Problem 1937: Maximum Number of Points with Cost

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

points

(

0-indexed

). Starting with

0

points, you want to

maximize

the number of points you can get from the matrix.

To gain points, you must pick one cell in

each row

. Picking the cell at coordinates

(r, c)

will

add

points[r][c]

to your score.

However, you will lose points if you pick a cell too far from the cell that you picked in the previous row. For every two adjacent rows

r

and

r + 1

(where

0 <= r < m - 1

), picking cells at coordinates

(r, c

1

)

and

(r + 1, c

2

)

will

subtract

abs(c

1

- c

2

)

from your score.

Return

the

maximum

number of points you can achieve

.

abs(x)

is defined as:

x

for

x >= 0

.

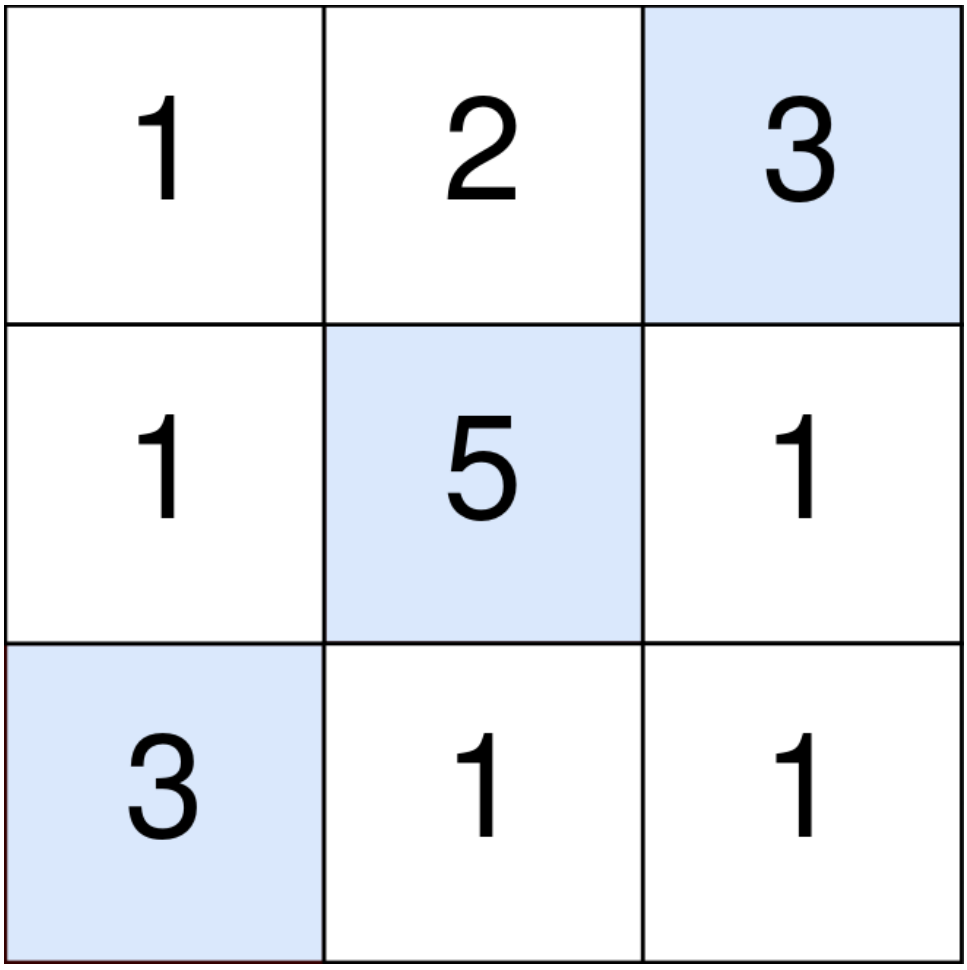
-x

for

x < 0

.

Example 1:



Input:

points = [[1,2,3],[1,5,1],[3,1,1]]

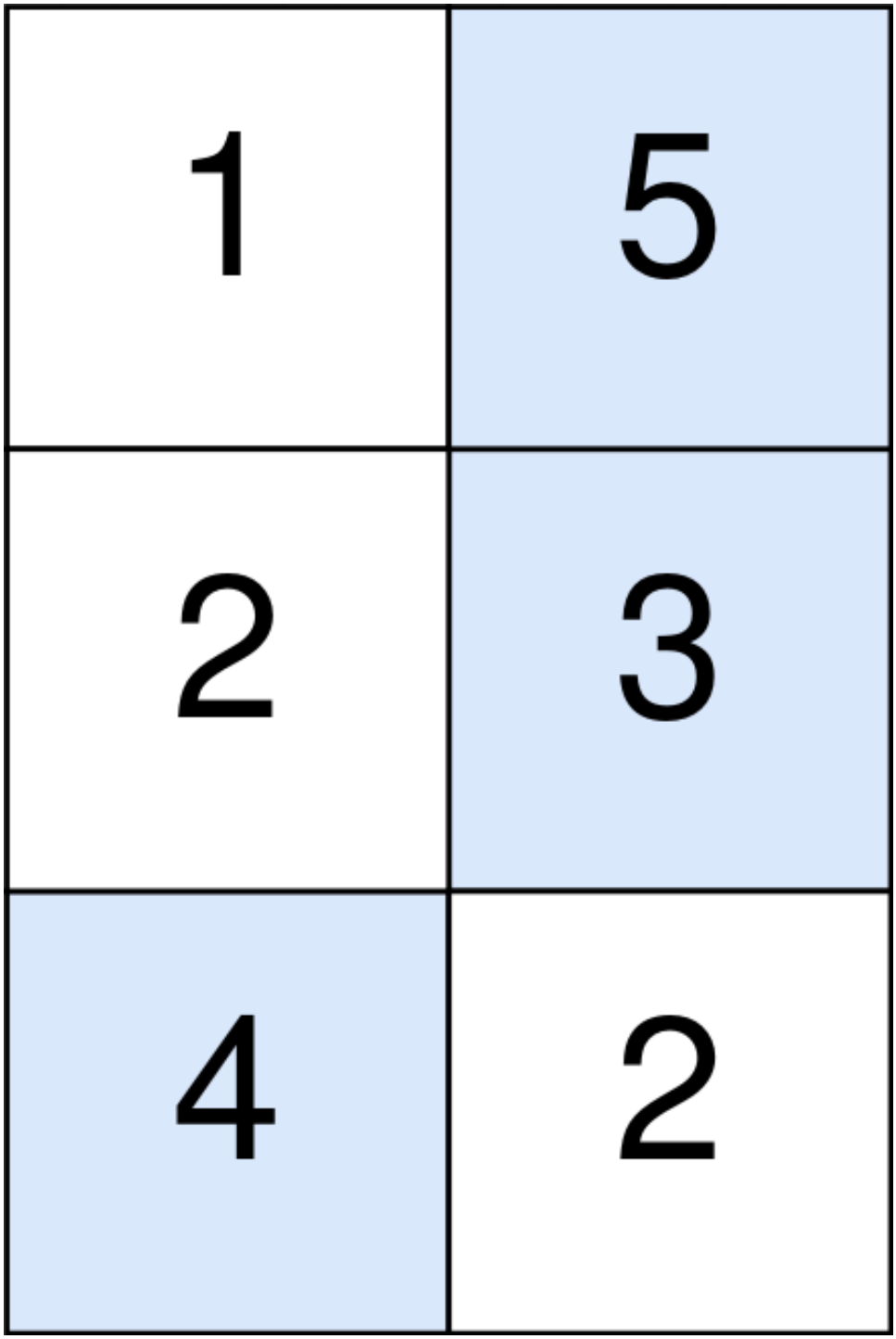
Output:

9

Explanation:

The blue cells denote the optimal cells to pick, which have coordinates (0, 2), (1, 1), and (2, 0). You add 3 + 5 + 3 = 11 to your score. However, you must subtract abs(2 - 1) + abs(1 - 0) = 2 from your score. Your final score is 11 - 2 = 9.

Example 2:



Input:

points = [[1,5],[2,3],[4,2]]

Output:

11

Explanation:

The blue cells denote the optimal cells to pick, which have coordinates (0, 1), (1, 1), and (2, 0). You add 5 + 3 + 4 = 12 to your score. However, you must subtract abs(1 - 1) + abs(1 - 0) = 1 from your score. Your final score is 12 - 1 = 11.

Constraints:

m == points.length

n == points[r].length

1 <= m, n <= 10

5

1 <= m * n <= 10

5

0 <= points[r][c] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long maxPoints(vector<vector<int>>& points) {

}
};
```

Java:

```
class Solution {
public long maxPoints(int[][] points) {
```

```
}
}
```

Python3:

```
class Solution:
def maxPoints(self, points: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxPoints(points: number[][]): number {

};
```

C#:

```
public class Solution {
public long MaxPoints(int[][] points) {

}
}
```

C:

```
long long maxPoints(int** points, int pointsSize, int* pointsColSize) {

}
```

Go:

```
func maxPoints(points [][]int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxPoints(points: Array<IntArray>): Long {

}
}
```

Swift:

```
class Solution {
func maxPoints(_ points: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_points(points: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[][]} points
# @return {Integer}
def max_points(points)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $points
* @return Integer
*/
function maxPoints($points) {

}
}
```

Dart:

```
class Solution {
int maxPoints(List<List<int>> points) {

}
}
```

Scala:

```
object Solution {
def maxPoints(points: Array[Array[Int]]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_points(points :: [[integer]]) :: integer
def max_points(points) do

end
end
```

Erlang:

```
-spec max_points(Points :: [[integer()]]) -> integer().
max_points(Points) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Points with Cost
* 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:
long long maxPoints(vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Points with Cost
* 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 long maxPoints(int[][] points) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Points with Cost
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 maxPoints(self, points: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Points with Cost
* 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[][]} points
* @return {number}
*/
var maxPoints = function(points) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Points with Cost
* 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 maxPoints(points: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Points with Cost
* 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 long MaxPoints(int[][] points) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Maximum Number of Points with Cost
* 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
*/

long long maxPoints(int** points, int pointsSize, int* pointsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Points with Cost
// 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 maxPoints(points [][]int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxPoints(points: Array<IntArray>): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxPoints(_ points: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Points with Cost
// 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 max_points(points: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} points
# @return {Integer}
def max_points(points)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $points
* @return Integer
*/
function maxPoints($points) {

}
}
```

Dart Solution:

```
class Solution {
int maxPoints(List<List<int>> points) {

}
}
```

Scala Solution:

```
object Solution {
def maxPoints(points: Array[Array[Int]]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_points(points :: [[integer]]) :: integer
def max_points(points) do

end
end
```

Erlang Solution:

```
-spec max_points(Points :: [[integer()]]) -> integer().
max_points(Points) ->
.
```

Racket Solution:

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