1. Team Name: Polynomial Lovers
2. Login: cs170-nr

3. Team Members:

* + - * David He
      * Jongyoon Lee
      * Edward Kim
      * Corey Zumar

4. We took a multifaceted approach, opting to invoke two different algorithms based on the nature of the instance we encountered. The first of these algorithms was a branch and bound algorithm modeled after the one given in the Algorithms textbook for the standard TSP paths problem. In order to satisfy the color-based requirements of the NPTSP problem, the branch and bound algorithm ceases to investigate a potential solution when it encounters a “BBBB” or “RRRR” sequence. Additionally, our algorithm makes use of a stack so as to facilitate a depth-first approach that allows the best discovered solution to be updated as quickly as possible after it commences.

The second algorithm we employed is a greedy algorithm that begins at any starting vertex we choose and greedily chooses the smallest edge to another vertex that is not yet part of the path. We then iteratively add vertices to build our path, adding the vertex with the smallest edge. If it is the case that we have added three vertices of the same color, then we force the algorithm to add vertices of the opposite color to avoid violating the non-partisan rule of visiting four consecutive colored vertices. Then, we keep doing this until all vertices are part of the path.

5. Our input files were generated at random. We used a Java API (JGraphT) to write a random graph generator that outputs graphs conforming to specific properties. Specifically, we erred on the side of using more vertices and edges, and we ensured that our graphs were not metric to avoid exploitation of metric approximation algorithms by the other teams.

6. The source code is divided into three distinct packages: a general graph tools and file I/O package, a package for the branch and bound algorithm, and a package for the greedy algorithm.

In order to parse an instance into a graph that can be manipulated as well as get the sizes of all instances for the purpose of executing the branch and bound algorithm in certain cases, the InstanceProcessor is critical. Additionally, the ExactSolverExecutor can be instantiated in order to execute the branch and bound algorithm on a subset of the instances specified by the “minSize” and “maxSize” parameters. Please refer to the **runExactSolver** method in the **Main** class for a working code sample.

For the greedy algorithm, we instantiate a GreedySolver instance that takes a graph previously created by InstanceProcessor as an input. Then, we can call the method **solveGreedily** to start solving this graph greedily. This method passes a pre-determined starting vertex as a parameter to the **executeGreedyAlgorithm** method. This parameter can be modified so that the algorithm starts on a different vertex.

7. Resources used:

* + - * JGraphT (A Java API for graphs) - <http://jgrapht.org/>