# Problem 1030: Matrix Cells in Distance Order

## Problem Information

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

## Problem Description

You are given four integers

row

,

cols

,

rCenter

, and

cCenter

. There is a

rows x cols

matrix and you are on the cell with the coordinates

(rCenter, cCenter)

.

Return

the coordinates of all cells in the matrix, sorted by their

distance

from

(rCenter, cCenter)

from the smallest distance to the largest distance

. You may return the answer in

any order

that satisfies this condition.

The

distance

between two cells

(r

1

, c

1

)

and

(r

2

, c

2

)

is

|r

1

- r

2

| + |c

1

- c

2

|

.

Example 1:

Input:

rows = 1, cols = 2, rCenter = 0, cCenter = 0

Output:

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

Explanation:

The distances from (0, 0) to other cells are: [0,1]

Example 2:

Input:

rows = 2, cols = 2, rCenter = 0, cCenter = 1

Output:

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

Explanation:

The distances from (0, 1) to other cells are: [0,1,1,2] The answer [[0,1],[1,1],[0,0],[1,0]] would also be accepted as correct.

Example 3:

Input:

rows = 2, cols = 3, rCenter = 1, cCenter = 2

Output:

[[1,2],[0,2],[1,1],[0,1],[1,0],[0,0]]

Explanation:

The distances from (1, 2) to other cells are: [0,1,1,2,2,3] There are other answers that would also be accepted as correct, such as [[1,2],[1,1],[0,2],[1,0],[0,1],[0,0]].

Constraints:

1 <= rows, cols <= 100

0 <= rCenter < rows

0 <= cCenter < cols

## Code Snippets

**C++:**

```
class Solution {
public:
vector<vector<int>> allCellsDistOrder(int rows, int cols, int rCenter, int
cCenter) {

}
};
```

**Java:**

```
class Solution {
public int[][] allCellsDistOrder(int rows, int cols, int rCenter, int
cCenter) {

}
}
```

**Python3:**

```
class Solution:
def allCellsDistOrder(self, rows: int, cols: int, rCenter: int, cCenter: int)
-> List[List[int]]:
```

**Python:**

```
class Solution(object):
def allCellsDistOrder(self, rows, cols, rCenter, cCenter):
"""
:type rows: int
:type cols: int
:type rCenter: int
:type cCenter: int
:rtype: List[List[int]]
"""
```

**JavaScript:**

```
/**
* @param {number} rows
* @param {number} cols
* @param {number} rCenter
* @param {number} cCenter
* @return {number[][]}
*/
var allCellsDistOrder = function(rows, cols, rCenter, cCenter) {

};
```

**TypeScript:**

```
function allCellsDistOrder(rows: number, cols: number, rCenter: number,
cCenter: number): number[][] {

};
```

**C#:**

```
public class Solution {
public int[][] AllCellsDistOrder(int rows, int cols, int rCenter, int
cCenter) {

}
}
```

**C:**

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** allCellsDistOrder(int rows, int cols, int rCenter, int cCenter, int*
returnSize, int** returnColumnSizes) {

}
```

**Go:**

```
func allCellsDistOrder(rows int, cols int, rCenter int, cCenter int) [][]int
{

}
```

**Kotlin:**

```
class Solution {
fun allCellsDistOrder(rows: Int, cols: Int, rCenter: Int, cCenter: Int):
Array<IntArray> {

}
}
```

**Swift:**

```
class Solution {
func allCellsDistOrder(_ rows: Int, _ cols: Int, _ rCenter: Int, _ cCenter:
Int) -> [[Int]] {

}
}
```

**Rust:**

```
impl Solution {
pub fn all_cells_dist_order(rows: i32, cols: i32, r_center: i32, c_center:
i32) -> Vec<Vec<i32>> {

}
}
```

**Ruby:**

```
# @param {Integer} rows
# @param {Integer} cols
# @param {Integer} r_center
# @param {Integer} c_center
# @return {Integer[][]}
def all_cells_dist_order(rows, cols, r_center, c_center)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer $rows
* @param Integer $cols
* @param Integer $rCenter
* @param Integer $cCenter
* @return Integer[][]
*/
function allCellsDistOrder($rows, $cols, $rCenter, $cCenter) {

}
}
```

**Dart:**

```
class Solution {
List<List<int>> allCellsDistOrder(int rows, int cols, int rCenter, int
cCenter) {

}
}
```

**Scala:**

```
object Solution {
def allCellsDistOrder(rows: Int, cols: Int, rCenter: Int, cCenter: Int):
Array[Array[Int]] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec all_cells_dist_order(rows :: integer, cols :: integer, r_center ::
integer, c_center :: integer) :: [[integer]]
def all_cells_dist_order(rows, cols, r_center, c_center) do

end
end
```

### Erlang:

```
-spec all_cells_dist_order(Rows :: integer(), Cols :: integer(), RCenter ::
integer(), CCenter :: integer()) -> [[integer()]].
all_cells_dist_order(Rows, Cols, RCenter, CCenter) ->
.
```

### Racket:

```
(define/contract (all-cells-dist-order rows cols rCenter cCenter)
(-> exact-integer? exact-integer? exact-integer? exact-integer? (listof
(listof exact-integer?)))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Matrix Cells in Distance Order
* Difficulty: Easy
* 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:
vector<vector<int>> allCellsDistOrder(int rows, int cols, int rCenter, int
cCenter) {

}
};
```

### Java Solution:

```
/**
* Problem: Matrix Cells in Distance Order
* Difficulty: Easy
```

```
* 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[][] allCellsDistOrder(int rows, int cols, int rCenter, int
cCenter) {

}
}
```

### Python3 Solution:

```
"""
Problem: Matrix Cells in Distance Order
Difficulty: Easy
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 allCellsDistOrder(self, rows: int, cols: int, rCenter: int, cCenter: int)
-> List[List[int]]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def allCellsDistOrder(self, rows, cols, rCenter, cCenter):
"""
:type rows: int
:type cols: int
:type rCenter: int
:type cCenter: int
```

```
:rtype: List[List[int]]
"""
```

### JavaScript Solution:

```
/**
* Problem: Matrix Cells in Distance Order
* Difficulty: Easy
* 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} rows
* @param {number} cols
* @param {number} rCenter
* @param {number} cCenter
* @return {number[][]}
*/
var allCellsDistOrder = function(rows, cols, rCenter, cCenter) {

};
```

### TypeScript Solution:

```
/**
* Problem: Matrix Cells in Distance Order
* Difficulty: Easy
* 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 allCellsDistOrder(rows: number, cols: number, rCenter: number,
cCenter: number): number[][] {
```

```
};
```

### C# Solution:

```
/*
* Problem: Matrix Cells in Distance Order
* Difficulty: Easy
* 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[][] AllCellsDistOrder(int rows, int cols, int rCenter, int
cCenter) {

}
}
```

### C Solution:

```
/*
* Problem: Matrix Cells in Distance Order
* Difficulty: Easy
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** allCellsDistOrder(int rows, int cols, int rCenter, int cCenter, int*
returnSize, int** returnColumnSizes) {
```

```
}
```

### Go Solution:

```
// Problem: Matrix Cells in Distance Order
// Difficulty: Easy
// 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 allCellsDistOrder(rows int, cols int, rCenter int, cCenter int) [][]int
{

}
```

### Kotlin Solution:

```
class Solution {
fun allCellsDistOrder(rows: Int, cols: Int, rCenter: Int, cCenter: Int):
Array<IntArray> {

}
}
```

### Swift Solution:

```
class Solution {
func allCellsDistOrder(_ rows: Int, _ cols: Int, _ rCenter: Int, _ cCenter:
Int) -> [[Int]] {

}
}
```

### Rust Solution:

```
// Problem: Matrix Cells in Distance Order
// Difficulty: Easy
// 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 all_cells_dist_order(rows: i32, cols: i32, r_center: i32, c_center:
i32) -> Vec<Vec<i32>> {

}
}
```

### Ruby Solution:

```
# @param {Integer} rows
# @param {Integer} cols
# @param {Integer} r_center
# @param {Integer} c_center
# @return {Integer[][]}
def all_cells_dist_order(rows, cols, r_center, c_center)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer $rows
* @param Integer $cols
* @param Integer $rCenter
* @param Integer $cCenter
* @return Integer[][]
*/
function allCellsDistOrder($rows, $cols, $rCenter, $cCenter) {

}
}
```

### Dart Solution:

```
class Solution {
List<List<int>> allCellsDistOrder(int rows, int cols, int rCenter, int
cCenter) {

}
}
```

### Scala Solution:

```
object Solution {
def allCellsDistOrder(rows: Int, cols: Int, rCenter: Int, cCenter: Int):
Array[Array[Int]] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec all_cells_dist_order(rows :: integer, cols :: integer, r_center ::
integer, c_center :: integer) :: [[integer]]
def all_cells_dist_order(rows, cols, r_center, c_center) do

end
end
```

### Erlang Solution:

```
-spec all_cells_dist_order(Rows :: integer(), Cols :: integer(), RCenter ::
integer(), CCenter :: integer()) -> [[integer()]].
all_cells_dist_order(Rows, Cols, RCenter, CCenter) ->
.
```

### Racket Solution:

```
(define/contract (all-cells-dist-order rows cols rCenter cCenter)
(-> exact-integer? exact-integer? exact-integer? exact-integer? (listof
(listof exact-integer?)))
)
```