Notes on my approximation implementation:

- Strategy
  - I chose to go with a greedy approach
  - The "graph" that is passed in is a dictionary of lists
    - Example:

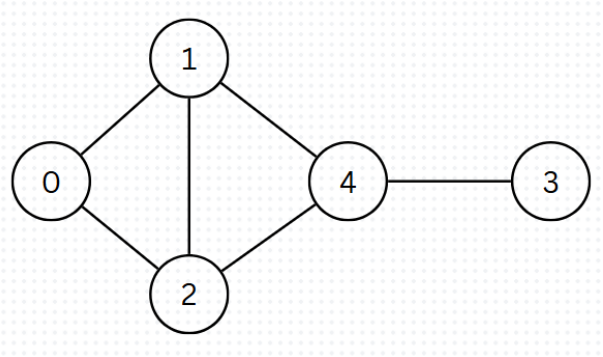
      {"A": ["B", "C", "D"],
      "B": ["A", "C", "D"],
      "C": ["A", "B", "D"],
      "D": ["A, "B", "C"]
      }
  - Starts by initializing a dictionary that is the length of how many vertices there are
    - This is to hold the color that each vertex gets set to
    - Initializes each color to -1 to represent that it has not been assigned a color
  - Set the very first vertex to color "0"
  - Creates a parallel list that will represent colors that have not yet been taken
  - Then loops through every vertex in the graph
    - First check to make sure that the vertex is set to -1
    - Runs through the list associated with each vertex representing the vertices neighbors
      - If that vertices neighbors are already colored, the program will mark that specific index to "False"
    - Once all neighbors have been checked (and all unavailable colors marked "False"), the program will iterate through the available_colors list and assign the first available color to that vertex
  - This makes the best decision that can be made in the moment, which makes this program run accurately and quickly

- Analytical Runtime
  - The runtime I have deduced is $O(V^2)$
    - The outer loop iterates over each vertex, which takes O(V)
    - For each vertex, we create a list of available colors, which takes O(V) space
      - The inner loop checks the colors of adjacent vertices, which can take O(E) time in the worst case, where E is the number of edges.

        - Total time spent in this part is O(V + E) for each vertex, leading to an overall complexity of O(V * (V + E)).
    - O(V*(V+E)) = O(V^2 + VE)
        - V^2 is the dominant term so our complexity is **O(V^2)**

- Non-optimal results
    - This took me a long time to figure out and it wasn't until I looked into this geeksforgeeks article for it to click
        - Article: https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/
    - The order in how the vertices are important
        - The greedy algorithm makes the best decision for itself in the moment and takes into account every neighbor
        - To find a non-optimal answer, I had to think through in what order will this fail
    - The input in our repo:

      6
      b a
      b d
      d a
      c e
      e a
      e d
    - Images of the graph that this represents:



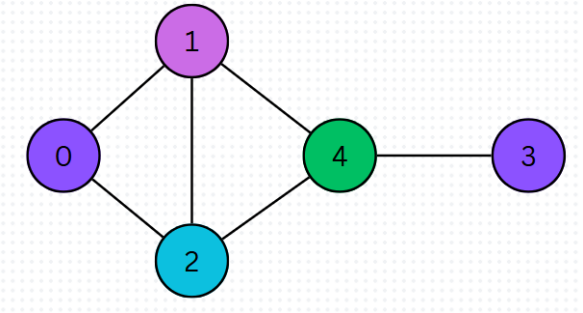
    - Note: yes I know the vertices are labeled as numbers instead of letters like the input, I drew it out on paper and messed with the text files so this is how we ended up with this

- Input (as represented on the graph)

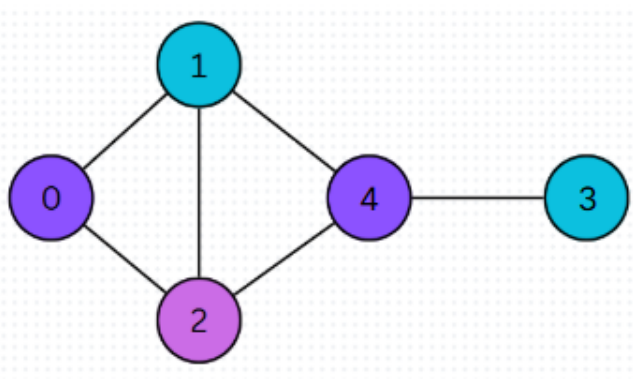
  6
  2 1
  2 2
  2 1
  3 4
  4 1
  4 2

- The optimal answer used 3 colors, however, with this specific input, my approximation algorithm uses 4 colors (pictures below)

- Approx:



- Exact:



- Once we figured this out, it was much easier to find inputs that show the faults of this approximation algorithm
  - These are showcased in our presentation in the “Augmented Solution” portion