Problem 2612: Minimum Reverse Operations

Problem Information

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

Problem Description

You are given an integer

n

and an integer

p

representing an array

arr

of length

n

where all elements are set to 0's, except position

p

which is set to 1. You are also given an integer array

banned

containing restricted positions. Perform the following operation on

arr

:

Reverse a

subarray

with size

k

if the single 1 is not set to a position in

banned

.

Return an integer array

answer

with

n

results where the

i

th

result is

the

minimum

number of operations needed to bring the single 1 to position

i

in

arr

, or -1 if it is impossible.

Example 1:

Input:

n = 4, p = 0, banned = [1,2], k = 4

Output:

[0,-1,-1,1]

Explanation:

Initially 1 is placed at position 0 so the number of operations we need for position 0 is 0.

We can never place 1 on the banned positions, so the answer for positions 1 and 2 is -1.

Perform the operation of size 4 to reverse the whole array.

After a single operation 1 is at position 3 so the answer for position 3 is 1.

Example 2:

Input:

n = 5, p = 0, banned = [2,4], k = 3

Output:

[0,-1,-1,-1,-1]

Explanation:

Initially 1 is placed at position 0 so the number of operations we need for position 0 is 0.

We cannot perform the operation on the subarray positions

[0, 2]

because position 2 is in banned.

Because 1 cannot be set at position 2, it is impossible to set 1 at other positions in more operations.

Example 3:

Input:

n = 4, p = 2, banned = [0,1,3], k = 1

Output:

[-1,-1,0,-1]

Explanation:

Perform operations of size 1 and 1 never changes its position.

Constraints:

1 <= n <= 10

5

0 <= p <= n - 1

0 <= banned.length <= n - 1

0 <= banned[i] <= n - 1

1 <= k <= n

banned[i] != p

all values in

banned

are

unique

Code Snippets

C++:

```
class Solution {
public:
vector<int> minReverseOperations(int n, int p, vector<int>& banned, int k) {

}
};
```

Java:

```
class Solution {
public int[] minReverseOperations(int n, int p, int[] banned, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minReverseOperations(n: number, p: number, banned: number[], k:
number): number[] {

};
```

C#:

```
public class Solution {
public int[] MinReverseOperations(int n, int p, int[] banned, int k) {

}
}
```

C:

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

```
int* minReverseOperations(int n, int p, int* banned, int bannedSize, int k,
int* returnSize) {

}
```

Go:

```
func minReverseOperations(n int, p int, banned []int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun minReverseOperations(n: Int, p: Int, banned: IntArray, k: Int): IntArray
{

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn min_reverse_operations(n: i32, p: i32, banned: Vec<i32>, k: i32) ->
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} p
```

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> minReverseOperations(int n, int p, List<int> banned, int k) {

}
}
```

Scala:

```
object Solution {
def minReverseOperations(n: Int, p: Int, banned: Array[Int], k: Int):
Array[Int] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec min_reverse_operations(N :: integer(), P :: integer(), Banned ::
[integer()], K :: integer()) -> [integer()].
min_reverse_operations(N, P, Banned, K) ->
.
```

Racket:

```
(define/contract (min-reverse-operations n p banned k)
(-> exact-integer? exact-integer? (listof exact-integer?) exact-integer?
(listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Reverse Operations
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> minReverseOperations(int n, int p, vector<int>& banned, int k) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: Minimum Reverse Operations
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] minReverseOperations(int n, int p, int[] banned, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Reverse Operations
Difficulty: Hard
Tags: array, graph, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def minReverseOperations(self, n: int, p: int, banned: List[int], k: int) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Reverse Operations
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} n
* @param {number} p
* @param {number[]} banned
* @param {number} k
* @return {number[]}
*/
var minReverseOperations = function(n, p, banned, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Reverse Operations
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

function minReverseOperations(n: number, p: number, banned: number[], k:
number): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Reverse Operations
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] MinReverseOperations(int n, int p, int[] banned, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Reverse Operations
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

```
int* minReverseOperations(int n, int p, int* banned, int bannedSize, int k,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Reverse Operations
// Difficulty: Hard
// Tags: array, graph, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func minReverseOperations(n int, p int, banned []int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun minReverseOperations(n: Int, p: Int, banned: IntArray, k: Int): IntArray
{

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Reverse Operations
// Difficulty: Hard
```

```
// Tags: array, graph, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn min_reverse_operations(n: i32, p: i32, banned: Vec<i32>, k: i32) ->
Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> minReverseOperations(int n, int p, List<int> banned, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minReverseOperations(n: Int, p: Int, banned: Array[Int], k: Int):
Array[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_reverse_operations(N :: integer(), P :: integer(), Banned ::
[integer()], K :: integer()) -> [integer()].
min_reverse_operations(N, P, Banned, K) ->
.
```

Racket Solution:

```
(define/contract (min-reverse-operations n p banned k)
(-> exact-integer? exact-integer? (listof exact-integer?) exact-integer?
(listof exact-integer?))
)
```