



TRIBHUVAN UNIVERSITY  
INSTITUTE OF ENGINEERING  
Pulchowk Campus

Lab No: **2**

A lab report on:

**Constraint Programming**

**Submitted By:**

Bibek Basyal  
075BCT097

**Submitted To:**

Department of Electronics  
and Computer Engineering

January 25, 2022

## What is Constraint Programming ?

---

*Constraint Programming (CP)* is a programming paradigm for solving combinatorial problems that draws on a wide range of techniques from artificial intelligence, computer science, and operations research. In constraint programming, users declaratively state the constraints on the feasible solutions for a set of decision variables. Constraints differ from the common primitives of imperative programming languages in that they do not specify a step or sequence of steps to execute, but rather the need to specify a method to solve these constraints. This typically draws upon standard methods like chronological backtracking and constraint propagation, but may use customized code like a problem specific branching heuristic.

Constraint programming takes its root from and can be expressed in the form of constraint logic programming, which embeds constraints into a logic program. This variant of logic programming is due to a specific class of constraints that were introduced in Prolog.

## Applications of Constraint Programming

---

The conceptual approach of the constraint programming are the key components of many intelligent systems. Therefore, *CP* is used in different application fields. Some of the fields are listed as:

- Artificial Intelligence
- Computer Network and Security
- Air Traffic Control
- Operations research
- Software Engineering

## Cryptarithmic Problem

---

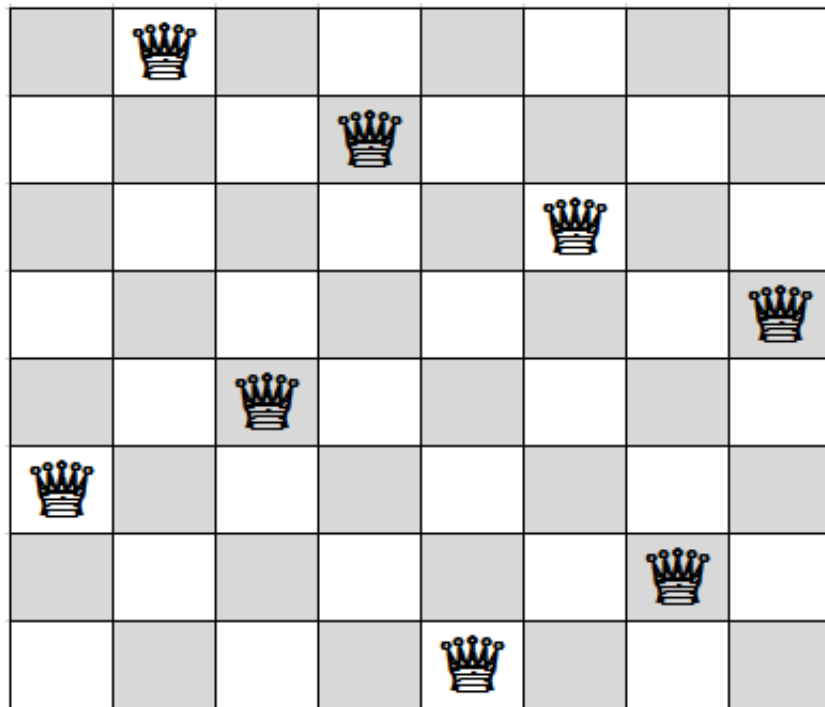
*Cryptarithmic Problem* is a type of constraint satisfaction problem where the solution is to find digits that is the unique replacement of the alphabets or other symbols in the problem statement. In cryptarithmic problem, the digits (0 - 9) get substituted by some possible alphabets or symbols remaining within the below mentioned rules:

- A unique digit should be substituted for an unique alphabet.
- Digits should only be used within the range 0 - 9.
- Carry overs in problems of two number addition is always 1.
- The result should satisfy the predefined arithmetic rules.

## Eight Queen Problem

---

The eight queens puzzle is based on the constraint satisfaction problem of arranging eight chess queens on an 8x8 chessboard so that none of them can capture each other. The color of the queens has no significance on this problem, and any queen is expected to be capable of striking any other. As a result, in order for a solution to be achieved, no two queens must share the same row, column, or diagonal. More generally, the n queens problem places n queens on an n\*n chessboard.



## Crypto Arithmetic Problems Solved

---

### 1. SEND + MORE = MONEY

$$\begin{array}{r} \phantom{0000} C_4 \phantom{00} C_3 \phantom{00} C_2 \phantom{00} C_1 \\ \phantom{0000} S \phantom{00} E \phantom{00} N \phantom{00} D \\ \phantom{0000} M \phantom{00} O \phantom{00} R \phantom{00} E \\ \hline M \phantom{00} O \phantom{00} N \phantom{00} E \phantom{00} Y \end{array}$$

The domain for alphabets is given by:

$$S, E, N, D, M, O, R, Y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The constraints are:

$$D + E = Y + 10 * C_1$$

$$N + R + C_1 = E + 10 * C_2$$

$$E + O + C_2 = N + 10 * C_3$$

$$S + M + C_3 = O + 10 * C_4$$

$$M = C_4$$

$$C_1, C_2, C_3, C_4 \in \{0, 1\}$$

And we have the constraint that no two alphabets should be assigned to the same number.

```
DOMAINS
int_list=integer*

PREDICATES
solution(int_list)
member(integer,int_list)

CLAUSES
solution([]).      %1
solution([S,E,N,D,M,O,R,Y]):-
C4=1,
member(C1,[0,1]),
member(C2,[0,1]),
member(C3,[0,1]),
```

```

% C1, C2, C3 will have values 0 or 1
member(E,[0,1,2,3,4,5,6,7,8,9]),
member(N,[0,1,2,3,4,5,6,7,8,9]),
member(D,[0,1,2,3,4,5,6,7,8,9]),
member(M,[0,1,2,3,4,5,6,7,8,9]),
member(O,[0,1,2,3,4,5,6,7,8,9]),
member(R,[0,1,2,3,4,5,6,7,8,9]),
member(Y,[0,1,2,3,4,5,6,7,8,9]),
member(S,[0,1,2,3,4,5,6,7,8,9]),

% S,E,N, D, M, O, R, Y will have values between 0 and 9.
% The values of S,E,N, D, M, O, R, Y must not be equal.
S<>E, S<>N, S<>D, S<>M, S<>O, S<>R, S<>Y,
E<>N, E<>D, E<>M, E<>O, E<>R, E<>Y,
N<>D, N<>M, N<>O, N<>R, N<>Y,
D<>M, D<>O, D<>R, D<>Y,
M<>O, M<>R, M<>Y,
O<>R, O<>Y,
R<>Y,

% Computation for solution
D+E=Y+10*C1,
N+R+C1=E+10*C2,
E+O+C2=N+10*C3,
S+M+C3=O+10*C4,
M=C4.
member(X, [X|_]).
member(X, [_|Z]):-
member(X,Z).

GOAL
solution([S,E,N,D,M,O,R,Y]).

```

- **Output:**

```

[Inactive C:\Users\bibek\AppData\Local\Temp\goal$000.exe]
S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
1 Solution|

```

## 2. TEN + TEN + FORTY = SIXTY

$$\begin{array}{r}
 \begin{array}{cccc}
 C_4 & C_3 & C_2 & C_1 \\
 F & O & R & T & Y \\
 & T & E & N & \\
 & T & E & N & \\
 \hline
 S & I & X & T & Y
 \end{array}
 \end{array}$$

The domain for alphabets is given by:

$$T, E, N, F, O, R, Y, S, I, X \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The constraints are:

$$N + N + Y = Y + 10 * C_1$$

$$E + E + T + C_1 = T + 10 * C_2$$

$$T + T + R + C_2 = X + 10 * C_3$$

$$O + C_3 = I + 10 * C_4$$

$$F + C_4 = S$$

$$C_1, C_2, C_3, C_4 \in \{0, 1, 2\}$$

And we have the constraint that no two alphabets should be assigned to the same number.

```

DOMAINS
int_list=integer*

PREDICATES
solution(int_list)
member(integer,int_list)

CLAUSES
solution([]).
solution([T,E,N,F,O,R,Y,S,I,X]):-
C4=1,
member(C1,[0,1,2]),
member(C2,[0,1,2]),
member(C3,[0,1,2]),

```

```

member(T,[0,1,2,3,4,5,6,7,8,9]),
member(E,[0,1,2,3,4,5,6,7,8,9]),
member(N,[0,1,2,3,4,5,6,7,8,9]),
member(F,[0,1,2,3,4,5,6,7,8,9]),
member(O,[0,1,2,3,4,5,6,7,8,9]),
member(R,[0,1,2,3,4,5,6,7,8,9]),
member(Y,[0,1,2,3,4,5,6,7,8,9]),
member(S,[0,1,2,3,4,5,6,7,8,9]),
member(I,[0,1,2,3,4,5,6,7,8,9]),
member(X,[0,1,2,3,4,5,6,7,8,9]),

T<>E,T<>N,T<>F,T<>O,T<>R,T<>Y,T<>S,T<>I,T<>X,
E<>N,E<>F,E<>O,E<>R,E<>Y,E<>S,E<>I,E<>X,
N<>F,N<>O,N<>R,N<>Y,N<>S,N<>I,N<>X,
F<>O,F<>R,F<>Y,F<>S,F<>I,F<>X,
O<>R,O<>Y,O<>S,O<>I,O<>X,
R<>Y,R<>S,R<>I,R<>X,
Y<>S,Y<>I,Y<>X,
S<>I,S<>X,
I<>X,

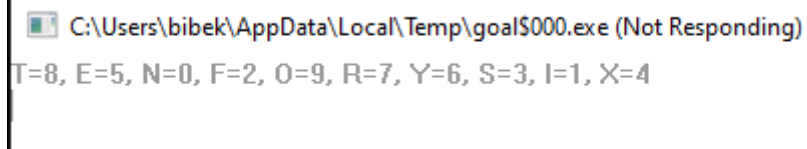
N+N+Y=Y+10*C1,
C1+E+E+T=T+10*C2,
C2+T+T+R=X+10*C3,
C3+O=I+10*C4,
C4+F=S.

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

GOAL
solution([T,E,N,F,O,R,Y,S,I,X]).

```

### • Output:



T=8, E=5, N=0, F=2, O=9, R=7, Y=6, S=3, I=1, X=4

### 3. VERY + GOOD = CODER

$$\begin{array}{r}
 \begin{array}{cccc}
 C_4 & C_3 & C_2 & C_1 \\
 V & E & R & Y \\
 G & O & O & D \\
 \hline
 C & O & D & E & R
 \end{array}
 \end{array}$$

The domain for alphabets is given by:

$$V, E, R, Y, G, O, D, C \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The constraints are:

$$Y + D = R + 10 * C_1$$

$$R + O + C_1 = E + 10 * C_2$$

$$E + O + C_2 = D + 10 * C_3$$

$$V + G + C_3 = O + 10 * C_4$$

$$C = C_4$$

$$C_1, C_2, C_3, C_4 \in \{0, 1, \}$$

And we have the constraint that no two alphabets should be assigned to the same number.

```

DOMAINS
int_list=integer*

PREDICATES
solution(int_list)
member(integer,int_list)

CLAUSES
solution([]).    %1
solution([V, E, R, Y, G, O, D, C]):-
C4=1,
member(C1,[0,1]),
member(C2,[0,1]),
member(C3,[0,1]),

% C1, C2, C3 will have values 0 or 1
member(V,[0,1,2,3,4,5,6,7,8,9]),

```



```

member(E,[0,1,2,3,4,5,6,7,8,9]),
member(R,[0,1,2,3,4,5,6,7,8,9]),
member(Y,[0,1,2,3,4,5,6,7,8,9]),
member(G,[0,1,2,3,4,5,6,7,8,9]),
member(O,[0,1,2,3,4,5,6,7,8,9]),
member(D,[0,1,2,3,4,5,6,7,8,9]),
member(C,[0,1,2,3,4,5,6,7,8,9]),

% V, E, R, Y, G, O, D, C will have values between 0 and 9.
% The values of V, E, R, Y, G, O, D, C must not be equal.
V<>E, V<>R, V<>Y, V<>G, V<>O, V<>D, V<>C,
E<>R, E<>Y, E<>G, E<>O, E<>D, E<>C,
R<>Y, R<>G, R<>O, R<>D, R<>C,
Y<>G, Y<>O, Y<>D, Y<>C,
G<>O, G<>D, G<>C,
O<>D, O<>C,
D<>C,

% Computation for solution
Y+D=R+10*C1,
R+O+C1=E+10*C2,
E+O+C2=D+10*C3,
V+G+C3=O+10*C4,
C=C4.
member(X, [X|_]).
member(X, [_|Z]):-
member(X,Z).

GOAL
solution([V, E, R, Y, G, O, D, C]).

```

- Output:

```

[Inactive C:\Users\bibek\AppData\Local\Temp\goal$000.exe]
V=5, E=9, R=6, Y=4, G=7, O=3, D=2, C=1
V=6, E=9, R=5, Y=2, G=7, O=4, D=3, C=1
V=7, E=9, R=5, Y=2, G=6, O=4, D=3, C=1
V=7, E=9, R=6, Y=4, G=5, O=3, D=2, C=1
4 Solutions|

```

## 8 Queen Program Solved

---

The 8 Queen program can be formulated with variables  $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$  and  $y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8$ . Here,  $x_i$  and  $y_i$  represent the row and column respectively. The solution for this problem is to assign values for x and y such that the constraint is satisfied. The problem can be formulated as:

$$P = \{(x_1, y_1), (x_2, y_2), \dots, (x_8, y_8)\}$$

where  $(x_i, y_i)$  gives the position of the  $i^{th}$  queen from  $1^{st}$  to the  $8^{th}$ .

So, it can be clearly seen that both the values of  $x_i$  and  $y_i$  range from 1 to 8.

The constraint are :

- No two queens should be in the same row i.e.  $y_i \neq y_j$  for  $i = 1$  to  $8$ ;  $j = 1$  to  $8$ ;  $i \neq j$ .
- No two queens should be in the same column i.e.  $x_i \neq x_j$  for  $i = 1$  to  $8$ ;  $j = 1$  to  $8$ ;  $i \neq j$ .
- There should not be two queens placed on the same diagonal line. That is,  $(y_i - y_j) \neq \pm(x_i - x_j)$  Now a solution to this problem is an instance of P wherein the above-mentioned constraints are satisfied

```
DOMAINS
cell = c(integer, integer)
list = cell*
int_list = integer*

PREDICATES
solution(list)
member(integer, int_list)
noattack(cell, list)

CLAUSES
solution([]).
solution([c(X,Y)|Others]):-
solution(Others),
member(Y,[1,2,3,4,5,6,7,8]),
noattack(c(X,Y),Others).
noattack(_, []).
noattack(c(X,Y),[c(X1,Y1)|Others]):-
Y<>Y1,
Y1-Y<>X1-X,
Y1-Y<>X-X1,
```


```

noattack(c(X,Y),Others).
member(X,[X|_]).
member(X,[_|Z]):-
member(X,Z).

GOAL
solution([c(1,1),c(2,B),c(3,C),c(4,8),c(5,2),c(6,F),c(7,G),c(8,H)]).

```

### • Output:

 [Inactive C:\Users\bibek\AppData\Local\Temp\goal\$000.exe]

```

A=1, B=6, C=8, D=3, E=7, F=4, G=2, H=5
A=3, B=8, C=4, D=7, E=1, F=6, G=2, H=5
A=6, B=3, C=7, D=4, E=1, F=8, G=2, H=5
A=7, B=4, C=2, D=8, E=6, F=1, G=3, H=5
A=4, B=6, C=8, D=2, E=7, F=1, G=3, H=5
A=2, B=6, C=1, D=7, E=4, F=8, G=3, H=5
A=2, B=4, C=6, D=8, E=3, F=1, G=7, H=5
A=3, B=6, C=8, D=2, E=4, F=1, G=7, H=5
A=6, B=3, C=1, D=8, E=4, F=2, G=7, H=5
A=8, B=4, C=1, D=3, E=6, F=2, G=7, H=5
A=4, B=8, C=1, D=3, E=6, F=2, G=7, H=5
A=2, B=6, C=8, D=3, E=1, F=4, G=7, H=5
A=7, B=2, C=6, D=3, E=1, F=4, G=8, H=5
A=3, B=6, C=2, D=7, E=1, F=4, G=8, H=5
A=4, B=7, C=3, D=8, E=2, F=5, G=1, H=6
A=4, B=8, C=5, D=3, E=1, F=7, G=2, H=6
A=3, B=5, C=8, D=4, E=1, F=7, G=2, H=6
A=4, B=2, C=8, D=5, E=7, F=1, G=3, H=6
A=5, B=7, C=2, D=4, E=8, F=1, G=3, H=6
A=7, B=4, C=2, D=5, E=8, F=1, G=3, H=6
A=8, B=2, C=4, D=1, E=7, F=5, G=3, H=6
A=7, B=2, C=4, D=1, E=8, F=5, G=3, H=6
A=5, B=1, C=8, D=4, E=2, F=7, G=3, H=6
A=4, B=1, C=5, D=8, E=2, F=7, G=3, H=6
A=5, B=2, C=8, D=1, E=4, F=7, G=3, H=6
A=3, B=7, C=2, D=8, E=5, F=1, G=4, H=6
A=3, B=1, C=7, D=5, E=8, F=2, G=4, H=6
A=8, B=2, C=5, D=3, E=1, F=7, G=4, H=6
A=3, B=5, C=2, D=8, E=1, F=7, G=4, H=6
A=3, B=5, C=7, D=1, E=4, F=2, G=8, H=6
A=5, B=2, C=4, D=6, E=8, F=3, G=1, H=7
A=6, B=3, C=5, D=8, E=1, F=4, G=2, H=7
A=5, B=8, C=4, D=1, E=3, F=6, G=2, H=7
A=4, B=2, C=5, D=8, E=6, F=1, G=3, H=7
A=4, B=6, C=1, D=5, E=2, F=8, G=3, H=7
A=6, B=3, C=1, D=8, E=5, F=2, G=4, H=7
A=5, B=3, C=1, D=6, E=8, F=2, G=4, H=7
A=4, B=2, C=8, D=6, E=1, F=3, G=5, H=7
A=6, B=3, C=5, D=7, E=1, F=4, G=2, H=8
A=6, B=4, C=7, D=1, E=3, F=5, G=2, H=8
A=4, B=7, C=5, D=2, E=6, F=1, G=3, H=8
A=5, B=7, C=2, D=6, E=3, F=1, G=4, H=8
92 Solutions

```

**Testing the goal by placing some of the queens explicitly:**

The above goal resulted in 92 solutions. This means there are 92 different ways to place the queens so that no two queens share the same row, column or diagonal.

- When  $[c(1, 1), c(2, B), c(3, C), c(4, D), c(5, E), c(6, F), c(7, G), c(8, H)]$

```
[Inactive C:\Users\bibek\AppData\Local\Temp\goal$000.exe]
B=7, C=4, D=6, E=8, F=2, G=5, H=3
B=7, C=5, D=8, E=2, F=4, G=6, H=3
B=5, C=8, D=6, E=3, F=7, G=2, H=4
B=6, C=8, D=3, E=7, F=4, G=2, H=5
4 Solutions
```

- When  $[c(1, A), c(2, B), c(3, C), c(4, 4), c(5, E), c(6, 8), c(7, G), c(8, H)]$

```
[Inactive C:\Users\bibek\AppData\Local\Temp\goal$000.exe]
A=2, B=5, C=7, E=1, G=6, H=3
A=5, B=7, C=1, E=2, G=6, H=3
A=6, B=3, C=7, E=1, G=2, H=5
3 Solutions
```

- When  $[c(1, 1), c(2, B), c(3, C), c(4, 8), c(5, 2), c(6, F), c(7, G), c(8, H)]$

```
[Inactive C:\Users\bibek\AppData\Local\Temp\goal$000.exe]
B=7, C=5, F=4, G=6, H=3
1 Solution
```

## C++ Programs

---

- SEND + MORE = MONEY

```
#include<iostream>
using namespace std;
int main()
{
    int s, e, n, d, m, o, r, y;
    int counter = 1;
    int send, more, money;
    for(s=0;s<=9;s++) {
        for(e=0;e<=9;e++) {
            for(n=0;n<=9;n++) {
                for(d=0;d<=9;d++) {
                    for(m=0;m<=9;m++) {
                        for(o=0;o<=9;o++) {
                            for(r=0;r<=9;r++) {
                                for(y=0;y<=9;y++) {
                                    send=s*1000+e*100+n*10+d*1;
                                    more=m*1000+o*100+r*10+e*1;
                                    money=m*10000+o*1000+n*100+e*10+y*1;
                                    if((s!=e)&&(s!=n)&&(s!=d)&&(s!=m)&&(s!=o)
                                        &&(s!=r)&&(s!=y)&&(e!=n)&&(e!=d)&&(e!=m)&&
                                        (e!=o)&&(e!=r)&&(e!=y)&&(n!=d)&&(n!=m)&&
                                        (n!=o)&&(n!=r)&&(n!=y)&&(d!=m)&&(d!=o)&&
                                        (d!=r)&&(d!=y)&&(m!=o)&&(m!=r)&&(m!=y)&&
                                        (o!=r)&&(o!=y)&&(r!=y)&&(send+more==money)
                                        &&(m==1))
                                    {
                                        cout<<"S="<<s<<"\t"<<"E="<<e<<"\t"
                                        <<"N="<<n<<"\t"<<"D="<<d<<"\t"<<
                                        "M="<<m<<"\t"<<"O="<<o<<"\t"<<"R="
                                        <<r<<"\t"<<"Y="<<y<<"\n";
                                        counter++;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return 0;
}
```

```
}
```

### Output:

```
ailab2 x
/home/bibek/CLionProjects/ailab2/cmake-build-debug/ailab2
S=9 E=5 N=6 D=7 M=1 O=0 R=8 Y=2

Process finished with exit code 0
```

### • Eight Queen Problem

```
#include <iostream>
using namespace std;

char chessBoard[8][8];

void makeChessBoard() {
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            cout << chessBoard[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
    cout << endl;
}

bool isSafe(int row, int col) {
    for (int i = 0; i < row; ++i) {
        if (chessBoard[i][col] == 'Q') return false;
    }
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (chessBoard[i][j] == 'Q') return false;
    }
    for (int i = row, j = col; i >= 0 && j < 8; j++, i--) {
        if (chessBoard[i][j] == 'Q') return false;
    }
    return true;
}

bool solve(int row) {
    if (row == 8) {
        makeChessBoard();
        return true;
    }
}
```

```

    bool result = false;
    for (int i = 0; i < 8; i++) {
        if (isSafe(row, i)) {
            chessBoard[row][i] = 'Q';
            // If the value of result will be false then only,
            // backtracking will occur
            result = solve(row + 1) || result;
            chessBoard[row][i] = '-';
        }
    }

    return result;
}

int main() {
    bool result = solve(0);
    if (!result) {
        cout << "The solution does not exist" << endl;
    }
    return 0;
}

```

**Output:**

```

- - - - - Q
- - - Q - - -
Q - - - - -
- - Q - - - -
- - - - - Q -
- Q - - - - -
- - - - - Q -
- - - - Q - -

```

## Discussion

---

Thus, the cryptarithmic and 8 queen problems can be solved using constraint programming. In order to solve a cryptarithmic problem, we need to assign each digit an unique alphabets and get the corresponding digit in the range of 0 to 9 for the problem to be solved correctly. However, to solve 8 Queens problem, the queens should be placed in such a position that no two queens share the same row or column or diagonal.

## Conclusion

---

Hence, in this way we can solve the constraint satisfaction problems.