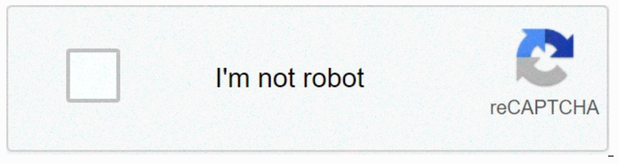

Continue

The celebrity problem leetcode

Suppose you are at a party with n people (marked from 0 to n - 1) and among them, there may be one celebrity. The definition of a celebrity is that everyone else n - 1people know him, but he/she knows nothing of them. Now you want to find out who the celebrity is or confirm that he's gone. The only thing you're allowed to do is ask questions like, Hi, A. Do you know B? to get information about whether A B. You need to find out a celebrity (or confirm that he is missing) by asking as few questions as possible (in asymptoptic). You get a helper function bool knows(s, b) that tells you if A knows B. Implement the function int findCelebrity(n), your function should reduce the number of calls you know. Keep it in mind: There will be exactly one celebrity if he's at the party. Bring back the celebrity label if there's a celebrity at the party. If there's no celebrity, go back to -1. Understand the problem: The problem can be transformed as a chart problem. We count the degree and beyond the degree for each person. Then find out a person with a grade n - 1 and beyond grade 0. Code (Java): /* Knows api is defined in parent class Relationship. boolean knows(int a, int b); */ public class Solution extends relationship { public int findCelebrity(int n) { if (n <= 1) { return -1; } int[] inDegree = new int[n]; int[] outDegree = new int[n]; for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { if (i != j & knows(i, j)) { outDegree[i]++; inDegree[j]+; } for (int i = 0; i < n; i++) { if (inDegree[i] == n - 1 & outDegree[i] == 0) { return and } Analysis: Time O(n^2). Space O(n). A O(n) time O(1) Space Solution: Use two pointers, left starts from 0, and right starts from n -1. Make sure he knows (left, right). If yes, left++. Second, right--. After the first step, we could find out a potential candidate. In the second step, we confirm the candidate by going through all person again. Each of them must know the candidate until the candidate is allowed to know anyone else. Code (Java): /* Knows api is defined in parent class Relationship. boolean knows(int a, int b); */ public class Solution extends Relationship { public int findCelebrity(int n) { if (n <= 1) { return -1; } int left = 0; int right = n - 1; // Step 1: Find a potential candidate until (left < right) { if (knows(left, right)) { left++; } other { right--; } } // Step 2: Confirm candidate int candidate = right; for (int i = 0; i < n; i++) { if (i != candidate & (!knows(i, candidate) || knows(candidate, i)) { return -1; } } Issue: 1:For each nod to track all incoming connections (in dictionary)Loop through dictionary and find nods that has n-1 linksAs there is more than one nod that has n-1 links return -1As there is no nod n-1 linkovi vraćaju -1 Else vraćaju sloceenost vremena: O (n2) Sloenost prostora: O (n) Vrijeme timeouta prilikom podnošenja leetcode. Kod: Pristup 2:Kredit: Leetcode forum za raspravu Odajte kroz 0 do n-1 kima kako biste pronašli slavnog kandidata koji moceda nikoga ne pozjena, u osnovi naprijed petlja Sve provjeriti ako kandidat celebrity zna bilo koju drugu osobu (unatrag pokazuju i) Konačno provjerite da li svistali kimovi znaju kandidat celebrityreturn node adko svi java vratiti -1Time sloceenost O(n) Slorainenost prostora O (1) Pretpostavimo da ste na zabavi s n ljudi (označen od 0 do n - 1) i među njima, postoji svibanj postojati jedna slavna osoba. Definicija slavne osobe je da ga svi ostali n - 1 ljudi poznaju, ali on/ona ne poznaje nijednu od njih. Sada hair (Ha) tko je slavna osoba ili potvrditi da ga nema. Jedino što vam je dopušteno je postavljati pitanja poput: Bok, A. Znate li B? kako biste dobili informaciju o tome zna li A B. Morate saznati slavnu osobu (ili potvrditi da ga nema) postavlja?i što manjean pitja (u asimtotiku). Vi ste dobili pomagač funkciju bool zna (a, b) koji vam govori da li A zna B. Implementirati funkciju int find Celebrity (n), vaa funkcija bi trebala smanjiti brojziva znati. Imajte na čemu: Bit će točno jedna slavna osoba ako je na zabavi. Vratite etiketu slavne osobe ako postoji slavna osoba na zabavi. Ako nema slavne osobe, vratite se -1. Rješenje first numbers individual N, from 1 to N. Start with the number 1 person and ask him if he knows the number 2 person. There are two kinds of situations: number 1 knows number 2: then number 1 must not be a celebrity, number 2 may be a celebrity. Number 1 does not know number 2: Number 2 must not be a celebrity, because a celebrity must have an N-1 person to know him. Based on the above results, it can be concluded: Number 1 knows number 2: then number 1 must not be a celebrity, number 2 may be a celebrity. Number 1 does not know number 2: Number 2 must not be a celebrity, number 1 may be a celebrity. Each person asked will eliminate one person, and then continue to ask the likely celebrity candidate and the next number of relationships, until the N individual can find the celebrity. The time complexity is O(N). 12345678910111213141516public int findCelebrity (int n) za (int i s 1; i slt; n; i s) s if (celebrity, i)) celebrity s i; s for (int i s 0; i s lt; n; i s) s if (i! s celebrity) s if (i! s celebrity, i) s! Reference Celebrity Problem Celebrity Problem Pretpostavimo da ste na zabavi snpeople (ožčnaeno od0ton - 1) i među njima, mocone postojati jedna slavna osoba. Definicija slavne osobe je da ga svi drugi - 1people znaju, ali on / ona ne zna ništa od njih. Sada hair (Ha) tko je slavna osoba ili potvrditi da ga nema. Jedino što vam je dopušteno je postavljati pitanja poput: Bok, A. Znaši B? da biste dobili about whether A B. You need to find out a celebrity (or confirm that he is gone) by asking as few questions as possible (in asymptoptic). You have got a functionbool helper knows(s, b)that tells you if A knows B. Implement a functional findCelebrity(n), your function should reduce the number of calls toknows. Keep it in mind: There will be exactly one celebrity if he's at the party. Bring back the celebrity label if there's a celebrity at the party. If there's no celebrity, come back-1. Although: O(3n): An explanation scy_usc here :P rva loop will stop at the candidate and who does not know anyone from i +1 to n-1. For any previous x<is it either know sb other or not known by sb other. The second loop will see if I know anyone from 0 to i-1. The third loop will check whether all party participants know x or not. O(2n): Explanation of czonzhu here The first pass is to select a candidate. If the candidate knows, then replace the candidate. The second pass is to check if the candidate is real. Solution(object) code class: def findCelebrity(self, n): :type n: int :rtype: int ix = 0 for and in range(n): if it knows(ix, i): ix = and if there is(ix, i) for and within range(ix)): return -1 if there is(ix) for and within range(n)): return -1 return ix In party N person is known to everyone only one person. Such a person may be present in the party, if yes, (s)does not know anyone in the party. We can only ask questions like does A B know? Find a stranger (celebrity) in a minimum number of questions. We can describe the problem entry as a series of numbers/characters represented by people in the party. We also have the hypothetical haveAcquaintance(A, B) function that comes back true if A knows B, falsely different. How can we solve the problem? Examples: INPUT: MATRIX = { {0, 0, 1, 0}, {0, 0, 1, 0}, {0, 0, 0}, {0, 0, 1, 0} } Output:id = 2 Explanation: Person with ID 2 knows no one, but everyone knows it Input: MATRIX = {0, 0, 1, 0}, {0, 0, 1, 0}, {0, 1, 0, 0}, {0, 0, 1, 0 } Output: No explanation We measure complexity in terms of calls made to HaveAcquaintance(). Method 1: This uses a chart to get to a specific solution. Approach: Use a chart to model your solution. Initialize the dedunies and overeat each vertex as 0. If A knows B, draw a directed edge from A to B, increase edegree B and outgrow A by 1. Construct all possible chart edges for each pair possible [i, j]. There are NC2 pairs. If a celebrity is present in the party, there will be one sink node in the chart with overtaken zeros and edegree from N-1. Algorithm: Create two fields that are edegree and outdegree, to store edegree and outdegree run nested loop, outer loop from 0 to n and inner loop from 0 to n. For every couple and, I check if I know j hen outdegree of i indegree of j Za svaki par 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-1 and outdegree is 0 Implementation: #include <bits tdc++.h>#include <list>using namespace std; #define N 8 bool MATRIX[N][N] = {{0, 0, 1, 0}, {0, 0, 1, 0}, {0, 0, 0, 0}, {0, 0, 1, 0}}; bool zna(int a, int b) { return MATRIX[a][b]; } int findCelebrity(int n) { int indegree[n]={0},outdegree[n]={0}; for(int i=0; i <n; i++)= {= for(int= j=0;><n;> <n; i++)= {= int= x=knows(i,j); outdegree[i]+=x; indegree[j]+=x; } = }= for(int= i=0;><n;> <n; i++)= if(indegree[i]== n-1= &= outdegree[i]== 0)= return= i;= return= -1;= }= int= main()= {= int= n=4; int= id=findCelebrity(n); id== -1= cout=></n;> < no= celebrity= := cout=> < celebrity= id= => < id;= return= 0;= }= celebrity= id= 2== = complexity= analysis: = time= complexity:= o(n2). a= nested= loop= is= run= traversing= the= array,= so= the= time= complexity= is= o(n2)= space= complexity:= o(n). since= extra= space= of= size= n= is= required.= approach= : the= problem= can= be= decomposed= into= a= combination= of= smaller= instances.= say,= if= the= celebrity= of= n-1= persons= is= known,= can= the= solution= to= n?= there= are= two= possibilities,= celebrity(n-1)= may= know= n,= or= n= already= knew= celebrity(n-1).= in= the= former= case,= n= will= be= a= celebrity= if= n.= in= the= latter= case,= check= that= celebrity(n-1)= doesn't= know= n. the= above-mentioned= approach= use= recursion= to= find= the= celebrity= among= n= persons.= it= is= necessary= to= find= a= celebrity= among= n-1= persons.= so= while= calculating= the= celebrity= of= i= persons= the= function= calls= itself= to= find= the= celebrity= of= i-1= persons= and= continues= doing= so= until= the= base= case= is= found.= algorithm= : = create= a= recursive= function= that= takes= an= integer= n.= check= the= base= case,= if= the= value= of= n= is= 0= then= return= 0.= call= the= recursive= function= and= get= the= id= of= celebrity= from= the= first= n-1= elements.= if= the= id= is= -1= then= assign= n= as= celebrity= and= return= the= value.= if= the= celebrity= of= first= n-1= elements= knows= n-1= and= n-1= does= not= know= the= celebrity= then= return= n-1,= (0= based= indexing)= if= the= celebrity= of= first= n-1= elements= does= not= know= n-1= and= n-1= knows= the= celebrity= then= return= id= of= celebrity= of= n-1= elements,= (0= based= indexing)= else= return= -1= create= a= wrapper= function= and= check= whether= the= id= returned= by= the= function= is= really= the= celebrity= or= not.= implementation : = #include=> <bits tdc++.h=> #include <list>koristeći namespace std; #define N 8 bool MATRIX[N][N] = {{0, 0, 1, 0},</list> </bits> </list> 0, 1, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}; bool zna(int a, int b) { return MATRIX[a][b]; } int findCelebrity(int n) { if(n == 1) return n - 1; int id = findCelebrity(n-1); ako(id == -1) povratak n-1; drugo ako(zna(id, n-1) & !zna(n-1, id)) { povratak n-1; } ostalo ako(zna(n-1, id) & !knows(id, n-1)) { povratni id; } povratak -1; } int Celebrity(int n) { int id = findCelebrity(n); ako(id == -1) povratak id; ostalo { int c1=0, c2=0; for(int i=0; i <n; i++)= if(i= !=id) {= c1+=knows(id, i);= c2+=knows(i, id);= }= if(c1==0 &= c2==n-1) return= id;= return= -1;= }= }= int= main()= {= int= n=4; int= id=Celebrity(n); id== -1= cout=></n;> < no= celebrity= := cout=> < celebrity= id= => < id;= return= 0;= }= celebrity= id= 2= = complexity= analysis: = time= complexity:= o(n). the= recursive= function= is= called= n= times,= so= the= time= complexity= is= o(n).= space= complexity:= o(1). as= no= extra= space= is= required.= approach:= there= are= some= observations= based= on= elimination= technique= (refer= polya's= how= to= solve= it= book). = if= a= knows= b,= then= a= can't= be= a= celebrity.= discard= a,= and= b= may= be= celebrity.= if= a= doesn't= know= b,= then= b= can't= be= a= celebrity.= discard= b,= and= a= may= be= celebrity.= repeat= above= two= steps= till= there= is= only= one= person.= ensure= the= remained= person= is= a= celebrity.= (what= is= the= need= of= this= step?) = algorithm: = create= a= stack= and= push= all= the= id's= in= the= stack.= run= a= loop= while= there= are= more= than= 1= element= in= the= stack.= pop= top= two= element= from= the= stack= (represent= them= as= a= and= b)= check= if= a= knows= b,= then= a= can't= be= a= celebrity= and= push= b= in= stack.= check= if= a= doesn't= know= b,= then= b= can't= be= a= celebrity= push= a= in= 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= 0= then= return= the= id= of= celebrity= else= return= -1.= implementation : = #include=> <bits tdc++.h=> #include <list>koristeći namespace std; #define N 8 bool MATRIX[N][N] = {{0, 0, 1, 0}, {0, 0, 1, 0}, {0, 0, 0}, {0, 0, 1, 0}}; bool zna(int a, int b) { return MATRIX[a][b]; } int findCelebrity(int n) { stack<int> s; int C; </int></list></bits>(int i = 0; i < n;= i++)= s.push(i);= int= a=s.top(); s.pop();= int= b=s.top(); s.pop();= while= (s.size()=> 1) { ako (zna(A, B)) { A = s.top(); s.pop(); } ostalo { B = s.top(); s.pop(); } ako(s.empty()) povratak -1; C = s.top(); s.pop(); ako (zna(C, B)) C = B; ako (zna(C, A)) C = A; za (int i = 0; i < n;= i++)= {= if= (= (i= !=C) &= (knows(c,= i)= || = !knows(i,= c))=)= return= -1;= }= return= c;= }= int= main()= {= int= n=4; int= id=findCelebrity(n); id== -1= cout=> < no= celebrity= := cout=> < celebrity= id= => < id;= return= 0;= }= import= java.util.stack;= class= gfg = {= static= int= matrix[][]={ { = 0,= 0,= 1,= 0= },= { = 0,= 0,= 1,= 0= },= { = 0,= 0,= 0,= 0= }, = { = 0,= 0,= 1,= 0= }= };= static= boolean= knows(int= a,= int= b) = {= boolean= res=(MATRIX[a][b] ==1) true= : = false;= return= res;= }= static= int= findcelebrity(int= n) = {=> <Integer>st = novi St <> (); int c; za (int i = 0; i < n;= i++) = {= st.push(i);= }= while= (st.size()=> 1) { int a = st.pop(); int b = st.pop(); ako (zna(a, b)) { st.push(b); } drugi st.push(a); } ako(st.empty()) povratak -1; c = st.pop(); za (int i = < n; i++) { if (i != c & (knows(c, i) || !knows(i, c))) return -1; } return c; } public static void main(String[] args) { int n = 4; int result = findCelebrity(n); if (result == -1) { n;= i++) = {= if= (i= !=c &= (knows(c,= i)= || = !knows(i,= c))= return= -1;= }= return= c;= }= public= static= void= main(string[]= args) = {= int= n=4; int= result=findCelebrity(n); if= (result== -1) = {=></ n; i++) { if (i != c & (knows(c, i) || !knows(i, c))) return -1; } return c; } public static void main(String[] args) { int n = 4; int result = findCelebrity(n); if (result == -1) { System.out.println(No Celebrity); } else System.out.println(Celebrity ID + result); } } Celebrity ID Complexity Analysis 2: Weather Complexity: O(n). The total number of comparisons is 3(N-1), so the temporal complexity is O(n). Space complexity: O(n). n additional space is required to store the stack. Approach: The idea is to use two pointers, one from the beginning and one from the end. Let's assume that the initial person is A, and the ultimate person is B. Otherwise, B can't be a celebrity. At the end of the loop, only one index will remain as a celebrity. Walk through each person and make sure it's a celebrity. Access Two pointers can be used where two pointers can be signed, one at the beginning and the other at the end and the elements can be compared and the search space can be reduced. Algorithm : Create two indices a and b, where = 0 and b = n-1 Start the loop while it is less than b. Check if he knows b, then he can't be a celebrity. so increment a, or a++ Else b can not be a celebrity, so decrement b, or b – Assign as a celebrity with a number of people who know a celebrity and the number of people a celebrity knows. If the number of people who know a celebrity is n-1 and the number of people a celebrity knows to be 0, return the celebrity id to -1. Implementation. #include <bits tdc++.h=>using namespace std; #define N 8 bool MATRIX[N][N] = {{0, 0, 1, 0}, {0, 0, 1, 0}, {0, 0, 0}, {0, 0, 1, 0} }; bool zna(int a, int b) { return MATRIX[a][b]; } int findCelebrity(int n) { int a = 0; int b = n - 1; dok < b)= {= if= (knows(a,= b))= a++;= else= b--;= }= for= (int= i=0; i=> < n;= i++)= {= if= (= (i= !=a) &= (knows(a,= i)= || = !knows(i,= a))=)= return= -1;= }= return= a;= }= int= main()= {= int= n=4; int= id=findCelebrity(n); id== -1= cout=> < no= celebrity= := cout=> < celebrity= id= => (a < id; return 0; } class GFG { static int MATRIX[][] = { { 0, 0, 1, 0 }, { 0, 0, 1, 0 }, { 0, 0, 0, 0 }, { 0, 0, 1, 0 } }; static boolean knows(int a, int b) { id;= return= 0;= }= class= gfg = {= static= int= matrix[][]={ { 0,= 0,= 1 ,= 0= },= { = 0,= 0,= 0,= 1,= 0= }, = 0,= 0,= 0,= 0= },= { = 0,= 0,= 1,= 0= }= };= static= boolean= knows(int= a,= int= b) = {=></ id; return 0; } class GFG { static int MATRIX[][] = { { 0, 0, 1, 0 }, { 0, 0, 1, 0 }, { 0, 0, 0, 0 }, { 0, 0, 1, 0 } }; static boolean knows(int a, int b) { > </bits> </bits> res = (MATRIX[a][b] == 1) ? truth: false; restitution; } static int findCelebrity(int n) { int a = 0; int b = n - 1; while (< b) { if (knows, b)) a++; other b--; } for (int i = 0; and < n; i++) { if (i != a & (knows,s, a))) return -1; } return a; } Main public static gap (String[] args) { int n = 4; int result = findCelebrity(n); if (result == -1) { System.out.println(No Celebrity); } else System.out.println(Celebrity ID + result); } using the system; class GFG { static int [,]MATRIX = {{ 0, 0, 1, 0 }, { 0, 0, 1, 0 }, { 0, 0, 0, 0 }, { 0, 0, 1, 0 }}; static bool knows (int a, int b) { bool res = (MATRIX[a, b] == 1) ? truth: false; restitution; } static int findCelebrity(int n) { int a = 0; int b = n - 1; while (< b) { if (knows, b)) a++; other b--; } for (int i = 0; and < n; i++) { if (i != a & (knows,s, i) || !knows(s, a))) return -1; } return a; } Public Static Gap Main() { int n = 4; int result = findCelebrity(n); if (result == -1) { Console.WriteLine(No Celebrity); } other Console.WriteLine(Celebrity ID + } <?php $MATRIX = array (array(0, 0, 1, 0), field(0, 0, 1, 0), field (0, 0, 0, 0), field(0, 0, 1, 0)); function knows($a, $b) { global $MATRIX; return $MATRIX[$a][$b]; } findCelebrity($n) { $a = 0; $b = $n - 1; while ($a < $b) { if ($a, $b)) $a++; otherwise $b--; } for ($i = 0; $i < $n; $i++) { if ($i != $a) and ($a, $i) || !knows($i, $a))) return -1; } return $a; } $n = 4; $id = findCelebrity ($n); if($id == -1) echo no celebrity ; the second echo of Celebrity ID, $id; ? > Celebrity ID 2 Complexity Analysis: Complexity of Time: O(n). The total number of comparisons is 2(N-1), so the temporal complexity is O(n). Space complexity : O(1). No extra space required. Write the code to find a celebrity. Do not use any data structures such as charts, stack, etc... you only have access to N and HaveAcquaintance(int, int). Deploy the algorithm using queues. What's your observation? Compare your solution with finding the maximum and minimum in the Tree series and tournament. What are the minimum number of comparisons we need (optimal number of haveAcquaintance())) calls? Attention reader! Don't stop studying now. Get hold of all the important DSA concepts with the DSA Self Paced course at a student-friendly price and you'll become industry-ready. Recommended posts:Improved By: vt_m, andrew1234, rohitkumar52, parthpatel16 parthpatel16