Backtracking

General Method

n-Queens Problem

Backtracking

- General Method
- The n Queens Problem
- Sum Of Subsets

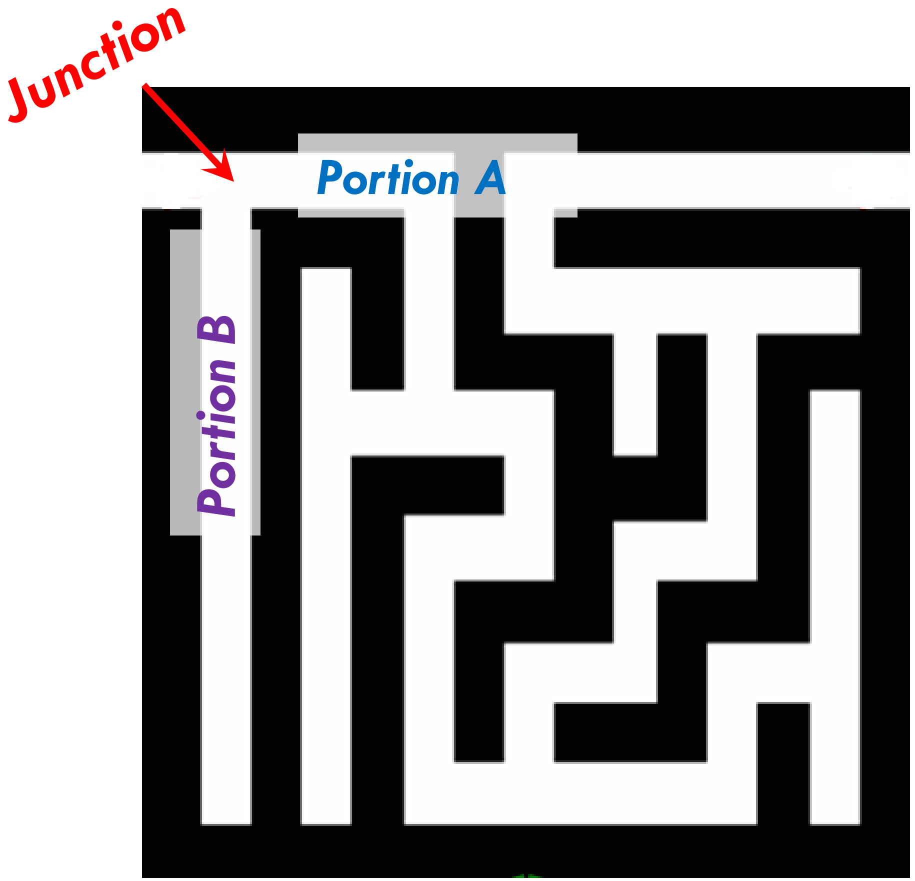
Backtracking

- 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"

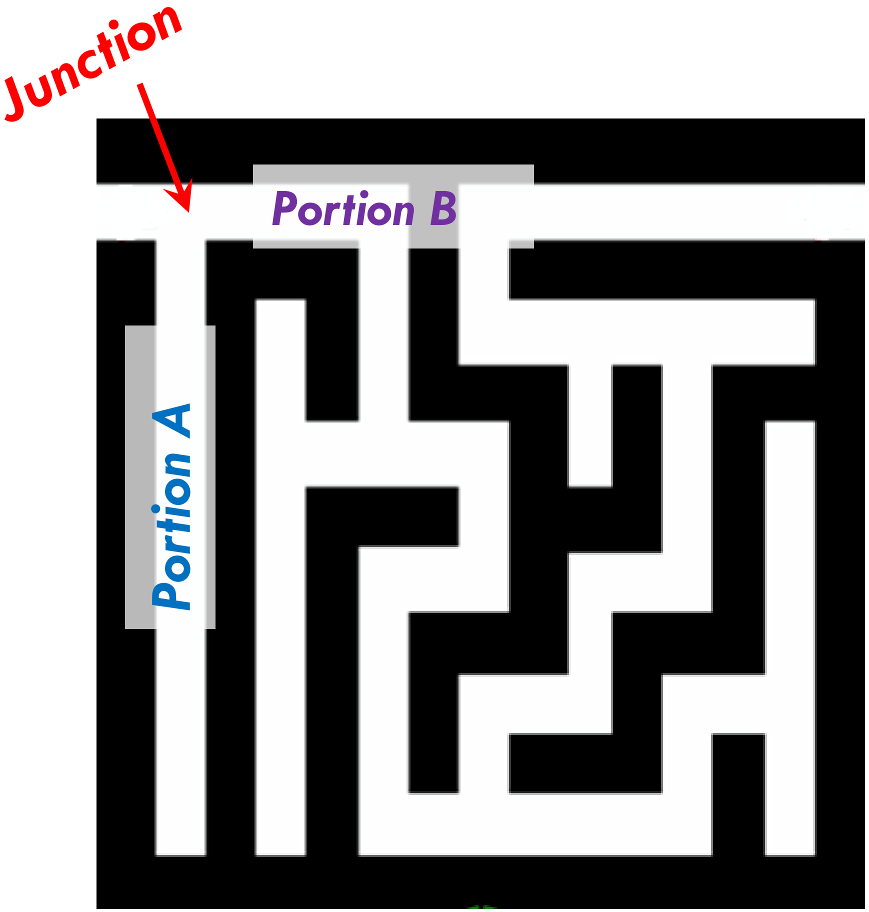
Solving a maze

- Given a maze, find a path from start to finish
- At each intersection, you have to decide between three or fewer choices:
  - Go straight
  - Go left
  - Go right
- You don't have enough information to choose correctly
- Each choice leads to another set of choices
- One or more sequences of choices may (or may not) lead to a solution
- Many types of maze problem can be solved with backtracking

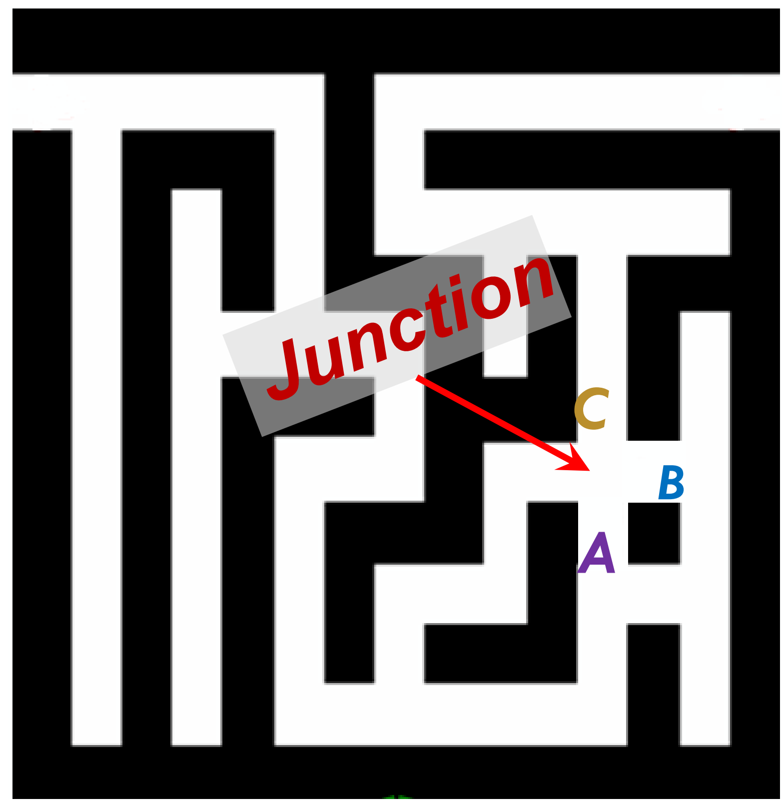
- At some point in a maze, you might have two options of which direction to go:



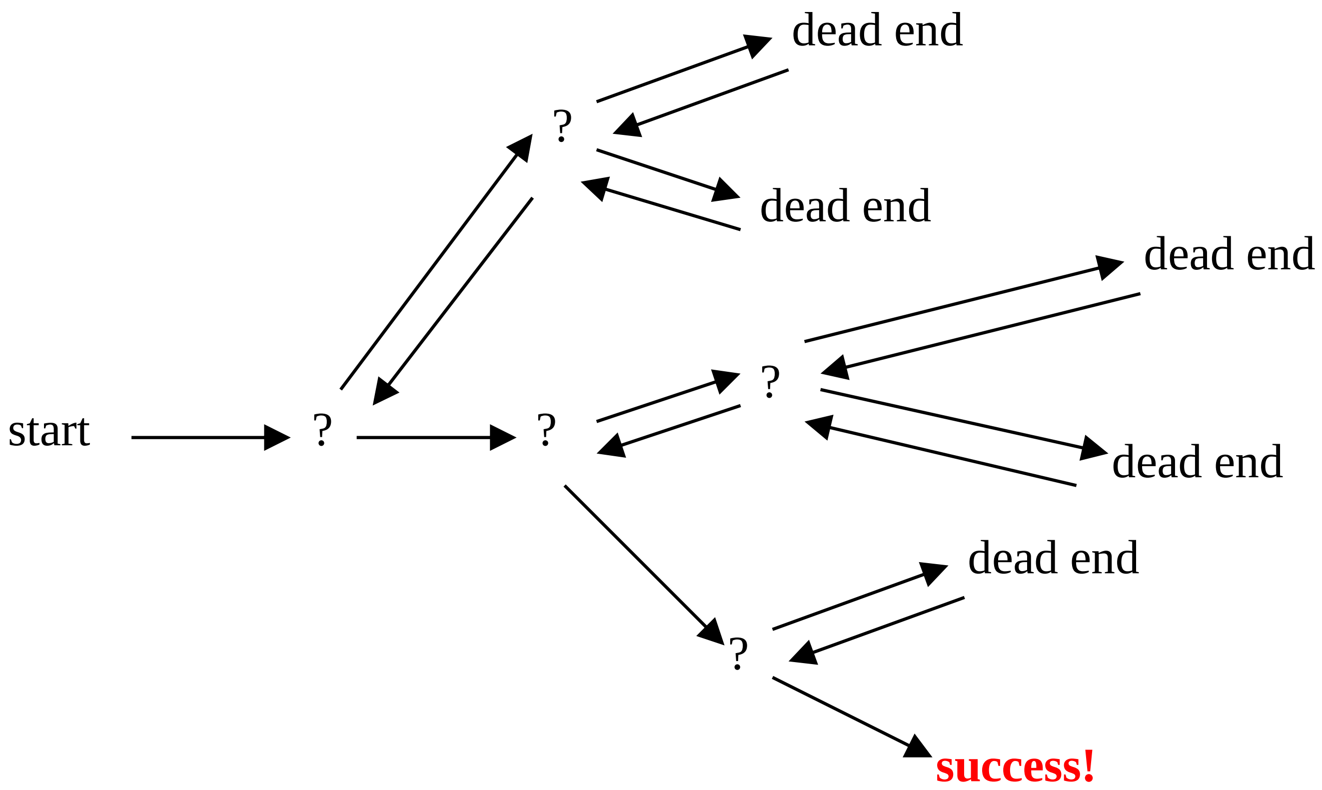
- One strategy would be to try going through **Portion A** of the maze.
    - If you get stuck before you find your way out, then you ***"backtrack"*** to the junction.
- At this point in time you know that **Portion A** will ***NOT*** lead you out of the maze,
    - so you then start searching in **Portion B**



- Clearly, at a single junction you could have even more than 2 choices.
- The backtracking strategy says to try each choice, one after the other,
  - if you ever get stuck, ***"backtrack"*** to the junction and try the next choice.
- If you try all choices and never found a way out, then there IS no solution to the maze.

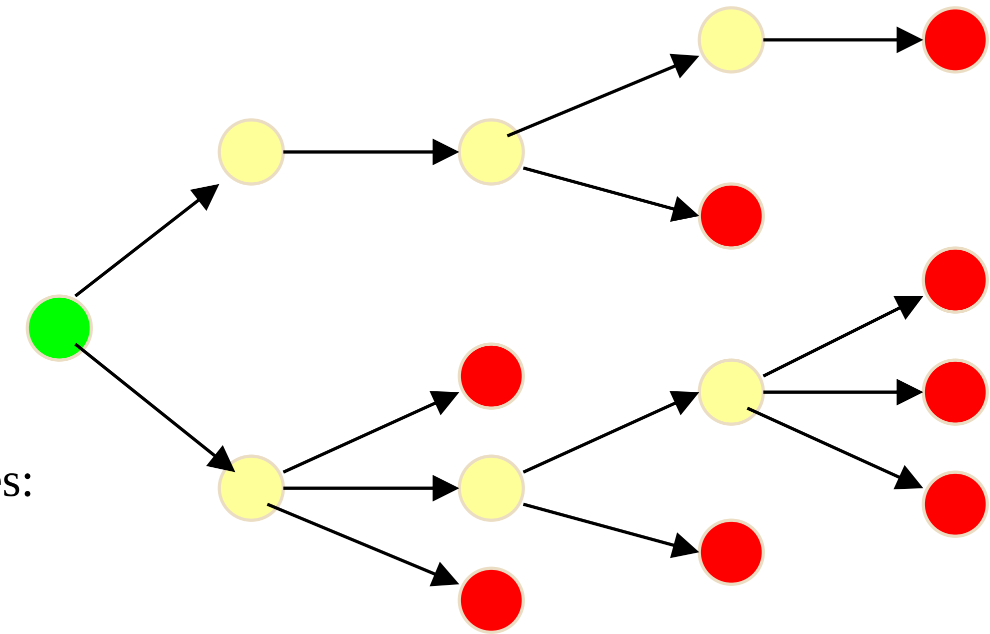


Backtracking (animation)



Terminology I

A tree is composed of nodes



There are three kinds of nodes:

The (one) root node

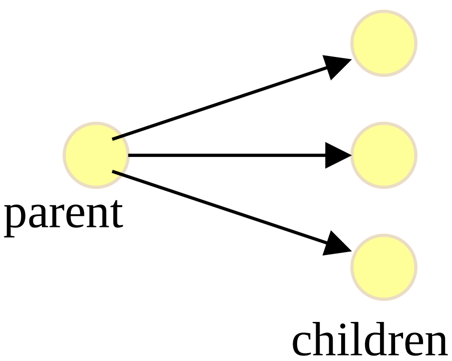
Internal nodes

Leaf nodes

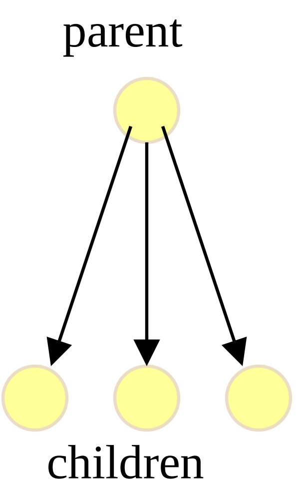
Backtracking can be thought of as searching a tree for a particular “goal” leaf node

Terminology II

- Each non-leaf node in a tree is a parent of one or more other nodes (its children)
- Each node in the tree, other than the root, has exactly one parent



Usually, however, we draw our trees *downward*, with the root at the top



Backtracking -General Method

- The name backtrack was first coined by D.H Lehmer in 1950s
- One of the most general techniques for algorithm design
- It can be used for problems that have a set of solutions
- The desired solution is expressible as an n-tuple(x1,x2…xn) where xi are chosen from some finite set si.
- Find one vector that maximizes(or minimizes) a criterion function P(x1,x2,……xn) of integers in a[1…n] where xi is the index in a of the i th smallest element

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,
     3.1. Explore C
        3.1.1. If C was successful, return “success”
  4. Return “failure”

Backtracking

N-Queen Problem

Problem:- The problem is to place n queens on an n-by-n chessboard so that no two queens attack each other by being in the same row, or in the same column, or in the same diagonal.

Observation:-

Case 1 : n=1

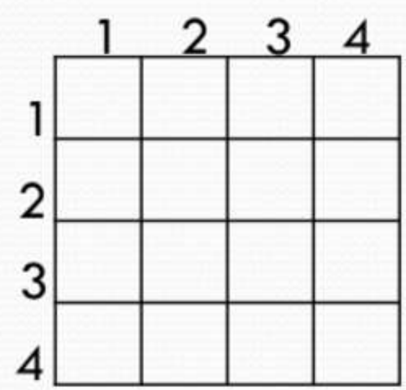
Case 2 : n=2

Case 3 : n=3

Case 4 : n=4

Backtracking

- **Case 4:** For example to explain the n-Queen problem we Consider n=4 using a 4-by-4 chessboard where 4-Queens have to be placed in such a way so that no two queen can attack each other.



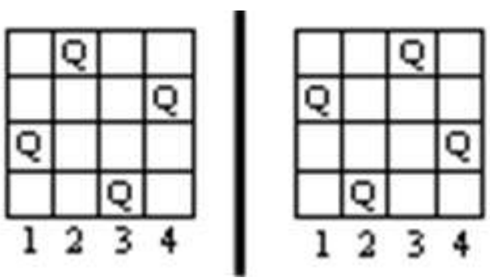
	Q		
			Q
Q			
		Q	

		Q	
Q			
			Q
	Q		

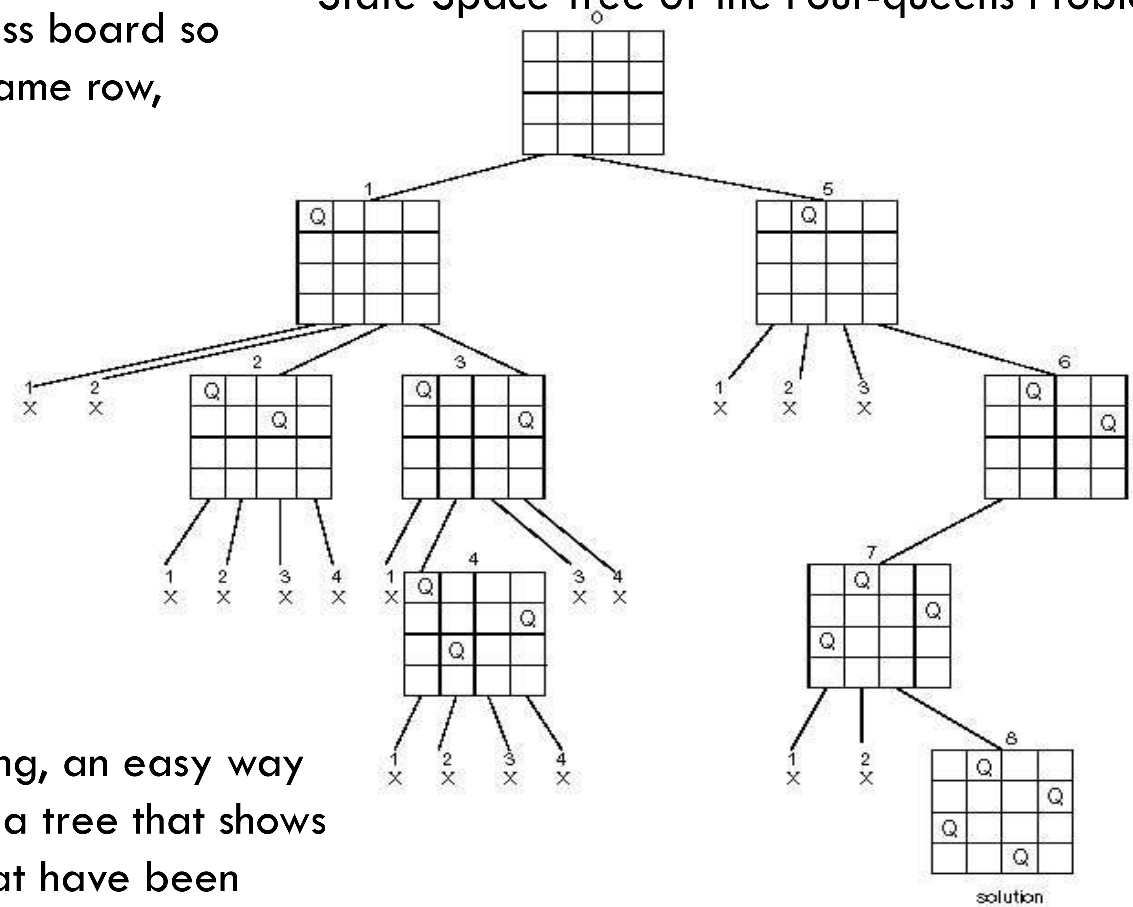
Backtracking – Four Queens Problem

Place *n* queens on an 4 by *4* chess board so that no two of them are on the same row, column, or diagonal

Two Solutions:



State Space Tree of the Four-queens Problem



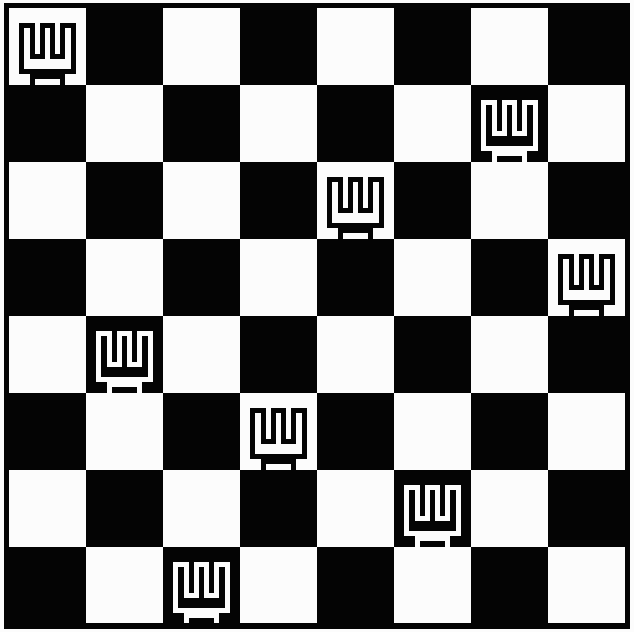
When we carry out backtracking, an easy way to visualize what is going on is a tree that shows all the different possibilities that have been tried.

Backtracking – Eight Queens Problem

- Find an arrangement of **8** queens on a single chess board such that no two queens are attacking one another.

- In chess, queens can move all the way down any row, column or diagonal (so long as no pieces are in the way).

  - Due to the first two restrictions, it's clear that each row and column of the board will have exactly one queen.



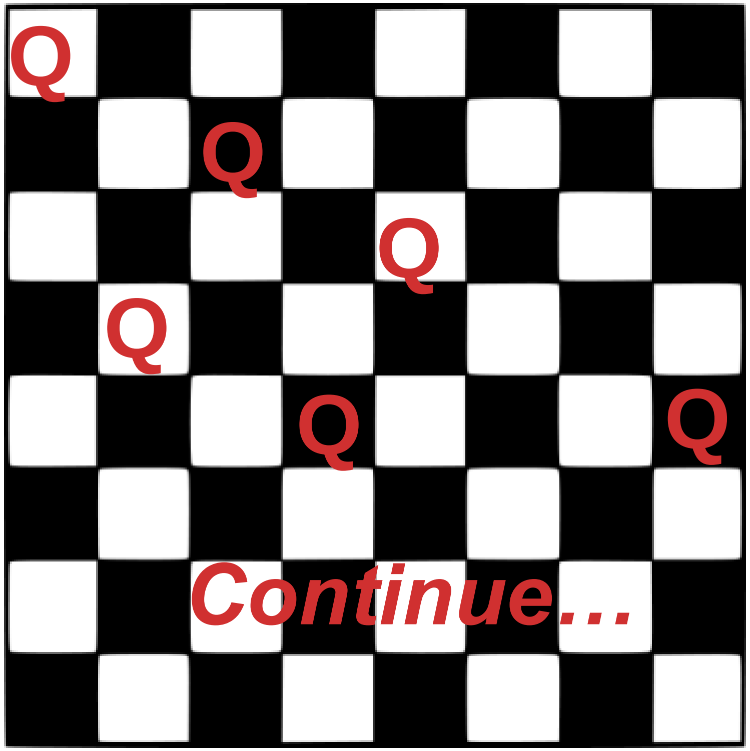
The 8 Queen's Problem

- Problem Definition
  - Place eight queens on a 8*8 chessboard so that no two of them are on the same row, column or diagonal
- Constraints
  - No two x_i can be the same
  - Each queen must be on a different row
  - No two queens can be on the same diagonal

The solution space consists of 8^8 tuples

Backtracking – Eight Queens Problem

- The backtracking strategy is as follows:
  1) Place a queen on the first available square in row 1.
  2) Move onto the next row, placing a queen on the first available square there (that doesn't conflict with the previously placed queens).
  3) Continue in this fashion until either:
     a) you have solved the problem, or
     b) you get stuck.
        - When you get stuck, remove the queens that got you there, until you get to a row where there is another valid square to try.



The 8 Queen's Problem-Example

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

The n Queen's Problem-Algorithm


```
Algorithm Place(k, i)
// Returns true if a queen can be placed in kth row and
// ith column. Otherwise it returns false. x[ ] is a
// global array whose first (k − 1) values have been set.
// Abs(r) returns the absolute value of r.
{
    for j := 1 to k − 1 do
        if ((x[j] = i) // Two in the same column
            or (Abs(x[j] − i) = Abs(j − k)))
                // or in the same diagonal
            then return false;
    return true;
}
```

```
Algorithm NQueens(k, n)
// Using backtracking, this procedure prints all
// possible placements of n queens on an n × n
// chessboard so that they are nonattacking.
{
    for i := 1 to n do
    {
        if Place(k, i) then
        {
            x[k] := i;
            if (k = n) then write (x[1 : n]);
            else NQueens(k + 1, n);
        }
    }
}
```