**Backtracking & N queens**

**What is backtracking?**

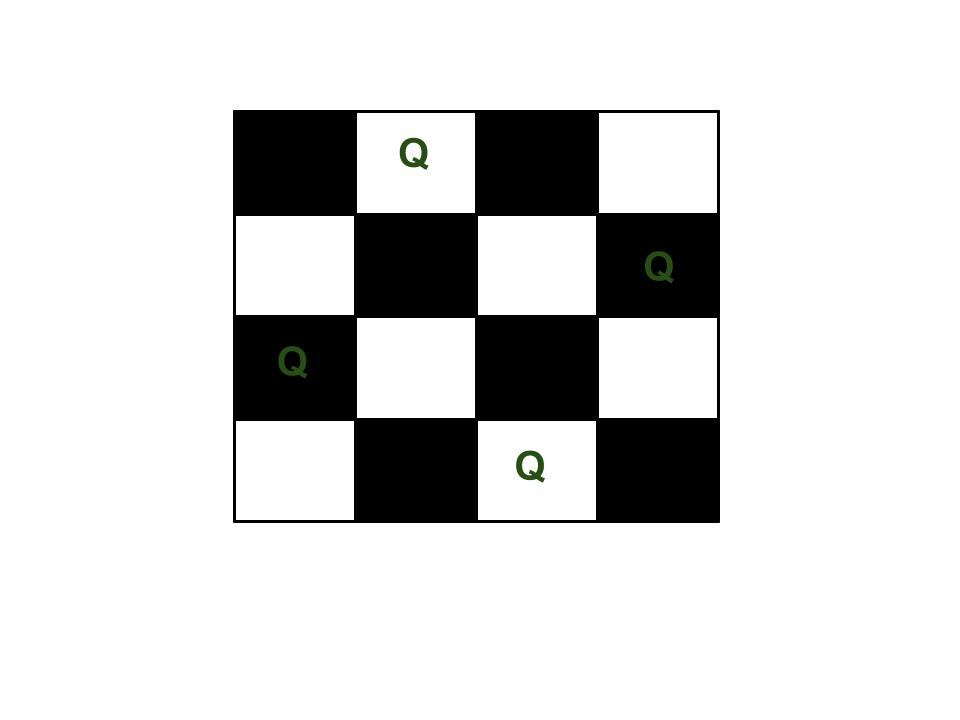
Backtracking is finding the solution of a problem whereby the solution depends on the previous steps taken. For example, in a maze problem, the solution depends on all the steps you take one-by-one. If any of those steps is wrong, then it will not lead us to the solution. In a maze problem, we first choose a path and continue moving along it. But once we understand that the particular path is incorrect, then we just come back and change it. This is what backtracking basically is.

In backtracking, we first take a step and then we see if this step taken is correct or not i.e., whether it will give a correct answer or not. And if it doesn’t, then we just come back and change our first step. In general, this is accomplished by recursion. Thus, in backtracking, we first start with a partial sub-solution of the problem (which may or may not lead us to the solution) and then check if we can proceed further with this sub-solution or not. If not, then we just come back and change it.

Thus, the general steps of backtracking are:

* start with a sub-solution
* check if this sub-solution will lead to the solution or not
* If not, then come back and change the sub-solution and continue again

**N queens on NxN chessboard**

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.[](https://media.geeksforgeeks.org/wp-content/uploads/N_Queen_Problem.jpg)

The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example, following is the output matrix for above 4 queen solution.

{ 0, 1, 0, 0}

{ 0, 0, 0, 1}

{ 1, 0, 0, 0}

{ 0, 0, 1, 0}

**Naive Algorithm**

Generate all possible configurations of queens on board and print a configuration that satisfies the given constraints.

while there are untried configurations

{

generate the next configuration

if queens don't attack in this configuration then

{

print this configuration;

}

}

**Backtracking Algorithm**

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

1) Start in the leftmost column

2) If all queens are placed

return true

3) Try all rows in the current column.

Do following for every tried row.

a) If the queen can be placed safely in this row

then mark this [row, column] as part of the

solution and recursively check if placing

queen here leads to a solution.

b) If placing the queen in [row, column] leads to

a solution then return true.

c) If placing queen doesn't lead to a solution then

unmark this [row, column] (Backtrack) and go to

step (a) to try other rows.

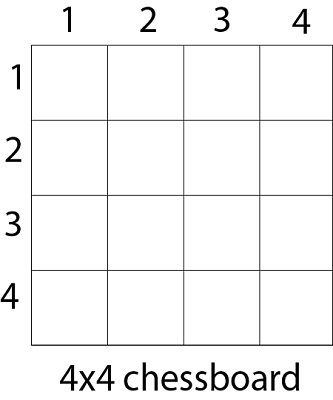
3) If all rows have been tried and nothing worked,

return false to trigger backtracking.

N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.

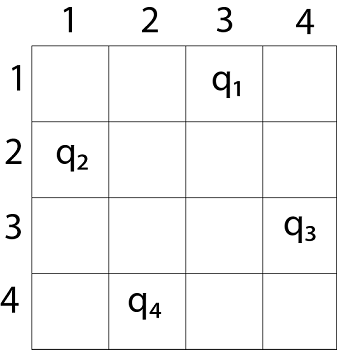
It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - queens problem.

Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.



Since, we have to place 4 queens such as q1 q2 q3 and q4 on the chessboard, such that no two queens attack each other. In such a conditional each queen must be placed on a different row, i.e., we put queen "i" on row "i."

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:

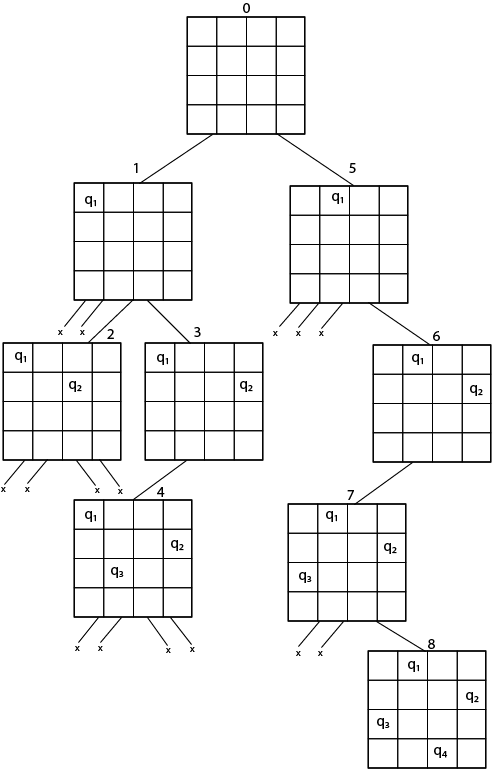
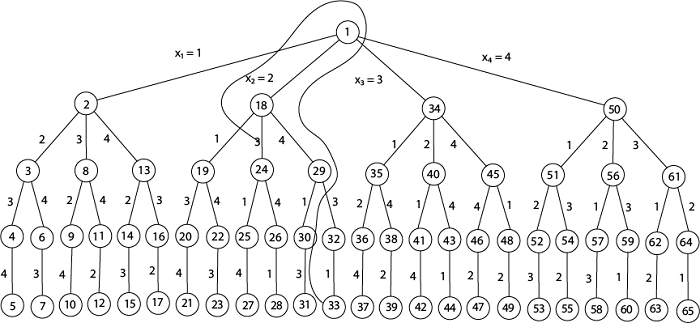


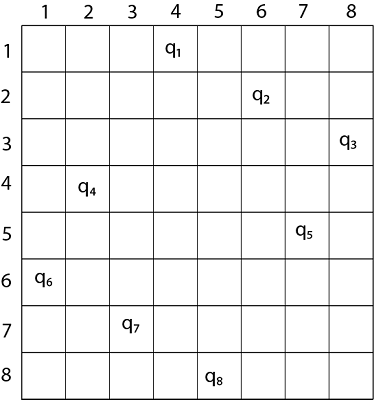
Fig shows the complete state space for 4 - queens problem. But we can use backtracking method to generate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.



**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|

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. [https://www.codesdope.com/blog/article/backtracking-explanation-and-n-queens-problem/#:~:text=One%20of%20the%20most%20common,attempted%20in%20a%20similar%20way.](https://www.codesdope.com/blog/article/backtracking-explanation-and-n-queens-problem/" \l ":~:text=One%20of%20the%20most%20common,attempted%20in%20a%20similar%20way.)
2. <https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/>
3. <https://www.javatpoint.com/n-queens-problems>

**Lecture Video:**

1. <https://youtu.be/0DeznFqrgAI>

**Online Notes:**

1. <http://vssut.ac.in/lecture_notes/lecture1428551222.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.

**In addition: PPT can be also be given.**