sieve –

```
vector<int> f[M+1];
for(int i = 1; i <= M; i++) {
    for(int j = i; j <= M; j += i) {
        f[j].push_back(i);
    }
}
```

Here M is maximum number in input, in our case $M = 10^5$. Above two loops may seem like $O(M^2)$ in first sight. But it is actually –

$$\begin{aligned}&\frac{M}{1} + \frac{M}{2} + \frac{M}{3} + \ldots + \frac{M}{M}\\ &= M\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots \frac{1}{M}\right)\\ &= O(M \ln M) \approx O(M \log M) \quad [\because e \approx 2]\end{aligned}$$

See Euler–Mascheroni constant for more details.

A thing to notice that in range $[1, 100000]$, a number can have atmost 128 divisors (numbers 83160 and 98280 has 128 divisors). So the loop on divisors while solving queries runs atmost 128 times. Thus, our solution is still linear.

Let M be the maximum number in the input, then –
Time Complexity: $O(M \log M + 128N + Q) = O(M \log M + N + Q)$
Memory Complexity: $O(M \log M)$.
Code Samples: Solution of Subtask 1, Full Solution.
Bonus: Solve the problem online. It is possible to solve it online in $O((N + Q) \log N)$ complexity.

1.8 H. Suggestion for Exam

Setter: Rezwan Arefin.
Tester: Arman Ferdous.

Problem Statement: There are N chapters in a book, there will be Q questions in the exam, you need to write K of them. Find the probability of being able to solve K questions if you read first M chapters.

Solution: If $N = M$ then answer is 1 obviously. Also if $M = 0$ then answer is 0. Lets assume that $N \neq M$.

Let's count total number of possible question sets. Lets say $F(N, Q)$ is number of possible sets using N chapters and Q questions. And let's say there will be a_i questions from i^{th} chapter. Then this should hold –

$$a_1 + a_2 + \ldots + a_N = Q$$

Number of tuples $(a_1, a_2, \ldots, a_N)$ that satisfy above can be calculated from Stars and Bars Theorem. That is –

$$F(N, Q) = \binom{Q + N - 1}{N - 1}$$

Now, let's count in how many sets we have at least K questions common from first M chapters. At first we can forget about "at least" part. Let's calculate in how many sets we have exactly X questions common for some X.

In such set, there should be X questions from first M chapters and $Q - X$ questions from last $N - M$ chapters. So number of possible sets is $F(M, X) \cdot F(N - M, Q - X)$.

That was for exactly X questions. Now, if we want at least K questions, then we can do that for $X = K, K + 1, \ldots, Q$ and add them up. So total number of sets where there are at least K question from first M chapters is -

$$\sum_{X=K}^{Q} F(M, X) \cdot F(N - M, Q - X)$$

So our probability is –

$$\frac{\sum_{X=K}^{Q} \binom{X+M-1}{M-1} \cdot \binom{Q-X+N-M-1}{N-M-1}}{\binom{Q+N-1}{N-1}}$$

We can see that as N, M, Q, K is atmost 30, all of those calculations can be fit into 64 bit integers.

Using above approach we can get result as $\frac{X}{Y}$ format. For first two subtasks we can get X and Y by backtracking (generate all possible sets). Then we can do some processing to get result turncated to 7^{th} digit, that will solve Subtask 1 and Subtask 3. For getting full score we can find decimal representation of $\frac{X}{Y}$ digit by digit. That is also easy -

```
printf("%d.", X/Y); X %= Y;
while(D--) {
  X *= 10;
  printf("%d", X / Y);
  X %= Y;
} puts("");
```

Time Complexity: $O(T \times (Q + D))$
Code Samples: Suggestion for Exam.

2 Code Samples

2.1 A. Set Queries Revisited

2.1.1 Subtask 1 Solution

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

int main(int argc, char const *argv[]) {
  set<int> A, B; ll sum = 0;
  int q, y; scanf("%d", &q); y = q;
  while(q--) {
    int t, x; scanf("%d %d", &t, &x);
    if(t == 1) {
      A.insert(x); sum += x;
      while(A.size() > y) {
        sum -= *A.begin();
        B.insert(*A.begin());
        A.erase(*A.begin());
      }
    } else if(t == 2) {
      if(B.find(x) != B.end()) B.erase(x);
      else {
        A.erase(x); sum -= x;
        if(B.size()) {
          A.insert(*B.begin());
          sum += *B.begin();
          B.erase(B.begin());
        }
      }
    } else {
      y = x;
      while(A.size() > y) {
        sum -= *A.begin();
        B.insert(*A.begin());
        A.erase(A.begin());
      }
      printf("%lld\n", sum);
    }

  }
}
```

2.1.2 Full solution using Treap

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

struct node {
  int sz, prior, key; ll sum;
  node *l, *r;
  node(int v = 0) {
    sz = 1, prior = rand(), key = sum = v;
    l = r = NULL;
  }
} *root;
typedef node* pnode;
```

```
int sz(pnode t) { return t ? t -> sz : 0; }
void update(pnode t) {
  if(t) {
    t -> sz = sz(t -> l) + sz(t -> r) + 1;
    t -> sum = t -> key;
    if(t -> l) t -> sum += t -> l -> sum;
    if(t -> r) t -> sum += t -> r -> sum;
  }
}
void split(pnode t, pnode &l, pnode &r, int k) {
  if(!t) l = r = NULL;
  else if(t -> key <= k)
    split(t -> r, t -> r, r, k), l = t;
  else split(t -> l, l, t -> l, k), r = t;
  update(t);
}
void merge(pnode &t, pnode l, pnode r) {
  if(!l || !r) t = l ? l : r;
  else if(l -> prior > r -> prior)
    merge(l -> r, l -> r, r), t = l;
  else merge(r -> l, l, r -> l), t = r;
  update(t);
}
void insert(pnode &t, pnode it) {
  if(!t) t = it;
  else if(t -> prior < it -> prior)
    split(t, it -> l, it -> r, it -> key), t = it;
  else insert(it -> key <= t -> key ? t -> l : t -> r, it);
  update(t);
}
void erase(pnode &t, int key) {
  if(!t) return;
  else if(t -> key == key) {
    pnode tmp = t;
    merge(t, t -> l, t -> r);
    free(tmp);
  } else erase(key < t -> key ? t -> l : t -> r, key);
  update(t);
}
int kth(pnode t, int k) {
  int cnt = sz(t -> l);
  if(cnt == k - 1) return t -> key;
  if(cnt >= k) return kth(t -> l, k);
  else return kth(t -> r, k - cnt - 1);
}

int main(int argc, char const *argv[]) {
  int q, sz = 0;
  scanf("%d", &q);
  while(q--) {
    int t, x;
    scanf("%d %d", &t, &x);
    if(t == 1) ++sz, insert(root, new node(x));
    else if(t == 2) --sz, erase(root, x);
    else {
      if(sz <= x) {
        printf("%lld\n", root -> sum);
      } else {
        int last = kth(root, sz - x);
        pnode l, r;
        split(root, l, r, last);
        printf("%lld\n", r -> sum);
        merge(root, l, r);
      }
    }
  }
}
```

2.1.3 Full solution using Sparse Segment Tree

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int M = 1e9 + 5;
struct segtree {
  struct node {
    ll sum; int cnt;
    node *l, *r;
    node() {
      l = r = NULL;
      sum = cnt = 0;
    }
    void merge() {
      sum = cnt = 0;
      if(l) sum += l -> sum, cnt += l -> cnt;
      if(r) sum += r -> sum, cnt += r -> cnt;
    }
  } *root;
  segtree() { root = new node(); }

  void update(node *t, int l, int r, int i, bool add) { // if add = 1, then add, else remove
    if(l == r) {
      if(add) t -> sum = i, t -> cnt = 1;
      else t -> sum = t -> cnt = 0;
      return;
    } int m = l + r >> 1;
    if(i <= m) { // update in left subtree
      if(!t -> l) t -> l = new node();
      update(t -> l, l, m, i, add);
    } else { // update in right subtree
      if(!t -> r) t -> r = new node();
      update(t -> r, m + 1, r, i, add);
    } t -> merge();
  }
  ll query(node* t, int l, int r, int i, int j) { // return sum of range a[i..j], [l, r] tree node
      range.
    if(r < i || l > j) return 0;
    if(i <= l && r <= j) return t -> sum;
    int m = l + r >> 1;
    ll ret = 0;
    if(t -> l) ret += query(t -> l, l, m, i, j);
    if(t -> r) ret += query(t -> r, m + 1, r, i, j);
    return ret;
  }
  int find(node* t, int l, int r, int y) { // returns first index with suffix sum <= y;
    if(l == r) return l;
    int m = l + r >> 1;
    if(!t -> r) t -> r = new node();
    if(t -> r -> cnt >= y) { // then answer exists in right side
      return find(t -> r, m + 1, r, y);
    } else {
      if(!t -> l) t -> l = new node();
      int c = t -> r -> cnt;
      return find(t -> l, l, m, y - c); // Ignoring right side, so y - cnt of right side.
    }
  }
  void update(int x, bool add) { update(root, 0, M, x, add); }
  ll query(int y) {
    int k = find(root, 0, M, y);
    return query(root, 0, M, k, M);
  }
} tree;
int main(int argc, char const *argv[]) {
  int q, sz = 0;
  scanf("%d", &q);
  while(q--) {
    int t, x;
    scanf("%d %d", &t, &x);
    if(t == 1) tree.update(x, 1);
    else if(t == 2) tree.update(x, 0);
    else printf("%lld\n", tree.query(x));
  }
}
```

2.2 B. HrSiam and Jetpack Joyride

2.2.1 Solution of Subtask 1

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int mod = 7 * 17 * (1 << 23) + 1;
int f[1010];

int main(int argc, char const *argv[]) {
  f[0] = 1;
  for(int i = 1; i <= 1000; i++) {
    f[i] = f[i - 1];
    for(int j = 0; j < i; j++) {
      f[i] += (ll) f[j] * f[i - j - 1] % mod;
      if(f[i] >= mod) f[i] -= mod;
    }
  }
  int t; scanf("%d", &t); while(t--) {
    int n; scanf("%d", &n);
    printf("%d\n", f[n >> 1]);
  }
}
```

2.2.2 Full Solution

```
#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int N = 1 << 18, mod = 7 * 17 * (1 << 23) + 1, g = 3;
int rev[N], w[N|1], inv_n;

inline int Pow(int a, int p) {
  int ret = 1; while(p) {
    if(p & 1) ret = (ll) ret * a % mod;
    a = (ll) a * a % mod;
    p >>= 1;
  } return ret;
}

void prepare(int &n) {
  int sz = 31 - __builtin_clz(n); sz = abs(sz);
  int r = Pow(g, (mod - 1) / n);
  inv_n = Pow(n, mod - 2);
  w[0] = w[n] = 1;
  for(int i = 1; i < n; ++i) w[i] = (ll)w[i - 1] * r % mod;
  for(int i = 1; i < n; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << sz - 1);
}

void ntt(int *a, int n, int dir) {
  for(int i = 1; i < n - 1; ++i)
    if(i < rev[i]) swap(a[i], a[rev[i]]);
  for(int m = 2; m <= n; m <<= 1) {
    for(int i = 0; i < n; i += m) {
      for(int j = 0; j < (m >> 1); ++j) {
        int &u = a[i + j], &v = a[i + j + (m >> 1)];
        int t = (ll)v * w[dir ? n - n / m * j : n / m * j] % mod;
        v = u - t < 0 ? u - t + mod : u - t;
        u = u + t >= mod ? u + t - mod : u + t;
      }
    }
  } if(dir) for(int i = 0; i < n; ++i) a[i] = (ll)a[i] * inv_n % mod;
}

int ta[N], tb[N];
void polyinv(int *a, int *b, int n) { // b(z) * a(z) = 1 (mod z^n)
  if(n == 1) return void(b[0] = Pow(a[0], mod - 2));
  polyinv(a, b, n >> 1);
  for(int i = 0; i < n; ++i)
    ta[i] = a[i], tb[i] = b[i];
  for(int i = n; i < (n << 1); ++i)
    ta[i] = tb[i] = 0;
  n <<= 1; prepare(n);
  ntt(ta, n, 0), ntt(tb, n, 0);
  for(int i = 0; i < n; ++i)
    b[i] = (ll) tb[i] * (2 + mod - (ll) ta[i] * tb[i] % mod) % mod;
  ntt(b, n, 1);
  fill(b + (n >> 1), b + n, 0);
}

int inv2 = Pow(2, mod - 2);
void polysqrt(int *a, int *b, int n) { // b(z) * b(z) = a(z) (mod z^n)
  if(n == 1) return void(b[0] = 1); // b[0] = x: x^2 \equiv a[0]
  polysqrt(a, b, n >> 1);
  polyinv(b, tb, n);
  for(int i = 0; i < n; ++i)
    ta[i] = a[i];
  for(int i = n; i < (n << 1); ++i)
    ta[i] = tb[i] = 0;
  n <<= 1; prepare(n);
  ntt(ta, n, 0); ntt(tb, n, 0);
  for(int i = 0; i < n; ++i)
    ta[i] = (ll) ta[i] * tb[i] % mod;
  ntt(ta, n, 1);
  for(int i = 0; i < n; ++i)
    b[i] = (ll) inv2 * (ta[i] + b[i]) % mod;
  fill(b + (n >> 1), b + n, 0);
}

int a[N], b[N], c[N];

int main(int argc, char const *argv[]) {
  a[0] = 1, a[1] = -6, a[2] = 1;
  polysqrt(a, b, N >> 1);

  b[0]++; if(b[0] >= mod) b[0] -= mod;
  b[1]--; if(b[1] < 0) b[1] += mod;

  polyinv(b, c, N >> 1);

  int t; scanf("%d", &t); while(t--) {
    int n; scanf("%d", &n);
    if(n & 1) puts("0");
    else {
      int ans = ((ll) c[n >> 1] << 1) % mod;
      printf("%d\n", ans);
    }
  }
}
```

2.3 C. Turn 'em All On

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int N = 1e5 + 10;
int a[N], n, k;

int main(int argc, char const *argv[]) {
  scanf("%d %d", &n, &k);
  for(int i = 0; i < n; ++i)
    scanf("%d", &a[i]);
  for(int i = 30; i >= 0; --i) {
    int cnt = 0;
    for(int j = 0; j < n; ++j)
      if(~a[j] >> i & 1) ++cnt; // If ith bit of a[j] is off, ++cnt;
    if(cnt <= k) { k -= cnt;
      for(int j = 0; j < n; ++j)
        if(~a[j] >> i & 1)
          a[j] = (1 << 30) - 1;
    }
  }
  int ans = a[0];
  for(int i = 1; i < n; i++) ans &= a[i];
  printf("%d\n", ans);
}
```

2.4 D. Stories S**k

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int maxn = 2e5 + 10;
set<ii> adj[maxn];
set<int> st;
int vis[maxn];

void dfs(int u, int par, int pid) {
  vis[u] = 1; int c = 0;
  vector<ii> lst(adj[u].begin(), adj[u] .end());
  for(ii x : lst) {
    int v = x.first, id = x.second;
    if(vis[v] || v == par) {
      adj[u].erase({v, id});
      continue;
    } ++c;
    st.insert(id);
    dfs(v, u, id);
  } if(!c) return;
  if((adj[u].size() & 1) && par != -1) {
    st.erase(pid);
    auto it = adj[par].find({u, pid});
    if(it != adj[par].end())
      adj[par].erase(it);
  }
}

int main(int argc, char const *argv[]) {
  int n, m;
  scanf("%d %d", &n, &m);

  if(n & 1) {
    puts("-1\n-1");
    return 0;
  }

  for(int i = 1; i <= m; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    adj[u].insert({v, i});
    adj[v].insert({u, i});
  }

  dfs(1, -1, -1);

  cout << st.size() << endl;
  for(int x : st) {
    cout << x << " ";
  }
}
```

2.5 E. Hash is Overrated!

2.5.1 Solution of Subtask 1

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int N = 50;
int n, x[N]; ll y[N];

int main(int argc, char const *argv[]) {
  scanf("%d", &n);
  for(int i = 0; i < n; i++) {
    scanf("%d %lld", &x[i], &y[i]);
  }
  for(int mask = 1; mask < (1ll << n); mask++) {
    int tot = 0;
    for(int i = 0; i < n; i++)
      if(mask >> i & 1) tot ^= y[i];
    if(tot == 0) {
      puts("YES");
      printf("%d\n", __builtin_popcount(mask));
      for(int i = 0; i < n; i++)
        if(mask >> i & 1) cout << x[i] << " ";
      puts("");
      return 0;
    }
  }
  puts("NO");
}
```

2.5.2 Full Solution

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int N = 1e5 + 10;
ll y[N]; int n, x[N];
set<int> mem[N];

int main(int argc, char const *argv[]) {
  int n; scanf("%d", &n);
  for(int i = 0; i < n; i++) {
    scanf("%d %lld", &x[i], &y[i]);
    mem[i].insert(x[i]);
  }
  int r = 0;
  for(int c = 62; c >= 0; c--) {
    int idx = -1;
    for(int i = r; i < n && idx < 0; i++)
      if(y[i] >> c & 1) idx = i;
    if(idx == -1) continue;

    swap(x[r], x[idx]);
    swap(y[r], y[idx]);
    swap(mem[r], mem[idx]);

    for(int i = 0; i < n; i++) if(i != r) {
      if(y[i] >> c & 1) {
        y[i] ^= y[r];
        set<int> &a = mem[i], &b = mem[r];
        for(int it : b) {
          if(a.find(it) != a.end())
            a.erase(it);
          else a.insert(it);
        }
      }
    } r++;
  }
  if(r == n) puts("NO");
  else {
    bool flag = 0;
    for(int i = r; i < n; i++) if(mem[i].size() <= 1000) {
      puts("YES");
      printf("%d\n", (int)mem[i].size());
      for(int x : mem[i]) printf("%d ", x);
      return 0;
    } puts("NO");
  }
}
```

2.6 F. Tight Four-Subsets

2.6.1 Solution of Subtask 1

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

int main () {
  int t; scanf("%d", &t); while(t--) {
    ll n; cin >> n;
    ll ret = (4 * n * n * n - 9 * n * n + 5 * n + 6) / 6;
    cout << ret << endl;
  }
}
```

2.6.2 Full Solution

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const ll mod = 998989898989898989LL;
const ll modinv6 = 832491582491582491LL;

ll mul(ll a, ll b) {
  if(b == 0) return 0;
  if(b & 1) return (mul(a, b - 1) + a) % mod;
  return (mul(a, b >> 1) << 1) % mod;
}

int main () {
  int t; scanf("%d", &t); while(t--) {
    ll n; cin >> n;
    ll ret = (mul(4, mul(n, mul(n, n))) + mul(5, n) + 6) % mod;
    ret -= mul(9, mul(n, n));
    if(ret < 0) ret += mod;
    ret = mul(ret, modinv6);
    cout << ret << endl;
  }
}
```

2.7 G. Change in GCD

2.7.1 Solution of Subtask 1

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int N = 1e5 + 10;

int a[N], n, q;

int main(int argc, char const *argv[]) {
  scanf("%d %d", &n, &q);
  for(int i = 1; i <= n; i++) {
    scanf("%d", &a[i]);
  }
  for(int i = 1; i <= q; i++) {
    int l, r, x;
    int cnt = 0, has = 0;
    scanf("%d %d %d", &l, &r, &x);
    for(int j = l; j <= r; j++) {
      if(a[j] % x == 0) cnt++;
      if(a[j] == x) has = 1;
    }
    if(cnt == r - l + 1) {
      if(has) puts("0");
      else puts("1");
    } else printf("%d\n", r - l + 1 - cnt);
  }
}
```

2.7.2 Full Solution

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int N = 1e5 + 10;

int a[N], n, q, ans[N], cnt[N];
int l[N], r[N], X[N];
vector<int> f[N];

struct query { int X, dir, id; };
vector<query> Q[N];
vector<int> pos[N];

int main(int argc, char const *argv[]) {
  for(int i = 1; i < N; i++)
    for(int j = i; j < N; j += i)
      f[j].push_back(i);

  scanf("%d %d", &n, &q);
  for(int i = 1; i <= n; i++) {
    scanf("%d", &a[i]);
    pos[a[i]].push_back(i);
  }

  for(int i = 1; i <= q; i++) {
    scanf("%d %d %d", &l[i], &r[i], &X[i]);
    Q[r[i]].push_back({X[i], +1, i});
    Q[l[i] - 1].push_back({X[i], -1, i});
  }

  for(int i = 1; i <= n; i++) {
    for(int d : f[a[i]]) cnt[d]++;
    for(query q : Q[i]) {
      ans[q.id] += q.dir * cnt[q.X];
    }
  }

  for(int i = 1; i <= q; i++) {
    if(ans[i] == r[i] - l[i] + 1) {
      vector<int> &v = pos[X[i]];
      int tot = upper_bound(v.begin(), v.end(), r[i]) -
          lower_bound(v.begin(), v.end(), l[i]);
      if(tot) puts("0");
      else puts("1");
    } else {
      printf("%d\n", r[i] - l[i] + 1 - ans[i]);
    }
  }
}
```

2.8 H. Suggestion for Exam

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<int, int> ii;

const int N = 70;
ll C[N][N];

int main(int argc, char const *argv[]) {
  C[0][0] = 1;
  for(int i = 1; i < N; i++) {
    C[i][0] = 1;
    for(int j = 1; j <= i; j++) {
      C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
    }
  }

  int t; scanf("%d", &t); while(t--) {
    int q, n, k, m, D;
    scanf("%d %d %d %d %d", &n, &q, &k, &m, &D);

    ll X, Y;

    if(n == m) X = 1, Y = 1;
    else if(m == 0) X = 0, Y = 1;
    else {
      X = 0, Y = C[q + n - 1][n - 1];
      for(int i = k; i <= q; i++) {
        X += C[i + m - 1][m - 1] * C[q - i + n - m - 1][n - m - 1];
      }
    }

    printf("%d.", X/Y); X %= Y;
    while(D--) {
      X *= 10;
      printf("%d", X / Y);
      X %= Y;
    } puts("");
  }
}
```