BackTracking

BACK TRACKING

- ✓ **Backtracking** is a general algorithm for finding all (or some) solutions to some computational problem, that *incrementally builds candidates to the solutions*, and abandons each partial candidate '*c*' ("backtracks") as soon as it determines that '*c*' cannot possibly be completed to a valid solution.

- ✓ Backtracking is an important tool for solving constraint satisfaction problems, such as *crosswords, verbal arithmetic, Sudoku, and many other puzzles.*

WHAT IS 8 QUEEN PROBLEM?

- ✓ The **eight queens puzzle** is the problem of placing eight chess queens on an 8 8 chessboard so that no two queens attack each other.
- ✓ Thus, a solution requires that no two queens share the same row, column, or diagonal.
- ✓ The eight queens puzzle is an example of the more general ***n*-queens problem** of placing *n* queens on an *n* *n* chessboard, where solutions exist for all natural numbers *n* with the exception of *1, 2 and 3*.
- ✓ The solution possibilities are discovered only up to *23 queen.*

```
START
1. begin from the leftmost column
2. if all the queens are placed,
   return true/ print configuration
3. check for all rows in the current column
   a) if queen placed safely, mark row and column; and
      recursively check if we approach in the current
      configuration, do we obtain a solution or not
   b) if placing yields a solution, return true
   c) if placing does not yield a solution, unmark and
      try other rows
4. if all rows tried and solution not obtained, return
   false and backtrack
END
```

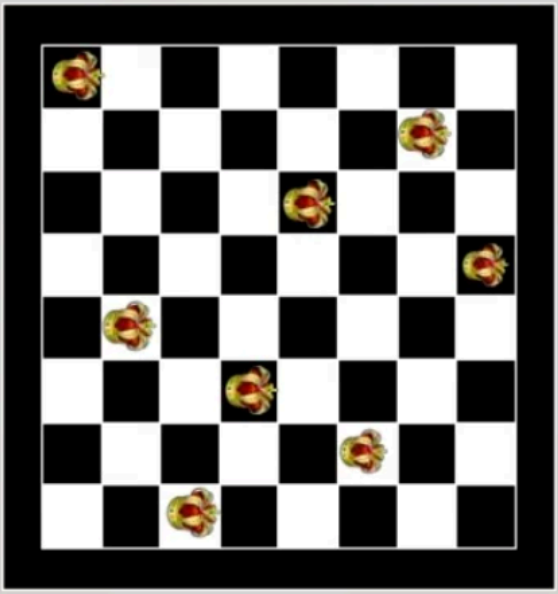
Time Complexity Analysis

- the isPossible method takes O(n) time
- for each invocation of loop in nQueenHelper, it runs for O(n) time
- the isPossible condition is present in the loop and also calls nQueenHelper which is recursive
- adding this up, the recurrence relation is:

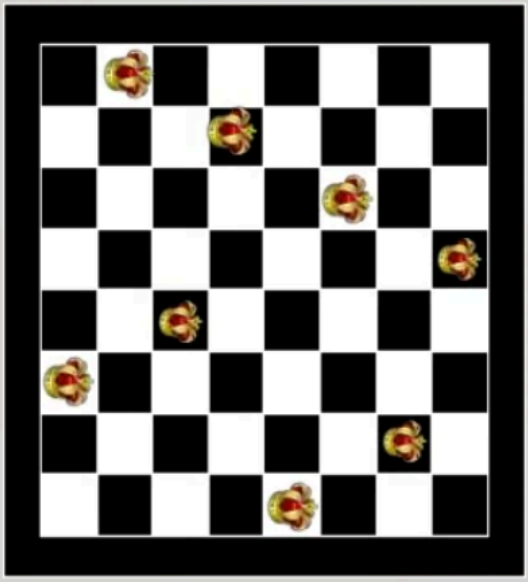
- $T(n) = O(n^2) + n * T(n-1)$

- solving the above recurrence by iteration or recursion tree,
- the time complexity of the nQueen problem is = O(N!)

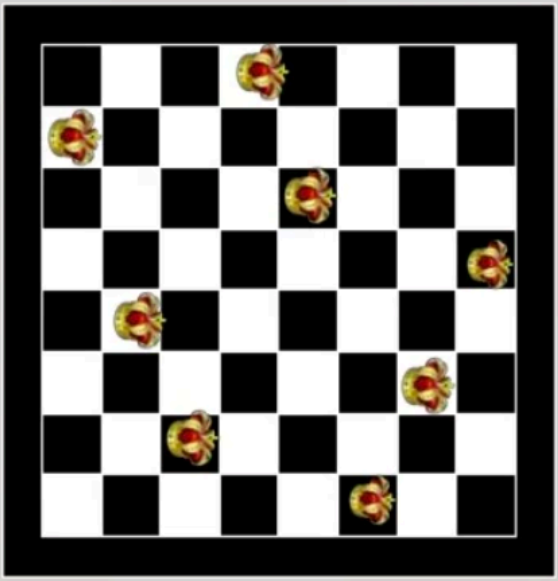
UNIQUE SOLUTION 3



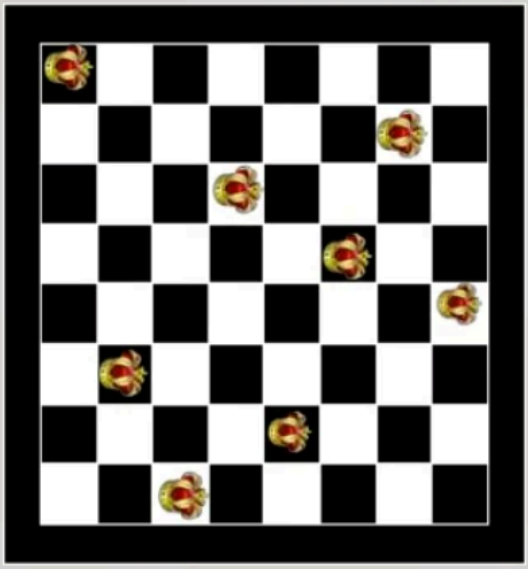
UNIQUE SOLUTION 1



UNIQUE SOLUTION 4



UNIQUE SOLUTION 2



Graph Colouring Problem

Naive Approach: Generate all possible configurations of colors. Since each node can be coloured using any of the m available colours, the total number of colour configurations possible are m^V.
After generating a configuration of colour, check if the adjacent vertices have the same colour or not. If the conditions are met, print the combination and break the loop.

Algorithm:

1. Create a recursive function that takes current index, number of vertices and output color array.
2. If the current index is equal to number of vertices. Check if the output color configuration is safe, i.e check if the adjacent vertices do not have same color. If the conditions are met, print the configuration and break.
3. Assign a color to a vertex (1 to m).
4. For every assigned color recursively call the function with next index and number of vertices
5. If any recursive function returns true break the loop and returns true.

Graph Colouring Problem

- Time Complexity: **$O(m^V)$**.
- Space Complexity: **O(V)** which is for storing the output array.