Problem 2033: Minimum Operations to Make a Uni-Value Grid

Problem Information

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

Problem Description

You are given a 2D integer

grid

of size

m x n

and an integer

x

. In one operation, you can

add

x

to or

subtract

x

from any element in the

grid

.

A

uni-value grid

is a grid where all the elements of it are equal.

Return

the

minimum

number of operations to make the grid

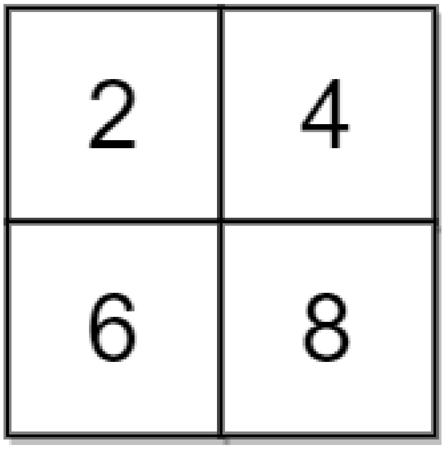
uni-value

. If it is not possible, return

-1

.

Example 1:



2	4
6	8

Input:

grid = [[2,4],[6,8]], x = 2

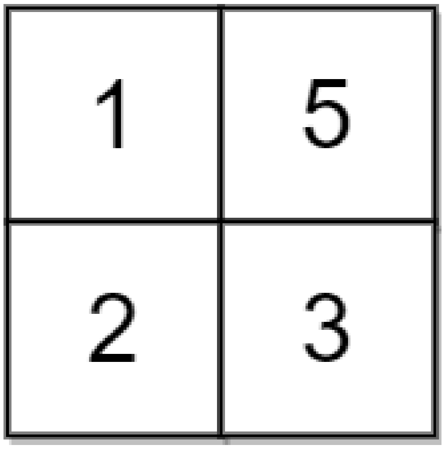
Output:

4

Explanation:

We can make every element equal to 4 by doing the following: - Add x to 2 once. - Subtract x from 6 once. - Subtract x from 8 twice. A total of 4 operations were used.

Example 2:



1	5
2	3

Input:

grid = [[1,5],[2,3]], x = 1

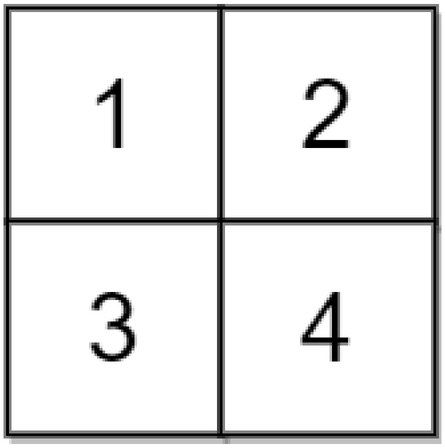
Output:

5

Explanation:

We can make every element equal to 3.

Example 3:



1	2
3	4

Input:

grid = [[1,2],[3,4]], x = 2

Output:

-1

Explanation:

It is impossible to make every element equal.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 10

5

1 <= m * n <= 10

5

1 <= x, grid[i][j] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int minOperations(vector<vector<int>>& grid, int x) {

}
};
```

Java:

```
class Solution {
public int minOperations(int[][] grid, int x) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, grid: List[List[int]], x: int) -> int:
```

Python:

```
class Solution(object):
def minOperations(self, grid, x):
"""
:type grid: List[List[int]]
:type x: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number} x
* @return {number}
*/
```

```
var minOperations = function(grid, x) {

};
```

TypeScript:

```
function minOperations(grid: number[][], x: number): number {

};
```

C#:

```
public class Solution {
public int MinOperations(int[][] grid, int x) {

}
}
```

C:

```
int minOperations(int** grid, int gridSize, int* gridColSize, int x) {

}
```

Go:

```
func minOperations(grid [][]int, x int) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(grid: Array<IntArray>, x: Int): Int {

}
}
```

Swift:

```
class Solution {
func minOperations(_ grid: [[Int]], _ x: Int) -> Int {


}
}
```

Rust:

```
impl Solution {
pub fn min_operations(grid: Vec<Vec<i32>>, x: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} x
# @return {Integer}
def min_operations(grid, x)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $x
* @return Integer
*/
function minOperations($grid, $x) {

}
}
```

Dart:

```
class Solution {
int minOperations(List<List<int>> grid, int x) {

}
}
```

Scala:

```
object Solution {
def minOperations(grid: Array[Array[Int]], x: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(grid :: [[integer]], x :: integer) :: integer
def min_operations(grid, x) do

end
end
```

Erlang:

```
-spec min_operations(Grid :: [[integer()]], X :: integer()) -> integer().
min_operations(Grid, X) ->
.
```

Racket:

```
(define/contract (min-operations grid x)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make a Uni-Value Grid
* Difficulty: Medium
* Tags: array, 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 minOperations(vector<vector<int>>& grid, int x) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make a Uni-Value Grid
* Difficulty: Medium
* Tags: array, 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 minOperations(int[][] grid, int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make a Uni-Value Grid
Difficulty: Medium
Tags: array, 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 minOperations(self, grid: List[List[int]], x: int) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, grid, x):
"""
:type grid: List[List[int]]
:type x: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make a Uni-Value Grid
* Difficulty: Medium
* Tags: array, 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[][]} grid
* @param {number} x
* @return {number}
*/
var minOperations = function(grid, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make a Uni-Value Grid
* Difficulty: Medium
* Tags: array, 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 minOperations(grid: number[][], x: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make a Uni-Value Grid
* Difficulty: Medium
* Tags: array, 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 MinOperations(int[][] grid, int x) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make a Uni-Value Grid
* Difficulty: Medium
* Tags: array, 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
*/

int minOperations(int** grid, int gridSize, int* gridColSize, int x) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make a Uni-Value Grid
// Difficulty: Medium
// Tags: array, 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 minOperations(grid [][]int, x int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(grid: Array<IntArray>, x: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ grid: [[Int]], _ x: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make a Uni-Value Grid
// Difficulty: Medium
// Tags: array, 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 min_operations(grid: Vec<Vec<i32>>, x: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} x
# @return {Integer}
def min_operations(grid, x)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $x
* @return Integer
*/
function minOperations($grid, $x) {

}
}
```

Dart Solution:

```
class Solution {
int minOperations(List<List<int>> grid, int x) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(grid: Array[Array[Int]], x: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(grid :: [[integer]], x :: integer) :: integer
def min_operations(grid, x) do

end
end
```

Erlang Solution:

```
-spec min_operations(Grid :: [[integer()]], X :: integer()) -> integer().
min_operations(Grid, X) ->
.
```

Racket Solution:

```
(define/contract (min-operations grid x)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```