

Backtracking

Mashiwat Tabassum Waishy

Lecturer

Department of

CSE



STAMFORD UNIVERSITY BANGLADESH

General Concepts

Algorithm strategy

Approach to solving a problem

May combine several approaches

Algorithm structure

Iterative \Rightarrow execute action in loop

Recursive \Rightarrow Re apply action to subproblem(s)

Problem type

Satisfying \Rightarrow find any satisfactory solution

Optimization \Rightarrow find **best** solutions (vs. cost metric)

A short list of categories

Many Algorithm types are to be considered:

Simple recursive algorithms



Backtracking algorithms

Divide and conquer algorithms

Dynamic programming algorithms

Greedy algorithms

Branch and bound algorithms

Brute force algorithms

Randomized algorithms

Backtracking

Suppose you have to make a series of *decisions*, among various *choices*, where

You don't have enough information to know what to choose

Each decision leads to a new set of choices

Some sequence of choices (possibly more than one) may be a solution to your problem

Backtracking is a methodical way of trying out various sequences of decisions, until you find one that **"works"**

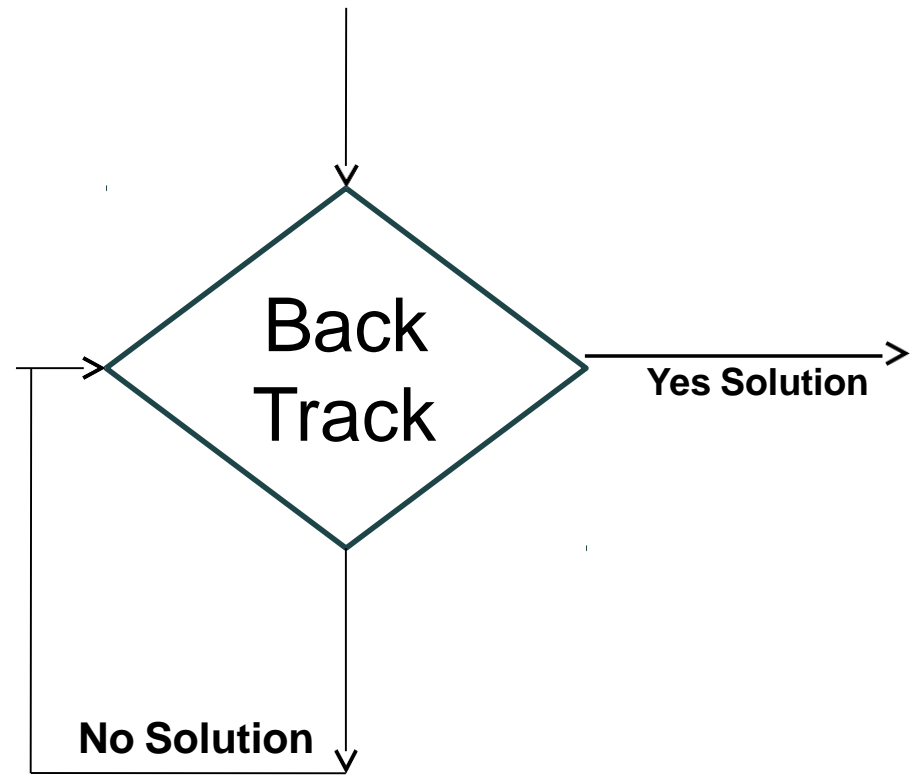
Backtracking Algorithm

Based on depth-first recursive search Approach

1. *Tests whether solution has been found*
2. *If found solution, return it*
3. *Else for each choice that can be made*
 - a) Make that choice
 - b) Recursion
 - c) If recursion returns a solution, return it
4. *If no choices remain, return failure*

Some times called "search tree"

Backtracking: Technique & Examples



Backtracking

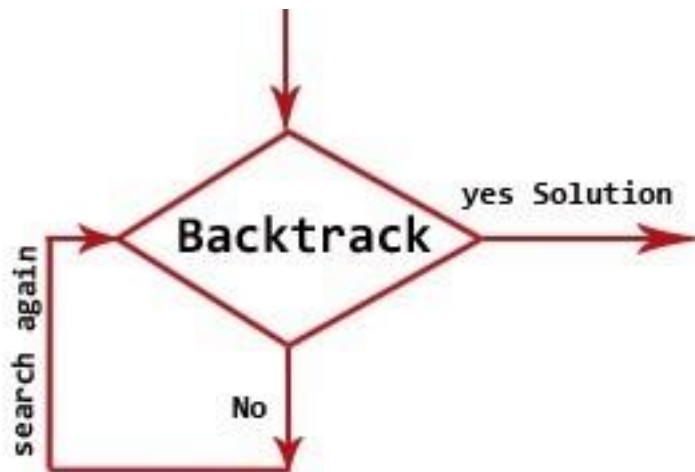
History

- 'Backtrack' the Word was first introduced by Dr. D.H. Lehmer in 1950s.
- R.J Walker Was the First man who gave algorithmic description in 1960.
- Later developed by S. Golomb and L. Baumert.

Backtracking

- What is Backtracking?

-----Backtracking is nothing but the modified process of the brute force approach. where the technique systematically searches for a solution to a problem among all available options. It does so by assuming that the solutions are represented by vectors (v_1, \dots, i_n) of values and by traversing through the domains of the vectors until the solutions is found.



Backtracking

- The Algorithmic Approach

- Backtracking systematically try and search possibilities to find the solution. Also it is an important process for solving constraint satisfaction problem like crossword, Sudoku and many other puzzles. It can be more efficient technique for parsing other combinatorial optimization problem.
- Basically the process is used when the problem has a number of options and just one solution has to be selected. After having a new option set means recursion, the procedure is repeated over and over until final stage.

Backtracking

```
Algorithm Backtrack (v1,Vi)
  If (V1,....., Vi) is a Solution Then
    Return (V1,..., Vi)
  For each v DO
    If (V1,.....,Vi) is acceptable vector THEN
      Sol = try (V1,...,Vi, V)
      If sol != () Then
        RETURN sol
  End
End
Return ( )
```

Backtracking

- **Advantages**

- Comparison with the Dynamic Programming, Backtracking Approach is more effective in some cases.
- Backtracking Algorithm is the best option for solving tactical problem.
- Also Backtracking is effective for constraint satisfaction problem.

- In greedy Algorithm, getting the Global Optimal Solution is a long procedure and depends on user statements but in Backtracking It Can Easily getable.
- Backtracking technique is simple to implement and easy to code.
 - Different states are stored into stack so that the data or Info can be usable anytime.
 - The accuracy is granted.

Backtracking

- Disadvantages

- Backtracking Approach is not efficient for solving strategic Problem.
- The overall runtime of Backtracking Algorithm is normally slow
- To solve Large Problem Sometime it needs to take the help of other techniques like Branch and bound.
- Need Large amount of memory space for storing different state function in the stack for big problem.
- Thrashing is one of the main problem of Backtracking.
- The Basic Approach Detects the conflicts too late.

Backtracking

- Application of Backtracking
 - Optimization and tactical problems
 - Constraints Satisfaction Problem
 - Electrical Engineering
 - Robotics
 - Artificial Intelligence
 - Genetic and bioinformatics Algorithm
 - Materials Engineering
 - Network Communication
 - Solving puzzles and path

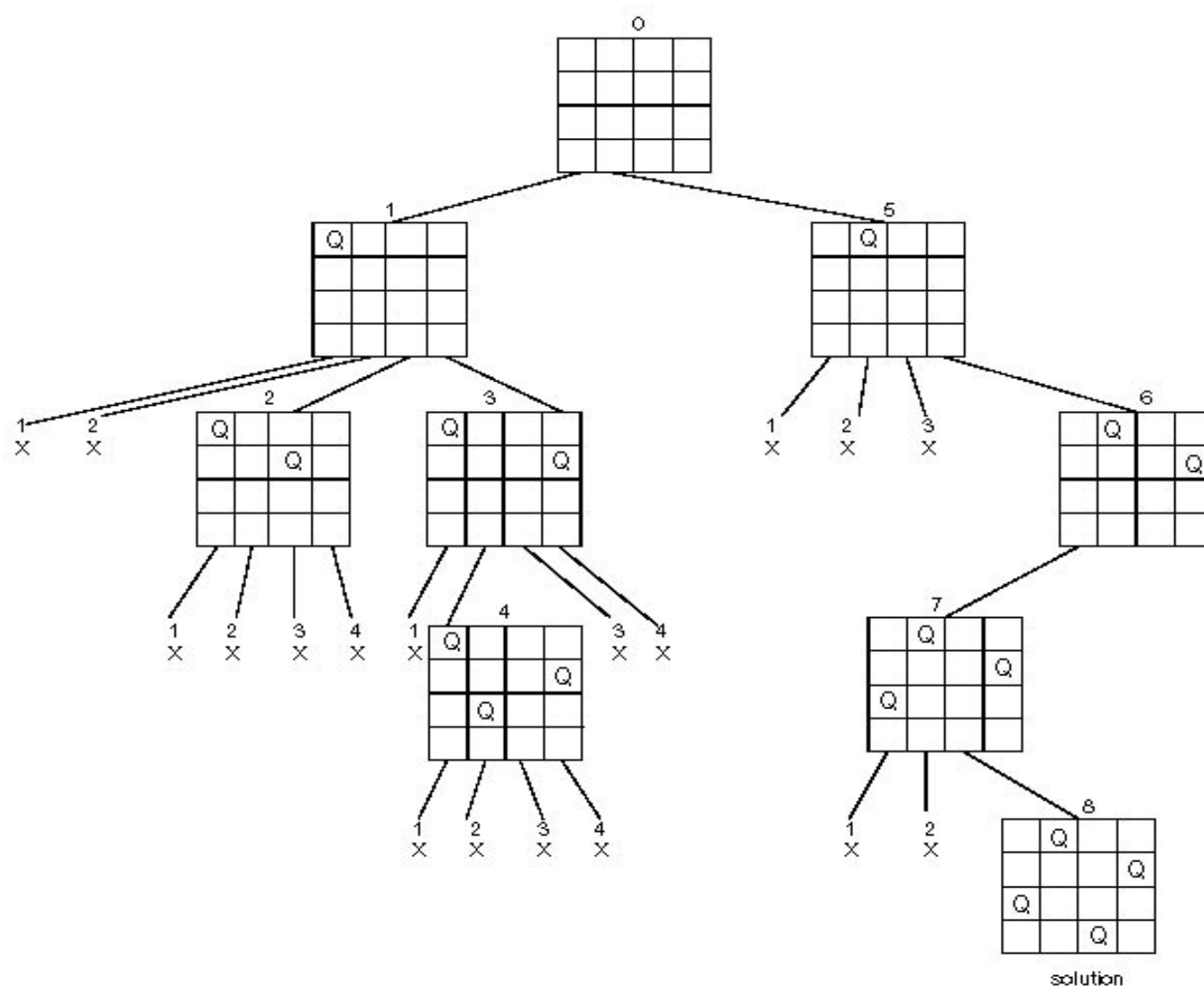
Backtracking

- **Some Problem Solved with Backtracking Technique**
 - N- Queens Problem
 - Sum of Subset
 - Sudoku Puzzle
 - Maze Generation
 - Hamiltonian Cycle

Example: The n -Queen problem

Place n queens on an n by n chess board so that no two of them are not in attacking mode i.e. on the same row, column, or diagonal

State Space Tree of the Four-queens Problem



The Backtracking Algorithm

Backtracking is really quite simple--we “explore” each node, as follows:

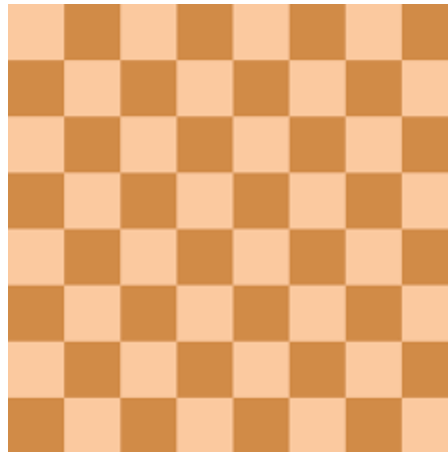
To “explore” node N:

1. *If N is a goal node, return “success”*
2. *If N is a leaf node, return “failure”*
3. *For each child C of N,*
 1. Explore C
 1. If C was successful, return “success”
4. *Return “failure”*

N-Queens Problem

- **History:**

First Introduced in 1848 which was known as 8- queens Puzzle. Surprisingly, The First Solution was created in 1950 by Franz Nauck. Nauck made an 8X8 Chessboard to find the first Feasible Solution.

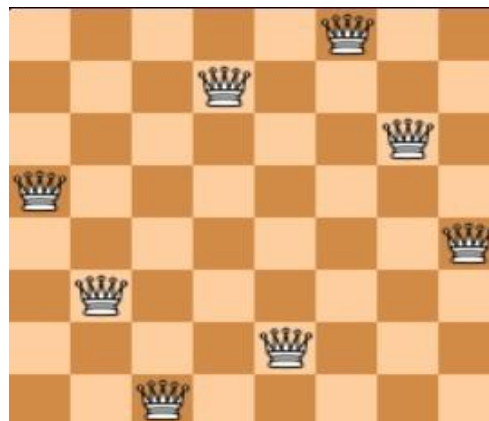


N-Queens Problem

- **Problem Description**

In a $N \times N$ square board N – number of queens need to be placed considering three Condition ---

- No two Queens can be placed in same row.
- No two Queens Can be places in same Column
- No two queens Can be placed in same Diagonal.



Example I: The n-queens problem

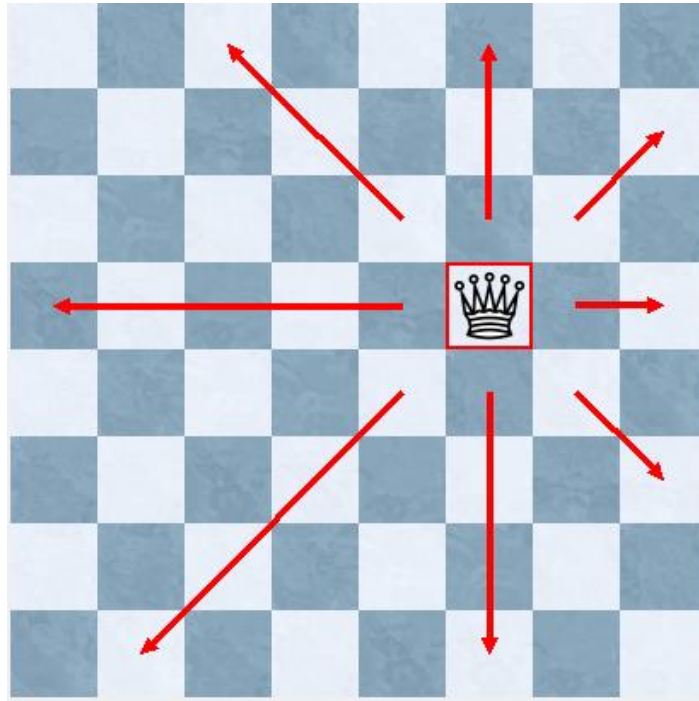
How can we place n queens on an $n \times n$ chessboard so that no two queens can capture each other?

A queen can move any number of squares horizontally, vertically, and diagonally.

Here, the possible target squares of the queen Q are marked with an X.

X			X			X	
	X		X		X		
		X	X	X			
X	X	X	Q	X	X	X	X
		X	X	X			
	X		X		X		
X			X			X	
			X				X

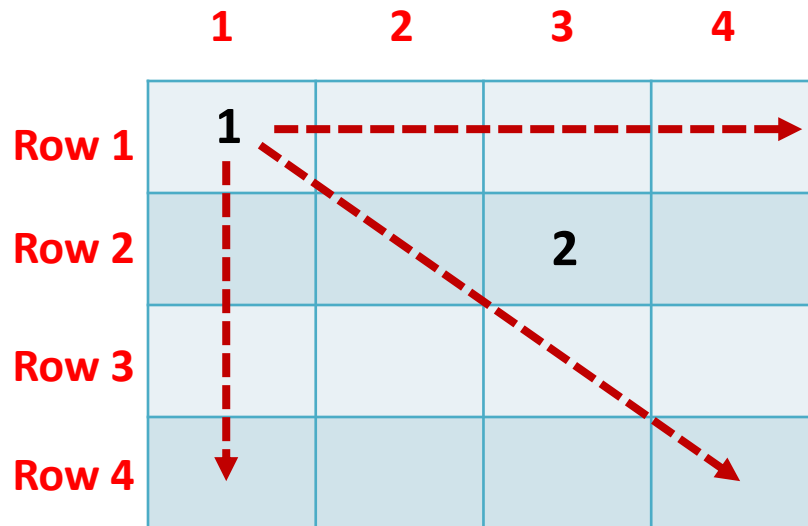
Attacking Position



Example: 4-queen problem

	1	2	3	4
Row 1	1			
Row 2				
Row 3				
Row 4				

Inserting 1st queen in the 1st row(1st position)

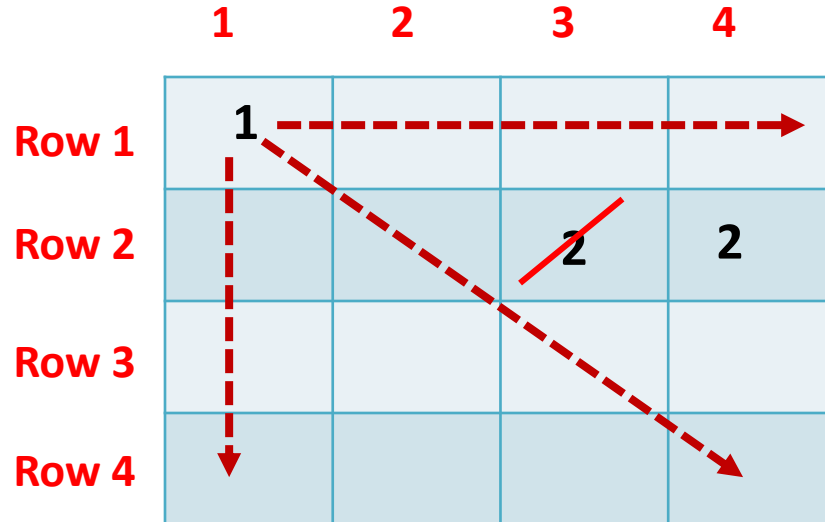


**Avoiding all the attacking position of 1st queen,
inserting 2nd queen in 2nd row(3rd position)**

	1	2	3	4
Row 1	1			
Row 2			2	
Row 3				
Row 4				

	1	2	3	4
Row 1	1			
Row 2			2	
Row 3				
Row 4				

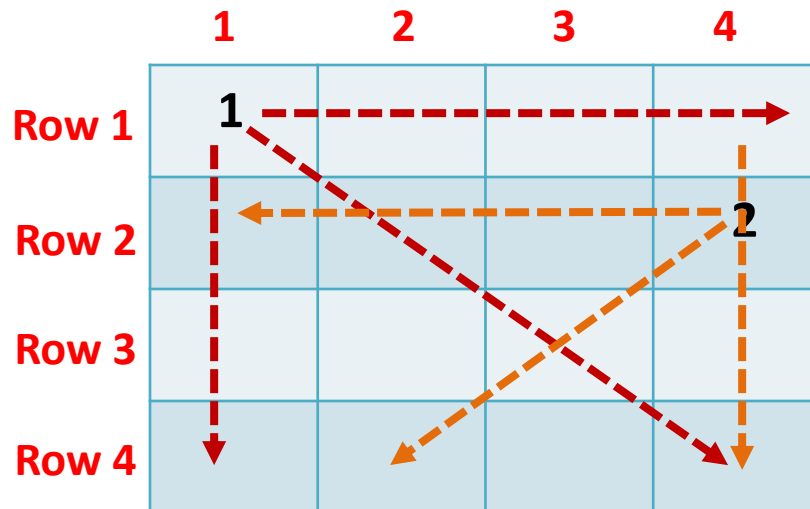
Avoiding all the attacking position of 1st & 2nd queen there is no safe place for 3rd queen in the 3rd column.

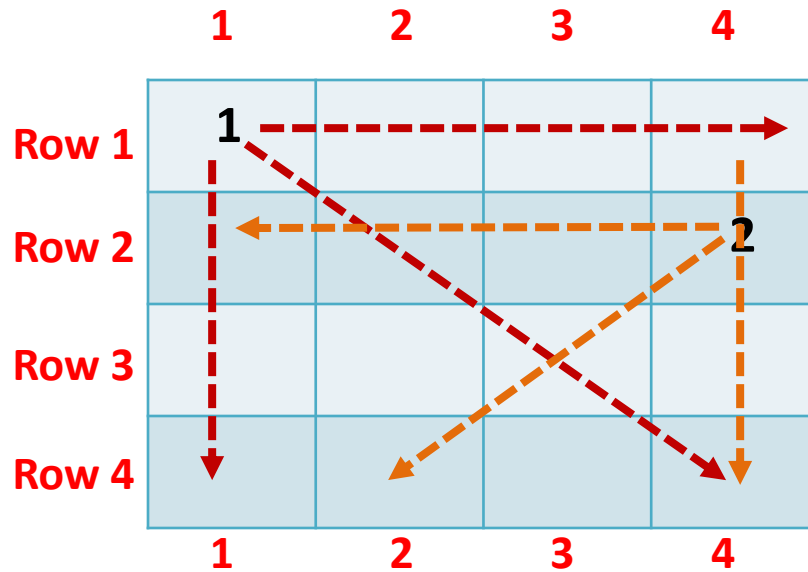


So Backtrack!!

Backtrack to the previous state(previous queen) and it to another safe position.

So, Avoiding all the attacking position of 1st queen, shifting 2nd queen to 2nd row(4th position)

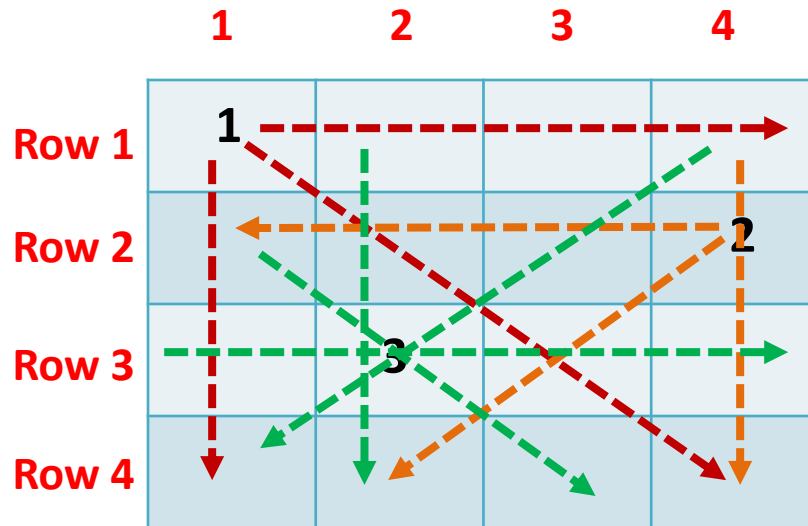




Now got a safe position for 3rd queen in the 3rd row.

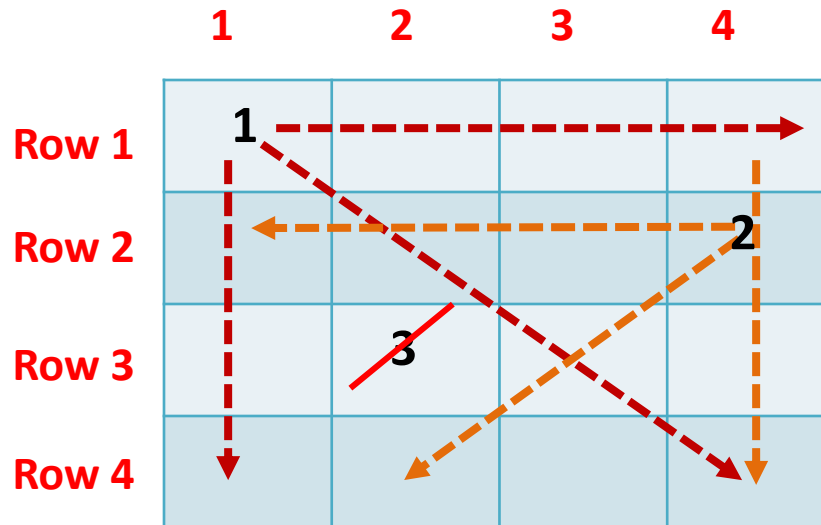
Avoiding all the attacking position of 1st & 2nd queen inserting 3rd queen in the 3rd row (2nd position)





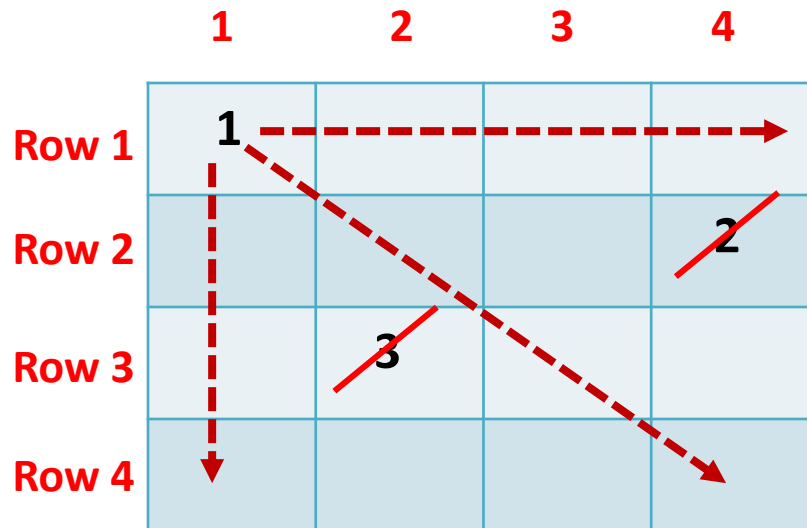
Avoiding all the attacking position of 1st, 2nd and 3rd queen there is no safe place for 4th queen in the 4th column.

So Backtrack!!



Backtrack to the previous state(previous queen[3]) and it to another safe position.
But there is no other safe place for 3rd queen in the 3rd row.

So continue Backtrack!!



Backtrack to the previous state(previous queen[2]) and it to another safe position.

But there is no other safe place for 2nd queen in the 2nd row.

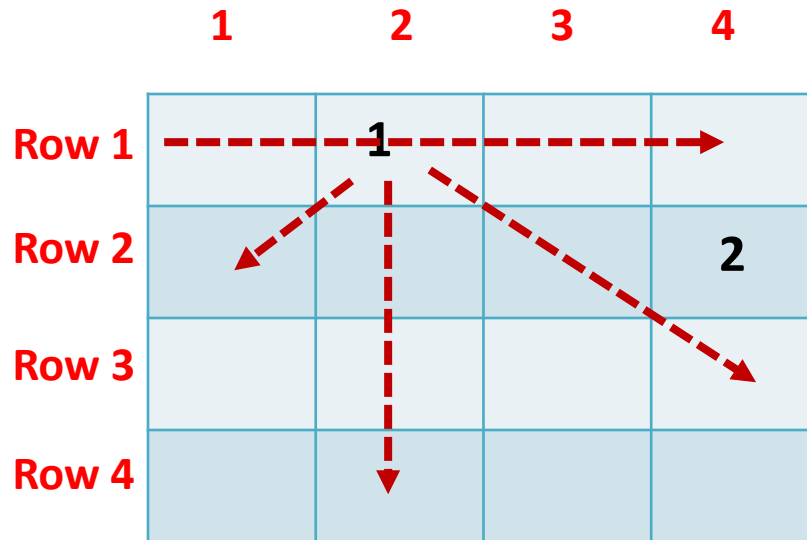
So continue Backtrack!!

	1	2	3	4
Row 1	1	1		
Row 2				2
Row 3		3		
Row 4				

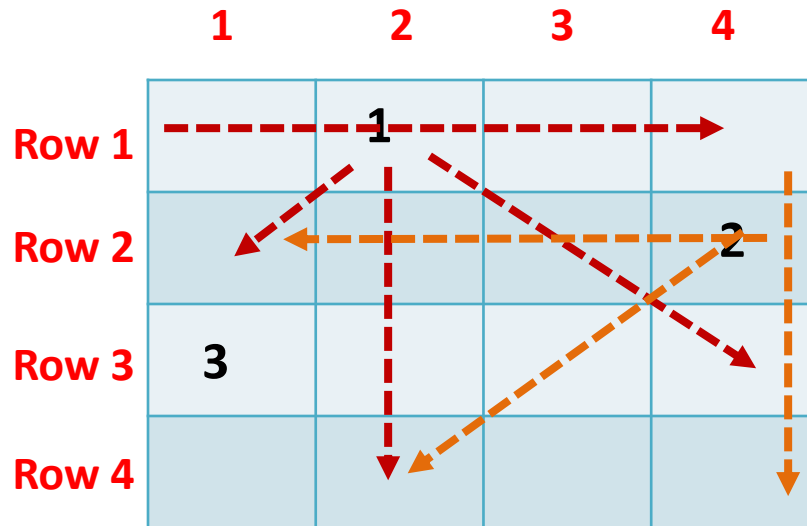
Backtrack to the previous state(previous queen[1]) and it to another safe position.

So, shifting 1st queen to another new position.

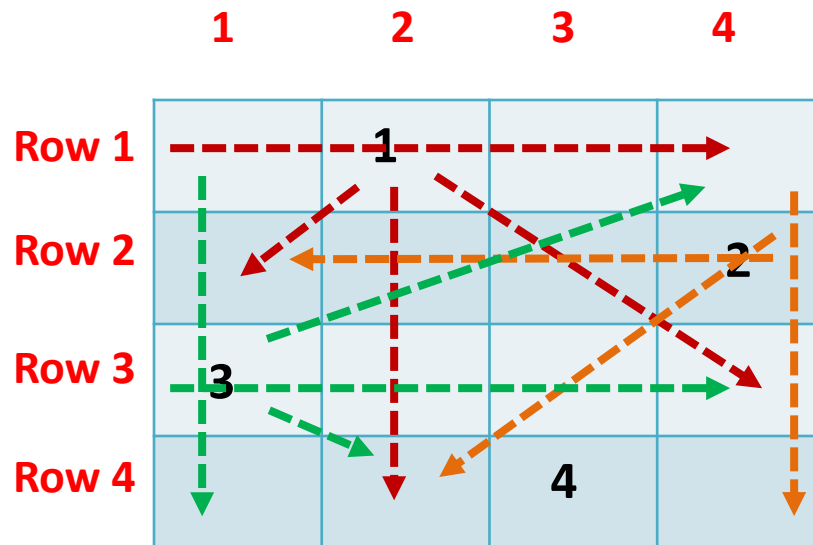
Now 1st queen is in 1st row(2nd position)



**Avoiding all the attacking position of 1st queen,
inserting 2nd queen in 2nd row(4th position)**



Avoiding all the
attacking position of 1st
& 2nd queen inserting
3rd queen in the 3rd
row(1st position)



Avoiding all the attacking position of 1st, 2nd and 3rd queen inserting 4th queen in the 4th row(3rd position)

	1	2	3	4
Row 1		1		
Row 2				2
Row 3	3			
Row 4			4	

All the 4 queens have been inserted .

Output: (2,4,1,3)

Another Output: (3, 1, 4, 2)

Backtracking Algorithm

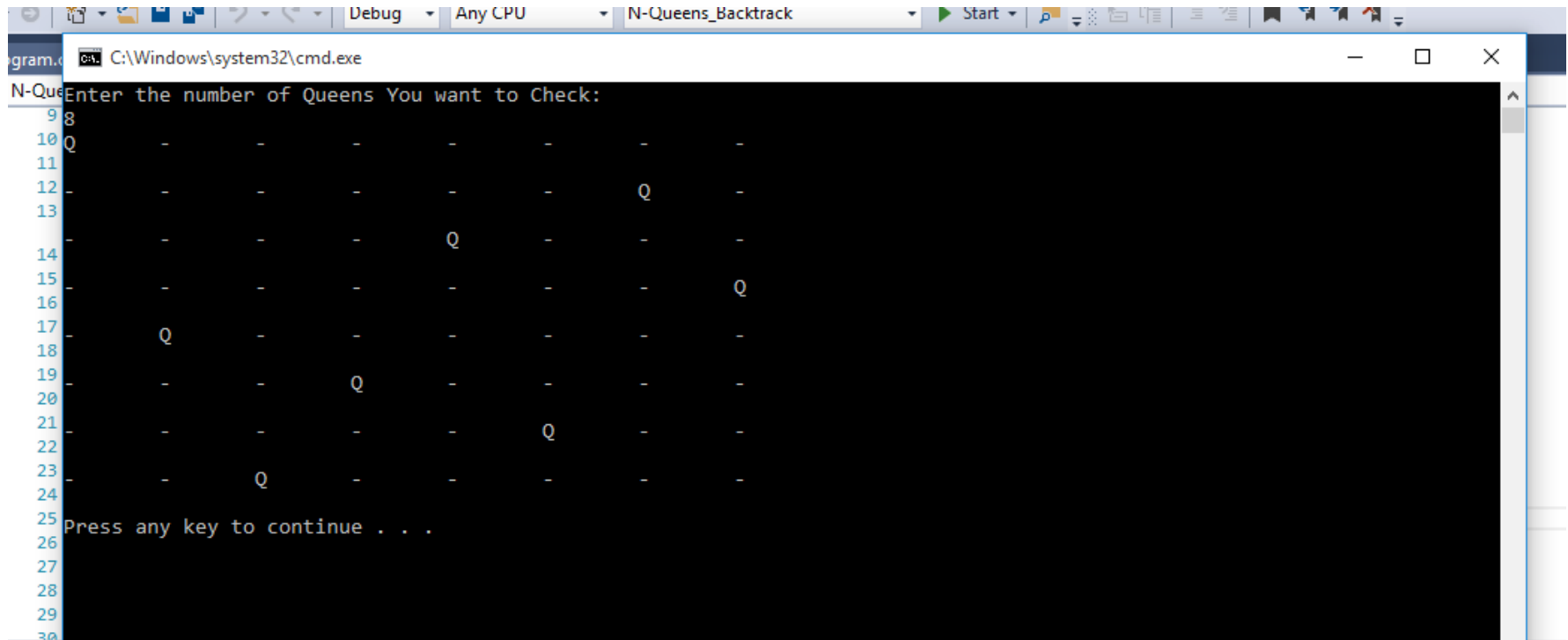
```
1  Algorithm Place( $k, i$ )
2  // Returns true if a queen can be placed in  $k$ th row and
3  //  $i$ th column. Otherwise it returns false.  $x[ ]$  is a
4  // global array whose first  $(k - 1)$  values have been set.
5  // Abs( $r$ ) returns the absolute value of  $r$ .
6  {
7      for  $j := 1$  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  }
```

The Algorithm will check each position $[i, j]$ for each queens . If any Suitable places found , It will place a queen on that position. If not Algorithm will try same approach for next position.

Backtracking Algorithm

```
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 }
```

N-Queens Problem



```
gram.d C:\Windows\system32\cmd.exe
N-Que Enter the number of Queens You want to Check:
9
10 Q - - - - -
11 - - - - -
12 - - - - Q -
13 - - - Q - -
14 - - - - Q -
15 - - - - - Q
16 - - - - -
17 - Q - - - -
18 - - - - -
19 - - - Q - -
20 - - - - -
21 - - - - Q -
22 - - - - -
23 - - Q - - -
24 - - - - -
25 Press any key to continue . . .
26
27
28
29
30
```

Thank you