CAB203 Graphs Project

Joshua Wlodarczyk: N11275561

Date:

## Tournament Structure:

The first problems ask to develop a structure for the tournament; with the intended purposing being to create a formula that can determine weather a set of games meet the required conditions set of every player having an appropriate opponent.

The tournament structure can be broken down into 2 main properties:

1. Every player plays against each other or there are a minimum of 2 other players they both play against.
2. All players have the same number of games.

The first property can be defined as for every player (*p*) in the set of games (*S*) is the neighbour of all other *p*, which is defined as possible opponents (*o*) or () *p* shares at least 2 neighbours with the *o* if they are not neighbours.

The second property can be defined as for every *S* find the number of neighbours for (*p*, *o*) and ensure they each have an equal amount.

Both properties involve the formula of finding the neighbours of (*p, o*)

The degrees formula can be used to determine the number of neighbours per element in *S* and Handshaking lemma can be used to help determine (*p, o*) all have at least 2 neighbours in common

From the stated properties I can be determined that the tournament structureneeds to be symmetric and irreflexive in nature.

The following graph depicts a game that meets the tournament structures requirements:

|  |  |  |  |
| --- | --- | --- | --- |
| Figure 1: | A = a, B = b, C = c, D = d   |  |  | | --- | --- | | Neighbours: | Degrees: | |

With this the properties can be given as the following 3 equation formula:

When implemented into python with *V* determining the neighbours of the players with *E counting* the number of games and determining checking if the *p* play against all *o* or share *o* in common with other *p* that are not *o*. *V* and *E* are sets with representing a Boolean statement that checks if the conditions of are met. If true, then all properties are met, and the game is valid otherwise return false.

## Potential referees:

The second problem requires that each referee is assigned a game to referee with some of the referees being players themselves. A formula meeting the listed 4 properties below will need to produce a set of games with assigned referees:

1. Every game needs at most one referee.
2. Every referee needs at most one game.
3. A Referee can’t be a player in a game they are playing in.
4. Referee can’t have a conflict in that game.

(note to self: Explain element naming schemes and sets as present in the formula)

Graph colour theory is also present in developing a formula as the *chromatic* number can be used as a representation for referees and their conflicts over the games. Graph theory follows the following formula:

The bipartition of 2 elements represented by A and B:

Formula (FIX):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Figure 2: | The following graph depicts the ideal set of games, the numbers, paired with referees, the letters. 3 different colours are present in the example.  (CHANGE) | | |  | | --- | | The formula can be broken down into 3 equations:  Equation 1: For each referee in the set of the games, do they have any conflicts and are they present in that game.  Equation 2: All referees who are a neighbour in a game must not be able to ref that game.  Equation 3: All referees are the neighbours of each conflict within the set of Conflicts. | | The formula written out: | |

When translating the formula to python

V = (games, referees) as in (game1, game2, game3, game4, ref1, ref2, ref3)

E = ((game1), referee1), (game2), referee2, etc)

## Assign referees:

Shortest Path?  
Bipartite?

* Neighbours
* Create edges between games without the same players or referees
* Use the union of games and neighbors to join games into groups

assignedReferees = { ('Alice', 'Bob'): 'Rene', ('Elaine', 'Charlie'): 'Dave' }

output = [{game1, game2}, {game1, game2}]

Problem three states that a formula needs to be developed that can schedule games into a tuple such that the first element is game 1 and the second element is game 2.

The assigned referee games that problem 2’s formula produces will set the games that need to be grouped. The formula will need to meet the 3 following properties:

1. Each person is involved in at most one game in any game group as player or referee.
2. Each game must have a different referee.
3. Each game is only grouped at most once.

Graph colour theory is present in developing a formula as the *chromatic* number can be used as a representation for the minimum number of possible timeslots that can be generated from assigned referees. Graph theory follows the following formula:

With the following formula representing that no 2 vertices are adjacent:

The following formula can be used for the of the Set:

The Following graph is a depiction of the structure of what the problem follows:

|  |  |
| --- | --- |
| Figure 3:  Note that vertices with the same colours have different players and referees.  Figure 4:  Note that the edge has been formed as either player1 or player 2 in game1 is the same in game2 or they have the same referee. | Figure 3 is a graph displaying how the games should be organised into timeslots. *A, B, C, D, E, F* each represent a different game (set of players) and are the vertices. The vertices are the keys of assigned referees (output of problem 2), each key is an individual game.  The edges represent the games that share the same referees or players. It indicates what can’t games can’t be scheduled at the same time.  As previously mentioned, the colours display the *chromatic* number and determine the number of timeslots, in this case there are 3.  The Formula can be written out as: |

## Games schedule:

Question 4

* Top Down
* Asymmetric, directed graph
* Add edges from referee to game for games the referee plays in, and from game to referee for games the referee is refereeing
* Use topOrdering(V, E) to sort

Determine if:

1. Players who are playing and referring other games play first.

Directed (the graph nature):

Topological Ordering:

Directed Graph

topOrdering says which referee should play first

The following figure displays a basic graphical structure of the problem:

|  |  |
| --- | --- |
| Figure 5:  A diagram of a diagram  Description automatically generated  Note: the example uses game groups with a game in each. The graph structure would still work if the game groups had 2 games.  is represented as all the referees and game groups.  is the direction from the referee to the game group to the referee that plays in that game group. | Figure 5 depicts an graph of the order game groups should be played based on the game referees play.  The constructed set from the figure:  From this we can indicated the graph is with the topological ordering being:  The topological order displays the order that referees ref a game group, as in refs and then can ref and then so on. |

The formula can be written out as:

Python will need to check topological output is a referee(key) or game group (set()).

## Player ranking:

Flow Control

Change?

1. If a player wins against there opponent it is a primary win.
2. If that opponent wins against another opponent the player has not played against and they win it is considered a secondary win for the player.
3. The player does not gain a secondary win for opponents they have played against.
4. Once a set amount of secondary wins has been met, the secondary wins will reset and the player will gain one primary win.

References:

Notes:  
Q1: Neighbours, Degrees, Neighbour Sets (to find if they share opponents?)

Redo working out use the equations in how you worked it out but when mathematically showing it create your own based on the question.

Show dot points in a mathematical equation.

Evey player needs to play against or they need 2 games

Q2:

Equation 1: For each referee in the set of the games, do they have any conflicts and are they present in that game.

Equation 2: All referees who are a neighbour in a game must not be able to ref that game.

Equation 3: All referees are the neighbours of each conflict within the set of Conflicts.

Q3: Spanning Path, Neighbour Set

Q4: TopOrdering, Find path, ShortestPath

Q5: Flow control