Problem 51: N-Queens

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

The

n-queens

puzzle is the problem of placing

n

queens on an

n x n

chessboard such that no two queens attack each other.

Given an integer

n

, return

all distinct solutions to the

n-queens puzzle

. You may return the answer in

any order

.

Each solution contains a distinct board configuration of the n-queens' placement, where

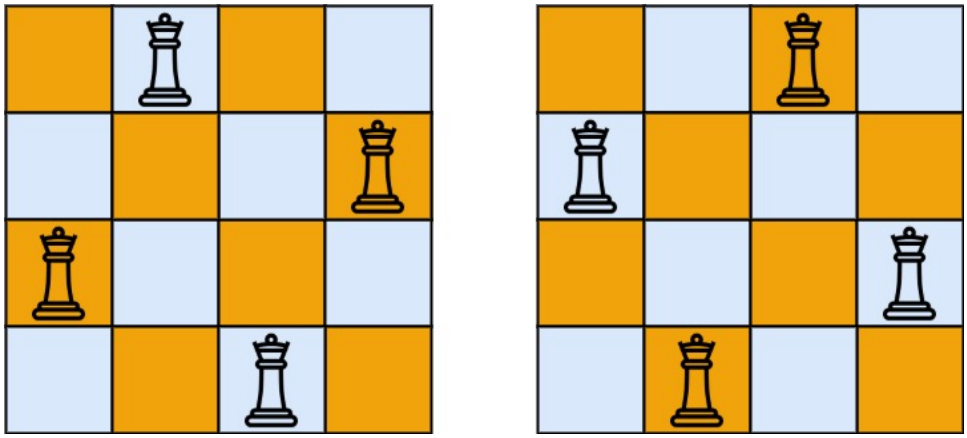
'Q'

and

'.'

both indicate a queen and an empty space, respectively.

Example 1:



Input:

n = 4

Output:

[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]

Explanation:

There exist two distinct solutions to the 4-queens puzzle as shown above

Example 2:

Input:

n = 1

Output:

[["Q"]]

Constraints:

1 <= n <= 9

Code Snippets

C++:

```
class Solution {
public:
vector<vector<string>> solveNQueens(int n) {

}
};
```

Java:

```
class Solution {
public List<List<String>> solveNQueens(int n) {

}
}
```

Python3:

```
class Solution:
def solveNQueens(self, n: int) -> List[List[str]]:
```

Python:

```
class Solution(object):
def solveNQueens(self, n):
```

```
"""
:type n: int
:rtype: List[List[str]]
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {string[][]}
*/
var solveNQueens = function(n) {

};
```

TypeScript:

```
function solveNQueens(n: number): string[][] {

};
```

C#:

```
public class Solution {
public IList<IList<string>> SolveNQueens(int n) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** solveNQueens(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func solveNQueens(n int) [][]string {

}
```

Kotlin:

```
class Solution {
fun solveNQueens(n: Int): List<List<String>> {

}
}
```

Swift:

```
class Solution {
func solveNQueens(_ n: Int) -> [[String]] {

}
}
```

Rust:

```
impl Solution {
pub fn solve_n_queens(n: i32) -> Vec<Vec<String>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {String[][]}
def solve_n_queens(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return String[][]
```

```
*/
function solveNQueens($n) {

}
}
```

Dart:

```
class Solution {
List<List<String>> solveNQueens(int n) {

}
}
```

Scala:

```
object Solution {
def solveNQueens(n: Int): List[List[String]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec solve_n_queens(n :: integer) :: [[String.t]]
def solve_n_queens(n) do

end
end
```

Erlang:

```
-spec solve_n_queens(N :: integer()) -> [[unicode:unicode_binary()]].
solve_n_queens(N) ->
.
```

Racket:

```
(define/contract (solve-n-queens n)
(-> exact-integer? (listof (listof string?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: N-Queens
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<string>> solveNQueens(int n) {

}
};
```

Java Solution:

```
/**
* Problem: N-Queens
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<List<String>> solveNQueens(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: N-Queens
Difficulty: Hard
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def solveNQueens(self, n: int) -> List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def solveNQueens(self, n):
"""
:type n: int
:rtype: List[List[str]]
"""
```

JavaScript Solution:

```
/**
* Problem: N-Queens
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {string[][]}
*/
var solveNQueens = function(n) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: N-Queens
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function solveNQueens(n: number): string[][] {

};
```

C# Solution:

```
/*
* Problem: N-Queens
* Difficulty: Hard
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<IList<string>> SolveNQueens(int n) {

}
}
```

C Solution:

```
/*
* Problem: N-Queens
* Difficulty: Hard
```

```
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** solveNQueens(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: N-Queens
// Difficulty: Hard
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func solveNQueens(n int) [][]string {

}
```

Kotlin Solution:

```
class Solution {
fun solveNQueens(n: Int): List<List<String>> {

}
}
```

Swift Solution:

```
class Solution {
func solveNQueens(_ n: Int) -> [[String]] {

}
}
```

Rust Solution:

```
// Problem: N-Queens
// Difficulty: Hard
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn solve_n_queens(n: i32) -> Vec<Vec<String>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {String[][]}
def solve_n_queens(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return String[][]
*/
function solveNQueens($n) {

}
}
```

Dart Solution:

```
class Solution {
List<List<String>> solveNQueens(int n) {

}
}
```

Scala Solution:

```
object Solution {
def solveNQueens(n: Int): List[List[String]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec solve_n_queens(n :: integer) :: [[String.t]]
def solve_n_queens(n) do

end
end
```

Erlang Solution:

```
-spec solve_n_queens(N :: integer()) -> [[unicode:unicode_binary()]].
solve_n_queens(N) ->
.
```

Racket Solution:

```
(define/contract (solve-n-queens n)
(-> exact-integer? (listof (listof string?)))
)
```