In [42]:
```
# ITCS 6150
# Noah Foster
# Project 3
print("Noah Foster's ITCS 6150 Project 3")
```

```
Noah Foster's ITCS 6150 Project 3
```

In [10]:
```
# Helper to determine if color is a conflicting choice...
def isAdjacentColor(state, color, colors, adjacentLists):
    adjacent = adjacentLists[state]
    for each in adjacent:
        if color == colors[each]:
            return True
    return False


# Helper to find the first singleton (in the order)
def findSingleton(order, possibleColors):
    for each in order:
        if len(possibleColors[each])==1:
            return each
    return False



# test line (prints True)
#print(isAdjacentColor("a",0,{"a":-1,"b":0,"c":-1},{"a":["b"],"b":["a"],"c":[]}))
```

In [43]:
```
def recurDFS(adjacentLists, order, colors, possibleColors): # a recursive basic DFS fun
    backtrack = 0
    if (len(order)==0): # our base case, we don't have any left to color
        #print("DONE!")
        return colors, 0
    curr = order[0] # the state we are coloring
    for each in possibleColors[curr]: # we want to loop through possible colors for the
        isAdjacent = isAdjacentColor(curr, each, colors, adjacentLists)
        if (isAdjacent==False):
            neworder = order.copy() # copy order so we don't destroy it
            popped = neworder.pop(0) # remove the state we just colored from the new or
            newcolors = {} # copy colors so we don't destroy it
            for each2 in colors:
                newcolors[each2]=colors[each2]
            newcolors[curr]=each
            answer, backs = recurDFS(adjacentLists, neworder, newcolors, possibleColors
            if answer!=False: # this branch didn't fail, passing answer up
                return answer, backs+backtrack
            else:
                backtrack+=backs
        else:
            backtrack+=1
    return False, backtrack # we ran out of colors for the state



def ColorDepthFirst(adjacentLists, orderReference): # helper to ready the data for the
    colors = {} # initializing variables
```

```
    maximum = 0
    done = False
    backtracks = 0
    while done!=True:
        for each in adjacentLists: # Intialize Colors as -1 to indicate blank
            colors[each]=-1
        possible = [] # set possible colors based on current max value
        for each in range(0,maximum+1):
            possible.append(each)
        #print(possible)
        possibleColors = {} # assign arrays of possible color values in dictionary
        for each in adjacentLists:
            possibleColors[each]=possible.copy()
        order = orderReference.copy() # make a copy of our priority order so we don't s
        answer, back = recurDFS(adjacentLists, order, colors, possibleColors)
        if answer==False: # the algorithm failed with this many colors, try one more
            maximum=maximum+1
            #print(maximum)
            #print("Backtracks: "+str(back))
        else:
            colors = answer # the algorithm didn't fail, assign and return.
            done=True
        backtracks+=back
    return colors, backtracks




def recurDFSForward(adjacentLists, order, colors, possibleColors): # a recursive DFS fu
    backtrack = 0 # initialize backtrack counter
    if (len(order)==0): # our base case, we don't have any left to color
        #print("DONE!")
        return colors, 0
    curr = order[0] # the state we are coloring
    if len(possibleColors[curr])==0: # if this state has no possible colors, we are abo
        backtrack+=1
    for each in possibleColors[curr]: # we want to loop through possible colors for the
        isAdjacent = isAdjacentColor(curr, each, colors, adjacentLists)
        if (isAdjacent==False):
            neworder = order.copy() # copy order so we don't destroy it
            popped = neworder.pop(0) # remove the state we just colored from the new or
            newcolors = {} # copy colors so we don't destroy it
            for each2 in colors:
                newcolors[each2]=colors[each2]
            newcolors[curr]=each
            newPossibleColors={} # copy possible colors so we don't destroy it
            for each2 in possibleColors:
                newPossibleColors[each2]=possibleColors[each2].copy()
            adjacents=adjacentLists[curr] # find and check adjacent states (forward che
            for each2 in adjacents:
                if each in newPossibleColors[each2]:
                    newPossibleColors[each2].remove(each)
            answer, backs = recurDFSForward(adjacentLists, neworder, newcolors, newPoss
            if answer!=False: # this branch didn't fail, passing answer up, adding up b
                return answer, backs+backtrack
            else:
                backtrack+=backs
        else:
            backtrack+=1
    return False, backtrack # we ran out of colors for the state. pass up backtrack cou
```

```
def ColorDepthFirstForward(adjacentLists, orderReference): # helper to ready the data f
    colors = {} # initializing variables
    maximum = 0
    done = False
    backtracks = 0
    while done!=True:
        for each in adjacentLists: # Intialize Colors as -1 to indicate blank
            colors[each]=-1
        possible = [] # set possible colors based on current max value
        for each in range(0,maximum+1):
            possible.append(each)
        #print(possible)
        possibleColors = {} # assign arrays of possible color values in dictionary
        for each in adjacentLists:
            possibleColors[each]=possible.copy()
        order = orderReference.copy() # make a copy of our priority order so we don't s
        answer, back = recurDFSForward(adjacentLists, order, colors, possibleColors)
        if answer==False: # the algorithm failed with this many colors, try one more
            maximum=maximum+1
            #print(maximum)
            #print("Backtracks: "+str(back))
        else:
            colors = answer # the algorithm didn't fail, assign and return.
            done=True
        backtracks+=back
    return colors, backtracks




def recurDFSForwardSingleton(adjacentLists, order, colors, possibleColors): # a recursi
    backtrack = 0
    if (len(order)==0): # our base case, we don't have any left to color
        #print("DONE!") # line used for testing
        return colors, 0
    singleton = findSingleton(order,possibleColors)
    if singleton==False:
        curr = order[0] # the state we are coloring
    else:
        # print("Singleton Found") # line used for initial testing
        curr = singleton
    if len(possibleColors[curr])==0: # if this state has no possible colors, we are abo
        backtrack+=1
    for each in possibleColors[curr]: # we want to loop through possible colors for the
        isAdjacent = isAdjacentColor(curr, each, colors, adjacentLists)
        if (isAdjacent==False):
            neworder = order.copy() # copy order so we don't destroy it
            popped = neworder.remove(curr) # remove the state we just colored from the
            newcolors = {} # copy colors so we don't destroy it
            for each2 in colors:
                newcolors[each2]=colors[each2]
            newcolors[curr]=each
            newPossibleColors={} # copy possible colors so we don't destroy it
            for each2 in possibleColors:
                newPossibleColors[each2]=possibleColors[each2].copy()
            adjacents=adjacentLists[curr] # find and check adjacent states (forward che
            for each2 in adjacents:
```

```
                if each in newPossibleColors[each2]:
                    newPossibleColors[each2].remove(each)
            answer, backs = recurDFSForwardSingleton(adjacentLists, neworder, newcolors
            if answer!=False: # this branch didn't fail, passing answer up
                return answer, backs+backtrack
            else:
                backtrack+=backs # counting backtracks for failure purposes
        else:
            backtrack+=1
    return False, backtrack # we ran out of colors for the state




def ColorDepthFirstForwardSingleton(adjacentLists, orderReference): # helper to ready t
    colors = {} # initializing variables
    maximum = 0
    done = False
    backtracks = 0
    while done!=True:
        for each in adjacentLists: # Intialize Colors as -1 to indicate blank
            colors[each]=-1
        possible = [] # set possible colors based on current max value
        for each in range(0,maximum+1):
            possible.append(each)
        #print(possible)
        possibleColors = {} # assign arrays of possible color values in dictionary
        for each in adjacentLists:
            possibleColors[each]=possible.copy()
        order = orderReference.copy() # make a copy of our priority order so we don't s
        answer, back = recurDFSForwardSingleton(adjacentLists, order, colors, possibleC
        if answer==False: # the algorithm failed with this many colors, try one more
            maximum=maximum+1

            #print(maximum)
            #print("Backtracks: "+str(back))
        else:
            colors = answer # the algorithm didn't fail, assign and return.
            done=True
        backtracks+=back
    return colors, backtracks
```

In [32]:

```
import heapq
# Helper to Heuristic-Sort the order list
def chooseState(order, adjacentLists, possibleColors):
    newOrder=[]
    heapq.heapify(newOrder)
    for each in order:
        index = order.index(each)
        numColors = len(possibleColors[each]) # MRV
        numAdjacent = 0
        for each2 in adjacentLists[each]:
            if each2 in order:
                numAdjacent+=1
        heapq.heappush(newOrder, (numColors, -1*numAdjacent, each))
```

```
    retNewOrder=[]
    for each in newOrder:
        (coln, adjn, val) = each
        retNewOrder.append(val)
    return retNewOrder

def chooseColors(state, adjacentLists, order, possibleColors):
    possibles = possibleColors[state]
    sortPossibles = []
    heapq.heapify(sortPossibles)
    adjacents = adjacentLists[state]
    for color in possibles:
        rulesout = 0
        for each in adjacents:
            if each in order:
                if color in possibleColors[each]:
                    rulesout+=1
        heapq.heappush(sortPossibles, (rulesout,color))
    retPossibles = []
    for each in sortPossibles:
        (rulesN,color)=each
        retPossibles.append(color)

    return retPossibles

# Helper to find the list of singletons
def findSingletons(order, possibleColors):
    singletons = []
    for each in order:
        if len(possibleColors[each])==1:
            singletons.append(each)
    if len(singletons)==0:
        return False
    else:
        return singletons
```

In [47]:

```
def recurDFSH(adjacentLists, order, colors, possibleColors): # a recursive basic DFS fu
    backtrack = 0
    if (len(order)==0): # our base case, we don't have any left to color
        #print("DONE!")
        return colors, 0
    order = order.copy()
    order = chooseState(order, adjacentLists, possibleColors) # running heuristic sort
    curr = order[0] # the state we are coloring
    possSort = chooseColors(curr, adjacentLists, order, possibleColors)
    for each in possSort: # we want to loop through possible colors for the state until
        isAdjacent = isAdjacentColor(curr, each, colors, adjacentLists)
        if (isAdjacent==False):
            neworder = order.copy() # copy order so we don't destroy it
            popped = neworder.pop(0) # remove the state we just colored from the new or
            newcolors = {} # copy colors so we don't destroy it
            for each2 in colors:
                newcolors[each2]=colors[each2]
            newcolors[curr]=each
            answer, backs = recurDFSH(adjacentLists, neworder, newcolors, possibleColor
            if answer!=False: # this branch didn't fail, passing answer up
                return answer, backs+backtrack
```

```
            else:
                backtrack+=backs
        else:
            backtrack+=1
    return False, backtrack # we ran out of colors for the state



def ColorDepthFirstH(adjacentLists, orderReference): # helper to ready the data for the
    colors = {} # initializing variables
    maximum = 0
    done = False
    backtracks = 0
    while done!=True:
        for each in adjacentLists: # Intialize Colors as -1 to indicate blank
            colors[each]=-1
        possible = [] # set possible colors based on current max value
        for each in range(0,maximum+1):
            possible.append(each)
        #print(possible)
        possibleColors = {} # assign arrays of possible color values in dictionary
        for each in adjacentLists:
            possibleColors[each]=possible.copy()
        order = orderReference.copy() # make a copy of our priority order so we don't s
        answer, back = recurDFSH(adjacentLists, order, colors, possibleColors)
        if answer==False: # the algorithm failed with this many colors, try one more
            maximum=maximum+1
            #print(maximum)
            #print("Backtracks: "+str(back))
        else:
            colors = answer # the algorithm didn't fail, assign and return.
            done=True
        backtracks+=back
    return colors, backtracks



def recurDFSForwardH(adjacentLists, order, colors, possibleColors): # a recursive DFS f
    backtrack = 0 # initialize backtrack counter
    if (len(order)==0): # our base case, we don't have any left to color
        #print("DONE!")
        return colors, 0
    order = order.copy()
    order = chooseState(order, adjacentLists, possibleColors) # running heuristic sort
    curr = order[0] # the state we are coloring
    if len(possibleColors[curr])==0: # if this state has no possible colors, we are abo
        backtrack+=1
    possSort = chooseColors(curr, adjacentLists, order, possibleColors)
    for each in possSort: # we want to loop through possible colors for the state until
        isAdjacent = isAdjacentColor(curr, each, colors, adjacentLists)
        if (isAdjacent==False):
            neworder = order.copy() # copy order so we don't destroy it
            popped = neworder.pop(0) # remove the state we just colored from the new or
            newcolors = {} # copy colors so we don't destroy it
            for each2 in colors:
                newcolors[each2]=colors[each2]
            newcolors[curr]=each
            newPossibleColors={} # copy possible colors so we don't destroy it
            for each2 in possibleColors:
```

```
                newPossibleColors[each2]=possibleColors[each2].copy()
            adjacents=adjacentLists[curr] # find and check adjacent states (forward che
            for each2 in adjacents:
                if each in newPossibleColors[each2]:
                    newPossibleColors[each2].remove(each)
            answer, backs = recurDFSForwardH(adjacentLists, neworder, newcolors, newPos
            if answer!=False: # this branch didn't fail, passing answer up, adding up b
                return answer, backs+backtrack
            else:
                backtrack+=backs
        else:
            backtrack+=1
    return False, backtrack # we ran out of colors for the state. pass up backtrack cou



def ColorDepthFirstForwardH(adjacentLists, orderReference): # helper to ready the data
    colors = {} # initializing variables
    maximum = 0
    done = False
    backtracks = 0
    while done!=True:
        for each in adjacentLists: # Intialize Colors as -1 to indicate blank
            colors[each]=-1
        possible = [] # set possible colors based on current max value
        for each in range(0,maximum+1):
            possible.append(each)
        #print(possible)
        possibleColors = {} # assign arrays of possible color values in dictionary
        for each in adjacentLists:
            possibleColors[each]=possible.copy()
        order = orderReference.copy() # make a copy of our priority order so we don't s
        answer, back = recurDFSForwardH(adjacentLists, order, colors, possibleColors)
        if answer==False: # the algorithm failed with this many colors, try one more
            maximum=maximum+1
            #print(maximum)
            #print("Backtracks: "+str(back))
        else:
            colors = answer # the algorithm didn't fail, assign and return.
            done=True
            backtracks+=back
    return colors, backtracks



def recurDFSForwardSingletonH(adjacentLists, order, colors, possibleColors): # a recurs
    backtrack = 0
    if (len(order)==0): # our base case, we don't have any left to color
        #print("DONE!") # line used for testing
        return colors, 0
    singletons = findSingletons(order,possibleColors)
    if singletons==False:
        order = order.copy()
        order = chooseState(order, adjacentLists, possibleColors) # running heuristic s
        curr = order[0] # the state we are coloring
    else:
        # print("Singleton Found") # line used for initial testing
        singletons = singletons.copy()
```

```
        singletons = chooseState(singletons, adjacentLists, possibleColors) # running h
        curr = singletons[0]
    if len(possibleColors[curr])==0: # if this state has no possible colors, we are abo
        backtrack+=1
    possSort = chooseColors(curr, adjacentLists, order, possibleColors)
    for each in possSort: # we want to loop through possible colors for the state until
        isAdjacent = isAdjacentColor(curr, each, colors, adjacentLists)
        if (isAdjacent==False):
            neworder = order.copy() # copy order so we don't destroy it
            popped = neworder.remove(curr) # remove the state we just colored from the
            newcolors = {} # copy colors so we don't destroy it
            for each2 in colors:
                newcolors[each2]=colors[each2]
            newcolors[curr]=each
            newPossibleColors={} # copy possible colors so we don't destroy it
            for each2 in possibleColors:
                newPossibleColors[each2]=possibleColors[each2].copy()
            adjacents=adjacentLists[curr] # find and check adjacent states (forward che
            for each2 in adjacents:
                if each in newPossibleColors[each2]:
                    newPossibleColors[each2].remove(each)
            answer, backs = recurDFSForwardSingletonH(adjacentLists, neworder, newcolor
            if answer!=False: # this branch didn't fail, passing answer up
                return answer, backs+backtrack
            else:
                backtrack+=backs # counting backtracks for failure purposes
        else:
            backtrack+=1
    return False, backtrack # we ran out of colors for the state



def ColorDepthFirstForwardSingletonH(adjacentLists, orderReference): # helper to ready
    colors = {} # initializing variables
    maximum = 0
    done = False
    backtracks = 0
    while done!=True:
        for each in adjacentLists: # Intialize Colors as -1 to indicate blank
            colors[each]=-1
        possible = [] # set possible colors based on current max value
        for each in range(0,maximum+1):
            possible.append(each)
        #print(possible)
        possibleColors = {} # assign arrays of possible color values in dictionary
        for each in adjacentLists:
            possibleColors[each]=possible.copy()
        order = orderReference.copy() # make a copy of our priority order so we don't s
        answer, back = recurDFSForwardSingletonH(adjacentLists, order, colors, possible
        if answer==False: # the algorithm failed with this many colors, try one more
            maximum=maximum+1
            #print(maximum)
            #print("Backtracks: "+str(back))
        else:
            colors = answer # the algorithm didn't fail, assign and return.
            done=True
        backtracks+=back
    return colors, backtracks
```

In [41]:

```
# I tried using geopandas for visualization, but its dependencies didn't want to instal

print("Loading Data and Performing Sanity Check...")
# America Adjacent States Data
AmericaAdjacentStates = {
    "Alabama":["Mississippi", "Tennessee", "Florida", "Georgia"],
    "Alaska":[],
    "Arizona":["Nevada", "New Mexico", "Utah", "California", "Colorado"],
    "Arkansas":["Oklahoma","Tennessee","Texas","Louisiana","Mississippi","Missouri"],
    "California":["Oregon","Arizona","Nevada"],
    "Colorado":["New Mexico","Oklahoma","Utah","Wyoming","Arizona","Kansas","Nebraska"]
    "Connecticut":["New York","Rhode Island","Massachusetts"],
    "Delaware":["New Jersey","Pennsylvania","Maryland"],
    "Florida":["Georgia","Alabama"],
    "Georgia":["North Carolina","South Carolina","Tennessee","Alabama","Florida"],
    "Hawaii":[],
    "Idaho":["Utah", "Washington","Wyoming","Montana","Nevada","Oregon"],
    "Illinois":["Kentucky","Missouri","Wisconsin","Indiana","Iowa","Michigan"],
    "Indiana":["Michigan","Ohio","Illinois","Kentucky"],
    "Iowa":["Nebraska","South Dakota","Wisconsin","Illinois","Minnesota","Missouri"],
    "Kansas":["Nebraska","Oklahoma","Colorado","Missouri"],
    "Kentucky":["Tennessee","Virginia","West Virginia","Illinois","Indiana","Missouri",
    "Louisiana":["Texas","Arkansas","Mississippi"],
    "Maine":["New Hampshire"],
    "Maryland":["Virginia","West Virginia","Delaware","Pennsylvania"],
    "Massachusetts":["New York","Rhode Island","Vermont","Connecticut","New Hampshire"]
    "Michigan":["Ohio","Wisconsin","Illinois","Indiana","Minnesota"],
    "Minnesota":["North Dakota","South Dakota","Wisconsin","Iowa","Michigan"],
    "Mississippi":["Louisiana","Tennessee","Alabama","Arkansas"],
    "Missouri":["Nebraska","Oklahoma","Tennessee","Arkansas","Illinois","Iowa","Kansas"
    "Montana":["South Dakota","Wyoming","Idaho","North Dakota"],
    "Nebraska":["Missouri","South Dakota","Wyoming","Colorado","Iowa","Kansas"],
    "Nevada":["Idaho","Oregon","Utah","Arizona","California"],
    "New Hampshire":["Vermont","Maine","Massachusetts"],
    "New Jersey":["Pennsylvania","Delaware","New York"],
    "New Mexico":["Oklahoma","Texas","Utah","Arizona","Colorado"],
    "New York":["Pennsylvania","Rhode Island","Vermont","Connecticut","Massachusetts","
    "North Carolina":["Tennessee","Virginia","Georgia","South Carolina"],
    "North Dakota":["South Dakota","Minnesota","Montana"],
    "Ohio":["Michigan","Pennsylvania","West Virginia","Indiana","Kentucky"],
    "Oklahoma":["Missouri","New Mexico","Texas","Arkansas","Colorado","Kansas"],
    "Oregon":["Nevada","Washington","California","Idaho"],
    "Pennsylvania":["New York","Ohio","West Virginia","Delaware","Maryland","New Jersey
    "Rhode Island":["Massachusetts","New York","Connecticut"],
    "South Carolina":["North Carolina","Georgia"],
    "South Dakota":["Nebraska","North Dakota","Wyoming","Iowa","Minnesota","Montana"],
    "Tennessee":["Mississippi","Missouri","North Carolina","Virginia","Alabama","Arkans
    "Texas":["New Mexico","Oklahoma","Arkansas","Louisiana"],
    "Utah":["Nevada","New Mexico","Wyoming","Arizona","Colorado","Idaho"],
    "Vermont":["New Hampshire","New York","Massachusetts"],
    "Virginia":["North Carolina","Tennessee","West Virginia","Kentucky","Maryland"],
    "Washington":["Oregon","Idaho"],
    "West Virginia":["Pennsylvania","Virginia","Kentucky","Maryland","Ohio"],
    "Wisconsin":["Michigan","Minnesota","Illinois","Iowa"],
    "Wyoming":["Nebraska","South Dakota","Utah","Colorado","Idaho","Montana"]
}
```

```
# Checking List Validity for spelling errors and length errors
AmericaStates={}
for each in AmericaAdjacentStates:
    AmericaStates[each] = True
    for each2 in AmericaAdjacentStates[each]:
        AmericaStates[each2]=True
print("Number of Unique States in America List: " + str(len(AmericaStates)))


# Australia Adjacent States Data
AustraliaAdjacentStates = {
    "New South Wales":["South Australia","Victoria","Queensland"],
    "Northern Territory":["South Australia","Queensland","Western Australia"],
    "Queensland":["New South Wales","South Australia","Northern Territory"],
    "South Australia":["New South Wales","Northern Territory","Queensland","Victoria","
    "Victoria":["New South Wales", "South Australia"],
    "Western Australia":["Northern Territory","South Australia"],
    "Tasmania":[]
}

# Checking List Validity for spelling errors and length errors
AustraliaStates={}
for each in AustraliaAdjacentStates:
    AustraliaStates[each] = True
    for each2 in AustraliaAdjacentStates[each]:
        AustraliaStates[each2]=True
print("Number of Unique States in Australia List: " + str(len(AustraliaStates)))
print("\n")
```

```
Loading Data and Performing Sanity Check...
Number of Unique States in America List: 50
Number of Unique States in Australia List: 7
```

In [56]:

```
import random
import timeit

print("This program colors maps of Australia and the US using CSP techniques")
print("Running on Four Random Australia Arrangements without Heuristic:")
AustraliaOrder1 = []
for each in AustraliaAdjacentStates:
    AustraliaOrder1.append(each)
random.shuffle(AustraliaOrder1)
AustraliaOrder2 = []
for each in AustraliaAdjacentStates:
    AustraliaOrder2.append(each)
random.shuffle(AustraliaOrder2)
AustraliaOrder3 = []
for each in AustraliaAdjacentStates:
    AustraliaOrder3.append(each)
random.shuffle(AustraliaOrder3)
AustraliaOrder4 = []
for each in AustraliaAdjacentStates:
    AustraliaOrder4.append(each)
random.shuffle(AustraliaOrder4)
print()
print("Order 1: \n"+str(AustraliaOrder1))
print("Order 2: \n"+str(AustraliaOrder2))
```

```
print("Order 3: \n"+str(AustraliaOrder3))
print("Order 4: \n"+str(AustraliaOrder4))


start = timeit.default_timer()
AustraliaColored1, backtracks1 = ColorDepthFirst(AustraliaAdjacentStates, AustraliaOrde
end = timeit.default_timer()
time1 = end-start

start = timeit.default_timer()
AustraliaColored2, backtracks2 = ColorDepthFirst(AustraliaAdjacentStates, AustraliaOrde
end = timeit.default_timer()
time2 = end-start

start = timeit.default_timer()
AustraliaColored3, backtracks3 = ColorDepthFirst(AustraliaAdjacentStates, AustraliaOrde
end = timeit.default_timer()
time3 = end-start

start = timeit.default_timer()
AustraliaColored4, backtracks4 = ColorDepthFirst(AustraliaAdjacentStates, AustraliaOrde
end = timeit.default_timer()
time4 = end-start

print()
print("Order 1 DFS:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AustraliaColored1)
print()
print("Order 2 DFS:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AustraliaColored2)
print()
print("Order 3 DFS:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AustraliaColored3)
print()
print("Order 4 DFS:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AustraliaColored4)
print()
avAusDFSback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAusDFStime = (time1+time2+time3+time4)/4

start = timeit.default_timer()
AustraliaColored1, backtracks1 = ColorDepthFirstForward(AustraliaAdjacentStates, Austra
end = timeit.default_timer()
time1 = end-start

start = timeit.default_timer()
AustraliaColored2, backtracks2 = ColorDepthFirstForward(AustraliaAdjacentStates, Austra
end = timeit.default_timer()
time2 = end-start

start = timeit.default_timer()
AustraliaColored3, backtracks3 = ColorDepthFirstForward(AustraliaAdjacentStates, Austra
end = timeit.default_timer()
time3 = end-start

start = timeit.default_timer()
AustraliaColored4, backtracks4 = ColorDepthFirstForward(AustraliaAdjacentStates, Austra
```

```
end = timeit.default_timer()
time4 = end-start

print()
print("Order 1 DFS w/ Forward:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AustraliaColored1)
print()
print("Order 2 DFS w/ Forward:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AustraliaColored2)
print()
print("Order 3 DFS w/ Forward:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AustraliaColored3)
print()
print("Order 4 DFS w/ Forward:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AustraliaColored4)
print()
avAusDFSFback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAusDFSFtime = (time1+time2+time3+time4)/4

start = timeit.default_timer()
AustraliaColored1, backtracks1 = ColorDepthFirstForwardSingleton(AustraliaAdjacentState
end = timeit.default_timer()
time1 = end-start

start = timeit.default_timer()
AustraliaColored2, backtracks2 = ColorDepthFirstForwardSingleton(AustraliaAdjacentState
end = timeit.default_timer()
time2 = end-start

start = timeit.default_timer()
AustraliaColored3, backtracks3 = ColorDepthFirstForwardSingleton(AustraliaAdjacentState
end = timeit.default_timer()
time3 = end-start

start = timeit.default_timer()
AustraliaColored4, backtracks4 = ColorDepthFirstForwardSingleton(AustraliaAdjacentState
end = timeit.default_timer()
time4 = end-start

print()
print("Order 1 DFS w/ Forward and Singleton:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AustraliaColored1)
print()
print("Order 2 DFS w/ Forward and Singleton:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AustraliaColored2)
print()
print("Order 3 DFS w/ Forward and Singleton:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AustraliaColored3)
print()
print("Order 4 DFS w/ Forward and Singleton:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AustraliaColored4)
print()
```

```
avAusDFSFSback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAusDFSFStime = (time1+time2+time3+time4)/4
```

```
This program colors maps of Australia and the US using CSP techniques
Running on Four Random Australia Arrangements without Heuristic:

Order 1:
['New South Wales', 'Tasmania', 'Northern Territory', 'South Australia', 'Queensland',
'Western Australia', 'Victoria']
Order 2:
['Northern Territory', 'Tasmania', 'South Australia', 'Western Australia', 'Queensland',
'Victoria', 'New South Wales']
Order 3:
['Queensland', 'Western Australia', 'Northern Territory', 'Victoria', 'New South Wales',
'South Australia', 'Tasmania']
Order 4:
['Northern Territory', 'Tasmania', 'Queensland', 'New South Wales', 'Victoria', 'South A
ustralia', 'Western Australia']

Order 1 DFS:
Took 9.87999992503319e-05 seconds and 28 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 2, 'South Australia': 1,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 2 DFS:
Took 7.639999967068434e-05 seconds and 22 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 2, 'South Australia': 1,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 3 DFS:
Took 7.779999941703863e-05 seconds and 21 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 0, 'South Australia': 2,
'Victoria': 0, 'Western Australia': 0, 'Tasmania': 0}

Order 4 DFS:
Took 8.380000144825317e-05 seconds and 26 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 1, 'South Australia': 2,
'Victoria': 1, 'Western Australia': 1, 'Tasmania': 0}


Order 1 DFS w/ Forward:
Took 0.0002597999991849065 seconds and 9 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 2, 'South Australia': 1,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 2 DFS w/ Forward:
Took 0.00020819999917875975 seconds and 6 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 2, 'South Australia': 1,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 3 DFS w/ Forward:
Took 0.00021290000222506933 seconds and 7 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 0, 'South Australia': 2,
'Victoria': 0, 'Western Australia': 0, 'Tasmania': 0}

Order 4 DFS w/ Forward:
Took 0.00017160000061267056 seconds and 5 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 1, 'South Australia': 2,
'Victoria': 1, 'Western Australia': 1, 'Tasmania': 0}


Order 1 DFS w/ Forward and Singleton:
Took 0.00012159999823779799 seconds and 5 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 2, 'South Australia': 1,
```

```
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 2 DFS w/ Forward and Singleton:
Took 0.0001111999990826007 seconds and 5 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 2, 'South Australia': 1,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 3 DFS w/ Forward and Singleton:
Took 0.00010969999857479706 seconds and 3 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 0, 'South Australia': 2,
'Victoria': 0, 'Western Australia': 0, 'Tasmania': 0}

Order 4 DFS w/ Forward and Singleton:
Took 0.00011729999823728576 seconds and 5 backtracks to find this solution:
{'New South Wales': 0, 'Northern Territory': 0, 'Queensland': 1, 'South Australia': 2,
'Victoria': 1, 'Western Australia': 1, 'Tasmania': 0}
```

In [59]:

```
print("Running on Four Random America Arrangements without Heuristic:")
AmericaOrder1 = []
for each in AmericaAdjacentStates:
    AmericaOrder1.append(each)
random.shuffle(AmericaOrder1)
AmericaOrder2 = []
for each in AmericaAdjacentStates:
    AmericaOrder2.append(each)
random.shuffle(AmericaOrder2)
AmericaOrder3 = []
for each in AmericaAdjacentStates:
    AmericaOrder3.append(each)
random.shuffle(AmericaOrder3)
AmericaOrder4 = []
for each in AmericaAdjacentStates:
    AmericaOrder4.append(each)
random.shuffle(AmericaOrder4)
print()
print("Order 1: \n"+str(AmericaOrder1))
print("Order 2: \n"+str(AmericaOrder2))
print("Order 3: \n"+str(AmericaOrder3))
print("Order 4: \n"+str(AmericaOrder4))


start = timeit.default_timer()
AmericaColored1, backtracks1 = ColorDepthFirst(AmericaAdjacentStates, AmericaOrder1)
end = timeit.default_timer()
time1 = end-start
print(".")
start = timeit.default_timer()
AmericaColored2, backtracks2 = ColorDepthFirst(AmericaAdjacentStates, AmericaOrder2)
end = timeit.default_timer()
time2 = end-start
print(".")
start = timeit.default_timer()
AmericaColored3, backtracks3 = ColorDepthFirst(AmericaAdjacentStates, AmericaOrder3)
end = timeit.default_timer()
time3 = end-start
print(".")
start = timeit.default_timer()
AmericaColored4, backtracks4 = ColorDepthFirst(AmericaAdjacentStates, AmericaOrder4)
end = timeit.default_timer()
time4 = end-start
```

```
print(".")
print()
print("Order 1 DFS:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AmericaColored1)
print()
print("Order 2 DFS:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AmericaColored2)
print()
print("Order 3 DFS:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AmericaColored3)
print()
print("Order 4 DFS:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AmericaColored4)
print()
avAmDFSback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAmDFStime = (time1+time2+time3+time4)/4

start = timeit.default_timer()
AmericaColored1, backtracks1 = ColorDepthFirstForward(AmericaAdjacentStates, AmericaOrd
end = timeit.default_timer()
time1 = end-start
print(".")
start = timeit.default_timer()
AmericaColored2, backtracks2 = ColorDepthFirstForward(AmericaAdjacentStates, AmericaOrd
end = timeit.default_timer()
time2 = end-start
print(".")
start = timeit.default_timer()
AmericaColored3, backtracks3 = ColorDepthFirstForward(AmericaAdjacentStates, AmericaOrd
end = timeit.default_timer()
time3 = end-start
print(".")
start = timeit.default_timer()
AmericaColored4, backtracks4 = ColorDepthFirstForward(AmericaAdjacentStates, AmericaOrd
end = timeit.default_timer()
time4 = end-start
print(".")
print()
print("Order 1 DFS w/ Forward:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AmericaColored1)
print()
print("Order 2 DFS w/ Forward:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AmericaColored2)
print()
print("Order 3 DFS w/ Forward:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AmericaColored3)
print()
print("Order 4 DFS w/ Forward:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AmericaColored4)
print()
avAmDFSFback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAmDFSFtime = (time1+time2+time3+time4)/4
```

```
start = timeit.default_timer()
AmericaColored1, backtracks1 = ColorDepthFirstForwardSingleton(AmericaAdjacentStates, A
end = timeit.default_timer()
time1 = end-start
print(".")
start = timeit.default_timer()
AmericaColored2, backtracks2 = ColorDepthFirstForwardSingleton(AmericaAdjacentStates, A
end = timeit.default_timer()
time2 = end-start
print(".")
start = timeit.default_timer()
AmericaColored3, backtracks3 = ColorDepthFirstForwardSingleton(AmericaAdjacentStates, A
end = timeit.default_timer()
time3 = end-start
print(".")
start = timeit.default_timer()
AmericaColored4, backtracks4 = ColorDepthFirstForwardSingleton(AmericaAdjacentStates, A
end = timeit.default_timer()
time4 = end-start
print(".")
print()
print("Order 1 DFS w/ Forward and Singleton:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AmericaColored1)
print()
print("Order 2 DFS w/ Forward and Singleton:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AmericaColored2)
print()
print("Order 3 DFS w/ Forward and Singleton:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AmericaColored3)
print()
print("Order 4 DFS w/ Forward and Singleton:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AmericaColored4)
print()
avAmDFSFSback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAmDFSFStime = (time1+time2+time3+time4)/4
```

```
Running on Four Random America Arrangements without Heuristic:

Order 1:
['Nevada', 'Hawaii', 'New York', 'Maine', 'Nebraska', 'West Virginia', 'South Carolina',
'New Mexico', 'Maryland', 'Wyoming', 'Connecticut', 'Washington', 'Colorado', 'Ohio', 'A
rizona', 'Vermont', 'Idaho', 'Missouri', 'Illinois', 'Tennessee', 'North Carolina', 'Kan
sas', 'California', 'Massachusetts', 'Utah', 'Texas', 'Michigan', 'Alaska', 'Oklahoma',
'Pennsylvania', 'Alabama', 'South Dakota', 'Arkansas', 'Montana', 'Georgia', 'Minnesot
a', 'Oregon', 'Virginia', 'Louisiana', 'New Hampshire', 'Wisconsin', 'Indiana', 'Delawar
e', 'Iowa', 'New Jersey', 'North Dakota', 'Mississippi', 'Kentucky', 'Florida', 'Rhode I
sland']
Order 2:
['Hawaii', 'Arizona', 'Alaska', 'Mississippi', 'Maryland', 'Tennessee', 'Idaho', 'North
Dakota', 'Wyoming', 'New Hampshire', 'Georgia', 'Louisiana', 'Arkansas', 'Indiana', 'Pen
nsylvania', 'Delaware', 'Virginia', 'South Dakota', 'Missouri', 'Nevada', 'Illinois', 'T
exas', 'North Carolina', 'Nebraska', 'Washington', 'Kentucky', 'Utah', 'Vermont', 'Oklah
oma', 'Montana', 'Iowa', 'Massachusetts', 'Wisconsin', 'Florida', 'Colorado', 'New Jerse
y', 'California', 'Connecticut', 'West Virginia', 'Maine', 'South Carolina', 'Michigan',
'Ohio', 'Minnesota', 'Rhode Island', 'Kansas', 'Alabama', 'New Mexico', 'Oregon', 'New Y
ork']
Order 3:
['Pennsylvania', 'Louisiana', 'Virginia', 'Ohio', 'Washington', 'Idaho', 'Oklahoma', 'We
```

```
st Virginia', 'New Hampshire', 'Missouri', 'Wisconsin', 'Kentucky', 'Massachusetts', 'Io
wa', 'Arkansas', 'Connecticut', 'Colorado', 'New York', 'Texas', 'Tennessee', 'Indiana',
'Michigan', 'Hawaii', 'Alabama', 'Kansas', 'Wyoming', 'Maryland', 'Alaska', 'Mississipp
i', 'North Carolina', 'Delaware', 'Nebraska', 'California', 'Nevada', 'Montana', 'Vermon
t', 'Minnesota', 'South Carolina', 'Georgia', 'Rhode Island', 'South Dakota', 'Arizona',
'Oregon', 'North Dakota', 'Maine', 'Illinois', 'Florida', 'New Jersey', 'New Mexico', 'U
tah']
Order 4:
['Georgia', 'Iowa', 'Maine', 'Alaska', 'West Virginia', 'Nevada', 'Kansas', 'Arizona',
'New Mexico', 'California', 'Arkansas', 'Colorado', 'Ohio', 'Vermont', 'Rhode Island',
'Pennsylvania', 'Utah', 'Mississippi', 'Delaware', 'Hawaii', 'Nebraska', 'Idaho', 'New Y
ork', 'Minnesota', 'New Hampshire', 'North Dakota', 'Alabama', 'Oklahoma', 'Louisiana',
'Maryland', 'Connecticut', 'Indiana', 'Illinois', 'Tennessee', 'North Carolina', 'Washin
gton', 'Texas', 'Oregon', 'South Carolina', 'Florida', 'Kentucky', 'Missouri', 'Wyomin
g', 'Wisconsin', 'Montana', 'South Dakota', 'Virginia', 'Massachusetts', 'New Jersey',
'Michigan']
.
.
.
.

Order 1 DFS:
Took 64.38911269999517 seconds and 20306062 backtracks to find this solution:
{'Alabama': 1, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 1, 'California': 2, 'Colorado': 2,
'Connecticut': 1, 'Delaware': 0, 'Florida': 0, 'Georgia': 2, 'Hawaii': 0, 'Idaho': 2, 'I
llinois': 0, 'Indiana': 2, 'Iowa': 3, 'Kansas': 1, 'Kentucky': 3, 'Louisiana': 0, 'Main
e': 0, 'Maryland': 1, 'Massachusetts': 2, 'Michigan': 3, 'Minnesota': 0, 'Mississippi':
2, 'Missouri': 2, 'Montana': 0, 'Nebraska': 0, 'Nevada': 0, 'New Hampshire': 3, 'New Jer
sey': 1, 'New Mexico': 0, 'New York': 0, 'North Carolina': 1, 'North Dakota': 1, 'Ohio':
1, 'Oklahoma': 3, 'Oregon': 1, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
0, 'South Dakota': 2, 'Tennessee': 0, 'Texas': 2, 'Utah': 3, 'Vermont': 1, 'Virginia':
2, 'Washington': 0, 'West Virginia': 0, 'Wisconsin': 1, 'Wyoming': 1}

Order 2 DFS:
Took 48.470338099999935 seconds and 15451015 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 0, 'Arkansas': 2, 'California': 2, 'Colorado': 2,
'Connecticut': 0, 'Delaware': 1, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 3, 'Indiana': 0, 'Iowa': 1, 'Kansas': 1, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 0, 'Massachusetts': 2, 'Michigan': 1, 'Minnesota': 3, 'Mississippi':
0, 'Missouri': 0, 'Montana': 3, 'Nebraska': 3, 'Nevada': 1, 'New Hampshire': 0, 'New Jer
sey': 0, 'New Mexico': 1, 'New York': 3, 'North Carolina': 2, 'North Dakota': 0, 'Ohio':
3, 'Oklahoma': 3, 'Oregon': 3, 'Pennsylvania': 2, 'Rhode Island': 1, 'South Carolina':
1, 'South Dakota': 2, 'Tennessee': 1, 'Texas': 0, 'Utah': 3, 'Vermont': 1, 'Virginia':
3, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 1}

Order 3 DFS:
Took 180.42782080000325 seconds and 81526889 backtracks to find this solution:
{'Alabama': 0, 'Alaska': 0, 'Arizona': 3, 'Arkansas': 3, 'California': 0, 'Colorado': 1,
'Connecticut': 0, 'Delaware': 2, 'Florida': 1, 'Georgia': 3, 'Hawaii': 0, 'Idaho': 1, 'I
llinois': 2, 'Indiana': 0, 'Iowa': 3, 'Kansas': 2, 'Kentucky': 3, 'Louisiana': 0, 'Main
e': 1, 'Maryland': 1, 'Massachusetts': 1, 'Michigan': 3, 'Minnesota': 2, 'Mississippi':
1, 'Missouri': 1, 'Montana': 0, 'Nebraska': 0, 'Nevada': 2, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 2, 'New York': 2, 'North Carolina': 1, 'North Dakota': 3, 'Ohio':
1, 'Oklahoma': 0, 'Oregon': 3, 'Pennsylvania': 0, 'Rhode Island': 3, 'South Carolina':
0, 'South Dakota': 1, 'Tennessee': 2, 'Texas': 1, 'Utah': 0, 'Vermont': 3, 'Virginia':
0, 'Washington': 0, 'West Virginia': 2, 'Wisconsin': 0, 'Wyoming': 2}

Order 4 DFS:
Took 19.71657379998942 seconds and 8205376 backtracks to find this solution:
{'Alabama': 3, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 0, 'California': 2, 'Colorado': 2,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 1, 'I
llinois': 2, 'Indiana': 0, 'Iowa': 0, 'Kansas': 0, 'Kentucky': 3, 'Louisiana': 2, 'Main
e': 0, 'Maryland': 3, 'Massachusetts': 3, 'Michigan': 3, 'Minnesota': 2, 'Mississippi':
1, 'Missouri': 1, 'Montana': 2, 'Nebraska': 3, 'Nevada': 0, 'New Hampshire': 1, 'New Jer
sey': 3, 'New Mexico': 0, 'New York': 1, 'North Carolina': 3, 'North Dakota': 0, 'Ohio':
```

```
1, 'Oklahoma': 3, 'Oregon': 3, 'Pennsylvania': 2, 'Rhode Island': 0, 'South Carolina':
1, 'South Dakota': 1, 'Tennessee': 2, 'Texas': 1, 'Utah': 3, 'Vermont': 0, 'Virginia':
1, 'Washington': 0, 'West Virginia': 0, 'Wisconsin': 1, 'Wyoming': 0}

.
.
.
.

Order 1 DFS w/ Forward:
Took 149.4272154000064 seconds and 2694153 backtracks to find this solution:
{'Alabama': 1, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 1, 'California': 2, 'Colorado': 2,
'Connecticut': 1, 'Delaware': 0, 'Florida': 0, 'Georgia': 2, 'Hawaii': 0, 'Idaho': 2, 'I
llinois': 0, 'Indiana': 2, 'Iowa': 3, 'Kansas': 1, 'Kentucky': 3, 'Louisiana': 0, 'Main
e': 0, 'Maryland': 1, 'Massachusetts': 2, 'Michigan': 3, 'Minnesota': 0, 'Mississippi':
2, 'Missouri': 2, 'Montana': 0, 'Nebraska': 0, 'Nevada': 0, 'New Hampshire': 3, 'New Jer
sey': 1, 'New Mexico': 0, 'New York': 0, 'North Carolina': 1, 'North Dakota': 1, 'Ohio':
1, 'Oklahoma': 3, 'Oregon': 1, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
0, 'South Dakota': 2, 'Tennessee': 0, 'Texas': 2, 'Utah': 3, 'Vermont': 1, 'Virginia':
2, 'Washington': 0, 'West Virginia': 0, 'Wisconsin': 1, 'Wyoming': 1}

Order 2 DFS w/ Forward:
Took 114.36733349999122 seconds and 1631801 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 0, 'Arkansas': 2, 'California': 2, 'Colorado': 2,
'Connecticut': 0, 'Delaware': 1, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 3, 'Indiana': 0, 'Iowa': 1, 'Kansas': 1, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 0, 'Massachusetts': 2, 'Michigan': 1, 'Minnesota': 3, 'Mississippi':
0, 'Missouri': 0, 'Montana': 3, 'Nebraska': 3, 'Nevada': 1, 'New Hampshire': 0, 'New Jer
sey': 0, 'New Mexico': 1, 'New York': 3, 'North Carolina': 2, 'North Dakota': 0, 'Ohio':
3, 'Oklahoma': 3, 'Oregon': 3, 'Pennsylvania': 2, 'Rhode Island': 1, 'South Carolina':
1, 'South Dakota': 2, 'Tennessee': 1, 'Texas': 0, 'Utah': 3, 'Vermont': 1, 'Virginia':
3, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 1}

Order 3 DFS w/ Forward:
Took 390.43853279999166 seconds and 9462360 backtracks to find this solution:
{'Alabama': 0, 'Alaska': 0, 'Arizona': 3, 'Arkansas': 3, 'California': 0, 'Colorado': 1,
'Connecticut': 0, 'Delaware': 2, 'Florida': 1, 'Georgia': 3, 'Hawaii': 0, 'Idaho': 1, 'I
llinois': 2, 'Indiana': 0, 'Iowa': 3, 'Kansas': 2, 'Kentucky': 3, 'Louisiana': 0, 'Main
e': 1, 'Maryland': 1, 'Massachusetts': 1, 'Michigan': 3, 'Minnesota': 2, 'Mississippi':
1, 'Missouri': 1, 'Montana': 0, 'Nebraska': 0, 'Nevada': 2, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 2, 'New York': 2, 'North Carolina': 1, 'North Dakota': 3, 'Ohio':
1, 'Oklahoma': 0, 'Oregon': 3, 'Pennsylvania': 0, 'Rhode Island': 3, 'South Carolina':
0, 'South Dakota': 1, 'Tennessee': 2, 'Texas': 1, 'Utah': 0, 'Vermont': 3, 'Virginia':
0, 'Washington': 0, 'West Virginia': 2, 'Wisconsin': 0, 'Wyoming': 2}

Order 4 DFS w/ Forward:
Took 43.713412000011886 seconds and 1130436 backtracks to find this solution:
{'Alabama': 3, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 0, 'California': 2, 'Colorado': 2,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 1, 'I
llinois': 2, 'Indiana': 0, 'Iowa': 0, 'Kansas': 0, 'Kentucky': 3, 'Louisiana': 2, 'Main
e': 0, 'Maryland': 3, 'Massachusetts': 3, 'Michigan': 3, 'Minnesota': 2, 'Mississippi':
1, 'Missouri': 1, 'Montana': 2, 'Nebraska': 3, 'Nevada': 0, 'New Hampshire': 1, 'New Jer
sey': 3, 'New Mexico': 0, 'New York': 1, 'North Carolina': 3, 'North Dakota': 0, 'Ohio':
1, 'Oklahoma': 3, 'Oregon': 3, 'Pennsylvania': 2, 'Rhode Island': 0, 'South Carolina':
1, 'South Dakota': 1, 'Tennessee': 2, 'Texas': 1, 'Utah': 3, 'Vermont': 0, 'Virginia':
1, 'Washington': 0, 'West Virginia': 0, 'Wisconsin': 1, 'Wyoming': 0}

.
.
.
.

Order 1 DFS w/ Forward and Singleton:
Took 3.9135104000015417 seconds and 13973 backtracks to find this solution:
{'Alabama': 1, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 1, 'California': 2, 'Colorado': 2,
```

```
'Connecticut': 1, 'Delaware': 0, 'Florida': 0, 'Georgia': 2, 'Hawaii': 0, 'Idaho': 2, 'I
llinois': 0, 'Indiana': 2, 'Iowa': 3, 'Kansas': 1, 'Kentucky': 3, 'Louisiana': 0, 'Main
e': 0, 'Maryland': 1, 'Massachusetts': 2, 'Michigan': 3, 'Minnesota': 0, 'Mississippi':
2, 'Missouri': 2, 'Montana': 0, 'Nebraska': 0, 'Nevada': 0, 'New Hampshire': 3, 'New Jer
sey': 1, 'New Mexico': 0, 'New York': 0, 'North Carolina': 1, 'North Dakota': 1, 'Ohio':
1, 'Oklahoma': 3, 'Oregon': 1, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
0, 'South Dakota': 2, 'Tennessee': 0, 'Texas': 2, 'Utah': 3, 'Vermont': 1, 'Virginia':
2, 'Washington': 0, 'West Virginia': 0, 'Wisconsin': 1, 'Wyoming': 1}

Order 2 DFS w/ Forward and Singleton:
Took 2.0616385000030277 seconds and 42377 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 0, 'Arkansas': 2, 'California': 2, 'Colorado': 2,
'Connecticut': 0, 'Delaware': 1, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 3, 'Indiana': 0, 'Iowa': 1, 'Kansas': 1, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 0, 'Massachusetts': 2, 'Michigan': 1, 'Minnesota': 3, 'Mississippi':
0, 'Missouri': 0, 'Montana': 3, 'Nebraska': 3, 'Nevada': 1, 'New Hampshire': 0, 'New Jer
sey': 0, 'New Mexico': 1, 'New York': 3, 'North Carolina': 2, 'North Dakota': 0, 'Ohio':
3, 'Oklahoma': 3, 'Oregon': 3, 'Pennsylvania': 2, 'Rhode Island': 1, 'South Carolina':
1, 'South Dakota': 2, 'Tennessee': 1, 'Texas': 0, 'Utah': 3, 'Vermont': 1, 'Virginia':
3, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 1}

Order 3 DFS w/ Forward and Singleton:
Took 0.338286700003664 seconds and 1767 backtracks to find this solution:
{'Alabama': 0, 'Alaska': 0, 'Arizona': 3, 'Arkansas': 3, 'California': 0, 'Colorado': 1,
'Connecticut': 0, 'Delaware': 2, 'Florida': 1, 'Georgia': 3, 'Hawaii': 0, 'Idaho': 1, 'I
llinois': 2, 'Indiana': 0, 'Iowa': 3, 'Kansas': 2, 'Kentucky': 3, 'Louisiana': 0, 'Main
e': 1, 'Maryland': 1, 'Massachusetts': 1, 'Michigan': 3, 'Minnesota': 2, 'Mississippi':
1, 'Missouri': 1, 'Montana': 0, 'Nebraska': 0, 'Nevada': 2, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 2, 'New York': 2, 'North Carolina': 1, 'North Dakota': 3, 'Ohio':
1, 'Oklahoma': 0, 'Oregon': 3, 'Pennsylvania': 0, 'Rhode Island': 3, 'South Carolina':
0, 'South Dakota': 1, 'Tennessee': 2, 'Texas': 1, 'Utah': 0, 'Vermont': 3, 'Virginia':
0, 'Washington': 0, 'West Virginia': 2, 'Wisconsin': 0, 'Wyoming': 2}

Order 4 DFS w/ Forward and Singleton:
Took 3.885738199998741 seconds and 42091 backtracks to find this solution:
{'Alabama': 3, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 0, 'California': 2, 'Colorado': 2,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 1, 'I
llinois': 2, 'Indiana': 0, 'Iowa': 0, 'Kansas': 0, 'Kentucky': 3, 'Louisiana': 2, 'Main
e': 0, 'Maryland': 3, 'Massachusetts': 3, 'Michigan': 3, 'Minnesota': 2, 'Mississippi':
1, 'Missouri': 1, 'Montana': 2, 'Nebraska': 3, 'Nevada': 0, 'New Hampshire': 1, 'New Jer
sey': 3, 'New Mexico': 0, 'New York': 1, 'North Carolina': 3, 'North Dakota': 0, 'Ohio':
1, 'Oklahoma': 3, 'Oregon': 3, 'Pennsylvania': 2, 'Rhode Island': 0, 'South Carolina':
1, 'South Dakota': 1, 'Tennessee': 2, 'Texas': 1, 'Utah': 3, 'Vermont': 0, 'Virginia':
1, 'Washington': 0, 'West Virginia': 0, 'Wisconsin': 1, 'Wyoming': 0}
```

In [61]:

```
print("Running on the same Four Random Australia Arrangements with Heuristic:")

print()
print("Order 1: \n"+str(AustraliaOrder1))
print("Order 2: \n"+str(AustraliaOrder2))
print("Order 3: \n"+str(AustraliaOrder3))
print("Order 4: \n"+str(AustraliaOrder4))


start = timeit.default_timer()
AustraliaColored1, backtracks1 = ColorDepthFirstH(AustraliaAdjacentStates, AustraliaOrd
end = timeit.default_timer()
time1 = end-start

start = timeit.default_timer()
AustraliaColored2, backtracks2 = ColorDepthFirstH(AustraliaAdjacentStates, AustraliaOrd
end = timeit.default_timer()
```

```
time2 = end-start

start = timeit.default_timer()
AustraliaColored3, backtracks3 = ColorDepthFirstH(AustraliaAdjacentStates, AustraliaOrd
end = timeit.default_timer()
time3 = end-start

start = timeit.default_timer()
AustraliaColored4, backtracks4 = ColorDepthFirstH(AustraliaAdjacentStates, AustraliaOrd
end = timeit.default_timer()
time4 = end-start

print()
print("Order 1 DFS w/ Heuristics:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AustraliaColored1)
print()
print("Order 2 DFS w/ Heuristics:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AustraliaColored2)
print()
print("Order 3 DFS w/ Heuristics:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AustraliaColored3)
print()
print("Order 4 DFS w/ Heuristics:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AustraliaColored4)
print()
avAusDFSHback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAusDFSHtime = (time1+time2+time3+time4)/4

start = timeit.default_timer()
AustraliaColored1, backtracks1 = ColorDepthFirstForwardH(AustraliaAdjacentStates, Austr
end = timeit.default_timer()
time1 = end-start

start = timeit.default_timer()
AustraliaColored2, backtracks2 = ColorDepthFirstForwardH(AustraliaAdjacentStates, Austr
end = timeit.default_timer()
time2 = end-start

start = timeit.default_timer()
AustraliaColored3, backtracks3 = ColorDepthFirstForwardH(AustraliaAdjacentStates, Austr
end = timeit.default_timer()
time3 = end-start

start = timeit.default_timer()
AustraliaColored4, backtracks4 = ColorDepthFirstForwardH(AustraliaAdjacentStates, Austr
end = timeit.default_timer()
time4 = end-start

print()
print("Order 1 DFS w/ Forward and Heuristics:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AustraliaColored1)
print()
print("Order 2 DFS w/ Forward and Heuristics:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AustraliaColored2)
print()
```

```
print("Order 3 DFS w/ Forward and Heuristics:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AustraliaColored3)
print()
print("Order 4 DFS w/ Forward and Heuristics:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AustraliaColored4)
print()
avAusDFSFHback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAusDFSFHtime = (time1+time2+time3+time4)/4

start = timeit.default_timer()
AustraliaColored1, backtracks1 = ColorDepthFirstForwardSingletonH(AustraliaAdjacentStat
end = timeit.default_timer()
time1 = end-start

start = timeit.default_timer()
AustraliaColored2, backtracks2 = ColorDepthFirstForwardSingletonH(AustraliaAdjacentStat
end = timeit.default_timer()
time2 = end-start

start = timeit.default_timer()
AustraliaColored3, backtracks3 = ColorDepthFirstForwardSingletonH(AustraliaAdjacentStat
end = timeit.default_timer()
time3 = end-start

start = timeit.default_timer()
AustraliaColored4, backtracks4 = ColorDepthFirstForwardSingletonH(AustraliaAdjacentStat
end = timeit.default_timer()
time4 = end-start

print()
print("Order 1 DFS w/ Forward and Singleton and Heuristics:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AustraliaColored1)
print()
print("Order 2 DFS w/ Forward and Singleton and Heuristics:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AustraliaColored2)
print()
print("Order 3 DFS w/ Forward and Singleton and Heuristics:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AustraliaColored3)
print()
print("Order 4 DFS w/ Forward and Singleton and Heuristics:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AustraliaColored4)
print()
avAusDFSFSHback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAusDFSFSHtime = (time1+time2+time3+time4)/4
```

```
Running on the same Four Random Australia Arrangements with Heuristic:

Order 1:
['New South Wales', 'Tasmania', 'Northern Territory', 'South Australia', 'Queensland',
'Western Australia', 'Victoria']
Order 2:
['Northern Territory', 'Tasmania', 'South Australia', 'Western Australia', 'Queensland',
'Victoria', 'New South Wales']
Order 3:
['Queensland', 'Western Australia', 'Northern Territory', 'Victoria', 'New South Wales',
```

```
'South Australia', 'Tasmania']
Order 4:
['Northern Territory', 'Tasmania', 'Queensland', 'New South Wales', 'Victoria', 'South A
ustralia', 'Western Australia']

Order 1 DFS w/ Heuristics:
Took 0.00017559999832883477 seconds and 17 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 2 DFS w/ Heuristics:
Took 0.0001576000067871064 seconds and 17 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 3 DFS w/ Heuristics:
Took 0.0001512000017100945 seconds and 17 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 4 DFS w/ Heuristics:
Took 0.00016589999722782522 seconds and 17 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}


Order 1 DFS w/ Forward and Heuristics:
Took 0.0002821000089170411 seconds and 0 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 2 DFS w/ Forward and Heuristics:
Took 0.00020619999850168824 seconds and 0 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 3 DFS w/ Forward and Heuristics:
Took 0.00015320000238716602 seconds and 0 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 4 DFS w/ Forward and Heuristics:
Took 0.00015100000018719584 seconds and 0 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}


Order 1 DFS w/ Forward and Singleton and Heuristics:
Took 0.00021120000747032464 seconds and 3 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 2 DFS w/ Forward and Singleton and Heuristics:
Took 0.0001812999980757013 seconds and 3 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 3 DFS w/ Forward and Singleton and Heuristics:
Took 0.00017410000145900995 seconds and 3 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}

Order 4 DFS w/ Forward and Singleton and Heuristics:
Took 0.00029490000451914966 seconds and 3 backtracks to find this solution:
{'New South Wales': 1, 'Northern Territory': 1, 'Queensland': 2, 'South Australia': 0,
```

```
'Victoria': 2, 'Western Australia': 2, 'Tasmania': 0}
```

In [62]:

```
print("Running on the same Four Random America Arrangements with Heuristic:")

print()
print("Order 1: \n"+str(AmericaOrder1))
print("Order 2: \n"+str(AmericaOrder2))
print("Order 3: \n"+str(AmericaOrder3))
print("Order 4: \n"+str(AmericaOrder4))


start = timeit.default_timer()
AmericaColored1, backtracks1 = ColorDepthFirstH(AmericaAdjacentStates, AmericaOrder1)
end = timeit.default_timer()
time1 = end-start
print(".")
start = timeit.default_timer()
AmericaColored2, backtracks2 = ColorDepthFirstH(AmericaAdjacentStates, AmericaOrder2)
end = timeit.default_timer()
time2 = end-start
print(".")
start = timeit.default_timer()
AmericaColored3, backtracks3 = ColorDepthFirstH(AmericaAdjacentStates, AmericaOrder3)
end = timeit.default_timer()
time3 = end-start
print(".")
start = timeit.default_timer()
AmericaColored4, backtracks4 = ColorDepthFirstH(AmericaAdjacentStates, AmericaOrder4)
end = timeit.default_timer()
time4 = end-start
print(".")
print()
print("Order 1 DFS w/ Heuristic:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AmericaColored1)
print()
print("Order 2 DFS w/ Heuristic:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AmericaColored2)
print()
print("Order 3 DFS w/ Heuristic:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AmericaColored3)
print()
print("Order 4 DFS w/ Heuristic:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AmericaColored4)
print()
avAmDFSHback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAmDFSHtime = (time1+time2+time3+time4)/4

start = timeit.default_timer()
AmericaColored1, backtracks1 = ColorDepthFirstForwardH(AmericaAdjacentStates, AmericaOr
end = timeit.default_timer()
time1 = end-start
print(".")
start = timeit.default_timer()
AmericaColored2, backtracks2 = ColorDepthFirstForwardH(AmericaAdjacentStates, AmericaOr
end = timeit.default_timer()
```

```
time2 = end-start
print(".")
start = timeit.default_timer()
AmericaColored3, backtracks3 = ColorDepthFirstForwardH(AmericaAdjacentStates, AmericaOr
end = timeit.default_timer()
time3 = end-start
print(".")
start = timeit.default_timer()
AmericaColored4, backtracks4 = ColorDepthFirstForwardH(AmericaAdjacentStates, AmericaOr
end = timeit.default_timer()
time4 = end-start
print(".")
print()
print("Order 1 DFS w/ Forward and Heuristics:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AmericaColored1)
print()
print("Order 2 DFS w/ Forward and Heuristics:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AmericaColored2)
print()
print("Order 3 DFS w/ Forward and Heuristics:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AmericaColored3)
print()
print("Order 4 DFS w/ Forward and Heuristics:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AmericaColored4)
print()
avAmDFSFHback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAmDFSFHtime = (time1+time2+time3+time4)/4

start = timeit.default_timer()
AmericaColored1, backtracks1 = ColorDepthFirstForwardSingletonH(AmericaAdjacentStates,
end = timeit.default_timer()
time1 = end-start
print(".")
start = timeit.default_timer()
AmericaColored2, backtracks2 = ColorDepthFirstForwardSingletonH(AmericaAdjacentStates,
end = timeit.default_timer()
time2 = end-start
print(".")
start = timeit.default_timer()
AmericaColored3, backtracks3 = ColorDepthFirstForwardSingletonH(AmericaAdjacentStates,
end = timeit.default_timer()
time3 = end-start
print(".")
start = timeit.default_timer()
AmericaColored4, backtracks4 = ColorDepthFirstForwardSingletonH(AmericaAdjacentStates,
end = timeit.default_timer()
time4 = end-start
print(".")
print()
print("Order 1 DFS w/ Forward and Singleton and Heuristics:")
print("Took "+str(time1)+" seconds and "+str(backtracks1)+" backtracks to find this sol
print(AmericaColored1)
print()
print("Order 2 DFS w/ Forward and Singleton and Heuristics:")
print("Took "+str(time2)+" seconds and "+str(backtracks2)+" backtracks to find this sol
print(AmericaColored2)
print()
```

```
print("Order 3 DFS w/ Forward and Singleton and Heuristics:")
print("Took "+str(time3)+" seconds and "+str(backtracks3)+" backtracks to find this sol
print(AmericaColored3)
print()
print("Order 4 DFS w/ Forward and Singleton and Heuristics:")
print("Took "+str(time4)+" seconds and "+str(backtracks4)+" backtracks to find this sol
print(AmericaColored4)
print()
avAmDFSFSHback = (backtracks1+backtracks2+backtracks3+backtracks4)/4
avAmDFSFSHtime = (time1+time2+time3+time4)/4
```

```
Running on the same Four Random America Arrangements with Heuristic:

Order 1:
['Nevada', 'Hawaii', 'New York', 'Maine', 'Nebraska', 'West Virginia', 'South Carolina',
'New Mexico', 'Maryland', 'Wyoming', 'Connecticut', 'Washington', 'Colorado', 'Ohio', 'A
rizona', 'Vermont', 'Idaho', 'Missouri', 'Illinois', 'Tennessee', 'North Carolina', 'Kan
sas', 'California', 'Massachusetts', 'Utah', 'Texas', 'Michigan', 'Alaska', 'Oklahoma',
'Pennsylvania', 'Alabama', 'South Dakota', 'Arkansas', 'Montana', 'Georgia', 'Minnesot
a', 'Oregon', 'Virginia', 'Louisiana', 'New Hampshire', 'Wisconsin', 'Indiana', 'Delawar
e', 'Iowa', 'New Jersey', 'North Dakota', 'Mississippi', 'Kentucky', 'Florida', 'Rhode I
sland']
Order 2:
['Hawaii', 'Arizona', 'Alaska', 'Mississippi', 'Maryland', 'Tennessee', 'Idaho', 'North
Dakota', 'Wyoming', 'New Hampshire', 'Georgia', 'Louisiana', 'Arkansas', 'Indiana', 'Pen
nsylvania', 'Delaware', 'Virginia', 'South Dakota', 'Missouri', 'Nevada', 'Illinois', 'T
exas', 'North Carolina', 'Nebraska', 'Washington', 'Kentucky', 'Utah', 'Vermont', 'Oklah
oma', 'Montana', 'Iowa', 'Massachusetts', 'Wisconsin', 'Florida', 'Colorado', 'New Jerse
y', 'California', 'Connecticut', 'West Virginia', 'Maine', 'South Carolina', 'Michigan',
'Ohio', 'Minnesota', 'Rhode Island', 'Kansas', 'Alabama', 'New Mexico', 'Oregon', 'New Y
ork']
Order 3:
['Pennsylvania', 'Louisiana', 'Virginia', 'Ohio', 'Washington', 'Idaho', 'Oklahoma', 'We
st Virginia', 'New Hampshire', 'Missouri', 'Wisconsin', 'Kentucky', 'Massachusetts', 'Io
wa', 'Arkansas', 'Connecticut', 'Colorado', 'New York', 'Texas', 'Tennessee', 'Indiana',
'Michigan', 'Hawaii', 'Alabama', 'Kansas', 'Wyoming', 'Maryland', 'Alaska', 'Mississipp
i', 'North Carolina', 'Delaware', 'Nebraska', 'California', 'Nevada', 'Montana', 'Vermon
t', 'Minnesota', 'South Carolina', 'Georgia', 'Rhode Island', 'South Dakota', 'Arizona',
'Oregon', 'North Dakota', 'Maine', 'Illinois', 'Florida', 'New Jersey', 'New Mexico', 'U
tah']
Order 4:
['Georgia', 'Iowa', 'Maine', 'Alaska', 'West Virginia', 'Nevada', 'Kansas', 'Arizona',
'New Mexico', 'California', 'Arkansas', 'Colorado', 'Ohio', 'Vermont', 'Rhode Island',
'Pennsylvania', 'Utah', 'Mississippi', 'Delaware', 'Hawaii', 'Nebraska', 'Idaho', 'New Y
ork', 'Minnesota', 'New Hampshire', 'North Dakota', 'Alabama', 'Oklahoma', 'Louisiana',
'Maryland', 'Connecticut', 'Indiana', 'Illinois', 'Tennessee', 'North Carolina', 'Washin
gton', 'Texas', 'Oregon', 'South Carolina', 'Florida', 'Kentucky', 'Missouri', 'Wyomin
g', 'Wisconsin', 'Montana', 'South Dakota', 'Virginia', 'Massachusetts', 'New Jersey',
'Michigan']
.
.
.
.

Order 1 DFS w/ Heuristic:
Took 278.83919030000106 seconds and 35973000 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 1, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 0, 'Main
e': 1, 'Maryland': 0, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
3, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 2, 'New Hampshire': 0, 'New Jer
sey': 3, 'New Mexico': 2, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 1, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 3, 'Utah': 3, 'Vermont': 2, 'Virginia':
```

```
3, 'Washington': 2, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 2 DFS w/ Heuristic:
Took 274.0997788000095 seconds and 35973000 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 1, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 0, 'Main
e': 1, 'Maryland': 0, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
3, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 2, 'New Hampshire': 0, 'New Jer
sey': 3, 'New Mexico': 2, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 1, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 3, 'Utah': 3, 'Vermont': 2, 'Virginia':
3, 'Washington': 2, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 3 DFS w/ Heuristic:
Took 288.4851125999994 seconds and 35973000 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 1, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 0, 'Main
e': 1, 'Maryland': 0, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
3, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 2, 'New Hampshire': 0, 'New Jer
sey': 3, 'New Mexico': 2, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 1, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 3, 'Utah': 3, 'Vermont': 2, 'Virginia':
3, 'Washington': 2, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 4 DFS w/ Heuristic:
Took 268.4815316000022 seconds and 35973000 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 1, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 1, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 0, 'Main
e': 1, 'Maryland': 0, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
3, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 2, 'New Hampshire': 0, 'New Jer
sey': 3, 'New Mexico': 2, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 1, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 3, 'Utah': 3, 'Vermont': 2, 'Virginia':
3, 'Washington': 2, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

.
.
.
.

Order 1 DFS w/ Forward and Heuristics:
Took 0.01161359999969136 seconds and 0 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 2, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 3, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
0, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 3, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 3, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 2, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 0, 'Utah': 1, 'Vermont': 2, 'Virginia':
0, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 2 DFS w/ Forward and Heuristics:
Took 0.011343999998643994 seconds and 0 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 2, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 3, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
0, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 3, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 3, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 2, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 0, 'Utah': 1, 'Vermont': 2, 'Virginia':
```

```
0, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 3 DFS w/ Forward and Heuristics:
Took 0.011236700011068024 seconds and 0 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 2, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 3, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
0, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 3, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 3, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 2, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 0, 'Utah': 1, 'Vermont': 2, 'Virginia':
0, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 4 DFS w/ Forward and Heuristics:
Took 0.013557199999922886 seconds and 0 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 2, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 3, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
0, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 3, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 3, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 2, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 0, 'Utah': 1, 'Vermont': 2, 'Virginia':
0, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

.
.
.
.

Order 1 DFS w/ Forward and Singleton and Heuristics:
Took 0.011942999990424141 seconds and 9 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 2, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 3, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
0, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 3, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 3, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 2, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 0, 'Utah': 1, 'Vermont': 2, 'Virginia':
0, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 2 DFS w/ Forward and Singleton and Heuristics:
Took 0.010549800004810095 seconds and 9 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 2, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 3, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
0, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 3, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 3, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 2, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 0, 'Utah': 1, 'Vermont': 2, 'Virginia':
0, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 3 DFS w/ Forward and Singleton and Heuristics:
Took 0.010254400011035614 seconds and 9 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 2, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 3, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
0, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 3, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 3, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 2, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 0, 'Utah': 1, 'Vermont': 2, 'Virginia':
```

```
0, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}

Order 4 DFS w/ Forward and Singleton and Heuristics:
Took 0.01413670000329148 seconds and 9 backtracks to find this solution:
{'Alabama': 2, 'Alaska': 0, 'Arizona': 2, 'Arkansas': 2, 'California': 0, 'Colorado': 0,
'Connecticut': 2, 'Delaware': 0, 'Florida': 1, 'Georgia': 0, 'Hawaii': 0, 'Idaho': 0, 'I
llinois': 1, 'Indiana': 3, 'Iowa': 2, 'Kansas': 2, 'Kentucky': 2, 'Louisiana': 1, 'Main
e': 1, 'Maryland': 3, 'Massachusetts': 1, 'Michigan': 2, 'Minnesota': 1, 'Mississippi':
0, 'Missouri': 0, 'Montana': 1, 'Nebraska': 1, 'Nevada': 3, 'New Hampshire': 0, 'New Jer
sey': 1, 'New Mexico': 3, 'New York': 0, 'North Carolina': 2, 'North Dakota': 2, 'Ohio':
0, 'Oklahoma': 1, 'Oregon': 2, 'Pennsylvania': 2, 'Rhode Island': 3, 'South Carolina':
1, 'South Dakota': 0, 'Tennessee': 1, 'Texas': 0, 'Utah': 1, 'Vermont': 2, 'Virginia':
0, 'Washington': 1, 'West Virginia': 1, 'Wisconsin': 0, 'Wyoming': 2}
```

In [70]:

```
print()
print("Final Results: ")
print("---------------------------")
print("DFS:")
print("\tAustralia:")
print("\t\tAverage Backtracks: "+str(avAusDFSback))
print("\t\tAverage Time: "+str(avAusDFStime)+" seconds")
print("\tUnited States:")
print("\t\tAverage Backtracks: "+str(avAmDFSback))
print("\t\tAverage Time: "+str(avAmDFStime)+" seconds")
print()
print("DFS w/ Forward Checking:")
print("\tAustralia:")
print("\t\tAverage Backtracks: "+str(avAusDFSFback))
print("\t\tAverage Time: "+str(avAusDFSFtime)+" seconds")
print("\tUnited States:")
print("\t\tAverage Backtracks: "+str(avAmDFSFback))
print("\t\tAverage Time: "+str(avAmDFSFtime)+" seconds")
print()
print("DFS w/ Forward Checking and Singletons:")
print("\tAustralia:")
print("\t\tAverage Backtracks: "+str(avAusDFSFSback))
print("\t\tAverage Time: "+str(avAusDFSFStime)+" seconds")
print("\tUnited States:")
print("\t\tAverage Backtracks: "+str(avAmDFSFSback))
print("\t\tAverage Time: "+str(avAmDFSFStime)+" seconds")
print()
print("DFS w/ Heuristics :")
print("\tAustralia:")
print("\t\tAverage Backtracks: "+str(avAusDFSHback))
print("\t\tAverage Time: "+str(avAusDFSHtime)+" seconds")
print("\tUnited States:")
print("\t\tAverage Backtracks: "+str(avAmDFSHback))
print("\t\tAverage Time: "+str(avAmDFSHtime)+" seconds")
print()
print("DFS w/ Forward Checking and Heuristics:")
print("\tAustralia:")
print("\t\tAverage Backtracks: "+str(avAusDFSFHback))
print("\t\tAverage Time: "+str(avAusDFSFHtime)+" seconds")
print("\tUnited States:")
print("\t\tAverage Backtracks: "+str(avAmDFSFHback))
print("\t\tAverage Time: "+str(avAmDFSFHtime)+" seconds")
print()
print("DFS w/ Forward Checking and Singletons and Heuristics:")
print("\tAustralia:")
print("\t\tAverage Backtracks: "+str(avAusDFSFSHback))
```

```
print("\t\tAverage Time: "+str(avAusDFSFSHtime)+" seconds")
print("\tUnited States:")
print("\t\tAverage Backtracks: "+str(avAmDFSFSHback))
print("\t\tAverage Time: "+str(avAmDFSFSHtime)+" seconds")
```

```
Final Results:
---------------------------
DFS:
        Australia:
                Average Backtracks: 24.25
                Average Time: 8.419999994657701e-05 seconds
        United States:
                Average Backtracks: 31372335.5
                Average Time: 78.25096134999694 seconds

DFS w/ Forward Checking:
        Australia:
                Average Backtracks: 6.75
                Average Time: 0.00021312500030035153 seconds
        United States:
                Average Backtracks: 3729687.5
                Average Time: 174.4866234250003 seconds

DFS w/ Forward Checking and Singletons:
        Australia:
                Average Backtracks: 4.5
                Average Time: 0.00011494999853312038 seconds
        United States:
                Average Backtracks: 25052.0
                Average Time: 2.5497934500017436 seconds

DFS w/ Heuristics :
        Australia:
                Average Backtracks: 17.0
                Average Time: 0.00016257500101346523 seconds
        United States:
                Average Backtracks: 35973000.0
                Average Time: 277.47640332500305 seconds

DFS w/ Forward Checking and Heuristics:
        Australia:
                Average Backtracks: 0.0
                Average Time: 0.00019812500249827280 seconds
        United States:
                Average Backtracks: 0.0
                Average Time: 0.011937875002331566 seconds

DFS w/ Forward Checking and Singletons and Heuristics:
        Australia:
                Average Backtracks: 3.0
                Average Time: 0.00021537500288104638 seconds
        United States:
                Average Backtracks: 9.0
                Average Time: 0.011720975002390333 seconds
```

In [ ]:

In [ ]: