Assignment 4 - Coding component

● Graded

Student

Tyler NGUYEN

Total Points

100 / 100 pts

Autograder Score

50.0 / 50.0

Passed Tests

1.1) Test case I (10/10)
1.2) Test case II (10/10)
1.3) Test case III (10/10)
1.4) Test case IV (10/10)
1.5) Test case V (10/10)

Question 2

Manual Grading **50** / 50 pts

✔ **– 0 pts** Correct

– 10 pts Not sufficient document/Not clean code.

– 30 pts Not implementing the algorithm in class.

– 50 pts Assignment is submitted after the allowed number of late days

– 50 pts Not runnable code.

Autograder Results

1.1) Test case I (10/10)

1.2) Test case II (10/10)

1.3) Test case III (10/10)

1.4) Test case IV (10/10)

1.5) Test case V (10/10)

Submitted Files

colourSAT.py Download

```python
#CPSC 413 Assignment 4 question 4
#Author: Tyler Nguyen
#UCID: 30158563
from pysat.solvers import Solver

class c_sat:

 def solve(self, graph):
  solver = Solver()
  num_of_nodes = len(graph) #number of nodes in the graph input that was given

  def get_node_colour_var(node, col): #helper function that is used to create unique variables for each possible colour of each node. that will be used in the SAT Problem
      return (node-1)*3 + col #it takes a node index and (col)our index which then it will
      #return a unique variable index, -1 adjusts the node number and *3 will create space
      #for three colour variables per node, +col will correct the final value to characterize between
      #the variables for different colours for the same node.

  #iterate through each node in the graph input in order to add clasuses to the SAT solver to
  #ensure that each node is coloured with exactly a colour which was the one of the constraints talked about in lecture.
  for i in range(1, num_of_nodes+1):
      solver.add_clause([get_node_colour_var(i, c) for c in range(1, 4)]) #each node should will have at least one colour
      for first_colour in range(1, 4):
          for second_colour in range(first_colour+1, 4):
              solver.add_clause([-get_node_colour_var(i, first_colour), -get_node_colour_var(i, second_colour)]) #each node cannot have two colours at the same time

  #iterate through the graph input to add clauses that will ensure that no two adjacent nodes are colour the same which the other constraint talked about in lecture.
  for i in range(num_of_nodes):
      for j in range(num_of_nodes):
          if graph[i][j] == 1: #represents an edge between node i and j, therefore need to
              #add clauses to the SAT solver that will not allow nodes i and j to be coloured with the same colour
              for c in range(1, 4):
                  solver.add_clause([-get_node_colour_var(i+1, c), -get_node_colour_var(j+1, c)]) #negating the odds that both nodes having the same colour

  return solver.solve()
```