Problem 1981: Minimize the Difference Between Target and Chosen Elements

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

mat

and an integer

target

.

Choose one integer from

each row

in the matrix such that the

absolute difference

between

target

and the

sum

of the chosen elements is

minimized

.

Return

the

minimum absolute difference

.

The

absolute difference

between two numbers

a

and

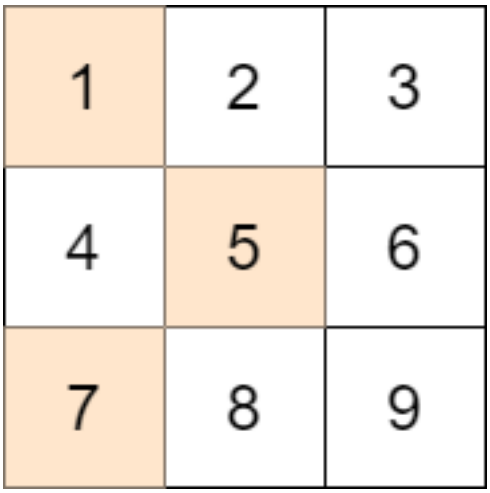
b

is the absolute value of

a - b

.

Example 1:



Input:

mat = [[1,2,3],[4,5,6],[7,8,9]], target = 13

Output:

0

Explanation:

One possible choice is to: - Choose 1 from the first row. - Choose 5 from the second row. - Choose 7 from the third row. The sum of the chosen elements is 13, which equals the target, so the absolute difference is 0.

Example 2:

Input:

mat = [[1],[2],[3]], target = 100

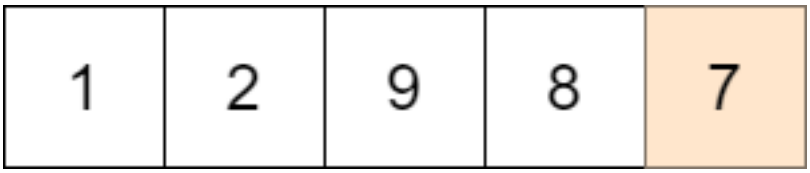
Output:

94

Explanation:

The best possible choice is to: - Choose 1 from the first row. - Choose 2 from the second row. - Choose 3 from the third row. The sum of the chosen elements is 6, and the absolute difference is 94.

Example 3:



Input:

mat = [[1,2,9,8,7]], target = 6

Output:

1

Explanation:

The best choice is to choose 7 from the first row. The absolute difference is 1.

Constraints:

m == mat.length

n == mat[i].length

1 <= m, n <= 70

1 <= mat[i][j] <= 70

1 <= target <= 800

Code Snippets

C++:

```
class Solution {
public:
int minimizeTheDifference(vector<vector<int>>& mat, int target) {

}
};
```

Java:

```
class Solution {
public int minimizeTheDifference(int[][] mat, int target) {

}
}
```

Python3:

```
class Solution:
def minimizeTheDifference(self, mat: List[List[int]], target: int) -> int:
```

Python:

```
class Solution(object):
def minimizeTheDifference(self, mat, target):
"""
:type mat: List[List[int]]
:type target: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} mat
* @param {number} target
* @return {number}
*/
var minimizeTheDifference = function(mat, target) {

};
```

TypeScript:

```
function minimizeTheDifference(mat: number[][], target: number): number {

};
```

C#:

```
public class Solution {
public int MinimizeTheDifference(int[][] mat, int target) {

}
}
```

C:

```
int minimizeTheDifference(int** mat, int matSize, int* matColSize, int
target) {

}
```

Go:

```
func minimizeTheDifference(mat [][]int, target int) int {

}
```

Kotlin:

```
class Solution {
fun minimizeTheDifference(mat: Array<IntArray>, target: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimizeTheDifference(_ mat: [[Int]], _ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimize_the_difference(mat: Vec<Vec<i32>>, target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} mat
# @param {Integer} target
# @return {Integer}
def minimize_the_difference(mat, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $mat
* @param Integer $target
* @return Integer
*/
function minimizeTheDifference($mat, $target) {

}
}
```

Dart:

```
class Solution {
int minimizeTheDifference(List<List<int>> mat, int target) {
```

```
}
}
```

Scala:

```
object Solution {
def minimizeTheDifference(mat: Array[Array[Int]], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimize_the_difference(mat :: [[integer]], target :: integer) ::
integer
def minimize_the_difference(mat, target) do

end
end
```

Erlang:

```
-spec minimize_the_difference(Mat :: [[integer()]], Target :: integer()) ->
integer().
minimize_the_difference(Mat, Target) ->
.
```

Racket:

```
(define/contract (minimize-the-difference mat target)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize the Difference Between Target and Chosen Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minimizeTheDifference(vector<vector<int>>& mat, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize the Difference Between Target and Chosen Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimizeTheDifference(int[][] mat, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize the Difference Between Target and Chosen Elements
Difficulty: Medium
Tags: array, dp
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimizeTheDifference(self, mat: List[List[int]], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimizeTheDifference(self, mat, target):
"""
:type mat: List[List[int]]
:type target: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimize the Difference Between Target and Chosen Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} mat
* @param {number} target
* @return {number}
*/
var minimizeTheDifference = function(mat, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize the Difference Between Target and Chosen Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimizeTheDifference(mat: number[][], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimize the Difference Between Target and Chosen Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinimizeTheDifference(int[][] mat, int target) {

}
}
```

C Solution:

```
/*
* Problem: Minimize the Difference Between Target and Chosen Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimizeTheDifference(int** mat, int matSize, int* matColSize, int
target) {

}
```

Go Solution:

```
// Problem: Minimize the Difference Between Target and Chosen Elements
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimizeTheDifference(mat [][]int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimizeTheDifference(mat: Array<IntArray>, target: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimizeTheDifference(_ mat: [[Int]], _ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimize the Difference Between Target and Chosen Elements
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimize_the_difference(mat: Vec<Vec<i32>>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} mat
# @param {Integer} target
# @return {Integer}
def minimize_the_difference(mat, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $mat
* @param Integer $target
* @return Integer
*/
function minimizeTheDifference($mat, $target) {

}
}
```

Dart Solution:

```
class Solution {
int minimizeTheDifference(List<List<int>> mat, int target) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def minimizeTheDifference(mat: Array[Array[Int]], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimize_the_difference(mat :: [[integer]], target :: integer) ::
integer
def minimize_the_difference(mat, target) do

end
end
```

Erlang Solution:

```
-spec minimize_the_difference(Mat :: [[integer()]], Target :: integer()) ->
integer().
minimize_the_difference(Mat, Target) ->
.
```

Racket Solution:

```
(define/contract (minimize-the-difference mat target)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```