Artificial Intelligence Lab Report
Lab2: CONSTRAINT PROGRAMMING

Submitted By: SUYOG DHAKAL (075BCT092)

Theory:

Constraint Programming:

Constraint programming is a developing software technology and technique for problem solving that deals with reasoning and computing. Constraint programming is based on the idea that computational problems can be explained in terms of limits imposed on a group of potential solutions. It works by integrating those restrictions into the programming environment.

Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is a problem that requires its solution within some limitations or conditions also known as constraints. It consists of the following:

1. A finite set of variables which stores the solution (V = {V1, V2, V3,....., Vn})
2. A set of discrete values known as domain from which the solution is picked (D = {D1, D2, D3,.....,Dn})
3. A finite set of constraints (C = {C1, C2, C3,....., Cn})

The following problems are some of the popular problems that can be solved using CSP:

1. Cryptarithmetic (Coding alphabets to numbers.) .
2. N-Queen (In an n-queen problem, N queens should be placed in an N*N matrix such that no queen shares the same row, column or diagonal.) .
3. Map Coloring (coloring different regions of map, ensuring no adjacent regions have the same color).
4. Crossword (everyday puzzles appearing in newspapers) Sudoku (a number grid) .
5. Latin Square Problem.

Cryptoarithmetic problem

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct.
We can perform all the arithmetic operations on a given cryptarithmetic problem.

The rules or constraints on a cryptoarithmetic problem are as follows:

1. There should be a unique digit to be replaced with a unique alphabet.
2. The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.
3. There should be only one carry forward, while performing the addition operation on a problem.
4. Digits should be from 0-9 only.
5. The problem can be solved from both sides, I.e. LHS or RHS

Eight Queen Problem

Eight queens problem is a constraint satisfaction problem. In this puzzle, we are given an 8 X 8 chessboard. The task is to place 8 queens on the board, such that no two queens attack each other. The queens attack each other only if they lie in same row, same column, or same diagonal. The problem can be formulated as P = {(x1, y1), (x2, y2) ……(x8, y8)} where (x1, y1) gives the position of the first queen and so on.

So it can be clearly seen that the domains for xi and yi are

Dx = {1, 2, 3, 4, 5, 6, 7, 8} and

Dy = {1, 2, 3, 4, 5, 6, 7, 8} respectively.

The constraints are

1. No two queens should be in the same row.

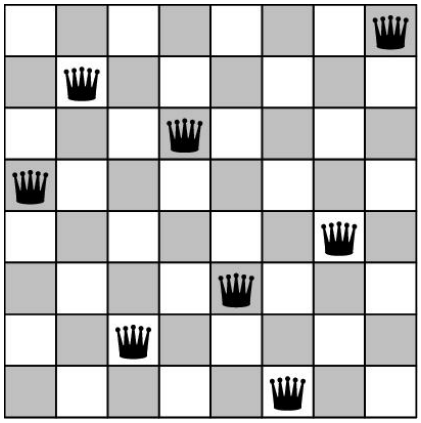
   That is, $y_i \neq y_j$ for i = 1 to 8; j = 1 to 8; $i \neq j$

2. No two queens should be in the same columns.

   That is, $x_i \neq x_j$ for i = 1 to 8; j = 1 to 8; $i \neq j$

3. There should not be two queens placed on the same diagonal line.

   That is, $(y_i - y_j) \neq \pm (x_i - x_j)$.



Assignments

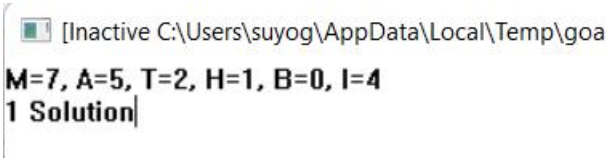
1.

```
%  MATH
% +MATH
%-------
% HABIT

DOMAINS
int_list=integer*

PREDICATES
solution(int_list)
member(integer,int_list)

CLAUSES
        solution([]).
        solution([M,A,T,H,B,I]):-
        C4=1,
        member(C1,[0,1]),
        member(C2,[0,1]),
        member(C3,[0,1]),

% C1, C2, C3 will have values 0 or 1

member(M,[0,1,2,3,4,5,6,7,8,9]),
member(A,[0,1,2,3,4,5,6,7,8,9]),
member(T,[0,1,2,3,4,5,6,7,8,9]),
member(H,[0,1,2,3,4,5,6,7,8,9]),
member(B,[0,1,2,3,4,5,6,7,8,9]),
member(I,[0,1,2,3,4,5,6,7,8,9]),

% M, A, T, H, B, I will have values between 0 and 9.
% The values of M, A, T, H, B, I must not be equal.

M<>A, M<>T, M<>H, M<>B, M<>I,
A<>T, A<>H, A<>B, A<>I,
T<>H, T<>B, T<>I,
H<>B, M<>I,
B<>I,

% Computation for solution
H + H = T + 10*C1,
T + T + C1 = I + 10*C2,
A + A + C2 = B + 10*C3,
```

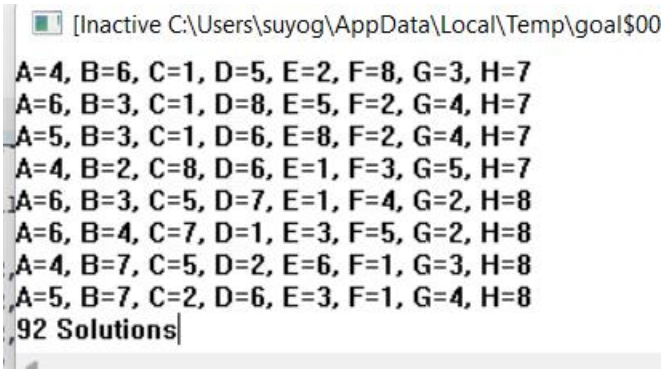
M + M + C3 = A + 10*C4,
H = C4.

member(X, [X|_]).
member(X, [_|Z]):-
member(X,Z).

GOAL
solution([M,A,T,H,B,I]).

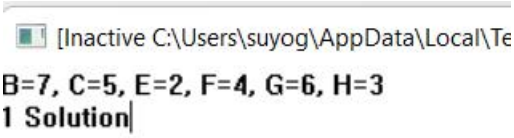


2.



Discussion:
The given program calculates all the ways in which 8 queen scan be places on a chess board so that the queens does not attack each other. The result obtained above shows that there are 92 different solutions to the problem. These solutions were obtained from the program.

Task: Finding the solution to the above problem few queens are fixed explicitly.
For case: solution([c(1,1),c(2,B),c(3,C),c(4,8),c(5,E),c(6,F),c(7,G),c(8,H)]).
The result is:



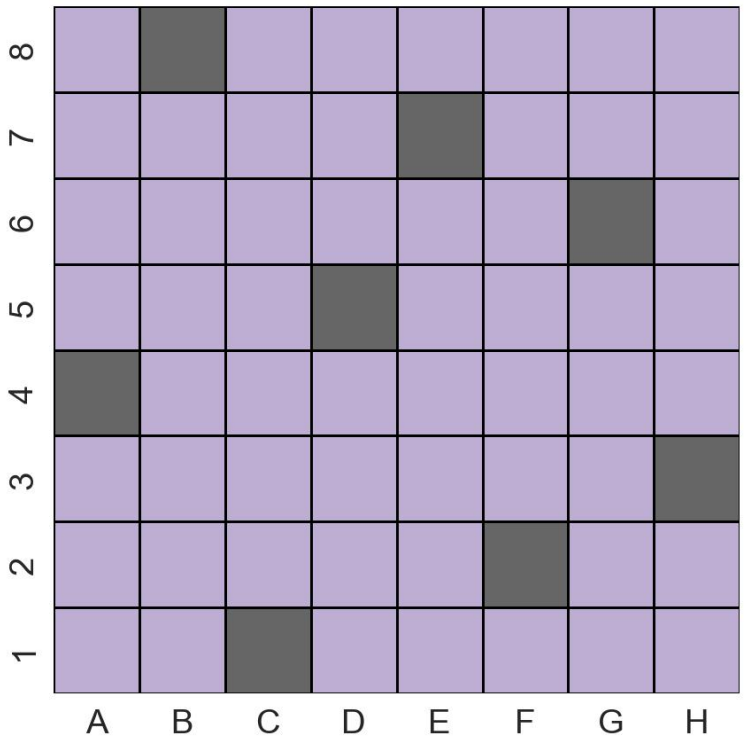
It is seen that when the position (1,1) and position (4,8) were fixed, then the queens could be arranged in only one way so that they do not attack each other.

3.
python code

```
import numpy as np
import copy
import seaborn as sns
import matplotlib.pyplot as plt
import string

def possible(grid, x, y):
    l = len(grid)
    for i in range(l):
        if grid[x][i] == 1:
            return False
    for i in range(l):
        if grid[i][y] == 1:
            return False
    for i in range(l):
        for j in range(l):
            if grid[i][j] == 1:
                if (abs(i-x) == abs(j-y)):
                    return False
    return True
def solve(grid):
    l = len(grid)
    for x in range(l):
        for y in range(l):
            if grid[x][y] == 0:
                if (possible(grid, x, y)):
                    grid[x][y] = 1
                    solve(grid)
                    if (sum(sum(a) for a in grid) == 8):
                        return grid
                    grid[x][y] = 0
    return grid
def plot(grid):
    l = len(grid)
    Ly = list(range(1, l+1))[::-1]
    ly = [str(i) for i in Ly]
    Lx = list(string.ascii_uppercase)
    lx = Lx[:8]
    plt.close('all')
    sns.set(font_scale=2)
    plt.figure(figsize=(10, 10))
    ax = plt.gca()
    ax.set_aspect(1)
    sns.heatmap(Solution, linewidths=.8, cbar=False, linecolor='black',
                cmap='Accent', center=0.4, xticklabels=lx, yticklabels=ly)
    plt.show()
N = 8
grid = np.zeros([N, N], dtype=int)
grid[3][3] = 1
grid = grid.tolist()
print('Solving {} by {} Chess Board\n'.format(N, N))
Solution = solve(copy.deepcopy(grid))
print(np.matrix(Solution))
plot(grid)
```

Output

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CO

[Running] python -u "d:\6th sem\AI\
Solving 8 by 8 Chess Board

[[0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 1 0 0 0 0]
 [1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1]
 [0 0 0 0 0 1 0 0]
 [0 0 1 0 0 0 0 0]]
```



4.

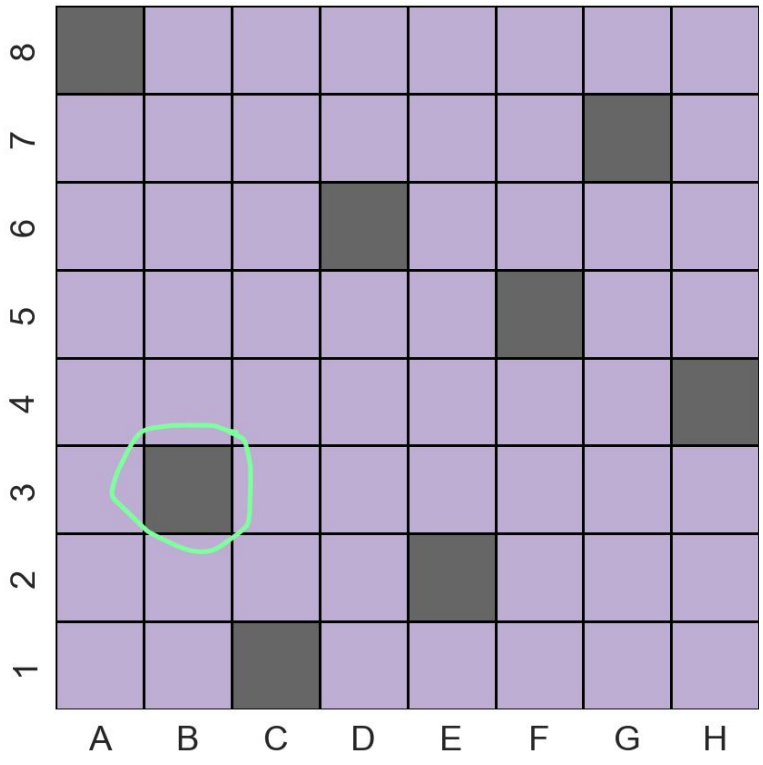
Python code

```
from re import X
import numpy as np
import copy
import seaborn as sns
import matplotlib.pyplot as plt
import string

def possible(grid,y,x):
  l=len(grid)
  for i in range(l):
    if grid[y][i]==1:
      return False
  for i in range(l):
    if grid[i][x]==1:
      return False
  for i in range(l):
    for j in range(l):
      if grid[i][j]==1:
        if abs(i - y) == abs(j - x):
          return False
  return True

def solve(grid):
  l=len(grid)
  for y in range(l):
    for x in range(l):
      if grid[y][x]==0:
        if possible(grid,y,x):
          grid[y][x]=1
          solve(grid)
          if sum(sum(a) for a in grid)==l:
            return grid
          grid[y][x]=0
  return grid

def plot(grid):
  l=len(grid)
  Ly=list(range(1,l+1))[::-1]
  ly = [str(i) for i in Ly]
  Lx=list(string.ascii_uppercase)
  lx=Lx[:l]
  plt.close('all')
  sns.set(font_scale = 2)
  plt.figure(figsize=(10,10))
  ax = plt.gca()
  ax.set_aspect(1)

  sns.heatmap(Solution,linewidths=.8,cbar=False,linecolor='black',cmap='Accent',center=0.4,xti
cklabels=lx,yticklabels=ly)
  plt.show()

position = input("enter queen's position (eg.A3) : ")
vertical = 0-(int(position[1])-8)
horizontal = ord(position[0].upper())-65
if(horizontal<0 or horizontal>8 or vertical<0 or vertical>8):
  print("ERROR, input position is out of bounds")
  exit(1)
```

```
N=8
grid=np.zeros([N,N],dtype=int)
grid[vertical][horizontal]=1
grid=grid.tolist()

print('Solving {} by {} Chess Board\n'.format(N,N))
Solution = solve(copy.deepcopy(grid))
print(np.matrix(Solution))
plot(grid)
```

Output

```
PS D:\6th sem\AI\Lab\lab2>  & 'C:\Users
ythonFiles\lib\python\debugpy\launcher'
enter queen's position (eg.A3) : b3
Solving 8 by 8 Chess Board

[[1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1]
 [0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 1 0 0 0 0 0]]
```



Discussion

In Lab2 we learned to solve various constraint satisfaction problems. In the beginning we learned about cryptoarithmetic problems using Visual Prolog. Then we solved 8 Queens problem without any queen's position being fixed as well as fixing some queens in certain position. We solved in both Visual prolog and in python.

Conclusion

Hence, we wrote program and output was analyzed and report was made accordingly.