Problem 3

My representation of a graph is a set of nodes and 2 maps of edges. The set of nodes represents all the nodes in the graph and the 2 maps of edges are for incoming edges and outgoing edges of nodes. Each map has a node as the key and a set of edges as the value, where the set of edges is the incoming/outgoing edges of the key node. This representation combines the adjacency list representation with a collection of nodes. While this isn't very space efficient (2x the total edges in a graph, 3x the total nodes in a graph), this makes the implementation of Graph somewhat easier and much more time efficient as a lot of Graph operations will use hashset/hashmap operations.

A collection of edges is space efficient and can be time efficient depending on the collection, but can make implementing certain operations difficult. An adjacency list is easy to implement, not as space efficient as a collection of edges, and isn't very time efficient if a list is used for the edges ($O(n)$ time to look for an edge in the list). An adjacency matrix is not space efficient ($O(n^2)$) but is very easy to implement and very time efficient. I ended up choosing my representation since I think it was a bit of a middle ground between space efficiency and time efficiency, as I didn't want to take $O(n^2)$ space but also didn't want to have inefficient operations.