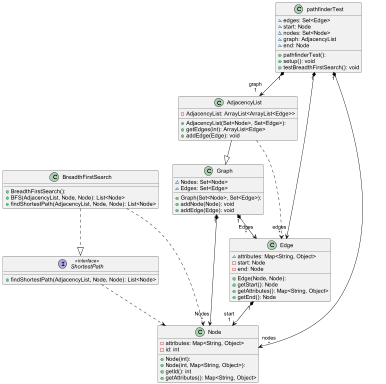
A4 Report

# Provide a class diagram of your pathfinder library



# Explain why your code is SOLID, and locate your technical debt. If you have used any pattern, describe them and justify your decision.

Single responsibility: Each class only does one thing, store data, test, calculate from data, etc..

Open-Closed: You cannot modify the functions or structures inside the classes but you can add other attributes to the graph, add other graph representations, or add other searching algorithms.

Liskov: You can get the edges from the graph and the nodes from the edges and do whatever you would normally do with an edge or node inside the outer object.

Interface: Client just interacts with the shortest path interface and does not care what the underlying algorithm is.

Dependency: Graph is abstract with the representations of the graph being the concrete object the user interacts with.

Technical Debt: Attributes are stored as a map of strings and values, meaning that the user would need to define a way to manage the keys without using a “magic” string to access the attributes each time. The edges are unweighted but adding weights and other algorithms to handle them would be easy too add on. More technical debt will likely reveal itself as I start to work with the library and I will either have to resolve it then.

# Explain why your tests are “reasonable”.

The testing methods check that the searching algorithm returns a path of the correct length with the correct start and end nodes. This tells us that the path is going between the right nodes and is of the correct length and not returning a longer path than is possible. First we test this with a basic graph, then we do it with the a boundary condition where the start and end are the same node, then we test where there is no possible path. This hits all the relevant parts of BICEP.