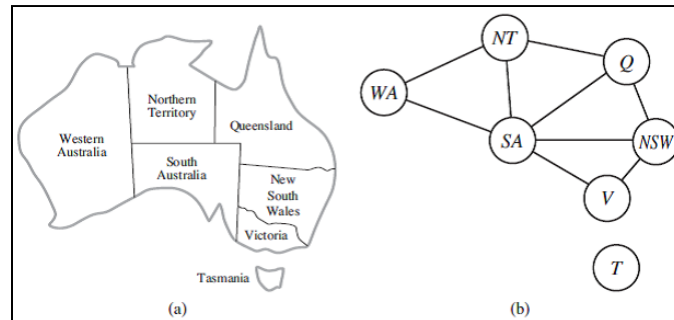


Harshit Aggarwal
RA1911003010782
Batch-D2

Lab-2 Map-Colouring Problem (CSP)

Problem : To color the regions of the given map such that no two adjacent states have the same color.
The states are the variables and the colors are the domains.



We will convert this CSP to a graph coloring problem. Depth first search is apt as the path by which solution should be reached is irrelevant.

Source Code:

```
colors = ['Red', 'Blue', 'Green']
```

```
states = ['wa', 'nt', 'sa', 'q', 'nsw', 'v']
```

```
neighbors = {}                                #adjacent pairing neighbors of different states
neighbors['wa'] = ['nt', 'sa']
neighbors['nt'] = ['wa', 'sa', 'q']
neighbors['sa'] = ['wa', 'nt', 'q', 'nsw', 'v']
neighbors['q'] = ['nt', 'sa', 'nsw']
neighbors['nsw'] = ['q', 'sa', 'v']
neighbors['v'] = ['sa', 'nsw']
```

```
colors_of_states = {}
```

```
def promising(state, color):    #function to check a promising color - returns a promising color
    for neighbor in neighbors.get(state):
        color_of_neighbor = colors_of_states.get(neighbor)
        if color_of_neighbor == color:    #same color (of neighbor and state) -> rejected
            return False

    return True                #if not same -> color accepted
```

```
def get_color_for_state(state):    #promising color is assigned to the state
    for color in colors:
        if promising(state, color):
            return color
```

```
def main():
    for state in states:
        colors_of_states[state] = get_color_for_state(state)

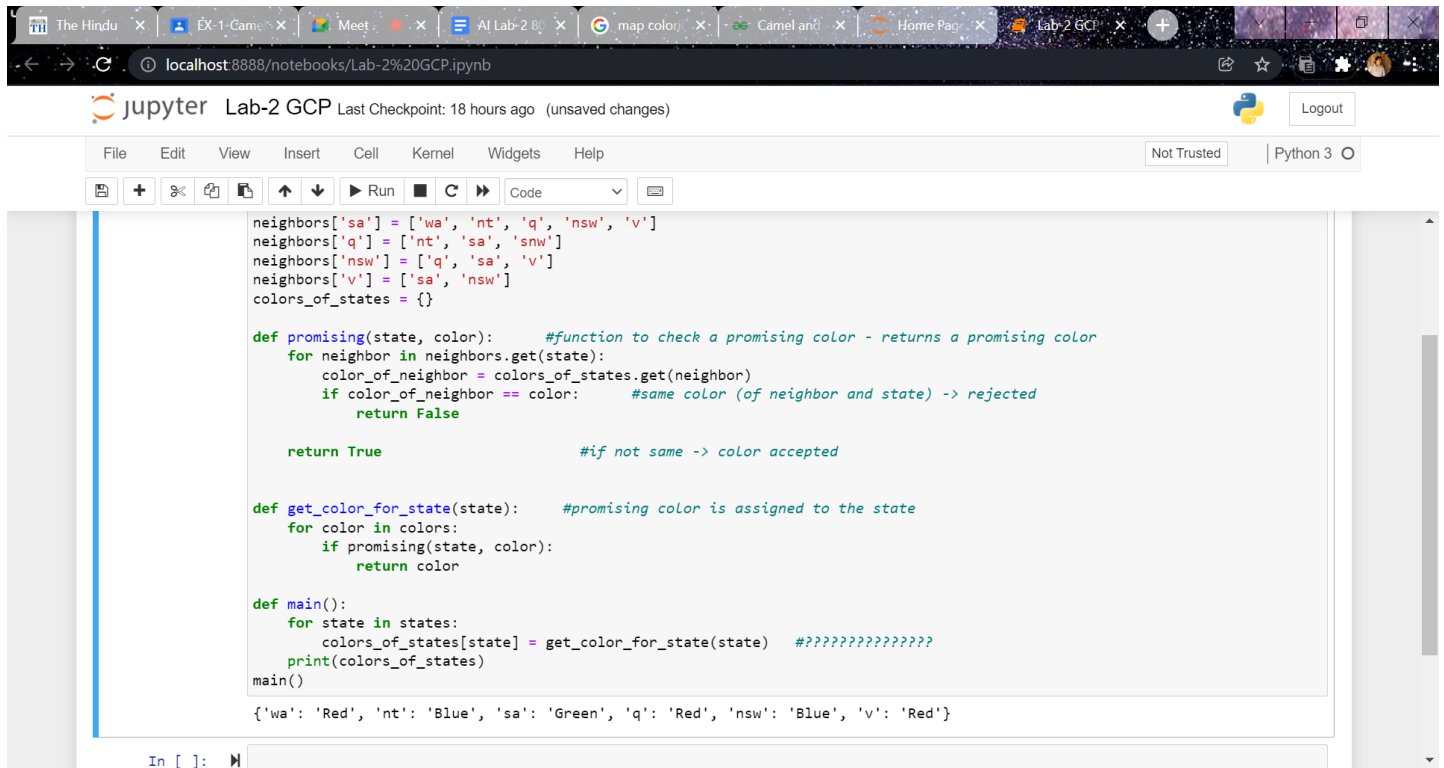
    print(colors_of_states)
```

```
main()
```

Result :

```
{'wa': 'Red', 'nt': 'Blue', 'sa': 'Green', 'q': 'Red', 'nsw': 'Blue', 'v': 'Red'}
```

Output Screenshot



The screenshot shows a JupyterLab interface in a web browser. The browser's address bar displays `localhost:8888/notebooks/Lab-2%20GCP.ipynb`. The JupyterLab header includes the logo, the notebook name "Lab-2 GCP", and a status message "Last Checkpoint: 18 hours ago (unsaved changes)". A "Logout" button is visible in the top right. The main menu bar contains "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu is a toolbar with icons for file operations, running the cell, and other functions. The notebook area displays a Python code cell with the following code:

```
neighbors['sa'] = ['wa', 'nt', 'q', 'nsw', 'v']
neighbors['q'] = ['nt', 'sa', 'nsw']
neighbors['nsw'] = ['q', 'sa', 'v']
neighbors['v'] = ['sa', 'nsw']
colors_of_states = {}

def promising(state, color):    #function to check a promising color - returns a promising color
    for neighbor in neighbors.get(state):
        color_of_neighbor = colors_of_states.get(neighbor)
        if color_of_neighbor == color:    #same color (of neighbor and state) -> rejected
            return False

    return True    #if not same -> color accepted

def get_color_for_state(state):    #promising color is assigned to the state
    for color in colors:
        if promising(state, color):
            return color

def main():
    for state in states:
        colors_of_states[state] = get_color_for_state(state)    #####
    print(colors_of_states)
main()

{'wa': 'Red', 'nt': 'Blue', 'sa': 'Green', 'q': 'Red', 'nsw': 'Blue', 'v': 'Red'}
```

At the bottom of the interface, the prompt `In []:` is visible next to an input field.