Name: Niraj Milind Amrutkar

Roll No. 332002

PRN: 22010910

Batch B1

Assignment No. 5

Constraint Satisfaction Problem

Aim:

Assignment on Constraint Satisfaction Problem: Implement graph colouring problem. OR Implementation of Constraint Satisfaction Problem for solving Crypt-arithmetic Problems. Solve any one of the follows: Backtracking: Constraint Satisfaction Problem –

1) Seating Arrangement or Computer CPU utilization.

2) Analyse data from twitter or news to categorize them useful reading list out of the whole clutter.

3) Read data from images using OCR (optical character recognition) and OMR (optical mark recognition) techniques – use open-source libs.

Objective:

To solve n-Queen problem: The local condition is that no two queens attack each other, i.e., are on the same row, or column, or diagonal.

Theory:

To represent squares on the chess board, we'll assign each an integer row and an integer column. We can ensure each of the eight queens isn't on the same column by assigning them sequentially the columns 1 through 8. The variables in our constraint-satisfaction problem is the column of the queen in question. The domains can be the possible rows (again 1 through 8). In the following code listing we jump to the end of our file where we define these variables and domains.

To represent squares on the chess board, we'll assign each an integer row and an integer column. We can ensure each of the eight queens isn't on the same column by assigning them sequentially the columns 1 through 8. The variables in our constraint-satisfaction problem is the column of the queen in question. The domains can be the possible rows (again 1 through 8). In the

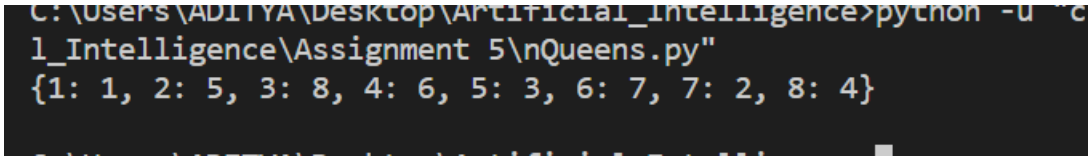
following code listing we jump to the end of our file where we define these variables and domains.

To solve the problem, we need a constraint that checks whether any two queens are on the same row or diagonal (they were all assigned different sequential columns to begin with). Checking for the same row is trivial, but checking for the same diagonal requires a little bit of math. If any two queens are on the same diagonal, the difference between their rows is the same as the difference between their columns. Note that we're now heading back to the top of our source file to enter the following code.

Code:

```
from cspFile import AbstractConstraint, CSP
from typing import Dict, List, Optional

class QueensConstraint(AbstractConstraint[int, int]):
    def __init__(self, columns: List[int]) -> None:
        super().__init__(columns)
        self.columns: List[int] = columns

    def satisfied(self, assignment: Dict[int, int]) -> bool:
        for q1c, q1r in assignment.items(): # q1c = queen 1 column, q1r =
queen 1 row
            for q2c in range(q1c + 1, len(self.columns) + 1): # q2c = queen 2
column
                if q2c in assignment:
                    q2r: int = assignment[q2c] # q2r = queen 2 row
                    if q1r == q2r: # same row?
                        return False
                    if abs(q1r - q2r) == abs(q1c - q2c): # same diagonal?
                        return False
        return True # no conflict

columns: List[int] = [1, 2, 3, 4, 5, 6, 7, 8]
rows: Dict[int, List[int]] = {}
for column in columns:
    rows[column] = [1, 2, 3, 4, 5, 6, 7, 8]
csp: CSP[int, int] = CSP(columns, rows)

csp.add_constraint(QueensConstraint(columns))
solution: Optional[Dict[int, int]] = csp.backtracking_search()
if solution is None:
    print("No solution found!")
else:
    print(solution)
```

Output:



Conclusion:

Here we solved n-queen problem using Constraint Satisfaction Problem.