ITCS 6150 – Intelligent Systems Project Report

Title: Map Coloring Problem

Members: Amit Shetty

Sourav Roy Choudhury

Chaitanya Kintali

Vamshi Goud

Date: 04-23-2019

Purpose:

- Compute the chromatic number of USA and Australia map.
- Experiment the CSP with the following methods
    - Depth first search only
    - Depth first search + forward checking
    - Depth first search + forward checking + propagation through singleton domains

- The order of variables needs to be defined in the following order MRV, Degree Constraint, and Least Constraining Value
- Present the results in a tabular format recording number of backtracking happened and the time to compute the result.

Project Details:

Chromatic Number – The Chromatic Number of a graph is the least number of colors needed to color the vertices of graph (G), such that no adjacent vertices have the same exact color.

Constraint satisfaction problem is the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy. Let's consider United states map, for a CSP the states are a set of variables, their domains are the four colors and the constraints are the restrictions of the neighboring states color.

The program is a collection of graph coloring algorithms for coloring the entire map of USA and AUSTRALIA. Such that no two states have same color.

Application Technical Details:

Programming Language: **Python**

Classes Used:

Graph

1) Attributes:

V = Number of Vertices / States for the Country

America = 50

Australia = 7

Graph = Adjacent Matrix with country adjacency mappings represented as 1.

stateDictionary: This variable holds the mapping between the numbers and the short hand notations of the state's.

Example: stateDictionary["1"] = "AL"

colorDictionary: This variable holds the mapping between the numbers and the colours.

Example: colorDictionary = {"1":"red", "2":"green", "3":"yellow", "4":"blue"}

ResultDictionary: ResultDictionary holds the mappings between the state's short hand notation's and the assigned color. \

Example: {"AL":"red"}

DomainDictionary: DomainDictionary holds the mappings between the country code and the list of available domain variables(COLORS).

[1:[1,2,3,4]]

2) Methods:

isSafe(self, v, colour, c):

This method takes the current state's colors assignment and checks for the current state colour assignment and returns true or false if it can be assigned or not.

graphColourUtil(self, m, colour, v):

This method makes recursive calls to itself. It has the heart logic of the map coloring. It checks for the break condition if all the states are coloured and returns true if all states are colored and breaks the recursive calls. This method also assigns colours to next states by either using heuristic functions / without-heuristic functions randomly selecting states. It also includes the logic of forward checking, backtracking, singleton, and using heuristics to choose which state to color next.

In Forward checking, once the colour assignment turns out to be incomplete for its children, it reset's the colours the current state of its domain variables.

getTheNeighbors(self, state):

This method returns the list of neighbouring states which aren't colored.

createdomainDictionary():

This method creates the DomainDictionary with key as state number, and value is initially with all the list of colours.

checkIfAllStatesColored(self, colors):

This methods checks if all the state's are assigned with colors.

checkSingletonConstraints(self, v):

This method checks and returns if any states have singleton states.

SingletonRemoveDomainVariables(self, v):

This method removes the currently assigned color from the current assigned state domain dictionary and reassigns the color to the adjacent state color's after backtracking.

SingletonHeartLogic(self, listofsingletonstates):

SingletonHeartLogic handles the core logic for singleton, by randomly selecting and returning the singleton state.

***MRV (self, domainDictionary, colours)*:**

This method returns the list of all states which has the least remaining values.

DegreeConstraint(self, domainDictionary, colours):

This method returns the list of the states with the state having the highest number of connection's and having most number of adjacent states.

LCV(self, domainDictionary, colours):

This method returns the list of the states with the state having the least number of connection's and having the least number of the adjacent states.

getTheNextState(self, domainDictionary, colours):

```
def getTheNextState(self, domainDictionary, colours):
    NextState = 0

    nextMRVStates = self.MRV(domainDictionary, colours)
    nextDCStates = self.DegreeConstraint(domainDictionary, colours)
    nextLCVStates = self.LCV(domainDictionary, colours)

    if (len(nextMRVStates)==1):
        NextState = nextMRVStates[0]
    elif(nextDCStates!=-1):
        NextState = nextDCStates
    else:
        NextState = nextLCVStates

    return NextState
```

This method returns the next state which is selected based upon the heuristic functions in the order

MRV

Degree Constraint

LCV

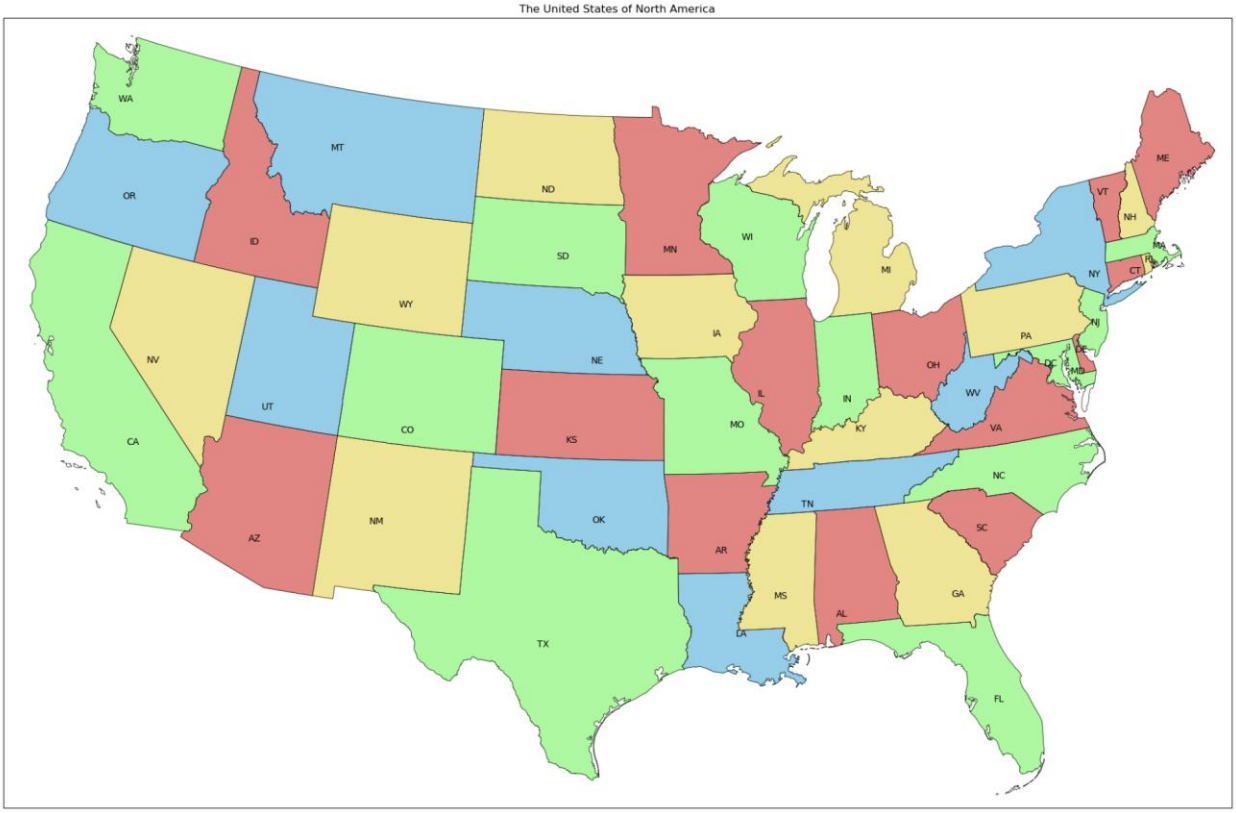
OUTPUT:

For America (Non Heuristic):

Dictionary –

For Singleton

```
WY ==> yellow
[Souravs-Air:Without_Heuristic souravroychoudhury$ python Singleton.py
 Solution exist and Following are the assigned colours:
 THE TOTAL TIME TOOK FOR EXEC ---------------- 482.45155692100525
 AL ==> red
 AK ==> red
 AZ ==> red
 AR ==> red
 CA ==> green
 CO ==> green
 CT ==> red
 DE ==> red
 FL ==> green
 GA ==> yellow
 HI ==> red
 ID ==> red
 IL ==> red
 IN ==> green
 IA ==> yellow
 KS ==> red
 KY ==> yellow
 LA ==> blue
 ME ==> red
 MD ==> green
 MA ==> green
 MI ==> yellow
 MN ==> red
 MS ==> yellow
 MO ==> green
 MT ==> blue
 NE ==> blue
 NV ==> yellow
 NH ==> yellow
 NJ ==> green
 NM ==> yellow
 NY ==> blue
 NC ==> green
 ND ==> yellow
 OH ==> red
 OK ==> blue
 OR ==> blue
 PA ==> yellow
 RI ==> yellow
 SC ==> red
 SD ==> green
 TN ==> blue
 TX ==> green
 UT ==> blue
 VT ==> red
 VA ==> red
 WA ==> green
 WV ==> blue
 WI ==> green
 WY ==> yellow
 Souravs-Air:Without_Heuristic souravroychoudhury$
```

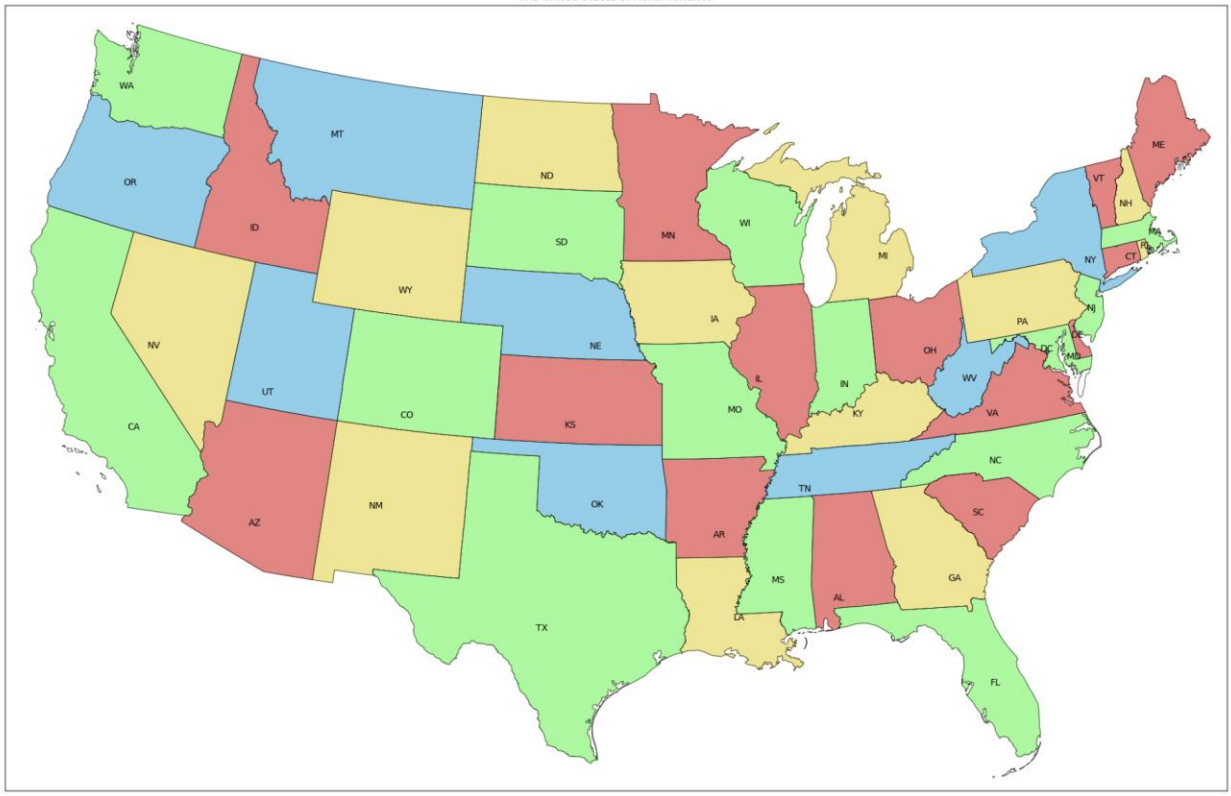

The United States of North America
WA
OR
CA
ID
NV
MT
WY
UT
AZ
CO
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MS
MI
IN
KY
TN
AL
OH
WV
GA
FL
SC
NC
VA
PA
NY
VT
NH
ME
MA
RI
CT
NJ
DE
MD
DC

Dictionary:

For Backtracking

```
BackTracking.py          ForwardChecking.py      Singleton.py
[Souravs-Air:Without_Heuristic souravroychoudhury$ python BackTracking.py
 Solution exist and Following are the assigned colours:
 THE TOTAL TIME TOOK FOR EXEC ---------------- 129.84686183929443
 AL ==> red
 AK ==> red
 AZ ==> red
 AR ==> red
 CA ==> green
 CO ==> green
 CT ==> red
 DE ==> red
 FL ==> green
 GA ==> yellow
 HI ==> red
 ID ==> red
 IL ==> red
 IN ==> green
 IA ==> yellow
 KS ==> red
 KY ==> yellow
 LA ==> yellow
 ME ==> red
 MD ==> green
 MA ==> green
 MI ==> yellow
 MN ==> red
 MS ==> green
 MO ==> green
 MT ==> blue
 NE ==> blue
 NV ==> yellow
 NH ==> yellow
 NJ ==> green
 NM ==> yellow
 NY ==> blue
 NC ==> green
 ND ==> yellow
 OH ==> red
 OK ==> blue
 OR ==> blue
 PA ==> yellow
 RI ==> yellow
 SC ==> red
 SD ==> green
 TN ==> blue
 TX ==> green
 UT ==> blue
 VT ==> red
 VA ==> red
 WA ==> green
 WV ==> blue
 WI ==> green
 WY ==> yellow
 Souravs-Air:Without_Heuristic souravroychoudhury$
```

The United States of North America

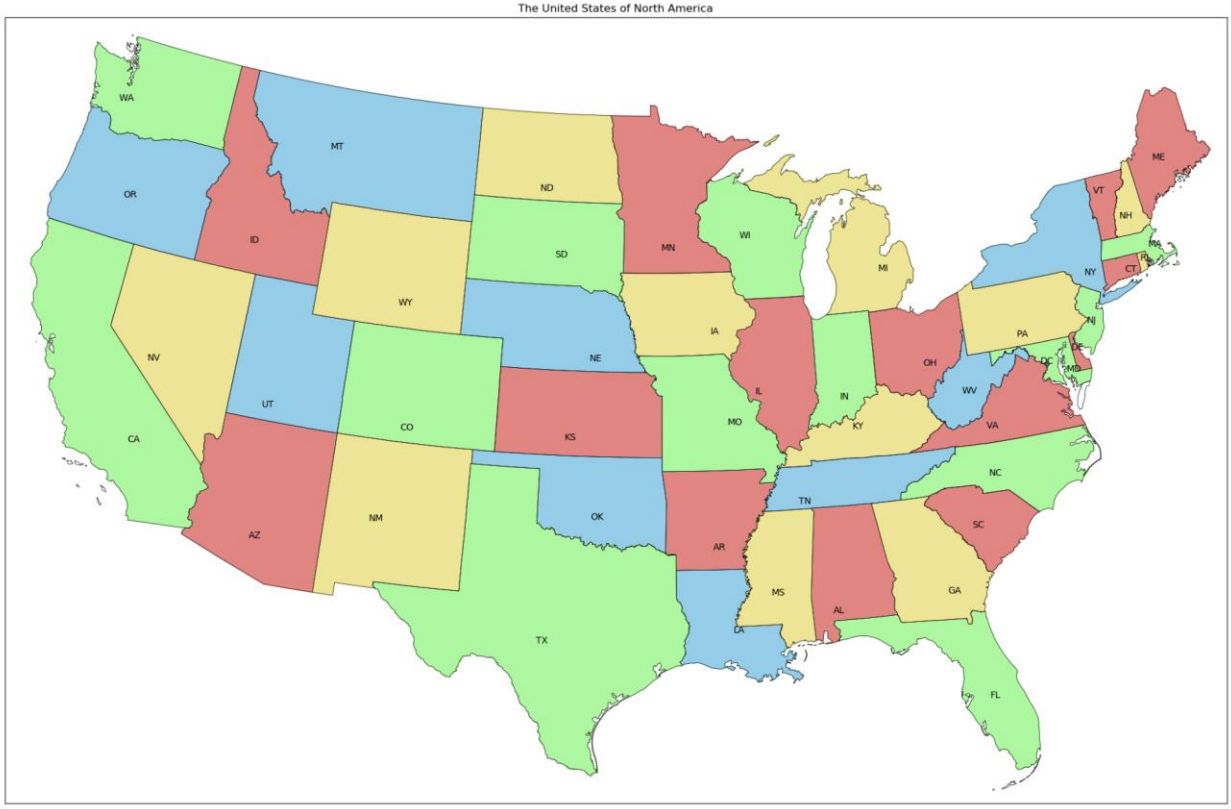


Dictionary:

For Forward Checking

```
Souravs-Air:Without_Heuristic souravroychoudhury$ python ForwardChecking.py
Solution exist and Following are the assigned colours:
THE TOTAL TIME TOOK FOR EXEC ---------------- 468.2438578605652
AL ==> red
AK ==> red
AZ ==> red
AR ==> red
CA ==> green
CO ==> green
CT ==> red
DE ==> red
FL ==> green
GA ==> yellow
HI ==> red
ID ==> red
IL ==> red
IN ==> green
IA ==> yellow
KS ==> red
KY ==> yellow
LA ==> blue
ME ==> red
MD ==> green
MA ==> green
MI ==> yellow
MN ==> red
MS ==> yellow
MO ==> green
MT ==> blue
NE ==> blue
NV ==> yellow
NH ==> yellow
NJ ==> green
NM ==> yellow
NY ==> blue
NC ==> green
ND ==> yellow
OH ==> red
OK ==> blue
OR ==> blue
PA ==> yellow
RI ==> yellow
SC ==> red
SD ==> green
TN ==> blue
TX ==> green
UT ==> blue
VT ==> red
VA ==> red
WA ==> green
WV ==> blue
WI ==> green
WY ==> yellow
Souravs-Air:Without_Heuristic souravroychoudhury$
```

The United States of North America



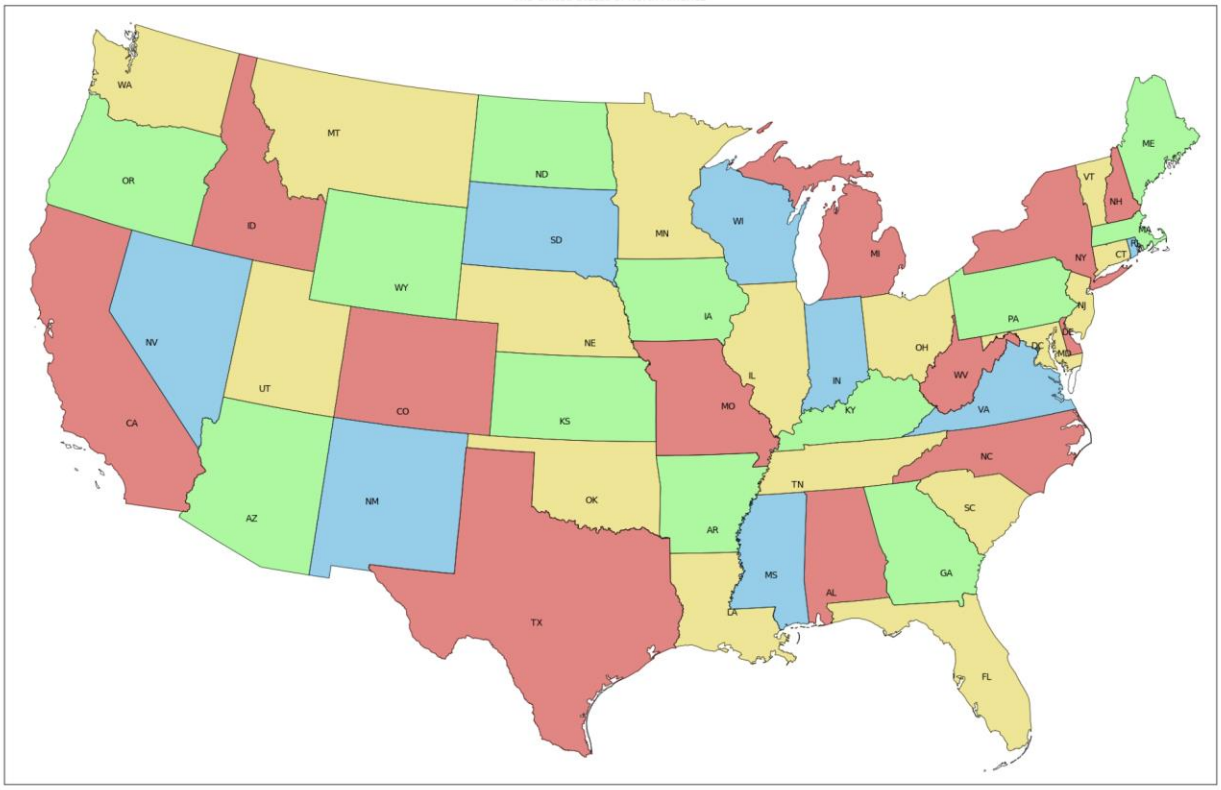
With Heuristics:

Dictionary –

For Singleton

```
Souravs-Air:heuristic souravroychoudhury$ python SingletonHeuristic.py
Solution exist and Following are the assigned colours:
THE TOTAL TIME TOOK FOR EXEC ---------------- 0.018642187118530273
AL ==> red
AK ==> red
AZ ==> green
AR ==> green
CA ==> red
CO ==> red
CT ==> yellow
DE ==> red
FL ==> yellow
GA ==> green
HI ==> red
ID ==> red
IL ==> yellow
IN ==> blue
IA ==> green
KS ==> green
KY ==> green
LA ==> yellow
ME ==> green
MD ==> yellow
MA ==> green
MI ==> red
MN ==> yellow
MS ==> blue
MO ==> red
MT ==> yellow
NE ==> yellow
NV ==> blue
NH ==> red
NJ ==> yellow
NM ==> blue
NY ==> red
NC ==> red
ND ==> green
OH ==> yellow
OK ==> yellow
OR ==> green
PA ==> green
RI ==> blue
SC ==> yellow
SD ==> blue
TN ==> yellow
TX ==> red
UT ==> yellow
VT ==> yellow
VA ==> blue
WA ==> yellow
WV ==> red
WI ==> blue
WY ==> green
Souravs-Air:heuristic souravroychoudhury$
```

The United States of North America

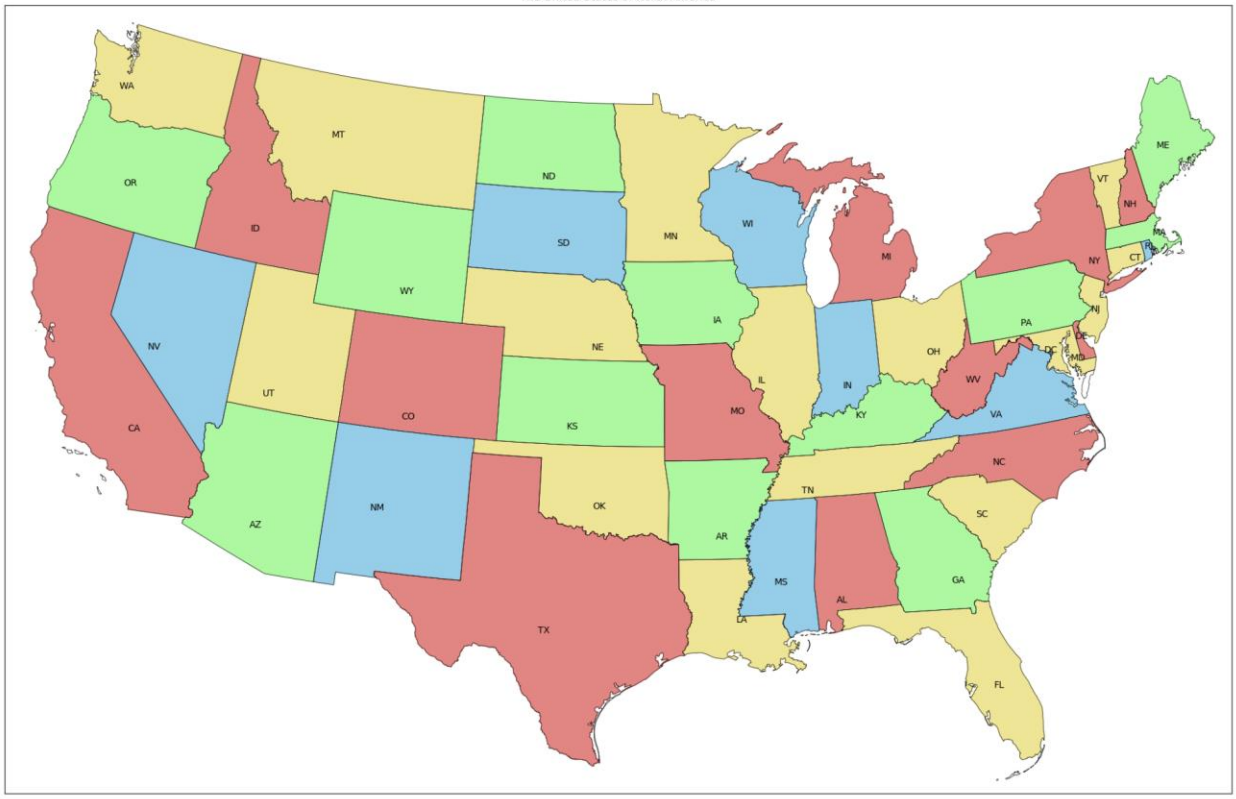


Dictionary –

For BackTracking

```
Souravs-Air:heuristic souravroychoudhury$ python BackTrackingHeuristic.py
Solution exist and Following are the assigned colours:
THE TOTAL TIME TOOK FOR EXEC ---------------- 0.01684093475341797
AL ==> red
AK ==> red
AZ ==> green
AR ==> green
CA ==> red
CO ==> red
CT ==> yellow
DE ==> red
FL ==> yellow
GA ==> green
HI ==> red
ID ==> red
IL ==> yellow
IN ==> blue
IA ==> green
KS ==> green
KY ==> green
LA ==> yellow
ME ==> green
MD ==> yellow
MA ==> green
MI ==> red
MN ==> yellow
MS ==> blue
MO ==> red
MT ==> yellow
NE ==> yellow
NV ==> blue
NH ==> red
NJ ==> yellow
NM ==> blue
NY ==> red
NC ==> red
ND ==> green
OH ==> yellow
OK ==> yellow
OR ==> green
PA ==> green
RI ==> blue
SC ==> yellow
SD ==> blue
TN ==> yellow
TX ==> red
UT ==> yellow
VT ==> yellow
VA ==> blue
WA ==> yellow
WV ==> red
WI ==> blue
WY ==> green
Souravs-Air:heuristic souravroychoudhury$
```

The United States of North America

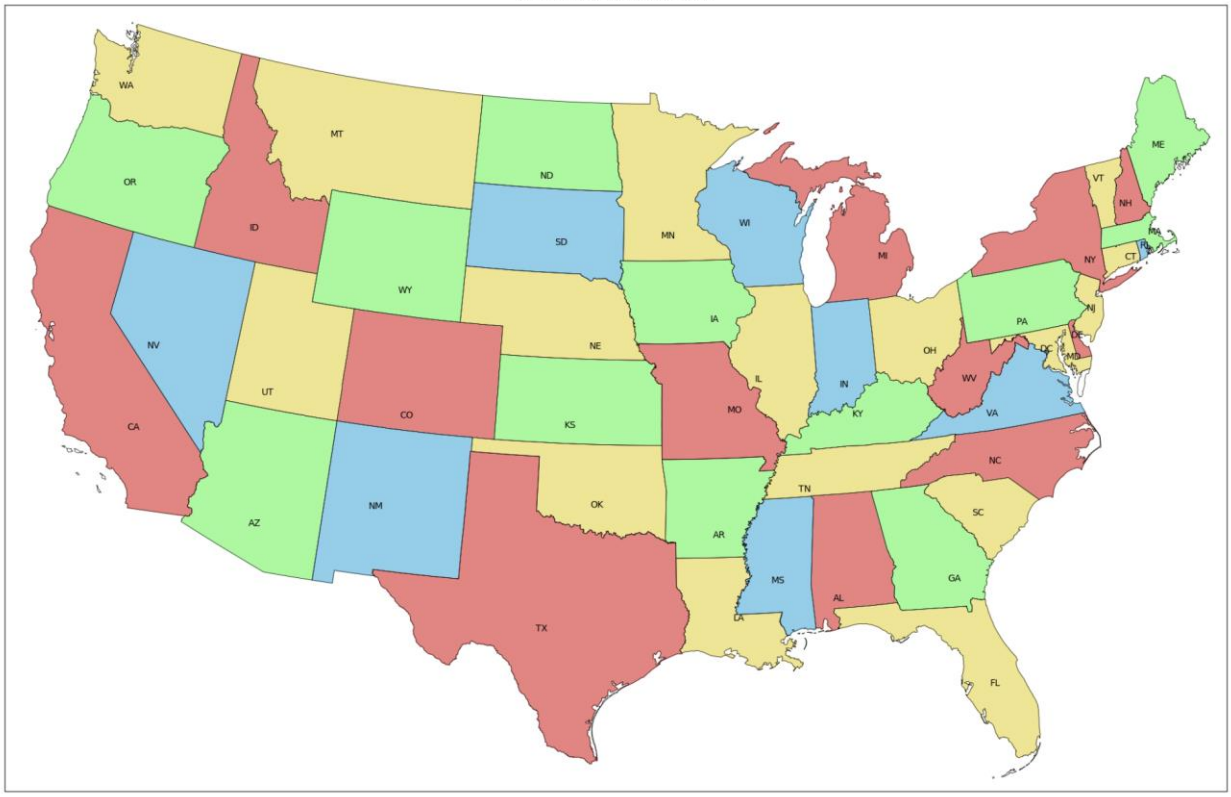


Dictionary –

For Forward Checking

```
[Souravs-Air:heuristic souravroychoudhury$ python ForwardCheckingHeuristic.py
Solution exist and Following are the assigned colours:
THE TOTAL TIME TOOK FOR EXEC ---------------- 0.017826080322265625
AL ==> red
AK ==> red
AZ ==> green
AR ==> green
CA ==> red
CO ==> red
CT ==> yellow
DE ==> red
FL ==> yellow
GA ==> green
HI ==> red
ID ==> red
IL ==> yellow
IN ==> blue
IA ==> green
KS ==> green
KY ==> green
LA ==> yellow
ME ==> green
MD ==> yellow
MA ==> green
MI ==> red
MN ==> yellow
MS ==> blue
MO ==> red
MT ==> yellow
NE ==> yellow
NV ==> blue
NH ==> red
NJ ==> yellow
NM ==> blue
NY ==> red
NC ==> red
ND ==> green
OH ==> yellow
OK ==> yellow
OR ==> green
PA ==> green
RI ==> blue
SC ==> yellow
SD ==> blue
TN ==> yellow
TX ==> red
UT ==> yellow
VT ==> yellow
VA ==> blue
WA ==> yellow
WV ==> red
WI ==> blue
WY ==> green
Souravs-Air:heuristic souravroychoudhury$
```

The United States of North America



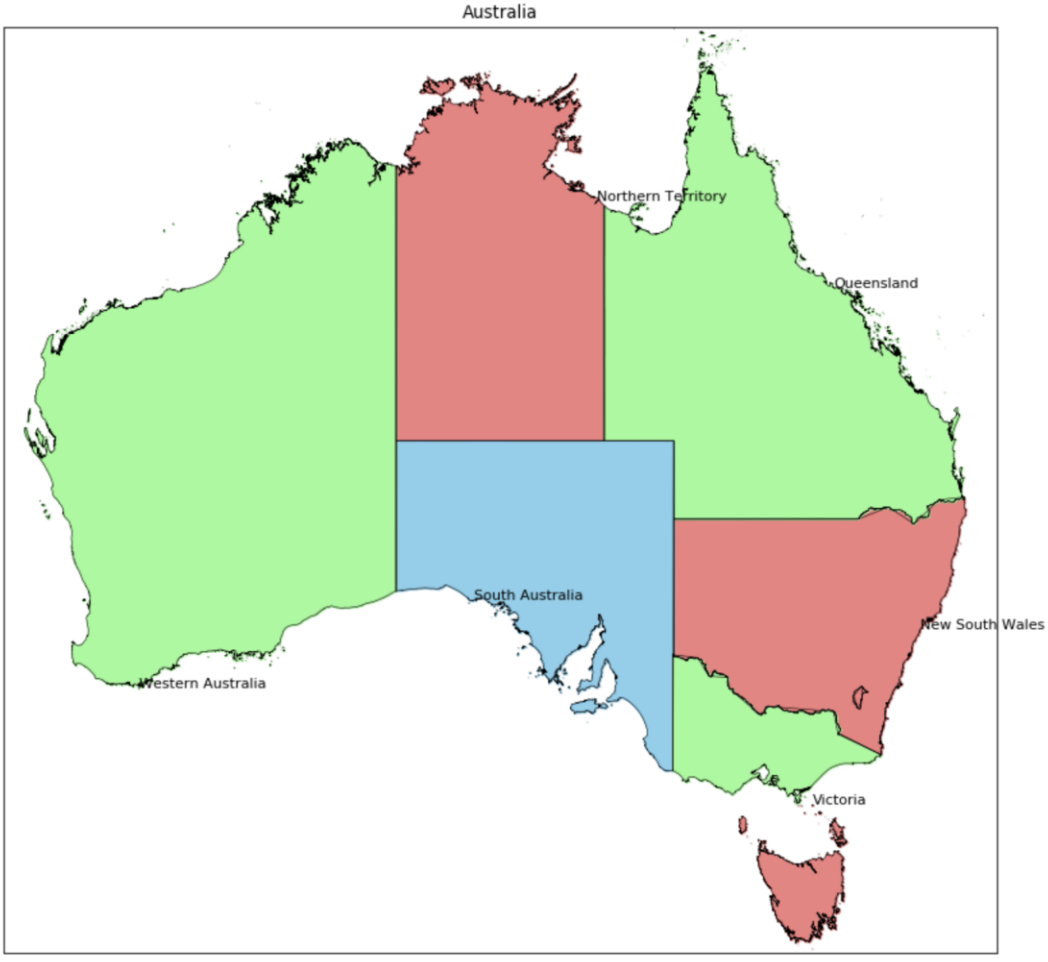
OUTPUT:

For Australia (Non Heuristic):

Dictionary –

For Singleton

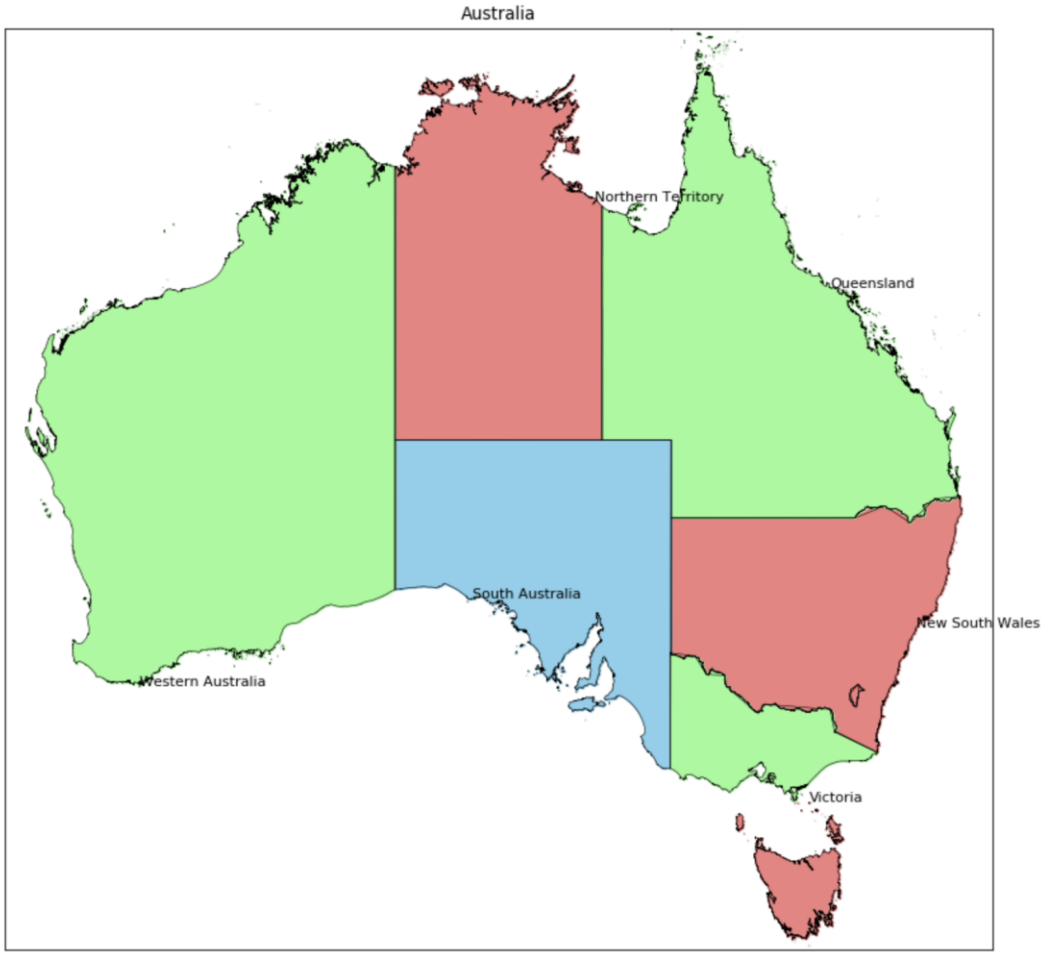
```
Western Australia ==> green
[Souravs-Air:Without_Heuristic souravroychoudhury$ python SingleTon.py
 Solution exist and Following are the assigned colours:
 THE TOTAL TIME TAKEN FOR EXEC ---------------- 0.0001480579376220703
 New South Wales ==> red
 Northern Territory ==> red
 Queensland ==> green
 South Australia ==> blue
 Tasmania ==> red
 Victoria ==> green
 Western Australia ==> green
 Souravs-Air:Without_Heuristic souravroychoudhury$
```



Dictionary –

For Backtracking

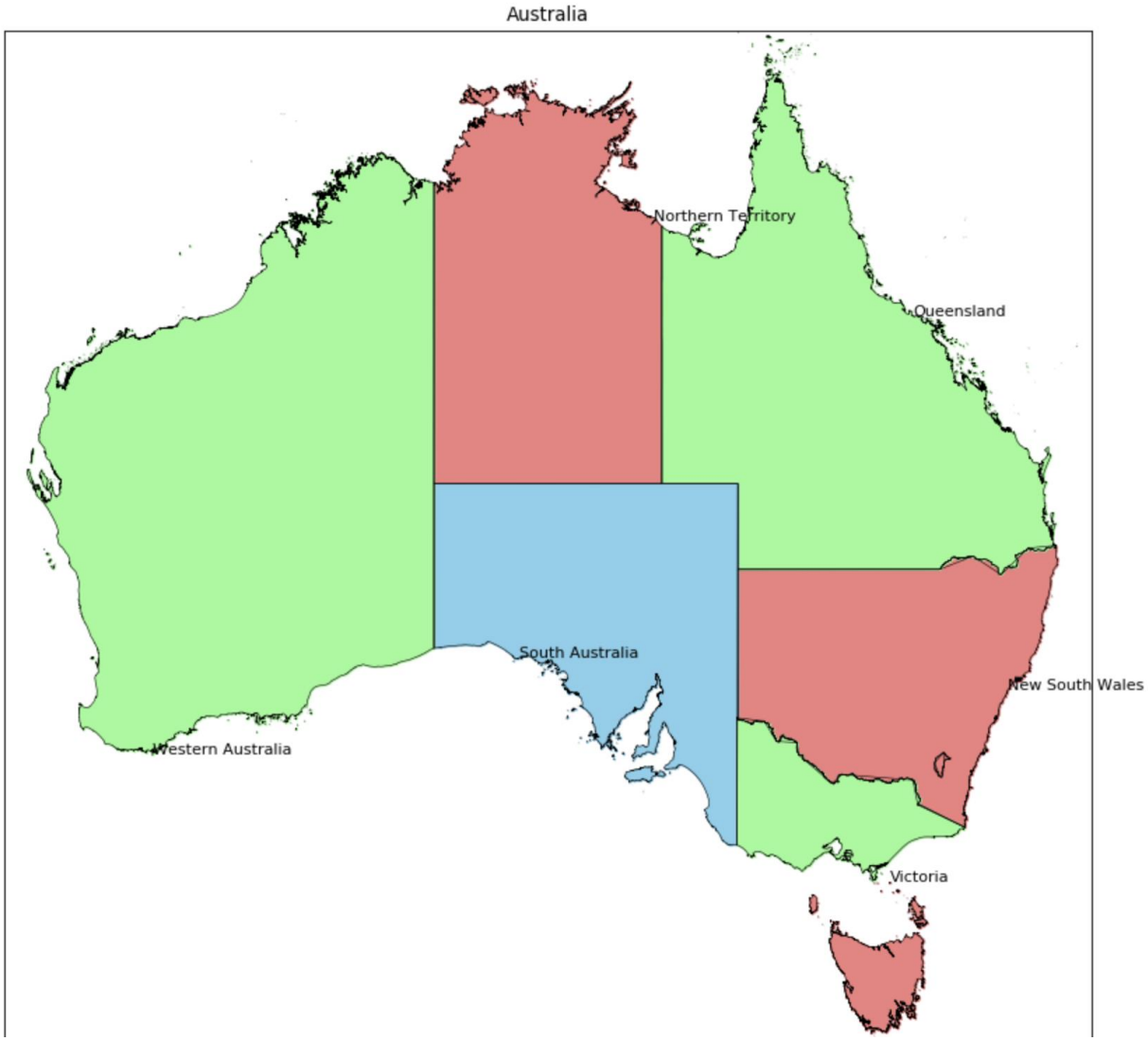
```
[Souravs-Air:Without_Heuristic souravroychoudhury$ python BackTracking.py
Solution exist and Following are the assigned colours:
THE TOTAL TIME TAKEN FOR EXEC ---------------- 0.00012183189392089844
New South Wales ==> red
Northern Territory ==> red
Queensland ==> green
South Australia ==> blue
Tasmania ==> red
Victoria ==> green
Western Australia ==> green
Souravs-Air:Without_Heuristic souravroychoudhury$
```



Dictionary –

For Forwardchecking

```
[Souravs-Air:Without_Heuristic souravroychoudhury$ python ForwardChecking.py
 Solution exist and Following are the assigned colours:
 THE TOTAL TIME TAKEN FOR EXEC ---------------- 0.00013518333435058594
 New South Wales ==> red
 Northern Territory ==> red
 Queensland ==> green
 South Australia ==> blue
 Tasmania ==> red
 Victoria ==> green
 Western Australia ==> green
 Souravs-Air:Without_Heuristic souravroychoudhury$
```

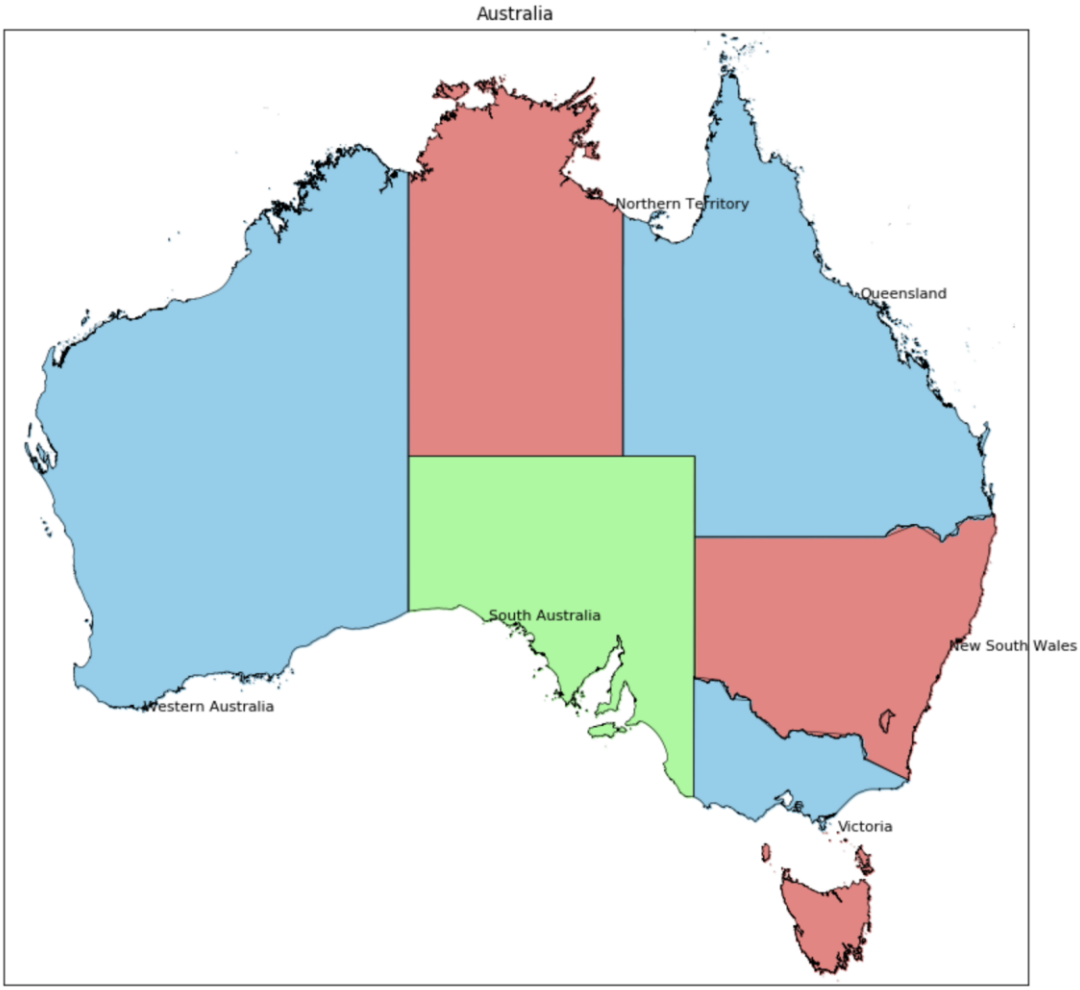


For Australia (Heuristic):

Dictionary –

For Singleton

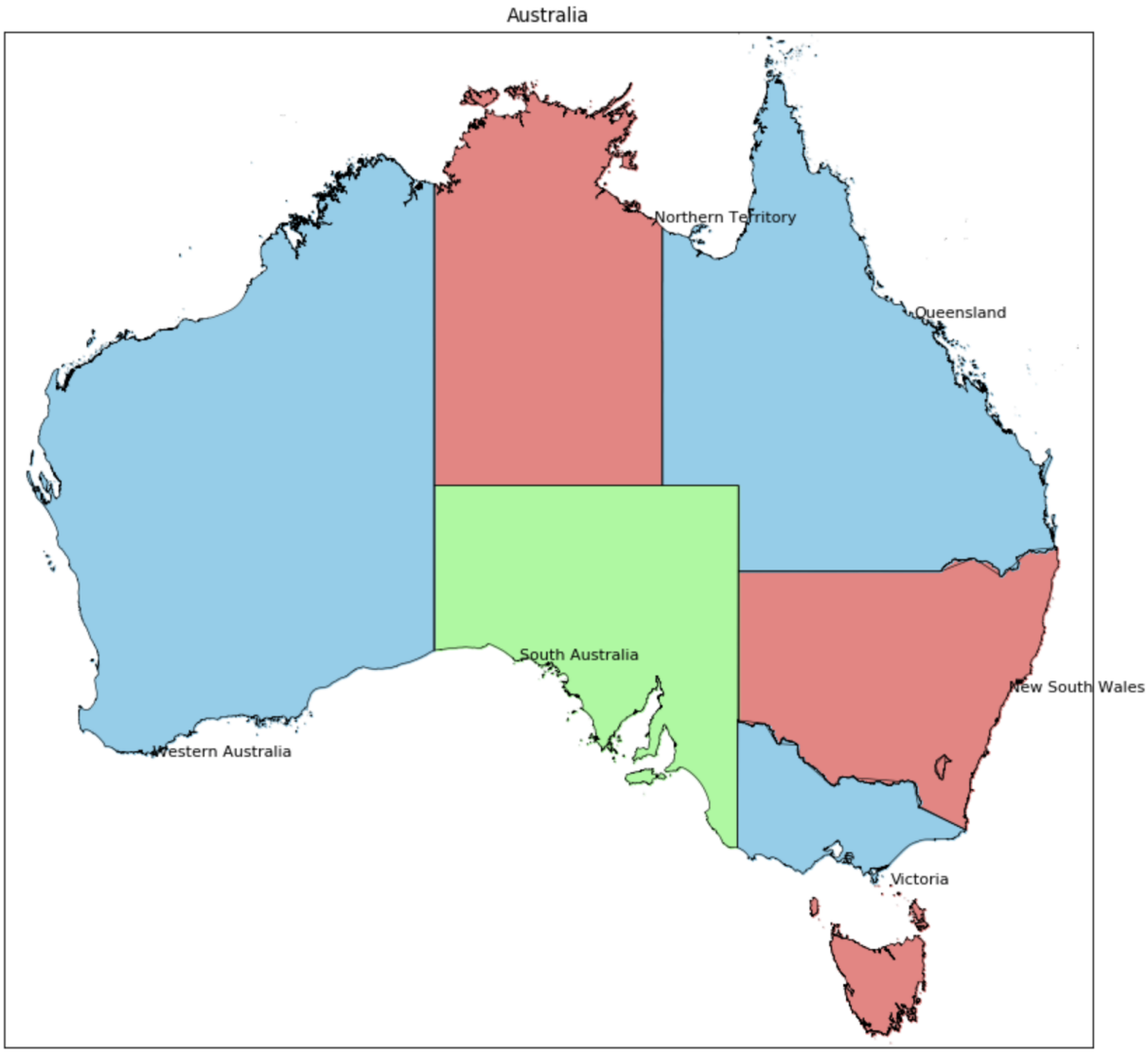
```
[Souravs-Air:heuristic souravroychoudhury$ python Singleton_Heuristic.py
 Solution exist and Following are the assigned colours:
 THE TOTAL TIME TAKEN FOR EXEC ---------------- 0.0004990100860595703
 New South Wales ==> red
 Northern Territory ==> red
 Queensland ==> blue
 South Australia ==> green
 Tasmania ==> red
 Victoria ==> blue
 Western Australia ==> blue
 Souravs-Air:heuristic souravroychoudhury$
```



Dictionary –

For Backtracking

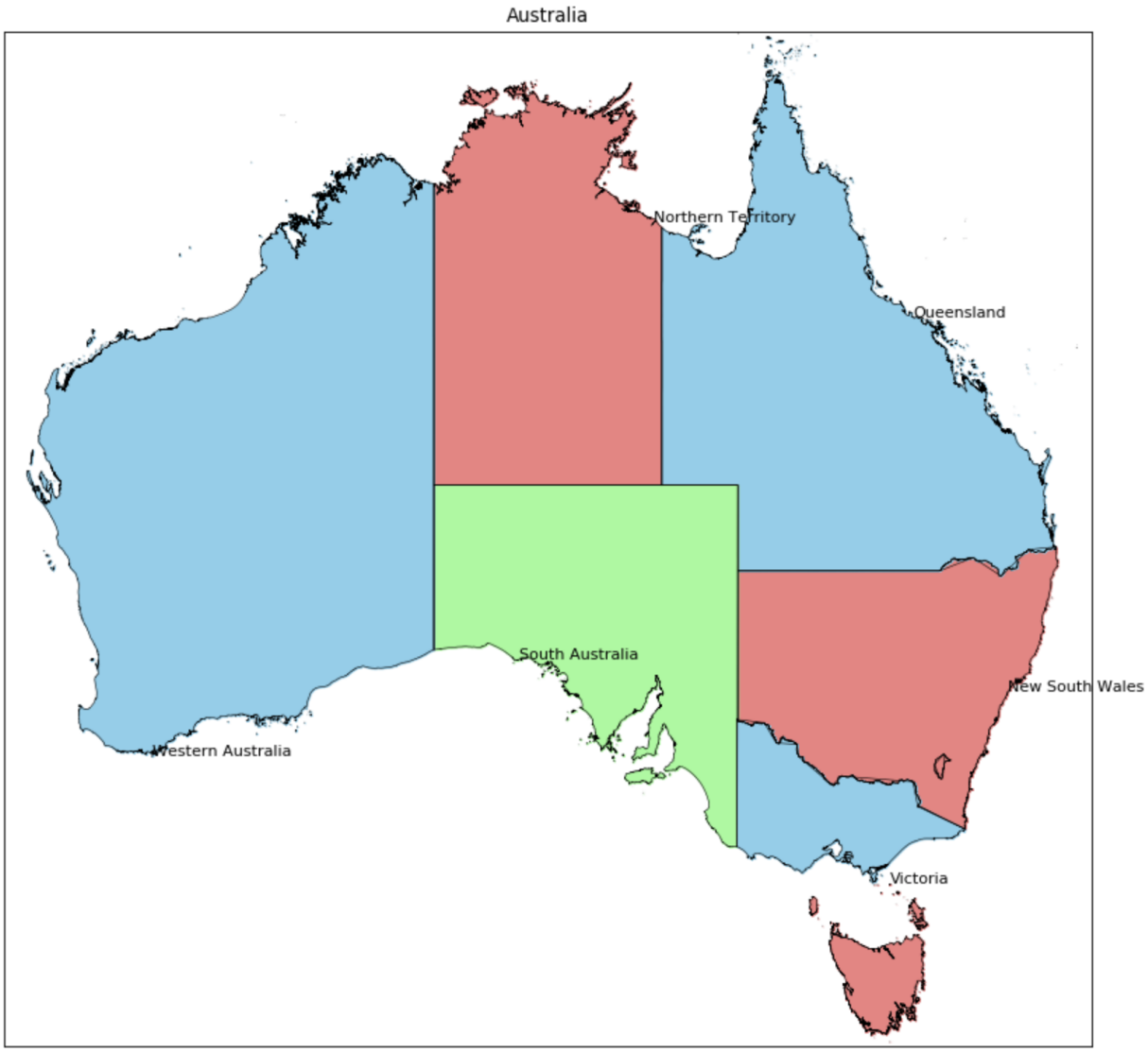
```
Western Australia ==> blue
[Souravs-Air:heuristic souravroychoudhury$ python BackTracking_Heuristic.py
Solution exist and Following are the assigned colours:
THE TOTAL TIME TAKEN FOR EXEC ---------------- 0.00031304359436035156
New South Wales ==> red
Northern Territory ==> red
Queensland ==> blue
South Australia ==> green
Tasmania ==> red
Victoria ==> blue
Western Australia ==> blue
Souravs-Air:heuristic souravroychoudhury$
```



Dictionary –

For Forward Checking

```
Souravs-Air:heuristic souravroychoudhury$ python ForwardChecking_Heuristic.py
Solution exist and Following are the assigned colours:
THE TOTAL TIME TAKEN FOR EXEC ---------------- 0.00036787986755371094
New South Wales ==> red
Northern Territory ==> red
Queensland ==> blue
South Australia ==> green
Tasmania ==> red
Victoria ==> blue
Western Australia ==> blue
Souravs-Air:heuristic souravroychoudhury$
```



Visualization:

Map Visualization is done using the python Basemap (https://matplotlib.org/basemap/) library which is part of the matoplotlib framework.

The framework uses shapefiles which are basically GIS files i.e. (Geographic information system) files which hold state boundary information for the application to color.

Due to the complexity of the program, the loading time of individual state colors and the screen refreshes to ensure a stable animation, the application will take some time to output the final result. Standard wait time for each approach that perform both the color assignments and the map animation are listed below. (All numbers are in seconds)

America	With Heuristic	Without Heuristic
DFS	43.3965969085693	178.46304988861
DFS + Forward Checking	42.6950631141662	478.620557069778
DFS + FC + Singleton	43.7649929523468	484.730099916458

Australia	With Heuristic	Without Heuristic
DFS	13.7751698493957	13.4441788196563
DFS + Forward Checking	13.5815649032592	13.7467761039733

DFS + FC + Singleton	13.4547619819641	13.3195769786834

Steps to set up basemap to run the application is as follows:

PART 1

NOTE: These steps are when application is run for America

1.Install basemap library (Anaconda installation is needed since conda package manager is used)

https://anaconda.org/anaconda/basemap

2. Download the zip file and extract it in some location. (Size: ~ 900 MB)

https://github.com/matplotlib/basemap/archive/v1.1.0.zip

3. Once installed check if the following code runs successfully in your environment.

python -c "from mpl_toolkits.basemap import Basemap"

4. Then run the Jupyter and open the attached notebook files

5. Search and modify the following line in the files:

map.readshapefile(**/Users/amitshetty/Downloads/**basemap-1.1.0/examples/st99_d00

Note that basemap-1.1.0 is a file that was downloaded in step 2

The bold letters is the path where user would have downloaded the file and extracted it.

6. Check if the path is correct by running the ‘ls’ command on the terminal

PART 2

NOTE: These steps are when application is run for Australia

1. Repeat steps 1-4 from part 1 if not already done.
2. Download the attached Australia_shape_files.zip and extract it.
3. In the Australia python notebook files, modify the following lines **/Users/amitshetty/Downloads**/Australia_States/Ashmore and Cartier Islands_AL4-AL4

   The section in bold is the location where the files have been extracted
   Run a ‘ls’ command from the terminal to check if the file path is valid.
4. The python notebook can now be executed. It. Might take some time as shown in the table above.