MIST Inter University Programming Contest 2019
MIST-IUPC'19

Hosted by MIST
Dhaka, Bangladesh

23rd February 2019
You get 14 Pages
9 Problems &
300 Minutes

Platform Support:

Problem Set By:

Contest Rules for MIST IUPC 2019

1. Solutions to the problems submitted for judging are called runs. Each run is judged as accepted or rejected by the judge, and the team is notified of the result. Only source code should be submitted, not the executables or any other files.
2. **Contest will be held on CodeMarshal online judge. Only this website will be accessible from contestant's pc. Any attempt to access other websites or the Internet will result in disqualification. Any attempt to tamper with the online judge will also result in disqualification.**
3. A contestant may submit a clarification request to the judges only through **CodeMarshal's** clarification system. If the judges agree that an ambiguity or error exists, a clarification will be issued to all contestants. Judges may decide not to answer a clarification at all in which case that particular clarification request will be marked as **IGNORED** in the CodeMarshal clarification page.
4. If teams believe that there is something wrong with the judge data they are strongly advised to use the CodeMarshal clarification system to communicate with the judges rather than meeting them in person after the contest.
5. Contestants are not to converse with anyone except members of their own team and personnel designated by the organizing committee while seated at the team desk. They may not even talk with their team members when they are walking around the contest floor to have food or any other purposes.
6. While the contest is scheduled for a particular time length **(five hours)**, the contest director has the authority to alter the length of the contest in the event of any unforeseen difficulties. Should the contest duration be altered, every attempt will be made to notify contestants in a timely and uniform manner.
7. A team may be disqualified for any activity that jeopardizes the contest such as dislodging extension cords, unauthorized modification of contest materials, distracting behavior or communicating with other teams. The judges can also recommend penalizing a team with additional penalty minutes for their distracting behavior.
8. 9-12 problems will be posed. So far as possible, problems will avoid dependence on detailed knowledge of a particular applications area or particular contest language.
9. **Rank-list will be frozen in the final hour of the contest. During this period, teams will only get verdict of their own submissions.**
10. Contestants cannot leave the contest arena during the contest without explicit permission from the judges. The contestants are not allowed to communicate with any other contestants (even contestants of the same team) or coaches when they are outside the contest arena.
11. Teams can bring any number of pages of printed materials with them. They can also bring five additional books. But they are not allowed to bring any electronic devices like calculator, CD, DVD, Pen-drive, IPOD, MP3/MP4 players, floppy disks, watches(smart, digital, analog) etc. **Teams CANNOT bring their own keyboard, mouse etc.**
12. With the help of the volunteers and judges, the contestants can have printouts of their codes for debugging purposes. Passing printout of codes to other teams is strictly prohibited and may cause disqualifications of teams involved.
13. Teams should inform the volunteers/judges if they don't get any verdict/reply within 10 minutes of submission/clarification. Teams should also notify the volunteers if they cannot log into CodeMarshal. These sort of complains will not be entertained after the contest.
14. Codes must not use any system command or use multi-threading. Contestants must not attempt to access any other computers other than their own in the network. Violating these rules may result in disqualification.
15. **Teams using Java should be extra careful about TL and ML, since problems are not tested with Java.**
16. **Each team will be given the same machine in the same location during mock and main contest. That's why, teams are strongly advised to attend the mock. Any issues during mock should be notified to the judges via the clarification system.**
17. **Any member of a team, if late, may not be allowed to enter the contest arena after 30 minutes from the starting of the contest.**
18. The decision of the judges is final.

IDEs:

1. CodeBlocks
2. Geany
3. IntelliJ Idea
4. Netbeans

Compilers (In CodeMarshal):

1. C, C++: GCC 8.2
    a. C compile command: `gcc -O2 -static filename.c -lm`
    b. C++ compile command: `g++ -O2 -static -std=c++11 filename.cpp`
2. Java: JDK 1.8 OpenJDK

Java compile command: `javac -J-Xmx2048M filename.java`

A

Problem A

Input: Standard Input
Output: Standard Output

Kings in 2D Grid

There will be a 2-Dimensional RxC grid, where R is the row number and C is the column number. The rows of this grid are numbered from top to bottom starting with index 1, and columns are numbered from left to right starting with index 1. In this grid, two kings are placed where one king is white and the other king is black. The white king is placed on (R1, C1) coordinate and the black king placed on (R2, C2) coordinate. They are initially placed on two different cells of the grid in such a way that they are not adjacent to each other (two cells on the grid are adjacent if they share a side or a corner). Alice plays with white king and Bob plays with black king.

In each turn, one can move his/her king to any adjacent cell from the current cell, given that the new position of the king can not be adjacent to the other king. If there is no valid move for the player in turn, the game ends immediately. Otherwise, if the game continues for 10^9 moves in total (counting both Alice and Bob's move), the game is called to an end.

Alice and Bob play alternatively. Alice will move first. Alice will always try to minimize the different cell number visited by the black king of Bob. Bob will always try to maximize the different cell number visited by his black king. Both Alice and Bob will play optimally.
You have to calculate the different number of cell Bob's black king can visit before the game ends.

Input

Input starts with test case number **T (1 ≤ T ≤ 2500)**. Each of the following **T** Lines contains six space separated integers **R, C, R1, C1, R2, C2** where **1 ≤ R, C ≤ 14, R × C ≤ 14;** and **maximum of R and C will be greater than 2;** and **1 ≤ R1, R2 ≤ R, 1 ≤ C1, C2 ≤ C. (R1, C1) and (R2, C2)** will not be adjacent to each other.

Output

For each test case, output a single line in the format "**Case X: Y**" without the quotes. Here, **X** is the case number and **Y** is the maximum number of distinct cells visited by the black king.

Sample Input	Output for Sample Input
2 1 7 1 4 1 1 2 5 2 1 1 5	Case 1: 1 Case 2: 4

B

Problem B

Input: Standard Input
Output: Standard Output

Mysterious LCM

You will be given an array **A** of **N** integers and another integer **X**. You have to find the minimum length of a subsequence from **A** such that, the LCM of all the elements in that subsequence is exactly **X**.

Input

The first line of the input contains a single integer **T**, which denotes the number of test cases. The first line of each test cases contains two integers **N** and **X**. The second line of a test case contains **N** space separated integers of the array **A**.

Constraints

- **$1 \leq T \leq 50$**
- **$1 \leq N \leq 50000$**
- **$1 \leq A_i, X \leq 10^{18}$**

The sum of **N** over all the test cases in an input file ≤ **10^6**

Output

For each test case, output a single line in the format "**Case T: D**" without the quotes. Here, **T** is the case number and **D** is the desired answer denoting the minimum length of a valid subsequence whose LCM is exactly **X**. If there is no solution, then print **-1** instead.

Sample Input	Output for Sample Input
3 8 60 2 9 12 15 16 21 25 40 8 90 2 36 44 49 50 63 64 81 8 138600 40 2200 45 216 175 735 15 36	Case 1: 2 Case 2: -1 Case 3: 3

Explanation

In our first sample case, the LCM of the elements in the valid subsequence (12, 15) is 60. There is no way to select a valid subsequence of length less than 2.

Warning: Dataset is huge. Please use faster I/O methods.

C

Problem C

Input: Standard Input
Output: Standard Output

Swipe Your Time Away

Bangladesh Association of Problematic Society (BAPS) is about to release their new mobile game SYTA (Swipe Your Time Away). According to the opinion of several game specialists and psychologists, this is going to be one of the greatest addictive game in the history of mankind. This game will ruin the civil society. Students will drop out of schools, family bonding and relationships will be destroyed, people will only swipe their time away just playing SYTA day and night.

SYTA is a board game played on **N** by **M** board (**N** rows and **M** columns). The left uppermost cell is numbered as $(1,1)$ and the right lowermost cell is numbered as (N,M). Each cell contains exactly one ball. The colors of these balls are of different types. There are five colors and they are: 1. Red 2.Green 3.Blue 4.Yellow and 5. Purple. The goal of the game is to determine the largest **Cross** shape with the same color within the board. Formally, cell (x,y) is a center of a **Cross** shape if it has the following properties:

1. In x^{th} row all the balls from column y_0 to y_1 are of same color
2. In y^{th} column all the balls from row x_0 to x_1 are of same color
3. $x_0 < x < x_1$
4. $y_0 < y < y_1$
5. The size of that **Cross** shape is, $D = (x_1 - x_0) + (y_1 - y_0) + 1$

We are looking for someone who can crack down this devastating game. All you need to do is to find out the size of the largest Cross shape in shortest period of time. If you can do so, world will be saved, and in return we will gift you one colorful balloon.

2	5	5	1	4	3
5	5	5	5	4	5
3	5	4	4	4	4
1	5	4	2	4	4
2	1	2	2	4	1

In the first test case, there are 2 Cross shapes.

cell (2,2) and (3,5) are the center of these Cross and the size of these cross shapes are 7 and 8 respectively.

So, the largest size is 8.

Input

Input starts with an integer **T** denoting the number of test cases. Each of the test cases starts with two integer **N** and **M** denoting the number of rows and columns in the board. Next, there will be **N** lines of inputs each containing **M** space separated integers. The j^{th} integer of the i^{th} line is C_{ij}, representing the color of ball in the cell numbered (i,j)

Output

For each test case, output a single line in the format "**Case T: D**" without the quotes. Here, **T** is the case number and **D** is the desired answer.

Constraints

- **1 ≤ T ≤ 45**
- **3 ≤ N ≤ 1000**, **3 ≤ M ≤ 1000** (In more than **90%** cases **3 ≤ N, M ≤ 200**)
- **1 ≤** C_{ij} **≤ 5**

Sample Input	Output for Sample Input
2 5 6 2 5 5 1 4 3 5 5 5 5 4 5 3 5 4 4 4 4 1 5 4 2 4 4 2 1 2 2 4 1 3 3 1 2 3 4 5 1 2 3 4	Case 1: 8 Case 2: 0

Warning: Dataset is huge. Please use faster I/O methods.

D

Problem D

Input: Standard Input
Output: Standard Output

DarkCity, CrimsonCity of FlightLand

Flightland is a wonderful country. There are **N** cities in Flightland. A city's position can be determined by 2D coordinate system. You can go from any city to another city using an airplane.

Say an airplane is departed from **City P** and landed on **City Q**. **City P**'s coordinate is **(Xp, Yp)** and **City Q**'s coordinate is **(Xq, Yq)**. Here is some calculation for air travel costs for a single flight:

- The airplane chooses a path from **City P** to **City Q** according to your choice. But remember, you have to choose a path such that the length of the path is always an integer. It's for the safety of the passengers. You don't have to worry much about that. And you can ignore the altitude of the airplane. So, you can think this path to be a straight line, polyline or curve on a 2D plane. Say the length of the path is **L**.
  Example: Say, the airplane took-off from **(0, 0)** and landed on **(5, 5)**. You can not choose the straight line between these points, as the length of the path will not be an integer. Rather you can choose a path from **(0, 0)** to **(1, 0)** to **(1, 2)** to **(5, 5)** with straight lines. In that case, the length of the path will be **8**, which is an integer. But these turning points of the airplane can also be non-integer coordinates and also the paths can be arbitrary curves too. Like you can choose an arc from **(0, 0)** to **(5, 5)** such that the length of the arc is **11**, **10**, **9** etc. Simply, you can choose any route plan as your choice. This can be a straight line, polyline, curve etc. But the length of the path must be an integer.
- The airplane has a mass of **A**, including the mass of all passenger, the luggages and some other stuffs. It's basically the mass of the whole plane except its fuel's mass.
- The airplane starts with **F** amount of fuel, which is a positive real number. You can choose the amount of fuel for the journey. So, the total mass of the airplane is **T** = **A** + **F**. It is obvious that the airplane needs sufficient amount of fuel to complete the flight. Otherwise the airplane will crash and you are never going to meet your destination.
- For a unit distance of flight, Airplane consumes **TotalMass** / **D** amount of fuel. So, you can see the plane is constantly(in this case, per unit distance) losing its fuel thus its total mass. Here, **D** is the consumption rate which is constant.
- Price of a unit amount of fuel is **C**.
- There is some extra base cost for take-off, landing, airport fee etc. Say this extra cost is **B**.
- So, the total cost for a single flight is **F** × **C** + **B**.
- Size of airplane and airports are negligible. You can think them as points.
- You can not do a mid-air refueling. That means you start your flight with **F** amount of fuel and you have to finish your flight using that fuel only.

- Each flight is independent, and they will be operated with a brand new airplane. So, if you have some fuel remaining on your last flight, you can not use it in this flight. Besides, the initial amount of fuel used for each flight can be different too.
- You can take flights only from one city to another, and every city has an airport. Note that, airports are situated only in the cities.

Here is an example of fuel consumption. Say, **L = 5, A = 100, D = 100**.
If you choose **F = 10,** then here is the fuel consumption history:

When airplane travels **0** unit distance, means when the airplane will take-off, **TotalMass = 100 + 10 = 110**. So, in the next unit distance airplane will need **110 / 100 = 1.1** unit of fuel.
When airplane travels **1** unit distance, **TotalMass = 100 + 8.9 = 108.9**. So, in the next unit distance airplane will need **108.9 / 100 = 1.089** unit of fuel.
When airplane travels **2** unit distance, **TotalMass = 100 + 7.811 = 107.811**. So, in the next unit distance airplane will need **107.811 / 100 = 1.07811** unit of fuel.
When airplane travels **3** unit distance, **TotalMass = 100 + 6.73289 = 106.73289**. So, in the next unit distance airplane will need **106.73289 / 100 = 1.0673289** unit of fuel.
When airplane travels **4** unit distance, **TotalMass = 100 + 5.6655611 = 105.6655611**. So, in the next unit distance airplane will need **105.6655611 / 100 = 1.056655611** unit of fuel.
When airplane travels **5** unit distance, The airplane will land and **4.608905489** unit of fuel will remain the airplane.

You live in DarkCity and your dream girl/boy lives in CrimsonCity. You want to go from DarkCity to Crimson city. You have to bear the full cost for each flight. Can you calculate the minimum cost to visit your dream girl/boy?

Input

Input starts with an integer, **TC**, denoting number of test cases. For each test case, the first line contains 5 integers **N**, **A**, **B**, **C**, **D** denoting number of cities in Flightland, mass of airplane, extra base cost per flight, fuel cost, consumption rate. The next **N** lines contain 2 integers denoting the coordinates for each city. First city is always DarkCity and the last city is always CrimsonCity.

Output

For each test case print a single line containing a real number that denotes the minimum cost.
Your answer will be accepted if its relative or absolute error does not exceed 10^{-9}.
Mathematically, if your answer is A and the jury's answer is B, then your answer is accepted if and only if $\frac{|A-B|}{max(1,|B|)} \leq 10^{-9}$.

Constraints

- **$1 \le TC \le 100$**
- **$2 \le N \le 200$**
- **$10^5 \le A \le 10^6$**
- **$1 \le B \le 10^7$**
- **$1 \le C \le 100$**
- **$10^9 \le D \le 2 \times 10^9$**
- **$0 \le$ coordinates $\le 10^9$**

Sample Input	Output for Sample Input
2 2 787379 570 57 1071473853 0 0 100000 100000 3 695972 544 5 1044478508 0 0 100000 100000 500000 500000	6494.1049259617 2900.6463588302

E

Problem E

Input: Standard Input
Output: Standard Output

Consecutive Letters

You are given a string S containing only uppercase English letters. There are Q queries. Each query can be of two types,

- **1 i**: Find the maximum size of the segment ***[b, e]*** where **0 ≤ *b* ≤ i ≤ e < |S|** and substring ***S[b...e]*** contains only the letter ***S[i]***. A Substring is a contiguous sequences of characters in a string.
- **2 i**: Change the character in index ***i*** with the character '**#**'.

For both type of queries, **S[i]** will not contain the character '#'.The characters of the string are indexed from 0.

Input

The first line contains number of test cases **T (1 ≤ T ≤ 25)**.
For each test cases, the first line contains the string **S (1 ≤ |S| ≤ 200000)**. The 2nd line contains number of queries **Q (1 ≤ Q ≤ 100000)**. Each of the next **Q** lines contains one query in the format mentioned in the problem statement.

Output

For each test case, first print the test case number and output of every query of type 1 in a single line.

Sample Input

```
2
AABBBCCCC
5
1 0
2 1
1 0
2 2
1 3
XXYYY
3
1 3
2 3
1 2
```

Output for Sample Input

```
Case 1:
2
1
2
Case 2:
3
1
```

Warning: The input file is huge, please use fast I/O.

F

Problem F

Input: Standard Input
Output: Standard Output

Palindromadness

You'll be given a string **S** of length **N**, containing lowercase English letters only. Let's define a function **f(x)** over this string.
f(x) denotes the number of quadruple of indices **(i, j, p, q)** where,

1. **A = S[i...j]** and **B = S[p...q]** are substrings of string **S** where **1 ≤ i, j, p, q ≤ N , i ≤ j and p ≤ q.** That is, **A** and **B** can partially overlap, or contain each other, or be completely separate or be the same substrings in string **S**.
2. Both **A** and **B** are palindromes.
3. **A** is a substring of **B**
4. Length of **A** is equal to **x**.

Substrings are contiguous sequences of characters in a string. Palindromes are strings, that read the same forwards and backwards.

You'll also be given a base and a mod value. You'll have to print $(\sum_{i=1}^{n} f(i) * base^{n-i}) \% mod$.

Input

First line of input will contain an integer **T (1 ≤ T ≤ 10^5)**, denoting the number of test cases. First line of each test case will contain three integers **N, base (1 ≤ base ≤ 10^9), mod (1 ≤ mod ≤ 2 × 10^9)**. Following line will contain a string of length **N**, containing lowercase English letters only. Summation of **N** over all test cases will not be greater than **10^6**.

Output

For each test case, first print the case number, starting with 1, followed by the answer for that case. Check the sample I/O for more details.

Sample Input

```
5
13 100 999
welcometomist
7 1000 1000000000
topspot
11 23167 21898192
abbabobaxab
21 123456 123456789
amanaplanacanalpanama
32 72817 728917897
bobxyxthehtxyxbuildercantfixyxit
```

Output for Sample Input

```
Case 1: 21
Case 2: 2000001
Case 3: 4104229
Case 4: 85459969
Case 5: 721428038
```

Explanation

The f(x) function(1 based) for the first three cases are given below.
Case 1: { 21, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
Case 2: { 28, 0, 3, 0, 2, 0, 1 }
Case 3: { 97, 2, 5, 1, 2, 0, 0, 0, 0, 0, 0 }

G

Problem G

Input: Standard Input
Output: Standard Output

Decode The Alien Message

Professor Neo has established contact with an alien civilization! His team have been intercepting encoded messages since then from the alien colony. Messages contained chunk of random integers. After getting no clue for weeks finally they were able to decode it.

The random integers are to be decoded according to their parity. An even integer correspond to 1, and an odd integer corresponds to 0. So, every chunk of integers can be converted to a string containing 0 and 1. As there are lots of messages, you came into play!

You have to write a code that decodes the messages. There are two points you have to be careful about:

1. There will be no chunk where all of the integers are odd. That means, in a chunk there will be at least one even integer. For example, there will be no chunk like: 5 5 5.
2. After the message is decoded, you have to skip the leading zeros before printing.
   For example, consider a chunk of three integers: 5 5 4. After decoding you will get 001, but you need to print 1.

Input

The first line will contain an integer **T (T ≤ 100)**.
Each test case will contain an integer **N (1 ≤ N ≤ 50)** in the first line, and the second line will contain **N** space separated positive integers within the range **[1, 1000]**.

Output

For each test case, output a single line in the format "**Case T: D**" without the quotes. Here, **T** is the case number and **D** is the desired decoded message.

Sample Input	Output for Sample Input
3 1 4 3 4 4 5 3 5 5 4	Case 1: 1 Case 2: 110 Case 3: 1

H

Problem H

Input: Standard Input
Output: Standard Output

Triangle Inside Rectangle Inside Pentagon

So last day I was teaching geometry to my cousin. He is interested in learning geometry but could not keep his attention. So I tried to make things interesting by asking him to give me any geometric challenges he can think of.

All was going good, but then I started to teach him regular convex polygon. A regular convex polygon is a polygon that is equiangular (all angles are equal in measure), equilateral (all sides have the same length) and all interior angles are strictly less than 180 degrees. Like equilateral triangle, square and so on. But then he gave me a challenge:

- *Brother, can you fit a equilateral triangle in a square*
- Ok, here it is.
- *Ok, now fit the square in a regular convex pentagon.*
- Umm, ok, this might take some time!
- *Then fit that in a regular convex hexagon.*
- Stop!
- *Then fit that in a regular convex heptagon.*
- Please Stop!!
- *Also make them as small as possible*
- ಠ_ಠ

You will be given the length of side of the equilateral triangle **s** and **N**. You have to print the area of minimum regular convex **N**-gon, which will contain a regular convex **N-1**-gon(if **N** > 3), which will contain a regular convex **N-2**-gon(if **N** > 4), which will contain a regular convex **N-3**-gon(if **N** > 5) and so on. Finally, a square(**4**-gon)(if **N** > 3) will contain the triangle.

Input

The first line contains an integer **T** (**T ≤ 10^5**), denoting the number of test cases. Each test case contains two integers **N** (**3 ≤ N ≤ 10^5**) and **s** (**1 ≤ s ≤ 10^3**).

Output

For each test case, print a single line containing a real number that denotes the minimum area. Your answer will be accepted if its relative or absolute error does not exceed 10^{-4}. Mathematically, if your answer is A and the jury's answer is B, then your answer is accepted if and only if $\frac{|A-B|}{max(1,|B|)} \leq 10^{-4}$.

Sample Input

```
3
3 1
5 4
10 1
```

Output for Sample Input

```
0.433012702
22.542684260
3.484763487
```

I

Problem I

Input: Standard Input
Output: Standard Output

Fibonacci Power Sum

The fibonacci series is defined as below:

fib(0) = 0, fib(1) = 1

fib(n) = fib(n-1) + fib(n-2) for n > 1

Given three integers **N**, **C** and **K,** find the summation of the following series:

$fib(0)^K + fib(C)^K + fib(2*C)^K + fib(3*C)^K + \ldots + fib(N*C)^K$

Since the answer can be huge, output it modulo **1000000007** (**10^9+7).**

Input

The first line contains an integer **T**, denoting the number of test cases. Each test case contains three space separated integers in the order: **N**, **C** and **K**.

Constraints

- **$1 \le T \le 100$**
- **$0 \le N \le 10^{15}$**
- **$1 \le C, K \le 10$**

Output

For each test case, output a single line in the format "**Case T: D**" without the quotes. Here, **T** is the case number and **D** is the desired answer denoting the sum of the series.

Sample Input	Output for Sample Input
5 10 1 1 5 2 2 3 3 4 1000000007 7 9 996969696969696 9 6	Case 1: 143 Case 2: 3540 Case 3: 1340448 Case 4: 880410497 Case 5: 689328397

Problem A (Kings in 2D Grid)

Setter: Suman Bhadra, Alternate Writer: Sabit Anwar Zahin, Shahed Shahriar

Editorial: You must notice that R*C <= 14. This makes the problem very simple. How many different grids are really possible? Let's divide our solution into a few cases.

Case 1 (R = 1, C = N or R = N, C = 1):
We have just one row or column. The kings will never be able to cross each other. So Bob will try to move his black king towards the white king as long as it's possible to make a valid move. And then he will just follow the reverse path.

Case 2 (R = 2, C = N or R = N, C = 2):
A similar solution to the first cases. Because the kings still can never cross each other.

Case 3 (R = 3, C = 4, or R = 4, C = 3, or R = 3, C = 5 or R = 5, C = 3):
Do backtracking. As there are maximum 14 distinct cells so we can also do bitmask dp. There are not many distinct configurations for this case. So one can also backtrack and precalculate the solution for all possible cases.

Time complexity: $O(2^{R*C})$ **or** $O(R * C)$

Problem B (Mysterious LCM)

Setter: Tarango Khan, Alternate Writer: Shahed Shahriar, Suman Bhadra

Editorial:
Hint 1: X can have a maximum of 15 distinct prime numbers in its prime factorization. Because the product of the smallest 15 prime numbers is close to 10^18.

Hint 2: For any number A, if A has a prime factor p which is not a prime factor of X then A will never contribute to our solution.

Hint 3: We can extend our previous hint a bit more. If A has a prime factor p and k is the maximum power such that p^k divides A then the maximum power of p in X must be >= k. Otherwise, A will never contribute to our solution.

From the above observations, we can easily discard the unnecessary numbers from the array. Now let's get to the solution.

Lets say X = p_1^e_1 * p_2^e_2 * .. * p_n^e_n, and maximum value of n is 15 (From our hint 1).

Now let's assign an n-bit mask to each index of the array. The mask will have j'th bit on, if the power of p_j in a_i is exactly e_j. That means if you take this number, you will get p_j^e_j in lcm.

Now the problem is converted to: "Given some masks. Find the minimum number of masks that ORs to (111….111), n 1s. Which is basically 2^n-1"

This can be calculated with DP or FWHT.

FWHT Approach:
You can do FWHT / OR Convolution n times on the count of the masks. And the first time you get a positive count of 2^n-1 mask print the iteration number. This has complexity O(n^2 2^n)

DP Approach:
Let dp[mask] = minimum number of masks you need to take to get a super mask of 'mask'.

Initially, dp[mask] = 1, if the mask is in the array. But you also need to mark 1 in all the masks that are sub masks of some masks from the array. (Confusing? That is if you are marking mask 1011, then you also need to mark 1010, 1001, 0011, 0010, 0001 etc)

Why? Because if you can make a mask selecting a single number from the array then definitely you can make their sub masks in a single move as well.

This can be done with SOS DP like thing in O(n*2^n). Basically, we can just propagate from the large masks to smaller masks.

Then the rest of the DP is obvious:

```
For each mask M:
    For each sub mask X of M:
        dp[M] = min(dp[M], dp[X] + dp[M ^ X]);
```

Our final answer is = DP[2^n-1]

Time complexity: $O(3^n)$

You can also do subset convolution on min-sum subring, with almost the same complexity as the previous approach: $O(n^2 * 2^n)$

Editorial credit: Rezwan Arefin

Problem C (Swipe Your Time Away)

Setter: Mohammad Ashraful Islam, Alternate Writer: Rafsan Jani

Editorial:
Let’s define an array V[][][] where,

V[r][c][0] = maximum number of consecutive cells of color C[r][c] starting from the position (r, c) to its left.
V[r][c][1] = maximum number of consecutive cells of color C[r][c] starting from the position (r, c) to its right.
V[r][c][2] = maximum number of consecutive cells of color C[r][c] starting from the position (r, c) to its down.
V[r][c][3] = maximum number of consecutive cells of color C[r][c] starting from the position (r, c) to its top.

This array V can be calculated iterating from the top left cell to the bottom right cell and then again from bottom right cell to the top left cell.

Now we can consider each of the cell (r, c) as the cross point (or center) of a plus sign and then find the maximum number of cells having the same color as C[r][c] which is connected to the cell (r, c) using our pre-calculated array V. The maximum one from all possible center of a cross sign is our final answer.

Time complexity: $O(R * C)$

Problem D (DarkCity, CrimsonCity of FlightLand)

Setter: Shahed Shahriar, Alternate Writer: Tarango Khan, Mehdi Rahman

Editorial: Let's say we want to calculate the minimum cost for a flight between city u and city v where the distance between u to v is P and our initial total fuel is F. If P is not an integer then obviously the nearest greater integer value is the length of our optimal path. Because we can always expand the straight line path in such a way that it becomes a curve.

Now let's see how the fuels get consumed in each step following the procedure explained in the problem statement.

After the first unit distance, airplane consumes $C1 = \frac{(A+F)}{D}$ amount of fuel. So the remaining fuel is $(F - C1)$.

After the second unit distance, airplane consumes $C2 = \frac{(A+F-C1)}{D}$ amount of fuel. So the remaining fuel is $(F - C1 - C2)$.

After the third unit distance, airplane consumes $C3 = \frac{(A+F-C1-C2)}{D}$ amount of fuel. So the remaining fuel is $(F - C1 - C2 - C3)$.

This way, after all the unit distances are covered, there might still be some fuels left. But if this really happens then did we really need to carry these extra fuels? No. So in the optimal process, the remaining amount of fuel must be zero after all the unit distances are covered.

From the above calculations, we can construct an equation for F which eventually gives us the solution, $F = \frac{(X * A)}{(D - X)}$, where $X = ((1 - R^P) * D)$ and $R = (1 - 1/D)$.

So the minimum cost for a single flight $= F * C + B$.

We have to calculate this cost for the flight between all pair of cities and then we can just run a Dijkstra or Floyd Warshall algorithm which will find the minimum total cost to reach the city N from the city 1.

Time complexity: $O(N^2 * log(N))$

Problem E (Consecutive Letters)

Setter: Shafaet Ashraf, Alternate Writer: Md. Imran Bin Azad, Rafsan Jani

Editorial: The problem can be solved using the union-find data structure or STL set. We can consider a substring of the same characters as a single component. For an update operation, we can just divide or merge the components connected to that specific position and update their parents accordingly. We'll also need to maintain the size of each components to answer our queries.

The detailed editorial can be found in the following blog post:
http://en.shafaetsplanet.com/problem-solving-consecutive-letters-mist-inter-university-contest-2019/

Problem F (Palindromadness)

Setter: Sourav Sen Tonmoy, Alternate Writer: Sabit Anwar Zahin

Prerequisite: Palindromic Tree [Paper, Tutorial (EN), Tutorial (BN)]

Editorial: Firstly let's discuss an $O(N^2)$ solution. Build the palindromic tree. Then generate and store the frequencies of all unique palindromes (read nodes) [Section 2.4, Proposition 3 of the original paper]. Now for each node v, traverse to all those nodes u, who are reachable from v via suffix link chains and tree parental chains. Here, obviously, palindrome stored at node u is a substring of palindrome stored at node v. So, we can now add $freq[u] * freq[v]$ to $f(length(u))$. Of course, we will also have to add $freq[v] * freq[v]$ to $f(length(v))$.

Now, how can we decrease the complexity, and move towards the desired solution? For that let's rewrite the previously discussed $O(N^2)$ solution. Previously we iterated over v, and looked for u. Now, we will iterate over u, and look for v. Who are v nodes? The nodes, from which, node u can be reached via suffix link chains and tree parental chains.

We can reach all nodes v, from node u, by checking the nodes in the subtrees of those nodes w, who marked u as their longest suffix palindrome. We will sum up the frequencies of palindromes of each node under all the subtrees beforehand. Then for all those w nodes, we will add $freq[u] * \Sigma freq[p]$ to $f(length(u))$, where p is a node under the subtree of node w. To avoid overcount, we will mark discovery and finishing times of all nodes. We will sort all those w nodes, by discover time. Then we can easily check whether the current w node is already under the subtree of a previously checked w node. In this solution, we will also have to add $freq[u] * freq[u]$ to $f(length(u))$.

Let's talk about correctness. Let, x be a node, who marked u as its longest suffix palindrome. Let, y be another node, who marked x as its longest suffix palindrome. Also let x not be reachable from y via parental chain. So are we missing out all those nodes under node y, to mark as node v? No, it can be observed that there exists at least one node p, which can be reached via parental chain from node y, where node p marks node u as its longest suffix palindrome.

Time Complexity: As each node marks exactly one node as its longest suffix palindrome, so over all u nodes, the total number of w nodes will be N. We are sorting these w nodes, using discovery time, which costs $O(N\ log\ N)$. And that is the complexity of this solution. As you can see, the complexity is mostly dominated by the sorting algorithm used. By using a linear sorting algorithm, we can even reach a linear time solution for this problem.

Our solution runs well under 1s. We offered 10s time limit to encourage any other interesting approaches, with slightly higher complexity. Unfortunately, nobody came close to a valid solution during the onsite contest or the online replay.

Problem G (Decode The Alien Message)

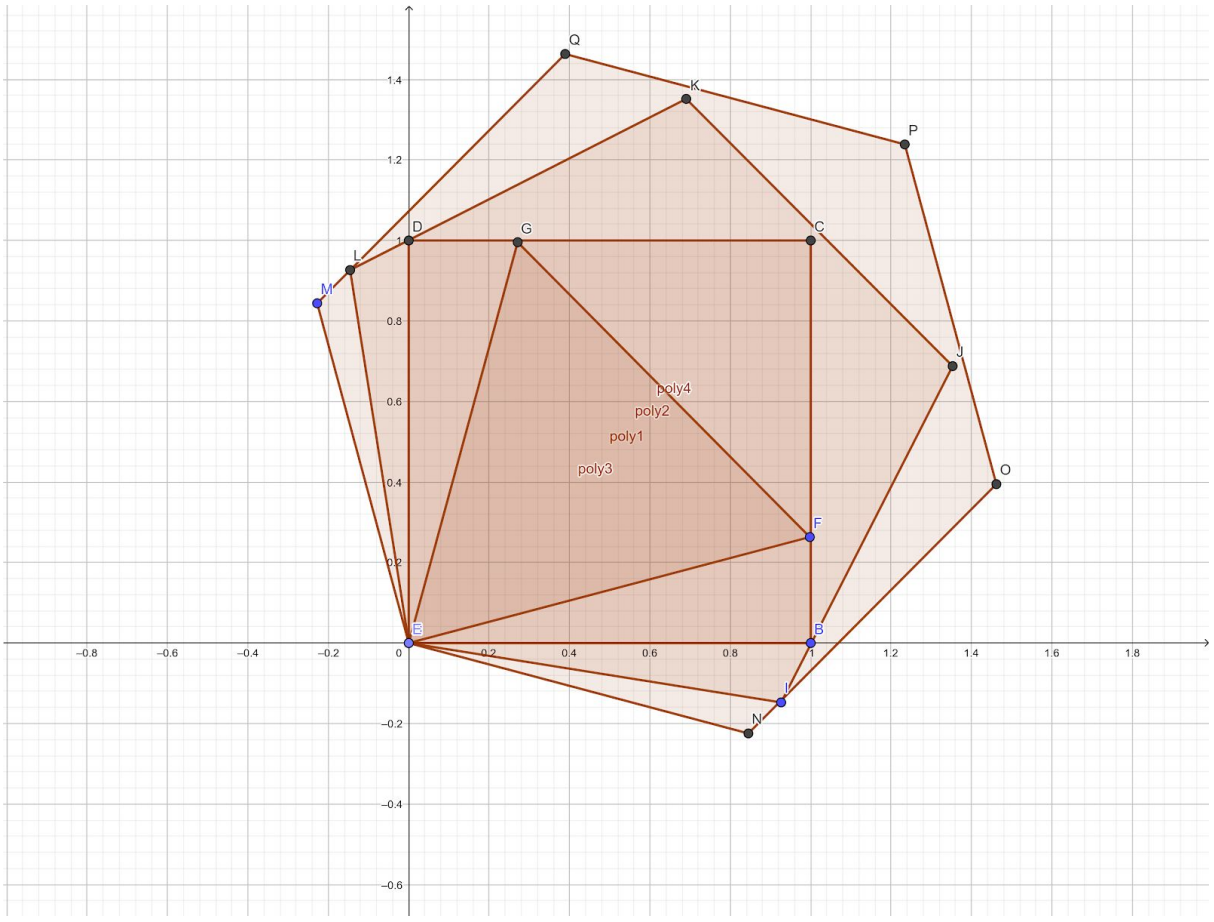
Setter: Mehdi Rahman, Alternate Writer: Muhammad Ridowan

Editorial: Very straight forward. Just make a list of the binary values and erase the trailing zeroes.

Problem H (Triangle Inside Rectangle Inside Pentagon)

Setter: Muhammad Ridowan, Alternate Writer: Mehdi Rahman, Mohammad Maksud Hossain

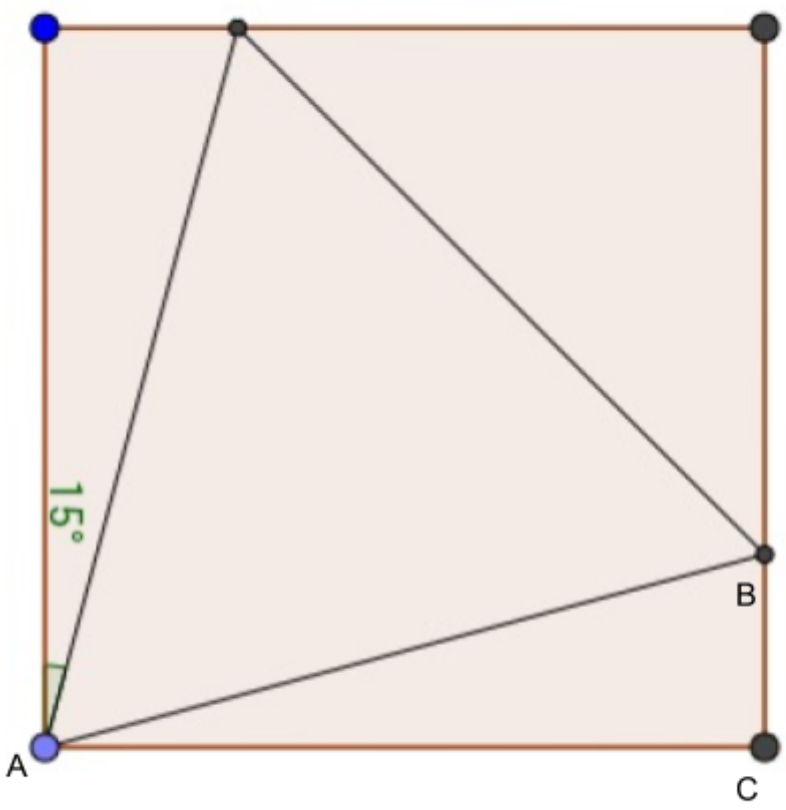
Editorial:



The optimal result is when the polygons have the same corner and placed in such a way that the internal polygon is just in the middle of that angle of the outer polygon. Prove of it is more of intuitive thinking that fitting like this will use the least space. Prove for triangle inside square and square inside pentagon can be found here

1. https://math.stackexchange.com/questions/59616/find-the-maximum-area-possible-of-equilateral-triangle-that-inside-the-given-squ
2. http://ken.duisenberg.com/potw/archive/arch98/981113sol.html

Now as finding minimum N-gon to contain a N-1-gon, let think their sides are s_{N-1} & s_N and angles a_{N-1} & a_N.



In the image

- <BAC = $(a_N-a_{N-1})/2$
- <ACB = a_N
- <ABC = Pi - $(a_N-a_{N-1})/2$ - a_N
- AB = s_{N-1} & AC = s_N

Now using sin rule of triangle, we can say

- AC/Sin<ABC = AB/Sin<ACB
- AC = AB * Sin<ABC /Sin<ACB
- $s_N = s_{N-1} * \text{Sin}(\text{Pi} - (a_N-a_{N-1})/2 - a_N) / \text{Sin}(a_N)$
- $s_N = s_{N-1} * \text{Sin}((a_N-a_{N-1})/2 + a_N) / \text{Sin}(a_N)$

As $a_N = \frac{(n-2)\pi}{n}$

So we can pre-compute that for N-gon, what is the multiple needed with the side of the triangle. Then for each query O(1) answer.

The area can be calculated by several means, a direct formula is $\frac{1}{4}ns^2\cot\left(\frac{\pi}{n}\right)$

Problem I (Fibonacci Power Sum)

Setter: Sabit Anwar Zahin, Alternate Writer: Sourav Sen Tonmoy

Editorial: The problem boils down to a linear recurrence, which can be derived using combinatorics (similar to expanding binomial coefficients). You should be able to figure it out after trying for **K=2 and C=1** in pen and paper.

For more details, check the following OEIS link, since the recurrence coefficients are basically rows of the signed fibonacci triangle.

Or if you're feeling lazy, you can simply use the Berlekamp-Massey algorithm :)

Time Complexity: $O(K^3 * log(N))$

Fun fact: Many contestants thought the recurrence would be of size **C*K** because of the constraints. **C** is actually irrelevant. The constraints were only kept relatively small with a large TL so that setter's Python solution can pass, and not to mislead the contestants.