NAME	**SURAJ VASANTRAO WARBHE**
ROLL NO.	**331066**
GR NO.	**21910631**
SUBJECT	**ARTIFICIAL INTELLIGENCE**

ASSIGNMENT NO. 5

AIM: Assignment on Constraint Satisfaction Problem and Implement graph coloring problem

THEORY:

CONSTRAINTS SATISFACTION PROBLEM:

Constraint Satisfaction Problems (CSPs) are mathematical questions defined as a set of objects whose state must satisfy a number of constraints. A constraint satisfaction problem (CSP) consists of-

1. Variables: It will be a tuple with the variable names.
2. Domains: It will be a dictionary with the variable names as keys, and the domains as values (in the form of any iterable you want).
3. Constraints will be a list of tuples with two components each: a tuple with the variables involved in the constraint, and a reference to a function that checks the constraint.

BACKTRACKING ALGORITHM:

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

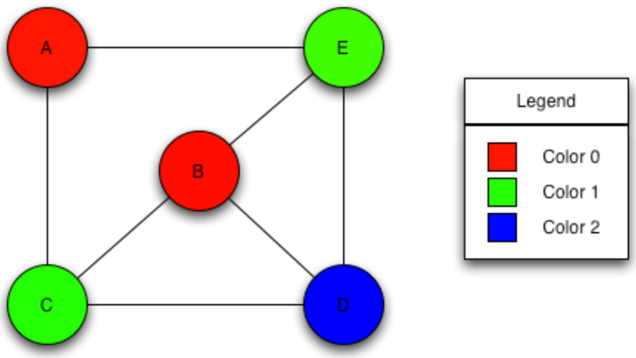
BACKTRACKING APPROACH:

The idea is to assign colors one by one to different vertices, starting from the vertex 0. Before assigning a color, check for safety by considering already assigned colors to the adjacent vertices i.e check if the adjacent vertices have the same color or not. If there is any color assignment that does not violate the conditions, mark the color assignment as part of the solution. If no assignment of color is possible then backtrack and return false.

ALGORITHM:

1. Create a recursive function that takes the graph, current index, number of vertices, and output color array.
2. If the current index is equal to the number of vertices. Print the color configuration in the output array.
3. Assign a color to a vertex (1 to m).
4. For every assigned color, check if the configuration is safe, (i.e. check if the adjacent vertices do not have the same color) recursively call the function with next index and number of vertices
5. If any recursive function returns true break the loop and return true.
6. If no recursive function returns true then return false.

SOURCE CODE:



```
# GRAPH COLORING- CONSTRAINTS SATISFACTION PROBLEM

# Vertices of Graph
nodes = {
    0: 'A',
    1: 'B',
    2: 'C',
    3: 'D',
    4: 'E',
}

# Graph connections
graph = {
    0: [2, 4],
    1: [2, 3, 4],
    2: [0, 1, 3],
```

```python
    3: [1, 2, 3],
    4: [0, 1, 3],
}

# Colors to be assigned
colors = ["Red", "Green", "Blue"]

# Function which checks whether color assign to vertex
def check(node, colour):
    for i in graph[node]:
        # if vertex is present in col_graph and already colored
        # then make condition false
        if i in col_graph and col_graph[i] == colour:
            return False
    return True


# Function assign color to uncolor vertex
def assign(node, colour):
    col_graph[node] = colour      # assign color


n = 0       # stores number of vertices
i = 0       # store number of colors
col_graph={}

while n < 5:
    assigned = 0            # Initialize all color default values as 0 (False
    for i in range(3):      # 3 == Three colors
        # checking validity
        if check(n, i) == True:
            # Start coloring
            assign(n, i)
            n += 1
            assigned = 1    # Mark as assigned with color
            break


    if assigned == 0:
        prevas = 0
        for x in range(3):
            if check(n-1, (col_graph[n-1]+1)%3) == True:   # Backtracking
                assign(n,(col_graph[n-1]+1)%3)
                prevas = 1
                break
        if prevas == 0:
            n -= 1


# Print Solution
for key, value in col_graph.items():
    print(nodes[key] + " : " + colors[value])
```

OUTPUT:

```
A : Red
B : Red
C : Green
D : Blue
E : Green
```

CONCLUSION:

1. From this assignment, we learnt the concept of Constraints Satisfaction Problem and Backtracking algorithm in Artificial Intelligence.
2. Also, implement Graph Coloring problem of Graph using Backtracking algorithm.