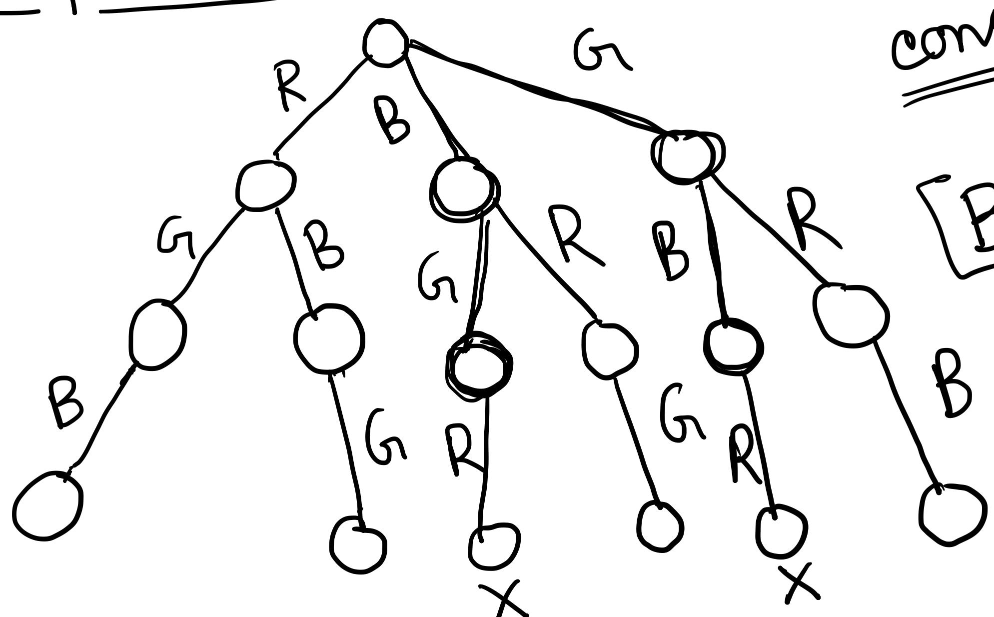
Backtracking → Another algorithm design technique

We will find all solutions.
(Brute force approach)

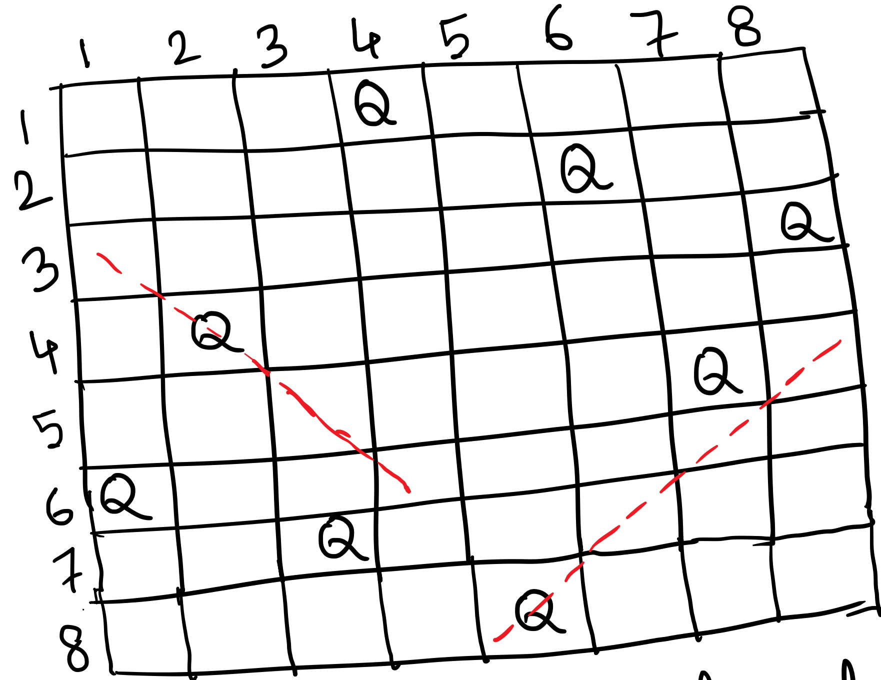
Balls: R G B

State space tree:



Backtracking
↓
constraint
⇓
[Bounding function]

N-Queen's problem: Place n-queens in a n×n chessboard so that no two of them can attack. i.e. no two of them are on the same row, column or diagonal.



8x8 chessboard

$n = 8$

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

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

Let, (i, j) & (k, l) are in same diagonal.

(5,8) (6,7) (7,6) (8,5)

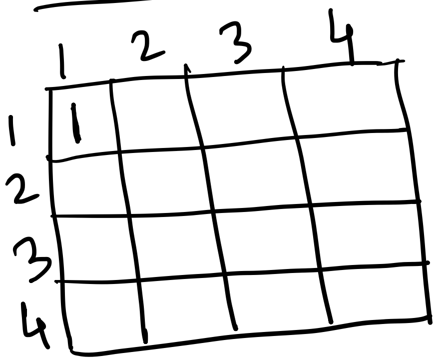
Let, (i, j) & (k, l) are in same diagonal.

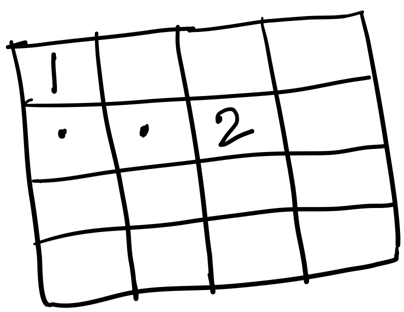
$(i-j) = (k-l)$
$\Rightarrow (j-l) = (i-k)$

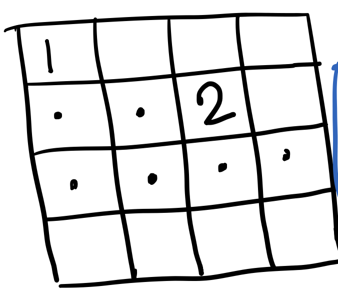
$(i+j) = (k+l)$
$\Rightarrow (j-l) = (k-i)$

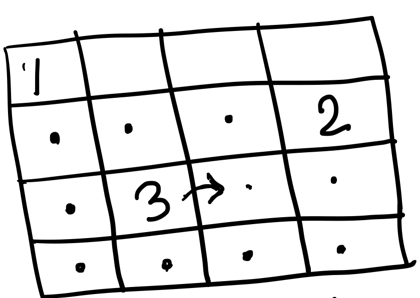
$\boxed{abs(j-l) = abs(i-k)}$

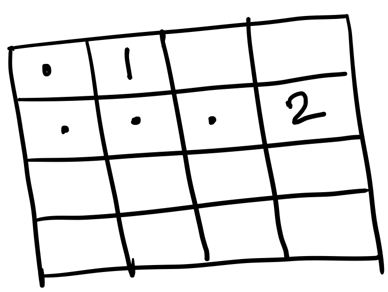
4x4 chessboard:

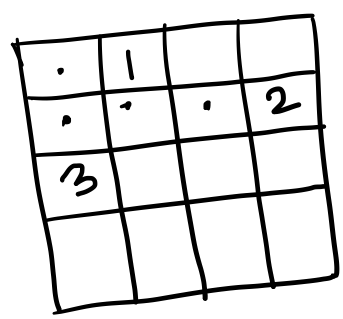


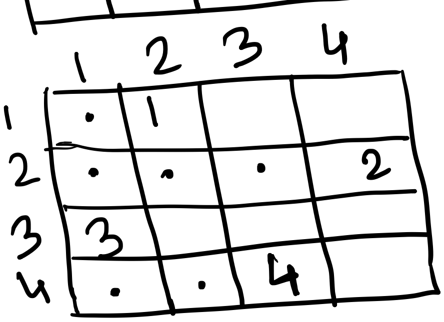












(2, 4, 1, 3) ⇒ Solution 1

(3, 1, 4, 2) ⇒ Solution 2

```
Algorithm NQueens(k, n)
{ // print all possible solutions using
  // backtracking
  for i := 1 to n do
  {
     if (Place(k, i)) then
     {
        x[k] := i
        if (k = n) then
            print (x[1 : n])
        else
            NQueens(k+1, n)
     }
  }
}
```

Global solution array

```
Algorithm Place(k, i)
{ // returns true if Q is placed at k-th row
  // and ith column, otherwise returns false
  for j := 1 to (k-1) do // check for all previous queens.
  { if ((x[j] = i) // in the same column
     or (Abs(x[j] - i) = Abs(j - k))) then
                            // in the same diagonal
        return false
  }
  return true
```

}