Problem 22: Generate Parentheses

Problem Information

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

Problem Description

Given

n

pairs of parentheses, write a function to

generate all combinations of well-formed parentheses

.

Example 1:

Input:

n = 3

Output:

["((()))","(()())","(())()","()(())","()()()"]

Example 2:

Input:

n = 1

Output:

["()"]

Constraints:

1 <= n <= 8

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** generateParenthesis(int n, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

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

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (generate-parenthesis n)
(-> exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Generate Parentheses
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Generate Parentheses
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Generate Parentheses
Difficulty: Medium
Tags: string, dp

Approach: String manipulation with hash map or two pointers
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Generate Parentheses
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Generate Parentheses
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Generate Parentheses
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Generate Parentheses
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** generateParenthesis(int n, int* returnSize) {

}
```

Go Solution:

```
// Problem: Generate Parentheses
// Difficulty: Medium
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func generateParenthesis(n int) []string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Generate Parentheses
// Difficulty: Medium
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (generate-parenthesis n)
(-> exact-integer? (listof string?))
)
```