

Problem 1238: Circular Permutation in Binary Representation

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given 2 integers

n

and

start

. Your task is return

any

permutation

p

of

(0,1,2....., $2^n - 1$)

such that :

$p[0] = \text{start}$

$p[i]$

and

$p[i+1]$

differ by only one bit in their binary representation.

$p[0]$

and

$p[2^n - 1]$

must also differ by only one bit in their binary representation.

Example 1:

Input:

$n = 2, \text{start} = 3$

Output:

[3,2,0,1]

Explanation:

The binary representation of the permutation is (11,10,00,01). All the adjacent element differ by one bit. Another valid permutation is [3,1,0,2]

Example 2:

Input:

$n = 3, \text{start} = 2$

Output:

[2,6,7,5,4,0,1,3]

Explanation:

The binary representation of the permutation is (010,110,111,101,100,000,001,011).

Constraints:

$1 \leq n \leq 16$

$0 \leq \text{start} < 2^n$

Code Snippets

C++:

```
class Solution {  
public:  
    vector<int> circularPermutation(int n, int start) {  
  
    }  
};
```

Java:

```
class Solution {  
public List<Integer> circularPermutation(int n, int start) {  
  
}  
}
```

Python3:

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

Python:

```
class Solution(object):  
    def circularPermutation(self, n, start):  
        """  
        :type n: int
```

```
:type start: int
:rtype: List[int]
"""

```

JavaScript:

```
/**
 * @param {number} n
 * @param {number} start
 * @return {number[]}
 */
var circularPermutation = function(n, start) {
};


```

TypeScript:

```
function circularPermutation(n: number, start: number): number[] {
};


```

C#:

```
public class Solution {
public IList<int> CircularPermutation(int n, int start) {

}
}
```

C:

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

}
```

Go:

```
func circularPermutation(n int, start int) []int {
```

```
}
```

Kotlin:

```
class Solution {  
    fun circularPermutation(n: Int, start: Int): List<Int> {  
          
    }  
}
```

Swift:

```
class Solution {  
    func circularPermutation(_ n: Int, _ start: Int) -> [Int] {  
          
    }  
}
```

Rust:

```
impl Solution {  
    pub fn circular_permutation(n: i32, start: i32) -> Vec<i32> {  
          
    }  
}
```

Ruby:

```
# @param {Integer} n  
# @param {Integer} start  
# @return {Integer[]}  
def circular_permutation(n, start)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer $n  
     * @param Integer $start
```

```
* @return Integer[]
*/
function circularPermutation($n, $start) {

}
}
```

Scala:

```
object Solution {
def circularPermutation(n: Int, start: Int): List[Int] = {

}
}
```

Solutions

C++ Solution:

```
/*
 * Problem: Circular Permutation in Binary Representation
 * Difficulty: Medium
 * Tags: math
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public:
vector<int> circularPermutation(int n, int start) {

};
};
```

Java Solution:

```
/**
 * Problem: Circular Permutation in Binary Representation
```

```

* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

class Solution {
    public List<Integer> circularPermutation(int n, int start) {
        return null;
    }
}

```

Python3 Solution:

```

"""
Problem: Circular Permutation in Binary Representation
Difficulty: Medium
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

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

```

Python Solution:

```

class Solution(object):
    def circularPermutation(self, n, start):
        """
        :type n: int
        :type start: int
        :rtype: List[int]
        """

```

JavaScript Solution:

```
/**  
 * Problem: Circular Permutation in Binary Representation  
 * Difficulty: Medium  
 * Tags: math  
 *  
 * Approach: Optimized algorithm based on problem constraints  
 * Time Complexity: O(n) to O(n^2) depending on approach  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
/**  
 * @param {number} n  
 * @param {number} start  
 * @return {number[]}   
 */  
var circularPermutation = function(n, start) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Circular Permutation in Binary Representation  
 * Difficulty: Medium  
 * Tags: math  
 *  
 * Approach: Optimized algorithm based on problem constraints  
 * Time Complexity: O(n) to O(n^2) depending on approach  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
function circularPermutation(n: number, start: number): number[] {  
  
};
```

C# Solution:

```
/*  
 * Problem: Circular Permutation in Binary Representation  
 * Difficulty: Medium  
 * Tags: math
```

```

/*
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public IList<int> CircularPermutation(int n, int start) {
        return null;
    }
}

```

C Solution:

```

/*
 * Problem: Circular Permutation in Binary Representation
 * Difficulty: Medium
 * Tags: math
 *
 * Approach: Optimized algorithm based on problem constraints
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

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

}

```

Go Solution:

```

// Problem: Circular Permutation in Binary Representation
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

```

```
func circularPermutation(n int, start int) []int {  
    }  
}
```

Kotlin Solution:

```
class Solution {  
    fun circularPermutation(n: Int, start: Int): List<Int> {  
        }  
        }  
}
```

Swift Solution:

```
class Solution {  
    func circularPermutation(_ n: Int, _ start: Int) -> [Int] {  
        }  
        }  
}
```

Rust Solution:

```
// Problem: Circular Permutation in Binary Representation  
// Difficulty: Medium  
// Tags: math  
//  
// Approach: Optimized algorithm based on problem constraints  
// Time Complexity: O(n) to O(n^2) depending on approach  
// Space Complexity: O(1) to O(n) depending on approach  
  
impl Solution {  
    pub fn circular_permutation(n: i32, start: i32) -> Vec<i32> {  
        }  
        }  
}
```

Ruby Solution:

```
# @param {Integer} n  
# @param {Integer} start  
# @return {Integer[]}
```

```
def circular_permutation(n, start)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer $n
     * @param Integer $start
     * @return Integer[]
     */
    function circularPermutation($n, $start) {

    }
}
```

Scala Solution:

```
object Solution {
  def circularPermutation(n: Int, start: Int): List[Int] = {
    }
}
```