Compete

Marathon Match 121

Challenge TCO21 Other Marathon Match Recommended THRIVE Articles

Key Information

1st	2nd
$0	$0

Unregister

Submit

Next Deadline: Submission | 6d 4h until current deadline ends

Show Deadlines

DETAILS REGISTRANTS (106) SUBMISSIONS (31) MY SUBMISSIONS (0) CHALLENGE FORUM

Challenge Overview

Important Links

- Submission-Review You can find your submissions *artifacts* here. Artifacts will contain output.txt's for both example test cases and provisional test cases with stdout/stderr and individual test case scores.
- Other Details For other details like Processing Server Specifications, Submission Queue, Example Test Cases Execution.

Overview

N teams play in a round-robin soccer tournament where each team plays every other team once. Each game has three possible outcomes: win for first team, win for second team or a draw. If a team wins the game then it receives **W** points. If the game is a draw then both teams receive **D** points. The losing team receives no points. Your task is to reconstruct the result of each game given the total number of goals scored, goals and points earned by each team. You will receive 1 point for predicting a game's outcome (win, loss, draw)

Support

correctly. You will receive an additional 2 points for also predicting the game's exact score.

Here is an example solution for seed=1 with N=6, W=3, D=1 and X=1. This solution predicts that each game ends in a 1:1 draw. Five of the games finished in a draw and two of them were exactly 1:1, so the solution

achieves 5*1 + 2*2 = 9 points.

INPUT:
team 0 scored 5 conceded 10 points 2
team 1 scored 4 conceded 12 points 1
team 2 scored 6 conceded 4 points 10
team 3 scored 9 conceded 4 points 9
team 4 scored 4 conceded 5 points 5
team 5 scored 11 conceded 4 points 13

OUTPUT:
game 0, team 0 vs team 1, predicted 1:1 actual 2:2 Correct outcome!
game 1, team 0 vs team 2, predicted 1:1 actual 1:2 Wrong
game 2, team 0 vs team 3, predicted 1:1 actual 1:3 Wrong
game 3, team 0 vs team 4, predicted 1:1 actual 1:1 Correct outcome and score!
game 4, team 0 vs team 5, predicted 1:1 actual 0:2 Wrong
game 5, team 1 vs team 2, predicted 1:1 actual 1:2 Wrong
game 6, team 1 vs team 3, predicted 1:1 actual 0:3 Wrong
game 7, team 1 vs team 4, predicted 1:1 actual 0:2 Wrong
game 8, team 1 vs team 5, predicted 1:1 actual 1:3 Wrong
game 9, team 2 vs team 3, predicted 1:1 actual 0:0 Correct outcome!
game 10, team 2 vs team 4, predicted 1:1 actual 1:0 Wrong
game 11, team 2 vs team 5, predicted 1:1 actual 1:2 Wrong
game 12, team 3 vs team 4, predicted 1:1 actual 1:1 Correct outcome and score!
game 13, team 3 vs team 5, predicted 1:1 actual 2:2 Correct outcome!
game 14, team 4 vs team 5, predicted 1:1 actual 0:2 Wrong

Simulation

The following are details on how the game outcomes are generated. In the beginning of the tournament, each team is assigned two values: an attack strength and a defence strength. These values do not change throughout the tournament. Each game is generated using **X** independent simulations. Each simulation consists of 6 rounds: 3 rounds when the first team is attacking and 3 rounds when the second team is attacking. One round of team 1 attacking team 2 is simulated like so:

- A1 is chosen uniformly at random between 1 and team 1's attack strength, inclusive.

- D2 is chosen uniformly at random between 1 and team 2's defence strength, inclusive.
- If A1 > D2 then team 1 scores one goal.

Similarly we simulate team 2 attacking team 1. Since there are 3 attacking rounds for each team, the most number of goals a team can score is 3. The final game score is the rounded average of all **X** simulation scores.

For example, suppose X=3 and we generated the following scores: 3:2, 3:0 and 2:2. The first team scored on average (3+3+2)/3 = 2.67 goals, while the second team scored on average (2+0+2)/3 = 1.33 goals. After the rounding the final game score will be recorded as 3:1. We use standard rounding, where 0.5 rounds up. Note that the final game score could be different to all the simulation scores, as it is the case here.

The game outcome is decided based on the game scores. The winning team is the one who has scored more goals. If both teams scored the same number of goals then the game is a draw.

Input

Your code will receive as input the following values, each on a separate line:

- **N**, the number of teams.
- **W**, the number of points awarded for winning a game.
- **D**, the number of points awarded for drawing a game.
- **X**, the number of simulations per game.
- **N** lines, where the i-th line contains the final result of the i-th team formatted as "goals_scored goals_conceded points_earned" (without the quotes).

Output

Your code should write to output the following:

- On the first line, K=**N***(**N**-1)/2, the total number of games played.
- K lines, where the i-th line contains your predicted result for the i-th game. Each line should be formatted as "scored1 scored2" (without the quotes), where scored1 is the number of goals scored by the first team and scored2 is the number of goals scored by the second team. The order of the games is given by this pseudo-code:

```
i=0
for team1 = 0 to N-1
  for team2 = team1+1 to N-1
    //i-th game is between team1 and team2
    i++
```

Scoring

The scorer will compare your predictions to the actual game results. You will receive 1 point for each correctly predicted game outcome and an extra 2 points if you also correctly predicted its score.

If your return was invalid, then your raw score on this test case will be -1. Possible reasons include:

- Not returning exactly K=**N***(**N**-1)/2 game outcomes.
- Incorrectly formatted game outcomes.

If your raw score for a test case is negative then your normalized score for that test case is 0. Otherwise, your normalized score for each test case is YOUR/MAX, where YOUR is your raw score and MAX is the largest positive raw score currently obtained on this test case (considering only the last submission from each competitor). Finally, the sum of all your test scores is normalized to 100.

Test Case Generation

Please look at the generate() method in visualizer's source code for the exact details about test case generation. Each test case is generated as follows:

- The number of teams **N** is chosen between **6** and **50**, inclusive.
- The number of points for winning a game **W** is chosen between **2** and **6**, inclusive.
- The number of points for drawing a game **D** is chosen between **1** and **W**-1, inclusive.
- The number of simulations per game **X** is chosen between **1** and **10**, inclusive.
- The attack strength of each team is chosen between **2** and **10**, inclusive.
- The defence strength of each team is chosen between **1** and **10**, inclusive.
- Generate all the game outcomes as described in the Simulation section above.
- All values are chosen uniformly at random.

Notes

- The time limit is 10 seconds per test case (this includes only the time spent in your code). The memory limit is 1024 megabytes.
- The compilation time limit is 30 seconds.
- There are 10 example test cases and 100 full submission (provisional) test cases. There will be 2000 test cases in the final testing.
- The match is rated.

Languages Supported

C#, Java, C++ and Python

Submission Format

Your submission must be a single ZIP file not larger than 500 MB, with your source code only.
Please Note: Please zip only the file. Do not put it inside a folder before zipping, you should directly zip the file.

Make sure you name your Source Code file as SoccerTournament.*<appropriate extension>*

SAMPLE SUBMISSIONS

Here are example solutions for different languages, modified to be executed with the visualizer. You may modify and submit these example solutions.

- Java Source Code - SoccerTournament.java
- C++ Source Code - SoccerTournament.cpp
- Python3.6 Source Code - SoccerTournament.py
- C# Source Code - SoccerTournament.cs

Please Note: You will have to **zip** the file before submitting. Please zip only the file. Do not put it inside a folder before zipping, you should directly zip the file.

Tools

An offline tester is available below. You can use it to test/debug your solution locally. You can also check its source code for an exact implementation of test case generation and score calculation. You can also find links to useful information and sample solutions in several languages.

DOWNLOADS

- Tester source code
- Tester jar

OFFLINE TESTER / VISUALIZER

In order to use the offline tester/visualizer tool for testing your solution locally, you'll have to include in your solution the main method that interacts with the tester/visualizer via reading data from standard input and writing data to standard output.

To run the tester with your solution, you should run:

java -jar tester.jar -exec "<command>" -seed <seed>

Here, <command> is the command to execute your program, and <seed> is seed for test case generation.
If your compiled solution is an executable file, the command will be the full path to it, for example, "C:\TopCoder\SoccerTournament.exe" or "~/topcoder/SoccerTournament".
In case your compiled solution is to be run with the help of an interpreter, for example, if you program in Java, the command will be something like "java -cp C:\TopCoder SoccerTournament".

Additionally you can use the following options:

- **-seed** <seed> Sets the seed used for test case generation, default is seed 1.
- **-debug** Print debug information.
- **-N** <N> Sets a custom number of teams.
- **-W** <W> Sets a custom number of points awarded for winning a game.
- **-D** <D> Sets a custom number of points awarded for drawing a game.
- **-X** <X> Sets a custom number of simulations per game.

Custom parameters also accept ranges. For example **-N 10,20** makes the number of teams to be randomly chosen between 10 and 20, inclusive. Finally, you can print any debug information of your solution to standard error, and it will be forwarded to the standard out of the tester.

Marathon local testers also have other options, including running a range of seeds with a single command, running more than one seed at time (multiple threads), controlling time limit, saving input/output/error and loading solution from a file. The usage of these other options are described here.

Payments

Topcoder will compensate members in accordance with our standard payment policies, unless otherwise specified in this challenge. For information on payment policies, setting up your profile to receive payments, and general payment questions, please refer to Payment Policies and Instructions.

ELIGIBLE EVENTS:
2021 Topcoder(R) Open

CHALLENGE TERMS:
Standard Terms for Topcoder Competitions v2.2

SHARE:

ID: 30146451

A Guide to Building a Career...

Oct 29, 2020

5 min

10 YEARS OF PROBLEM WRITING:...

Oct 29, 2020

5 min read

Ratings

Oct 29, 2020