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 X 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)

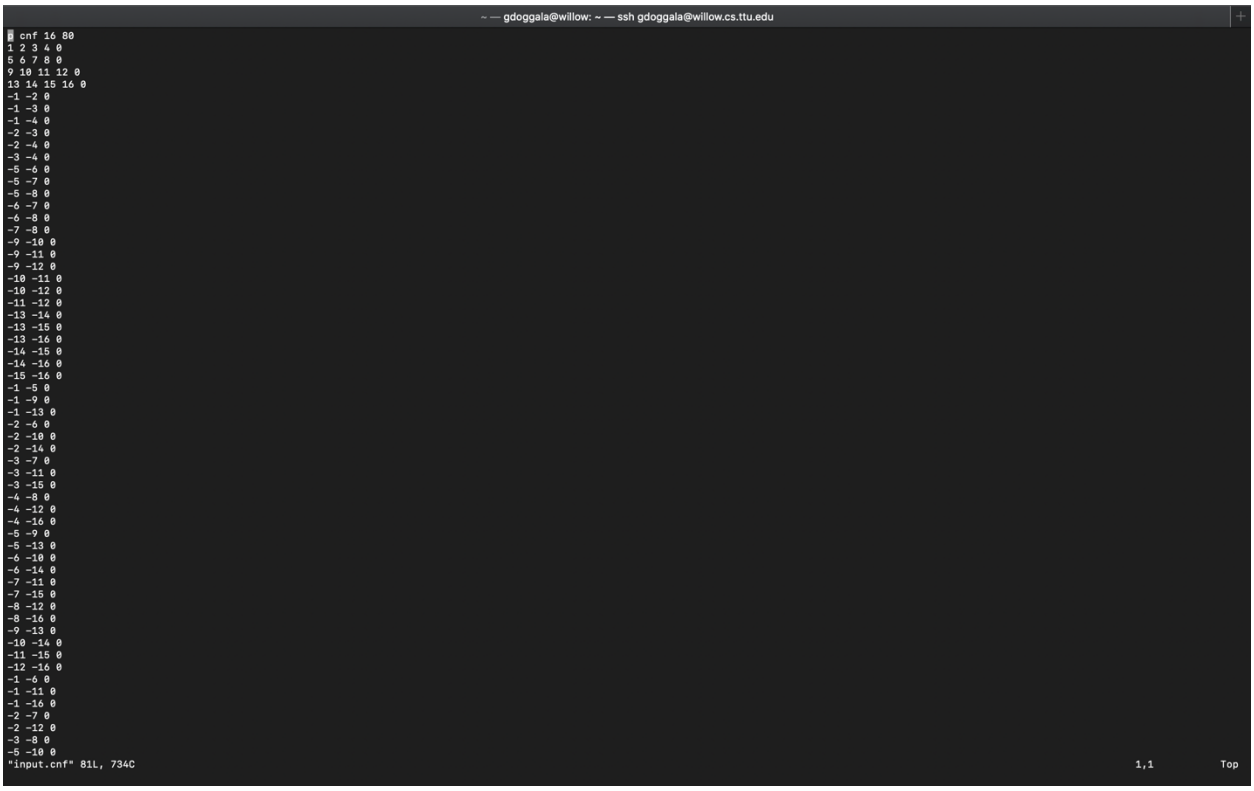
Conjunctive normal form (CNF) is an important normal form for propositional logic. It is an approach to Boolean logic that expresses formulas as conjunctions of clauses with an AND or OR. Each clause connected by a conjunction, or AND, must be either a literal or contain a disjunction, or OR operator.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

SAT Solver (Satisfiability solver):

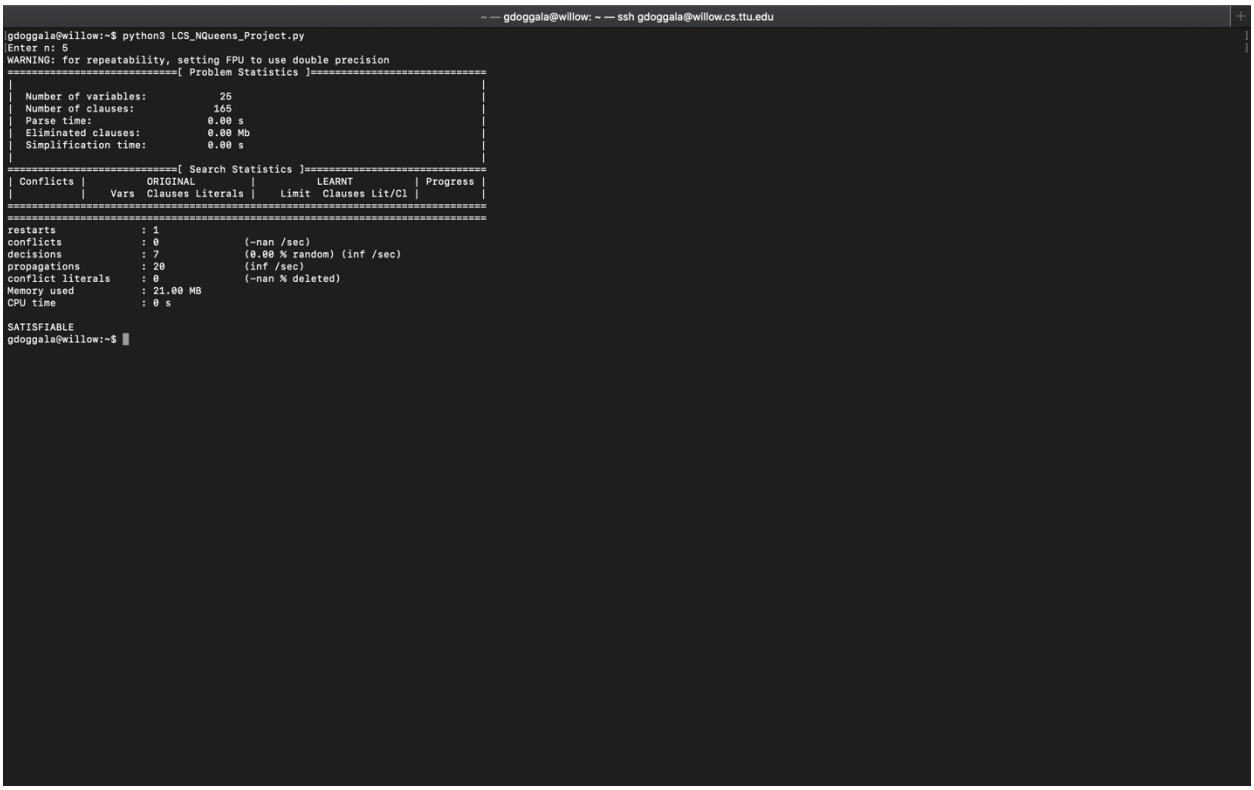
A propositional logic formula which is also known as Boolean expression, is built from variables, operators like AND (conjunction – '∧'), OR (disjunction – '∨'), NOT (negation – '¬'). A formula is said to be satisfiable if it can be made TRUE by assigning appropriate truth values to its variables. The 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.

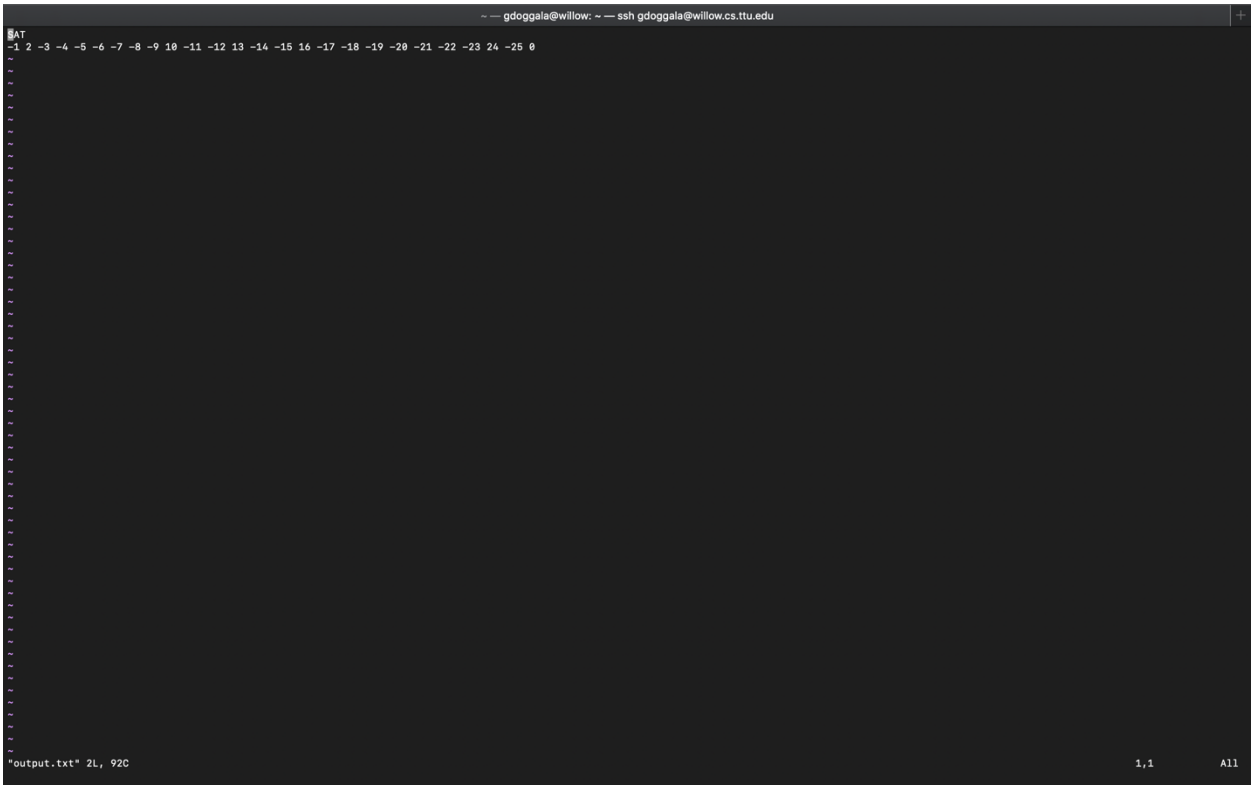
Open the input.cnf file using ‘vi input.cnf’



Now run the following Linix command to retrieve the output ‘Python3 LCS_NQueens_Project.py’. and give your input value. The output will be saved in ‘output.txt’ file.



Output is saved in ‘output.txt’. Open output.txt using ‘vi output.txt’



Code walkthrough:

Input: The chess board, the colomn 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 colomn, 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 **Xij is true iff there is a queen at Xij**:

X11 X12 X13 X14

X21 X22 X23 X24

X31 X32 X33 X34

X41 X42 X43 X44

In simple terms:

X11 + X12 + X13 + X14 = 1

can be converted as:

(a) One of the variable must be true.

This is equivalent to: X11 V X12 V X13 V X14

(b) If X1i is true, then X1j is not true for every j!=i.

This is equivalent to:

X1i->!X1j

Which is again equivalent to:

!X1i V !X1j

Hence X11 + X12 + X13 + X14 = 1 can be converted to 7 clauses:

X11 V X12 V X13 V X14

!X11 V !X12

!X11 V !X13

!X11 V !X14

!X12 V !X13

!X12 V !X14

!X13 V !X14