ICPC Dhaka Regional Preliminary Contest 2022

Hosted By Green University of Bangladesh

Problem A: A Game with Grandma

Setter:

- Shafaet Ashraf

Tester & Alter writers:

- Nafis Sadique
- Aleksa Plavsic
- Arghya Pal

Problem type:

- Game Theory, DP

Solution idea:

If you're familiar with Grundy Numbers, you can use the divide and conquer technique to solve it. To do this, loop through the columns of the grid and place a box in all possible positions. Whenever a box is placed, it divides the whole grid into two parts. Solve each sub-problem recursively by calculating the Grundy number. The base case is when the grid size is less than 1. This solution has a time complexity of O(N^3) and will pass the time limit.

Solution 2:
Consider a pair of columns, i and i+1 where we cannot place a tile occupying those two columns. Then we can separate the board into two parts, columns <= i and columns > i, and moves on one part does not affect the other. Let's find all such i and divide the board to separate 3xk boards, where we can place tiles in any pair of columns in each board. Now, we need to use the theory of grundy numbers to calculate grundy values for each sub-board and combine them to get the final answer.

Note that, we can only place at most one tile in any column. So, in the smaller 3xk boards, it doesn't matter if there are any blocked cells, because they don't impede our ability to place tiles. So, we can treat them as 3xk empty boards.

Now all there is left is to calculate grundy values for 3xk boards. You can do it with a dp, considering all positions where to put a tile. This is an O(n^2) sol. Actually, if you generate grundy numbers, you find that they form a periodic sequence for high values of n. So, grundy can be calculated in O(1). However, this was not necessary to solve the problem.

Problem B: Transform the Array

Setter:

- Raihat Zaman Neloy

Tester & Alter writers:

- Md. Mahbubul Hasan
- H. M. Ashiqul Islam
- Muhiminul Islam Osim

Problem type:

- Number theory, finding prime divisors

Solution idea:

For the first type of transformation - Let's find two sets of primes - P_1 and P_2. P_1 will contain all the prime numbers that are prime divisors of at least one number in array **A**, similarly P_2 will contain all the prime numbers that are prime divisors of at least one number in array **B**. Now, for this type of transformation, output will be "Yes" iff $P_2 \subseteq P_1$.

For the second type of transformation - we need to reduce both of the arrays with the GCD of array A. Then if both of the arrays become equal then we can print Yes, otherwise No. This is true because the largest number we can multiply or divide array A with is their GCD.

Problem C: Omago Virus

Setter:

- Pritom Kundu

Contributors:

- Nafis Sadique (Alter)
- Derek Kisman, Md Mahbubul Hasan (Idea Contribution)

Problem type:

- Combinatorics, DP

Solution idea:

Solution 1:

Let **dp[i]** be the expected number of days to infect all persons given that **i** people are infected today. Clearly, dp[n] = 0 and dp[m] is our answer. Then,

$$dp[i] = 1 + \sum_{j \geq i}^{n} f(i, j) \cdot dp[j]$$

where **f(i, j)** is the probability that **j** people are infected on the next day given that currently **i** people are infected. Note that **dp[i]** depends on itself, but we can easily fix that by assuming dp[i] = x and solving for x.

Now, it's left to calculate **f(i, j)**. We can calculate **f(i, j)** using the principle of inclusion and exclusion. Take any set of uninfected people S. Let |S| = x. The probability that, none of them will be infected after all i people spread infections is $\left(\frac{\binom{n-x-1}{k}}{\binom{n-1}{k}}\right)^i$. There are $\binom{n-i}{x}$ such sets. By the principle of inclusion exclusion, probability that exactly n-j people will stay uninfected out of the rest n-i is

$$f(i,\ j)\ =\ \sum_{x=n-j}^{n-i}(-\ 1)^{x-(n-j)}\binom{x}{n-j}\binom{n-i}{x}\left(\frac{\binom{n-x-1}{k}}{\binom{n-1}{k}}\right)^i$$

Complexity: O(n^3)

Solution 2:

Let, dp[i][j] be the number of days (including today) to infect all persons given that currently i people are infected and j people will spread infections today. Clearly, dp[n][i] = 1 and our answer is dp[m][m]. Then,

$$dp[i][j]\ =\ \sum_{e=0}^{k} f(i,\ e)\ \cdot dp[i\ +\ e][j\ -\ 1] \text{ for } j\ \geq\ 1 \text{ and } dp[i][0]\ =\ 1\ +\ dp[i][i]$$

where f(i, e) is the probability that e new people will be infected after a person spreads infections given that currently i people are infected. It is easy to see that $f(i,\ e)\ =\ \frac{\binom{i-1}{k-e}\binom{n-i}{e}}{\binom{n-1}{k}}$

However, this solution has a problem, note that dp[i][0] depends on dp[i][i] and dp[i][j] depends on dp[i][j-1]. So, there is a circle of dependencies dp[i][i] -> dp[i][i-1] -> dp[i][i-2] -> … -> dp[i][0] -> dp[i][i]. We can fix this, by assuming that dp[i][i] = x, and use the formula for dp[i][j] to calculate dp[i][i] in the form a+bx and solve for x.

Complexity: O(n^2k)

Problem D: Omicron Juice 2023

Setter:

- Md Mahbubul Hasan

Tester & Alterwriters:

- Nafis Sadique

Problem type:

- Adhoc

Solution idea:

We will need some case analysis. First of all, if all the glass contains the same amount we are done. If not, then let's check how many glasses contain juice amount less than K. If all of them have less than K amount, then we can not pour juice to cups, so it is not solvable. If two glasses have less than K amount, then same logic. Although we might be able to pour from one of the glasses, we can not pour from any other glass to the second cup. Now let's check if the total amount of juice is equally distributable. If equally distributable, then we also know the target amount for each glass. It's actually enough to check that "the difference of initial amount in a glass and target amount must be divisible by k" (for each glass). Why? Try to simulate some examples.

Problem E: Cyclic Palindrome

Setter:

- Md. Muhiminul Islam Osim

Contribution:

- Pritom Kundu (Alter)
- Derek Kisman, Md Mahbubul Hasan (Idea Contribution)

Problem type:

- Fast Fourier Transform, Combinatorics

Solution idea:

For a particular shift of the string **S**, the beauty of the shift is 0 if there are two non-matching, English characters in mirrored positions. Otherwise, let's say p is the number of pairs of mirrored positions where both characters in the shift are '?'. Then, the beauty of the shift is 52^p % 10^9+7 as we can replace any of the 52 characters in those matching places to make the shift a palindrome. If one of the matching characters is '?' and the other is some English character, then we can replace '?' with the character, so there is a single way in this case.

Now, the main problem is how we can calculate the beauty for each of the n shifts. Here, FFT comes! We can add string **S** to itself to make string **SS**, then multiply some coefficients considering two strings (both **SS**) using FFT. We can apply a FFT for each character separately to find the matching positions of that character. But, this will be too slow.

We can use two FFTs. One is for calculating the matching of the English characters and the other is for calculating the matching of '?'. In order to match some English character with '?', we can set the coefficient to 0 for the place where the character is '?' and for other cases, we set the coefficients in such a way that after multiplication we get 1 for each matched character (hint: root of unity, https://cp-algorithms.com/algebra/fft.html#string-matching). We need some extra calculation from the resultant polynomial from FFT in order to fetch the matching of each shift. We check the validity of a potential palindrome using the second FFT. Also, the second FFT helps us calculate the number of ways to generate a palindrome for a cyclic shift. Finally, we can add the beauty of all shifts to our answer.

Overall Time Complexity: O(**T** * |**S**| * $\log_2$(|**S**|)).

Problem F: Proximity Card Data Statistics

Setter:

- Shahriar Manzoor

Tester & Alterwriters:

- Md Mahbubul Hasan

Problem type:

- Implementation

Solution idea:

Store the personal information data in suitable data structure and sort it by email address and then by timestamp to find multiple entries from the same student and to find the last entry. After that the required counting can be done very easily.

The limits set for the problem (Around 1200 students) is low for two reasons (1) Any novice implementation also passes (2) The data used in this problem is actually real life data of partial students of a CSE Department, so the number of students was limited. Many novice teams may not realise that two timestamps can be compared simply by strcmp() or similar function (As month is fixed to 2) and go for longer implementation.

Problem G: Portals and Roads

Setter:

- Tariq Bin Salam

Tester & Alterwriters:

- Pritom Kundu
- Arghya Pal

Problem type:

- DP

Solution idea:

If we build roads between nodes **i** and **i+1** for all **(a ≤ i < b)** then all the nodes **a, a+1, a+2,…, b** are connected by roads only. We can call it a segment of roads (segment **[a - b]**). We can have at most one node **c (a ≤ c ≤ b)** which will be connected to some portals. We note that the node has to be in the middle, so **c = (a + b) / 2** [except for segment containing node 1], to achieve min answer. We can call **c** the midpoint of the segment **[a-b]**. For each segment we can build one portal between its midpoint, **c** and node **1** to minimise the ans. Then all the nodes are connected in a tree structure.

A road of a segment can be visited several times to reach some node in that segment, but no road outside that segment will be visited because of the portal structure. So travelling time can be calculated independently for each segment. We can calculate the answer doing a dp of time complexity of **O (N * R * R)** and memory complexity of **O (N * R)** where state : number of remaining nodes * number of remaining roads

Denote, **dp[r][n]** = min ans to connect **n** nodes using exactly **r** roads
Then, **dp[r][n] = min (dp[r - i][n - i - 1] + cost)** for all **0 ≤ i ≤ r**; where **i** = number of roads used for the current segment. Cost of all possible segments can be pre-calculated.

Segment containing node 1 has to be handled separately.

Problem H: Mr. Dumb the police in charge

Setter:

- Mohammad Ashraful Islam

Solution: Raihat Zaman Neloy

Tester & Alterwriters:

- Pritom Kundu
- Arghya Pal

Problem type:

- Data structures, centroid decomposition

Solution idea:

This problem requires centroid decomposition to solve. According to the problem we need to find the distance of other nodes from the current node. To answer this, in the centroid decomposed tree, we can travel upwards to the root starting from the current node, and on each node on the path, we can try to find the closest node currently available in the queries. So, on each level of the centroid decomposed tree, we can maintain another DS, set, segment tree or whatever we are comfortable with to insert and delete the available nodes from queries. The overall runtime of this solution will be $N*\log^2(N)$ where N is the number of nodes in the tree.

Problem I: Image Cropping

Setter:

- Md. Muhiminul Islam Osim

Tester & Alterwriters:

- Nafis Sadique

Problem type:

- Binary Search, Geometry

Solution idea:

As the pivot point is at the middle of the cropped image and corner points need to be integer, it's obvious that for a valid cropping, four corner points should be (pivot.x - x * **W**, pivot.y - x * **H**), (pivot.x + x * **W**, pivot.y - x * **H**), (pivot.x + x * **W**, pivot.y + x * **H**) and (pivot.x - x * **W**, pivot.y + x * **H**), where x is an integer. We need to make x big enough so that the area of the cropped image will be maximum and the corner points don't go outside the given convex quadrilateral. We can easily do binary search on x, check the validity of four corner points for any particular x while searching and finally get the maximised area.

Overall Time Complexity: O(**T** * $\log_2$(maximum value of the coordinates)).

Problem J: Product Quality Analyst

Setter:

- Mohammad Ashraful Islam

Tester & Alterwriters:

- Md. Muhiminul Islam Osim

Problem type:

- Observation

Solution idea:

It's a nice observation that the gap between the times when two products are verified is always the same and the time gap is always the maximum value of t_i. We can calculate the time taken to verify the first product which is Summation of t_i and calculate the time for the other products using the time gap. So, the final time when all the products will be verified is ((Summation of t_i) + ((K - 1) * maximum value of t_i)).

Overall Time Complexity: O(**T** * **N**).