Computational thinking, Group 4

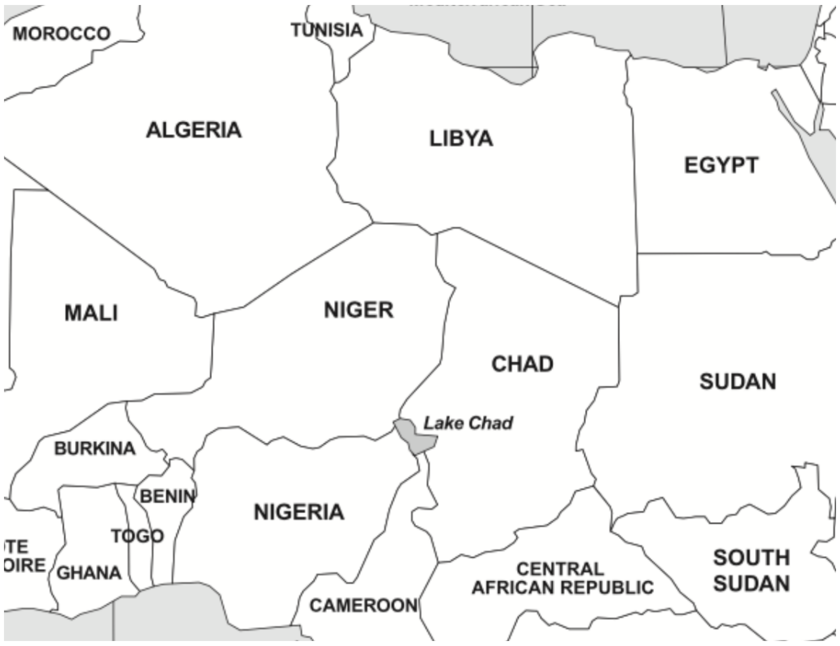
Coursework 3, Graphing abstraction

University of Manchester

Participants

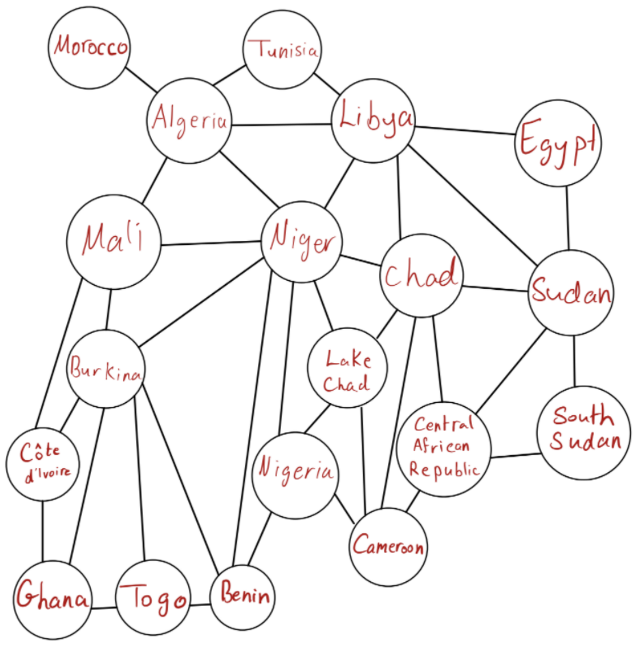
- Joseph Hayes
- Zak Allen
- Fatimah Patel
- Mohammed Alnusif
- Alaa Dahlawi
- Shuhua Shao

Task 1



Graphing Abstraction of Graph

1



[1] The graphing abstraction is the map broken down into a graph abstraction. The lines representing the borders between countries.

Definitions:

ColourList is a list of unique, different, non-repeating colours: [red, blue, green, yellow, …]

UnavailableColourList is a list of the colours that cannot be used to colour a country on the map

ColouredCountryList is a list of countries paired with their colour on the map

Making a graph abstraction:

Pseudocode:

```
START

Input the map

For each country in map

        {

            Create a node with the name of the country

        }

For each node in map

        {

           If the node's country shares a border with another node's country in the map

           Then

                 Create an edge between the two nodes

        }

Output the Graph
```

END

Colouring the graph:

```
START
Input the Graph

Create empty UnavailableColourList

Create empty ColouredCountryList

Create ColourList


For each node in graph

{

    Clear UnavailableColourList

    If node share edge with other nodes

        Then add the colours of other nodes in UnavailableColourList

# In the IF statement it always uses the first colour from ColourList that is not in
UnavailableColourList

    For each colour in ColourList

    {

    # UnavailableColourList contains the colours that the current node cannot share an edge
with so if colour is not in UnavailableColourList it can be used.

    If colour is not in UnavailableColourList

        Then

            colour the node with this colour and add [node name, colour] to
        ColouredCountryList
        }
}
Output **ColouredCountryList**
**END**
# The output would be something like [[Egypt, Red], [Morocco,Blue],...] for example.
```

Task 2

Definitions:

UnavailableGuestList is a list of names of the guests that cannot be on the same table because they dislike each other.

GuestList is the list of guest names that was introduced

TableList is the list of tables with all of their respective guests

Node's name : is the guest name

Graph Abstraction:

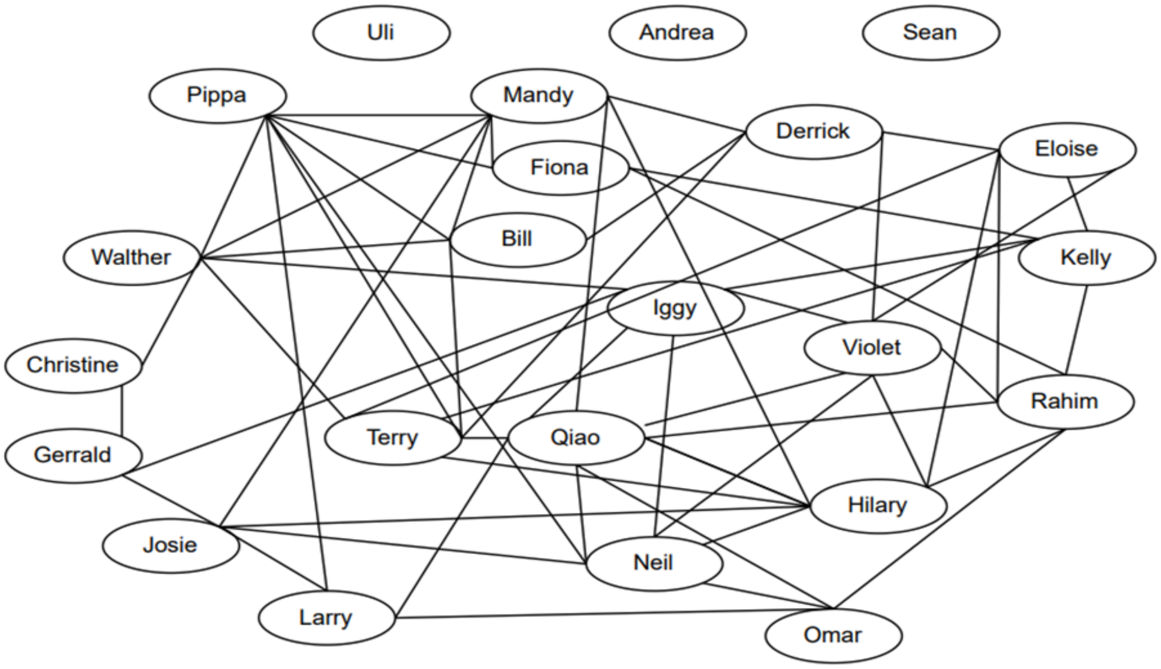
Pseudocode for the graph abstraction:

```
Input the list of guests
For each guest in GuestList
        {
                Create a node with the name of the guest
        }
For each guest in GuestList
        {
```

```
    If the **guest** dislikes another guest and there is no edge between them
            Then
                Create an edge between the two nodes
    }
Output the **graph**
```

[2]

graph:



<u>Pseudocode:</u>

[2] This is the graph abstraction for the guest list. The edge is a 2 way system showing which people dislike. The nodes are the people on the quest list

Putting guest in tables

```
Input graph

Create empty TableList

Create empty UnavailableGuestList

Add table to TableList

For each node in graph

{

        Clear UnavailableGuestList

        IF node ShareEdge with other nodes

                Then add the name of node to UnavailableGuestList

        For each Table in TableList

                {

                        IF Table does not contain someone from node's UnavailableGuestList

                            Then add the name of the node to Table

                            And exit the Table loop

                }

                        IF node's name is not in any table of TableList

                                Then

                                 Add a new table in TableList

                                Add the node's name to the newly added Table

                }

}

Output TableList

# For example TableList : [[Bill,Violet,..],[Neil,Killy,.],...]
```

Task 3

The graphing abstraction in both tasks are different as for the first task the edges represent the locational relationships (borders) whereas in task 2 the edges represent where guests don't want to sit next to. In task 1 the countries do not have a choice on where they sit compared to in task 2 where guests do have a choice on where they do not want to sit next to. Moreover, in task 2, because our edges were based on which guests were not liked by others, we were left with three guests that had no connection to other guests as they were liked by all guests so no edge was needed for them.

The graph abstractions for task 1 and task 2 are similar in that they represent relationships of incompatibility; People who cannot share adjacent locations, countries that cannot share like colours. They both define a set of rules for the sorting/colouring operation that you perform on them. Both problems are alike in that all possible solutions are not commutative with one another; i.e. the order in which you seat each person next to each other, the colours and order of country colouring. Hence both algorithms have to consider the range of all possibilities, if the smallest table/colour count is desired.