# Min Graph Coloring Approximation Notes

**Explain Strategy:**

The approximate solution used uses a greedy algorithm.

- This greedy solution visits the vertices one by one.
- At each vertex, the algorithm assigns the smallest color number not used by its colored neighbors.
- This creates a valid coloring of the graph, but most likely not the best solution.

This solution's answer solely depends on the order the vertices are visited, so I added an element of randomness to it.

- I repeat the coloring algorithm a set number of times.
- Each time randomizing the order of the vertices I visit.
- The program keeps the best solution after all the iterations.

**Analytical Runtime:**

n = number of vertices
m = number of edges
T = number of iterations (right now I use 100)

1. Reading the graph (read_graph):
    - Reading in m edge lines and doing dictionary lookups costs O(m) amortized because the dictionary operations are O(1).
    - Building the adjacency list costs O(n), which makes the total cost O(n + m).
2. One iteration of greedy_coloring_random_order:
    - First it randomly shuffles the visiting order which costs O(n).
    - Loop through vertices in random order. For each vertex:
        - Get neighbors colors: costs O(degree(v))
        - O(degree(v)) = O(2m) = O(m)
    - Full Cost: O(n + m)
3. Repeating for T iterations:
    - Each iteration cost O(n + m)
    - Total with iterations O(T * (n + m))
4. Final printing: O(n)

Overall Runtime: O(T * (n + m))