Monika Gnyp
10734125
CS370: NQueens Problem
Route A.
i)
Assuming the nxn board, let the set S={ x_{11},x_{12}...x_{nn} (general case: x_{ij} , for 0<i<=n and 0<j<=n) } be the squares on the board. The elements of S are called variables and it is clear that we have n^2 elements in S. Each variable can be either true (presence of a queen) or false (no queen). Firstly let's define basic rules that a given configuration of queens has to follow in order to be considered a valid solution for NQueens problem. Each rule is an implication which is then rewritten in terms of disjunctions and conjunctions so we can arrive at the formula in conjunctive normal form which is suitable for SAT solvers (since it is an instance of SAT problem).

➔ Rows:

1. There must be exactly 1 queen in each row. We have that $x_{i1} \vee x_{i2} \vee x_{i3} \dots \vee x_{in}$.
2. We also have to ensure that once a particular variable is set to true, all other variable in that row are false. It follows that $x_{i1} \rightarrow (\neg x_{i2} \wedge \neg x_{i3} \dots \wedge \neg x_{in})$. We can simplify this expression to $p \rightarrow \neg q$, where p means queen present and q is the set of remaining squares (variables) in that row. By material implication we have that :
   $p \rightarrow q \Rightarrow \neg p \vee q$, and so we have that: $p \rightarrow \neg q \Rightarrow \neg p \vee \neg q$. We have to generate a set of clauses of that kind and and them together. For instance if n = 4 and i = 1 we have that :
   $(\neg x_{11} \vee \neg x_{12}) \wedge (\neg x_{11} \vee \neg x_{13}) \wedge (\neg x_{11} \vee \neg x_{14}) \wedge (\neg x_{12} \vee \neg x_{13}) \wedge (\neg x_{12} \vee \neg x_{14}) \wedge (\neg x_{13} \vee \neg x_{14})$.
3. Combining rule 1 with rule 2 we have: $(x_{i1} \vee x_{i2} \vee x_{i3} \dots \vee x_{in}) \wedge (\neg x_{i1} \vee \neg x_{i2}) \wedge (\neg x_{i1} \vee \neg x_{i3}) \dots (\neg x_{in-1} \vee \neg x_{in})$ for some integer i.
   In case of n = 4, this can be described as: $(x_{11} \vee x_{12} \vee x_{13} \vee x_{14}) \wedge (\neg x_{11} \vee \neg x_{12}) \wedge (\neg x_{11} \vee \neg x_{13}) \wedge (\neg x_{11} \vee \neg x_{14}) \wedge (\neg x_{12} \vee \neg x_{13}) \wedge (\neg x_{12} \vee \neg x_{14}) \wedge (\neg x_{13} \vee \neg x_{14})$.

➔ Columns:

1. There must be exactly 1 queen in each column. We have that $x_{1j} \vee x_{2j} \vee x_{3j} \dots \vee x_{jn}$.
2. The same rule applies as in case of the rows: once a particular variable is set to true, all other variable in that column are false. It follows that $x_{1j} \rightarrow (\neg x_{2j} \wedge \neg x_{3j} \dots \wedge \neg x_{nj})$, which simplifies to $p \rightarrow \neg q$ and using the material implication rule we arrive at
   $p \rightarrow \neg q \Rightarrow \neg p \vee \neg q$. We have to generate a set of clauses of that kind and and them together. For instance if n = 4 and j = 1 we have that :
   $(\neg x_{11} \vee \neg x_{21}) \wedge (\neg x_{11} \vee \neg x_{31}) \wedge (\neg x_{11} \vee \neg x_{41}) \wedge (\neg x_{21} \vee \neg x_{31}) \wedge (\neg x_{21} \vee \neg x_{41}) \wedge (\neg x_{31} \vee \neg x_{41})$.
3. Combining rule 1 with rule 2 we have: $(x_{1j} \vee x_{2j} \vee x_{3j} \dots \vee x_{nj}) \wedge (\neg x_{1j} \vee \neg x_{2j}) \wedge (\neg x_{1j} \vee \neg x_{3j}) \dots (\neg x_{n-1j} \vee \neg x_{nj})$ for some integer j.
   In case of n = 4 and j = 1 , this can be described as: $(x_{11} \vee x_{21} \vee x_{31} \vee x_{41}) \wedge (\neg x_{11} \vee \neg x_{21}) \wedge (\neg x_{11} \vee \neg x_{31}) \wedge (\neg x_{11} \vee \neg x_{41}) \wedge (\neg x_{21} \vee \neg x_{31}) \wedge (\neg x_{21} \vee \neg x_{41}) \wedge (\neg x_{31} \vee \neg x_{41})$.

➔ Diagonals:

1. There must be at most 1 queen in each diagonal. We can reuse the derivation based on material implication rule from point 2 describe above (the rule used is the same in case of the rows and columns). This time we don't have to ensure that 1 queen has to be placed along any given diagonal (I am referring to point 1 in columns and point 1 in rows), so this condition is not taken into consideration. Thus we have that:

$(\neg x_{ij} \vee \neg x_{(i+1)(j+1)}) \wedge (\neg x_{(i+1)(j+1)} \vee \neg x_{(i+2)(j+2)}) \ldots (\neg x_{(i+n-2)(j+n-2)} \vee \neg x_{(i+n-1)(j+n-1)})$ for some integers i and j denoting the indices. Please note that there are 2 types of diagonals (right to left and left to right) and a bounds check is required when creating the clauses since the diagonals differ in length (from 2 elements up to n elements). Analogically to the rule for right to left diagonals presented above, the left to right rule can be described as: $(\neg x_{ij} \vee \neg x_{(i-1)(j+1)}) \wedge (\neg x_{(i-1)(j+1)} \vee \neg x_{(i-2)(j+2)}) \ldots (\neg x_{(i-(n-2))(j+(n-2))} \vee \neg x_{(i-(n-1))(j+(n-1))})$ for some integers i and j denoting the indices.

Combining (anding using a logical ∧) the rules for rows, columns and diagonals gives a complete set of clauses in propositional logic in CNF. We then port it into a CNF file by adhering to the syntax (specifying the number of clauses and variables, numbering the variables 1 … n, replacing ∨ with whitespaces and ∧ with zeros). We have now reduced nQueens problem to boolean satisfiability problem. A SAT solver (I used Sat4j) fed with such a formula outputs the set of solutions (that is a set of variable assignments that satisfy the formula) which in turn is also a set of solutions to the nQueens problem. The reduction is polynomial time many-to-one since the algorithm described above translates nQueens problem into an instance of SAT in polynomial time, that is there must be n^2 variables and we must generate $C(n, k) + 1$ (n over k) + 1 (the binomial coefficient, "n choose k") clauses per case. Since we have 3 cases (rows, columns, diagonals) and we pick 2 variables at a time (k = 2) plus the original statement (there must be 1 queen per row and 1 queen per column) apart from the diagonals (there must be at most one queen per diagonal) we have: $2*(C(n, 2) + 1) + C(n, 2)$ clauses with n^2 variables. Once the set of clauses has been generated it cannot be modified (if we treat the reduction algorithm as an oracle, the oracle can be queried only once). Such SAT problem can be solved in polynomial time (in case of most SAT instances, since SAT is NP-complete and no algorithm exists to solve all instances of SAT).

ii)

Firstly it is worth noting that there are polynomial time algorithms to solve nQueens problem if the expected output is a solution to the problem (Hoffman, Loessi and Moore. *Constructions for the Solution of the m Queens Problem*, Mathematics Magazine, p. 66-72, 1969.). Finding the number of all solutions and subsequently finding all solutions is considered a much harder problem, this is in NP. Therefore we will refer to the latter when proving the reduction from nQueens to boolean satisfiability problem.

Proof:

Suppose that SAT problem is NP-Complete. The proof that this is the case was published by Cook and Levin (cook-levin theorem) in 1971 (Cook) and 1973 (Levin) and 1971 (Cook) and 1973 (Levin) and can be found in the original papers and wikipedia. It's natural consequence is that each problem in NP can be reduced in polynomial time using a deterministic Turing Machine to SAT problem.

1. We prove SAT to be NP-Complete (via Cook-Levin theorem) and we prove nQueens to be in NP (using backtracking), we won't include the proofs, they can be found online.
2. Let kQ be an instance of nQueens problem such that k = 1 or k>=4 (we have an nQueens instance for k queens). Since nQueens is an NP problem, it has a polynomial time verifier V that decides if a given solution is a valid one for a particular instance of nQueens problem. Let w be a solution to kQ. Construct a decider V1 from V such that

        - V1 accepts if w is a valid solution to kQ.
        - V1 rejects if w is not a valid solution to kQ.
    3. Let zSAT be an k2 instance of SAT problem (where k corresponds to kQ).
    4. Using algorithm described in i) we can construct such zSAT in polynomial time with 2*(C(k, 2) + 1) + C(k, 2) clauses and with k2 variables.
    5. It follows that:
        - if kQ is an instance of nQueens, then zSAT must be an instance of SAT
        - if zSAT is an instance of of SAT, then kQ must be is an instance of nQueens

Alternative Proof by contradiction (if the reduction is correct no solution should exist for n = 2 and n = 3):

Suppose f is a function that reduces the nQueens problem to SAT problem with n variables (maps nQueens for some n to nSAT): f = { $<N_1,n>$ N_1 is a function (mapping / polynomial time many-to-one reduction) that on input n translates the nQueens problem to a boolean formula in conjunctive normal form (that is it reduces nQueens problem to SAT } using n variables. Let M_1 be a recognizer to SAT problem: M_1 = { $<N_1,fCNF>$ N_1 is a Turing Machine that on input fCNF accepts if fCNF is satisfiable. N_1 uses backtracking to establish if a formula is satisfiable, thus it can potentially loop forever (for large instances of SAT, please note that SAT is NP-complete)}. Let M_2 be an algorithm that solves instances of SAT problem: M_2 = { $<N_2,fCNF>$ N_2 is an algorithm that on input fCNF produces solutions to the instance of SAT problem described by fCNF (boolean formula in CNF). N_2 has been described in i) }.

1. Run f on input n to obtain corresponding fCNF. This is a reduction from nQueens to nSAT.
2. Run M_1 on input fCNF. If M_1 accepts, accept. Otherwise reject, fCNF is not satisfiable.
3. Run M_2 on input fCNF. M_2 outputs a solution k if M_1 accepted, otherwise it outputs nothing.
4. Construct a decider X from M_2 that verifies the solution to SAT problem in polynomial time: X = { $<N_3,fCNF, k>$ fCNF is a boolean formula, k is an assignment of true/false values to fCNF, N_3 is a Turing Machine that accepts if k satisfies fCNF, otherwise it rejects }.
5. If k exists run X on input k. If M_1 accepted, X must accept. Otherwise X must reject.

If n = 3 or n =2 we can clearly see that M_1 will reject since there exists no assignment of true/false such that the formula created by f is satisfiable, M_2 must output no solutions, thus k must not exist.

iii)

Firstly we have to use the algorithm described in i) to generate a boolean formula in conjunctive normal form for some value of n. Such a formula is fed to the SAT solver. If the formula is satisfiable, the SAT solver will output a set of solutions, which we can enumerate. Each solution from the set can be translated back to the nQueens problem domain by attaching labels to chessboard squares that represent variables. If a variable is found to be set to true, the appropriate chessboard square is populated with a queen, otherwise it stays empty.

A strategy to enumerating all solutions is described below:

1. Generate a set of clauses for a particular value of n using the algorithm described in i).
2. Feed the formula into a SAT solver and obtain a first solution.

3. Append the first solution to the original formula (not the solution and and it with the original). That way the resulting formula has the same logical meaning with the restriction that the solution already seen is not considered a valid assignment (which is achieved by noting the solution).
4. Repeat for each solution noting and appending it to the original formula (using logical ands). The formula will keep on expanding.
5. Run the algorithm described above until the formula becomes unsatisfiable.
6. Once the formula becomes unsatisfiable there exists no more possible solutions.

iv)

A solution is considered fundamental (unique) if can not be derived by symmetry operations (reflections and rotations) from other solutions. Thus per each solution we potentially have 8 solutions that can be produced by the symmetry operations (4 rotations (by 0 degrees = original solution, by 90 degrees, 180 degrees and by 270 degrees) and 4 reflections (one reflection per each rotation including the original solutions). Let f(x) be a function that on input x (some solution in string format) translates x into a boolean matrix and produces rotations and reflections described above, which are then translated back into strings in CNF form. f(x) outputs a set of those strings. Thus we can describe an algorithm that filters out unique solutions from the set of all solutions:

1. Run the algorithm described in i) to obtain a set of solutions to nQueens problem.
2. Copy the solutions into a temporary set of uniqueSolutions (since we want to keep the set of original solutions, this however is not necessary).
3. Assume that i is an integer such that i >= 0 and i < the size of uniqueSolutions. Iterate over the elements of uniqueSolutions. Run f(x) on each element in the set (that is uniqueSolutions.get(i)) and store the output of f(x) in temp. Run a second loop thru all remaining elements in uniqueSolutions.
4. If temp contains a solution that is also in uniqueSolutions, remove that solution from uniqueSolutions.
5. Continue until no more solutions are to be examined in uniqueSolutions.
6. At the end of the algorithm, the remaining solutions in uniqueSolutions are indeed fundamental since all symmetric solutions to those have be prior removed.