**Summarizing the content:**

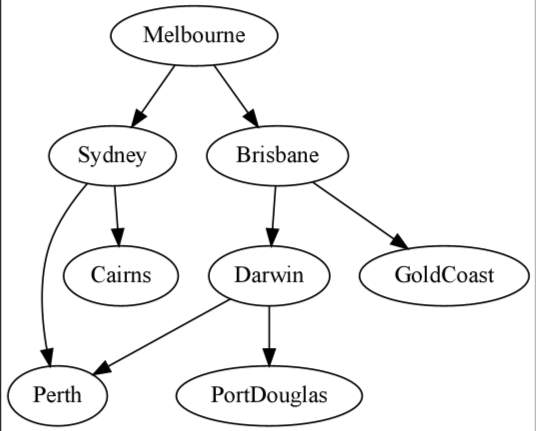
1. Summaries the main points in this module. You may include references to the learning objectives.
2. Introduction to graphs.
3. Undirected and Directed Graphs.
4. Representation of Graphs: Adjacency Matrix and Adjacency List.
5. Depth-First search.
6. Breadth-First Search.
7. How is this useful?
8. DFS: Useful for topological sorting.
9. DFS: in-order traversals of BSTs.
10. BFS: Useful for finding shortest paths.
11. BFS: for testing bipartiteness.
12. DFS and BFS: Useful for exploring graphs, finding connected components.
13. How do you plan to use this information?
14. There is one more method when creating data structures.
15. Trees are similar to graphs, and the two can be considered crosswise.
16. Provide summary of your reading list — external resources, websites, book chapters, code libraries, etc.
17. <https://en.wikipedia.org/wiki/Complex_network#References>
18. <https://en.wikipedia.org/wiki/List_of_information_graphics_software>
19. <https://www.jianshu.com/p/95907d83bfe9>
20. <https://segmentfault.com/a/1190000020445075>
21. <https://zditect.com/article/20479166.html>

**Reflecting on the content:**

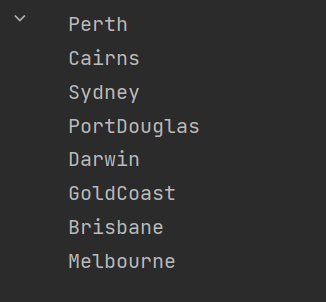
1. What is the most important thing you learnt in this module?
2. Definition of graph.
3. The difference between directed and undirected graphs.
4. Cost of Adjacency Matrix and Adjacency List.
5. Applications of DFS and BFS.
6. How does this relate to what you already know?
7. Connected to the Internet, each data packet is sent through the Internet.
8. When installing the environment, you also need to consider the dependencies of each package. You can use the DFS algorithm to perform topological sorting, and then install the package.
9. Reflect on the code that was given to you in the lab. You can take the screen shot of your python code and add image or just provide the code as text in your report. A good reflection includes:

Activity1:

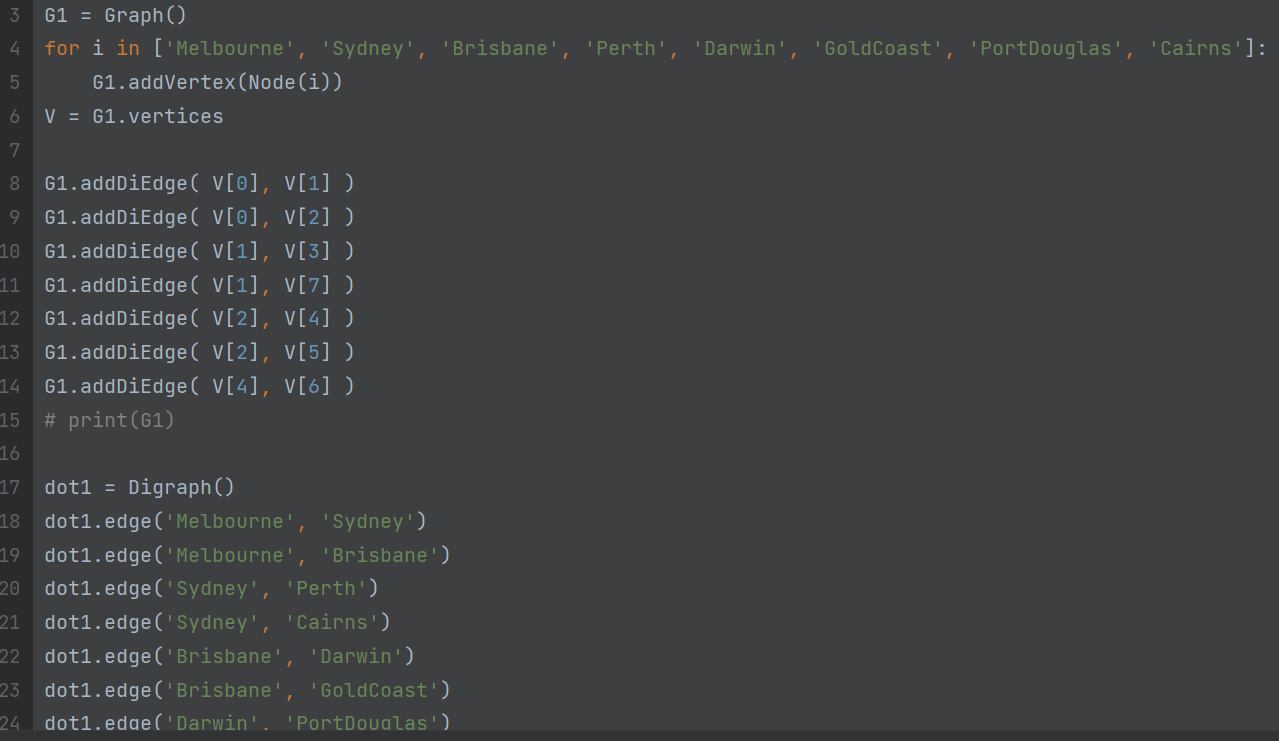
The structure of the graph is as follows:

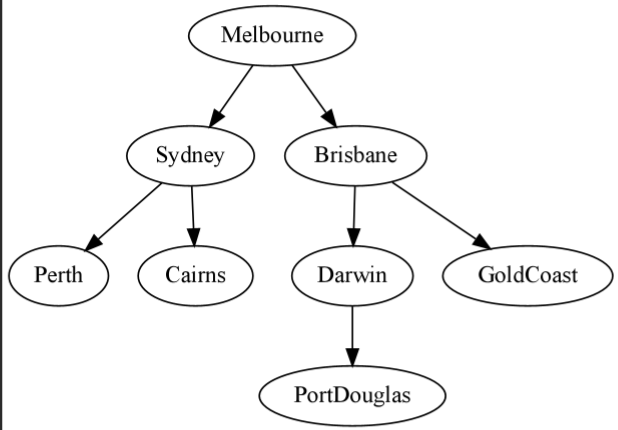


DFS algorithm running results:



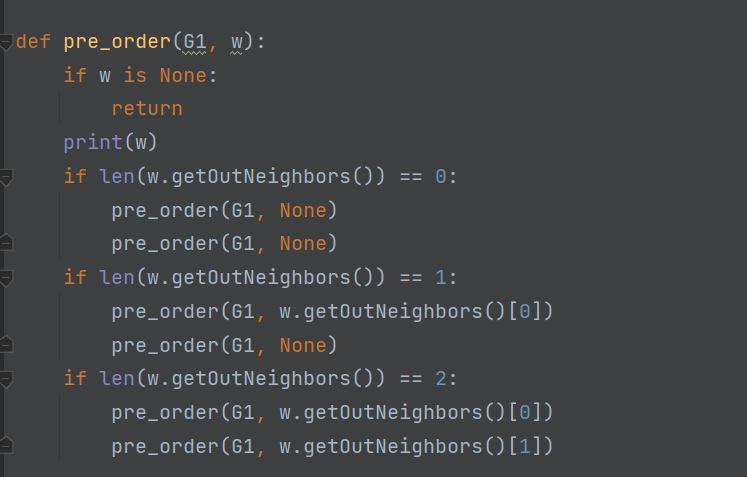
traversal of binary tree, build binary tree:

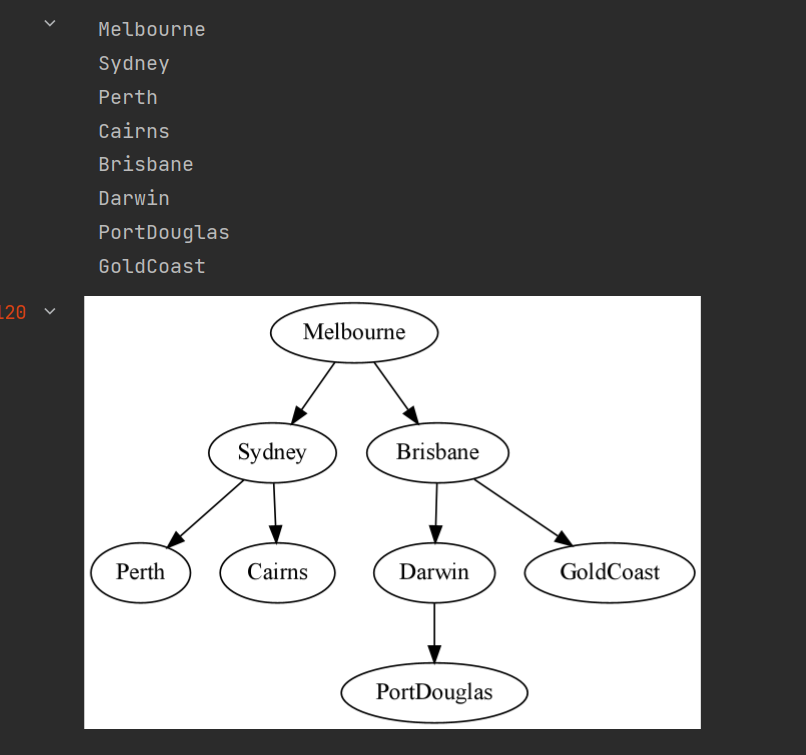




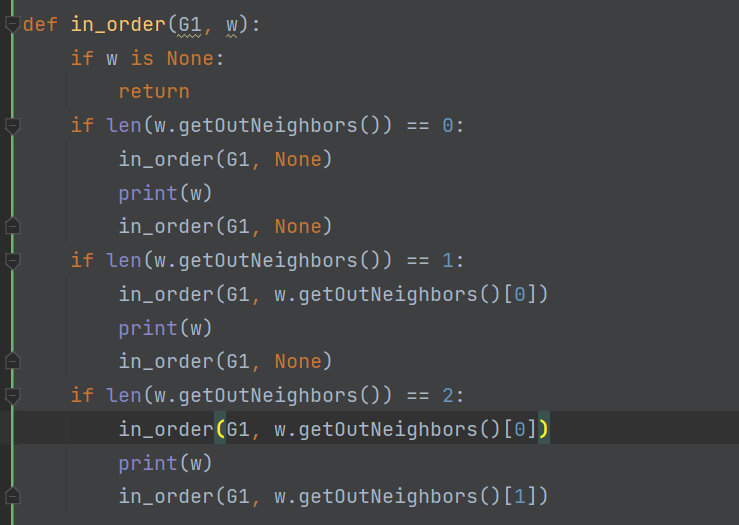
1. pre-order.

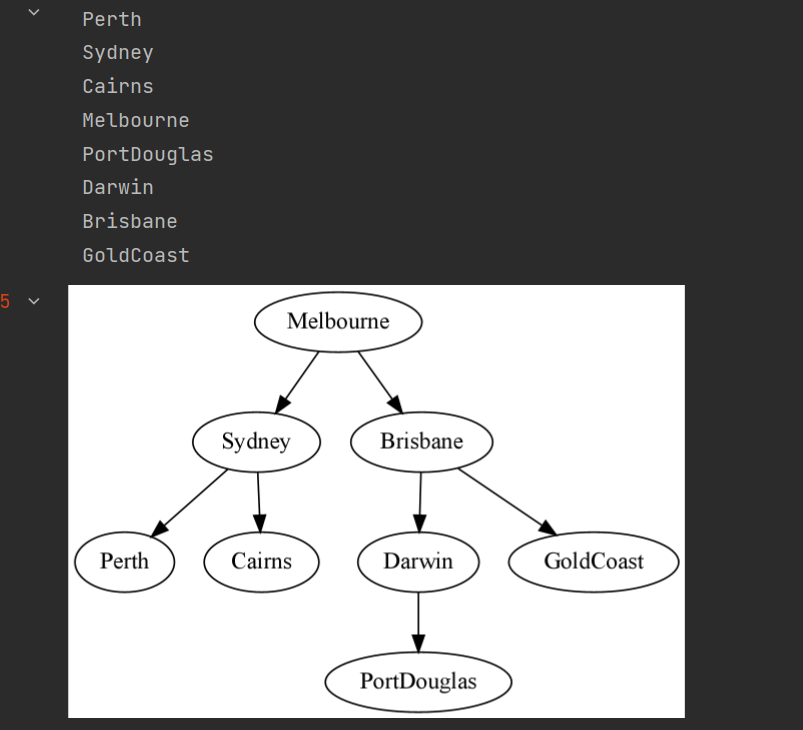
Code and result:



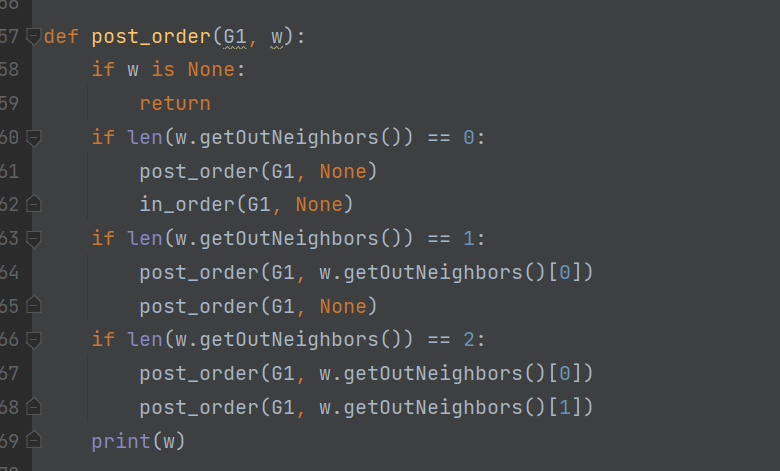


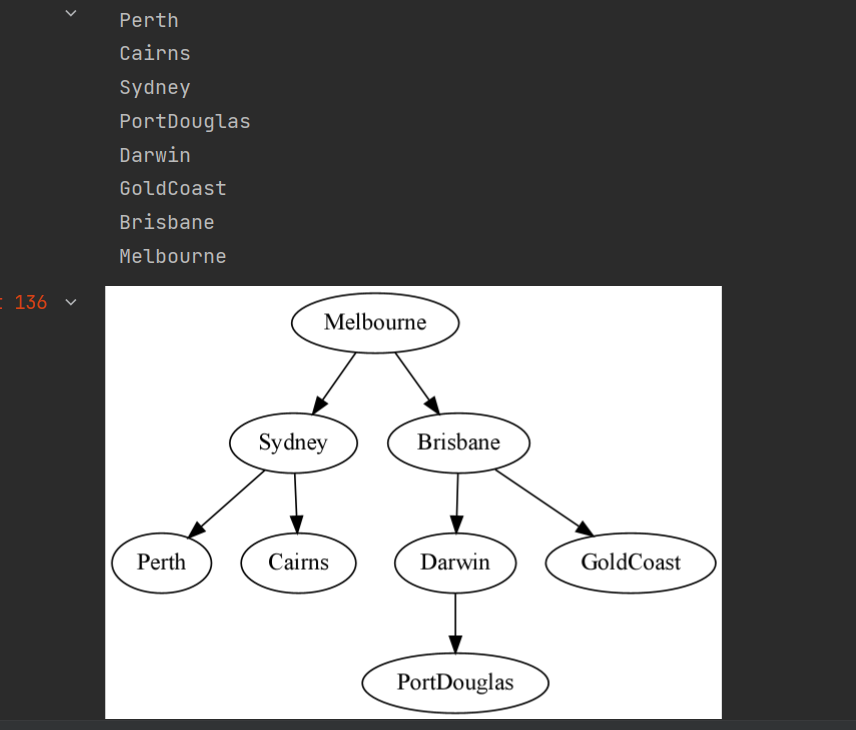
1. in-order:



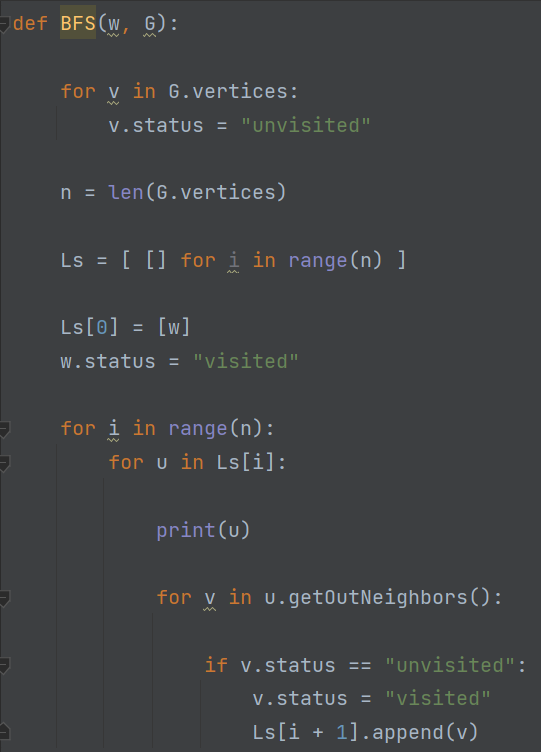


1. post-order:



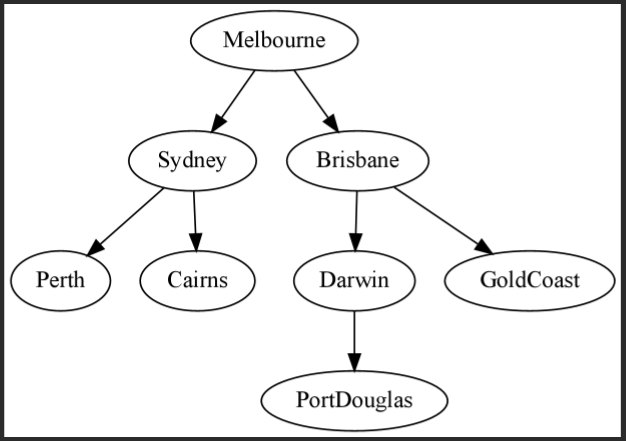


Activity2:

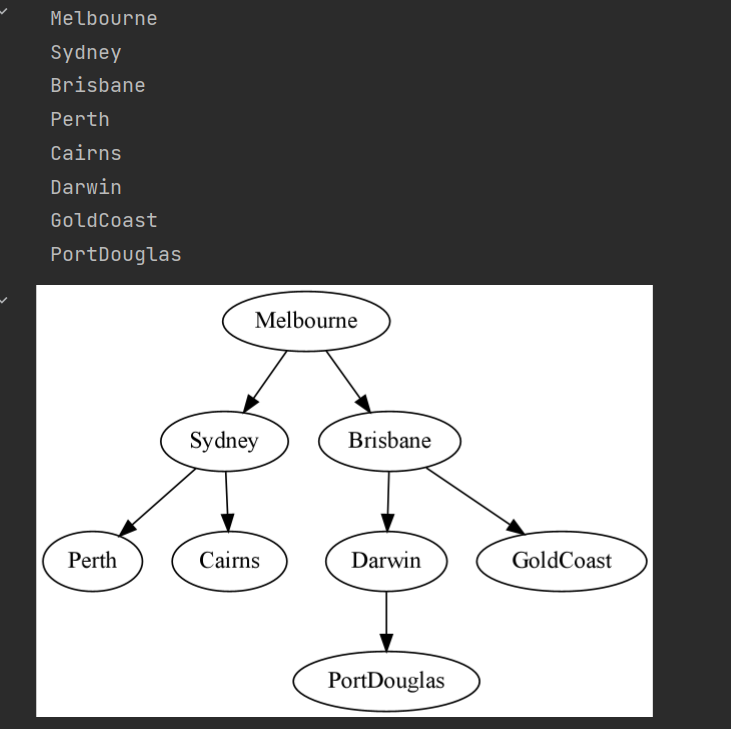


1. First mark all nodes as unvisited.
2. put the starting node into a double list.
3. Put the starting node's outgoing neighbor list into the double list as well.
4. Take out each element in the outgoing neighbor list, treat each element as a starting node, and repeat step 1.
5. Only unvisited nodes are put into the double list, and after being placed in the double column, they are marked as visited.

build a binary tree:



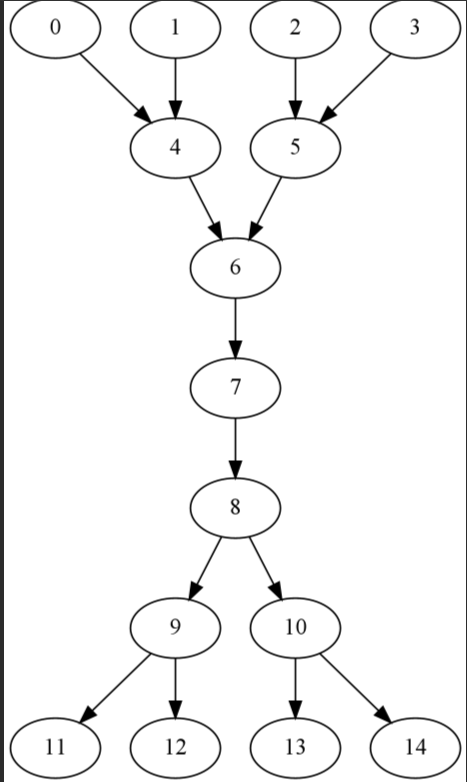
Traverse all elements in the tree using BFS:

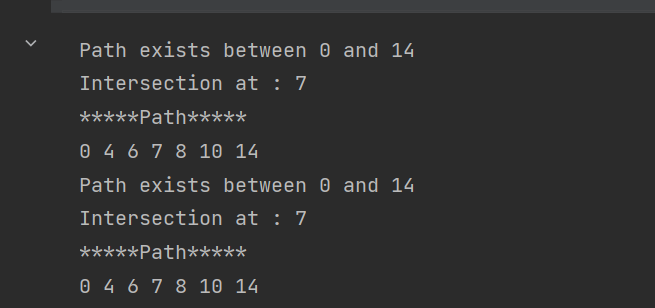


Activity3:

The graph of the build is as follows, the paths of 0 and 14 are searched bilaterally.

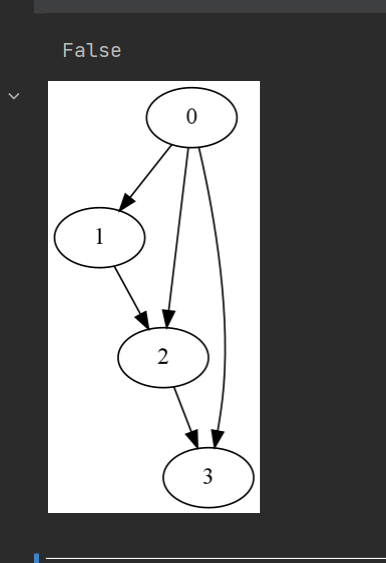
This is an undirected graph.

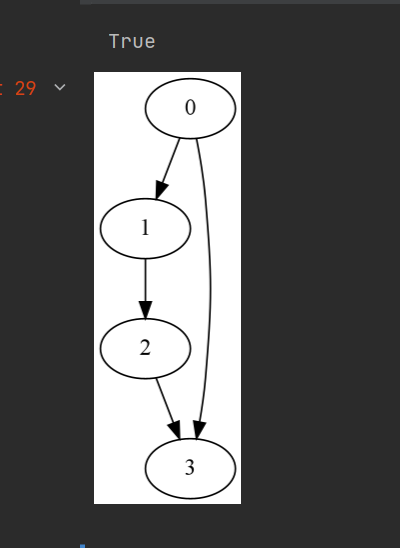




Activity4:

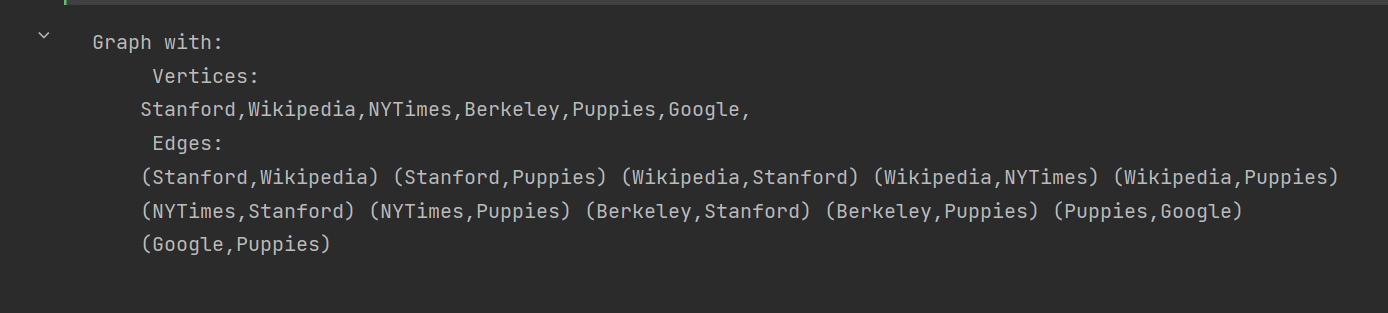
Build the graph:



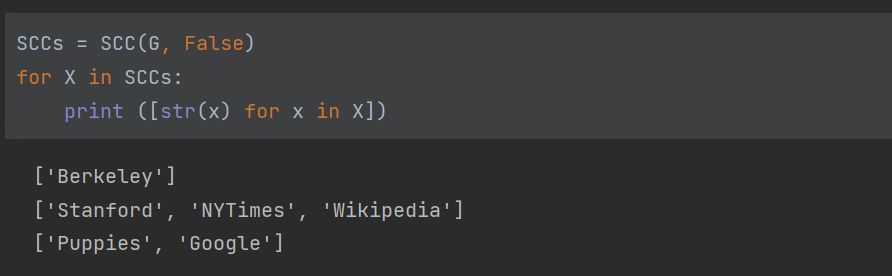


Activity5:

graph construction:



Strongly connected components of a graph:



Use DFS to calculate strongly connected components:



Another test example for computing connected components of a graph:

