ReadMe:

# Introduction:

This is a project to demonstrate my coding capability towards solving the given coding challenge. Because the question in the challenge can be answered much efficiently by proof, I have given myself the goal to focus on demonstrating the way I would have approached this problem as a project.

# Project Description:

The Plan:

Initially the project is divided into three key area:

* Input/initialize
* Algorithm
* Display/Presentation

The idea is the three areas should be independent module and communicate via set interface so that any part can be maintained/upgraded independently (lots of benefits).

The Reality:

The project took about 10 hours, from start to finish, to a degree I’m satisfied as deliverable that it completes the goal outlined in the Introduction.

The project is implemented in iterations, each iteration there was a set goal to achieve, then next iteration builds on top of it, the following roughly describe each iteration in order as well as time spent.

Read and think about the question, coming up the design for read and storing the graph and two algorithm of choice--- 4 hours.

Coded Graph class for barebone read and forming triangle logic ---- 0.5 hour.

Validate correctness for Graph input ---- 0.5 hour

Coded Algorithm 1 and 2 and added necessary class (linkedNode.cs) and additional helper function in Graph.cs -----1.5 hour

Validate the correctness of both algorithm, debugging ---- 1 hour

Added additional code refactoring: logging, error handling, additional read option, better logic. ----0.5 hour

Documents and comments and finishing touch ---- 2 hour

The Display/Presentation part got scraped (but it was planned) because I was going to use DirectX to draw it on screen and do some animation to demonstrate the algorithm steps, it was definitely fun to build something like that but it is out of the scope and requires much more effort. So the Display strategy became “put a break points in the middle and read the data structure” Which became the essential step for validating correctness for prototyping.

High Lights:

The Highlight of this demonstration is:

* Highly adjustable data structure for storing graph (extensible)
* Color is represented as small prime numbers, so each type of the triangle has unique representation, it is also more extensible than hard encode each type of triangle. It can be useful in more areas (such as determining containment, reachability and so on)
* Datadriven, depending on the input (more colors, more nodes, although unfortunately triangle faces are enforced, but it can be changed by adding an interface layer IFace.cs), graph can look like differently yet other modules would still work.
* From above points you can see the project was built with enhancement in mind, if the question changed (graph changed, query changed) a lot of the codes in here can be reused easily.
* Due to the way we represent color as prime number, we have an easy way to compute permutation.
* Two algorithms were presented, a simple DFS and a brute force permutation. Simple pruning was incorporated.

Improvements:

* More Interface Usage, e.g. LinkedNode can be entirely convert into an interface, but in the future enhancement.
* More domain separation, some of the logic can belong to a separate class this way makes the base class more re-usable.
* Parallelization support, always fun to build. (and with this data structure, I believe it’s easy to make the enhancement)
* Better Logging. We have just a barebone logging since I was mostly focus on developing the data structure, given the timeframe.
* More increments in commits to show project progression.
* Not all edge cases are handled, for example, if the graph is empty, the program will fail. But the error catching was kept at minimum level for the scope of this project.

Unit Test:

Although some validation was done during development, it was based on spot checking. Accuracy and the correctness must be tested. The ideal unit tests involve the follows:

* Graph construction, count edges, nodes, colors, degrees.
* A simpler graph to validate the correctness of the algorithms, validate traversal sequence, number of visits per node.
* Edge graph or edge cases.
* Graph legality

Instruction:

To Install:

Unzip folder. Open with visual studio community 2017. Build project

To run:

Executable.

To view:

Basic output supported.

For source code see the source folder.

To Change Graph:

Modify Graph.cs to change basic settings such as number of colors.

See description in StaticGraph.cs for setting up the graph.

Either point to towards to a specific file or modify the StaticGraph.cs