Groups of Friends

View Try Insights

Recommended Time: 36 mins Points: 75 10 test cases (2 samples)

Skills: Problem Solving (Intermediate)

Coding Data Structures Graphs MEDIUM Algorithms Depth First Search Problem Solving

Friendship can be represented as a matrix of 0's and 1's, with 1 indicating that two people are friends with each other. For example, the friendship list for person *0* with respect to persons *0* through *5* might be described as *101100*. This means that person *0* is friends with *0, 2* and *3 (everybody should be friends with themselves!)*. For this problem we will assume that friendship is transitive. In other words, if person *1* is friends with person *3* and person *3* is friends with person 7, then person *1* is also considered to be friends with 7 through person 3. A *clique* is composed of all of the people who are friends with one another, whether directly or transitively.

Example

Consider the following direct friendship matrix:

	0	1	2
0	1	1	0
1	1	1	0
2	0	0	1

Persons *0* and *1* are direct friends, while person *2* is not. There are *2* friendship cliques {0,1} and {2}.

Note: The method signatures may vary slightly depending on the requirements of the chosen language. For example, C language will have an argument that represents the number of rows and columns in the matrix. Also, Java will receive a list rather than an array.

Function Description

Complete the function *countCliques* in the editor below.

countCliques has the following parameter(s):

friends is an array of strings, where each string consists of characters 0 and 1. The string *friends[i]* represents the friendship of person i with respect to others

Returns:

int: an integer that represents the number of friendship cliques

Constraints

- $1 \le n \le 300$
- $|friends| = n$
- Each *friends[i]* contains a binary string of *n* zeros and ones. *friends* is a square matrix.

Binary Tree Search

View Try Insights

Recommended Time: 13 mins Points: 75 14 test cases (3 samples)

Skills: Problem Solving (Intermediate)

Coding Data Structures Binary Trees EASY Algorithms Problem Solving Interviewer Guidelines

In a binary search tree, each node holds a *value* and a reference to as many as *2 child* nodes, or *children*. The *root* node has no ancestors. The children are called *left* and *right,* and *subtrees* rooted at *left* and *right* are the left and right subtrees. If each node is considered the root of a subtree, each node value in its left subtree must be less than its own value. Likewise, each node in its right subtree must have a greater or equal value to the root. This allows for efficient searching.

For each value in a list of integers, determine if it is present in a tree . If it is, return the integer *1,* otherwise, return *0.*

Function Description

Complete the function *isPresent* in the editor below.

isPresent has the following parameter(s):

BSTreeNode root: reference to the root node of a tree of integers

int val[q] : an array of integer items to search for

Returns:

int[q]: an integer array where each value at index *i* denotes whether *val[i]* is found in the BST or not

Constraints

- $1 \le n, q \le 10^5$
- $1 \le val[i] \le 5 \times 10^4$

What's the order?

View Try Insights

Recommended Time: 40 mins Points: 75 12 test cases (2 samples)

Coding Problem Solving Arrays

A number of teams have been competing with each other for the past few years, and you have been keeping track of their yearly rankings in the competition. Unfortunately, you've misplaced last year's ranking, and you are trying to think of a way to reconstruct last year's ranking based on this year's ranking. That might be possible because you've also kept track of which teams improved their ranking from the previous year, and which ones didn't. Now you need to write a program that is able to calculate whether last year's ranking can be reconstructed from the information you have or not. And if it is possible, your program should return last year's ranking.

Function Description

Complete the function *lastYearRanking* in the editor below.

lastYearRanking has the following parameter(s):

integer array currentRanking: this is a list of integers containing this year's ranking, from first to last, with teams identified by numbers between 1 and |currentRanking|.

integer array better: the list of teams (in no particular order) that went up the ranking from last year to this year.

integer array worse: the list of teams (in no particular order) that went down the ranking from last year to this year. The lists better and worse will of course be disjoint (no common elements)

Returns:

integer array: return last year's ranking if it's possible to infer that ranking from the information provided. If it is not possible to infer last year's ranking from the information provided, the lastYearRanking function should return an array containing only the number 0, i.e. [0].

Constraints

- $1 \leq$ n_teams ≤ 100

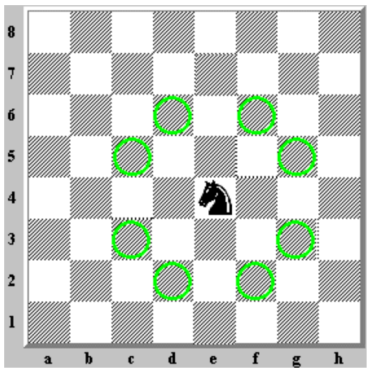
Runner Pawn

View Try Insights

Recommended Time: 25 mins Points: 75 12 test cases (2 samples)

Coding Problem Solving Data Search

In a chess board, given the start positions of a **black Knight** and a **white Pawn**, determine whether the black knight can prevent the white pawn from promoting to a Queen (reaching last rank). Assume that the white Pawn will move up to board one square at each round, while the Knight can move around as Knights usually do (see picture below).



Note that if it is the Pawn's turn to move but the Knight is blocking the Pawn then the game ends in a draw (stalemate). In all test cases you can assume that the game starts with White playing (Pawn) and the white Pawn will start in any file and any rank other than the last rank (1-7).

Chess Notation. In Chess the rows numbered 1 - 8 are called ranks, and row 8 is the last rank. The columns are called files. The algebraic notation for chess uses a combination of the letters "abcdefgh" and the numbers "12345678" to specify the location of a piece.

8	a8	b8	c8	d8	e8	f8	g8	h8
7	a7	b7	c7	d7	e7	f7	g7	h7
6	a6	b6	c6	d6	e6	f6	g6	h6
5	a5	b5	c5	d5	e5	f5	g5	h5
4	a4	b4	c4	d4	e4	f4	g4	h4
3	a3	b3	c3	d3	e3	f3	g3	h3
2	a2	b2	c2	d2	e2	f2	g2	h2
1	a1	b1	c1	d1	e1	f1	g1	h1
	a	b	c	d	e	f	g	h

For instance, the knight in the first picture above is positioned at **e4** and from there it can jump to any of the following positions **e6**, **f6**, **g5**, **g3**, **f2**, **d2**, **c3** or **c5**. A white Pawn will move up the board. For instance, a pawn starting at **g2** will move to **g3** in the next round and then **g4** and so on. In an actual game of chess a while pawn will start in the second rank and can move two places on the first move, but you do not apply this rule here. A pawn in our case can start at any file or rank (other than the last rank) and will always move up one rank at each round.

Function Description

Complete the function *knightAndPawn* in the editor below.

knightAndPawn has the following parameter(s):

string knight: a string containing the starting position of the knight using algebraic Chess notation
string pawn: a string containing the starting position of the pawn using algebraic Chess notation

Returns:

string: The function should return one of the following four messages (**the function must return the string, not print it!**):

"Pawn can promote without being captured"
"Knight can capture pawn before promotion"
"Knight can capture pawn immediately after promotion"
"Knight can block pawn before promotion (stalemate)"

Note: *In case it's possible to the Knight to either capture the pawn early or capture immediately after promotion, you should prefer to capture it early (before promotion).*

Quickest goal

View Try Insights

Recommended Time: 20 mins Points: 75 12 test cases (2 samples)

Coding

The local football team has hired you as their new coach. You want to solve the following problem. Given a placement of your players on the field, you want to figure out what is the least time for them to have an attempt on goal in a single play. The play must start from your goalie and the players can only pass the ball, without carrying it, or make an attempt on goal. You can assume no obstacles, so any of your players can pass to any other player or have a shot on goal.

The number of your players on the field is a variable n and can vary from 3 to 100. Each placement of your players on the field is a sequence of coordinates (x,y), one for each player, where x and y are measured in meters. By convention, the coordinates of your goalie are (0,0) and those of the opponent goal are (100,0) (which we assume is a single point). Each of your players has also an assigned initial passing speed and an initial shooting speed, both measured in m/s (meters/sec). A ball travelling on the pitch with initial velocity V_0 can cover a maximum distance of:

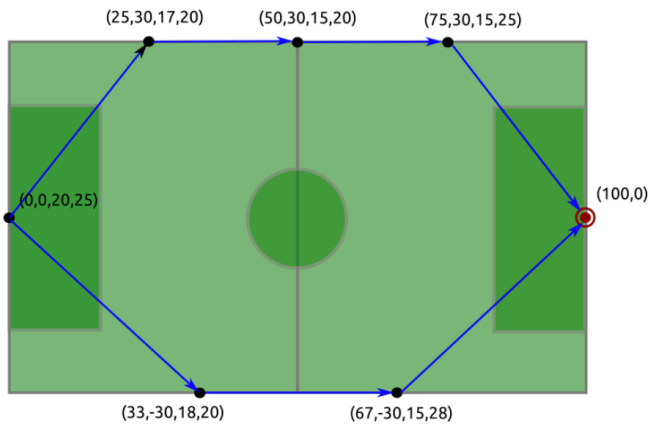
$$d_{max} = \frac{V_0^2}{2k}$$

where k is a friction coefficient equal to 4 m/s^2. For any distance d that is between 0 and d_{max}, the time t for the ball to cover d is given by the formula:

$$t = \frac{V_0 - \sqrt{V_0^2 - 2kd}}{k}$$

Example

Here is a player configuration with n = 6 players:



For each player we have a tuple (x,y,Vp_0,Vs_0), where (x,y) are the player's coordinates, Vp_0 their initial passing speed, and Vs_0 their initial shooting speed. E.g. your goalie has an initial passing speed of 20 m/s, and an initial shooting speed of 25 m/s. Let us consider two simple plays:

1. The goalie passes to the leftmost player on the top sideline (in approximately 2.66s), who passes it on to the middle top player (approx. 1.89s), who passes it on to the top right player (approx. 2.5s), who then shoots on goal (approx. 1.83s). The total time for this play is approximately 8.88s.
2. The goalie passes to the leftmost player on the bottom sideline (in approx. 3.36s), who passes it on the bottom right player (approx. 2.70s), who then shoots on goal (approx. 1.83s). The total time for this play is approximately 7.89s.

Given the distances between players, the two plays above are the best candidates for the quickest attempt on goal. That is because:

- If the goalie tried to make a direct attempt on goal, the ball would need to travel 100m with an initial speed of 25 m/s. However, with that initial speed, the ball can travel at most 78.125m.
- Each player at the sidelines can only pass to the player on its left/right (if there is one), and only the players at the right end can make an attempt on goal.

Therefore, the play in case 2 above yields the quickest attempt on goal.

Function description

You are asked to implement a function *quickestGoal* which takes as input a 2D integer array *players* with rows of the form [x,y,Vp_0,Vs_0], representing your players on the field. The goalie is always given in the first row.
Your function should return the least time for your players to have an attempt on goal **rounded down to the previous integer** (e.g. 23.3, 23.0, 23.6 all become 23).

An attempt on goal counts only if the ball can reach the opponent goal. If an attempt on goal is not possible in a play, your function should return -1.

Note: The function signature may vary slightly depending on the requirements of the chosen language. For example, C language will have an argument that represents the number of rows and columns in the matrix. Also, Java will receive a list of lists rather than a matrix.

Engrenage

View Try Insights

Recommended Time: 25 mins Points: 75 12 test cases (2 samples)

Coding Problem Solving Discreet Math

A cogwheel turning clockwise will turn the next cogwheel in the mechanism anti-clockwise, and vice versa. Given a sequence of cogwheels where the last cogwheel has an arrow pointing right (0 degrees), **determine the angle of that arrow** (rounded to the nearest integer between 0 and 365) **once the first cogwheel makes a full clockwise turn**. The mechanism is described by a list of numbers, where each number specifies the size of that cogwheel (number of teeth).

For instance, if we have a mechanism with three cogwheels of sizes 7, 8 and 6

then once the first cogwheel makes a full turn, the arrow in the last cogwheel will be 320 degrees from the starting position

You should always measure angles going counter-clockwise from the starting arrow position, like in the example above.

Function Description

Complete the function *lastWheelAngle* in the editor below.

lastWheelAngle has the following parameter(s):

integer array wheels: an array listing the number of teeth in each of the cogwheels of the mechanism, from the first to the last.

Returns:

integer: the final angle of the arrow on the last cogwheel (rounded to the nearest integer between 0 and 365) once the first cogwheel completes one full turn.

Constraints

- The mechanism will have between 1 and 100 cogwheels and each cogwheel will have between 5 and 20 teeth.

DNA Sequences

View Try Insights

Recommended Time: 30 mins Points: 75 10 test cases (2 samples)

Coding Algorithms Problem Solving

Given a list of DNA fragments (strings containing the letters A, C, G and T), put these fragments together in a single sequence such that the final sequence is the lexicographically smallest possible. For example, 'A' < 'G', 'CG' < 'CT' and 'GT' < 'GTA'.

Example
DNA fragments = ['A','CT','AG','GT']

Return 'AAGCTGT'.

Example
DNA fragments = ['G','GG','GGA','GGGC']

Return 'GGAGGGCGGG'.

Function Description
Complete the function *smallestDNA* in the editor below.

smallestDNA has the following parameter(s):
string fragments[n]: an array of DNA fragments

Returns:
string: the lexicographically smallest DNA that can be formed by concatenating the given DNA fragments

Constraints

- $1 \le n \le 3500$
- $1 \le$ length of *fragments[i]* ≤ 1000
- Each *fragments[i]* is composed of upper the cases letters A, C, G and T

Staring Birds

View Try Insights

Recommended Time: 15 mins Points: 75 14 test cases (2 samples)

Coding Problem Solving Lists

Several birds are sitting on a fence. They are each looking left or right. Each bird counts how many other birds are looking in their direction. What is the total number of birds that they count all together.

Function Description

Complete the function *birdCount* in the editor below.

birdCount has the following parameter(s):

> *string* birds: an array of characters (either > or <) describing the direction that each bird is looking. The character < indicates that the bird is looking left, whereas > indicates that the bird is looking right.

Returns:

> *integer*: the total number of birds that they count all together.

Constraints

- $1 \le |birds| \le 10^3$, where |s| denotes the length of the input string birds

Ideal Numbers

View Try Insights

Recommended Time: 20 mins Points: 75 10 test cases (2 samples)

Coding Problem Solving Discreet Math

Let us call a number "ideal" if the sum of its (strictly smaller) divisors is equal to the number itself. For instance, the strictly smaller divisors of 6 are 1, 2 and 3, and indeed 1 + 2 + 3 = 6, so 6 is an "ideal" number. If that sum is less than the number we will call the number "under-ideal", and if it is more than the number, we will call it "over-ideal". Write a program that will classify a given number as "ideal", "under-ideal" or "over-ideal".

Function Description

Complete the function *classify* in the editor below.

classify has the following parameter(s):
integer n: the number that you need to "classify" according to the description above.

Returns:
string: return the classification of the number n, i.e. the classify function should return one of the following strings: "ideal", "under-ideal" or "over-ideal".

Constraints

- $2 \le n \le 10^5$