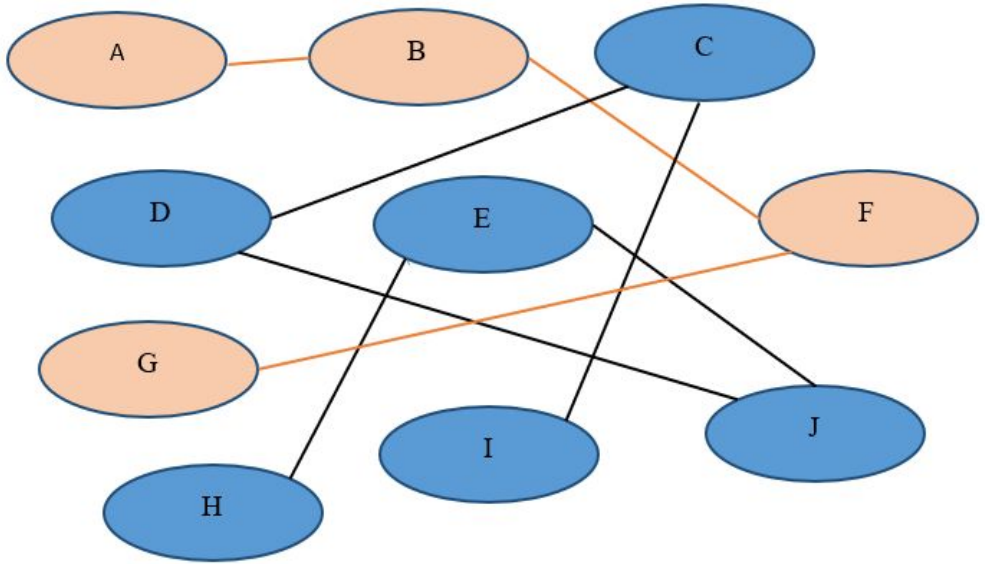
# OUTLINE

- Introduction
- Assignment Breakdown
- Clues and Pointers
- Starter Code
- Summary

# Introduction

What are connected items?

Graphical Representation.




Group1 = {A, B, F, G}

Group2 = {I, C, D, J, E, H}

# Assignment Breakdown

**Instructions.**

**Input File Structure**

(x1, y1, x2, y2, group number)
(13, 2, 13, 3, 1)
(15, 2, 15, 3, 1)
(15, 4, 15, 5, 1)
(15, 3, 15, 4, 1)
(12, 1, 13, 2, 1)
(14, 2, 15, 2, 1)
(15, 5, 15, 6, 1)
(13, 3, 14, 1, 1)

**Output File Structure**

Format: (x1, y1, x2, y2, group number)
(14, 2, 15, 2, 1)
(15, 2, 15, 3, 1)
(15, 3, 15, 4, 1)
(15, 4, 15, 5, 1)
(15, 5, 15, 6, 1)
(12, 1, 13, 2, 2)
(13, 2, 13, 3, 2)
(13, 3, 14, 1, 2)

# Implementation Clues

- Reading the file.
  - (x1, y1, x2, y2, group number)
  - (1999, 1892, 1996, 1994, 3)

$$coordNumRepresentation = x_{coord} * Number\ columns\ a\ matrix\ has + y_{coord}$$

- Assume we set rows = 10000, and columns = 10000. Then we can get a position by

  **position = (x1 * 10000) + y1;**

  **position = (1999 * 10000) + 1892;**

  - What do you store at this **position**?
    - Next Position
    - Boolean value
- Graph Representation.
  - Adjacency Matrix
  - Adjacency List

# Implementation Clues Ctd...

- Building the groups
  - Depth First Search
  - Breadth First Search
- Implementation
  - HashMap
    - Key Value Pair with the group count as the key and array of coordinates as the group
  - Heap
    - Keep the starting coordinate of the group and the count of values
    - Keep a pointer to the next group
  - Arrays
    - Keep the starting coordinate of the group and the count of values
  - Need for boolean value or boolean array
- All these implementations require sorting by the count of values in the group. End of a group is determined by no next connecting element.
- Groups with same count, consider groups with smaller x1 elements first, if these match, then look at y1 elements.

# Starter Code...

Put your code in the method below.

**ConnectedItems::getConnectedItems(char* inputFilePath, char* outputFilePath)**

You are free to write as many methods or classes as you wish, as long as they are referenced in the above method either indirectly or directly.

Do not create any new header or cpp files.

Run your code as in the structure below.

**./homework inputFilePath outputFilePath**

# Summary

- Read the file by help of adjacency list or adjacency matrix.
- Use DFS or BFS algorithms to create and map out the groups.
- Use hash maps, heaps or arrays to store the groups.
- Sort the data structure used above based on number of items in the group, consider x1 and y1 if the group count is the same.
- Implementations discussed here are merely suggestions.
- The limit of implementations is based on the limit of your imagination.