# N – Queen problem

# the project problem:

# N – Queen problem by Backtracking algorithm

Course: **Artificial Intelligence.**

Date: 0**3 / 06 / 2022**

Team Member:   
1- Mohamed Abdelmoniem Mohamed Sayed.

(fourth year in bioinformatics 2018).

Table of Contents :

**1 Introduction/Executive Summary …………………3** 1.1 What Is N-Queen problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3  
 1.2 Solve problem . . . . . . . . . . . . . . . ……………………………….. . . 4  
 1.3 Algorithm used is Backtracking……………………………………. 6  
**2 Methodology ………………………………………….7** 2.1 Describe the main algorithms. . . . . . . . ……... . . . . . . . . . . . . . . . . . 7

2.2 algorithmic steps. . . . . . . . . . . . . . . ………………………………… 9  
 2.3 Pseudo Code. . . . . . . . . . . . . . . . ………………………... . . . . . . . . 10  
 2.4 Flowchart:. . . . . . . . . . . . . . . . . . . . . . . . . . ………………………… 11

2.5 how algorithm can solve the problem . …………………….. . . . . . . . 13

2.6 Complexity for code……………………………………..…………….13

**3 Experimental Simulation …………………………….14**

3.1 the programming language……………………………………………..14

3.2 the primary function …………………………………………………...14

3.3 the test cases………………………………………………………….…15

3.4 Illustration ……………………………………..………………………16

**4 Results ………………………………………………….18**

**5 Conclusions ……………………………………………..20**

**6 References ……………………………………………….21**

**7 Appendix …………………………………………………22**

**Introduction/Executive Summary:**

N-Queen Problem :

The N queen problem is one of the challenging concepts in computer science, the principle of the n-queen problem as follows: N is the number of queens that is placed on the n\*n chessboard. The queens cannot be interacting with each other. Each queen has a unique path to travel. a solution requires that no two queens share the same row, column, or diagonal. All possible solutions can be found for each level. Most of these solutions are based on a particular prescription to allocate queens on a chessboard, the factual conclusion of petite-size problems displayed.

The problem was first posed in the mid-19th century. In the modern era, it is often used as an example problem for various computer programming techniques.

8 – queen :

The eight queens puzzle is a special case of the more general n queens problem of placing n non-attacking queens on an n × n chessboard. Solutions exist for all natural numbers n with the exception of n = 2 and n = 3. Although the exact number of solutions is only known for n ≤ 27, the asymptotic growth rate of the number of solutions is (0.143 n) ^ n .

Solve problem :

Analyze the consider problem solvers and algorithms :

1. Suppose you have 8 chess queens and a chess board of Size 8\*8.

2. Can the queens be placed on the board so that no two queens are attacking each other?

3. Two queens are not allowed in the same row.

4. Two queens are not allowed in the same row, or in the same column.

5. Two queens are not allowed in the same row, or in the same column, or along the same diagonal.

6. The number of queens and the size of the board can vary.

7. It seems hard to generate one valid placement.

8. But it is easy to check whether a placement is valid or not.

9. We will write a program(backtracking alogorithm) which tries to find a way to place N queens on an N x N chess board.

10. The program uses a stack to keep track of where each queen is placed.

11. Each time the program decides to place a queen on the board, the position of the new queen is stored in

a record which is placed in the stack.

12. We also have an integer variable to keep track of how many rows have been filled so far.

13. Each time we try to place a new queen in the next row, we start by placing the queen in the first column.

14. If there is a conflict with another queen, then we shift the new queen to the next column.

15. If another conflict occurs, the queen is shifted rightward again.

16. When there are no conflicts, we stop and add one to the value of filled.

17. Let's look at the third row. The first position we try has a conflict.

18. So we shift to column 2. But another conflict arises.

19. Then we shift to the third column. Yet another conflict arises.

20. We shift to column 4. There's still a conflict in column 4, so we try to shift rightward again.

21. When we run out of room in a row: pop the stack, reduce filled by 1 and continue working on the previous

row.

22. Now we continue working on row 2, shifting the queen to the right.

23. This position has no conflicts, so we can increase filled by 1, and move to row 3.

24. In row 3, we start again at the first column.

Algorithm used is Backtracking :

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

For example, consider the SudoKo solving Problem, we try filling digits one by one. Whenever we find that current digit cannot lead to a solution, we remove it (backtrack) and try next digit. This is better than naive approach (generating all possible combinations of digits and then trying every combination one by one) as it drops a set of permutations whenever it backtracks.

**Methodology:**

Describe the main algorithms:

Backtracking: Backtracking is a form of recursion. The usual scenario is that you are faced with a number of

options, and you must choose one of these. After you make your choice you will get a new set of options; just

what set of options you get depends on what choice you made. This procedure is repeated over and over until you

reach a final state. If you made a good sequence of choices, your final state is a goal state; if you didn't, it isn't.

Conceptually, you start at the root of a tree; the tree probably has some good leaves and some bad leaves, though

it may be that the leaves are all good or all bad. You want to get to a good leaf. At each node, beginning with the

root, you choose one of its children to move to, and you keep this up until you get to a leaf.

Backtracking is a technique used to solve problems with a large search space, by systematically trying and

eliminating possibilities. Backtracking is a methodical way of trying out various sequences of decisions, until you

find one that “works”. When we carry out backtracking, an easy way to visualize what is going on is a tree that

shows all the different possibilities that have been tried. On the board we will show a visual representation of

solving the 4 Queens problem (placing 4 queens on a 4x4 board where no two attack one another). The neat thing

about coding up backtracking is that it can be done recursively, without having to do all the bookkeeping at once.

Instead, the stack or recursive calls does most of the book keeping track of which queens we've placed, and which

combinations we've tried so far, etc. sets are used as input to enhance the

Backtracking technique by removing threatening cells.

algorithmic steps:

1. Place the queens column wise, start from the left most column

2. If all queens are placed.

1. Return true and print the solution matrix.

3. Else

1. Try all the rows in the current column.

2. Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to

solve the rest of the problem recursively.

3. If placing the queen in above step leads to the solution return true.

4. If placing the queen in above step does not lead to the solution, BACKTRACK, mark the current cell

in solution matrix as 0 and return false.

4. If all the rows are tried and nothing worked, return false and print NO SOLUTION

Pseudo Code:

Boolean solve (Node n)

{

if n is a leaf node

{

if the leaf is a goal node, return true

else return false

}

else

{

for each child c of n

{

if solve(c) succeeds, return true

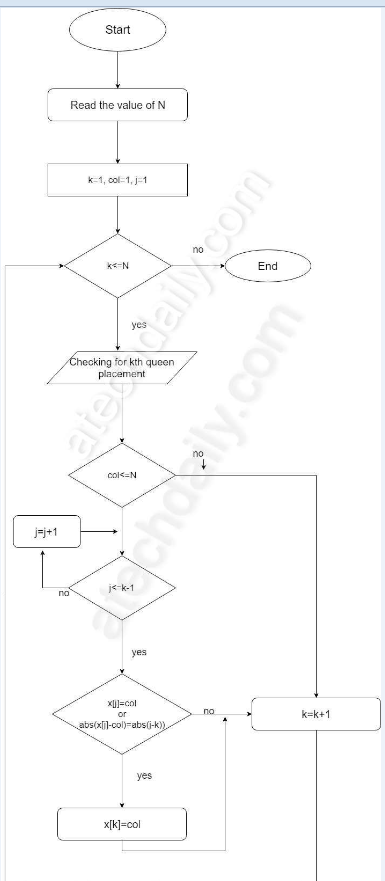
}

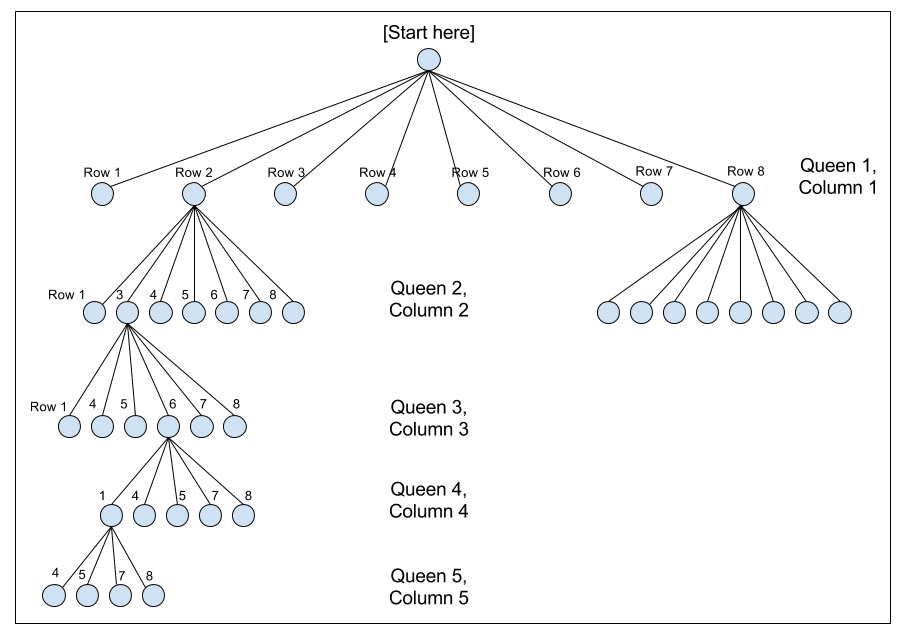
Return false

}

}

Flowchart:





how algorithm can solve the problem:

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.

Complexity for code:

Time Complexity: O(N!)

Memory Space: O(N)

Experimental Simulation:

**the programming language is used in the project is:** c++.

c++ language is suitable for solve this problem with Backtracking algorithm.

the primary function used to implement the Backtracking algorithm are:

1- check Function for check this board is true or false.

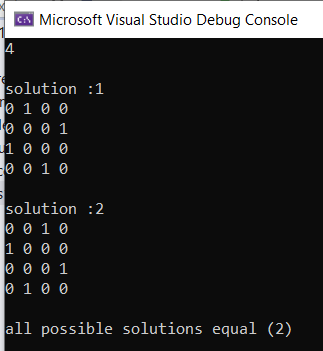
2- fast Function to fast the code

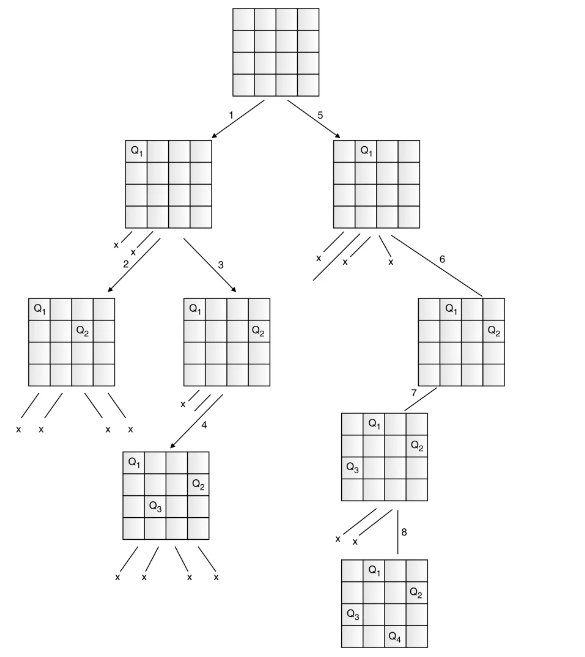
3- main Function

the test cases:

example n = 4

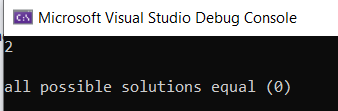
answer = 2





example n = 2

answer = 0



Because all possible solution will be diagonal or at the same row or column

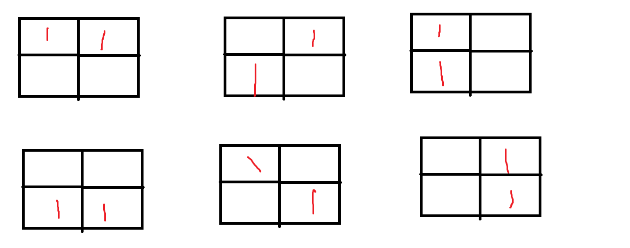


Illustration:

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.

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

return false to trigger backtracking.

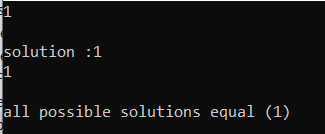
**Results :**

|  |  |
| --- | --- |
| input | output |
| 1 | 1 |
| 2 | 0 |
| 3 | 0 |
| 4 | 2 |
| 5 | 10 |
| 6 | 4 |
| 7 | 40 |
| 8 | 92 |
| 9 | 352 |
| 10 | 724 |
| 11 | 2680 |
| 12 | 14200 |
| 13 | 73713 |

Console:

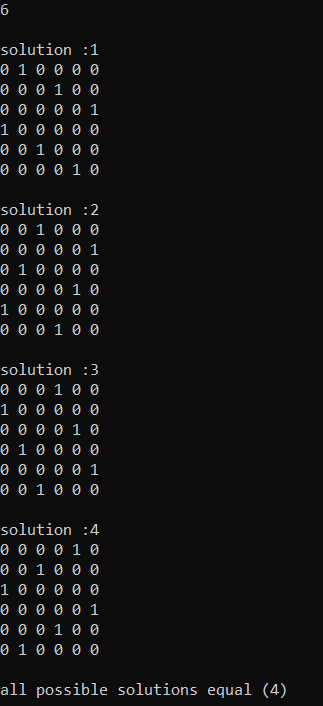
N=1

Answer = 1



N = 6

Answer = 4



**Conclusions:**

the efficiency of the traditional backtracking algorithm maybe improved by the use of a hybrid approach taking advantage of sets to reduce the number of trials and error attempts. Time taken to solve the n-queen problem in the backtracking is more than that of the Tuned hybrid technique. Space taken to solve the n-queen problem in the backtracking is more than that of the Tuned hybrid technique. Complexity Analysis can be improved using different algorithms and that approach will be applied on the one of the applications of the N-Queen Problem to obtain the fast and better solution. Complexity Analysis can be based in the time, space, convergence-rate and conflict-minimization.

**References:**

1. <https://en.wikipedia.org/wiki/Eight_queens_puzzle>
2. <https://www.geeksforgeeks.org/backtracking-introduction/>
3. <https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/>
4. <https://www.dotnetoffice.com/2022/02/solving-n-queens-problem-in-c-print-all.html>
5. <https://codecrucks.com/n-queen-problem/>
6. <https://atechdaily.com/posts/Flowchart-and-Algorithm-for-N-Queen-Problem>
7. <http://www.ijesrt.com/issues%20pdf%20file/Archive-2017/February-217/10.pdf>
8. <https://www.researchgate.net/publication/333661204_Survey_on_N-Queen_Problem_with_Genetic_Algorithm>

**Appendix:**

Source code:

[**https://ideone.com/KNXZY5**](https://ideone.com/KNXZY5)

another idea:

[**https://ideone.com/t8IfVd**](https://ideone.com/t8IfVd)