Data Structures - Lab Worksheet

Graphs

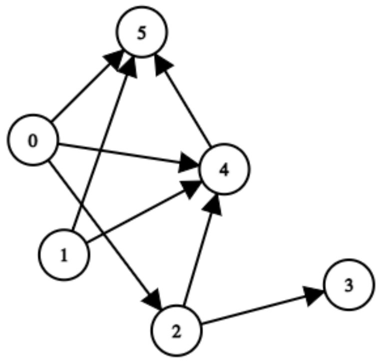
Start by downloading the provided coding canvas for graphs.

To use the visualization tools in `graph_output_tools.py`, you need to install three extra python libraries. First install Graphviz, then type the following command lines in your terminal:

```
$ pip install matplotlib
$ pip install networkx
$ pip install pygraphviz
```

You can browse the following example output of a full test run to see what it should look like once you've implemented all of the methods.

Question 1 - Adjacency Matrix



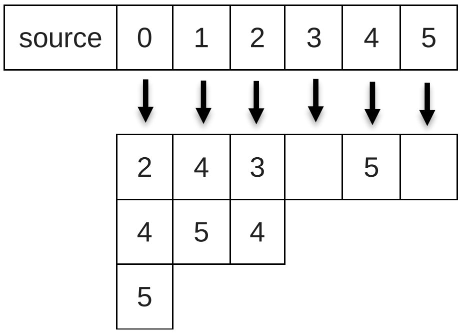
Give the adjacency matrix and the adjacency lists representation of the graph in the figure above.

Answer:

Adjacency Matrix:

destination / source	0	1	2	3	4	5
0	0	0	1	0	1	1
1	0	0	0	0	1	1
2	0	0	0	1	1	0
3	0	0	0	0	0	0
4	0	0	0	0	0	1
5	0	0	0	0	0	0

Adjacency Lists:



Q: Which do you think is better suited for representing dense graphs? What about sparse ones?

Answer:

For the dense graphs, Adjacency Matrix is better suited because even if many edges are present, the space complexity remains O(N^2), and accessing whether two nodes are connected is O(1) time complexity, which is efficient for dense graphs where many such checks might be necessary.

For the sparse graphs, Adjacency Lists is better suited. As its space complexity is O(N+E), where E is the number of edges, sparse graphs ensure that E is much less than N^2, so Adjacency Lists is much more space-efficient than Adjacency Matrix. For checking specific edge, its disadvantage against Adjacency Matrix is that it has O(D) time complexity where D represents a node's degree. But it will not be too much in the case of sparse graphs.

Question 2 - Graph exploration

Implement the breadth-first search and the depth-first search functions in module `graph_algorithms`.

Implement the breadth-first search iteratively (method `bfs`) and the depth-first search recursively (function `recursiveDFS` initiates the first recursion that then repeats in `recursive_dfs`).

Question 3 - Connectivity Graph

Implement the function `computeConnectivity` in module `graph_algorithms`. It takes in an existing graph G as argument, and outputs another graph G' which is the transitive closure of G.

Q: Why do the mesh and the strongly-connected graph have identical transitive closures?

Answer: This is because for both graphs, each node is connected to every other node.

Question 4 - Minimal Spanning Tree

Implement the function `computeMinimumSpanningTree` in module `graph_algorithms`. It takes in an existing graph G and a vertex identifier Root as arguments, and outputs another graph G' which is a minimal spanning tree of G with Root as the root of the tree.

Q: Can two different minimal spanning trees be associated with the same graph?

Answer: Yes. For example, BFS, DFS, level-order DFS (where we implemented in the coding), etc. can all be used to form different MST's for a graph. In general, MST is not unique.