AI Project 2: CSP

How to run

Type Down `python3 main.py` in the terminal then input your filename like so

```
> python3 main.py
Enter the filename: input1.txt
```

Outputs

Output1.txt

```
NSW = G
NT = G
Q = B
SA = R
WA = B
V = B
```

Output2.txt

```
R1 = R
R2 = B
R3 = Y
R4 = G
R5 = B
R6 = Y
R7 = R
R8 = G
```

Code

```
# Ethan Philpott
# Project 2
# AI
# Dec 2021

import copy

# Reads lines from filename (takes file_name)
def read_file(file_name):
    lines = []
    with open(file_name) as f:
        for line in f.readlines():
            # Remove newline character and then split line based on spaces
            line = line.strip()
            line = line.split(' ')
            # If blank line then we skip
            if line == ['']:
                continue
            # Add in line
            lines.append(line)
    return lines

# Stores data in a dictionary based on 2d list inputted (takes lines)
def store_data(lines):
    # Converts strings to ints in constraints 2d array
    for i in range(3, len(lines)):
        for j in range(0, len(lines[i])):
```

```
            lines[i][j] = int(lines[i][j])

    # Stores data in dictionary and returns it
    return {
        'domains': lines[1],
        'assignments': [copy.deepcopy(lines[2]) for i in range(0, int(lines[0][0]))],
        'constraints': lines[3:]
    }

# Calculate MRV (takes assignments)
def mrv(assignments):
    # Holds the max MRV value found
    max = len(assignments[0])
    # Holds the index of the max MRV values found
    max_items = [0]
    # Search for max items
    for i in range(1, len(assignments)):
        item = assignments[i]
        if len(item) > max:
            max = len(item)
            max_items = [i]
        elif len(item) == max:
            max_items.append(i)
    return max_items

# Calculate degree (takes assignments, constraints, variables)
def degree(assignments, constraints, variables):
    # Holds the max degree and list of the indexes of the max degree items
    max = 0
    max_items = []
    # Go through variables
    for var in variables:
        degree = 0
        # Go through constraints
        row = constraints[var]
        for i in range(0, len(row)):
            # If we have a 1 and the item it relates to isn't assigned then we increment degree
            if row[i] == 1 and len(assignments[i]) != 1:
                degree += 1
        # Update max if necessary and append
        if degree > max:
            max = degree
            max_items = [var]
        elif degree == max:
            max_items.append(var)

    return max_items

# Checks if we can assign a given color (takes assignments, constraints, var, domain)
def checkNeighbors(assignments, constraints, var, domain):
    # Go through constraints
    row = constraints[var]
    for i in range(0, len(row)):
        if row[i] == 1 and len(assignments[i]) == 1:
            if assignments[i][0] == domain:
                return False
    return True

# Calculate inference (takes assignments, constraints, var, domain)
def inference(assignments, constraints, var, domain):
    # Go through constraints
    row = constraints[var]
    for i in range(0, len(row)):
        if row[i] == 1 and domain in assignments[i]:
            assignments[i].remove(domain)
            if len(assignments[i]) == 0:
```

```
            if len(assignments[i]) == 0:
                return None
    return assignments

# Calculate backtrack (takes constraints and assignments)
def backtrack(constraints, assignments):
    # Check if we are done
    done = True
    for elem in assignments:
        if len(elem) > 1:
            done = False
        elif len(elem) == 0:
            # We have an empty assignment so we return failure
            return None
    if done:
        return assignments

    # Get variables
    variables = mrv(assignments)
    variables = degree(assignments, constraints, variables)

    # If we have no variables then we failed
    if variables == []:
        return None

    # Select variable
    variable = variables[0]

    for domain in assignments[variable]:
        # Check if we can assign the domain to the variable
        if checkNeighbors(assignments, constraints, variable, domain):
            # Keep a copy of old domain in case we fail
            old_assignments = copy.deepcopy(assignments)
            # Assign domain to variable
            assignments[variable] = [domain]
            inferences = inference(assignments, constraints, variable, domain)
            if inferences != None:
                # Recurse
                assignments = inferences
                assignments = backtrack(constraints, copy.deepcopy(assignments))
                if assignments != None:
                    return assignments
            # If we failed then we unassign everything (might not need to do this technically since we are deep copying)
            assignments = old_assignments
    # Return failure
    return None

# Displays results (takes domains and result)
def display_results(domains, result):
    # Print results
    f = open("Output.txt", "w")
    if(result == None):
        f.write("No solution exists")
        print("No solution exists")
    else:
        for i in range(0, len(result)):
            text = domains[i] + " = " + result[i][0]
            print(text)
            if (i < len(result) - 1):
                f.write(text + "\n")
            else:
                f.write(text)
    f.close()

# Main function
def main():
```

```
    # Get filename
    file_name = str(input('Enter the filename: ')).strip()

    # Get data
    lines = read_file(file_name)
    data = store_data(lines)

    result = backtrack(data['constraints'], data['assignments'])
    display_results(data['domains'], result)

if __name__ == "__main__":
    main()
```