**1. Problem Discussion :**

You are given a number n, representing the number of people in a party. You are given n strings of n length containing 0's and 1's. If there is a '1' in ith row and jth spot, then person i knows about person j. A celebrity is defined as somebody who knows no other person but everybody else knows him. Please don't consider whether a person knows himself/herself or not! In reality, everyone should know who they are and what their purpose in life is. Print the indices of all the celebrities in the party, and there is no celebrity, then print "none". Note: There will be at least 2 people in the party, i.e. N >= 2. A party with only the party organizer does not make sense, isn't it? Example: Consider a party of 4 people: with the array of strings as: 0000 1011 1101 1110 In this scenario, the person with index 0 is the celebrity as everybody knows him but he does not know anybody else. Hence, the answer will be 0. I recommend you to watch the question video, where the problem statement is explained in detail.

**2. Approach :**

Deducing Algorithm:

First, let us try to analyze the number of celebrities that are possible. Q) Can there be more than 1 celebrity? R) No.There can be at maximum 1 celebrity only. We can prove this by contradiction. Let us assume that there are 2 celebrities i and j (i != j). Since, celebrity knows no one, hence mat[i][j] = 0 (celebrity i should not know j). But since, everyone should know the celebrity, hence mat[i][j] = 1 (celebrity j should be known by i). Thus, there is a contradiction, and only one of i or j can be a celebrity. Hence, we proved that there cannot be no more than 1 celebrity possible. Q) Can there be no (0) celebrity in the entire party? R) Yes It is possible that there is no celebrity at all. Consider this case of 4 people in the party: 0000 1011 0101 1110 There is no person who is known by everyone (except himself). Hence there is no celebrity, thus the answer will be "none" in this case. Until now, we figured out that there could be either 0 or 1 celebrity possible. Now, we should focus on deducing an algorithm to solve this problem.

Brute Force - O(n^2) Time & O(1) Space

A naive approach can be to consider every person as a potential celebrity, and check if it's entire row is filled with 0 (he knows nobody) and entire column if filled with 1 (everybody knows him). Please skip the arr[i][i] element (as we need not consider the case of a person knowing himself or not). As soon as we find any person, we return the element's index. If no celebrity is found, we return "none". Since we will be checking a row and a column for each of n people, the time complexity will be O(n \* (n + n)) = O(2 \* n^2) = O(n^2). And we are not using any extra space (matrix or the array of strings is the input), thus it is an O(1) space solution.

Optimal Solution - O(n) Time & O(n) Space

What we are about to do is similar to what we used to do to tackle those MCQ questions during our school! Maybe, this can bring some nostalgic moments in your head. And the approach is ELIMINATION APPROACH. Consider an element Arr[i][j] (i != j). There can be only two cases possible: Arr[i][j] = 1: It means person i knows the person j. If the person 'i' knows the person 'j', then i cannot be a celebrity, as there is a necessary condition that the celebrity should not know anyone else. Arr[i][j] = 0: It means person i does not know the person j. If the person 'i' don't the person 'j', then j cannot be a celebrity, as there is a necessary condition that the celebrity is known by everyone. Hence, picking any 2 candidates, we are able to eliminate the chances of one becoming a celebrity. So, we will start with a pool of all the candidates as possible celebrities, and one-by-one, we will keep on eliminating one person from this pool of possible candidates, and finally we will be left with one potential candidate. Q) Will the algorithm end here after getting the last candidate. Will it surely be a celebrity? R) No. There can be a case that no person in the entire party is a celebrity. Hence, after reducing to only 1 person as a potential candidate, we will check if this person is a celebrity or not (by traversing in it's row and it's column). If this last candidate is a celebrity, then we will print it's answer else there is no celebrity and hence print "none". Q) How will we create the pool of possible candidates and pick 2 candidates among them and then eliminate one? R) We can take the help of stack data structure. We will add all the candidates in the stack, then keep on popping 2 candidates from the stack, then eliminating one out of the two, and pushing the other candidate back into the stack, until there is only 1 element remaining in the stack/pool of candidates. Please refer to the solution video if you find difficulty in understanding the algorithm completely.

**3. Pseudo Code**

Create a stack and push all the indices (0 to n-1) in the stack. Keep on popping 2 elements from stack until there is only 1 element in stack: Let the two popped elements have index i and j. (i != j will always hold). If arr[i][j] is 1, then push back j into the stack (i cannot be celebrity, hence eliminated) Else (arr[i][j] is 0), then push back i into the stack (j cannot be celebrity, hence eliminated). At last, there will be one person remaining in the stack. Pop it out. Let the potential celebrity be pot. and check if it satisfies celebrity conditions. Check if the row = pot is filled with all 0s or not (except when col = pot). Check if the col = pot is filled with all 1s or not (except when row = pot). If the entire row is 0 and the entire col is 1 (except arr[pot][pot]), then pot is a celebrity, else there is no celebrity (print "none"). How about first trying by yourself without reading the code we provide?

**4. Code:**

import java.io.\*; import java.util.\*; public class Main { public static void main(String[] args) throws Exception { // write your code here BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); int n = Integer.parseInt(br.readLine()); int[][] arr = new int[n][n]; for (int j = 0; j < n; j++) { String line = br.readLine(); for (int k = 0; k < n; k++) { arr[j][k] = line.charAt(k) - '0'; } } findCelebrity(arr); } public static void findCelebrity(int[][] arr) { // if a celebrity is there print it's index (not position), // if there is not then print "none" Stack < Integer > st = new Stack < > (); for (int i = 0; i < arr.length; i++) { st.push(i); } while (st.size() > 1) { int i = st.pop(); int j = st.pop(); if (arr[i][j] == 1) { st.push(j); } else { st.push(i); } } int pot = st.pop(); boolean flag = true; for (int i = 0; i < arr.length; i++) { if (i != pot) { if (arr[i][pot] == 0 || arr[pot][i] == 1) { flag = false; break; } } } if (flag) { System.out.println(pot); } else { System.out.println("none"); } } }

**5. Analysis**

We are just applying an elimination approach, do it a try!

Time Complexity :

On Each iteration, we are popping two elements and pushing one element back. So, what we are doing overall is reducing the number of elements in stack (potential celebrities) by 1 each time. We will keep on reducing the size until it becomes 0 or 1. Hence the time complexity of this part of the algorithm is O(n). After we are left with a single potential celebrity candidate, we are checking whether it is actually a celebrity or not by checking it's row and column, hence time complexity will be O(n + n) = O(2 \* n). Thus, the overall time complexity will be O(3 \* n) = O(n) only.

SPACE COMPLEXITY :

We started with pushing all the elements into the stack. Hence we are using O(n) auxiliary space.

**6. Extra Gyaan (Knowledge):**

Instead of using stack space, we can also solve it using Two Pointers or Recursion. But the algorithm will remain the same, i.e. we will have to follow the elimination approach. It is just that two pointers will work on the matrix itself to eliminate the candidates and will be taking constant O(1) extra space . Consider the case when we are not given with the matrix (array of strings), instead given an API to check whether person i knows person j. In this case, the two-pointer approach (which we have not discussed) will not work (as it works on the matrix itself). Hence, only stack or recursive solutions will work, which will take O(n) space. I hope you enjoyed solving the problem with me. We will come with another problem 'Merge Overlapping Intervals' for you to solve, until then Happy Coding!