Problem 3276: Select Cells in Grid With Maximum Score

Problem Information

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

Problem Description

You are given a 2D matrix

grid

consisting of positive integers.

You have to select

one or more

cells from the matrix such that the following conditions are satisfied:

No two selected cells are in the

same

row of the matrix.

The values in the set of selected cells are

unique

.

Your score will be the

sum

of the values of the selected cells.

Return the

maximum

score you can achieve.

Example 1:

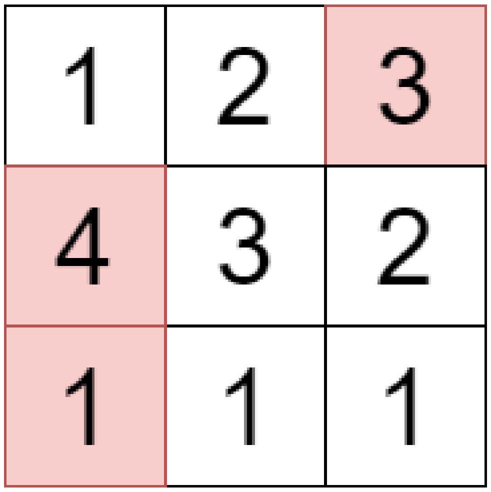
Input:

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

Output:

8

Explanation:



We can select the cells with values 1, 3, and 4 that are colored above.

Example 2:

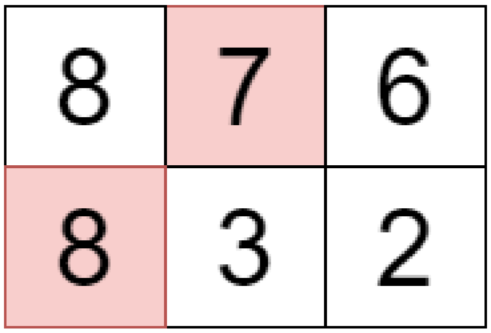
Input:

grid = [[8,7,6],[8,3,2]]

Output:

15

Explanation:



We can select the cells with values 7 and 8 that are colored above.

Constraints:

1 <= grid.length, grid[i].length <= 10

1 <= grid[i][j] <= 100

Code Snippets

C++:

```
class Solution {
public:
int maxScore(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int maxScore(List<List<Integer>> grid) {
}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number}
*/
var maxScore = function(grid) {

};
```

TypeScript:

```
function maxScore(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxScore(IList<IList<int>> grid) {

}
}
```

C:

```
int maxScore(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func maxScore(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxScore(grid: List<List<Int>>): Int {

}
}
```

Swift:

```
class Solution {
func maxScore(_ grid: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_score(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer}
def max_score(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function maxScore($grid) {

}
}
```

Dart:

```
class Solution {
int maxScore(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def maxScore(grid: List[List[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_score(grid :: [[integer]]) :: integer
def max_score(grid) do

end
end
```

Erlang:

```
-spec max_score(Grid :: [[integer()]]) -> integer().
max_score(Grid) ->
.
```

Racket:

```
(define/contract (max-score grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Select Cells in Grid With Maximum Score
* Difficulty: Hard
* 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 maxScore(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Select Cells in Grid With Maximum Score
* Difficulty: Hard
* 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 maxScore(List<List<Integer>> grid) {
}
}
```

Python3 Solution:

```
"""
Problem: Select Cells in Grid With Maximum Score
Difficulty: Hard
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 maxScore(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Select Cells in Grid With Maximum Score
* Difficulty: Hard
* 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[][]} grid
* @return {number}
*/
var maxScore = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Select Cells in Grid With Maximum Score
* Difficulty: Hard
* 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 maxScore(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Select Cells in Grid With Maximum Score
* Difficulty: Hard
* 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 MaxScore(IList<IList<int>> grid) {

}
}
```

C Solution:

```
/*
* Problem: Select Cells in Grid With Maximum Score
* Difficulty: Hard
* 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 maxScore(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Select Cells in Grid With Maximum Score
// Difficulty: Hard
// 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 maxScore(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxScore(grid: List<List<Int>>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxScore(_ grid: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Select Cells in Grid With Maximum Score
// Difficulty: Hard
// 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 max_score(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def max_score(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function maxScore($grid) {

}
}
```

Dart Solution:

```
class Solution {
int maxScore(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def maxScore(grid: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_score(grid :: [[integer]]) :: integer
def max_score(grid) do

end
end
```

Erlang Solution:

```
-spec max_score(Grid :: [[integer()]]) -> integer().
max_score(Grid) ->
.
```

Racket Solution:

```
(define/contract (max-score grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```