Assignment No-04(Group B)

Title: - Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

Objectives:-

Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

Problem Statement:-

Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

Software and Hardware requirements:-

1. **Operating system:** Linux- Ubuntu 16.04 to 17.10, or Windows 7 to 10,
2. **RAM-** 2GB RAM (4GB preferable)
3. You have to install **Python3** or higher version

Theory-

A constraint satisfaction problem consists of three components, X,D, and C:

X is a set of variables, {X1, . . . ,Xn}.

D is a set of domains, {D1, . . . ,Dn}, one for each variable.

C is a set of constraints that specify allowable combinations of values.

Each domain Di consists of a set of allowable values, {v1, . . . , vk} for variable Xi. Each constraint Ci consists of a pair _scope, rel _, where scope is a tuple of variables that participate in the constraint and rel is a relation that defines the values that those variables can take on. A relation can be represented as an explicit list of all tuples of values that satisfy the constraint, or as an abstract relation that supports two operations: testing if a tuple is a member of the relation and enumerating the members of the relation. For example, if X1 and X2 both have the domain {A,B}, then the constraint saying the two variables must have different values can be written as <(X1,X2), [(A,B), (B,A)]> or as <(X1,X2),X1 ≠ X2>.

To solve a CSP, we need to define a state space and the notion of a solution. Each state in a CSP is defined by an assignment of values ASSIGNMENT to some or all of the variables, {Xi =vi,Xj = vj , . . .}.

An assignment that does not violate any constraints is called a consistent or legal assignment.

A complete assignment is one in which every variable is assigned, and a solution to a CSP is a consistent, complete assignment. A partial assignment is one that assigns values to only some of the variables.

The N-Queens problem is a puzzle of placing exactly N queens on an NxN chessboard, such that no two queens can attack each other in that configuration. Thus, no two queens can lie in the same row,column or diagnol.

In the backtracking algorithm, we consider possible configurations one by one and backtrack if we hit a dead end.

The branch and bound solution is somehow different, it generates a partial solution until it figures that there's no point going deeper as we would ultimately lead to a dead end.

In the backtracking approach, we maintain an 8x8 binary matrix for keeping track of safe cells (by eliminating the unsafe cells, those that are likely to be attacked) and update it each time we place a new queen. However, it required O(n^2) time to check safe cell and update the queen.

In the 8 queens problem, we ensure the following:

1. No two queens share a row
2. No two queens share a column
3. No two queens share the same left diagnol
4. No two queens share the same right diagnol

We already ensure that the queens do not share the same column by the way we fill out our auxiliary matrix (column by column). Hence, only the left out 3 conditions are left out to be satisfied.

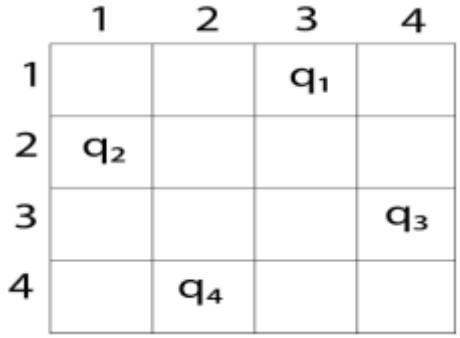
Applying the backtracking approach:

Now, we place queen q1 in the very first acceptable position (1, 1). Next, we put queen q2 so that both these queens do not attack each other. We find that if we place q2 in column 1 and 2, then the dead end is encountered.

Thus the first acceptable position for q2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q3' safely. So we backtrack one step and place the queen 'q2' in (2, 4), the next best possible solution.

Then we obtain the position for placing 'q3' which is (3, 2). But later this position also leads to a dead end, and no place is found where 'q4' can be placed safely. Then we have to backtrack till 'q1' and place it to (1, 2) and then all other queens are placed safely by moving q2 to (2, 4), q3 to (3, 1) and q4 to (4, 3).

That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem. For another possible solution, the whole method is repeated for all partial solutions. The other solutions for 4 -queens problems is (3, 1, 4, 2) i.e.



The implicit tree for 4 - queen problem for a solution (2, 4, 1, 3) is as follows

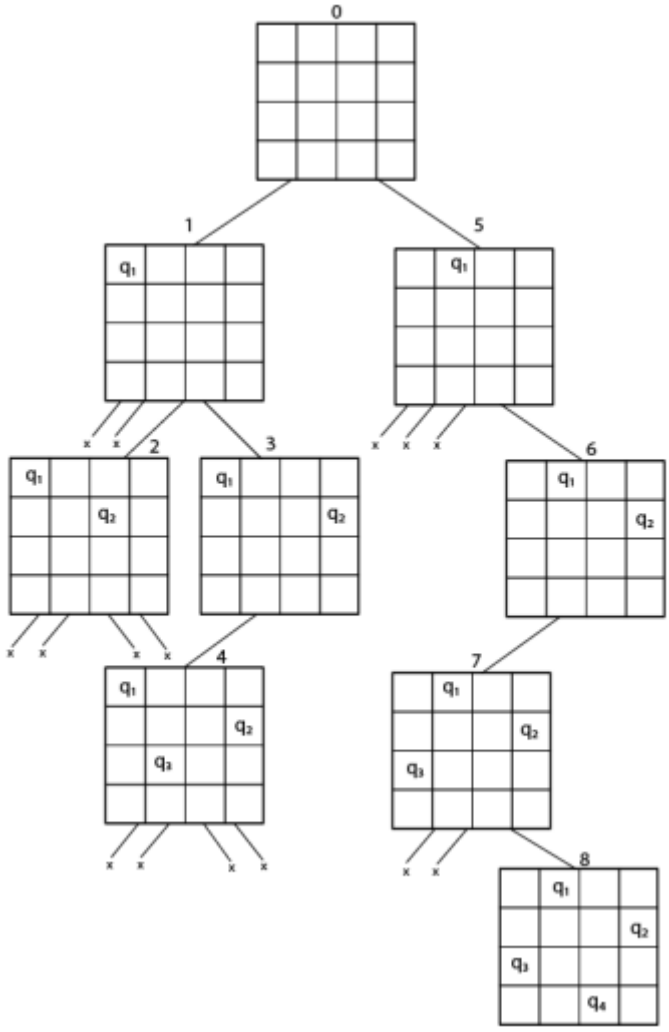
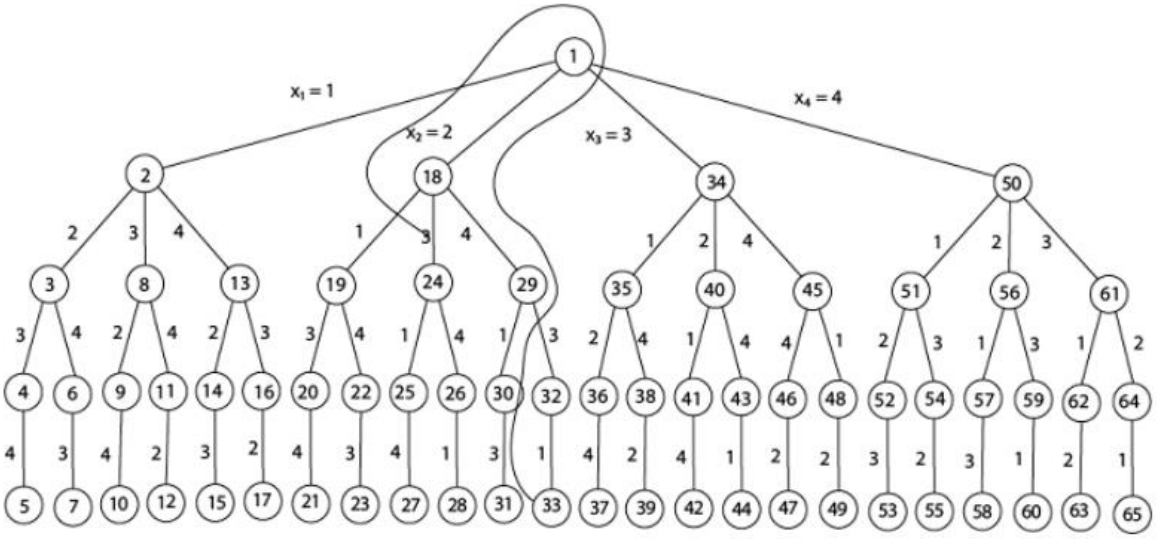


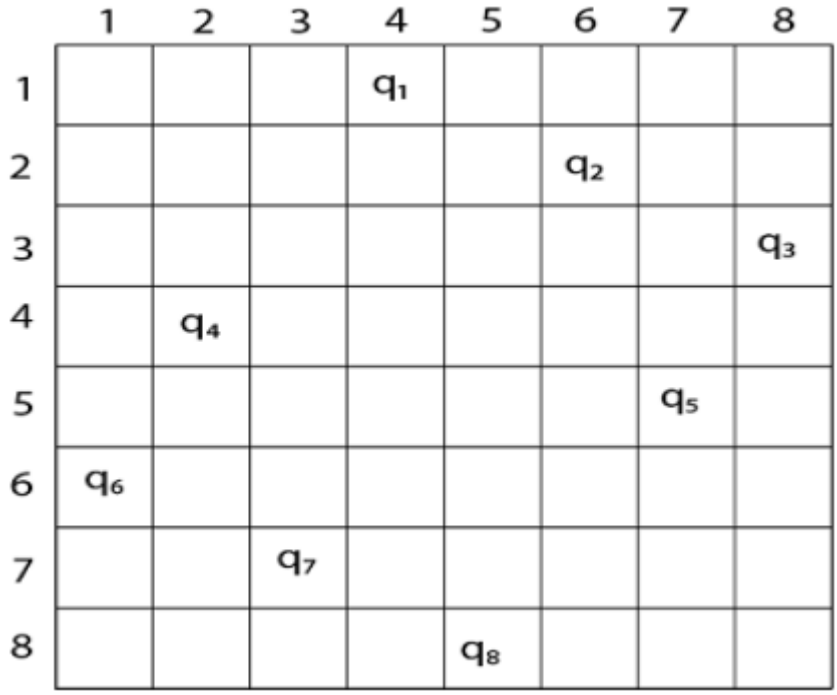
Fig shows the complete state space for 4 - queens problem. But we can use backtracking method to generate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.



4 - Queens solution space with nodes numbered in DFS

It can be seen that all the solutions to the 4 queens problem can be represented as 4 - tuples (x1, x2, x3, x4) where xi represents the column on which queen "qi" is placed.

One possible solution for 8 queens problem is shown in fig:



1. Thus, the solution for 8 -queen problem for (4, 6, 8, 2, 7, 1, 3, 5).
2. If two queens are placed at position (i, j) and (k, l).
3. Then they are on same diagonal only if (i - j) = k - l or i + j = k + l.
4. The first equation implies that j - l = i - k.
5. The second equation implies that j - l = k - i.
6. Therefore, two queens lie on the duplicate diagonal if and only if |j-l|=|i-k|

Applying the branch and bound approach:

The branch and bound approach suggets that we create a partial solution and use it to ascertain whether we need to continue in a particular direction or not. For this problem, we create 3 arrays to check for conditions 1,3 and 4. The boolean arrays tell which rows and

diagnols are already occupied. To achieve this, we need a numbering system to specify which queen is placed.

The indexes on these arrays would help us know which queen we are analyzing. Preprocessing - create two NxN matrices, one for top-left to bottom-right diagonal, and other for top-right to bottom-left diagonal. We need to fill these in such a way that two queens sharing same top-left_bottom right diagonal will have same value in slashDiagonal and two queens sharing same top-right_bottom-left diagonal will have same value in BackSlashDiagnol.

slashDiagnol(row)(col) = row + col

backSlashDiagnol(row)(col) = row - col + (N-1) { N = 8 }

{ we added (N-1) as we do not need negative values in backSlashDiagnol }

7	6	5	4	3	2	1	0
8	7	6	5	4	3	2	1
9	8	7	6	5	4	3	2
10	9	8	7	6	5	4	3
11	10	9	8	7	6	5	4
12	11	10	9	8	7	6	5
13	12	11	10	9	8	7	6
14	13	12	11	10	9	8	7

slash diagnol[row][col] = row + col

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14

backslash diagnol[row][col] = row-col+(N-1)

For placing a queen i on row j, check the following:

1. Whether row 'j' is used or not
2. Whether slashDiagnol 'i+j' is used or not
3. Whether backSlashDiagnol 'i-j+7' is used or not

If the answer to any one of the following is true, we try another location for queen i on row j, mark the row and diagnols; and recur for queen i+1.

Application:

The CSP, Backtracking and Branch & Bound technique are used in many applications like Map coloring, optimization problems and network programing.

Input:

Board with N x N size.

Output:

The algorithm places all the N Queens in N different columns with the given constraints.

Conclusion:

The N queen problem is implemented using python language with two techniques called Backtracking and branch & bound.