Problem 3470: Permutations IV

Problem Information

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

Problem Description

Given two integers,

n

and

k

, an

alternating permutation

is a permutation of the first

n

positive integers such that no

two

adjacent elements are both odd or both even.

Return the

k-th

alternating permutation

sorted in

lexicographical order

. If there are fewer than

k

valid

alternating permutations

, return an empty list.

Example 1:

Input:

n = 4, k = 6

Output:

[3,4,1,2]

Explanation:

The lexicographically-sorted alternating permutations of

[1, 2, 3, 4]

are:

[1, 2, 3, 4]

[1, 4, 3, 2]

[2, 1, 4, 3]

[2, 3, 4, 1]

[3, 2, 1, 4]

[3, 4, 1, 2]

← 6th permutation

[4, 1, 2, 3]

[4, 3, 2, 1]

Since

k = 6

, we return

[3, 4, 1, 2]

.

Example 2:

Input:

n = 3, k = 2

Output:

[3,2,1]

Explanation:

The lexicographically-sorted alternating permutations of

[1, 2, 3]

are:

[1, 2, 3]

[3, 2, 1]

← 2nd permutation

Since

k = 2

, we return

[3, 2, 1]

.

Example 3:

Input:

n = 2, k = 3

Output:

[]

Explanation:

The lexicographically-sorted alternating permutations of

[1, 2]

are:

[1, 2]

[2, 1]

There are only 2 alternating permutations, but

k = 3

, which is out of range. Thus, we return an empty list

[]

.

Constraints:

1 <= n <= 100

1 <= k <= 10

15

Code Snippets

C++:

```
class Solution {
public:
vector<int> permute(int n, long long k) {

}
};
```

Java:

```
class Solution {
public int[] permute(int n, long k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number[]}
*/
var permute = function(n, k) {

};
```

TypeScript:

```
function permute(n: number, k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] Permute(int n, long k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* permute(int n, long long k, int* returnSize) {

}
```

Go:

```
func permute(n int, k int64) []int {

}
```

Kotlin:

```
class Solution {
fun permute(n: Int, k: Long): IntArray {

}
}
```

Swift:

```
class Solution {
func permute(_ n: Int, _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn permute(n: i32, k: i64) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer[]}
def permute(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer[]
*/
function permute($n, $k) {

}
}
```

Dart:

```
class Solution {
List<int> permute(int n, int k) {

}
}
```

Scala:

```
object Solution {
def permute(n: Int, k: Long): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec permute(n :: integer, k :: integer) :: [integer]
def permute(n, k) do

end
end
```

Erlang:

```
-spec permute(N :: integer(), K :: integer()) -> [integer()].
permute(N, K) ->
```

```
.
```

Racket:

```
(define/contract (permute n k)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Permutations IV
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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<int> permute(int n, long long k) {

}
};
```

Java Solution:

```
/**
* Problem: Permutations IV
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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 int[] permute(int n, long k) {

}
}
```

Python3 Solution:

```
"""
Problem: Permutations IV
Difficulty: Hard
Tags: array, graph, math, sort

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 permute(self, n: int, k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Permutations IV
* Difficulty: Hard
* Tags: array, graph, math, sort
*
```

```
* 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
* @param {number} k
* @return {number[]}
*/
var permute = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Permutations IV
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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 permute(n: number, k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Permutations IV
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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 int[] Permute(int n, long k) {

}
}
```

C Solution:

```
/*
* Problem: Permutations IV
* Difficulty: Hard
* Tags: array, graph, math, sort
*
* 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
*/

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

}
```

Go Solution:

```
// Problem: Permutations IV
// Difficulty: Hard
// Tags: array, graph, math, sort
//
// 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 permute(n int, k int64) []int {

}
```

Kotlin Solution:

```
class Solution {
fun permute(n: Int, k: Long): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func permute(_ n: Int, _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Permutations IV
// Difficulty: Hard
// Tags: array, graph, math, sort
//
// 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 permute(n: i32, k: i64) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer[]}
def permute(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer[]
*/
function permute($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> permute(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def permute(n: Int, k: Long): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec permute(n :: integer, k :: integer) :: [integer]
def permute(n, k) do

end
end
```

Erlang Solution:

```
-spec permute(N :: integer(), K :: integer()) -> [integer()].
permute(N, K) ->
.
```

Racket Solution:

```
(define/contract (permute n k)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```