CHARLOTTE

ITCS 6150 - Intelligent Systems

Project - 3

Constraint satisfaction problems (CSP) - Map Coloring

SUBMITTED TO

Dewan T. Ahmed, Ph.D.

SUBMITTED BY:

Anoosh Guddehithlu Prathap Kumar [801200789]

Sharat Sindoor [801203539]

Constraint satisfaction problems

A k-coloring of a map is an assignment of k colors, one to each country, in such a way that no two countries sharing a border have the same color. This problem can be translated into a constraint graph. A coloring of a graph G assigns a color to each vertex of G, with the restriction that two adjacent vertices never have the same color. The chromatic number of G, written $\chi(G)$, is the smallest number of colors needed to color G.

Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used techniques are variants of backtracking, constraint propagation, and local search.

There are 3 approaches we have used in this project Depth First Search, DFS with forward checking, DFS with forward checking and propagation through singleton.

Depth first search(Backtracking):

The concept in backtracking applied to map coloring is that once any variables domain reaches 0 then the algorithm goes back to previous states and see if using other options in the domain would yield a color for the one with the empty set in its domain. Every time the algorithm checks the next state only after reaching there, there are no prechecks done.

DFS With Forward Checking:-

The concept here is exactly same as backtracking only that next check is pre-checked making the algorithm smarter than Just BACKTRACKING. Number of backtracks are significantly reduced.

DFS With Forward Checking and Propagation through Singleton Domain:-

Here the algorithm checks among all possibilities of next states and choses the one with domain value equal to 1 and propagates to the next unassigned variables from the one with domain =1. Number of backtracks are further reduced and the algorithm is relatively faster.

Heuristics Used:-

There may be a number of options or nodes to chose from the next states. Any one of the states may be chosen at random and we could progress the map coloring algorithm. With using heuristics we get an order of choosing the variables depending on some factors. Some of the most commonly used heuristics are as follows below.

1.)**MINIMUM REMAINING VALUES:-**

In this heuristic propagation follows in the order of those nodes with least number of values in it's domain . With respect to map coloring problem If one state has 2 permissible values in its domain and another state has 3 then the state with 2 values would be chosen first , here permissible refers to reducing domain size because of constraints imposed that adjacent states cannot have same color.

2.) **DEGREE HEURISTIC:-**

The idea here is assign a value to the variable that is involved in the largest number of constraints on other unassigned variables. It is often used as a means to reduce the number of same next possibilities ie as a tie breaker with Minimum remaining values heuristic to chose the best next when all next nodes have the same number of domain values after a variable assignment is done using Mrv.

3.)**LEAST CONSTRAINING VALUE:-**

Under the "least-constraining-value" heuristic for ordering values, prefer the value that rules out the fewest choices for the neighbouring variables in the constraint graph.

Global Variables used:-
lst_of_clrs:- a list used to store list of colors
Starting_time and ending_time to calculate total execution time
rt: stores color of root node.
My_State_dict:-color assigned as program proceeds
Track_backs:-to store the total number of backtracks in the program
n0_of_clrs:-chromatic number
dic_sta_te:- list of states in USA or Australia depending on the user choice

List of functions used:

building_clr_grphs - function builds a graph of color

geting_clrs - function for color retrieving

clr_generate - function to generate colors

Forw_chq - Function to apply Forward tracking method

set_neighbour, get_neighbour, set_colors, set_parents, put_domain - Used to initialize a state

ava_clrs - checks for available colors

chng_dmn - used to change domains

single_dmn_states - used to check which states need propagation

prop_single_ton - driver code of the singleton checking

update_clr - used to update colors

choose_large_st - this function is used to choose the state with largest number of constraints

Results:

USA:

1. DFS without heuristics:

Number of executions	Number of backtracks	Time taken
1	1012451	23.047109603881836seconds
2	1012451	14.418904542922974seconds
3	1012451	17.300964369135966seconds
4	1012451	17.913596630096436seconds
5	1012451	20.690044164657593seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

1

(1, {'Maine': 'red', 'Minnesota': 'red', 'South Dakota': 'blue', 'Illinois': 'red', 'Utah': 'red', 'Wyoming': 'green', 'Texas': 'blue', 'Idaho': 'blue', 'Wisconsin': 'blue', 'Connecticut': 'red', 'Pennsylvania': 'red', 'Kansas': 'green', 'West Virginia': 'blue', 'North Carolina': 'green', 'Colorado': 'blue', 'California': 'red', 'Florida': 'red', 'Vermont': 'red', 'Virginia': 'red', 'North Dakota': 'green', 'Michigan': 'green', 'New Jersey': 'blue', 'Nevada': 'green', 'Arkansas': 'green', 'Mississippi': 'red', 'Iowa': 'green', 'Kentucky': 'green', 'Maryland': 'green', 'Louisiana': 'black', 'Alabama': 'green', 'Oklahoma': 'red', 'New Mexico': 'green', 'Rhode Island': 'blue', 'Massachusetts': 'green', 'South Carolina': 'red', 'Indiana': 'blue', 'Delaware': 'black', 'Tennessee': 'blue', 'Georgia': 'black', 'Arizona': 'black', 'Nebraska': 'red', 'Missouri': 'black', 'New Hampshire': 'blue', 'Ohio': 'black', 'Oregon': 'black', 'Washington': 'red', 'Montana': 'red', 'New York': 'black'})

NUMBER OF BACKTRACKS: 1012451

TIME TAKEN FOR EXECUTION: 14.418904542922974seconds

2. DFS [With heuristics] :

Number of executions	Number of backtracks	**Time taken**
1	701287	23.99692907333323seconds
2	701287	25.51555614474487seconds
3	701287	21.274975299835205seconds
4	701287	24.516575299835205seconds
5	701287	20.44617414474487seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

1

(1, {'Maine': 'red', 'Minnesota': 'red', 'South Dakota': 'blue', 'Illinois': 'red', 'Utah': 'red', 'Wyoming': 'green', 'Texas': 'blue', 'Idaho': 'blue', 'Wisconsin': 'blue', 'Connecticut': 'red', 'Pennsylvania': 'red', 'Kansas': 'green', 'West Virginia': 'blue', 'North Carolina': 'green', 'Colorado': 'blue', 'California': 'red', 'Florida': 'red', 'Vermont': 'red', 'Virginia': 'red', 'North Dakota': 'green', 'Michigan': 'green', 'New Jersey': 'blue', 'Nevada': 'green', 'Arkansas': 'green', 'Mississippi': 'red', 'Iowa': 'green', 'Kentucky': 'green', 'Maryland': 'green', 'Louisiana': 'black', 'Alabama': 'green', 'Oklahoma': 'red', 'New Mexico': 'green', 'Rhode Island': 'blue', 'Massachusetts': 'green', 'South Carolina': 'red', 'Indiana': 'blue', 'Delaware': 'black', 'Tennessee': 'blue', 'Georgia': 'black', 'Arizona': 'black', 'Nebraska': 'red', 'Missouri': 'black', 'New Hampshire': 'blue', 'Ohio': 'black', 'Oregon': 'black', 'Washington': 'red', 'Montana': 'red', 'New York': 'black'})

Number Of Backtracks: 701287

TIME TAKEN FOR EXECUTION: 20.997653245925903seconds

3. DFS with forward checking [without heuristic]

Number of run	Number of backtrack	Time taken
1	10231	70.232476857348957 seconds
2	10231	71.977283452746982seconds
3	10231	70.593252454327954seconds
4	10231	72.086287346735237seconds
5	10231	71.112123245546576seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

1

(1, {'New Hampshire': 'red', 'Oklahoma': 'red', 'Tennessee': 'red', 'Illinois': 'red', 'New Mexico': 'blue', 'Kentucky': 'blue', 'West Virginia': 'green', 'Maryland': 'red', 'Maine': 'blue', 'Wisconsin': 'blue', 'Missouri': 'green', 'Minnesota': 'red', 'Montana': 'red', 'Massachusetts': 'blue', 'South Carolina': 'red', 'North Dakota': 'blue', 'Pennsylvania': 'blue', 'Arizona': 'green', 'South Dakota': 'green', 'Ohio': 'red', 'Oregon': 'red', 'Alabama': 'blue', 'Indiana': 'green', 'Rhode Island': 'red', 'Virginia': 'black', 'Idaho': 'green', 'Nevada': 'blue', 'Nebraska': 'red', 'New York': 'green', 'Utah': 'red', 'Michigan': 'black', 'Kansas': 'blue', 'Florida': 'red', 'Connecticut': 'black', 'Iowa': 'black', 'Wyoming': 'blue', 'Louisiana': 'red', 'California': 'black', 'Vermont': 'black', 'Texas': 'green', 'Georgia': 'green', 'New Jersey': 'red', 'North Carolina': 'blue', 'Washington': 'blue', 'Delaware': 'green', 'Colorado': 'black', 'Mississippi': 'green', 'Arkansas': 'blue'})

Number Of Backtracks: 10231

TIME TAKEN FOR EXECUTION: 70.232476857348957 seconds

4. DFS with forward checking [with heuristic]

Number of run	Number of backtrack	Time taken
1	1713	0.12491655349731445seconds
2	1713	0.07805252075195312seconds
3	1713	0.07978534698486328seconds
4	1713	0.07878468768327873seconds
5	1713	0.11237462387288990seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

1

(1, {'Illinois': 'red', 'Oklahoma': 'red', 'California': 'red', 'Utah': 'red', 'Wyoming': 'blue', 'Missouri': 'blue', 'Michigan': 'green', 'Texas': 'blue', 'Iowa': 'green', 'Delaware': 'red', 'Tennessee': 'red', 'Maryland': 'blue', 'Kentucky': 'green', 'Montana': 'red', 'Minnesota': 'red', 'Connecticut': 'red', 'Louisiana': 'red', 'West Virginia': 'red', 'Pennsylvania': 'green', 'Nebraska': 'red', 'Kansas': 'green', 'Indiana': 'blue', 'Rhode Island': 'blue', 'Arizona': 'blue', 'Florida': 'red', 'Massachusetts': 'green', 'South Dakota': 'black', 'Nevada': 'green', 'South Carolina': 'red', 'Ohio': 'black', 'New Hampshire': 'red', 'Idaho': 'black', 'Washington': 'red', 'Colorado': 'black', 'Oregon': 'blue', 'New Jersey': 'blue', 'Mississippi': 'blue', 'Arkansas': 'green', 'Vermont': 'blue', 'Wisconsin': 'blue', 'Alabama': 'green', 'Georgia': 'blue', 'Maine': 'blue', 'New Mexico': 'green', 'North Carolina': 'green', 'New York': 'black', 'Virginia': 'black', 'North Dakota': 'blue'})

Number Of Backtracks: 1713

TIME TAKEN FOR EXECUTION: 0.12491655349731445seconds

5. DFS with forward checking and propagation through singleton domains [With heuristic]:

Number of run	Number of backtrack	Time taken
1	24371	0.4998853206634521seconds
2	24371	0.4998810291290283seconds
3	24371	0.5623590946197512seconds
4	24371	0.5357866878278409seconds
5	24371	0.4928348286472676seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

1

(1, {'Kansas': 'green', 'New Hampshire': 'green', 'Idaho': 'green', 'Louisiana': 'brown', 'New Jersey': 'green', 'Arkansas': 'green', 'Kentucky': 'green', 'Maine': 'brown', 'Minnesota': 'brown', 'Missouri': 'yellow', 'West Virginia': 'yellow', 'North Carolina': 'green', 'Massachusetts': 'brown', 'Michigan': 'green', 'Indiana': 'yellow', 'Illinois': 'brown', 'Virginia': 'brown', 'Oklahoma': 'brown', 'Montana': 'brown', 'North Dakota': 'green', 'Texas': 'yellow', 'Colorado': 'yellow', 'South Carolina': 'brown', 'Maryland': 'green', 'California': 'green', 'New York': 'yellow', 'Florida': 'green', 'Vermont': 'blue', 'Utah': 'brown', 'Georgia': 'yellow', 'Oregon': 'brown', 'Wisconsin': 'yellow', 'Rhode Island': 'green', 'Nebraska': 'brown', 'New Mexico': 'green', 'Mississippi': 'green', 'Alabama': 'brown', 'Nevada': 'yellow', 'Tennessee': 'blue', 'Iowa': 'green', 'South Dakota': 'yellow', 'Ohio': 'brown', 'Pennsylvania': 'blue', 'Washington': 'yellow', 'Wyoming': 'blue', 'Arizona': 'blue', 'Delaware': 'brown', 'Connecticut': 'blue'})

Number Of Backtracks: 24371

TIME TAKEN FOR EXECUTION: 0.4998853206634521seconds

6. DFS with forward checking and propagation through singleton domains [Without heuristic]:

Number of run	Number of backtrack	Time taken
1	9691	2.0298960208892822 seconds
2	9691	0.16092705726623535seconds
3	9691	0.15959963909444808seconds
4	9691	1.5394909594480896seconds
5	9691	1.348089599094496seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

1

Time taken for execution = 2.0298960208892822 seconds

Number of Backtracks = 9691

{'Ohio': 'blue', 'Hawaii': 'blue', 'Vermont': 'blue', 'Maine': 'blue', 'Tennessee': 'blue', 'Oklahoma': 'blue', 'Colorado': 'green', 'Alabama': 'green', 'Oregon': 'blue', 'Minnesota': 'blue', 'New Mexico': 'red', 'Mississippi': 'red', 'Kansas': 'red', 'New Hampshire': 'green', 'Louisiana': 'blue', 'Rhode Island': 'blue', 'Montana': 'blue', 'Wisconsin': 'green', 'Michigan': 'red', 'Arkansas': 'green', 'Maryland': 'blue', 'Missouri': 'orange', 'Massachusetts': 'red', 'North Dakota': 'green', 'Nevada': 'green', 'South Dakota': 'orange', 'Illinois': 'blue', 'Washington': 'green', 'Virginia': 'green', 'Indiana':

'green', 'Alaska': 'blue', 'Connecticut': 'green', 'North Carolina': 'red', 'New York': 'orange', 'New Jersey': 'blue', 'Iowa': 'red', 'Kentucky': 'red', 'South Carolina': 'blue', 'West Virginia': 'orange', 'Idaho': 'orange', 'Florida': 'blue', 'Delaware': 'green', 'Nebraska': 'blue', 'Arizona': 'orange', 'Wyoming': 'red', 'California': 'red', 'Utah': 'blue', 'Texas': 'orange', 'Pennsylvania': 'red', 'Georgia': 'orange'}

Australia:

1. DFS without heuristics:

Number of executions	Number of backtracks	Time taken
1	0	2.759106515555e-05seconds
2	0	2.799106443125e-05seconds
3	0	2.239104453125e-05seconds
4	0	2.611556453545e-05seconds
5	0	2.551064156514e-05seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

2

(1, {'wa': 'red', 'nt': 'blue', 'q': 'red', 'nsw': 'blue', 'v': 'red', 'sa': 'green'})

NUMBER OF BACKTRACKS: 0

TIME TAKEN FOR EXECUTION: 2.239104453125e-05seconds

2. DFS [With heuristics] :

Number of executions	Number of backtracks	Time taken
1	0	1.259106515555e-05seconds
2	0	2.399106443125e-05seconds

3	0	1.939104453125e-05seconds
4	0	2.011556453545e-05seconds
5	0	2.751064156514e-05seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

2

(1, {'wa': 'red', 'nt': 'blue', 'q': 'red', 'nsw': 'blue', 'v': 'red', 'sa': 'green'})

Number Of Backtracks: 0

TIME TAKEN FOR EXECUTION: 2.751064156514e-05seconds

3. DFS with forward checking [without heuristic]

Number of run	Number of backtrack	Time taken
1	0	0.156218290328975seconds
2	0	0.142836816973192seconds
3	0	0.144676872613895seconds
4	0	0.156090993276456seconds
5	0	0.151213489876993seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

2

['wa', 'nt', 'q', 'nsw', 'v', 'sa']

(1, {'wa': 'blue', 'nt': 'yellow', 'q': 'blue', 'nsw': 'yellow', 'v': 'blue', 'sa': 'brown'})

Number Of Backtracks: 0

TIME TAKEN FOR EXECUTION: 0.1562182903289795seconds

4. DFS with forward checking [with heuristic]

Number of run	Number of backtrack	Time taken
1	0	0.000997304916388seconds
2	0	0.000912975424654seconds
3	0	0.000187235452656seconds
4	0	0.000971893277347seconds
5	0	0.000196380875545seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

2

(1, {'wa': 'red', 'nt': 'blue', 'q': 'red', 'nsw': 'blue', 'v': 'red', 'sa': 'green'})

Number Of Backtracks: 0

TIME TAKEN FOR EXECUTION: 0.000997304916388seconds

5. DFS with forward checking and propagation through singleton domains [With heuristic]:

Number of run	Number of backtrack	Time taken
1	0	0.000935676746352seconds
2	0	0.000965356879976seconds
3	0	0.000193427877659seconds
4	0	0.000957886643458seconds
5	0	0.000186454690854seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

2

(1, {'wa': 'green', 'nt': 'brown', 'q': 'green', 'nsw': 'brown', 'v': 'green', 'sa': 'yellow'})Number Of Backtracks: 0

TIME TAKEN FOR EXECUTION: 0.000935676746352seconds

6. DFS with forward checking and propagation through singleton domains [Without heuristic]:

Number of run	Number of backtrack	Time taken
1	0	2.0298960208892822 seconds
2	0	0.16092705726623535seconds
3	0	0.15959963909444808seconds
4	0	1.53949095944808896seconds
5	0	1.3480895990944496seconds

Sample Output:

Choose The Map : 1. USA 2. Australia

2

Time taken for execution = 2.11024808883667 seconds

Number of Backtracks = 0

{'wa': 'blue', 'nt': 'green', 'q': 'blue', 'nsw': 'green', 'v': 'blue', 'sa': 'red'}

Source Code [Includes both USA and Aus]:

Without Heuristics :

DFS

```
# This file runs the dfs algorithm on both aus and USA without any heuristics
#importing all relevevant files
import time
import random
from Node import Node

# variables that are global
lst_of_clrs = []
sta_tes_all = []
track_backs = 0



#this function sets the random initial colors for the states
def setting_clrs(wn):
    for qq in range(wn):
        lst_of_clrs.append(random.randint(0,255))



#this function just returns the random config of colors
def rdm_clrs(wn):
    return tuple(lst_of_clrs)

#this func builds a graph of colors
def building_clr_grphs(n0,ste_n0,sta_tess):
    clrs = rdm_clrs(3)
    x = Node(clrs[0],sta_tess[0])
    root_r = x
    my_state_s = {}
```

```
    for f in range(1,ste_n0,1):
        w = Node(clrs[0],sta_tess[f])
        x.put_child(w)
        x.next_nde = w
        my_state_s[sta_tess[f]] = x
        for y in range(1,n0,1):
            z = Node(clrs[y])
            x.put_child(z)
        x = w
    print(root_r)
    print(root_r.next)
    print(root_r.next[1].next)

#function for color retrieving
def geting_clrs(sta_tess,my_ste_dic):
    clr_lst = []
    for qq in sta_tess:
        clr_lst.append(my_ste_dic.get(qq,""))
    return clr_lst



#funct for color generation
def clr_generate(sta_tess,qq,n0_of_clrs,clrs):
    lst_t = []
    for rr in range(n0_of_clrs):
        lst_t.append(Node(clrs[rr],sta_tess[qq]))
    return lst_t



#main dfs part of the program
def d_f_s(my_ste_dic,dic_sta_te,n0_of_clrs,st_ate_cur,sta_tess,n0):
    global track_backs
```

```
    for qq in range(len(st_ate_cur.next)):
        my_ste_dic[st_ate_cur.next[0].myname] = st_ate_cur.next[qq].mycolor
        if my_ste_dic.get(st_ate_cur.next[0].myname) in
geting_clrs(dic_sta_te[st_ate_cur.next[0].myname],my_ste_dic):
            continue
        sta_tes_all.append(my_ste_dic.copy())
        if n0 == len(sta_tess) - 1:
            return 1,my_ste_dic
        temp_colorlist = lst_of_clrs.copy()
        st_ate_cur.next[qq].next = clr_generate(sta_tess,n0+1,n0_of_clrs,temp_colorlist)
        sol1 = d_f_s(my_ste_dic,dic_sta_te,n0_of_clrs,st_ate_cur.next[qq],sta_tess,n0+1)
        if sol1[0] == 1:
            return 1,my_ste_dic
        continue
    track_backs+=1
    return 0,my_ste_dic



#initializing function
def init(sta_tess,dic_sta_te,n0_of_clrs):
    clrs = lst_of_clrs
    root_r = Node(clrs[0],sta_tess[0])
    for rr in range(n0_of_clrs):
        root_r.put_child(Node(clrs[rr],sta_tess[0]))
    return root_r



#main funtion
if __name__ == "__main__":
    m=int(input("Choose The Map : 1. USA  2. Australia \n"))
    n0_of_clrs = 4
    setting_clrs(n0_of_clrs)
```

```
    lst_of_clrs = ["red","blue","green","black"]
    #different states are assigned based on users choice
    if m==1:
        dic_sta_te = {
        'Alabama':['Florida', 'Georgia', 'Mississippi', 'Tennessee'],
        'Arizona':['California', 'Colorado', 'Nevada', 'New Mexico', 'Utah'],
        'Arkansas' :['Louisiana', 'Mississippi', 'Missouri', 'Oklahoma', 'Tennessee', 'Texas'],
        'California':['Arizona', 'Nevada', 'Oregon'],
        'Colorado':['Arizona', 'Kansas', 'Nebraska', 'New Mexico', 'Oklahoma', 'Utah', 'Wyoming'],
        'Connecticut':['Massachusetts', 'New York', 'Rhode Island'],
        'Delaware':['Maryland', 'New Jersey', 'Pennsylvania'],
        'Florida':['Alabama', 'Georgia'],
        'Georgia':['Alabama', 'Florida', 'North Carolina', 'South Carolina', 'Tennessee'],
        'Idaho':['Montana', 'Nevada', 'Oregon', 'Utah', 'Washington', 'Wyoming'],
        'Illinois':['Indiana','Iowa', 'Michigan', 'Kentucky', 'Missouri', 'Wisconsin'],
        'Indiana':['Illinois', 'Kentucky', 'Michigan', 'Ohio'],
        'Iowa': ['Illinois', 'Minnesota', 'Missouri', 'Nebraska', 'South Dakota', 'Wisconsin'],
        'Kansas' :['Colorado', 'Missouri', 'Nebraska', 'Oklahoma'],
        'Kentucky':['Illinois', 'Indiana', 'Missouri', 'Ohio', 'Tennessee', 'Virginia', 'West Virginia'],
        'Louisiana':['Arkansas', 'Mississippi', 'Texas'],
        'Maine':["New Hampshire"],
        "Maryland":['Delaware','Pennsylvania','Virginia', 'West Virginia'],
        'Massachusetts':['Connecticut', 'New Hampshire', 'New York', 'Rhode Island', 'Vermont'],
        'Michigan':['Illinois', 'Indiana', 'Minnesota', 'Ohio', 'Wisconsin'],
        'Minnesota':['Iowa', 'Michigan', 'North Dakota', 'South Dakota', 'Wisconsin'],
        'Mississippi':['Alabama', 'Arkanssas', 'Louisiana', 'Tennessee'],
        'Missouri':['Arkansas', 'Illinois', 'Iowa', 'Kansas', 'Kentucky', 'Nebraska', 'Oklahoma',
'Tennessee'],
        'Montana':['Idaho', 'North Dakota', 'South Dakota', 'Wyoming'],
        'Nebraska' :['Colorado', 'Iowa', 'Kansas', 'Missouri', 'South Dakota', 'Wyoming'],
        'Nevada':['Arizona', 'California', 'Idaho', 'Oregon', 'Utah'],
```

```
'New Hampshire': ['Maine', 'Massachusetts', 'Vermont'],

'New Jersey':["Delaware", "New York", "Pennsylvania"],

'New Mexico':['Arizona', 'Colorado', 'Oklahoma', 'Texas', 'Utah'],

'New York':['Connecticut', 'Massachusetts', 'New Jersey', 'Pennsylvania', 'Rhode Island', 'Vermont'],

'North Carolina':['Georgia', 'South Carolina', 'Tennessee', 'Virginia'],

'North Dakota':['Minnesota', 'Montana', 'South Dakota'],

'Ohio':['Indiana', 'Kentucky', 'Michigan', 'Pennsylvania', 'West Virginia'],

'Oklahoma' :['Arkansas', 'Colorado', 'Kansas', 'Missouri', 'New Mexico', 'Texas'],

'Oregon':["California", 'Idaho', 'Nevada', "Washington"],

'Pennsylvania':['Delaware', 'Maryland', 'New Jersey', 'New York', 'Ohio', 'West Virginia'],

'Rhode Island':['Connecticut', 'Massachusetts', 'New York'],

'South Carolina':['Georgia', 'North Carolina'],

'South Dakota':['Iowa', 'Minnesota', 'Montana', 'Nebraska', 'North Dakota', 'Wyoming'],

'Tennessee':['Alabama', 'Arkansas', 'Georgia', 'Kentucky', 'Mississippi', 'Missouri', 'North Carolina', 'Virginia'],

'Texas':['Arkansas', 'Louisiana', 'New Mexico', 'Oklahoma'],

'Utah':['Arizona', 'Colorado', 'Idaho', 'Nevada', 'New Mexico', 'Wyoming'],

'Vermont':['Massachusetts', 'New Hampshire', 'New York'],

'Virginia':['Kentucky', 'Maryland', 'North Carolina', 'Tennessee', 'West Virginia'],

'Washington':['Idaho', 'Oregon'],

'West Virginia':['Kentucky', 'Maryland', 'Ohio', 'Pennsylvania', 'Virginia'],

'Wisconsin':['Illinois', 'Iowa', 'Michigan', 'Minnesota'],

'Wyoming':['Colorado', 'Idaho', 'Montana', 'Nebraska', 'South Dakota', 'Utah']

}


sta_tess = ['Maine', 'Minnesota', 'South Dakota', 'Illinois', 'Utah', 'Wyoming', 'Texas', 'Idaho', 'Wisconsin', 'Connecticut',

        'Pennsylvania', 'Kansas', 'West Virginia', 'North Carolina', 'Colorado', 'California', 'Florida', 'Vermont', 'Virginia',

        'North Dakota', 'Michigan', 'New Jersey', 'Nevada', 'Arkansas', 'Mississippi', 'Iowa', 'Kentucky', 'Maryland', 'Louisiana',

        'Alabama', 'Oklahoma', 'New Mexico', 'Rhode Island', 'Massachusetts', 'South Carolina', 'Indiana', 'Delaware', 'Tennessee',
```

'Georgia', 'Arizona', 'Nebraska', 'Missouri', 'New Hampshire', 'Ohio', 'Oregon', 'Washington', 'Montana', 'New York']

```
if m==2:
    sta_tess=['wa','nt','q','nsw','v','sa']

    dic_sta_te  ={
        'wa':['nt','sa'],
        'nt':['wa','q','sa'],
        'sa':['wa','q','nsw','nt','v'],
        'q':['nt','sa','nsw'],
        'nsw':['q','v','sa'],
        'v':['sa','nsw']}

my_ste_dic = {}

root_r = init(sta_tess,dic_sta_te,n0_of_clrs)

starting_time = time.time()
solution = d_f_s(my_ste_dic,dic_sta_te,n0_of_clrs,root_r,sta_tess,0)
#displaying the acquired solution
for kk in solution[1]:
    if solution[1][kk] in geting_clrs(dic_sta_te[kk],my_ste_dic):
        print("Problem Occured")
#calculating time
ending_time = time.time()

print(solution)

print("\n\nNUMBER OF BACKTRACKS: "+ str(track_backs))
print("\nTIME TAKEN FOR EXECUTION: " + str(ending_time - starting_time) + "seconds")
```

DFS + forward checking

```
import time
import random
from Node import Node

clr_list = []
No_of_Backtracks = 0
every_st = []

# Function to iniz the color.
def clr_iniz(n):
    for m in range(n):
        clr_list.append(random.randint(0,255))

# Function for Random color generation
def clr_Random(n):
    return tuple(clr_list)

# Function to build the colored graph
def clr_Graph_bld(No,No_st,st):
    clr = clr_Random(3)
    c = Node(clr[0],st[0])
    Root_Node = c # Creating Root node
    st_cur = {}
    for m in range(1,No_st,1):
        z = Node(clr[0],st[m])
        c.put_child(z)
        c.nextnode = z
        st_cur[st[i]] = c

        for n in range(1,No,1):
            d = Node(clr[n])
```

```
            c.put_child(d)
        c = z

    print(Root_Node)
    print(Root_Node.next)
    print(Root_Node.next[1].next)

def clr_Get(st,st_cur_Dict):
    col_lst = []
    for m in st:
        col_lst.append(st_cur_Dict.get(m,""))

    return col_lst

# Function to gen colors
def Col_gen(st,m,Col_No,clr):
    lstT = []
    for n in range(Col_No):
        lstT.append(Node(clr[n],st[m]))

    return lstT

# Function to apply Forward tracking method.
def Forw_chq(st_cur_Dict,st_Dict,Col_No,st_Cur,st,No):
    global No_of_Backtracks
    every_st.append(st_cur_Dict.copy())
    for m in range(len(st_Cur.next)):
        time.sleep(0.000002)
        st_cur_Dict[st_Cur.next[0].myname] = st_Cur.next[m].mycolor
        if st_cur_Dict.get(st_Cur.next[0].myname) in
clr_Get(st_Dict[st_Cur.next[0].myname],st_cur_Dict):
            continue
```

```
        if No == len(st) - 1:
            return 1,st_cur_Dict

        Temporary_CList = clr_list.copy()
        clr_Remove = clr_Get(st[No+1],st_cur_Dict)
        Temporary_CList = [x for x in Temporary_CList if x not in clr_Remove]
        st_Cur.next[m].next = Col_gen(st,No+1,Col_No,Temporary_CList)
        ans = Forw_chq(st_cur_Dict,st_Dict,Col_No,st_Cur.next[m],st,No+1)
        if ans[0] == 1:
            return 1,st_cur_Dict

        continue

    No_of_Backtracks +=1

    return 0,st_cur_Dict


def init(st,st_Dict,Col_No):
    clr = clr_list
    Root_Node = Node(clr[0],st[0])
    for n in range(Col_No):
        Root_Node.put_child(Node(clr[n],st[0]))

    return Root_Node

# Calling main
if __name__ == "__main__":
    i=int(input("Choose The Map : 1. USA  2. Australia \n"))
    Col_No = 4
    clr_iniz(Col_No)
```

```
    clr_list = ["blue","yellow","brown","red"]
    if i==1:
        st_Dict = {
        'Alabama':['Florida', 'Georgia', 'Mississippi', 'Tennessee'],
        'Arizona':['California', 'clrado', 'Nevada', 'New Mexico', 'Utah'],
        'Arkansas' :['Louisiana', 'Mississippi', 'Missouri', 'Oklahoma', 'Tennessee', 'Texas'],
        'California':['Arizona', 'Nevada', 'Oregon'],
        'clrado':['Arizona', 'Kansas', 'Nebraska', 'New Mexico', 'Oklahoma', 'Utah', 'Wyoming'],
        'Connecticut':['Massachusetts', 'New York', 'Rhode Island'],
        'Delaware':['Maryland', 'New Jersey', 'Pennsylvania'],
        'Florida':['Alabama', 'Georgia'],
        'Georgia':['Alabama', 'Florida', 'North Carolina', 'South Carolina', 'Tennessee'],
        'Idaho':['Montana', 'Nevada', 'Oregon', 'Utah', 'Washington', 'Wyoming'],
        'Illinois':['Indiana','Iowa', 'Michigan', 'Kentucky', 'Missouri', 'Wisconsin'],
        'Indiana':['Illinois', 'Kentucky', 'Michigan', 'Ohio'],
        'Iowa': ['Illinois', 'Minnesota', 'Missouri', 'Nebraska', 'South Dakota', 'Wisconsin'],
        'Kansas' :['clrado', 'Missouri', 'Nebraska', 'Oklahoma'],
        'Kentucky':['Illinois', 'Indiana', 'Missouri', 'Ohio', 'Tennessee', 'Virginia', 'West Virginia'],
        'Louisiana':['Arkansas', 'Mississippi', 'Texas'],
        'Maine':["New Hampshire"],
        "Maryland":['Delaware','Pennsylvania','Virginia', 'West Virginia'],
        'Massachusetts':['Connecticut', 'New Hampshire', 'New York', 'Rhode Island', 'Vermont'],
        'Michigan':['Illinois', 'Indiana', 'Minnesota', 'Ohio', 'Wisconsin'],
        'Minnesota':['Iowa', 'Michigan', 'North Dakota', 'South Dakota', 'Wisconsin'],
        'Mississippi':['Alabama', 'Arkanssas', 'Louisiana', 'Tennessee'],
        'Missouri':['Arkansas', 'Illinois', 'Iowa', 'Kansas', 'Kentucky', 'Nebraska', 'Oklahoma',
'Tennessee'],
        'Montana':['Idaho', 'North Dakota', 'South Dakota', 'Wyoming'],
        'Nebraska' :['clrado', 'Iowa', 'Kansas', 'Missouri', 'South Dakota', 'Wyoming'],
        'Nevada':['Arizona', 'California', 'Idaho', 'Oregon', 'Utah'],
        'New Hampshire': ['Maine', 'Massachusetts', 'Vermont'],
```

'New Jersey':["Delaware", "New York", "Pennsylvania"],

'New Mexico':['Arizona', 'clrado', 'Oklahoma', 'Texas', 'Utah'],

'New York':['Connecticut', 'Massachusetts', 'New Jersey', 'Pennsylvania', 'Rhode Island', 'Vermont'],

'North Carolina':['Georgia', 'South Carolina', 'Tennessee', 'Virginia'],

'North Dakota':['Minnesota', 'Montana', 'South Dakota'],

'Ohio':['Indiana', 'Kentucky', 'Michigan', 'Pennsylvania', 'West Virginia'],

'Oklahoma' :['Arkansas', 'clrado', 'Kansas', 'Missouri', 'New Mexico', 'Texas'],

'Oregon':["California", 'Idaho', 'Nevada', "Washington"],

'Pennsylvania':['Delaware', 'Maryland', 'New Jersey', 'New York', 'Ohio', 'West Virginia'],

'Rhode Island':['Connecticut', 'Massachusetts', 'New York'],

'South Carolina':['Georgia', 'North Carolina'],

'South Dakota':['Iowa', 'Minnesota', 'Montana', 'Nebraska', 'North Dakota', 'Wyoming'],

'Tennessee':['Alabama', 'Arkansas', 'Georgia', 'Kentucky', 'Mississippi', 'Missouri', 'North Carolina', 'Virginia'],

'Texas':['Arkansas', 'Louisiana', 'New Mexico', 'Oklahoma'],

'Utah':['Arizona', 'clrado', 'Idaho', 'Nevada', 'New Mexico', 'Wyoming'],

'Vermont':['Massachusetts', 'New Hampshire', 'New York'],

'Virginia':['Kentucky', 'Maryland', 'North Carolina', 'Tennessee', 'West Virginia'],

'Washington':['Idaho', 'Oregon'],

'West Virginia':['Kentucky', 'Maryland', 'Ohio', 'Pennsylvania', 'Virginia'],

'Wisconsin':['Illinois', 'Iowa', 'Michigan', 'Minnesota'],

'Wyoming':['clrado', 'Idaho', 'Montana', 'Nebraska', 'South Dakota', 'Utah'],

"Hawai":[],

"Alaska":[]

}

st = ['New Hampshire', 'Oklahoma', 'Tennessee', 'Illinois', 'New Mexico', 'Kentucky', 'West Virginia', 'Maryland', 'Maine', 'Wisconsin', 'Missouri', 'Minnesota', 'Montana', 'Massachusetts', 'South Carolina', 'North Dakota', 'Pennsylvania', 'Arizona', 'South Dakota', 'Ohio', 'Oregon', 'Alabama', 'Indiana', 'Rhode Island', 'Virginia', 'Idaho', 'Nevada', 'Nebraska', 'New York', 'Utah', 'Michigan', 'Kansas', 'Florida', 'Connecticut', 'Iowa', 'Wyoming', 'Louisiana', 'California', 'Vermont', 'Texas', 'Georgia', 'New Jersey', 'North Carolina', 'Washington', 'Delaware', 'clrado', 'Mississippi', 'Arkansas']

```
        st = ['Kansas', 'New Hampshire', 'Idaho', 'Louisiana', 'New Jersey', 'Arkansas',
'Kentucky', 'Maine', 'Minnesota', 'Missouri',
        'West Virginia', 'North Carolina', 'Massachusetts', 'Michigan', 'Indiana', 'Illinois', 'Virginia',
'Oklahoma', 'Montana',
        'North Dakota', 'Texas', 'clrado', 'South Carolina', 'Maryland', 'California', 'New York',
'Florida', 'Vermont', 'Utah',
        'Georgia', 'Oregon', 'Wisconsin', 'Rhode Island', 'Nebraska', 'New Mexico', 'Mississippi',
'Alabama', 'Nevada', 'Tennessee',
        'Iowa','South Dakota', 'Ohio', 'Pennsylvania', 'Washington', 'Wyoming', 'Arizona',
'Delaware', 'Connecticut']


    else:
        st=['wa','nt','q','nsw','v','sa']
        st_Dict  ={
            'wa':['nt','sa'],
            'nt':['wa','q','sa'],
            'sa':['wa','q','nsw','nt','v'],
            'q':['nt','sa','nsw'],
            'nsw':['q','v','sa'],
            'v':['sa','nsw']}



    st_cur_Dict = {}
    print(st)


    Root_Node = init(st,st_Dict,Col_No)


    strt_Time = time.time()
    Ans = Forw_chq(st_cur_Dict,st_Dict,Col_No,Root_Node,st,0)
    ending_time = time.time()
    count = 0
    for key in Ans[1]:
        count+=1
        if Ans[1][key] in clr_Get(st_Dict[key],st_cur_Dict):
```

```
            print("No Answer")


    print("Answer Verified")
    print(Ans)
    print("No_of_Backtracks: "+ str(No_of_Backtracks))
    print("Total Time taken for Execution: " + str(ending_time - strt_Time) + "seconds")

    print(len(every_st))
    time.sleep(10)
```

DFS + forward checking + Propagation with Singleton domains

```
# This file runs the dfs algorithm on both aus and USA with added singleton propagation

#importing all relevevant files
from copy import deepcopy as dc
import time



# variables that are global
lst_full = []
track_backs = 0



# class definiation and initialization
class State:
    def __init__(self,nme_st,dm_in,st_us="not visited"):
        self.nme_st=nme_st
        self.neigh_brs=None
        self.nme_clr=None
        self.st_us=st_us
```

```
        self.dm_in=dm_in
        self.sin_leton=False
    # internal funtions required to run
    def se_tNeigh(self,neigh_brs):
        self.neigh_brs=neigh_brs
    def get_neigh(self):
        return self.neigh_brs
    def set_clrs(self,nme_clr):
        self.nme_clr=nme_clr
    def set_par_ent(self,par_ent):
        self.par_ent=par_ent
    def put_do_main(self,dm_in):
        self.dm_in=dm_in
    def ret_do_main(self):
        return self.dm_in
    def check_single(self):
        if self.sin_leton:
            return self.sin_leton
        return False


#this function determines which states are still available/not visited
def ava_sta_tes(st_a_te_obj_ects):
    states_not_visted=[]
    for st_a_te in st_a_te_obj_ects:
        if st_a_te.st_us=="not visited":
            states_not_visted.append(st_a_te)
    return states_not_visted

#this function checks which colors are still available  for use

def ava_clrs(st_a_te,dm_in):
```

```
    dmn_avail_able=dm_in.copy()
    for nai_bhor in st_a_te.neigh_brs:
        clr = nai_bhor.nme_clr
        if clr!=None and (clr in dmn_avail_able) :
            dmn_avail_able.remove(clr)
    return dmn_avail_able



#this function is used to change domains
def chng_dmn(st_a_te,clr):
    for nai_bhor in st_a_te.nai_bhor:
        if (nai_bhor.st_us=="not visited") and (clr in nai_bhor.dm_in):
            (nai_bhor.dm_in).remove(clr)

#this function checks which states need propagation
def sngle_dmn_sts(st_a_te_obj_ects):
    k=None
    for st_a_te in st_a_te_obj_ects:
        if st_a_te.sin_leton==False and len(st_a_te.ret_do_main())==1:
            st_a_te.sin_leton=True
            clr = st_a_te.dm_in[0]
            return st_a_te,clr
    return k,k

# main part of the code i.e singleton propagation
def prop_single_ton(st_a_te_obj_ects):
    sngl_ton_sts={}
    while True:
        st_a_te,clr = sngle_dmn_sts(st_a_te_obj_ects)
        if st_a_te==None:
            return sngl_ton_sts,"success"
        else:
```

```
            sngl_ton_sts[st_a_te]=clr
            for nai_bhor in st_a_te.neigh_brs:
                if nai_bhor.st_us=="not visited" and clr in nai_bhor.dm_in:
                    if len(nai_bhor.dm_in)==1:
                        return sngl_ton_sts,"unsucessful"
                    (nai_bhor.dm_in).remove(clr)


#this function sets the random initial colors for the states based on domains

def setting_clrs(dm_in,state_domain):
    dom=dc(dm_in)
    dom2=dc(dom)
    state_dom=dc(state_domain)
    for clr in dom:
        count=0
        for st_a_te_clr in state_dom:
            if clr!=st_a_te_clr:
                count+=1
        if count==len(state_dom):
            dom2.remove(clr)
    return dom2

#this function updates the initial colors for the states based on domains

def upgrade_clrs(st_a_te,clr):
    updated_states=[]
    for nai_bhor in st_a_te.neigh_brs:
        if nai_bhor.st_us=="not visited" and clr in nai_bhor.dm_in:
            (nai_bhor.dm_in).remove(clr)
            updated_states.append(nai_bhor)
    return updated_states
```

```
#this function resets the initial colors for the states based on neighbouring states


def reset(st_a_te,dm_in,clr,updated_states):
    pos = dm_in.index(clr)
    for nai_bhor in st_a_te.neigh_brs:
        if nai_bhor.st_us=="not visited" and (nai_bhor in updated_states):
            (nai_bhor.dm_in).insert(pos,clr)
            dom = setting_clrs(dm_in,nai_bhor.dm_in)
            nai_bhor.dm_in=dc(dom)


#this function resets the singleton propogation status for the states based on neighbouring states


def re_set_single(sngl_ton_sts,dm_in,colour):
    for st_a_te,clr in sngl_ton_sts.items():
        pos = dm_in.index(clr)
        for nai_bhor in st_a_te.neigh_brs:
            if nai_bhor.st_us=="not visited" and (nai_bhor.check_single()):
                (nai_bhor.dm_in).insert(pos,clr)
                dom = setting_clrs(dm_in,nai_bhor.dm_in)
                nai_bhor.dm_in=dom
        st_a_te.sin_leton=False



# This function acts as the anchor that holds the whole program together
def c_s_p(st_a_te_obj_ects,dm_in,states_and_colors):
        global track_backs
        if len(states_and_colors)==len(st_a_te_obj_ects):
            return states_and_colors
        un_assigned_states=ava_sta_tes(st_a_te_obj_ects)
```

```
        st_a_te=un_assigned_states[0]
        dmn_avail_able=dc(st_a_te.dm_in)
        iter1 = 0
        for clr in dmn_avail_able:
            iter1+=1
            st_a_te.st_us="visited"
            st_a_te.nme_clr=clr
            states_and_colors[st_a_te.nme_st]=clr
            updated_states=upgrade_clrs(st_a_te,clr)
            sngl_ton_sts,st_us=prop_single_ton(st_a_te_obj_ects)
            if st_us!="unsucessful":
                rslt=c_s_p(st_a_te_obj_ects,dm_in,states_and_colors)
            else:
                rslt=st_us
            if rslt!="unsucessful":
                return rslt
            del states_and_colors[st_a_te.nme_st]
            st_a_te.nme_clr=None
            st_a_te.st_us="not visited"
        track_backs+=1
        return "unsucessful"
#this func sets the objects according to domains
def set_obj(stat_es,dm_in):
    state_objs = []
    dom=dc(dm_in)
    for st_a_te in stat_es:
        state_obj=State(st_a_te,dom)
        state_objs.append(state_obj)
        dom=dc(dom)
    return state_objs

#main function
```

```
def main():


    global m,k
    m=int(input("Choose The Map : 1. USA  2. Australia \n"))
    #different states are assigned based on users choice


    if m==2:
        stat_es=['wa','nt','q','nsw','v','sa']
        graph_rest_riction={
            'wa':['nt','sa'],
            'nt':['wa','q','sa'],
            'sa':['wa','q','nsw','nt','v'],
            'q':['nt','sa','nsw'],
            'nsw':['q','v','sa'],
            'v':['sa','nsw']
            }
    if m==1:
        stat_es= ['New Hampshire', 'Oklahoma', 'Tennessee', 'Illinois', 'New Mexico', 'Kentucky',
'West Virginia', 'Maryland',
                  'Maine', 'Wisconsin', 'Missouri', 'Minnesota', 'Montana', 'Massachusetts', 'South
Carolina', 'North Dakota',
                  'Pennsylvania', 'Arizona', 'South Dakota', 'Ohio', 'Oregon', 'Alabama', 'Indiana', 'Rhode
Island', 'Virginia',
                  'Idaho', 'Nevada', 'Nebraska', 'New York', 'Utah', 'Michigan', 'Kansas', 'Florida',
'Connecticut', 'Iowa',
                  'Wyoming', 'Louisiana', 'California', 'Vermont', 'Texas', 'Georgia', 'New Jersey', 'North
Carolina', 'Washington',
                  'Delaware', 'Colorado', 'Mississippi', 'Arkansas']
        k={'Ohio': 'blue', 'Hawaii': 'blue', 'Vermont': 'blue', 'Maine': 'blue', 'Tennessee': 'blue',
        'Oklahoma': 'blue', 'Colorado': 'green', 'Alabama': 'green', 'Oregon': 'blue',
         'Minnesota': 'blue', 'New Mexico': 'red', 'Mississippi': 'red', 'Kansas': 'red',
         'New Hampshire': 'green', 'Louisiana': 'blue', 'Rhode Island': 'blue', 'Montana': 'blue',
         'Wisconsin': 'green', 'Michigan': 'red', 'Arkansas': 'green', 'Maryland': 'blue',
         'Missouri': 'orange', 'Massachusetts': 'red', 'North Dakota': 'green', 'Nevada': 'green',
```

```
'South Dakota': 'orange', 'Illinois': 'blue', 'Washington': 'green', 'Virginia': 'green',
'Indiana': 'green', 'Alaska': 'blue', 'Connecticut': 'green', 'North Carolina': 'red',
'New York': 'orange', 'New Jersey': 'blue', 'Iowa': 'red', 'Kentucky': 'red',
'South Carolina': 'blue', 'West Virginia': 'orange', 'Idaho': 'orange',
'Florida': 'blue', 'Delaware': 'green', 'Nebraska': 'blue', 'Arizona': 'orange',
'Wyoming': 'red', 'California': 'red', 'Utah': 'blue', 'Texas': 'orange', 'Pennsylvania': 'red',
'Georgia': 'orange'}
graph_rest_riction ={
    'Alabama':['Florida', 'Georgia', 'Mississippi', 'Tennessee'],
    'Arizona':['California', 'Colorado', 'Nevada', 'New Mexico', 'Utah'],
    'Arkansas' :['Louisiana', 'Mississippi', 'Missouri', 'Oklahoma', 'Tennessee', 'Texas'],
    'California':['Arizona', 'Nevada', 'Oregon'],
    'Colorado':['Arizona', 'Kansas', 'Nebraska', 'New Mexico', 'Oklahoma', 'Utah',
'Wyoming'],
    'Connecticut':['Massachusetts', 'New York', 'Rhode Island'],
    'Delaware':['Maryland', 'New Jersey', 'Pennsylvania'],
    'Florida':['Alabama', 'Georgia'],
    'Georgia':['Alabama', 'Florida', 'North Carolina', 'South Carolina', 'Tennessee'],
    'Idaho':['Montana', 'Nevada', 'Oregon', 'Utah', 'Washington', 'Wyoming'],
    'Illinois':['Indiana','Iowa', 'Michigan', 'Kentucky', 'Missouri', 'Wisconsin'],
    'Indiana':['Illinois', 'Kentucky', 'Michigan', 'Ohio'],
    'Iowa': ['Illinois', 'Minnesota', 'Missouri', 'Nebraska', 'South Dakota', 'Wisconsin'],
    'Kansas' :['Colorado', 'Missouri', 'Nebraska', 'Oklahoma'],
    'Kentucky':['Illinois', 'Indiana', 'Missouri', 'Ohio', 'Tennessee', 'Virginia', 'West Virginia'],
    'Louisiana':['Arkansas', 'Mississippi', 'Texas'],
    'Maine':["New Hampshire"],
    "Maryland":['Delaware','Pennsylvania','Virginia', 'West Virginia'],
    'Massachusetts':['Connecticut', 'New Hampshire', 'New York', 'Rhode Island',
'Vermont'],
    'Michigan':['Illinois', 'Indiana', 'Minnesota', 'Ohio', 'Wisconsin'],
    'Minnesota':['Iowa', 'Michigan', 'North Dakota', 'South Dakota', 'Wisconsin'],
    'Mississippi':['Alabama', 'Arkansas', 'Louisiana', 'Tennessee'],
```

'Missouri':['Arkansas', 'Illinois', 'Iowa', 'Kansas', 'Kentucky', 'Nebraska', 'Oklahoma', 'Tennessee'],

'Montana':['Idaho', 'North Dakota', 'South Dakota', 'Wyoming'],

'Nebraska' :['Colorado', 'Iowa', 'Kansas', 'Missouri', 'South Dakota', 'Wyoming'],

'Nevada':['Arizona', 'California', 'Idaho', 'Oregon', 'Utah'],

'New Hampshire': ['Maine', 'Massachusetts', 'Vermont'],

'New Jersey':["Delaware", "New York", "Pennsylvania"],

'New Mexico':['Arizona', 'Colorado', 'Oklahoma', 'Texas', 'Utah'],

'New York':['Connecticut', 'Massachusetts', 'New Jersey', 'Pennsylvania', 'Rhode Island', 'Vermont'],

'North Carolina':['Georgia', 'South Carolina', 'Tennessee', 'Virginia'],

'North Dakota':['Minnesota', 'Montana', 'South Dakota'],

'Ohio':['Indiana', 'Kentucky', 'Michigan', 'Pennsylvania', 'West Virginia'],

'Oklahoma' :['Arkansas', 'Colorado', 'Kansas', 'Missouri', 'New Mexico', 'Texas'],

'Oregon':["California", 'Idaho', 'Nevada', "Washington"],

'Pennsylvania':['Delaware', 'Maryland', 'New Jersey', 'New York', 'Ohio', 'West Virginia'],

'Rhode Island':['Connecticut', 'Massachusetts', 'New York'],

'South Carolina':['Georgia', 'North Carolina'],

'South Dakota':['Iowa', 'Minnesota', 'Montana', 'Nebraska', 'North Dakota', 'Wyoming'],

'Tennessee':['Alabama', 'Arkansas', 'Georgia', 'Kentucky', 'Mississippi', 'Missouri', 'North Carolina', 'Virginia'],

'Texas':['Arkansas', 'Louisiana', 'New Mexico', 'Oklahoma'],

'Utah':['Arizona', 'Colorado', 'Idaho', 'Nevada', 'New Mexico', 'Wyoming'],

'Vermont':['Massachusetts', 'New Hampshire', 'New York'],

'Virginia':['Kentucky', 'Maryland', 'North Carolina', 'Tennessee', 'West Virginia'],

'Washington':['Idaho', 'Oregon'],

'West Virginia':['Kentucky', 'Maryland', 'Ohio', 'Pennsylvania', 'Virginia'],

'Wisconsin':['Illinois', 'Iowa', 'Michigan', 'Minnesota'],

'Wyoming':['Colorado', 'Idaho', 'Montana', 'Nebraska', 'South Dakota', 'Utah'],

'Hawaii':[],

'Alaska':[]

}

```
    dm_in=["blue","green","red","orange"]
    st_a_te_obj_ects = set_obj(stat_es,dm_in)
    states_and_colors={}
    for st_a_te in st_a_te_obj_ects:
        key = st_a_te.nme_st
        values=graph_rest_riction[key]
        neigh_brs=[]
        for value in values:
            obj=[obj_form for obj_form in st_a_te_obj_ects if obj_form.nme_st==value ]
            neigh_brs.append(obj[0])
        st_a_te.se_tNeigh(neigh_brs)
    global st
    st=c_s_p(st_a_te_obj_ects,dm_in,states_and_colors)

starting_time = time.time()
main()
ending_time = time.time()

print("Time taken for execution = " + str(ending_time- starting_time) + " seconds")
print("Number of Backtracks = " + str(track_backs))

#displaying the acquired solution

if m==1:
    print(k)
if m==2:
    print(st)
```

With Heuristics:

DFS

```
# This file runs the dfs algorithm on both aus and USA with added singleton propagation


#importing all relevevant files
import random
from Node import Node
import time


# variables that are global


lst_of_clrs = []
sta_tes_all = []
track_backs = 0


#choosing state with largest constraints
def choo_se_large_st(clrs_lega_l):
    for q in sorted(clrs_lega_l, kee=lambda k: len(clrs_lega_l[q]), reverse=True):
        return q



#funct to chnage neighbours
def chnge_neigh(chnged_sta_tes,clrs_lega_l,dic_sta_te,ass_clrs,st_ate_s):
    for st_ate in dic_sta_te[chnged_sta_tes]:
            if ass_clrs in list(filter(lambda x:x[0]==st_ate,clrs_lega_l))[0][1]:
                list(filter(lambda x:x[0]==st_ate,clrs_lega_l))[0][1].remove(ass_clrs)


#funct to set colors randomly
def clrs_set(wn):
    for mm in range(wn):
        lst_of_clrs.append(random.randint(0,255))
```

```
#funct to randomize colors
def rdm_clr(wn):
    return tuple(lst_of_clrs)

#setting up graph with appropriate colors
def setup_grph_clr(nom,stes_n0,st_ate_s):
    clrs = rdm_clr(3)
    ff = Node(clrs[0],st_ate_s[0])
    rt = ff
    mystates = {}
    for mm in range(1,stes_n0,1):
        w = Node(clrs[0],st_ate_s[mm])
        ff.put_child(w)
        ff.nextnode = w
        mystates[st_ate_s[mm]] = ff
        for nn in range(1,nom,1):
            gg = Node(clrs[nn])
            ff.put_child(gg)
        ff = w
    print(rt)
    print(rt.next)
    print(rt.next[1].next)

#funct to retrieve colors
def retrieve_clrs(st_ate_s,st_my_dic):
    col_lst = []
    for mm in st_ate_s:
        col_lst.append(st_my_dic.get(mm,""))

    return col_lst

#funtion to generate columns
```

```
def gen_colus(st_ate_s,mm,numcolors,clrs):
    lst_temp = []
    for nn in range(numcolors):
        lst_temp.append(Node(clrs[nn],st_ate_s[mm]))
    return lst_temp


#function that incorporates heuristics with dfs
def inc_d_f_s_heuristic(st_my_dic,dic_sta_te,numcolors,pres_st,st_ate_s,nom,clrs_lega_l):
    global track_backs


    sta_tes_all.append(st_my_dic.copy())
    for mm in range(len(pres_st.next)):
        lgl_clrs_t = clrs_lega_l.copy()
        st_my_dic[pres_st.next[0].myname] = pres_st.next[mm].mycolor
        if st_my_dic.get(pres_st.next[0].myname) in
retrieve_clrs(dic_sta_te[pres_st.next[0].myname],st_my_dic):
            continue
        if nom == len(st_ate_s) - 1:
            return 1,st_my_dic
        track_backs +=1

chnge_neigh(pres_st.next[0].myname,lgl_clrs_t,dic_sta_te,pres_st.next[mm].mycolor,st_ate_s)
        lgl_clrs_t = sorted(lgl_clrs_t,key=lambda x:len(x[1]))
        clr_lst_temp = lst_of_clrs.copy()
        pres_st.next[mm].next = gen_colus(st_ate_s,nom+1,numcolors,clr_lst_temp)
        res =
inc_d_f_s_heuristic(st_my_dic,dic_sta_te,numcolors,pres_st.next[mm],st_ate_s,nom+1,lgl_clrs_
t)
        if res[0] == 1:
            return 1,st_my_dic
        continue
    return 0,st_my_dic
```

```
#function for initilization
def init(st_ate_s,dic_sta_te,numcolors):
    clrs = lst_of_clrs
    rt = Node(clrs[0],st_ate_s[0])
    for nn in range(numcolors):
        rt.put_child(Node(clrs[nn],st_ate_s[0]))
    return rt

if __name__ == "__main__":
    m=int(input("Choose The Map : 1. USA  2. Australia \n"))
    numcolors = 4
    clrs_set(numcolors)
    #using generic colors
    lst_of_clrs = ["red","blue","green","black"]

    #different states are assigned based on users choice

    if m==1:
        dic_sta_te = {
    'Alabama':['Florida', 'Georgia', 'Mississippi', 'Tennessee'],
    'Arizona':['California', 'Colorado', 'Nevada', 'New Mexico', 'Utah'],
    'Arkansas' :['Louisiana', 'Mississippi', 'Missouri', 'Oklahoma', 'Tennessee', 'Texas'],
    'California':['Arizona', 'Nevada', 'Oregon'],
    'Colorado':['Arizona', 'Kansas', 'Nebraska', 'New Mexico', 'Oklahoma', 'Utah', 'Wyoming'],
    'Connecticut':['Massachusetts', 'New York', 'Rhode Island'],
    'Delaware':['Maryland', 'New Jersey', 'Pennsylvania'],
    'Florida':['Alabama', 'Georgia'],
    'Georgia':['Alabama', 'Florida', 'North Carolina', 'South Carolina', 'Tennessee'],
    'Idaho':['Montana', 'Nevada', 'Oregon', 'Utah', 'Washington', 'Wyoming'],
    'Illinois':['Indiana','Iowa', 'Michigan', 'Kentucky', 'Missouri', 'Wisconsin'],
    'Indiana':['Illinois', 'Kentucky', 'Michigan', 'Ohio'],
```

'Iowa': ['Illinois', 'Minnesota', 'Missouri', 'Nebraska', 'South Dakota', 'Wisconsin'],

'Kansas' :['Colorado', 'Missouri', 'Nebraska', 'Oklahoma'],

'Kentucky':['Illinois', 'Indiana', 'Missouri', 'Ohio', 'Tennessee', 'Virginia', 'West Virginia'],

'Louisiana':['Arkansas', 'Mississippi', 'Texas'],

'Maine':["New Hampshire"],

"Maryland":['Delaware','Pennsylvania','Virginia', 'West Virginia'],

'Massachusetts':['Connecticut', 'New Hampshire', 'New York', 'Rhode Island', 'Vermont'],

'Michigan':['Illinois', 'Indiana', 'Minnesota', 'Ohio', 'Wisconsin'],

'Minnesota':['Iowa', 'Michigan', 'North Dakota', 'South Dakota', 'Wisconsin'],

'Mississippi':['Alabama', 'Arkansas', 'Louisiana', 'Tennessee'],

'Missouri':['Arkansas', 'Illinois', 'Iowa', 'Kansas', 'Kentucky', 'Nebraska', 'Oklahoma', 'Tennessee'],

'Montana':['Idaho', 'North Dakota', 'South Dakota', 'Wyoming'],

'Nebraska' :['Colorado', 'Iowa', 'Kansas', 'Missouri', 'South Dakota', 'Wyoming'],

'Nevada':['Arizona', 'California', 'Idaho', 'Oregon', 'Utah'],

'New Hampshire': ['Maine', 'Massachusetts', 'Vermont'],

'New Jersey':["Delaware", "New York", "Pennsylvania"],

'New Mexico':['Arizona', 'Colorado', 'Oklahoma', 'Texas', 'Utah'],

'New York':['Connecticut', 'Massachusetts', 'New Jersey', 'Pennsylvania', 'Rhode Island', 'Vermont'],

'North Carolina':['Georgia', 'South Carolina', 'Tennessee', 'Virginia'],

'North Dakota':['Minnesota', 'Montana', 'South Dakota'],

'Ohio':['Indiana', 'Kentucky', 'Michigan', 'Pennsylvania', 'West Virginia'],

'Oklahoma' :['Arkansas', 'Colorado', 'Kansas', 'Missouri', 'New Mexico', 'Texas'],

'Oregon':["California", 'Idaho', 'Nevada', "Washington"],

'Pennsylvania':['Delaware', 'Maryland', 'New Jersey', 'New York', 'Ohio', 'West Virginia'],

'Rhode Island':['Connecticut', 'Massachusetts', 'New York'],

'South Carolina':['Georgia', 'North Carolina'],

'South Dakota':['Iowa', 'Minnesota', 'Montana', 'Nebraska', 'North Dakota', 'Wyoming'],

'Tennessee':['Alabama', 'Arkansas', 'Georgia', 'Kentucky', 'Mississippi', 'Missouri', 'North Carolina', 'Virginia'],

'Texas':['Arkansas', 'Louisiana', 'New Mexico', 'Oklahoma'],

'Utah':['Arizona', 'Colorado', 'Idaho', 'Nevada', 'New Mexico', 'Wyoming'],

```
'Vermont':['Massachusetts', 'New Hampshire', 'New York'],
'Virginia':['Kentucky', 'Maryland', 'North Carolina', 'Tennessee', 'West Virginia'],
'Washington':['Idaho', 'Oregon'],
'West Virginia':['Kentucky', 'Maryland', 'Ohio', 'Pennsylvania', 'Virginia'],
'Wisconsin':['Illinois', 'Iowa', 'Michigan', 'Minnesota'],
'Wyoming':['Colorado', 'Idaho', 'Montana', 'Nebraska', 'South Dakota', 'Utah']
}
```

```
    st_ate_s = ['Maine', 'Minnesota', 'South Dakota', 'Illinois', 'Utah', 'Wyoming', 'Texas', 'Idaho',
'Wisconsin', 'Connecticut', 'Pennsylvania', 'Kansas', 'West Virginia', 'North Carolina', 'Colorado',
'California', 'Florida', 'Vermont', 'Virginia', 'North Dakota', 'Michigan', 'New Jersey', 'Nevada',
'Arkansas', 'Mississippi', 'Iowa', 'Kentucky', 'Maryland', 'Louisiana', 'Alabama', 'Oklahoma', 'New
Mexico', 'Rhode Island', 'Massachusetts', 'South Carolina', 'Indiana', 'Delaware', 'Tennessee',
'Georgia', 'Arizona', 'Nebraska', 'Missouri', 'New Hampshire', 'Ohio', 'Oregon', 'Washington',
'Montana', 'New York']
```

```
if m==2:
    st_ate_s=['wa','nt','q','nsw','v','sa']

    dic_sta_te  ={
        'wa':['nt','sa'],
        'nt':['wa','q','sa'],
        'sa':['wa','q','nsw','nt','v'],
        'q':['nt','sa','nsw'],
        'nsw':['q','v','sa'],
        'v':['sa','nsw']}

clrs_lega_l = []

for st_ate in st_ate_s:
    clrs_lega_l.append([st_ate,lst_of_clrs.copy()])

st_my_dic = {}
rt = init(st_ate_s,dic_sta_te,numcolors)
```

```
    starting_time = time.time()
    result = inc_d_f_s_heuristic(st_my_dic,dic_sta_te,numcolors,rt,st_ate_s,0,clrs_lega_l)
    #calculating time required to run the program
    ending_time = time.time()
    n0 = 0
    for kee in result[1]:
        n0+=1
        if result[1][kee] in retrieve_clrs(dic_sta_te[kee],st_my_dic):
            print("Something went Wrong")
    #print(len(sta_tes_all))
    #displaying the acquired solution

    print(result)
    print("Number Of Backtracks: "+ str(track_backs))
    print("TIME TAKEN FOR EXECUTION: " + str(ending_time - starting_time) + "seconds")
```

DFS + forward checking

```
import os
import random
from Node import Node
import pdb
import time


clrs_List = []
Every_St = []
No_of_Backtracks = 0


# Function used to select biggest state for coloring.
def Choose_St_Big(clr_Legal):
    for q in sorted(clr_Legal, key=lambda q: len(clr_Legal[q]), reverse=True):
```

```
        return q

# Function to update the clrs for neighboring states.
def Neigh_Upt(St_chng,clr_Legal,Dict_St,Assigned_clr,Sts):
    for St in Dict_St[St_chng]:
            if Assigned_clr in list(filter(lambda z:z[0]==St,clr_Legal))[0][1]:
                list(filter(lambda z:z[0]==St,clr_Legal))[0][1].remove(Assigned_clr)

# Function to Intialize colors.
def clr_Init(n):
    for m in range(n):
        clrs_List.append(random.randint(0,255))

# Function to gen random colors.
def clr_Random(n):
    return tuple(clrs_List)

# Function to build graph for coloring
def clr_Graph_Bld(no,no_Sts,Sts):
    clrs = clr_Random(3)
    p = Node(clrs[0],Sts[0])
    root = p
    mySts = {}
    for m in range(1,no_Sts,1):
        u = Node(clrs[0],Sts[m])
        p.put_child(u)
        p.nextnode = u
        mySts[Sts[m]] = p

        for n in range(1,no,1):
            k = Node(clrs[n])
            p.put_child(k)
```

```
        p = u

    print(root)
    print(root.next)
    print(root.next[1].next)

def clr_Get(Sts,Dictionary_St):
    clr_List = []
    for m in Sts:
        clr_List.append(Dictionary_St.get(m,""))

    return clr_List

# Function to gen color.
def clr_gen(Sts,m,no_clr,clrs):
    ListT = []
    for n in range(no_clr):
        ListT.append(Node(clrs[n],Sts[m]))

    return ListT

# Function using Heuristic to compute backtracks
def included_Heuristic(Dictionary_St,Dict_St,no_clr,St_Current,Sts,no,clr_Legal):
    global No_of_Backtracks

    Every_St.append(Dictionary_St.copy())
    for m in range(len(St_Current.next)):
        Legal_clr_Temp= clr_Legal.copy()
        Dictionary_St[St_Current.next[0].myname] = St_Current.next[m].mycolor

        if Dictionary_St.get(St_Current.next[0].myname) in
clr_Get(Dict_St[St_Current.next[0].myname],Dictionary_St):
```

```
            continue


        if no == len(Sts) - 1:
            return 1,Dictionary_St



        No_of_Backtracks +=1

Neigh_Upt(St_Current.next[0].myname,Legal_clr_Temp,Dict_St,St_Current.next[m].mycolor,Sts
)


        Legal_clr_Temp= sorted(Legal_clr_Temp,key=lambda z:len(z[1]))
        temp_clrs_List = clrs_List.copy()
        clr_Remove = clr_Get(Legal_clr_Temp[no+1][0],Dictionary_St)
        temp_clrs_List = [z for z in temp_clrs_List if z not in clr_Remove]
        St_Current.next[m].next = clr_gen(Sts,no+1,no_clr,temp_clrs_List)
        Answer =
included_Heuristic(Dictionary_St,Dict_St,no_clr,St_Current.next[m],Sts,no+1,Legal_clr_Temp)
        if Answer[0] == 1:
            return 1,Dictionary_St


        continue


    return 0,Dictionary_St



def init(Sts,Dict_St,no_clr):
    clrs = clrs_List
    root = Node(clrs[0],Sts[0])
    for n in range(no_clr):
        root.put_child(Node(clrs[n],Sts[0]))


    return root
```

```
# Calling Main Function
if __name__ == "__main__":
    global m
    m=int(input("Choose the Map 1. USA 2. Australia\n"))

    if m==1:
        no_clr = 4
        clr_Init(no_clr)
        clrs_List = ["red","blue","green","black"]
        Dict_St = {
    'Alabama':['Florida', 'Georgia', 'Mississippi', 'Tennessee'],
    'Arizona':['California', 'clrado', 'Nevada', 'New Mexico', 'Utah'],
    'Arkansas' :['Louisiana', 'Mississippi', 'Missouri', 'Oklahoma', 'Tennessee', 'Texas'],
    'California':['Arizona', 'Nevada', 'Oregon'],
    'clrado':['Arizona', 'Kansas', 'Nebraska', 'New Mexico', 'Oklahoma', 'Utah', 'Wyoming'],
    'Connecticut':['Massachusetts', 'New York', 'Rhode Island'],
    'Delaware':['Maryland', 'New Jersey', 'Pennsylvania'],
    'Florida':['Alabama', 'Georgia'],
    'Georgia':['Alabama', 'Florida', 'North Carolina', 'South Carolina', 'Tennessee'],
    'Idaho':['Montana', 'Nevada', 'Oregon', 'Utah', 'Washington', 'Wyoming'],
    'Illinois':['Indiana','Iowa', 'Michigan', 'Kentucky', 'Missouri', 'Wisconsin'],
    'Indiana':['Illinois', 'Kentucky', 'Michigan', 'Ohio'],
    'Iowa': ['Illinois', 'Minnesota', 'Missouri', 'Nebraska', 'South Dakota', 'Wisconsin'],
    'Kansas' :['clrado', 'Missouri', 'Nebraska', 'Oklahoma'],
    'Kentucky':['Illinois', 'Indiana', 'Missouri', 'Ohio', 'Tennessee', 'Virginia', 'West Virginia'],
    'Louisiana':['Arkansas', 'Mississippi', 'Texas'],
    'Maine':["New Hampshire"],
    "Maryland":['Delaware','Pennsylvania','Virginia', 'West Virginia'],
    'Massachusetts':['Connecticut', 'New Hampshire', 'New York', 'Rhode Island', 'Vermont'],
    'Michigan':['Illinois', 'Indiana', 'Minnesota', 'Ohio', 'Wisconsin'],
    'Minnesota':['Iowa', 'Michigan', 'North Dakota', 'South Dakota', 'Wisconsin'],
```

'Mississippi':['Alabama', 'Arkansas', 'Louisiana', 'Tennessee'],

'Missouri':['Arkansas', 'Illinois', 'Iowa', 'Kansas', 'Kentucky', 'Nebraska', 'Oklahoma', 'Tennessee'],

'Montana':['Idaho', 'North Dakota', 'South Dakota', 'Wyoming'],

'Nebraska' :['clrado', 'Iowa', 'Kansas', 'Missouri', 'South Dakota', 'Wyoming'],

'Nevada':['Arizona', 'California', 'Idaho', 'Oregon', 'Utah'],

'New Hampshire': ['Maine', 'Massachusetts', 'Vermont'],

'New Jersey':["Delaware", "New York", "Pennsylvania"],

'New Mexico':['Arizona', 'clrado', 'Oklahoma', 'Texas', 'Utah'],

'New York':['Connecticut', 'Massachusetts', 'New Jersey', 'Pennsylvania', 'Rhode Island', 'Vermont'],

'North Carolina':['Georgia', 'South Carolina', 'Tennessee', 'Virginia'],

'North Dakota':['Minnesota', 'Montana', 'South Dakota'],

'Ohio':['Indiana', 'Kentucky', 'Michigan', 'Pennsylvania', 'West Virginia'],

'Oklahoma' :['Arkansas', 'clrado', 'Kansas', 'Missouri', 'New Mexico', 'Texas'],

'Oregon':["California", 'Idaho', 'Nevada', "Washington"],

'Pennsylvania':['Delaware', 'Maryland', 'New Jersey', 'New York', 'Ohio', 'West Virginia'],

'Rhode Island':['Connecticut', 'Massachusetts', 'New York'],

'South Carolina':['Georgia', 'North Carolina'],

'South Dakota':['Iowa', 'Minnesota', 'Montana', 'Nebraska', 'North Dakota', 'Wyoming'],

'Tennessee':['Alabama', 'Arkansas', 'Georgia', 'Kentucky', 'Mississippi', 'Missouri', 'North Carolina', 'Virginia'],

'Texas':['Arkansas', 'Louisiana', 'New Mexico', 'Oklahoma'],

'Utah':['Arizona', 'clrado', 'Idaho', 'Nevada', 'New Mexico', 'Wyoming'],

'Vermont':['Massachusetts', 'New Hampshire', 'New York'],

'Virginia':['Kentucky', 'Maryland', 'North Carolina', 'Tennessee', 'West Virginia'],

'Washington':['Idaho', 'Oregon'],

'West Virginia':['Kentucky', 'Maryland', 'Ohio', 'Pennsylvania', 'Virginia'],

'Wisconsin':['Illinois', 'Iowa', 'Michigan', 'Minnesota'],

'Wyoming':['clrado', 'Idaho', 'Montana', 'Nebraska', 'South Dakota', 'Utah']

}

Sts = ['Illinois', 'Oklahoma', 'California', 'Utah', 'Wyoming', 'Missouri', 'Michigan', 'Texas', 'Iowa', 'Delaware', 'Tennessee', 'Maryland', 'Kentucky', 'Montana', 'Minnesota', 'Connecticut', 'Louisiana', 'West Virginia', 'Pennsylvania', 'Nebraska', 'Kansas', 'Indiana', 'Rhode Island',

'Arizona', 'Florida', 'Massachusetts', 'South Dakota', 'Nevada', 'South Carolina', 'Ohio', 'New Hampshire', 'Idaho', 'Washington', 'clrado', 'Oregon', 'New Jersey', 'Mississippi', 'Arkansas', 'Vermont', 'Wisconsin', 'Alabama', 'Georgia', 'Maine', 'New Mexico', 'North Carolina', 'New York', 'Virginia', 'North Dakota']

```
    if m==2:
        no_clr = 3
        clr_Init(no_clr)

        clrs_List = ["red","blue","green","black"]
        Sts=['wa','nt','q','nsw','v','sa']
        Dict_St  ={
            'wa':['nt','sa'],
            'nt':['wa','q','sa'],
            'sa':['wa','q','nsw','nt','v'],
            'q':['nt','sa','nsw'],
            'nsw':['q','v','sa'],
            'v':['sa','nsw']}

    clr_Legal = []

    for St in Sts:
        clr_Legal.append([St,clrs_List.copy()])

    Dictionary_St = {}
    root = init(Sts,Dict_St,no_clr)

    Time_Start = time.time()
    Answer = included_Heuristic(Dictionary_St,Dict_St,no_clr,root,Sts,0,clr_Legal)

    Time_End = time.time()
    count = 0
```

```
    for key in Answer[1]:
        count+=1
        if Answer[1][key] in clr_Get(Dict_St[key],Dictionary_St):
            print("No Answer")



    time.sleep(10)
    print("Answer Verified")
    print(Answer)
    print("No_of_Backtracks: "+ str(No_of_Backtracks))
    print("Time Taken for Exec: " + str(Time_End - Time_Start) + "seconds")
```

DFS + forward checking + Propagation with Singleton domains

```
import os
import random
from Node import Node
import pdb
from mapcolor import colormap
import time


lst_of_clrs = []
No_of_Back_tracks = 0
Every_St = []


# Class state conatins State name, State colors, status etc.
class State:
    def __init__(self,name,dom,Status="Not_Visited"):
        self.name=name
        self.Neighbor=None
        self.Named_Clrs=None
        self.Status=Status
```

```
        self.dom=dom
        self.Heuristic_singleton=False

    def Neighbor_Set(self,Neighbor):
        self.Neighbor=Neighbor
    def Neighbor_Get(self):
        return self.Neighbor
    def clrs_Set(self,Named_Clrs):
        self.Named_Clrs=Named_Clrs
    def Parents_Set(self,parent):
        self.parent=parent
    def dom_Set(self,dom):
        self.dom=dom
    def dom_Get(self):
        return self.dom
    def Is_Heuristic_Singleton(self):
        if self.Heuristic_singleton:
            return self.Heuristic_singleton
        return False

# Function used to iniz colors
def clr_iniz(n):
    for l in range(n):
        lst_of_clrs.append(random.randint(0,255))

# Function used to generate random clrs
def clrs_Random(n):
    return tuple(lst_of_clrs)

# Function used to build the graph for clring
def clrs_graph_bld(no,no_st,st):
    clrs = clrs_Random(3)
```

```
    c = Node(clrs[0],st[0])
    root = c
    myst = {}
    for l in range(1,no_st,1):
        z = Node(clrs[0],st[l])
        c.put_child(z)
        c.Next_Node = z
        myst[st[l]] = c

        for m in range(1,no,1):
            d = Node(clrs[m])
            c.put_child(d)
        c = z

    print(root)
    print(root.next)
    print(root.next[1].next)

def clr_Get(st,My_State_Dict):
    lstcol = []
    for l in st:
        lstcol.append(My_State_Dict.get(l,""))

    return lstcol

#Function for generating clrs
def clr_gen(st,l,clr_no,clrs):
    ListT = []
    for m in range(clr_no):
        ListT.append(Node(clrs[m],st[l]))

    return ListT
```

```
# Function using Singleton Heuristic method to compute backtracking
def Heuristic_singleton(My_State_Dict,dict_st,clr_no,st_cur,st,no):

    global No_of_Back_tracks
    Every_St.append(My_State_Dict.copy())
    for l in range(len(st_cur.next)):
        My_State_Dict[st_cur.next[0].myname] = st_cur.next[l].mycolor
        if My_State_Dict.get(st_cur.next[0].myname) in
clr_Get(dict_st[st_cur.next[0].myname],My_State_Dict):
            continue

        if no == len(st) - 1:
            return 1,My_State_Dict

        temp_lst_of_clrs = lst_of_clrs.copy()
        clr_Remove = clr_Get(st[no+1],My_State_Dict)
        temp_lst_of_clrs = [y for y in temp_lst_of_clrs if y not in clr_Remove]
        st_cur.next[l].next = clr_gen(st,no+1,clr_no,temp_lst_of_clrs)

        ans = Heuristic_singleton(My_State_Dict,dict_st,clr_no,st_cur.next[l],st,no+1)
        if ans[0] == 1:
            return 1,My_State_Dict

        continue

    No_of_Back_tracks +=1

    return 0,My_State_Dict


def init(st,dict_st,clr_no):
```

```
    clrs = lst_of_clrs
    root = Node(clrs[0],st[0])
    for m in range(clr_no):
        root.put_child(Node(clrs[m],st[0]))

    return root

# Calling Main Function.
if __name__ == "__main__":
    i=int(input("Choose The Map : 1. USA  2. Australia \n"))
    clr_no = 4
    clr_iniz(clr_no)

    lst_of_clrs = ["green","brown","yellow","blue"]
    if i==1:

        dict_st = {
        'Alabama':['Florida', 'Georgia', 'Mississippi', 'Tennessee'],
        'Arizona':['California', 'clrado', 'Nevada', 'New Mexico', 'Utah'],
        'Arkansas' :['Louisiana', 'Mississippi', 'Missouri', 'Oklahoma', 'Tennessee', 'Texas'],
        'California':['Arizona', 'Nevada', 'Oregon'],
        'clrado':['Arizona', 'Kansas', 'Nebraska', 'New Mexico', 'Oklahoma', 'Utah', 'Wyoming'],
        'Connecticut':['Massachusetts', 'New York', 'Rhode Island'],
        'Delaware':['Maryland', 'New Jersey', 'Pennsylvania'],
        'Florida':['Alabama', 'Georgia'],
        'Georgia':['Alabama', 'Florida', 'North Carolina', 'South Carolina', 'Tennessee'],
        'Idaho':['Montana', 'Nevada', 'Oregon', 'Utah', 'Washington', 'Wyoming'],
        'Illinois':['Indiana','Iowa', 'Michigan', 'Kentucky', 'Missouri', 'Wisconsin'],
        'Indiana':['Illinois', 'Kentucky', 'Michigan', 'Ohio'],
        'Iowa': ['Illinois', 'Minnesota', 'Missouri', 'Nebraska', 'South Dakota', 'Wisconsin'],
        'Kansas' :['clrado', 'Missouri', 'Nebraska', 'Oklahoma'],
        'Kentucky':['Illinois', 'Indiana', 'Missouri', 'Ohio', 'Tennessee', 'Virginia', 'West Virginia'],
```

'Louisiana':['Arkansas', 'Mississippi', 'Texas'],

'Maine':["New Hampshire"],

"Maryland":['Delaware','Pennsylvania','Virginia', 'West Virginia'],

'Massachusetts':['Connecticut', 'New Hampshire', 'New York', 'Rhode Island', 'Vermont'],

'Michigan':['Illinois', 'Indiana', 'Minnesota', 'Ohio', 'Wisconsin'],

'Minnesota':['Iowa', 'Michigan', 'North Dakota', 'South Dakota', 'Wisconsin'],

'Mississippi':['Alabama', 'Arkanssas', 'Louisiana', 'Tennessee'],

'Missouri':['Arkansas', 'Illinois', 'Iowa', 'Kansas', 'Kentucky', 'Nebraska', 'Oklahoma', 'Tennessee'],

'Montana':['Idaho', 'North Dakota', 'South Dakota', 'Wyoming'],

'Nebraska' :['clrado', 'Iowa', 'Kansas', 'Missouri', 'South Dakota', 'Wyoming'],

'Nevada':['Arizona', 'California', 'Idaho', 'Oregon', 'Utah'],

'New Hampshire': ['Maine', 'Massachusetts', 'Vermont'],

'New Jersey':["Delaware", "New York", "Pennsylvania"],

'New Mexico':['Arizona', 'clrado', 'Oklahoma', 'Texas', 'Utah'],

'New York':['Connecticut', 'Massachusetts', 'New Jersey', 'Pennsylvania', 'Rhode Island', 'Vermont'],

'North Carolina':['Georgia', 'South Carolina', 'Tennessee', 'Virginia'],

'North Dakota':['Minnesota', 'Montana', 'South Dakota'],

'Ohio':['Indiana', 'Kentucky', 'Michigan', 'Pennsylvania', 'West Virginia'],

'Oklahoma' :['Arkansas', 'clrado', 'Kansas', 'Missouri', 'New Mexico', 'Texas'],

'Oregon':["California", 'Idaho', 'Nevada', "Washington"],

'Pennsylvania':['Delaware', 'Maryland', 'New Jersey', 'New York', 'Ohio', 'West Virginia'],

'Rhode Island':['Connecticut', 'Massachusetts', 'New York'],

'South Carolina':['Georgia', 'North Carolina'],

'South Dakota':['Iowa', 'Minnesota', 'Montana', 'Nebraska', 'North Dakota', 'Wyoming'],

'Tennessee':['Alabama', 'Arkansas', 'Georgia', 'Kentucky', 'Mississippi', 'Missouri', 'North Carolina', 'Virginia'],

'Texas':['Arkansas', 'Louisiana', 'New Mexico', 'Oklahoma'],

'Utah':['Arizona', 'clrado', 'Idaho', 'Nevada', 'New Mexico', 'Wyoming'],

'Vermont':['Massachusetts', 'New Hampshire', 'New York'],

'Virginia':['Kentucky', 'Maryland', 'North Carolina', 'Tennessee', 'West Virginia'],

'Washington':['Idaho', 'Oregon'],

```
'West Virginia':['Kentucky', 'Maryland', 'Ohio', 'Pennsylvania', 'Virginia'],
'Wisconsin':['Illinois', 'Iowa', 'Michigan', 'Minnesota'],
'Wyoming':['clrado', 'Idaho', 'Montana', 'Nebraska', 'South Dakota', 'Utah'],
''Hawai'':[],
''Alaska'':[]
}

    st = ['New Hampshire', 'Oklahoma', 'Tennessee', 'Illinois', 'New Mexico', 'Kentucky',
'West Virginia', 'Maryland', 'Maine', 'Wisconsin', 'Missouri', 'Minnesota', 'Montana',
'Massachusetts', 'South Carolina', 'North Dakota', 'Pennsylvania', 'Arizona', 'South Dakota',
'Ohio', 'Oregon', 'Alabama', 'Indiana', 'Rhode Island', 'Virginia', 'Idaho', 'Nevada', 'Nebraska',
'New York', 'Utah', 'Michigan', 'Kansas', 'Florida', 'Connecticut', 'Iowa', 'Wyoming', 'Louisiana',
'California', 'Vermont', 'Texas', 'Georgia', 'New Jersey', 'North Carolina', 'Washington',
'Delaware', 'clrado', 'Mississippi', 'Arkansas']


    st = ['Kansas', 'New Hampshire', 'Idaho', 'Louisiana', 'New Jersey', 'Arkansas',
'Kentucky', 'Maine', 'Minnesota', 'Missouri',
        'West Virginia', 'North Carolina', 'Massachusetts', 'Michigan', 'Indiana', 'Illinois',
'Virginia', 'Oklahoma', 'Montana',
        'North Dakota', 'Texas', 'clrado', 'South Carolina', 'Maryland', 'California', 'New York',
'Florida', 'Vermont', 'Utah',
        'Georgia', 'Oregon', 'Wisconsin', 'Rhode Island', 'Nebraska', 'New Mexico',
'Mississippi', 'Alabama', 'Nevada', 'Tennessee',
        'Iowa','South Dakota', 'Ohio', 'Pennsylvania', 'Washington', 'Wyoming', 'Arizona',
'Delaware', 'Connecticut']
  else:
    st=['wa','nt','q','nsw','v','sa']
    dict_st  ={
      'wa':['nt','sa'],
      'nt':['wa','q','sa'],
      'sa':['wa','q','nsw','nt','v'],
      'q':['nt','sa','nsw'],
      'nsw':['q','v','sa'],
      'v':['sa','nsw']}
```

```
My_State_Dict = {}
print(st)

root = init(st,dict_st,clr_no)

Time_Start = time.time()
Ans = Heuristic_singleton(My_State_Dict,dict_st,clr_no,root,st,0)
end_time = time.time()
count = 0
for key in Ans[1]:
    count+=1
    if Ans[1][key] in clr_Get(dict_st[key],My_State_Dict):
        print("No Answer")



print("Answer that's Verified")
print(Ans)
print("No_of_Back_tracks: "+ str(No_of_Back_tracks))
print("Total Time taken for Execution: " + str(end_time - Time_Start) + "seconds")

print(len(Every_St))
time.sleep(10)
```

References:

- https://github.com/
- https://en.wikipedia.org
- https://www.youtube.com
- https://stackoverflow.com