The Celebrity Problem - GeeksforGeeks

Source: https://www.geeksforgeeks.org/the-celebrity-problem/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund The Celebrity Problem Last Updated : 25 Aug, 2025 Given a square matrix mat[][] of size n x n , where mat[i][j] == 1 means person i knows person j, and mat[i][j] == 0 means person i does not know person j, find the celebrity person where, A celebrity is defined as someone who: Is known by everyone else Does not know anyone (except themselves) Return the index of the celebrity if one exists, otherwise return -1. Note : It is guaranteed that mat[i][i] == 1 for all i Examples: Input: mat[][] = [[1, 1, 0], [0, 1, 0], [0, 1, 1]] Output: 1 Explanation: 0th and 2nd person both know 1. Therefore, 1 is the celebrity. Input: mat[][] = [[1, 1], [1, 1]] Output: -1 Explanation: The two people at the party both know each other. None of them is a celebrity. Input: mat[][] = [[1]] Output: 0 Try it on GfG Practice Table of Content [Naive Approach] Using Adjacency List - O(n^2) Time and O(n) Space [Better Approach] Using Stack - O(n) Time and O(n) Space [Expected Approach] Using Two Pointers - O(n) Time and O(1) Space [Naive Approach] Using Adjacency List - O(n 2) Time and O(n) Space The idea is to model the problem as a directed graph where edges represent “knows” relationships. A celebrity, if present, will be the single vertex with indegree = n and outdegree = 1 , meaning they are known by everyone else but does not know anyone (except themselves). Step by Step Implementation: Create two arrays indegree and outdegree, to store the indegree and outdegree Run a nested loop, the outer loop from 0 to n and inner loop from 0 to n. For every pair i, j check if i knows j then increase the outdegree of i and indegree of j. For every pair i, j check if j knows i then increase the outdegree of j and indegree of i. Run a loop from 0 to n and find the id where the indegree is n and outdegree is 1. C++ #include <iostream> #include <vector> using namespace std ; int celebrity (vector < vector < int >> & mat) { int n = mat . size (); // indegree and outdegree array vector < int > indegree (n , 0), outdegree (n , 0); // query for all edges for (int i = 0 ; i < n ; i ++) { for (int j = 0 ; j < n ; j ++) { int x = mat [i][j]; // set the degrees outdegree [i] += x ; indegree [j] += x ; } } // find a person with indegree n-1 // and out degree 0 for (int i = 0 ; i < n ; i ++) if (indegree [i] == n && outdegree [i] == 1) return i ; return -1 ; } int main () { vector < vector < int > > mat = {{ 1 , 1 , 0 }, { 0 , 1 , 0 }, { 0 , 1 , 1 }}; cout << celebrity (mat); return 0 ; } Java class GfG { static int celebrity (int [][] mat) { int n = mat . length ; // indegree and outdegree array int [] indegree = new int [n] ; int [] outdegree = new int [n] ; // query for all edges for (int i = 0 ; i < n ; i ++) { for (int j = 0 ; j < n ; j ++) { int x = mat [i][j] ; // set the degrees outdegree [i] += x ; indegree [j] += x ; } } // find a person with indegree n-1 // and out degree 0 for (int i = 0 ; i < n ; i ++) if (indegree [i] == n && outdegree [i] == 1) return i ; return - 1 ; } public static void main (String [] args) { int [][] mat = { { 0 , 1 , 0 }, { 0 , 0 , 0 }, { 0 , 1 , 0 } }; System . out . println (celebrity (mat)); } } Python def celebrity (mat): n = len (mat) # indegree and outdegree array indegree = [0] * n outdegree = [0] * n # query for all edges for i in range (n): for j in range (n): x = mat [i][j] # set the degrees outdegree [i] += x indegree [j] += x # find a person with indegree n-1 # and out degree 0 for i in range (n): if indegree [i] == n and outdegree [i] == 1 : return i return - 1 if __name__ == "__main__" : mat = [[1 , 1 , 0], [0 , 1 , 0], [0 , 1 , 1]] print (celebrity (mat)) C# using System ; using System.Collections.Generic ; class GfG { static int celebrity (int [,] mat) { int n = mat . GetLength (0); // indegree and outdegree array int [] indegree = new int [n]; int [] outdegree = new int [n]; // query for all edges for (int i = 0 ; i < n ; i ++) { for (int j = 0 ; j < n ; j ++) { int x = mat [i , j]; // set the degrees outdegree [i] += x ; indegree [j] += x ; } } // find a person with indegree n-1 // and out degree 0 for (int i = 0 ; i < n ; i ++) if (indegree [i] == n && outdegree [i] == 1) return i ; return - 1 ; } static void Main () { int [,] mat = { { 1 , 1 , 0 }, { 0 , 1 , 0 }, { 0 , 1 , 1 } }; Console . WriteLine (celebrity (mat)); } } JavaScript function celebrity (mat) { let n = mat .

length ; // indegree and outdegree array let indegree = new Array (n). fill (0); let outdegree = new Array (n). fill (0); // query for all edges for (let i = 0 ; i < n ; i ++) { for (let j = 0 ; j < n ; j ++) { let x = mat [i][j]; // set the degrees outdegree [i] += x ; indegree [j] += x ; } } // find a person with indegree n-1 // and out degree 0 for (let i = 0 ; i < n ; i ++) if (indegree [i] === n && outdegree [i] === 1) return i ; return - 1 ; } // Driver Code let mat = [[1 , 1 , 0], [0 , 1 , 0], [0 , 1 , 1]]; console . log (celebrity (mat)); Output 1 [Better Approach] Using Stack - O(n) Time and O(n) Space The idea is to use a stack to eliminate non-celebrities by comparing pairs. If one person knows the other, the first is discarded; otherwise, the second is discarded. Repeat until one candidate remains, then verify if they meet the celebrity conditions. Step by Step Implementation: Create a stack and push all the ids in the stack. Run a loop while there are more than 1 element in the stack. Pop the top two elements from the stack (represent them as A and B) If A knows B, then A can't be a celebrity and push B in the stack. Else if A doesn't know B, then B can't be a celebrity push A in the stack. Assign the remaining element in the stack as the celebrity. Run a loop from 0 to n-1 and find the count of persons who knows the celebrity and the number of people whom the celebrity knows. => If the count of persons who knows the celebrity is n-1 and the count of people whom the celebrity knows is 0 then return the id of the celebrity else return -1. C++ #include <iostream> #include <vector> #include <stack> using namespace std ; int celebrity (vector < vector < int >> & mat) { int n = mat . size (); stack < int > st ; // push everybody in stack for (int i = 0 ; i < n ; i ++) st . push (i); // find a potential celebrity while (st . size () > 1) { int a = st . top (); st . pop (); int b = st . top (); st . pop (); // if a knows b if (mat [a][b]) { st . push (b); } else { st . push (a); } } // potential candidate of celebrity int c = st . top (); st . pop (); // Check if c is actually // a celebrity or not for (int i = 0 ; i < n ; i ++) { if (i == c) continue ; // if any person doesn't // know 'c' or 'c' doesn't // know any person, return -1 if (mat [c][i] || ! mat [i][c]) return -1 ; } return c ; } int main () { vector < vector < int > > mat = {{ 1 , 1 , 0 }, { 0 , 1 , 0 }, { 0 , 1 , 1 }}; cout << celebrity (mat); return 0 ; } Java import java.util.Stack ; class GfG { static int celebrity (int [][] mat) { int n = mat . length ; Stack < Integer > st = new Stack <> (); // push everybody in stack for (int i = 0 ; i < n ; i ++) st . push (i); // find a potential celebrity while (st . size () > 1) { int a = st . peek (); st . pop (); int b = st . peek (); st . pop (); // if a knows b if (mat [a][b] != 0) { st . push (b); } else { st . push (a); } } // potential candidate of celebrity int c = st . peek (); st . pop (); // check if c is actually // a celebrity or not for (int i = 0 ; i < n ; i ++) { if (i == c) continue ; // if any person doesn't // know 'c' or 'c' doesn't // know any person, return -1 if (mat [c][i] != 0 || mat [i][c] == 0) return - 1 ; } return c ; } public static void main (String [] args) { int [][] mat = { { 1 , 1 , 0 }, { 0 , 1 , 0 }, { 0 , 1 , 1 } }; System . out . println (celebrity (mat)); } } Python import sys def celebrity (mat): n = len (mat) st = [] # push everybody in stack for i in range (n): st . append (i) # find a potential celebrity while len (st) > 1 : a = st . pop () b = st . pop () # if a knows b if mat [a][b]: st . append (b) else : st . append (a) # potential candidate of celebrity c = st . pop () # check if c is actually # a celebrity or not for i in range (n): if i == c : continue # if any person doesn't # know 'c' or 'c' doesn't # know any person, return -1 if mat [c][i] or not mat [i][c]: return - 1 return c if __name__ == "__main__" : mat = [[1 , 1 , 0], [0 , 1 , 0], [0 , 1 , 1]] print (celebrity (mat)) C# using System ; using System.Collections.Generic ; class GfG { static int celebrity (int [,] mat) { int n = mat . GetLength (0); Stack < int > st = new Stack < int > (); // push everybody in stack for (int i = 0 ; i < n ; i ++) st . Push (i); // Find a potential celebrity while (st . Count > 1) { int a = st . Pop (); int b = st . Pop (); // if a knows b if (mat [a , b] != 0) { st . Push (b); } else { st . Push (a); } } // Potential candidate of celebrity int c = st . Pop (); // Check if c is actually // a celebrity or not for (int i = 0 ; i < n ; i ++) { if (i == c) continue ; // If any person doesn't // know 'c' or 'c' doesn't // know any person, return -1 if (mat [c , i] != 0 || mat [i , c] == 0) return - 1 ; } return c ; } static void Main () { int [,] mat = { { 0 , 1 , 0 }, { 0 , 0 , 0 }, { 0 , 1 , 0 } }; Console . WriteLine (celebrity (mat)); } } JavaScript function celebrity (mat) { let n = mat . length ; let st = []; // push everybody in stack for (let i = 0 ; i < n ; i ++) st . push (i); // find a potential celebrity while (st . length > 1) { let a = st . pop (); let b = st . pop (); // if a knows b if (mat [a][b] !== 0) { st . push (b); } else { st . push (a); } } // potential candidate of celebrity let c = st . pop (); // Check if c is actually // a celebrity or not for (let i = 0 ; i < n ; i ++) { if (i === c) continue ; // If any person doesn't // know 'c' or 'c' doesn't // know any person, return -1 if (mat [c][i] !== 0 || mat [i][c] === 0) return - 1 ; } return c ; } // Driver Code let mat = [[1 , 1 , 0], [0 , 1 , 0], [0 , 1 , 1]]; console . log (celebrity (mat)); Output 1 [Expected Approach] Using Two Pointers - O(n) Time and O(1) Space The idea is to use two pointers , one from start and one from the end. Assume the start person is A, and the end person is B. If A knows B, then A must not be the celebrity. Else, B must not be the celebrity. At the end of the loop, only one index will be left as a celebrity. Go through each person again and check whether this is the celebrity. Step by Step Implementation: Create two indices i and j, where i = 0 and j = n-1 Run a loop until i is less than j. Check if i knows j, then i can't be a celebrity. so increment i, i.e. i++ Else j cannot be

a celebrity, so decrement j, i.e. j-- Assign i as the celebrity candidate Now at last check whether the candidate is actually a celebrity by re-running a loop from 0 to n-1 and constantly checking if the candidate knows a person or if there is a candidate who does not know the candidate. Then we should return -1. else at the end of the loop, we can be sure that the candidate is actually a celebrity. C++ #include <iostream> #include <vector> using namespace std ; int celebrity (vector < vector < int >> & mat) { int n = mat . size (); int i = 0 , j = n - 1 ; while (i < j) { // j knows i, thus j can't be celebrity if (mat [j][i] == 1) j -- ; // else i can't be celebrity else i ++ ; } // i points to our celebrity candidate int c = i ; // check if c is actually // a celebrity or not for (i = 0 ; i < n ; i ++) { if (i == c) continue ; // if any person doesn't // know 'c' or 'c' doesn't // know any person, return -1 if (mat [c][i] || ! mat [i][c]) return -1 ; } return c ; } int main () { vector < vector < int > > mat = {{ 1 , 1 , 0 }, { 0 , 1 , 0 }, { 0 , 1 , 1 }}; cout << celebrity (mat); return 0 ; } Java class GfG { static int celebrity (int [][] mat) { int n = mat . length ; int i = 0 , j = n - 1 ; while (i < j) { // j knows i, thus j can't be celebrity if (mat [j][i] == 1) j -- ; // else i can't be celebrity else i ++ ; } // i points to our celebrity candidate int c = i ; // check if c is actually // a celebrity or not for (i = 0 ; i < n ; i ++) { if (i == c) continue ; // if any person doesn't // know 'c' or 'c' doesn't // know any person, return -1 if (mat [c][i] != 0 || mat [i][c] == 0) return - 1 ; } return c ; } public static void main (String [] args) { int [][] mat = { { 1 , 1 , 0 }, { 0 , 1 , 0 }, { 0 , 1 , 1 } }; System . out . println (celebrity (mat)); } } Python def celebrity (mat): n = len (mat) i = 0 j = n - 1 while i < j : # j knows i, thus j can't be celebrity if mat [j][i] == 1 : j -= 1 # else i can't be celebrity else : i += 1 # i points to our celebrity candidate c = i # check if c is actually # a celebrity or not for i in range (n): if i == c : continue # if any person doesn't # know 'c' or 'c' doesn't # know any person, return -1 if mat [c][i] or not mat [i][c]: return - 1 return c if __name__ == "__main__" : mat = [[1 , 1 , 0], [0 , 1 , 0], [0 , 1 , 1]] print (celebrity (mat)) C# using System ; using System.Collections.Generic ; class GfG { static int celebrity (int [,] mat) { int n = mat . GetLength (0); int i = 0 , j = n - 1 ; while (i < j) { // j knows i, thus j can't be celebrity if (mat [j , i] == 1) j -- ; // else i can't be celebrity else i ++ ; } // i points to our celebrity candidate int c = i ; // Check if c is actually // a celebrity or not for (i = 0 ; i < n ; i ++) { if (i == c) continue ; // If any person doesn't // know 'c' or 'c' doesn't // know any person, return -1 if (mat [c , i] != 0 || mat [i , c] == 0) return - 1 ; } return c ; } static void Main () { int [,] mat = { { 1 , 1 , 0 }, { 0 , 1 , 0 }, { 0 , 1 , 1 } }; Console . WriteLine (celebrity (mat)); } } JavaScript function celebrity (mat) { let n = mat . length ; let i = 0 , j = n - 1 ; while (i < j) { // j knows i, thus j can't be celebrity if (mat [j][i] === 1) j -- ; // else i can't be celebrity else i ++ ; } // i points to our celebrity candidate let c = i ; // Check if c is actually // a celebrity or not for (let i = 0 ; i < n ; i ++) { if (i === c) continue ; // If any person doesn't // know 'c' or 'c' doesn't // know any person, return -1 if (mat [c][i] || ! mat [i][c]) return - 1 ; } return c ; } // Driver Code let mat = [[1 , 1 , 0], [0 , 1 , 0], [0 , 1 , 1]]; console . log (celebrity (mat)); Output 1 Comment Article Tags: Article Tags: Stack Matrix DSA Microsoft Amazon Google Flipkart Fab.com Zoho United Health Group + 6 More