NOTES for the Map coloring

Map coloring algorithm is a way to color regions on a map such that no two adjacent regions have the same color. This algorithm has many practical applications, such as scheduling tasks, scheduling exams, and assigning frequencies to cell towers. The map coloring problem is an example of a graph coloring problem. In this problem, we represent the map as a graph, where each region is a node and two nodes are connected if their corresponding regions share a border. The simplest approach to solve the map coloring problem is to use a brute-force algorithm, which tests all possible color combinations until it finds a valid one. However, this method can be very time-consuming and impractical for large maps. A more efficient algorithm is the greedy algorithm, which assigns colors to nodes one by one, based on the colors of its neighbors. The greedy algorithm chooses the smallest possible color that is not used by its neighbors. While the greedy algorithm is fast, it does not always find the optimal solution. In some cases, it may produce a suboptimal coloring with more colors than necessary. Other variations of map coloring algorithm include backtracking and constraint satisfaction algorithms. These methods can be more effective than the greedy algorithm for certain maps but may be more complex to implement.

An example of usage of the map coloring, coloring US map

```
import geopandas as gpd
import matplotlib.pyplot as plt
import random

# Load US state boundaries using geopandas
us_states = gpd.read_file('https://www2.census.gov/geo/tiger/GENZ2018/shp/cb_2018_us_state_500k.zip')

# Define a list of colors
colors = ['#FFC0CB', '#008000', '#FFD700', '#00FFFF', '#8B0000', '#ADD8E6', '#FFA500', '#000080']

# Assign an initial color to each state
us_states['color'] = random.choice(colors)

# Loop through each state
for index, row in us_states.iterrows():
    # Get a list of all neighboring states
    neighbors = us_states[us_states.geometry.touches(row.geometry)]
    # Check the color of all neighboring states
    neighbor_colors = set(neighbors['color'])
    # If a neighboring state has the same color, assign a different color to the current state
    if row['color'] in neighbor_colors:
```

```
            unused_colors = set(colors) - neighbor_colors
            if len(unused_colors) > 0:
                us_states.at[index, 'color'] = random.choice(list(unused_colors))

    # Plot the US map with colored states
    us_states.plot(column='color', cmap='tab20', linewidth=0.5, edgecolor='grey', figsize=(20, 10))
    plt.axis('off')
    plt.show()
```

The code is an implementation of a map coloring algorithm to color US states using Python packages like geopandas and matplotlib. The map coloring problem involves coloring regions (in this case, US states) on a map in such a way that no two adjacent regions have the same color.

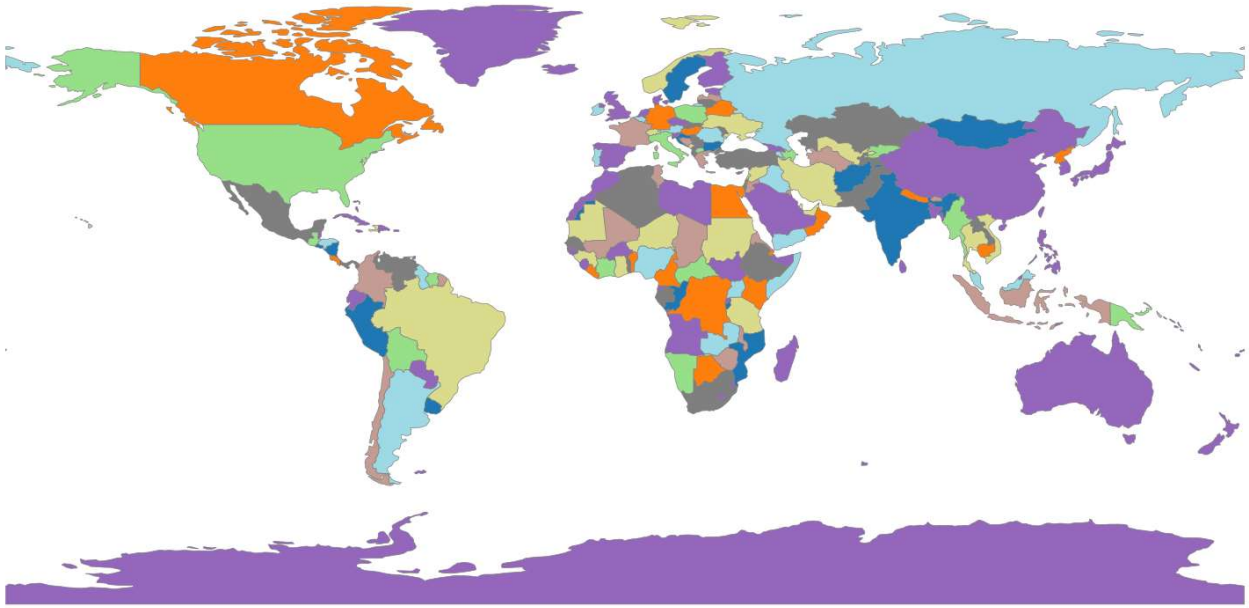
Here's what the code does step-by-step:

It loads the US state boundaries data from a URL using geopandas. It defines a list of colors to use for coloring the states. It assigns an initial random color to each state. It loops through each state to check if any neighboring state has the same color. If a neighboring state has the same color, it assigns a different color to the current state. Finally, it plots the US map with colored states using matplotlib. The map coloring algorithm used in this code is a simple one. It starts with assigning random colors to each state and then iteratively checks if any neighboring state has the same color. If yes, it assigns a different color to the current state. This process continues until no two adjacent states have the same color.

Overall, this code is a simple example of how map coloring can be implemented using Python packages. However, for larger maps with more regions, more complex algorithms may be needed to ensure optimal coloring.

Another example where the world map is colored now

```
import geopandas as gpd
import matplotlib.pyplot as plt
import random

# Load US state boundaries using geopandas
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

# Define a list of colors
colors = ['#FFC0CB', '#008000', '#FFD700', '#00FFFF', '#8B0000', '#ADD8E6', '#FFA500', '#000080']

# Assign an initial color to each state
world['color'] = random.choice(colors)

# Loop through each state
for index, row in world.iterrows():
    # Get a list of all neighboring states
    neighbors = world[world.geometry.touches(row.geometry)]
    # Check the color of all neighboring states
    neighbor_colors = set(neighbors['color'])
    # If a neighboring contry has the same color, assign a different color to the current state
    if row['color'] in neighbor_colors:
        unused_colors = set(colors) - neighbor_colors
        if len(unused_colors) > 0:
            world.at[index, 'color'] = random.choice(list(unused_colors))

# Plot the US map with colored states
world.plot(column='color', cmap='tab20', linewidth=0.5, edgecolor='grey', figsize=(20, 10))
plt.axis('off')
plt.show()
```



This does the same thing as done by the earlier code snippet except that, this does it for the world map