

8 Queens problem

Dr. Bibhudatta Sahoo

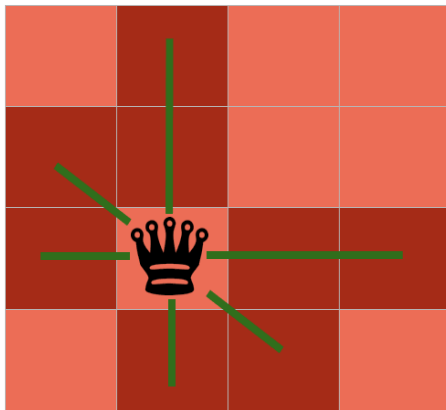
Communication & Computing Group

Department of CSE, NIT Rourkela

Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

N-Queens Problem

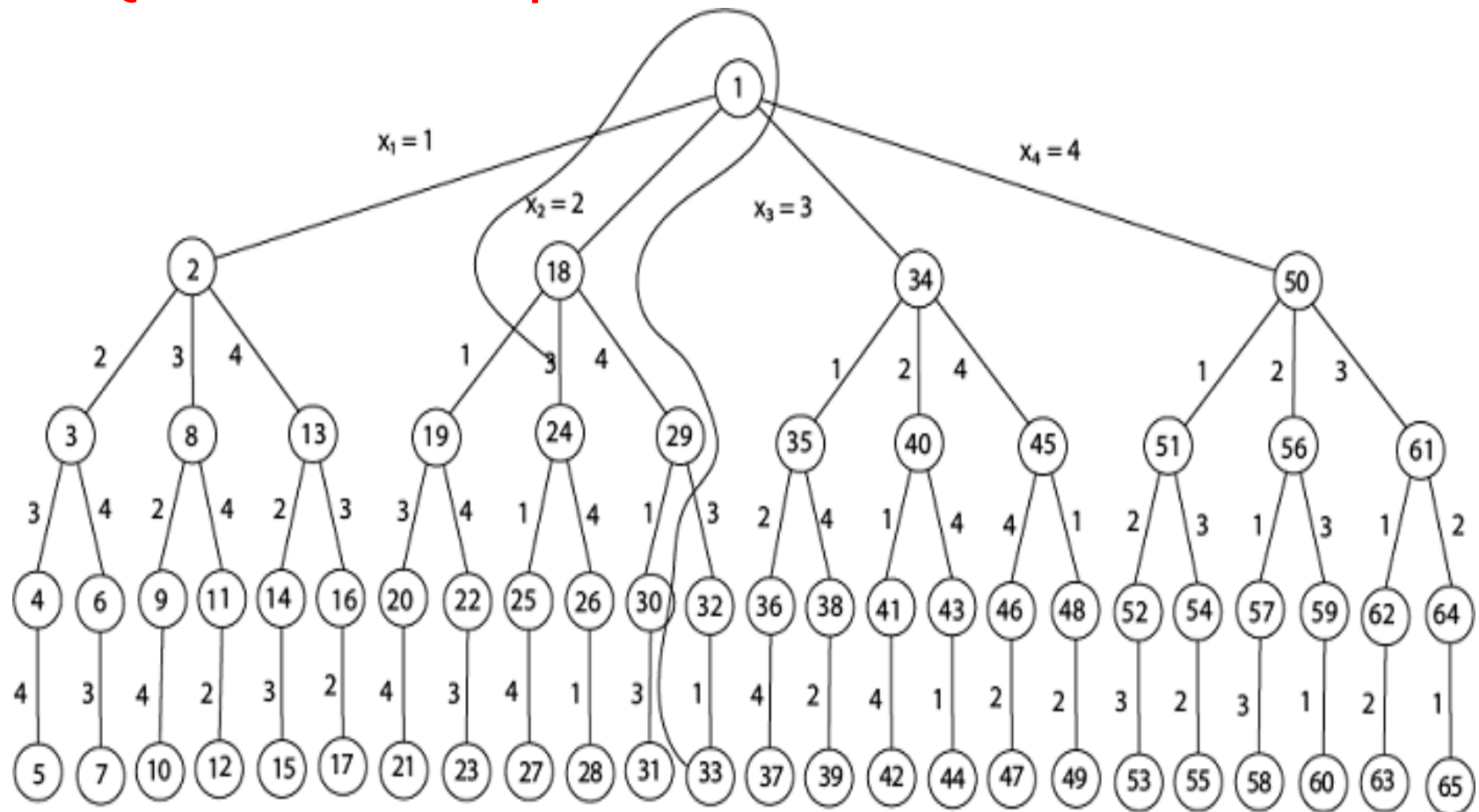
- **N - Queens problem** is to place **n - queens** in such a manner on an $n \times n$ chessboard that no **queens** attack each other by being in the same row, column or diagonal.
- It can be seen that for $n = 1$, the **problem** has a trivial solution, and no solution exists for $n = 2$ and $n = 3$. So first we will consider the 4 queens problem and then generate it to n - queens problem.
- Given a 4×4 chessboard and number the rows and column of the chessboard 1 through 4.



Cells attacked by the queen

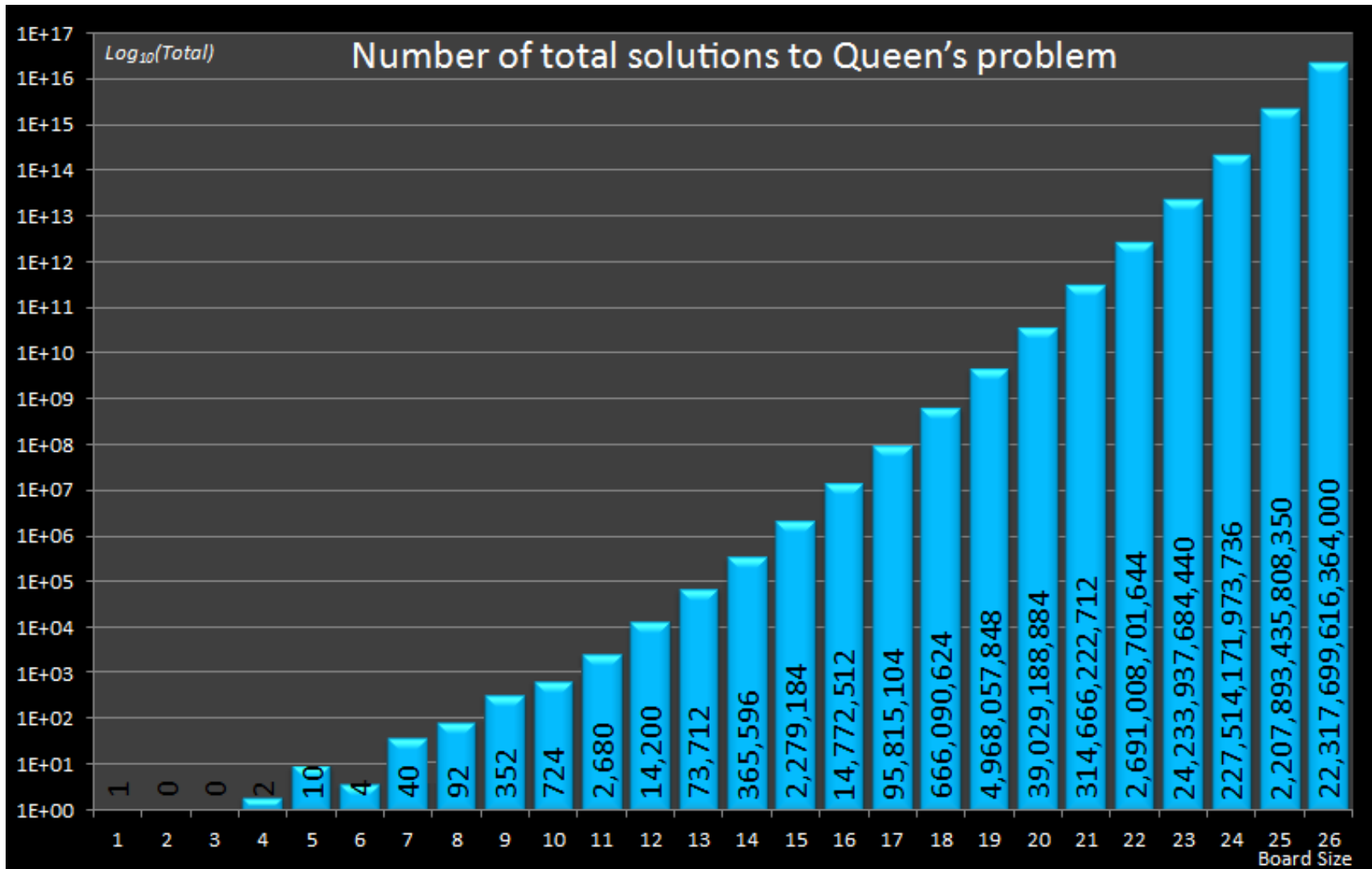
	1	2	3	4
1			q ₁	
2	q ₂			
3				q ₃
4		q ₄		

4- - Queens solution space with nodes numbered in DFS



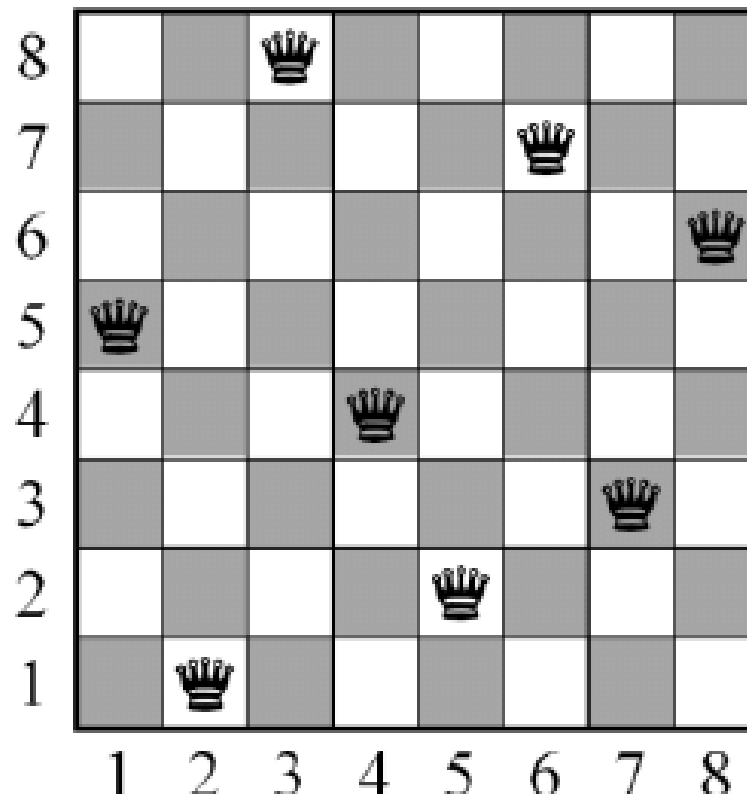
all the solutions to the 4 queens problem can be represented as 4 - tuples (x_1, x_2, x_3, x_4) where x_i represents the column on which queen " q_i " is placed.

Number of total solutions to Queen's problem



The eight queens

- The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other; thus, a **solution** requires that no two queens share the same row, column, or diagonal.



8- queen problem

- The problem of finding all solutions to the 8-queens problem can be quite computationally expensive, as there are **4,426,165,368** (i.e., ${}_{64}C_8$) possible arrangements of eight queens on an 8×8 board, but only **92 solutions**.
- **N Queens Completion is NP Complete.**
- The problem of putting eight **queens** on the chess board so as no **queen** attacks another is a solved problem, as is placing **n queens** on an $n \times n$ board.
- However if you place some **queens** on the board and ask for a completion then the problem is **NP complete**

8- queen problem

- For 4-Queens there are 256 different configurations.
- For 8-Queens there are 16,777,216 configurations.
- For 16-Queens there are 18,446,744,073,709,551,616 configurations.
- This would take about 12,000 years on a fast modern machine.
- In general we have n^n configurations for NQueens.

8- queen problem

- Let us consider, $N = 8$. Then 8-Queens Problem is to place eight queens on an 8×8 chessboard so that no two “attack”, that is, no two of them are on the same row, column, or diagonal
- Let us number the rows and columns of the chessboard 1 through 8
- The queens can also be numbered 1 through 8.
- Since each queen must be on a different row, we can without loss of generality assume **queen i** is to be placed on **row i**.
- All solutions to the 8-queens problem can therefore be represented as 8-tuples (x_1, \dots, x_8) , where x_i is the **column** on which **queen i** is placed.

8- queen problem

- The **explicit constraints** using this formulation are $S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $1 \leq i \leq 8$.
- Therefore the solution space consists of 8^8 8-tuples.
- The **implicit constraints** for this problem are that no two x_i 's can be the same (i.e. all queens must be on different columns) and no two queens can be on the same diagonal.
- This realization reduces the size of the solution space from 8^8 tuples to $8!$ tuples.

8- queen problem

- The promising function must check whether two queens are in the same column or diagonal:
- Suppose two queens are placed at positions (i, j) and (k, l) Then:
- **Column Conflicts:** Two queens conflict if their x_i values are identical.
- **Diag 45 conflict:** Two queens i and j are on the same 45° diagonal if: $i - j = k - l$. This implies, $j - l = i - k$
- **Diag 135 conflict:** $i + j = k + l$. This implies, $j - l = k - i$

Therefore, two queens lie **on the same diagonal** if and only if: $|j - l| = |i - k|$, Where, j be the column of object in row i for the i^{th} queen and l be the column of object in row k for the k^{th} queen.

One solution to 8-queen Problem

column →

row ↓

	1	2	3	4	5	6	7	8
1				Q				
2						Q		
3								Q
4		Q						
5							Q	
6	Q							
7			Q					
8					Q			

(4, 6, 8, 2, 7, 1, 3, 5)

To check the diagonal clashes

To check the diagonal clashes, let us take the following tile configuration:

	*						
				*			
*							
							*
			*				
						*	
		*					
					*		

In this example, we have:

i	1	2	3	4	5	6	7	8
x_i	2	5	1	8	4	7	3	6

case whether the queens on
are conflicting or not. In this

case $(i, j) = (3, 1)$ and $(k, l) = (8, 6)$. Therefore:

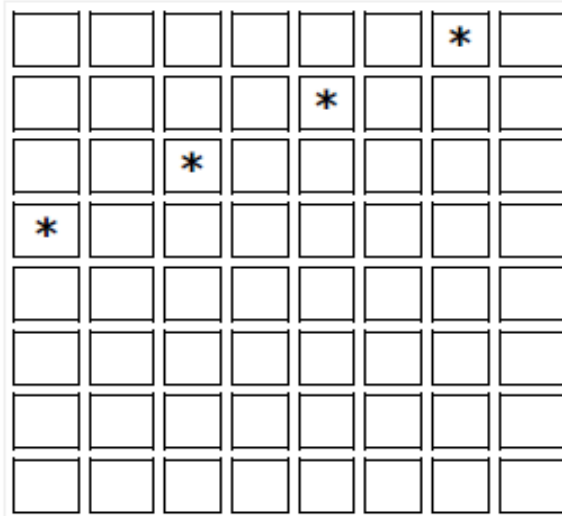
Let us consider for the
3rd row and 8th row

$$\begin{aligned}|j - l| &= |i - k| \Rightarrow |1 - 6| = |3 - 8| \\ &\Rightarrow 5 = 5\end{aligned}$$

In the above example we have, $|j - l| = |i - k|$, so the two queens are attacking.
This is not a solution.

Example

- Suppose we start with the feasible sequence 7, 5, 3, 1.

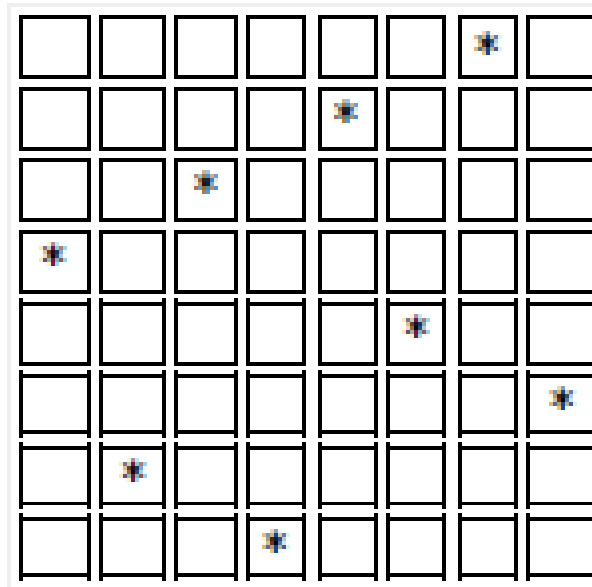


- Step 1: Add to the sequence the next number in the sequence 1, 2, . . . , 8 not yet used.
- Step 2: If this new sequence is feasible and has length 8 then STOP with a solution. If the new sequence is feasible and has length less than 8, repeat Step 1.
- Step 3: If the sequence is not feasible, then *backtrack* through the sequence until we find the *most recent* place at which we can exchange a value. Go back to Step 1.

1	2	3	4	5	6	7	8	Remarks
7	5	3	1					
7	5	3	1*	2*				$ j - i = 1 - 2 = 1$ $ i - k = 4 - 5 = 1$
7	5	3	1	4				
7*	5	3	1	4	2*			$ j - i = 7 - 2 = 5$ $ i - k = 1 - 6 = 5$
7	5	3*	1	4	6*			$ j - i = 3 - 6 = 3$ $ i - k = 3 - 6 = 3$
7	5	3	1	4	8			
7	5	3	1	4*	8	2*		$ j - i = 4 - 2 = 2$ $ i - k = 5 - 7 = 2$
7	5	3	1	4*	8	6*		$ j - i = 4 - 6 = 2$ $ i - k = 5 - 7 = 2$
7	5	3	1	4	8			Backtrack
7	5	3	1	4				Backtrack
7	5	3	1	6				
7*	5	3	1	6	2*			$ j - i = 1 - 2 = 1$ $ i - k = 7 - 6 = 1$
7	5	3	1	6	4			
7	5	3	1	6	4	2		
7	5	3*	1	6	4	2	8*	$ j - i = 3 - 8 = 5$ $ i - k = 3 - 8 = 5$
7	5	3	1	6	4	2		Backtrack
7	5	3	1	6	4			Backtrack
7	5	3	1	6	8			
7	5	3	1	6	8	2		
7	5	3	1	6	8	2	4	SOLUTION

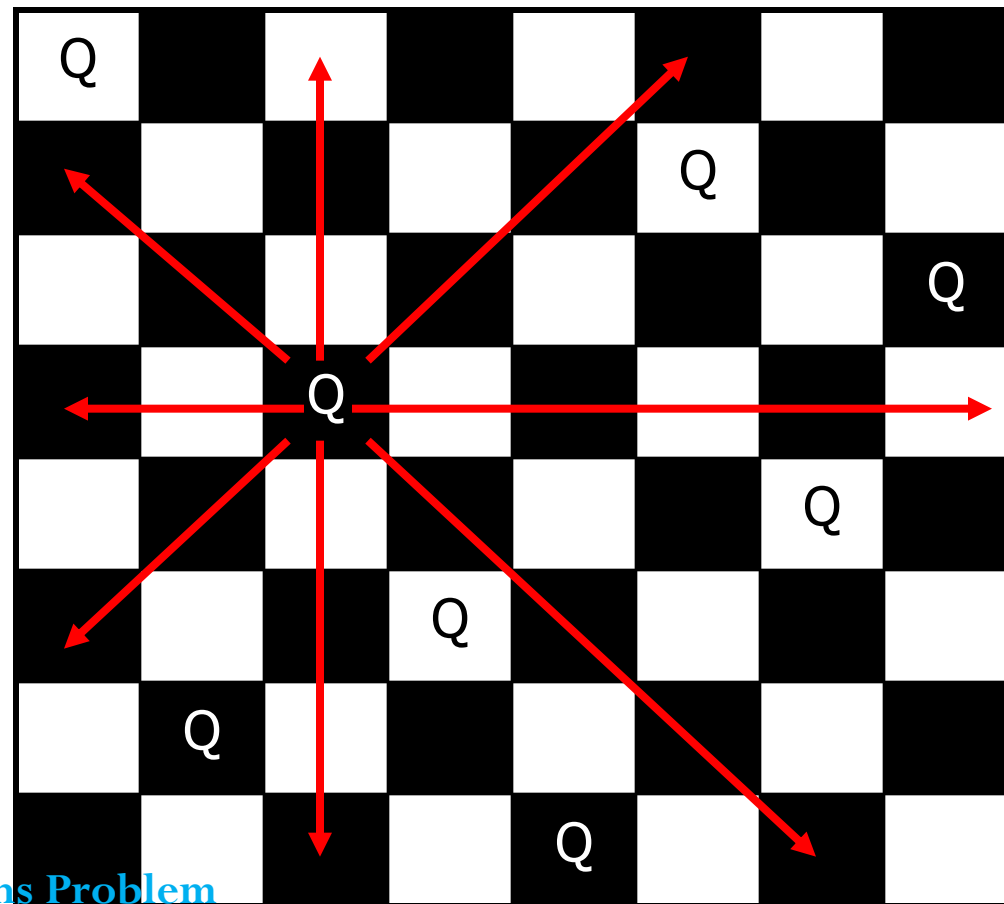
Solution to 8 Queens Problem

- On a chessboard, the **solution obtained** will look like:



The "8 Queens" problem

- Consider the problem of trying to place 8 queens on a chess board such that no queen can attack another queen.
- What are the "choices"?
- How do we "make" or "un-make" a choice?
- How do we know when to stop?

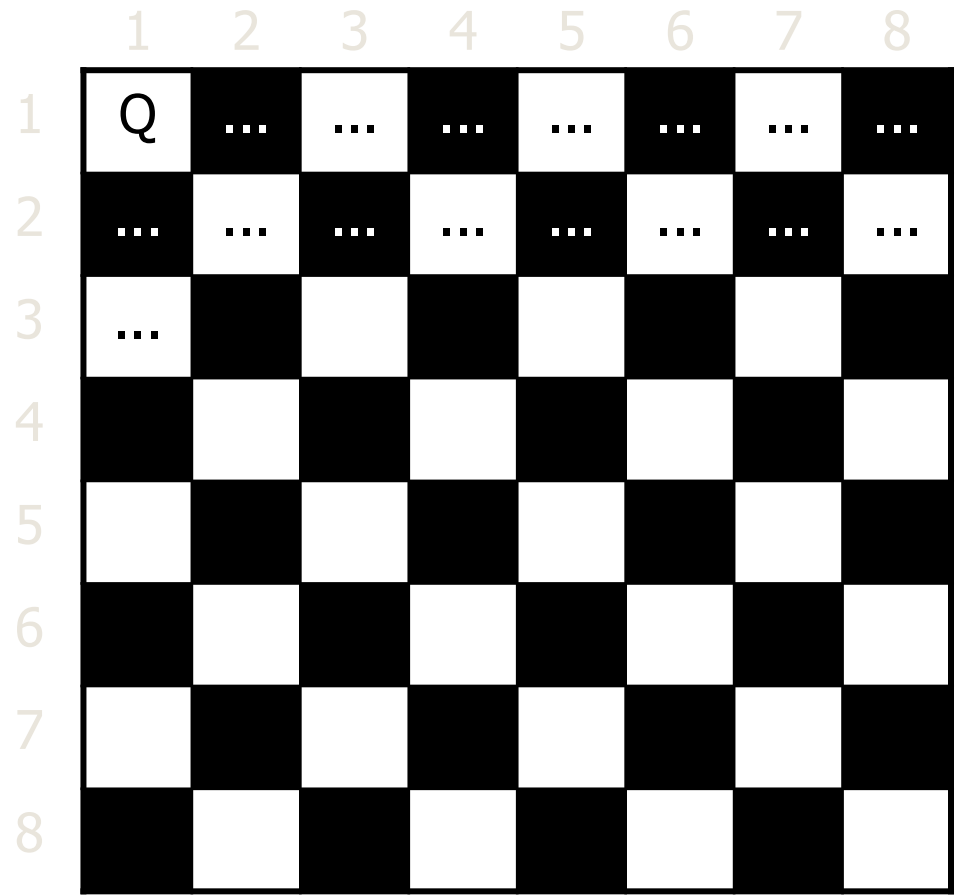


Backtracking algorithm design for 8-queen problem

- To solve this problem, we will make use of the **Backtracking algorithm**. The backtracking algorithm, in general checks all possible configurations and test whether the required result is obtained or not.
- For the given problem, we will explore all possible positions the queens can be relatively placed at. The solution will be correct when the number of placed queens = 8.
- The time complexity of this approach is **$O(n!)$** .
- **Input Format** - the number 8, which does not need to be read, but we will take an input number for the sake of generalization of the algorithm to an **$n \times n$** chessboard.
- **Output Format** - all matrices that constitute the possible solutions will contain the numbers 0(for empty cell) and 1(for a cell where queen is placed). Hence, the output is a set of binary matrices.

Naive algorithm

- for (each square on board):
 - Place a queen there.
 - Try to place the rest of the queens.
 - Un-place the queen.
- How large is the solution space for this algorithm?
 - $64 * 63 * 62 * \dots$



The "8 Queens" problem

First Idea: Consider every possible placement

- – How many placements are there?

$$\binom{n^2}{n} \Rightarrow \binom{64}{8} = 4,426,165,368$$

Second Solution Idea

- Don't place 2 queens in the same row.
- Now how many positions must be checked?

Represent a positioning as a vector $[x_1, \dots, x_8]$

Where each element is an integer 1, ..., 8.

$$n^n = 8^8 = 16,777,216$$

Better algorithm idea

- Observation: In a working solution, exactly 1 queen must appear in each row and in each column.
- Redefine a "choice" to be valid placement of a queen in a particular column.
- How large is the solution space now?
 - $8 * 8 * 8 * \dots$

	1	2	3	4	5	6	7	8
1	Q					
2						
3		Q	...					
4			...					
5			Q					
6								
7								
8								

Third Solution Idea

- Don't place 2 queens in the same row or in the same column.
- Generate all permutations of (1,2...8)
 - Now how many positions must be checked?

$$n! = 8! = 40,320$$

- We went from $C(n^2, n)$ to n^n to $n!$
 - And we're happy about it!
- We applied *explicit constraints* to shrink our search space.

Q			

let us first consider an empty chessboard and start by placing the first queen on cell `chessboard[0][0]`

Q			
		Q	

`chessboard[1][2]` is the only possible position where the second queen can be placed

Q			
		Q	
X	X	X	X

no queen can be placed further as queen 1 is in column 1, queen 2 is diagonally opposite to columns 2 and 3; and column 3 has queen 2 in it. Since number of queens is not 4, this is an infeasible solution and will not be printed

	Q		

the next iteration of our algorithm will begin with the second column and start placing the queens again

	Q		
			Q
Q			

queens 2 and 3 are easily placed onto the chessboard without creating the possibility of attacking one another.

	Q		
			Q
Q			
		Q	

the 4th queen has also been placed accordingly. Since the number of queens = 4, this solution will be printed.

The "8 Queens" problem

Two queens are placed at positions (i, j) and (k, l) .

They are on the same diagonal only if

$$i - j = k - l \text{ or } i + j = k + l$$

The first equation implies

$$j - l = i - k$$

The second implies

$$j - i = k - l$$

Two queens lie on the same diagonal iff

$$|j - l| = |i - k|$$

The "8 Queens" problem

For instance $P1=(8,1)$ and $P2=(1,8)$

So $i=8$, $j=1$ and $k=1$, $l=8$

$$i + j = k + l$$

$$8+1 = 1+8 = 9$$


$$j - l = k - i$$

$$1 - 8 = 1 - 8$$

Hence $P1$ and $P2$ are on the same diagonal


Algorithm 7.5 All solutions to the n-queens problem

```
1  Algorithm NQueens( $k, n$ )
2  // Using backtracking, this procedure prints all
3  // possible placements of  $n$  queens on an  $n \times n$ 
4  // chessboard so that they are nonattacking.
5  {
6      for  $i := 1$  to  $n$  do
7      {
8          if Place( $k, i$ ) then
9          {
10              $x[k] := i$ ;
11             if ( $k = n$ ) then write ( $x[1 : n]$ );
12             else NQueens( $k + 1, n$ );
13         }
14     }
15 }
```



Algorithm 7.4 Can a new queen be placed ?

```
1  Algorithm Place( $k, i$ )
2  // Returns true if a queen can be placed in  $k$ th row and
3  //  $i$ th column. Otherwise it returns false.  $x[ ]$  is a
4  // global array whose first  $(k - 1)$  values have been set.
5  // Abs( $r$ ) returns the absolute value of  $r$ .
6  {
7      for  $j := 1$  to  $k - 1$  do
8          if  $((x[j] = i) // \text{Two in the same column}$ 
9              or  $(\text{Abs}(x[j] - i) = \text{Abs}(j - k)))$ 
10             // or in the same diagonal
11             then return false;
12      return true;
13  }
```



Recursive backtracking algorithm design

```
1  Algorithm Backtrack( $k$ )
2  // This schema describes the backtracking process using
3  // recursion. On entering, the first  $k - 1$  values
4  //  $x[1], x[2], \dots, x[k - 1]$  of the solution vector
5  //  $x[1 : n]$  have been assigned.  $x[ ]$  and  $n$  are global.
6  {
7      for (each  $x[k] \in T(x[1], \dots, x[k - 1])$ ) do
8          {
9              if ( $B_k(x[1], x[2], \dots, x[k]) \neq 0$ ) then
10                 {
11                     if ( $x[1], x[2], \dots, x[k]$  is a path to an answer node)
12                         then write ( $x[1 : k]$ );
13                     if ( $k < n$ ) then Backtrack( $k + 1$ );
14                 }
15             }
16 }
```

```
1  Algorithm NQueens( $k, n$ )
2  // Using backtracking, this procedure prints all
3  // possible placements of  $n$  queens on an  $n \times n$ 
4  // chessboard so that they are nonattacking.
5  {
6      for  $i := 1$  to  $n$  do
7          {
8              if Place( $k, i$ ) then
9                  {
10                      $x[k] := i$ ;
11                     if ( $k = n$ ) then write ( $x[1 : n]$ );
12                     else NQueens( $k + 1, n$ );
13                 }
14             }
15 }
```

Eight queens problem – Place

Return true if a queen can be placed in K^{th} row and i^{th} column otherwise false $x[]$ is a global array whose first $(k-1)$ value have been set. $\text{Abs}(x)$ returns absolute value of r

1. Algorithm Place(K, i)
2. {
3. For $j = 1$ to $k-1$ do
4. If $((x[j] = i) \quad // \text{two in the same column}$
5. Or $(\text{Abs}(x[j] - i) = \text{Abs}(j - k)) \quad // \text{same diagonal}$
6. then return false
7. Return true
8. }

Computing time $O(k-1)$

Solution to N-Queens Problem

Using backtracking it prints all possible placements of n Queens on a $n \times n$ chessboard so that they are not attacking

1. Algorithm NQueens(K, n)
2. {
3. For $i = 1$ to n do
4. { if place(K, i) then
5. { $x[K] := i$
6. If ($k = n$) then //obtained feasible sequence of length n
7. write ($x[1:n]$) //print the sequence
8. Else Nqueens($K+1, n$) //sequence is less than the length so backtrack
9. }
10. }
11. }

Space complexity

- For Algorithm [7.4 & 7.5] it is $O(n)$. The algorithm uses an auxiliary array of length n to store just n positions.

Time complexity

- The **Place** algorithm takes $O(n)$ time as it iterates through our array every time.
- For each invocation of the **NQueens** algorithm, there is a loop which runs for $O(n)$ time.
- In each iteration of this loop, there is **Place** invocation which is $O(n)$ and a recursive call with a smaller argument.
- If we add all this up and define the run time as $T(n)$.

$$\text{Then } T(N) = O(N^2) + N \times T(N-1).$$

- If you draw a recursion tree using this recurrence, the final term will be something like $n^3 + n!O(1)$.
- By the definition of Big O , this can be reduced to $O(n!)$ running time.

Conclusions

- N-Queens itself is only a toy problem; it's the methods for solving it that you're supposed to be learning. Because it's an easily-understood but nontrivial problem, it can demonstrate not just the “standard” solution with backtracking but also techniques such as search space reduction using symmetry, constraint programming, or evolutionary algorithms.

Thanks for Your Attention!



Exercises

Exercises

1. Algorithm `NQueens` can be made more efficient by redefining the function `Place(k, i)` so that it either returns the next legitimate column on which to place the k th queen or an illegal value. Rewrite both functions (Algorithms 7.4 and 7.5) so they implement this alternate strategy.
2. For the n -queens problem we observe that some solutions are simply reflections or rotations of others. For example, when $n = 4$, the two solutions given in Figure 7.9 are equivalent under reflection.

Observe that for finding inequivalent solutions the algorithm need only set $x[1] = 2, 3, \dots, \lceil n/2 \rceil$.

- (a) Modify `NQueens` so that only inequivalent solutions are computed.
- (b) Run the n -queens program devised above for $n = 8, 9$, and 10 . Tabulate the number of solutions your program finds for each value of n .

Exercises

3. Given an $n \times n$ chessboard, a knight is placed on an arbitrary square with coordinates (x, y) . The problem is to determine $n^2 - 1$ knight moves such that every square of the board is visited once if such a sequence of moves exists. Present an algorithm to solve this problem.

Appendix

Time complexity

- $T(n) = n T(n-1) + O(n^2)$ which translates to $O(N!)$ time complexity approximately.
- The running time of your algorithm is at most $N(N-1)(N-2)\cdots(N-K+1)$ i.e., $N!/(N-K)!$ This is $O(N^K)$, i.e., exponential in K .
- Now number of possible arrangements of N Queens on $N \times N$ chessboard is $N!$, given you are *skipping row or column, already having a queen placed*.
- So average and worst case complexity of the solution is $O(N!)$ (since, you are checking all the possible solutions i.e. N^N arrangements). The best case occurs if you find your solution before exploiting all possible arrangements. This depends on your implementation.
- And if you need all the possible solutions, the best, average and worst case complexity remains $O(N!)$