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.

Graph bipartite theory is present in developing a formula as the players need to be separated from the referees in each game games. Graph theory follows the following formula:

The bipartition of 2 elements represented by A and B:

Matching of the bipartite graph will also be used to produce a set of matched games:

The following figure depicts the structure of the games that needs to be produce:

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | Figure 2: | The following bipartite graph depicts the ideal set of games with the letters representing the referees and the numbers representing the players.  The neighbours of the referees are the players and show how the games should be formatted. | | The vertices, , are the set of all players and referees.  The edges, , are the set of games, the first player and second player in that game, to referees.  The formula can be broken down into 3 equations:  is the set of all games.  is the set of all referees.  is the set of conflict for each referee.  represents weather is assigned to  Equation 1: All games need at most one referee and each referee needs at most one game.  Equation 2: For each referee in the set of the games, do they have any conflicts and are they present in that game.  Equation 3: All referees who are a player (neighbour) in a game must not be able to ref that game. |

When translating the formula to python

Check if players are in the csv file

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

E = ((player1, player2), referee1), (player1, player2), referee2, etc)

Use max matching to match the games

## Assign referees:

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 stated as:  Let be the set of players and referees that are playing.  Let be the set of all games.  Let be the referees assigned to each game.  Let be the referees assigned to each game.  Let be the game groups where is the game in .  Each person is in at most one game group as player or referee:  Each game group is group once and has a different referee. |

The python code closely follows the explanation and graph structure with elements such as and being sets and games and refs being used as list to check that each referee and game meets the set requirements of Equation 1 and 2. and are used with from the graphs.py module. The represented as the chromatic colouring of the graph is used to produce a list of dictionaries with from the same python module, all the elements of this list are extracted from this and returned.

## Games schedule:

The fourth problem wants to order the game schedule produce from the previous problem so that referees who are players play in the first game group. The problem can be broken down into one simple property:

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

From this it can be determined that the problem is a directed graph with the start point being the referee that will ref the first game group that need to be played first and the end point being the last game group played. The start point game group will have players that are also are referees, with the end point will have players that have no games to referee.

Directed graph theory will be used as the problem is a directed graph:

With the directed graph topological ordering can be used to state the order of the vertices in the directed graph:

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:

Let R be the topological order for the game schedule.

Let be the set of game groups.

Let be the set of assigned referee games.

Let be the set of game schedules.

Let be the referee assigned to each assigned referee game.

Let be the game groups where is the game in .

Let be the game groups in game schedule.

From the topological order create a schedule of game groups:

Out of the game schedule, T, order the game groups such that the game groups that referees play in ref games first and referees with no games ref their games after. The property of the problem can be met, referees who are players play first:

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

## Player ranking:

Flow Control

Note: maybe have another look if have time

The fifth and final problem looks at finding the finding the maximum possible score for each player with the score coming from the games one by each player. The following problem can be broken down into 4 main properties:

1. If a player wins against their opponent, it is a primary win, player gains primary number of tokens.
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, opponents gain primary number of tokens and player gains secondary number of tokens.
3. The player does not gain a secondary win from that opponent when they play against other opponents they have played against.
4. A capacity is set to determine the maximum number of tokens a player can receive from their opponents.

The following properties can be solved using flow theory. The use of the maximum flow formula into the drain can be used to find the flow output of each game. The formula is as such:

E: games from winner to loser

V: each player

f: the flow of each edge

w: the capacity of each edge

using max flow will find the flow output (f) of each game based on w.

|  |  |
| --- | --- |
| A diagram of a diagram  Description automatically generatedFigure 6:  A = Alice  B = Bob  C = Charlie  D = Dave | Figure 6 displays a representation of how the scoring should be played out and follows the example of test 1 in TestScores from test\_project.py. The max scoring for each player is calculated by adding together the flow output for each of the vertices like so:  ¾ = primary win  2/4 for A comes from the 2 points given from the secondary wins granted from B and D victories (1/4 each)  A: 3 + 3 + 2 = 8  B: 3  D: 3  C: 0 |

V: All players

E: Direction of the flow graph from the source to the sink for each player

Let be the edges for to the sink, representing the winning player and representing the losing opponent in E.

Let be the set of primary wins per player.

Let be the set of secondary wins per player.

Let represent the flow from player to opponent .

Primary wins for each player:

Secondary wins for each player:

Capacity constraint for each player:

Maximum Token amount for each player:

References:

Lecture 6

Lecture 7

Lecture 8

(Reference python files)

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:

The formula written out:

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