COMPUTER SCIENCE AND DATA ANALYTICS

Course: **Artificial Intelligence**

Assignment 2

CSP Graph Coloring

Student: **Shikhaliyev Anar**

Instructor:

Amrinder Arora

Ziyan Guo

Baku 2023

1. Problem Description

In this project, to address this graph coloring problem, we should create a CSP algorithm. The progrom should have a search algorithm to solve the CSP as well as heuristics such as MRC (minimum remaining values) and LCV (least restricting value) for supporting the search, and finally constraint propagation through AC3.

2. Code analysis

I have written an algorithm in **python** to approach the Graph Coloring problem using the Constraint Satisfaction Problem (CSP) approach. The problem involves assigning colors to vertices of a given graph such that no two adjacent vertices have the same color. The program reads an input file that contains the graph details and the number of colors available for coloring the vertices. The program outputs the assignment of colors to each vertex or a message indicating that it is not possible to color the graph. The code is structured into several functions that work together to solve the problem.

- **2.1 processFile()**
  is the driver function that takes the input file and solves the graph coloring problem using the GraphColoringCSP class. If a valid solution is found, it prints the color assigned to each vertex, otherwise it prints that it is not possible to color the graph.

- **2.2 takeInputs(fName)**
  takes a file name as an argument and reads a graph from a file and returns it as a dictionary. This function is used to read the input data from a file which also can contain comments with the help of '#' sign.

- **2.3 GraphColoringCSP Classs**

GraphColoringCSP's constructor accepts two inputs, graph and num colors, where graph is a dictionary defining the graph and num colors is an integer reflecting the maximum number of colors that is used to color the graph. The constructor sets up a few instance variables, including the domain dictionary, which translates each vertex in the graph to a collection of color values.

The class consist of several functions: (**Each of this functions have unit-tests)**

1. checkValid()
   The graph initialization process calls this function. To check whether the graph is valid, the constructor invokes this function. A ValueError is reported and additional calculation is avoided if a node is linked to itself or is incorrectly related to its neighbors. Here, you can see that an entry like **(1,1)** in one of the inputs causes this mistake.

```
PS C:\Users\Anar\Desktop\GWU-AI\assignmen2> python .\main.py .\sampleIncorrectInput.txt
Traceback (most recent call last):
  File "C:\Users\Anar\Desktop\GWU-AI\assignmen2\main.py", line 180, in <module>
    processFile()
  File "C:\Users\Anar\Desktop\GWU-AI\assignmen2\main.py", line 172, in processFile
    csp = GraphColoringCSP(graph, num_colors)
  File "C:\Users\Anar\Desktop\GWU-AI\assignmen2\main.py", line 13, in __init__
    self.checkValid()
  File "C:\Users\Anar\Desktop\GWU-AI\assignmen2\main.py", line 18, in checkValid
    raise ValueError(f"Node {node} is linked to itself.")
ValueError: Node 1 is linked to itself.
```

2. MRVgetUnassignedArea()
   This function returns the most constrained unassigned area in the graph, based on the Minimum Remaining Values (MRV) heuristic. It takes a color_map as input, which is a dictionary mapping nodes to their assigned colors (if any).

3. LCVgetOrderedDomainValues()
   This function returns the domain values for a given node, ordered by the number of conflicts they produce with the colors of neighboring nodes. It takes a node and a color_map as input.

4. countConflicts(node, color, color map)
   This function returns the number of conflicts that occur when the given node is assigned the given color, based on the current color_map. It is used by the LCVgetOrderedDomainValues() function.

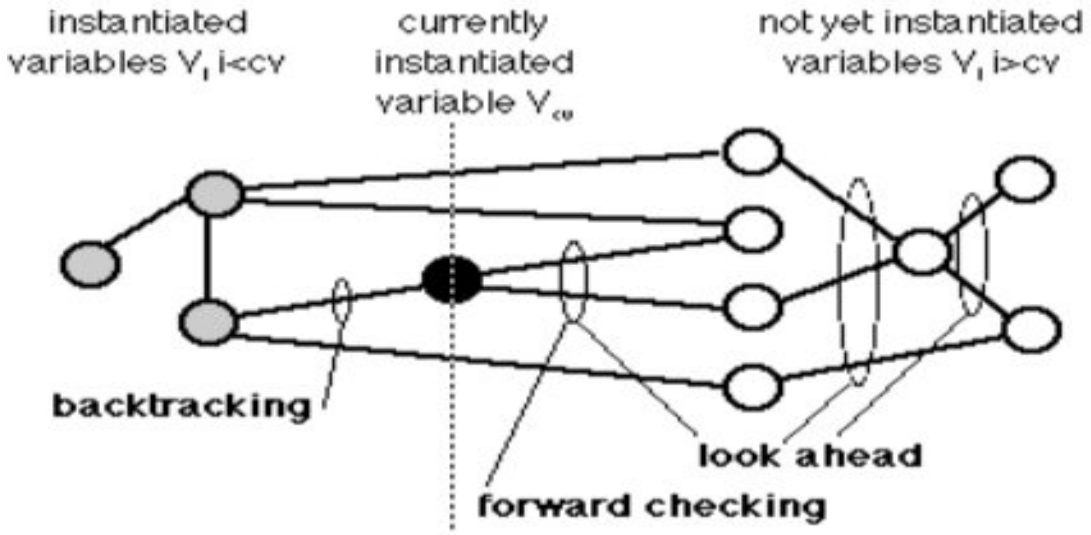
```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (Xi, Xj) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue

---------------------------------------------------------------------

function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[Xi] do
    if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
      then delete x from DOMAIN[Xi]; removed ← true
  return removed
```

5. AC_3()
   The AC-3 arc consistency method is implemented by this function. It uses queue of tuples representing arcs between nodes. All potential arcs are initially added to the queue if no queue is supplied. When arc consistency is attained, the function returns True, otherwise, it returns False.

6. removeInconsistentValues()
   This function removes inconsistent values from the domain of node i, based on the constraints imposed by node j. It returns True if any values are removed, and False otherwise. It is used by the AC_3() function.

```
===> Before: 1: {0, 1, 2}, 3: {0, 1, 2}
===> After:  1: {1, 2}, 3: {0, 1, 2}
```

7. isValidColor()
   This method determines if applying the specified color to the specified vertex will conflict with any limitations imposed by the current color map. The backtrack() method uses it to determine if a given color is appropriate for a certain node.



8. backtrack()
   The backtrack method implements the backtracking algorithm, which **recursively** searches for a solution by assigning a value to a variable, checking if it is consistent with the constraints, and then moving to the next variable until all variables are assigned or a contradiction is detected. The method uses the MRV and LCV heuristics to choose the variable and value to assign, and it also performs forward checking to propagate the constraints.

9. forwardChecking()
   The forwardChecking method propagates constraints by removing values from the domains of the unassigned variables that create conflicts with the newly assigned value.

10. undoForwardChecking()
    The undoForwardChecking method undoes the effects of forward checking when a partial assignment is undone during backtracking.

11. solve()
    Finally, this function first applies the AC-3 algorithm to enforce arc consistency on the graph, then uses backtracking search to find a valid solution to the graph coloring problem.

3. Unit tests

In order to make sure the propgram works, I have written a Python unittest module that tests my heuristics and other functions, implemented in the main module.

In order to make sure the propgram works, I have written a Python unittest module that tests, backtrack&forward checking seperately in different condition and outcomes, and the other class is testing rest of the functins of the main module.

The test file consists of two test classes TestBackTrackAlgorithm and GraphColoringCSPTestCase which test backtrack & forward checking seperately in different condition and outcomes, as well as the other other functins respectively. The tests are implemented using **Python's built-in unittest module**.

The GraphColoringCSPTestCase class tests correctness of various functions:

- test_graph_validity – raises error if graph is invalid. This test case intentially left as failing case
- test_MRVgetUnassignedArea - checks whether the MRVgetUnassignedArea method returns the correct node with the minimum remaining values (MRV) that has not been assigned a color yet.
- test_LCVgetOrderedDomainValues - checks whether the LCVgetOrderedDomainValues method returns the domain values for a given node ordered by the least-constraining value (LCV) heuristic.
- test_countConflicts - checks whether the countConflicts method returns the correct number of conflicts for a given node and color assignment.
- test_removeInconsistentValues
- test_isValidColor
- test_AC3

The TestBackTrackAlgorithm provides a test scenraiosfor the backtracking algorithm, explaining its working and advantages. It also suggests some testing strategies for evaluating the algorithm's performance and correctness, such as generating test cases with known solutions and comparing the output of the algorithm with the expected results. It also tests forward checking and undoForwardChecking.

4. Source Code

https://github.com/anar-sixeliyev/assignmen2

5. Sample execution

CODE:

```
PS C:\Users\Anar\Desktop\GWU-AI\assignmen2> python .\main.py .\sample01.txt
Vertex 1 is assigned color 0
Vertex 2 is assigned color 1
Vertex 3 is assigned color 2
Vertex 4 is assigned color 2
Vertex 5 is assigned color 1
Vertex 6 is assigned color 0
Vertex 7 is assigned color 3
PS C:\Users\Anar\Desktop\GWU-AI\assignmen2> python .\main.py .\sample04.txt
Vertex 0 is assigned color 0
Vertex 1 is assigned color 1
Vertex 2 is assigned color 2
Vertex 3 is assigned color 1
Vertex 4 is assigned color 2
Vertex 5 is assigned color 1
Vertex 6 is assigned color 2
Vertex 7 is assigned color 2
Vertex 8 is assigned color 0
Vertex 9 is assigned color 0
Vertex 10 is assigned color 3
Vertex 11 is assigned color 0
Vertex 12 is assigned color 1
Vertex 13 is assigned color 3
Vertex 14 is assigned color 2
Vertex 15 is assigned color 0
Vertex 16 is assigned color 3
Vertex 17 is assigned color 1
Vertex 18 is assigned color 3
Vertex 19 is assigned color 2
Vertex 20 is assigned color 0
Vertex 21 is assigned color 1
Vertex 22 is assigned color 3
PS C:\Users\Anar\Desktop\GWU-AI\assignmen2> python .\main.py .\sample05.txt
        It is not possible to color this Graph
PS C:\Users\Anar\Desktop\GWU-AI\assignmen2>
```

UNIT TESTS:

```
PS C:\Users\Anar\Desktop\GWU-AI\assignmen2> python .\test-csp.py
Usage: python main.py <filename>
...........
----------------------------------------------------------------------
Ran 11 tests in 0.002s

OK
```