

# **Project. Satisfiability test of clauses and its application**

Gopichand Reddy Doggala

N-Queens Problem

## **Introduction**

In chess, a queen can attack horizontally, vertically, and diagonally. Our problem is to place  $N$  queens on a  $N$  by  $N$  chessboard so that no two of the queens attack each other. It can be converted to a propositional satisfiability problem, where each Boolean (T or F) represents one cell of the chessboard. If the Boolean is set to true, there is a queen on the cell, and is false the cell is empty. A binary matrix ( $N \times N$ ) is used to display the positions of  $N$  Queens, where no queens can attack other queens in horizontally or vertically or diagonally.

## **CNF (Conjunctive Normal Form)**

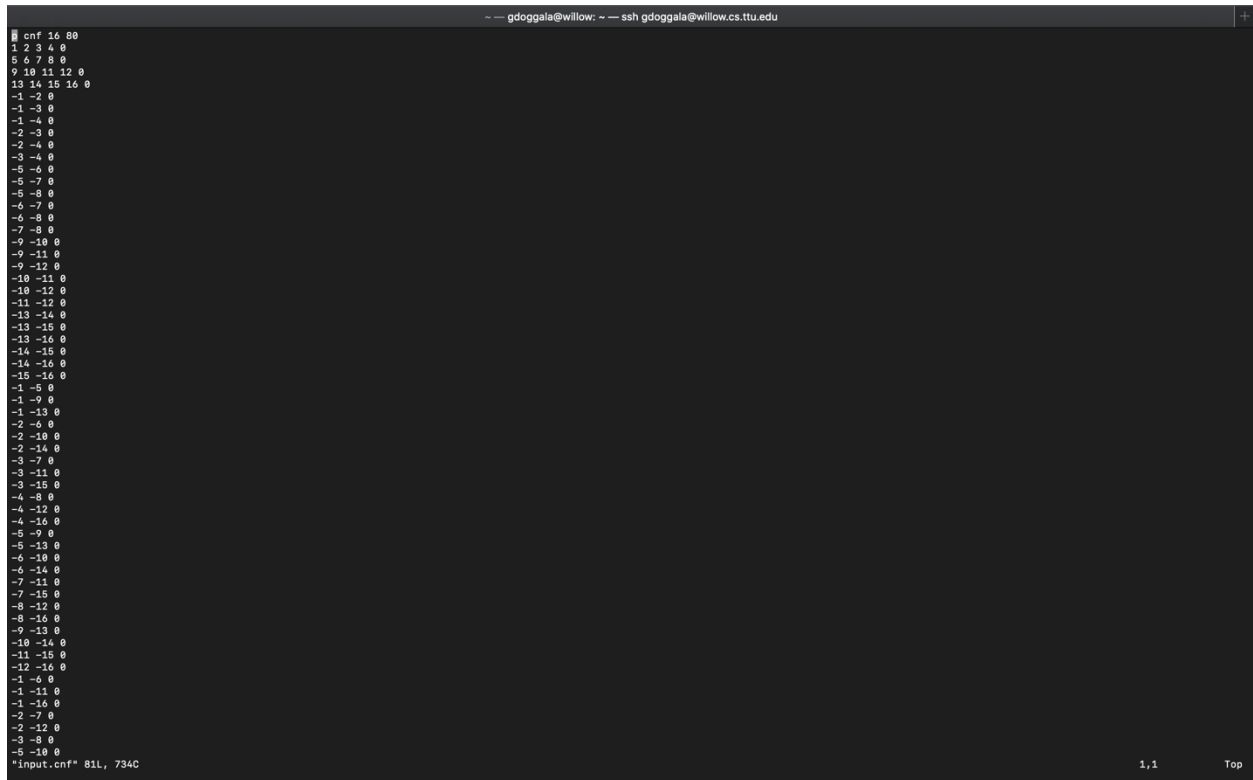
Conjunctive normal form (CNF) is an important normal form for propositional logic. It is an approach to Boolean logic that expresses formulas as conjunctions of clauses with an AND or OR. Each clause connected by a conjunction, or AND, must be either a literal or contain a disjunction, or OR operator. CNF is useful for automated theorem proving.

## **SAT Solver (Satisfiability solver):**

A propositional logic formula which is also known as Boolean expression, is built from variables, operators like AND (conjunction – ‘ $\wedge$ ’), OR (disjunction – ‘ $\vee$ ’), NOT (negation – ‘ $\neg$ ’). A formula is said to be satisfiable if it can be made TRUE by assigning appropriate truth values to its variables. The SAT is given a formula, to check whether it is satisfiable. A SAT solver is a tool that takes as input a CNF formula and outputs either a satisfying Boolean assignment to the variables used in the CNF formula if the formula is consistent or UNSAT if it is not.

In code we created an Input file in Conjunctive Normal Form(CNF) format and enter the input values in it – input.cnf.

Open the input.cnf file using ‘vi input.cnf’



```
cnf 16 80
1 2 3 4 0
5 6 7 8 0
9 10 11 12 0
13 14 15 16 0
-1 -2 0
-1 -3 0
-1 -4 0
-2 -3 0
-2 -4 0
-3 -4 0
-5 -6 0
-5 -7 0
-5 -8 0
-6 -7 0
-6 -8 0
-7 -8 0
-9 -10 0
-9 -11 0
-9 -12 0
-10 -11 0
-10 -12 0
-11 -12 0
-13 -14 0
-13 -15 0
-13 -16 0
-14 -15 0
-14 -16 0
-15 -16 0
-1 -5 0
-1 -9 0
-1 -13 0
-2 -6 0
-2 -10 0
-2 -14 0
-3 -7 0
-3 -11 0
-3 -15 0
-4 -8 0
-4 -12 0
-4 -16 0
-5 -9 0
-5 -13 0
-6 -10 0
-6 -14 0
-7 -11 0
-7 -15 0
-8 -12 0
-8 -16 0
-9 -13 0
-10 -14 0
-11 -15 0
-12 -16 0
-1 -6 0
-1 -11 0
-1 -16 0
-2 -7 0
-2 -12 0
-3 -8 0
-5 -10 0
```

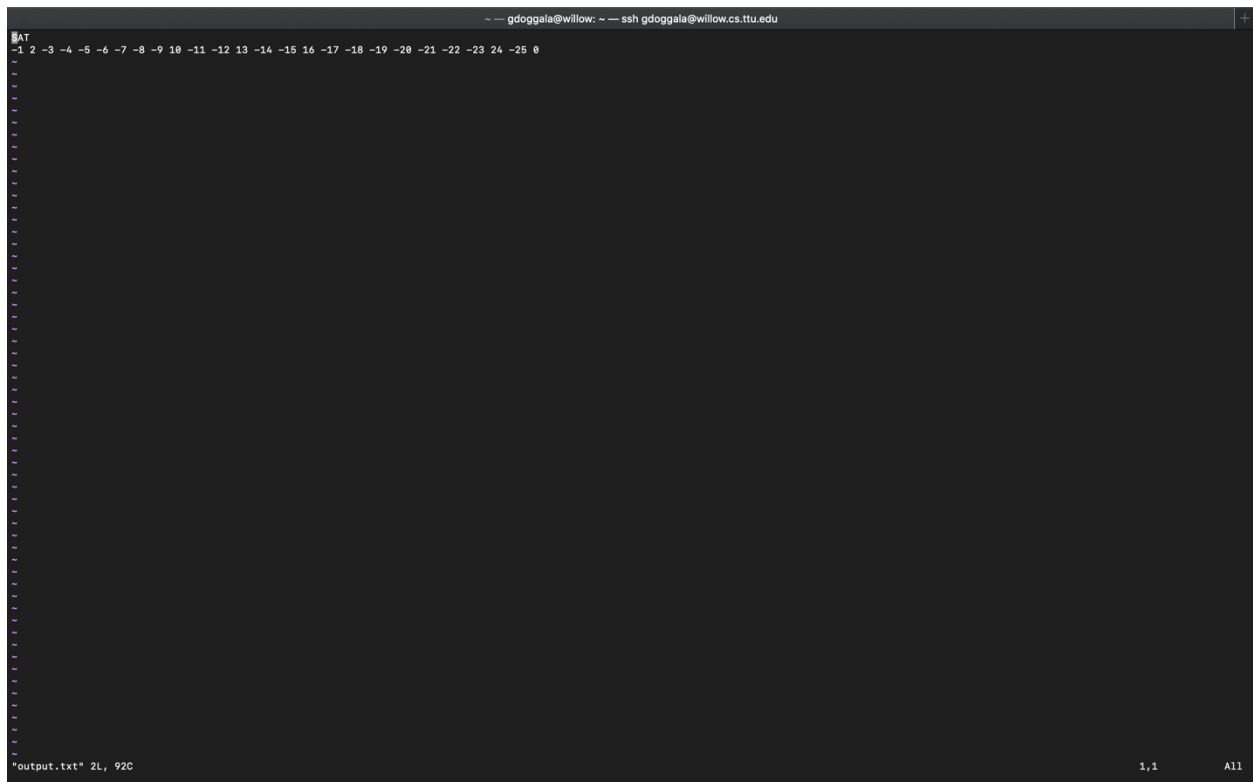
"input.cnf" 81L, 734C 1,1 Top

Now run the following Linux command to retrieve the output ‘Python3 LCS\_NQueens\_Project.py’. and give your input value. The output will be saved in ‘output.txt’ file.

```
gdoggala@willow:~$ python3 LCS_NQueens_Project.py
Enter n: 5
WARNING: for repeatability, setting FPU to use double precision
===== Problem Statistics =====
| Number of variables:      25 |
| Number of clauses:       165 |
| Parse time:              0.00 s |
| Eliminated clauses:      0.00 Mb |
| Simplification time:     0.00 s |
=====
===== Search Statistics =====
| Conflicts | ORIGINAL | LEARNNT | Progress |
|  Vars  | Clauses | Literals | Limit | Clauses | Lit/Cl |
=====
restarts      : 1
conflicts     : 0          (-nan /sec)
decisions     : 7          (0.00 % random) (inf /sec)
propagations  : 20         (inf /sec)
conflict literals : 0          (-nan % deleted)
Memory used   : 21.00 MB
CPU time      : 0 s

SATISFIABLE
gdoggala@willow:~$
```

Output is saved in 'output.txt'. Open output.txt using 'vi output.txt'



```
gdoggala@willow: ~ — ssh gdoggala@willow.cs.ttu.edu
-1 2 -3 -4 -5 -6 -7 -8 -9 10 -11 -12 13 -14 -15 16 -17 -18 -19 -20 -21 -22 -23 24 -25 0
```

## Code walkthrough:

**Input:** The chess board, the column where the queen is trying to be placed.

**Output:** The position matrix where queens are placed.

Begin

if all columns are filled, then

return true

for each row of the board, do

if isValid(board, i, col), then

set queen at place (i, col) in the board

if solveNQueen(board, col+1) = true, then

return true

otherwise remove queen from place (i, col) from board.

done

return false

End

**Input:** The chess board, row and the column of the board.

**Output:** True when placing a queen in row and place position is a valid or not.

Begin

if there is a queen at the left of current column, then

return false

if there is a queen at the left upper diagonal, then

return false

if there is a queen at the left lower diagonal, then

return false;

return true //otherwise it is valid place

End

### **Problem Explanation:**

N-Queens problem as a SAT problem. The N-Queens problem can be converted to a propositional (or Boolean) satisfiability problem and solved very efficiently.

We will use a 4-Queens problem as an example.

First assign a Boolean variable to each cell of the 4X4 board with the meaning of **X<sub>ij</sub> is true iff there is a queen at X<sub>ij</sub>**:

X<sub>11</sub> X<sub>12</sub> X<sub>13</sub> X<sub>14</sub>

X<sub>21</sub> X<sub>22</sub> X<sub>23</sub> X<sub>24</sub>

X<sub>31</sub> X<sub>32</sub> X<sub>33</sub> X<sub>34</sub>

X<sub>41</sub> X<sub>42</sub> X<sub>43</sub> X<sub>44</sub>

In simple terms:

$$X_{11} + X_{12} + X_{13} + X_{14} = 1$$

can be converted as:

(a) One of the variable must be true.

This is equivalent to:  $X_{11} \vee X_{12} \vee X_{13} \vee X_{14}$

(b) If  $X_{1i}$  is true, then  $X_{1j}$  is not true for every  $j \neq i$ .

This is equivalent to:

$$X_{1i} \rightarrow \neg X_{1j}$$

Which is again equivalent to:

$$\neg X_{1i} \vee \neg X_{1j}$$

Hence  $X_{11} + X_{12} + X_{13} + X_{14} = 1$  can be converted to 7 clauses:

$$X_{11} \vee X_{12} \vee X_{13} \vee X_{14}$$

$$\neg X_{11} \vee \neg X_{12}$$

$$\neg X_{11} \vee \neg X_{13}$$

$$\neg X_{11} \vee \neg X_{14}$$

$$\neg X_{12} \vee \neg X_{13}$$

$$\neg X_{12} \vee \neg X_{14}$$

$$\neg X_{13} \vee \neg X_{14}$$