# Problem 957: Prison Cells After N Days

## Problem Information

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

## Problem Description

There are

8

prison cells in a row and each cell is either occupied or vacant.

Each day, whether the cell is occupied or vacant changes according to the following rules:

If a cell has two adjacent neighbors that are both occupied or both vacant, then the cell becomes occupied.

Otherwise, it becomes vacant.

Note

that because the prison is a row, the first and the last cells in the row can't have two adjacent neighbors.

You are given an integer array

cells

where

cells[i] == 1

if the

i

th

cell is occupied and

cells[i] == 0

if the

i

th

cell is vacant, and you are given an integer

n

.

Return the state of the prison after

n

days (i.e.,

n

such changes described above).

Example 1:

Input:

cells = [0,1,0,1,1,0,0,1], n = 7

Output:

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

Explanation:

The following table summarizes the state of the prison on each day: Day 0: [0, 1, 0, 1, 1, 0, 0, 1] Day 1: [0, 1, 1, 0, 0, 0, 0, 0] Day 2: [0, 0, 0, 0, 1, 1, 1, 0] Day 3: [0, 1, 1, 0, 0, 1, 0, 0] Day 4: [0, 0, 0, 0, 0, 1, 0, 0] Day 5: [0, 1, 1, 1, 0, 1, 0, 0] Day 6: [0, 0, 1, 0, 1, 1, 0, 0] Day 7: [0, 0, 1, 1, 0, 0, 0, 0]

Example 2:

Input:

cells = [1,0,0,1,0,0,1,0], n = 1000000000

Output:

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

Constraints:

cells.length == 8

cells[i]

is either

0

or

1

.

1 <= n <= 10

9

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> prisonAfterNDays(vector<int>& cells, int n) {

}
};
```

**Java:**

```
class Solution {
public int[] prisonAfterNDays(int[] cells, int n) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

```
};
```

**TypeScript:**

```
function prisonAfterNDays(cells: number[], n: number): number[] {

};
```

**C#:**

```
public class Solution {
public int[] PrisonAfterNDays(int[] cells, int n) {

}
}
```

**C:**

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

}
```

**Go:**

```
func prisonAfterNDays(cells []int, n int) []int {

}
```

**Kotlin:**

```
class Solution {
fun prisonAfterNDays(cells: IntArray, n: Int): IntArray {

}
}
```

**Swift:**

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

}
}
```

**Rust:**

```
impl Solution {
pub fn prison_after_n_days(cells: Vec<i32>, n: i32) -> Vec<i32> {

}
}
```

**Ruby:**

```
# @param {Integer[]} cells
# @param {Integer} n
# @return {Integer[]}
def prison_after_n_days(cells, n)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $cells
* @param Integer $n
* @return Integer[]
*/
function prisonAfterNDays($cells, $n) {

}
}
```

**Dart:**

```
class Solution {
List<int> prisonAfterNDays(List<int> cells, int n) {

}
```

```
}
```

**Scala:**

```
object Solution {
def prisonAfterNDays(cells: Array[Int], n: Int): Array[Int] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec prison_after_n_days(cells :: [integer], n :: integer) :: [integer]
def prison_after_n_days(cells, n) do

end
end
```

**Erlang:**

```
-spec prison_after_n_days(Cells :: [integer()], N :: integer()) ->
[integer()].
prison_after_n_days(Cells, N) ->
.
```

**Racket:**

```
(define/contract (prison-after-n-days cells n)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Prison Cells After N Days
* Difficulty: Medium
* Tags: array, math, hash
```

```
*
* 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> prisonAfterNDays(vector<int>& cells, int n) {

}
};
```

### Java Solution:

```
/**
* Problem: Prison Cells After N Days
* Difficulty: Medium
* Tags: array, math, hash
*
* 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[] prisonAfterNDays(int[] cells, int n) {

}
}
```

### Python3 Solution:

```
"""
Problem: Prison Cells After N Days
Difficulty: Medium
Tags: array, math, hash

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Prison Cells After N Days
* Difficulty: Medium
* Tags: array, math, hash
*
* 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[]} cells
* @param {number} n
* @return {number[]}
*/
var prisonAfterNDays = function(cells, n) {

};
```

### TypeScript Solution:

```
/**
* Problem: Prison Cells After N Days
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 prisonAfterNDays(cells: number[], n: number): number[] {

};
```

### C# Solution:

```
/*
* Problem: Prison Cells After N Days
* Difficulty: Medium
* Tags: array, math, hash
*
* 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[] PrisonAfterNDays(int[] cells, int n) {

}
}
```

### C Solution:

```
/*
* Problem: Prison Cells After N Days
* Difficulty: Medium
* Tags: array, math, hash
*
* 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* prisonAfterNDays(int* cells, int cellsSize, int n, int* returnSize) {

}
```

**Go Solution:**

```
// Problem: Prison Cells After N Days
// Difficulty: Medium
// Tags: array, math, hash
//
// 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 prisonAfterNDays(cells []int, n int) []int {

}
```

**Kotlin Solution:**

```
class Solution {
fun prisonAfterNDays(cells: IntArray, n: Int): IntArray {

}
}
```

**Swift Solution:**

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

}
}
```

**Rust Solution:**

```
// Problem: Prison Cells After N Days
// Difficulty: Medium
// Tags: array, math, hash
//
// 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 prison_after_n_days(cells: Vec<i32>, n: i32) -> Vec<i32> {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} cells
# @param {Integer} n
# @return {Integer[]}
def prison_after_n_days(cells, n)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $cells
* @param Integer $n
* @return Integer[]
*/
function prisonAfterNDays($cells, $n) {

}
}
```

### Dart Solution:

```
class Solution {
List<int> prisonAfterNDays(List<int> cells, int n) {
```

```
}
}
```

### Scala Solution:

```
object Solution {
def prisonAfterNDays(cells: Array[Int], n: Int): Array[Int] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec prison_after_n_days(cells :: [integer], n :: integer) :: [integer]
def prison_after_n_days(cells, n) do

end
end
```

### Erlang Solution:

```
-spec prison_after_n_days(Cells :: [integer()], N :: integer()) ->
[integer()].
prison_after_n_days(Cells, N) ->
.
```

### Racket Solution:

```
(define/contract (prison-after-n-days cells n)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```