Assignment 3

Brian Moyles – 21333461

1. Suggest a way of representing the graph in a relational database

My approach would require 3 tables:

- A table for the player details
- A table for the game details
- A table to represent the graphs

Player table

For the purpose of this assignment, the only necessary details needed to store in the player table is player id and player name.

playerID (P Key)	playerName
1	Brian
2	Colm
3	Dave
4	Frank

Game Table

The game table is needed in this question to be able to reference what game the graphs refer to. For this assignment I will only store the game id and the date of the game, but this can be expanded to other details if needed.

gameID (P Key)	gameDate
1	2-5-23
2	12-5-23
3	16-5-23

Graph Table

The graph table will represent the graph of the players. The table structure is shown below:

graphID (P Key)	gameID (F Key)	Player1ID (F Key)	Player2ID (F Key)	Distance
1	1	1	2	3
1	1	1	3	5
1	1	1	4	1
1	1	2	3	6
1	1	2	4	2
1	1	3	4	4

The structure of this table has 5 columns:

- graphID = stores the id of the graph in which it references. For every graph created and stored, there should be that many instances of a graphID (one for every player on the pitch) e.g., 22 instances of graphID = 1 as there are 22 player locations to track for that graph.

- gameId = references the game it is linked to. [Foreign key to game table]
- player1ID = the player the data refers to. [Foreign Key to the player table]
- player2ID = the second player the data refers to (Foreign key to the player table)
- Distance = the distance between player 1 and player 2

I believe this is a straightforward way to represent these graphs in a relational database. It clearly shows which graph and game the data belongs to. It also clearly states the player relationships between each player. The database would be set up so each player id would have an entry for each player relationship (e.g., player 1 has a relationship with player 2,3,4 etc but not with himself). It would start every entry at [playerID + 1], ensuring no repeat entries and no relationship with itself (e.g., player 2 entries start with player 3 as player 2 relationship with player 1 has already been entered, and player 2 does not have a relationship with himself). The distance between each node is then stored. It clearly shows the distance between each player in the database.

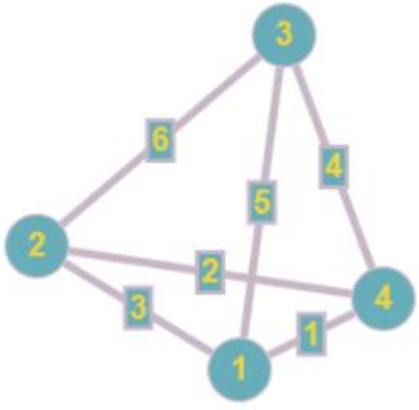
2. Suggest a suitable means to represent the data in a data structure

I will be using an adjacency matrix to represent the data in a data structure.

Adjacency Matrix

	1	2	3	4
1	0	3	5	1
2	3	0	6	2
3	5	6	0	4
4	1	2	4	0

The adjacency matrix is a suitable approach as you can clearly store the data. The data is structured to show the distance between each node. For example, the entry at [1,2] = 3 is the distance (3) between playerID = 1 and playerID = 2. By storing the data as shown. It allows for easy interpretation of the data structure and creation of the graph which shows the distances between the players. The graph with the data will be an undirected graph so that, although the data is duplicated due to the structure of an adjacency matrix, it is not entirely redundant. Due to the fact that each adjacency matrix will represent a team, each adjacency matrix to store data will be 11x11 for the 11 players on the pitch (this can be extended for subs but not necessary for this example). The matrix is relatively small, so the duplicate data won't be an issue.



The above shows how the above adjacency matrix data structure can be used to plot the data in a graph and display the distance between each player. Using the example database in question 1 as reference, we can see above the relationships player 1 (Brian) has with the other players. The number represents the player id and the distance between players is displayed on the edges between them. As shown, player 1 (Brian) has a distance of 3 between player 2 (Colm) on the pitch at the time of storing the data.

Note: due to players not having a distance with themselves, the adjacency matrix values for their own entries (e.g., [1,1], [2,2], etc) would be 0. If I were to expand this data structure (which is not necessary for this assignment), I would develop the adjacency matrix to store the player x,y coordinates in their own entries. This way the positions and distances would be in the same data structure, making for a very efficient data structure.

3. Suggest an algorithm to measure the similarity of 2 graphs.

My approach to measure the similarity of 2 graphs is to compare the average weight of the graphs.

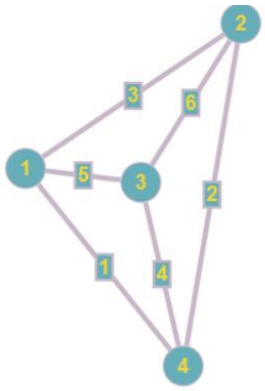
To calculate the average weight of a graph:

- Sum of all distances / number of nodes

I would then subtract the 2 weights and get the absolute value, and the closer the value to 0, the more similar they are

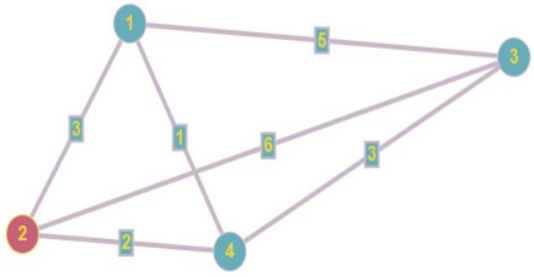
Example:

- Graph 1



Average weight: 6+3+5+4+2+1 / 4 = 5.25

- Graph 2



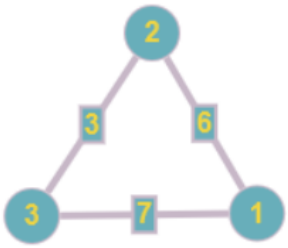
Average weight: 6+5+3+1+2+3 / 4 = 5

- Subtract average weights:
  5.25 – 5 = 0.25

Because the similarity difference is close to 0 (0.25) it shows the 2 example graphs are very similar

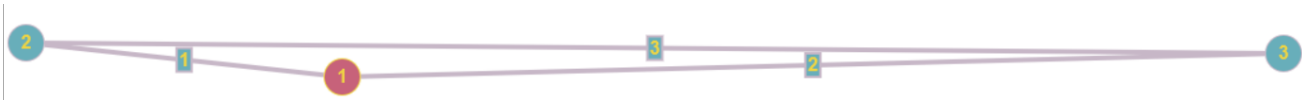
Example 2

- Graph 1



Average weight: 7+3+6 = / 3 = 5.33

- Graph 2



Average weight: 1+3+2 = / 3 = 2

- Subtract average weights:
  5.33 – 2 = 3.33

The difference (3.33) is not close to 0 and therefore the graphs are not very similar

4.

a) Given sample data, write code/pseudo code to calculate the degree of each node for any snapshot (a given timestamp).

My approach for this can be broken down into steps:

- Iterate through half the matrix (due to half the matrix representing the entire graph)
- Ignore every value that violates the constraint
- Calculate the degree of each node with the remaining values

Code

```
/**
 * Question 4 Part A
 * @author Brian Moyles - 21333461
 */
public class question4 {

    public static int[][] matrix;

    public static void main(String[] args) {
```

```
        matrix = new int[4][4];
        int numVertices = 4;
        int k = 2;
        int numDegree = 0;

        // Create the Adjacency Matrix
        matrix[0][0] = 0;
        matrix[0][1] = 3;
        matrix[0][2] = 5;
        matrix[0][3] = 1;
        matrix[1][0] = 3;
        matrix[1][1] = 0;
        matrix[1][2] = 6;
        matrix[1][3] = 2;
        matrix[2][0] = 5;
        matrix[2][1] = 6;
        matrix[2][2] = 0;
        matrix[2][3] = 4;
        matrix[3][0] = 1;
        matrix[3][1] = 2;
        matrix[3][2] = 4;
        matrix[3][3] = 0;

        // Display the Matrix
        System.out.println("\nMatrix");
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }

        // This can be used to calculate the degree of each node for a given
timestamp
        System.out.println("\nDegrees of each node, excluding values <= K");
        for (int i = 0; i < numVertices; i++) {
            System.out.println("\ni = " +(i+1));
            numDegree = 0; // Reset degree for each node
            for (int j = 0; j < numVertices; j++) {
                if (matrix[i][j] > k) {
                    System.out.print(matrix[i][j] + " ");
                    numDegree++;
                }
            }
            System.out.println("\nDegree of Node " + (i+1) + ": " + numDegree);
        }
    }
}
```

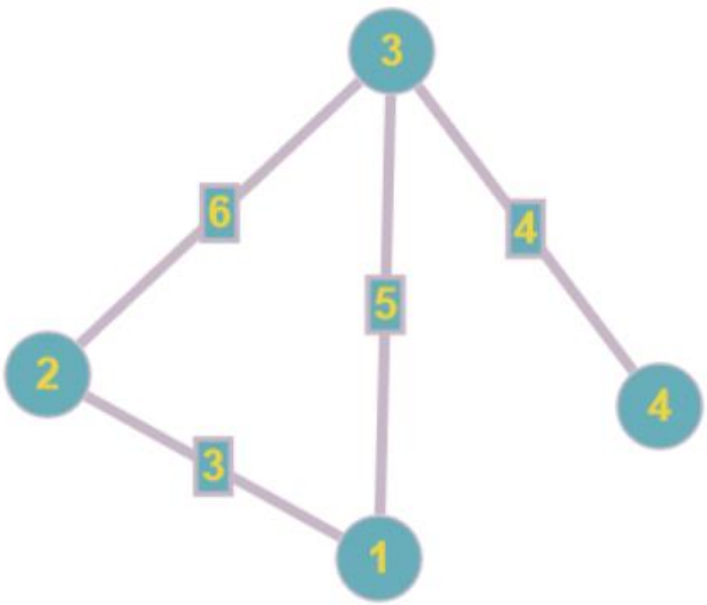
Output

```
Matrix
0 3 5 1
3 0 6 2
5 6 0 4
1 2 4 0

Degrees of each node, excluding values <= K

i = 1
3 5
Degree of Node 1: 2

i = 2
3 6
Degree of Node 2: 2

i = 3
5 6 4
Degree of Node 3: 3

i = 4
4
Degree of Node 4: 1
```

Explanation

The above code shows my workings too calculate the degrees for each node. Using the adjacency matrix and graph in part 2, I created that matrix in java. I then printed the matrix to the console to confirm that the data is correct. The next code snippet can be used within a function and called at any given timestamp to perform the task of finding the node with the most degrees. It iterates through half of the adjacency matrix and compares the values to the constraint, which is 2 in this example. Those values are disregarded in the calculation. It then checks the values for each node and calculates each degree based on paths with distance above the constraint.

The below graph is the new graph without the paths that violate the constraint



b) Given sample data, write code/pseudo code to determine which node(s) is on the most paths?

To determine the node(s) with the most paths I will use 2 functions. The first function will use a variable to store the greatest number of paths a single node has. This will be updated if there is a node found with more.

An array will be used to store any node with the most paths.

A count function will count the number of paths per node.

Each number of paths will then be checked with the previous current number of most paths. If the number of current paths is greater than the previous number of most paths, then the array will be emptied and the new node with the most path is added. If any node also has the most paths, it will be added to the current array.

When every node is checked, the array with the nodes with the most paths will be returned.

Pseudocode

```
func FindNodeWithMostPaths(matrix) {
    mostPaths; variable to track node with most paths
    nodesWithMostPaths[]; list to store nodes with most paths

    iterate through each node:
        call a count func to calculate number of paths starting from current node
        if numberOfPathsPerCurrentNode > mostPaths :
            update the mostPaths variable
            Clear the nodesWithMostPaths array and add the current node
        else if numberOfPathsPerCurrentNode = mostPaths :
            add current node to the array

    return nodesWithMostPaths array to display node(s) with most paths
}

func Count() {
    visited[]; create array to track visited nodes
    count = 1; variable to store number of paths from current node

    add current node to visited array
    check all neighbors of current node

    if neighbor is not in visited array
        increment count by calling recursive function on that neighbor
        remove neighbor from visited array
        return count
}
```