# Week 03 Tutorial

## Advanced Graph Traversal

### Aims

This exercise aims to get you to:

- Implement graph traversal algorithms (DFS, BFS, connectivity)
- Implement complex graph structures (directed, weighted)
- Use the disjoin set to generate the connected components for undirected graphs.

### Exercise 1: Basic Graph Traversal

1. Program the breadth-first traversal algorithm for the graph in Figure 1 starting from A.
   a. Traversal sequence example (not unique): A-B-C-E-D-F-G-H-I.
   b. Please refer to the function **BFS(G, v)**.

2. Program the depth-first traversal algorithm for the graph in Figure 1 starting from A.
   a. Traversal sequence example (not unique): A-B-C-D-E-G-I-H-F.
   b. Please refer to the function **DFS(G, v)**.

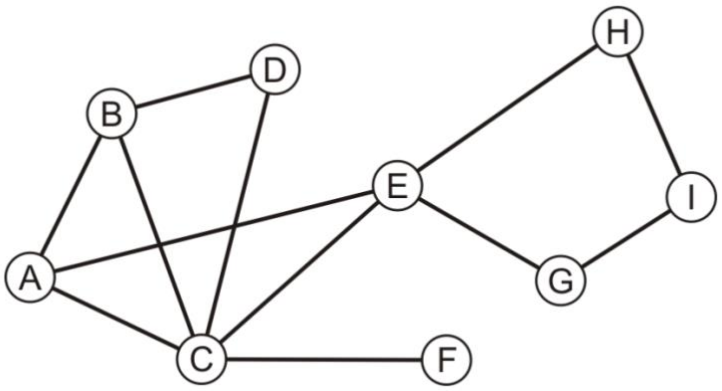


**Figure 1**

3. Load the graph in Figure 2 and check the connectivity between the following node pairs: (A, B), (A, C), (A, D) and (A, E).
    a. The answers: connected, unconnected, unconnected, connected.
    b. Please refer to the function **connectivity(G, u, v)**.

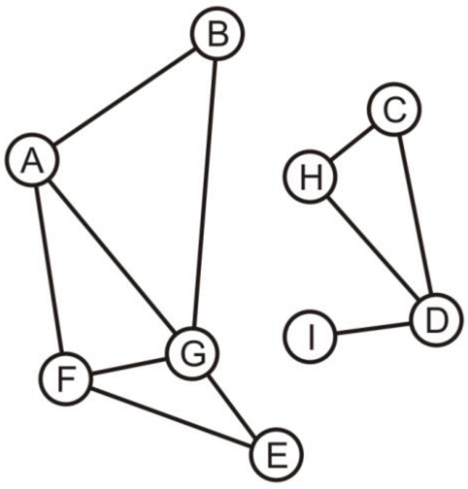


**Figure 2**

## Exercise 2: Complex Graph Structures

1. Change the definition of class ‘SimpleGraph’ to ‘DirectedWeightedGraph’ by yourself. The new class should support the load of directed weighted graphs and allows self-loop and multiple edges between two vertices.
2. Load the directed weighted graph in Figure 3 using ‘DirectedWeightedGraph’.
3. Implement a function, in which when inputting any vertex *v*, it outputs the sum of the weights of all the indegree edges of *v*. Please refer to the function **sumIndegree(G, u)**.
    a. Examples:

| Input | Output |
|---|---|
| A | 1 |
| C | 0 |
| E | 12 |
| G | 1 |

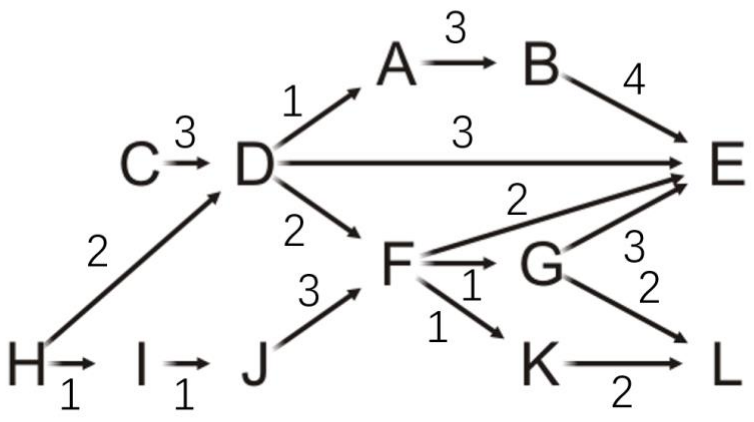


**Figure 3**

## Exercise 3: Connected Components

1. Load the graph in Figure 4 via the class ‘UndirectedGraph’ in ex_3&4.py.
2. The function connectComponents(G, method) in ex_3&4.py which inputs the graph in Figure 4 will compute all the connected components.
3. Implement the class **QuickFind** in ex_3&4.py which use the attempt1 algorithm as illustrated in the lecture slides.
4. Implement the class **UnionFind** that should use the disjoint set data structure for maximum efficiency as illustrated in the lecture slides.
5. The output result should output the connected component each vertex belongs to and the total number of connected components. Each connected component can be represented by any unique identifier, e.g., the root vertex in the disjoint structure or the vertex’s sequence number in the ‘vertex_dict’ of ‘UndirectedGraph’.

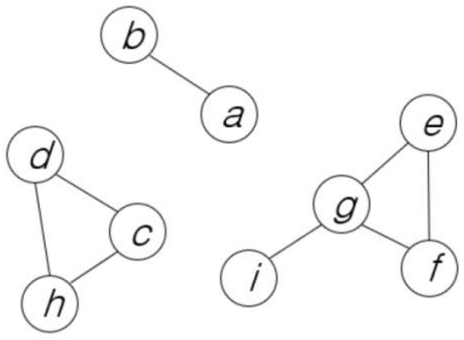
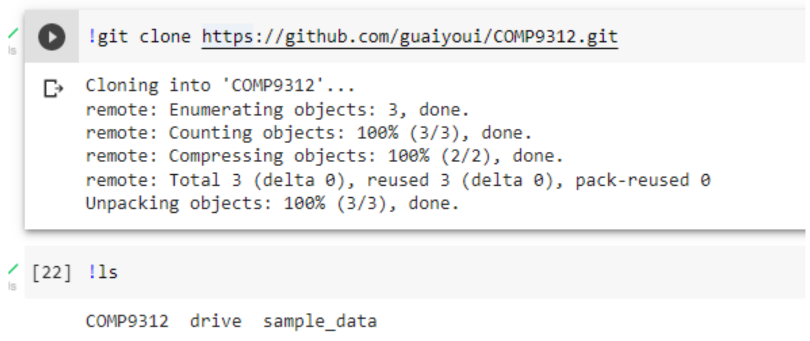


**Figure 4**

## Exercise 4: Efficiency On Medium Dataset

1. Use the dataset in the dataset_30k.txt which contains 30000 nodes and three components.
2. Download the dataset from github in colab using the command "!git clone https://github.com/guaiyoui/COMP9312.git". And use "!ls" to check if the download is successful.



```
!git clone https://github.com/guaiyoui/COMP9312.git

Cloning into 'COMP9312'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

```
[22] !ls

COMP9312  drive  sample_data
```

3. If you are running code on your own machine, then make a new folder named "COMP9312" and put the "dataset_30k.txt" into this folder.
4. Run the whole code and compare the running time of the two algorithms in Exercise 3..