Week 10
Graph Definitions and Representations

Suggested matertials :

- https://www.youtube.com/watch?v=c8P9kB1eun4
- https://www.youtube.com/watch?v=eEQ00TKw1Ww
- https://www.youtube.com/watch?v=1XC3p2zBK34
- https://www.youtube.com/watch?v=tVuEZakQxhQ
- https://www.youtube.com/watch?v=LFKZLXVO-Dg

Fundamental review questions:

1) What are the components of a graph?

2) Give a real-life example of a graph that is directed.

3) Give a real-life example of a graph that is undirected.

4) What does it mean when a graph is called “connected”?

5) What does it mean when a graph is called “fully connected”?

6) Is a tree connected graph?

7) A tree is a specific type of graph. What makes a tree distinguished from graph?

8) What is rooted tree?

9) What is a leaf?

10) How many edges are there in a tree having *n* nodes?

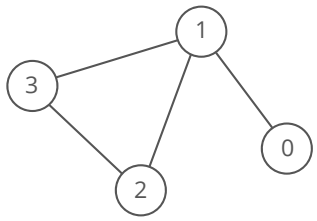
11) How many simple path(s) is(are) there between a pair of tree nodes?

12) What is a “weighted” graph?

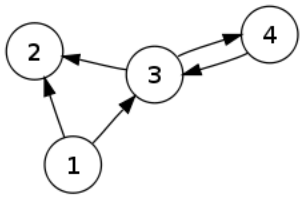
Review questions about Graph representation in program:

13) For each of the graph below, give its representation as an *edge list*, a *collection of adjacency lists*, and an *adjacency matrix*.

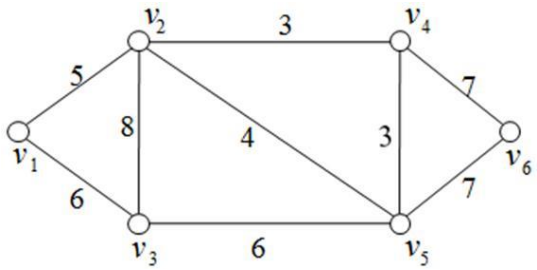
a)



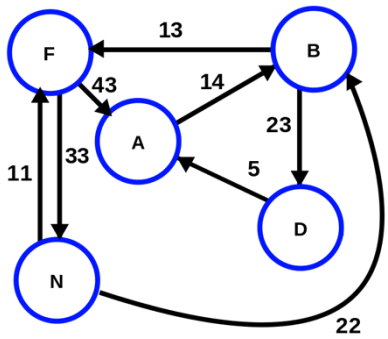
b)



c)



d)



14) Write a Python 3 program that reads an edge list as input and keep the graph in the program as an adjacency matrix.

- You may add an extra input to your program for the number of vertices and the number of edges.
- Test your program with the example graphs in step 12.

15) Write a Python 3 program that reads an edge list as input and keep the graph in the program as a collection of adjacency lists. Test your program with the example graphs in step 12.

- You may add an extra input to your program for the number of vertices and the number of edges.
- Test your program with the example graphs in step 13.

Application Example: Topological sort

Given a Directed Acyclic Graph (DAG), where vertices represent tasks & edges represent dependencies, order tasks without violating dependencies.

(Acyclic means that no cycle exists in the graph)

The provided "topological_sort.py" contains code that will produce a solution for topological sort problem, given the Directed Acyclic Graph in the form of a collection of adjacency lists.

Importing the module with the following line. Be sure to keep in module in the same folder as your program.

```
from topological_sort import *
```

16) Study the provided topological_sort.py in order to understand how it can be utilized. As of now, there is no need to understand how its algorithm solved the topological sort problem.
(for those who are curious, the algorithm is based on Depth-First-Search technique)

17) Write a Python 3 program that imports the provided topological_sort module and makes use of it to solve topological sort problem. Test the program with the two given example graph (graph1.in and graph2.in).